



31-03-2015

Open Call Deliverable OCR-DS2.1

NSI Compliance Testing Suite 1.0

Architecture and User Manuals (NSI-CONTEST)

Open Call Deliverable OCR-DS2.1

Grant Agreement No.: 605243
Activity: NA1
Task Item: 10
Nature of Deliverable: R (Report)
Dissemination Level: PU (Public)
Lead Partner: NXW
Document Code: GN3PLUS14-1299-60
Authors: Giada Landi (NXW), Gino Carrozzo (NXW), Andrea Gronchi (NXW), Giacomo Bernini (NXW), Bartosz Belter (PSNC), Michał Giertych (PSNC), Michał Balcerkiewicz (PSNC)

© GEANT Limited on behalf of the GN3plus project.

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7 2007–2013) under Grant Agreement No. 605243 (GN3plus).

Abstract

This deliverable provides the architectural specification and the installation and user documentation for the NSI Compliance Testing Suite 1.0 prototype. The document describes the components of the NSI CTS system developed in the NSI-CONTEST project and the different test scenarios supported by the platform. Moreover, the deliverable provides a guideline for developers of NSI protocol stacks to verify the compliance of their implementation with the NSI v2.0 protocol specification using the NSI CTS tools.

Table of Contents

Executive Summary	1
1 Introduction	2
2 NSI protocol test scenarios	3
2.1 Testing an NSI implementation: the workflow	4
2.1.1 Running tests	4
2.1.2 Getting test results	6
2.1.3 Continuing tests	6
2.2 Testing Requester role	7
2.3 Testing Provider role	8
2.4 Testing Aggregator role	9
3 NSI Compliance Testing Suite System Architecture	11
3.1 NSI CTS Web Portal	12
3.1.1 CTS REST interface	14
3.2 NSI Reference Implementation and Local Test Runner	14
3.2.1 Test Runner REST interface	15
3.3 NSI-CTS platform workflows	20
4 NSI CTS Prototype	24
4.1 Software release	24
4.2 Installation and configuration guide	25
4.2.1 CTS configuration	26
4.2.2 Local Test Runner configuration	29
4.3 User guide	30
1.1.2 SUT configuration and CLI commands for demonstration	35
5 Conclusions	39
Appendix A Sample Scenarios	40
A.1 Requester scenarios	40
A.1.1 Single reservation	40
A.2 Provider scenarios	41
A.2.1 Single reservation	41
A.2.2 Single failed reservation	42
A.2.3 Reserve abort	43

	A.2.4 Multiple provisions	44
Appendix B	NSI-CTS REST API for Users	47
B.1	Calls	47
	B.1.1 Retrieve user projects	48
	B.1.2 Get single project information	48
	B.1.3 Get scenario templates	49
	B.1.4 Get all project testcases	50
	B.1.5 Get single project testcase	51
	B.1.6 Create a new project testcase	52
	B.1.7 Create a new certification testcase	53
References		54
Glossary		54

Table of Figures

Figure 2.1: Workflow to execute a test	5
Figure 2.2: Workflow to collect the results for a test	6
Figure 2.3: Workflow to continue tests	7
Figure 2.4: Scenario to test a Requester Agent	8
Figure 2.5: Scenario to test a Provider Agent	8
Figure 2.6: Scenario to test an Aggregator Agent	9
Figure 3.1: NSI CS v2.0 Compliance Test Suite	11
Figure 3.2: High-level architecture	12
Figure 3.3: NSI CTS web portal – Login and registration page	13
Figure 3.4: NSI CTS web portal – test results	14
Figure 3.5: NSI-RI and local test runner	14
Figure 3.6: Example of local test descriptor for a Provider SUT	18
Figure 3.7: Example of local test descriptor for a Requester SUT	19
Figure 3.8: Test report for a successful result	19
Figure 3.9: Test report for a failed result	20
Figure 3.10: Topology exchange	20
Figure 3.11: Provider workflow	21
Figure 3.12: Requester workflow	22
Figure 3.13: Aggregator workflow	23

Figure 4.1: CTS web portal main page	31
Figure 4.2: Project topology configuration	32
Figure 4.3: Project test cases	33
Figure 4.4: Single test results	33
Figure 4.5: CTS administration panel - overview	34
Figure 4.6: Test scenario template form	34

Table of Tables

Table 2.1: Network Service Agents defined in NSI CS v2.0 specification	3
Table 2.2: Steps to execute a test	5
Table 2.3: Steps to collect the results for a test	6
Table 2.4: Steps to continue tests	7
Table 3.1: NSI-RI components	15
Table 3.2: Test Runner REST API	16
Table 3.3: Test Runner REST API: <i>"POST /job"</i>	17
Table 3.4: Test Runner REST API: <i>"GET /job/{jobID}"</i>	17
Table 3.5: Workflow to test a Provider NSA	21
Table 3.6: Workflow to test a Requester NSA	22
Table 3.7: Workflow to test an Aggregator NSA	23
Table 4.1: NSI CTS: VM structure	25
Table 4.2: NSI CTS: ports used in the start-up configuration	25
Table 4.3: NSI CTS configuration parameters	27
Table 4.4: NSI CTS database configuration parameters	27
Table 4.5: SUT configuration parameters	36
Table 4.6: SUT CLI commands	37
Table 4.7: Reserve request configuration parameters	38

Executive Summary

This deliverable describes the NSI Compliance Testing Suite (CTS) developed in the NSI-CONTEST project. The NSI CTS allows NSI developers to test their own implementation of the NSI Connection Service (NSI CS) protocol stack with a reference implementation of the NSI CS v2.0, compliant with the standard released from the Open Grid Forum NSI WG. This report describes the NSI CTS architecture and software prototype, providing the documentation to install the NSI CTS platform and access the NSI CTS service through the web portal. The document is organized as follows:

- The introduction in section 1 provides a brief description of the NSI CS v2.0 protocol and explains the benefits of introducing a reference implementation of the NSI stack and a validation service to certificate the compliance of different NSI implementations with the standard.
- Section 2 defines the workflows to access and use the NSI CTS service from the perspective of an NSI developer, detailing the test scenarios to validate different types of NSI Agents (Requesters, Providers or Aggregators).
- Section 3 describes the architecture and the components of the NSI CTS, specifying the interfaces of the different modules and the internal workflows in the NSI CTS.
- Section 4 provides the documentation about the NSI CTS software prototype, which is released in a dedicated Virtual Machine where all the components have been installed with a pre-defined configuration. This section provides guidelines about how to modify the default configuration and how to use the NSI CTS tool and service.

The appendixes provides sample templates for NSI test descriptors and documents the REST interface of the NSI CTS, provided as alternative interface to the web portal to support the programmable definition and instantiation of NSI test sessions.

1 Introduction

The Network Service Interface (NSI) v2.0 protocol is an Open Grid Forum (OGF) open standard to integrate different systems for network resource provisioning in a multi-domain environment. It is designed as a common interface for both service providers and their clients to request and provide the network connections. The abstract messaging for network services simplifies and hides their implementation details while, at the same time, it still offers the same level of service independently from the technologies behind the scene. The protocol allows to provide different network features through its extension mechanisms. One of the services specified in the NSI standards suite is the Connection Service (NSI CS) [NSI-CS-2.0], for the provisioning of point-to-point connections. The NSI CS v2.0 is exposed as a WebService interface for software implementations.

NSI-CONTEST stands for **N**etwork **S**ervice Interface **C**onformance **T**est Suite (CTS) and it provides a testing platform prototype to validate NSI agent implementations from NSI developers. It is designed to support the development process of NSI CS v2.0 third party software with the objective of validating the compliance of the different implementation with the standard protocol. The NSI CTS offers a way of rapid building a complete testing environment with different defined scenarios that simplifies the execution and the repetition of NSI compliance tests. The CTS platform is fully independent from the specific user implementation and the whole testing process is done through the NSI protocol itself which means that neither CTS nor user software depends on each other. User agents act as Software Under Test (SUT) and they are treated as black boxes: their functionalities are examined through their behaviour and the conditions on the NSI protocol stack. For this purpose, an NSI Reference Implementation (NSI RI) has been provided to simulate other reference NSI agents in the testing environment, acting according to the standard NSI protocol defined in the NSI CS v2.0 specification [NSI-CS-2.0]. Through this approach, the NSI CTS design has provided a modular platform where test processing and management are independent from the actual implementation of the NSI reference protocol, so that it can be easily updated to support new versions of the NSI CS protocol or extended to cover any future NSI service, beyond the initial connection service.

The document is structured as follows:

- Section 2 introduces test scenarios and defines the workflows for the validation of a protocol stack implemented by an NSI developer.
- Section 3 defines the architecture of the NSI CTS platform and its interfaces.

Section 4 provides the user documentation for the software prototype.

2 NSI protocol test scenarios

The NSI Compliance Testing Suite (NSI CTS) provides a reference implementation of the NSI Connection Service (NSI-CS) v2.0 and allows developers of NSI v2.0 protocol stacks to verify the compliance of their implementation with the standard specification. The platform can be instantiated as an online service that provides a single point of contact for NSI developers to execute customizable, isolated and repeatable tests. The NSI CTS platform can emulate different scenarios to validate the behaviour of all the categories of Network Service Agents (NSAs) defined in the NSI-CS specification: (ultimate) Requester Agents, Aggregator Agents and (ultimate) Provider Agents (see Table 2.1).

Network Service Agent (NSA)	Description
(ultimate) Requester Agent	The originator of a network connection service request. It may be a component of a middleware application.
Aggregator Agent	An NSI agent which coordinates requests to more than one child NSAs and aggregates their responses.
(ultimate) Provider Agent	An NSA which coordinates with the local Network Resource Manager (NRM) to manage and allocate network resources as required in an NSI request.

Table 2.1: Network Service Agents defined in NSI CS v2.0 specification

The NSI CTS has the objective to verify the functions strictly related to the handling of the NSI protocol, for example the parsing and formatting of NSI messages, including checks about syntax and semantics of message parameters, and the protocol workflows. The validation of the NSI workflows is performed checking the behavior of the NSI protocol Finite State Machines (FSMs) simulating a variety of configurable conditions in the NSI Agent instances which communicate with the NSI Device Under Test (DUT). This approach allows to cover a wide range of scenarios beyond the typical successful provisioning of an end-to-end connection, including longer processing time at the NRM to allocate resources, timeout expirations, or failures due to lack of available network resources. On the other hand, all the functions that are not directly related to the NSI standard protocol, like the algorithms for path computation, the NRM in the user domain manager or the associated data plane, the authentication and authorization mechanisms or custom extensions to the NSI Connection Service are considered out of scope.

This chapter describes the overall workflow that must be followed by a generic NSI developer to run a test and get the results and specifies the scenarios supported by the NSI CTS for the different NSI Agent roles. For each role, it describes the actions and the expected interactions between the NSI DUT developed by the NSI CTS users and the NSI Reference Implementation (NSI-RI) instances deployed by the NSI CTS platform.

2.1 Testing an NSI implementation: the workflow

This section presents the different steps which an NSI developer needs to perform to execute a test that validates an implementation of the NSI CS protocol through the NSI CTS and collect the resulting report.

2.1.1 Running tests

The validation tests for NSI implementations can be started and configured from the NSI CTS web portal. The portal provides a set of pre-defined templates which the NSI developer can directly re-use to request the execution of a test. Templates can be also used as starting point to specify customized test descriptors with different parameters.

The overall workflow to specify and execute a test is shown in Figure 2.1 and detailed in Table 2.2, for both manual and automated test. The NSI CTS platform has been designed specifically to support automated tests, in order to reduce the actions required on the developer's side.

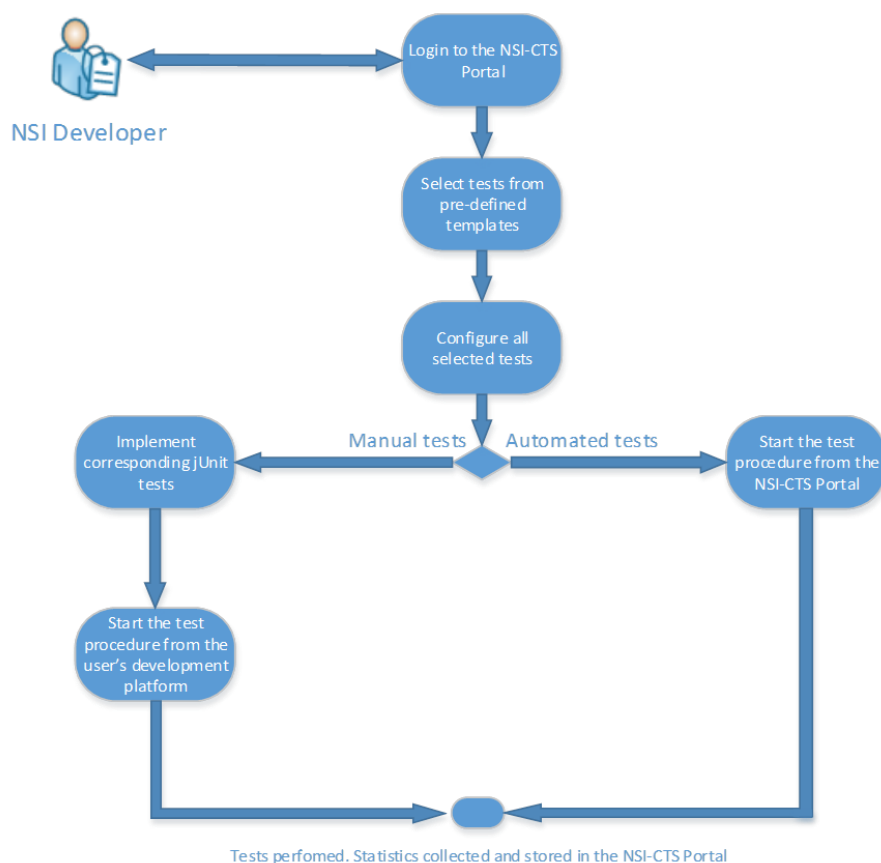


Figure 2.1: Workflow to execute a test

Step	Description	
Step 1	The NSI developer registers to the NSI CTS portal.	
Step 2	The NSI developer select the tests to be run from pre-defined templates.	
Step 3	The NSI developer configures the tests for running (e.g. specifying parameters like timeout, NRM behaviour, etc.)	
Step 4	Automated Tests: After the developer's trigger, tests are automatically executed by the NSI CTS.	Manual Tests: The test procedure is initiated from the user's development platform (e.g. Eclipse). This option requires further actions from the developer, e.g. the implementation of relevant jUnit test cases.
Step 5	Results are collected and stored in the NSI CTS portal.	

Table 2.2: Steps to execute a test

2.1.2 Getting test results

The NSI CTS web portal allows the NSI developer to collect the results of the performed tests, as shown in Figure 2.2 and Table 2.3.

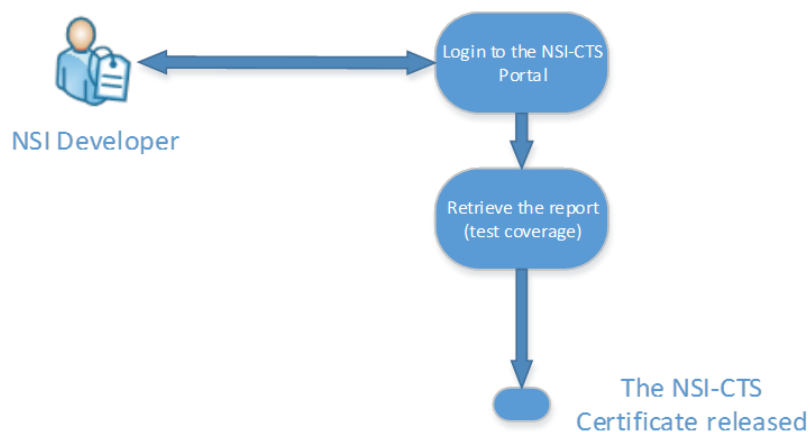


Figure 2.2: Workflow to collect the results for a test

Step	Description
Step 1	The NSI developer logs in to the NSI CTS portal.
Step 2	After successful login, the NSI developer retrieves the test report, which includes the test coverage.
Step 3	The NSI developer receives the NSI-CTS certificate, which summarizes the compliance with the NSI v2.0 standard.

Table 2.3: Steps to collect the results for a test

2.1.3 Continuing tests

The NSI developer can execute a sequence of different tests in a single or different sessions. The workflow for the single test execution is the same specified in section 2.1.1. The workflows is represented in Figure 2.3 and Table 2.4.

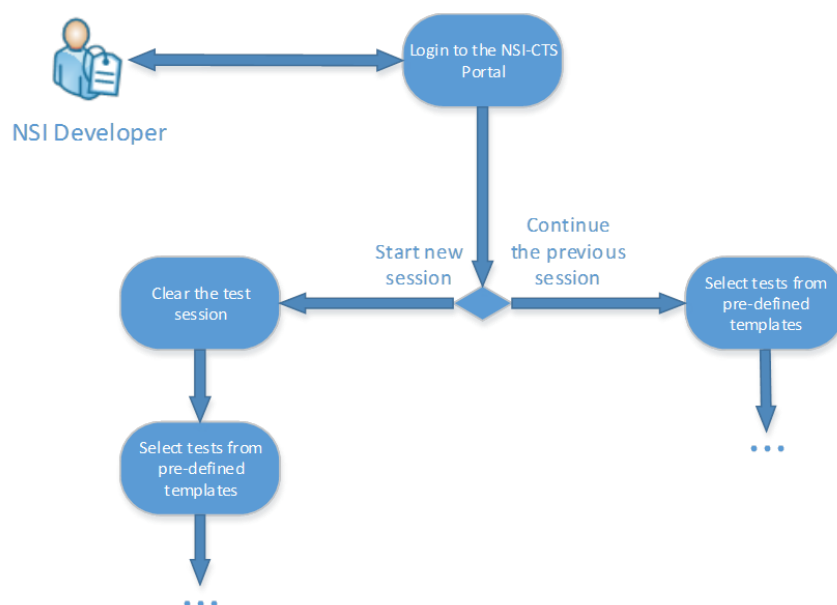


Figure 2.3: Workflow to continue tests

Step	Description	
Step 1	The NSI developer logs in to the NSI CTS portal.	
Step 2	The NSI developer decides to start a new testing session, or continue the previous one (e.g. to increase the coverage of tests).	
Step 3	Start a new session: The NSI-CTS clears the testing session. The user proceeds with a selection of pre-defined tests from the portal.	Continuation of a previous session: The user selects new tests to be performed and continues with testing.

Table 2.4: Steps to continue tests

2.2 Testing Requester role

A scenario which tests a Requester NSA is used to validate an NSI client application and involves two entities, as shown in Figure 2.4:

- The NSI DUT which acts as the ultimate Requester NSA
- An instance of the NSI Reference Implementation (NSI-RI) which acts as a generic Provider NSA. In particular, this NSI RI instance could either represent the ultimate provider or an aggregator, thus hiding all the complexity of an inter-domain topology (e.g. the generation of errors related to remote domains).

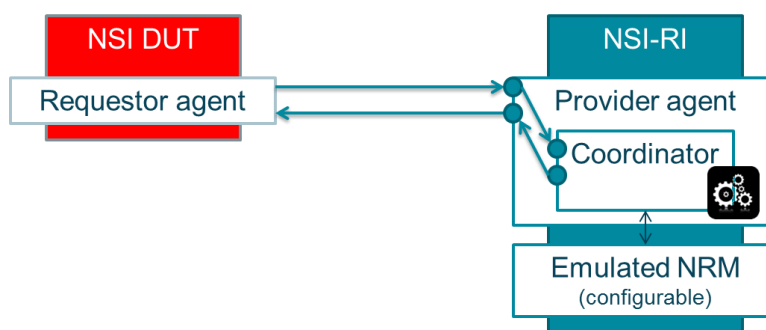


Figure 2.4: Scenario to test a Requester Agent

This type of test is initiated by the DUT which sends a set of NSI request messages. The NSI-CTS does not trigger the requester, but it may provide the user with guidance about how to proceed according to the test definition. In order to evaluate the DUT behavior, the NSI CTS collects the messages sent by the DUT requester and validates them according to the test template and configuration through the NSI-RI. In particular, the NSI-RI instance performs the following three actions:

1. Validation of the formatting of the received request messages.
2. Validation of the sequence of the received messages according to the NSI FSMs and verification of their contents and parameters according to the test configuration.
3. Generation of events and messages, compliant with the NSI FSMs, according to the configuration specified by the user.

2.3 Testing Provider role

A scenario which tests an ultimate Provider NSA is used to validate an NSA which manages a single domain. As in the previous case, it involves only two entities (Figure 2.5):

- The NSI DUT which acts as the ultimate Provider NSA
- An instance of the NSI Reference Implementation (NSI-RI) which acts as a generic Requester NSA. This NSI RI instance could either represent the ultimate requester or an aggregator that hides all the chain of previous domains.

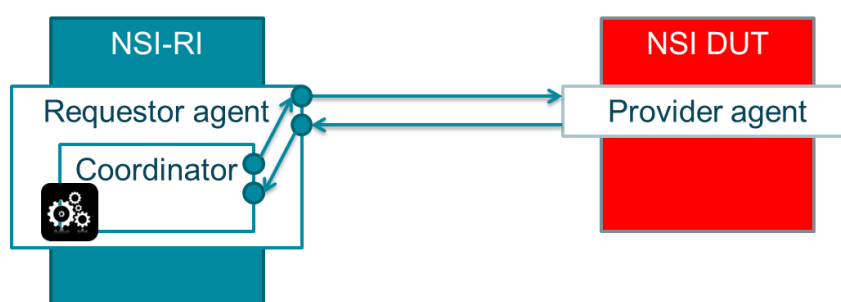


Figure 2.5: Scenario to test a Provider Agent

The NSI CTS starts the test triggering the NSI-RI which represents the Requester agent. The NSI-RI generates the messages according to the test scenario selected by the user (e.g. a full reservation cycle with reserve, reserveCommit and provision phases). The user needs also to specify some additional service parameters in the configuration, for example the service URL, the provider NSA, or the pair of termination points (STPs). As further option, some of these parameters may be also retrieved automatically from the topology service. However, this would require a standard mechanism and protocol to advertise and exchange topology information.

During the test execution, the NSI-RI checks the formatting of the NSI messages received from the DUT and validate the sequence and the content of the messages according to the NSI FSMs and the expected evolution of the NSI session, as declared in the test description provided by the user.

2.4 Testing Aggregator role

The aggregator role scenario is adopted in multi-domain environments, where the DUT provides the NSI implementation for a domain in the middle of the topology. In general, both chains and tree topology structures are supported. However, it is a task of the user to provide the list of the domains involved in the connection service and the list of the domains which will be contacted by the aggregator.

The entities involved in the aggregator role scenarios are shown in Figure 2.6.

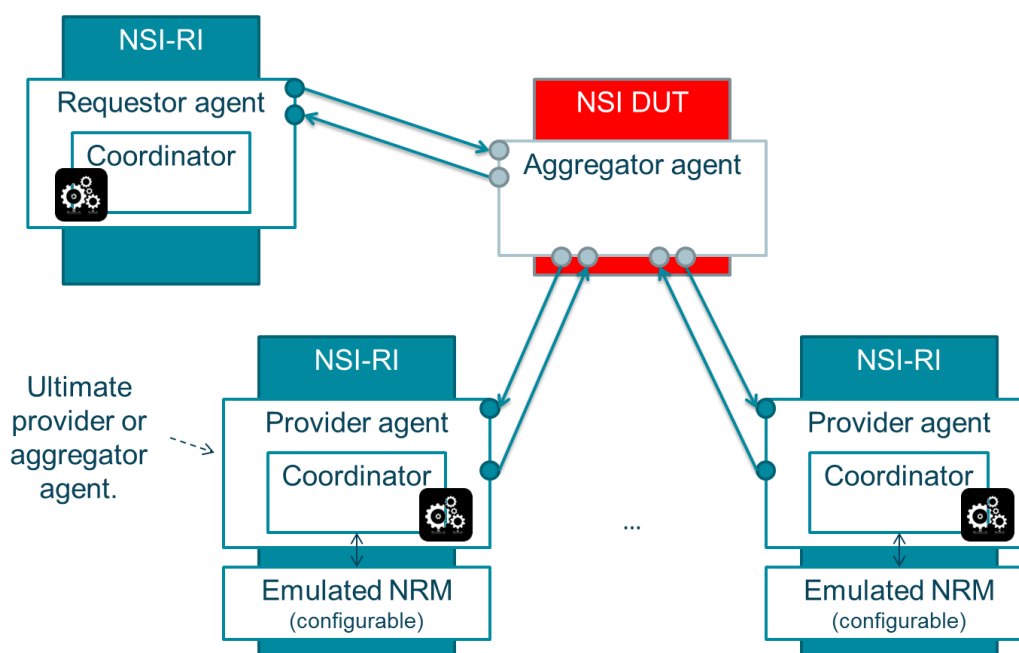


Figure 2.6: Scenario to test an Aggregator Agent

The NSI DUT acts as Aggregator Agent and receives NSI request messages from an NSI-RI instance with the role of the Requester Agent. This single instance may represent the ultimate Requester or another intermediate Aggregator which invokes the DUT. Depending on the scenario configuration, one or

more NSI-RI instances with the role of Provider Agents must be deployed to simulate the provisioning of the network resources and the responses to the NSI requests generated by the DUT. Each NSI-RI with the Provider Agent role may represent an ultimate provider or another aggregator, hiding the entire chain of the following agents. The behaviour of each NSI-RI can be configured individually, e.g. in terms of NRM processing delay, resource availability, generation of error messages. This approach allows to emulate the behaviour of complex environments through the simple configuration of a limited number of NSI-RIs, independently on the real dimension of the desired topology.

3 NSI Compliance Testing Suite System Architecture

The NSI Compliance Testing Suite provides an online service which can be used by NSI software developers to verify the conformance of their NSI Agent implementation to the protocol standard of the NSI Connection Service v2.0. As shown in Figure 3.1, the NSI CTS offers a web interface where the user can register to the NSI-CONTEST service, request and configure a new test session, start the execution of the test and collect the final results. The CTS platform supports two types of user interfaces: (i) a Graphical User Interface (GUI) through a web portal which can be accessed through a common web browser and (ii) a RESTful API which can be used to automate all the procedures for test creation, configuration and execution.

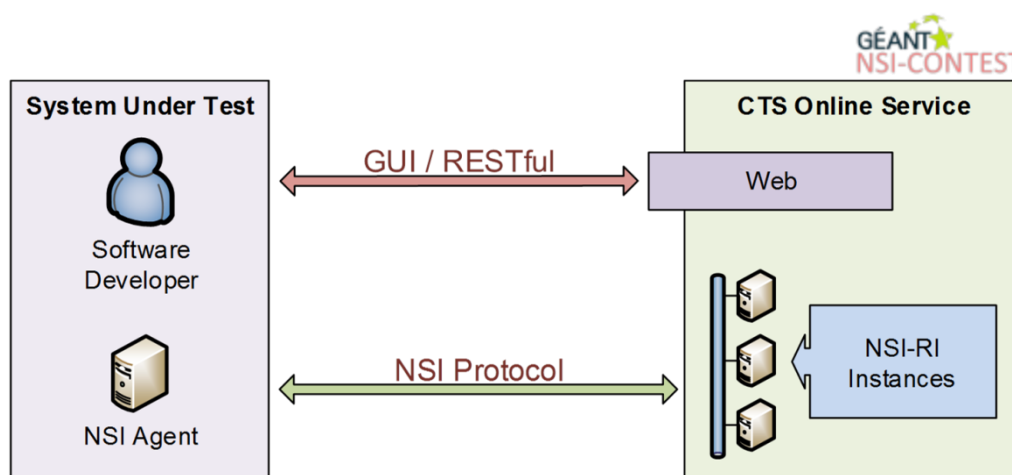


Figure 3.1: NSI CS v2.0 Compliance Test Suite

The NSI developer can use the NSI CTS interface to select the desired test template from the pre-defined set of requester, provider or aggregator roles (see sections 2.2 - 2.4) and to configure a set of parameters which define a customized testing scenario. The NSI CTS platform is responsible to deliver on-demand an isolated testing environment with a number of NSI-RI instances deployed and configured according to the specification provided in the user request. When a test is executed, the NSI Agent under test exchanges NSI protocol messages with the NSI-RI instances of the related testing environment. These distributed NSI-RI instances are responsible to analyse the incoming traffic, react in compliance with the FSMs defined in the NSI specification and according to the workflow configured by the user in the test descriptor, verify the correct format of the received messages and their

compliance with the logical evolution of the NSI session. The local test results generated in each NSI-RI instance are stored internally and forwarded to a centralized collector where they are combined to compile the final test result.

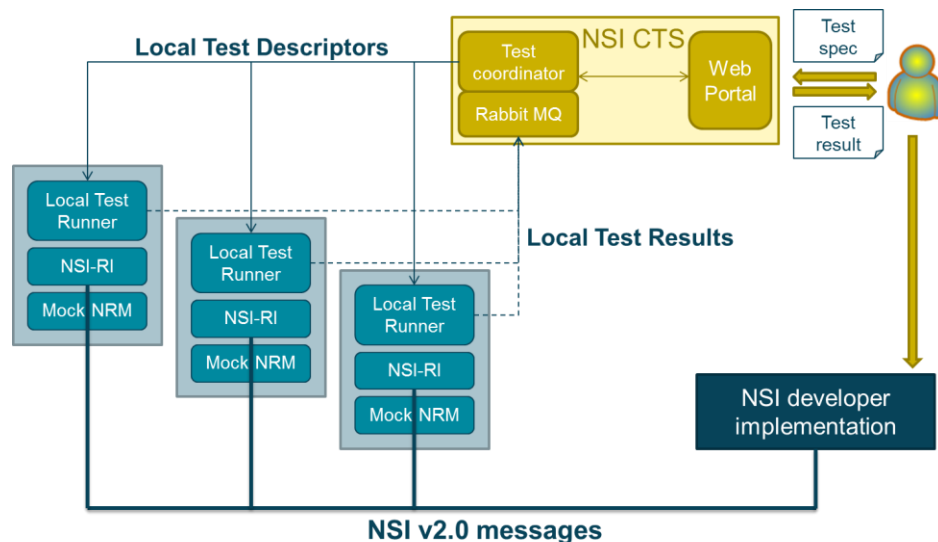


Figure 3.2: High-level architecture

The interaction between the centralized core engine backend in the NSI CTS and the distributed NSI-RI instances is shown in Figure 3.2. The test description (*Test spec*) specified by the NSI developer in the web portal through the test template and its configurable parameters are elaborated in the test coordinator. This module builds the local test descriptors which constitute the configuration input for the distributed NSI-RI instances and coordinates the execution of the whole test. The interaction between test coordinator and NSI-RI instances is based on a RESTful interface (see section 3.2.1) and it is mediated through an entity called Local Test Runner. It is responsible to manage the local test execution in each NSI-RI node, configuring the behaviour of the mock NRM that emulates resource allocation in the related network domain, triggering the actions in the NSI protocol stacks and collecting the distributed test results. These preliminary results are stored internally and made available through the REST API, but are also forwarded to a RabbitMQ in the NSI CTS core engine. Finally, based on this distributed information, the test coordinator generates a consolidated test result, which can be retrieved by the user through a RESTful interface or visualized in the web portal.

3.1 NSI CTS Web Portal

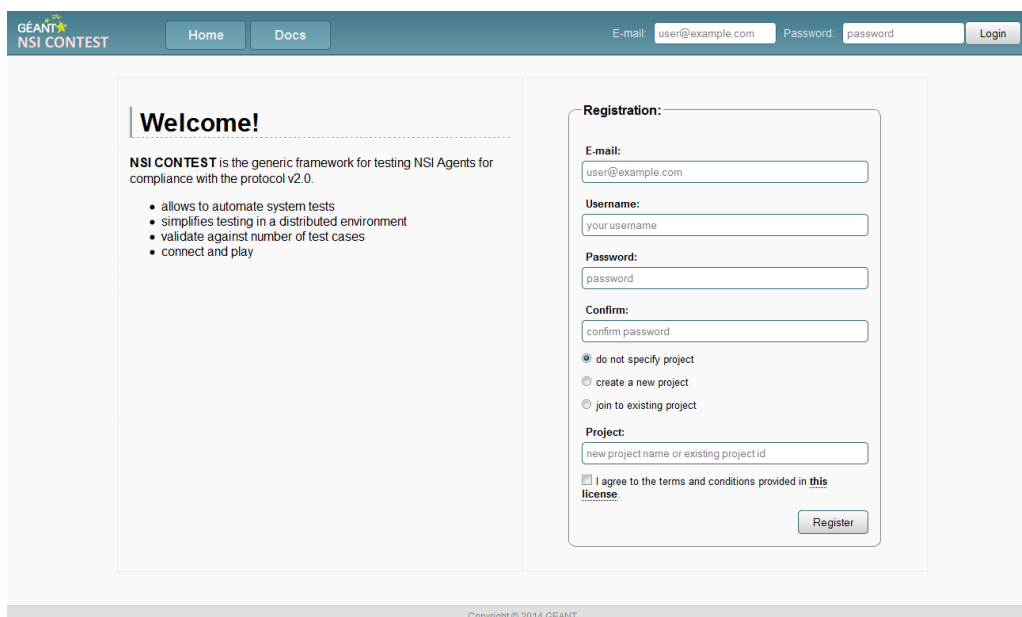
The front end for the NSI-CONTEST platform is a web portal that manage users, test scenarios and coordinates the execution of tests in order to validate the systems under test. There are two web interfaces for the users exposed by the system: web GUI and REST API. The first one is dedicated to the human users while the latter one is for automatic systems (e.g. user external scripts).

The web GUI interface is designed for both external users and the NSI-CONTEST service administrators. The administrative panel is intended to provide an overview of running tests, resource utilization and

allows to manage projects and users. In this section administrators are able to define templates for test scenarios and set the certification test cases.

The User section is focused on project and test execution. The users are able to provide in the system new projects that represent single working instances of the user SUT. The project configuration consist of some initial data (like type of user agent) and topology information exchange for testing scenarios. System generates basic topology for agents that has to be updated within the user agent and the user has to provide its own configuration in order to complete the testing topology. In order to start testing, the user is able to define custom or certification test case that is a set of (pre)selected test scenarios that are executed by the NSI-CONTEST platform to validate the user agent compliance with the NSI CS v2.0 protocol. The user is responsible to provide running instance of its own SUT before starting the testing process and it has to be running during the whole procedure. It means that tests are executed in the given order and the user NSI agent has to be able to handle all the tests without restarting user SUT between them.

To simplify the manual execution of test cases in the web GUI user is also able to utilize a REST API interface (section 3.1.1) that is intended to automate the repeating actions in order to run testing procedures. For example using custom scripts the user may start the SUT instance, run tests and provide the results in Jenkins.



The screenshot shows the NSI CONTEST web portal. At the top, there is a header with the GEANT NSI CONTEST logo, navigation links for 'Home' and 'Docs', and a login section with fields for 'E-mail' (containing 'user@example.com') and 'Password' (containing 'password'), and a 'Login' button. The main content area is divided into two columns. The left column has a 'Welcome!' heading followed by a description of NSI CONTEST as a generic framework for testing NSI Agents for compliance with the protocol v2.0. Below this is a bulleted list of features: 'allows to automate system tests', 'simplifies testing in a distributed environment', 'validate against number of test cases', and 'connect and play'. The right column contains a 'Registration:' form. This form includes fields for 'E-mail' (filled with 'user@example.com'), 'Username' (filled with 'your username'), 'Password' (filled with 'password'), and 'Confirm' (filled with 'confirm password'). Below these fields are three radio button options: 'do not specify project' (selected), 'create a new project', and 'join to existing project'. There is also a 'Project' field with the placeholder text 'new project name or existing project id'. At the bottom of the form is a checkbox labeled 'I agree to the terms and conditions provided in this license.' and a 'Register' button. The footer of the page states 'Copyright © 2014 GEANT'.

Figure 3.3: NSI CTS web portal – Login and registration page

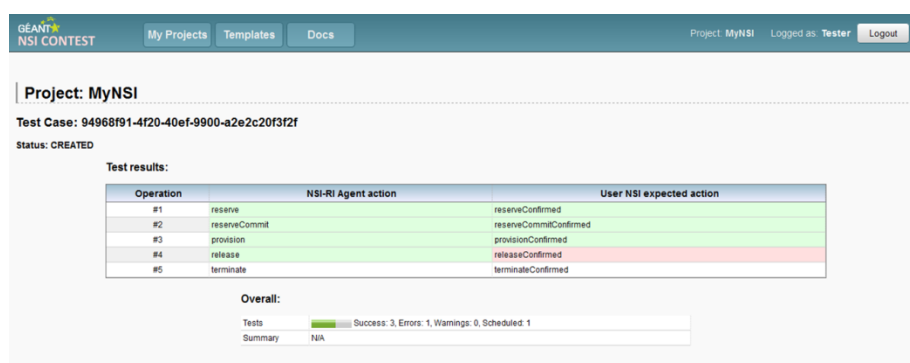


Figure 3.4: NSI CTS web portal – test results

3.1.1 CTS REST interface

As an alternative for the web GUI interface, which is basically designed for human users, the NSI-CTS offers a REST API interface that allows to manage and control tests and test cases in an automatic way. Through this approach it is possible to integrate the NSI-CTS with external applications (e.g. user scripts) that run specific tests against the user NSI agent implementation.

The REST APIs are described in details in Appendix B and they contain the definitions of methods for running test cases with selected tests or running certification process. There is also a set of query methods for retrieving and following the execution of test cases.

3.2 NSI Reference Implementation and Local Test Runner

In the NSI-CTS framework, each test environment is dynamically deployed with a number of NSI nodes running a reference implementation of the NSI CS v2.0 protocol. These nodes are instantiated on demand to create the topology specified by the user and their actions and configuration are managed via REST APIs by the Test Coordinator located in the CTS core engine.

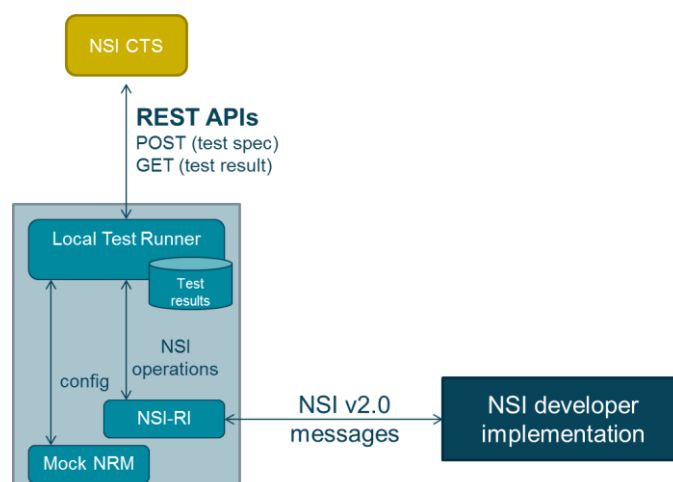


Figure 3.5: NSI-RI and local test runner

Each reference NSI node, as depicted in Figure 3.5, exchanges NSI messages with the other NSI Agents of the test environment, including the NSI Agent developed by the user. Moreover, when acting as a general Provider Agent, it also runs an emulated NRM with a customizable behaviour. Table 3.1 lists the different software components of a reference NSI node.

Component	Description
Local Test Runner	A software module written in java language, using the Spring framework [Spring], which manages the test lifecycle and execution in each NSI-RI node, allocating in a dynamic way new instances of the NSI-RI stack and interacting with them. It exposes a REST API to receive configuration and action commands or informative requests from the centralized NSI CTS backend. On its southbound, it interacts with the NRM for configuration issues and with the NSI-RI to trigger NSI actions. It also intercepts the NSI messages received at the NSI-RI to validate their compliance with the test description and elaborate the local test results which are stored in the database.
Test Results DB	<p>A local HSQL database (HSQLDB [HSQLDB]) used to store the partial results of the test, with details on the actions performed on the local NSI node, the type of messages received from the remote nodes and the errors detected during the test execution.</p> <p>This built-in database is mainly used to temporarily store all the metadata of the NSI-RI instanced launched and terminated in the past, but it is normally wiped clean and recreated at every launch of the Local Test Runner web service, erasing all previous history. However, this behaviour is configurable in case the history of the runs needs to be preserved across multiple stop and restart cycles of the REST web service.</p>
NSI-RI	A software module written in java language which provides the implementation of the NSI-CS v2.0 [NSI-CS-2.0] protocol stack. It includes the functionalities for parsing and formatting of the NSI SOAP messages and for the management of the three FSMs defined by the protocol, i.e. the Reservation State Machine (RSM), the Provision State Machine (PSM) and the Lifecycle State Machine (LSM). It offers java interfaces to trigger NSI actions, like the generation and forwarding of NSI messages and the initiation and management of NSI sessions. The original NSI implementation has been extended with enhanced logging capabilities to detect errors and monitor the different steps in a test execution.
Mock NRM	A software module integrated with the NSI-RI component to emulate the behaviour of the network resource allocation for a Provider Agent. It can be configured to emulate different conditions, including failures or delays in the connection provisioning for the associated domain.

Table 3.1: NSI-RI components

3.2.1 Test Runner REST interface

The interaction between the centralized NSI CTS core engine and the distributed NSI-RI instance is handled by the Local Test Runner module, which implements a web server to offer REST APIs. The content of the HTTP messages is formatted following the json language, where test description is specified with XML strings.

Operation	URI	Description
POST	/job	This operation requests the configuration and execution of a new test with a given local test description.
GET	/job/{jobID}	This operation returns the status of an existing test and, if completed, its report.

Table 3.2: Test Runner REST API

Table 3.2 lists the two operations available on the Test Runner REST API, while the format and parameters of the request and reply messages are specified in Table 3.3 and Table 3.4.

POST /job		
Body of HTTP request:		Body of HTTP response:
<pre>{ "requesterId": "your_name", "xmlData": "... xml data ..." }</pre>		<pre>{ "id": "... some UUID ...", "status": "... some status value ...", "error": "... some error msg ..." }</pre>
Parameter	Type	Description
requesterId	String	The identifier of the requester
xmlData	String with XML data	The local test descriptor, with the specification of the configuration parameters and the desired NSI session evolution (i.e. specification of NSI messages to be sent or received). Some examples of test descriptor are provided in appendix.
Id	String	The unique identifier of the test. It is an uuid value which can be used by the requester in the following requests to track the execution of the tasked job. It is generated by the REST web service engine upon instantiation of a new test, and never reused.
Status	Enum	The status of the test. It can assume the following values: <ul style="list-style-type: none"> QUEUED - job accepted and put in the execution queue, but not yet processed; STARTED - an NSI-RI has been instantiated and possibly started, but the test scenario execution has yet to complete;

		<ul style="list-style-type: none"> ABORTED_ERROR - a major error (due to configuration or other conflict) has occurred. This should normally not happen; COMPLETED_OK - the test scenario execution has completed, the test went through all its stages successfully. The NSI-RI instance is no longer alive and its resources have been collected. A test report XML is available for retrieval. COMPLETED_ERROR - the test scenario execution has completed, the test did not go through all its stages due to unexpected error. The NSI-RI instance is no longer alive and its resources have been collected. A test report XML is available for retrieval.
Error	String	A human readable message to describe the occurred error.

Table 3.3: Test Runner REST API: “POST /job”

GET /job/{jobID}		
--		Body of HTTP response: <pre>{ "id": "{jobId}", "status": "... some status value ...", "testReport": null }</pre>
Parameter	Type	Description
Id	String	The unique identifier of the test, generated by the REST web service engine upon instantiation of a new test and returned in the response of the POST operation.
Status	Enum	The status of the test. It can assume the same values of the POST response, plus the following additional one: <ul style="list-style-type: none"> NOT_FOUND - the specified jobID does not match any known job instance.
testReport	String with XML data	A string containing the XML data of the output test report from the NSI-RI. This string will be populated only after the "status" reaches COMPLETED_OK or COMPLETED_ERROR.

Table 3.4: Test Runner REST API: “GET /job/{jobID}”

The following pictures shows an example of local test descriptor in XML format for a test scenario involving a SUT as Provider (Figure 3.6) and a test scenario involving a SUT as Requester (Figure 3.7). The scenario specification includes three main elements:

- Description: it provides a textual description of the test scenario
- Configuration: it provides the configuration of the testing environment and the involved NSA instances (e.g. listening port, STPs, NSA identifiers)
- Section: it describes the desired evolution of the NSI session through an ordered sequence of operations. Each operation may indicate:
 - a. an action (i.e. send a specific NSI message) with the specification of the expected reaction from the SUT; in this case the operation “*name*” element reflects the name of the NSI message to be sent, e.g. “*reserve*”, while the “*expected*” element indicates which message is expected from the SUT. This type of operation is used to describe scenarios for Provider SUTs, where the NSI-RI is the entity which triggers the NSI requests.
 - b. a passive behaviour to wait for an action from the SUT (i.e. wait to receive a specific NSI message); in this case the operation name element is “*listen*”, while the “*expected*” element still indicates which message is expected from the SUT. This type of operation is used to describe scenarios for Requester SUTs, where the NSI message exchange is initiated by the SUT itself.

The NSI compliance validation is performed checking if a valid message is received from the SUT within the specified time interval and if the received messages correspond to the expected one. A simple check is based just on the message type (e.g. reserve, reserveCommit, etc.), but the test descriptor may also specify further parameters that need to be matched in the message body. This option is enabled through the “*match*” element of the operation statement, as shown in grey in Figure 3.7.

```
<scenario id="1" peer="ciccio">
  <description>TESTCASE_NAME</description>
  <configuration>
    <option name="nsi_listen_port" value="9090" />
    <option name="reply_to_endpoint" value="http://127.0.0.1:9090/nsicontest/ConnectionRequester" />
    <option name="remote_endpoint" value="http://127.0.0.1:9091/nsicontest/ConnectionProvider" />
    <option name="provider_nsa" value="test.provider" />
    <option name="requester_nsa" value="test.requester" />
  </configuration>
  <section id="sect1" interval="500">
    <operation id="oper1" name="reserve" interval="2500" continueOnError="false"
    expected="reserveConfirmed" />
    <operation id="oper2" name="reserveCommit" interval="2500" expected="reserveCommitConfirmed" />
    <operation id="oper2" name="provision" interval="2500" expected="provisionConfirmed" />
    <operation id="oper2" name="release" interval="2500" expected="releaseConfirmed" />
    <operation id="oper2" name="terminate" interval="2500" expected="terminateConfirmed" />
  </section>
</scenario>
```

Figure 3.6: Example of local test descriptor for a Provider SUT

```

<scenario id="1" peer="ciccio">
  <description>TESTCASE_NAME</description>
  <configuration>
    <option name="nsi_listen_port" value="9090" />
    <option name="reply_to_endpoint" value="http://127.0.0.1:9090/nsicontest/ConnectionRequester" />
    <option name="remote_endpoint" value="http://127.0.0.1:9091/nsicontest/ConnectionProvider" />
    <option name="provider_nsa" value="test.provider" />
    <option name="requester_nsa" value="test.requester" />
  </configuration>
  <section id="sect1" interval="500">
    <operation id="oper1" name="listen" expected="reserve" failOperation="false">
      <!-- expectedRequestedNSA="sut.requester.nsi.1"
      <match name="startTime">2014-06-21 15:50:10.000 GMT</match>
      <match name="startTimeMin">2014-06-21 15:50:10.000 GMT</match>
      <match name="startTimeMax">2014-06-21 15:50:10.000 GMT</match>
      <match name="endTime">2014-06-21 15:55:10.000 GMT</match> [...]
      <match name="version">0</match>
      <match name="versionMin">0</match> [...]
      <match name="bidirectional">true</match>
      <match name="symmetricPath">true</match>
      <match name="capacity">9600</match> [...]
      <match name="requestedDestStp">urn:ogf:network:test.provider:2013:topology:egress?vlan=1000-2000</match> [...]
      -->
    </operation>
    <operation id="oper1" name="listen" expected="reserveCommit" />
    <operation id="oper1" name="listen" expected="provision" />
    <operation id="oper1" name="listen" expected="release" />
    <operation id="oper1" name="listen" expected="terminate" />
  </section>
</scenario>

```

Figure 3.7: Example of local test descriptor for a Requester SUT

Examples of test report are show below, with a successful test report (Figure 3.8) and a failed test report (Figure 3.9). For each operation, the test report specifies the result with a boolean value in the “success” element and the received event (i.e. the received NSI message).

```

<section id="sect1" interval="500">
  <operation interval="2500" continueOnError="false" failOperation="false" expected="reserveConfirmed"
  name="reserve" id="oper1">
    <result>
      <success>true</success>
      <receivedEvent>response.reserveConfirmed</receivedEvent>
    </result>
  </operation>
  <operation interval="2500" continueOnError="false" failOperation="false"
  expected="reserveCommitConfirmed" name="reserveCommit" id="oper2">
    <result>
      <success>true</success>
      <receivedEvent>response.reserveCommitConfirmed</receivedEvent>
    </result>
  </operation>
  [...]
</section>

```

Successful result

Figure 3.8: Test report for a successful result

```

<section id="sect1" interval="500">
  [...]
  <operation interval="2500" continueOnError="false" failOperation="false" expected="releaseConfirmed"
  name="release" id="oper4">
    <result>
      <success>false</success>
      <receivedEvent>response.releaseFailed</receivedEvent>
    </result>
  </operation>
  [...]
</section>

```

Failed result

Figure 3.9: Test report for a failed result

3.3 NSI-CTS platform workflows

This section presents in details the workflows and the message exchanges between the different components of the NSI-CTS platform and the SUT when testing the different NSI Agent roles described in sections 2.2, 2.3, and 2.4 (i.e. requester, provider and aggregator roles).

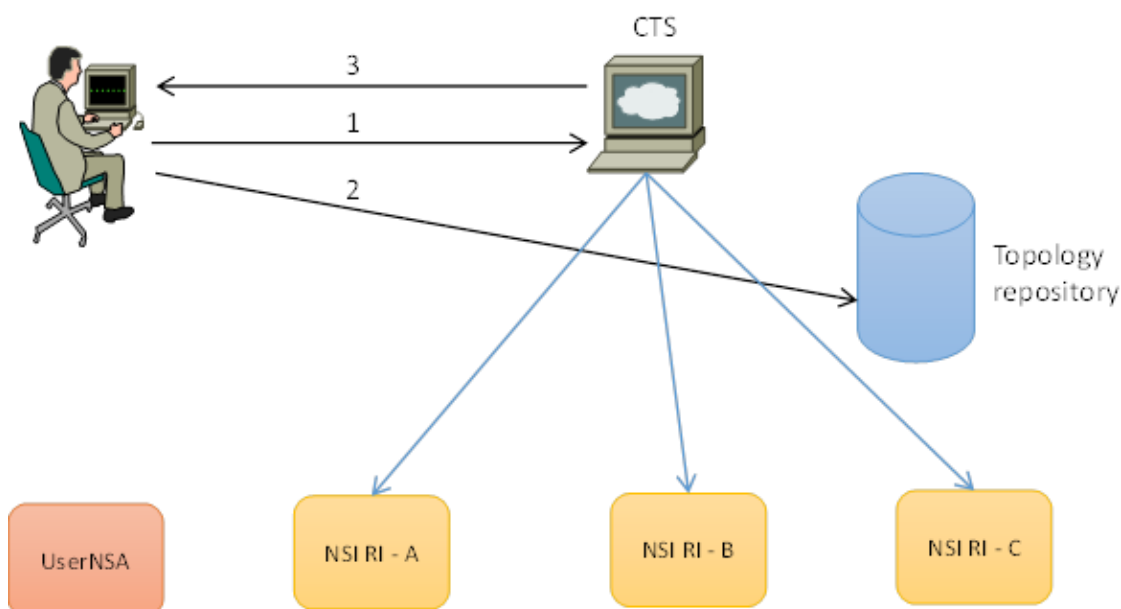


Figure 3.10: Topology exchange

In order to perform a generic User NSA – NSI RI test, topology exchange must occur:

- Each NSA must be aware of other NSAs topology
- Prior to launching test scenarios, user must provide his topology and indicate peering links towards NSI-RIs
- CTS will generate topologies for user and NSI-RIs

Since the way topologies are exchanged is not yet defined in the NSI standard, in the NSI CTS we use a topology repository where topology information is stored (see Figure 3.10).

Independently on the specific NSA role under test, a testing session starts with three common steps, as follows:

1. The user logs into CTS and selects role he/she wants to test (Requester, Provider, Aggregator). Role determines number of NSI-RI instances.
2. The user is asked to prepare his/her topology with peering links towards NSI-RI and upload it to the Topology repository. The CTS validates the topology and generates topologies for the NSI-RI instances, placing them in the repository. The CTS instantiates and preconfigures the NSI-RI instances with specific settings (port, providerNSA, topology).
3. The CTS informs the user about NSI-RI topologies availability and encourages to launch tests.

The following pictures and tables describe the procedures specific for each NSA role.

Provider workflow (Figure 3.11)

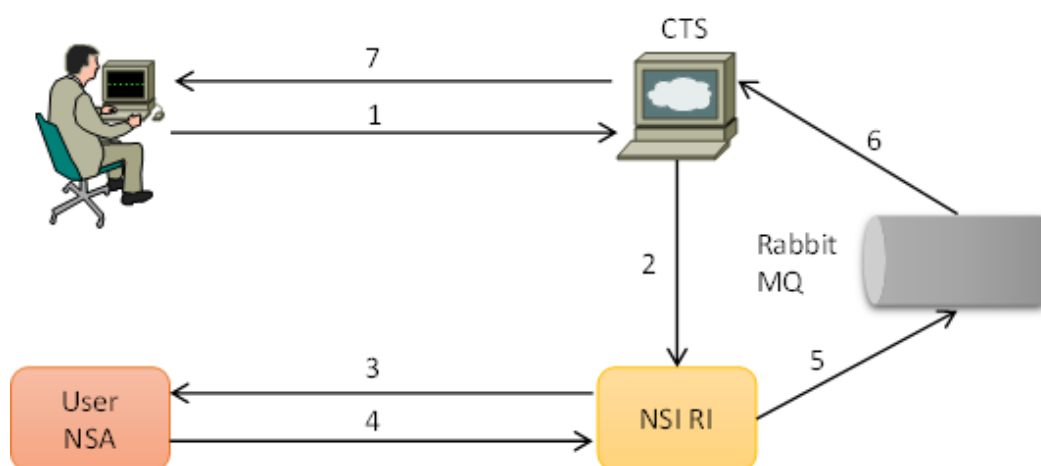


Figure 3.11: Provider workflow

Step	Description
Step 1	The user orders the NSI CTS to run a test through the web portal.
Step 2	The CTS submits the selected test to the Test Runner on the NSI-RI instance.
Step 3	The NSI-RI instance sends NSI messages based on requested test to the User NSA (SUT).
Step 4	The User NSA (SUT) is expected to respond to the NSI-RI instance.
Step 5	The NSI-RI instance analyses the NSI message response and sends results to the RabbitMQ through the Test Runner.
Step 6	The CTS receives notifications of the test results and presents it to the user.
Step 7	The user can view the test results.

Table 3.5: Workflow to test a Provider NSA

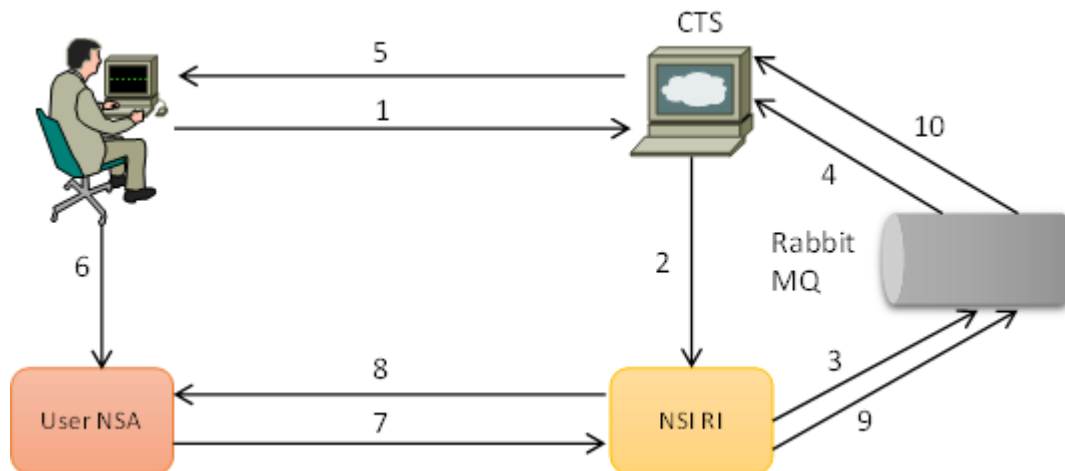
Requester workflow (Figure 3.12)

Figure 3.12: Requester workflow

Step	Description
Step 1	The user orders the NSI CTS to run a test through the web portal.
Step 2	The CTS submits the selected test to the Test Runner on the NSI-RI instance.
Step 3	The NSI-RI instance sends a confirmation to the RabbitMQ – it is ready to receive NSI messages.
Step 4	The RabbitMQ notifies CTS about the NSI-RI instance readiness.
Step 5	The CTS instructs the user to send NSI messages with his/her NSI Agent.
Step 6	The user triggers his/her NSA to send an NSI message destined to NSI-RI.
Step 7	The user NSA delivers a message to the NSI-RI instance.
Step 8	The NSI-RI instance receives the NSI message from the user NSA, validates it and responds to user NSA.
Step 9	The NSI-RI instance also puts results on the RabbitMQ.
Step 10	The RabbitMQ delivers the results to the CTS.
Step 11	The CTS presents the test result to the user.

Table 3.6: Workflow to test a Requester NSA

Aggregator workflow (Figure 3.13)

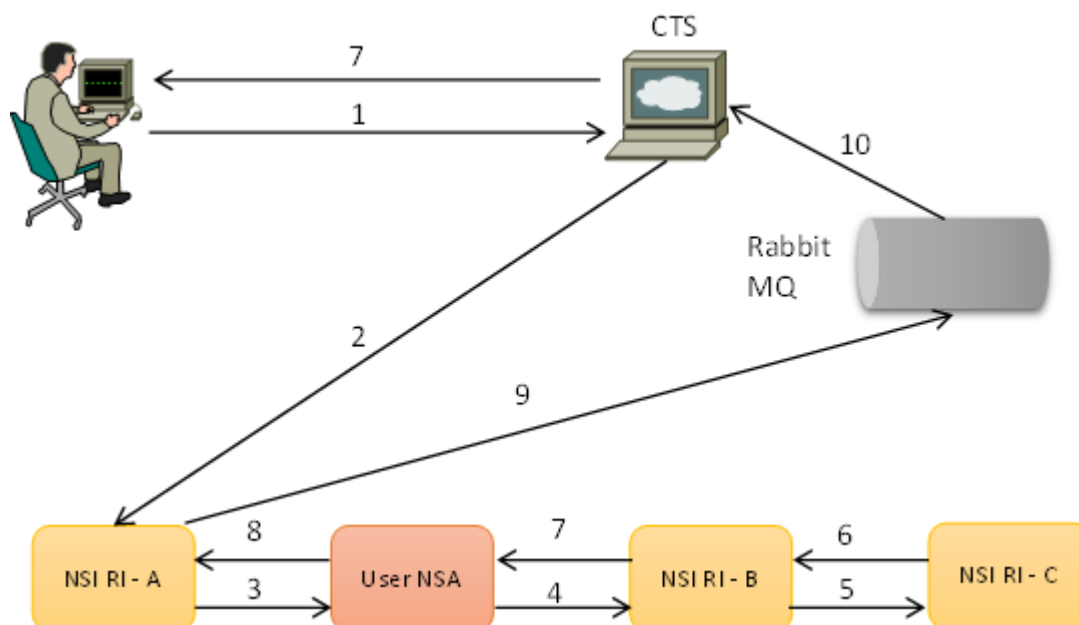


Figure 3.13: Aggregator workflow

Step	Description
Step 1	The user orders the NSI CTS to perform a multi-domain test through the web portal.
Step 2	The CTS uploads test on NSI-RI-A (same step for NSI-RI-B and NSI-RI-C).
Step 3	The NSI-RI-A sends a multi-domain request to the user NSA, also keeps message queue informed (not shown to keep the picture clean, in general every time NSI-RI receives or sends a message, it also make RabbitMQ aware of this fact).
Step 4	The User NSA (SUT) is expected to send a correct NSI message to NSI-RI-B.
Step 5	The NSI-RI-B analyzes the NSI message from the user NSA for correctness and checks if it meets test requirements. If Explicit Route Object is involved, request is passed to NSI-RI-C, otherwise response is dispatched to the user NSA.
Step 6	Response from NSI-RI-C to NSI-RI-B if testing for ERO.
Step 7	The NSI-RI-B sends a reply to the user NSA.
Step 8	The user NSA completes its test by responding to NSA-RI-A
Step 9	The NSA-RI-A puts the final test results on the RabbitMQ.
Step 10	The CTS is notified about the results and presents them to the user.

Table 3.7: Workflow to test an Aggregator NSA

4

NSI CTS Prototype

The software prototype of the NSI CTS is released in a Virtual Machine (VM) where all the software packages which compose the centralized NSI CTS core engine, including the web portal, and the distributed NSI-RI nodes have been already installed and configured for a sample environment. This section describes the VM structure (section 4.1) and provides some guidelines for the configuration of the different components (section 4.2). Finally, section 4.3 describe how to create a new test using the web portal.

4.1 Software release

The NSI CTS software components are released as a single VM “NSI-CONTEST-v1.0”, which can be downloaded at the following link:

<https://box.psnc.pl/f/23decf778d/>

The virtual appliance image is an Ubuntu 14.04.2 LTS based distribution, equipped with all the software packages required to run the NSI CTS components. The following list presents the main libraries and programs installed in the VM:

- Java JDK 1.7
- Maven (and related dependencies)
- Ant
- Python and python-pika
- RabbitMQ server
- Postgresql
- Apache Tomcat 7+
- Groovy Gradle 2+

The VM image, exported as OVA file, can be imported into any OVA compliant virtualization tools (e.g. VMWare Player or Oracle VirtualBox). When up and running, the VM can be accessed through a

remote desktop client and it is configured so that it does not require credentials. Username and password to access directly the VM are:

Username: nextworks

Password: nextworks

The VM is structured in a set of folders located in `/home/nextworks` directory, where the different software modules and configuration files are installed, as detailed in Table 4.1.

Folder	Description
apache-tomcat-7.0.57	The Tomcat web container to run the centralized NSI CTS core engine and web portal (see section 3.1). The .war package is deployed in the webapps/ROOT folder.
nsi-ri	Folder with the source code, the binary files and zip archive of the NSI-RI and mock NRM components (see section 3.2). The zip archive is used by the Local Test Runner to dynamically instantiate the local reference NSA instances.
test-runner	Folder with the source code and the binary files of the Local Test Runner component (see section 3.2).
Sut	A sample SUT NSI implementation for testing and demonstration purposes.
Conf	Folder with sample configuration files of the NSI-CTS web portal, the Local Test Runner and the sample SUT NSI implementation.

Table 4.1: NSI CTS: VM structure

4.2 Installation and configuration guide

The VM is provided with a basic configuration that allows to run the different components of the NSI-CTS platform in a simple environment. The ports defined in the start-up configuration are the following:

Component	Port
CTS web portal	12080
Local Test Runner (REST API)	12090
NSI-RI	12190
SUT (NSI)	12191

Table 4.2: NSI CTS: ports used in the start-up configuration

The user can modify this basic configuration provided in the VM, as described in the following subsections.

4.2.1 CTS configuration

The sample configuration files for the CTS are located in `/home/nextworks/conf/nsi-cts` folder. The general configuration of the CTS must be specified in the `cts.properties` file, where the user can specify the credentials of the administrator, some default values for the profile of new users, timers for maximum test duration and frequency of polling for results requests and the url where the Test Runner APIs can be invoked. The `log4j.properties` file is used to specify logging options, like the logging level and the location of the log file. Finally, the `db.properties` file specifies the database used by the CTS portal. In the start-up configuration, the portal uses a postgresql database, which has been already created in the VM. In a different environment, this database can be created with the following commands:

```
# sudo postgres
```

```
# psql
```

```
-> create user 'cts' with createdb password 'cts';
```

```
# createdb -U cts -W -h localhost cts
```

The example database user and password can be changed and have to be updated in `db.properties` file.

cts.properties – NSI-CTS application configuration

Property name	Description
admin.email	Admin user that is created when CTS is created for the first time when empty database is initialized. Default: <code>admin@localhost</code>
admin.password	Admin password that is assigned when the admin user is created during initialization. Default: <code>nsicts</code> Please change this value before deploying CTS.
user.role.default	This is default role that is assigned to any new registered user. Available roles: <code>ROLE_USER</code> , <code>ROLE_ADMIN</code> . This value should be set to default one. Default: <code>ROLE_USER</code>
https.enabled	Enable https transport. If true all communication with CTS will be redirected to https.

Property name	Description
	Default: false
topology.location	Directory where topology files are stored.
topology.maxFileSize	Maximum size of uploaded topology. Default: 1MB
tests.timeout	Maximum time (in sec.) that single test can run. After exceeding timeout test is rejected and failed. Default: 300
testRunner.url	Url to TestRunner component.

Table 4.3: NSI CTS configuration parameters

db.properties – Database configuration

This file offer to customize database settings. In order to use specific type of DB it is required to install in WEB-INF/lib appropriate database connector (JDBC driver), e.g.:

- PostgreSQL - <https://jdbc.postgresql.org/download.html>
- MySQL - <http://dev.mysql.com/downloads/connector/j/>

For testing purposes it is possible to use build in in-memory database H2 (<http://www.h2database.com>). However in this case all the stored data will be lost after stopping or restarting NSI-CTS service.

Property name	Description
database.driver	Class name for database driver.
database.url	Url to CTS database.
database.username	Username for CTS database.
database.password	Password for CTS database.
hibernate.dialect	Class name for Hibernate dialect for specific database (driver)
hibernate.show_sql	Logs SQL statements. Default: false
hibernate.hbm2ddl.auto	Strategy for updating or creating new database whenever service is started. Please do not modify this value. Default: update

Table 4.4: NSI CTS database configuration parameters

Example configuration for PostgreSQL:

```
# Postgres

database.driver=org.postgresql.Driver

database.url=jdbc:postgresql://localhost:5432/cts

database.username=cts

    database.password=cts

    hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect

    hibernate.show_sql=false

hibernate.hbm2ddl.auto=update
```

Example configuration for PostgreSQL:

```
# MySQL

database.driver=com.mysql.jdbc.Driver

database.url=jdbc:mysql://localhost:3306/cts

database.username=cts

    database.password=cts

    hibernate.dialect=org.hibernate.dialect.MySQL5Dialect

    hibernate.show_sql=false

hibernate.hbm2ddl.auto=update
```

Example configuration for H2:

```
# H2

database.driver=org.h2.Driver

database.url=jdbc:h2:mem:ctsTest
```



```
database.username=
```

```
database.password=
```

```
hibernate.dialect=org.hibernate.dialect.H2Dialect
```

```
hibernate.show_sql=false
```

```
hibernate.hbm2ddl.auto=update
```

log4j.properties – Logging facility

Configuration of NSI-CTS allows to customize logging strategy. User should define appropriate level of logging information and location of output files. More details are available in the Log4J documentation - <http://logging.apache.org/log4j/1.2/> and it is out of the scope of this deliverable

4.2.2 Local Test Runner configuration

The sample configuration file (*application.properties*) for the Local Test Runner is located in */home/nextworks/conf/test-runner* folder. In order to change the REST listen HTTP port, the user can add the following line:

```
server.port = NNNN
```

where NNNN should be over 1024 to allow the Test Runner being run as non-root.

The Local Test Runner includes a built-in HSQL database which is used to temporarily store all the metadata of the NSI-RI instanced launched and terminated in the past. This database, normally, is wiped clean and recreated at every launch of the web service, erasing all previous history. In case the history of the runs is to be preserved across multiple stop and restart cycles of the REST web service, the following line should be added in the *application.properties* file:

```
spring.jpa.hibernate.ddl-auto=update
```

The Local Test Runner starts and manages NSI-RI instances by allocating them dynamically. Each time an NSI-RI run is requested, an NSI-RI engine is copied over a temporary directory and run from there. The NSI-RI serving as the template for instantiation of other instances must be provided to the Local Test Runner engine whilst contained in a zip file. The ZIP file shall contain a recursive archive of a complete (able to run) NSI-RI instance. The "startConsole.sh" script is run to execute the NSI-RI instance. The engine normally checks for the existence of the file in *../nsi-ri/nsi.zip*. This can be overridden in the *application.properties* file, adding the line:

```
nsi-ri-runner.zipFilePath=path/to/nsi.zip
```

4.3 User guide

The NSI CTS platform can be started with the following procedure:

1. Start the SUT (only in case of demonstration scenarios):

```
# cd ~/sut
```

```
# ./start.sh
```

2. Start the Local Test Runner:

```
# cd ~/test-runner
```

```
# ./start.sh
```

3. Start the CTS core engine:

```
# cd ~/apache-tomcat-7.0.57/bin
```

```
# ./startup.sh
```

On the VM desktop the user can also find three icons with shortcuts to automatically start the different components. When the system is up and running, the CTS web portal (see Figure 4.1) can be accessed from a browser at the url: *http://<ip-address>:12080*. The admin user can access the portal with the following credentials (configurable as explained in section 4.2.1):

username: admin@localhost

password: nsicts

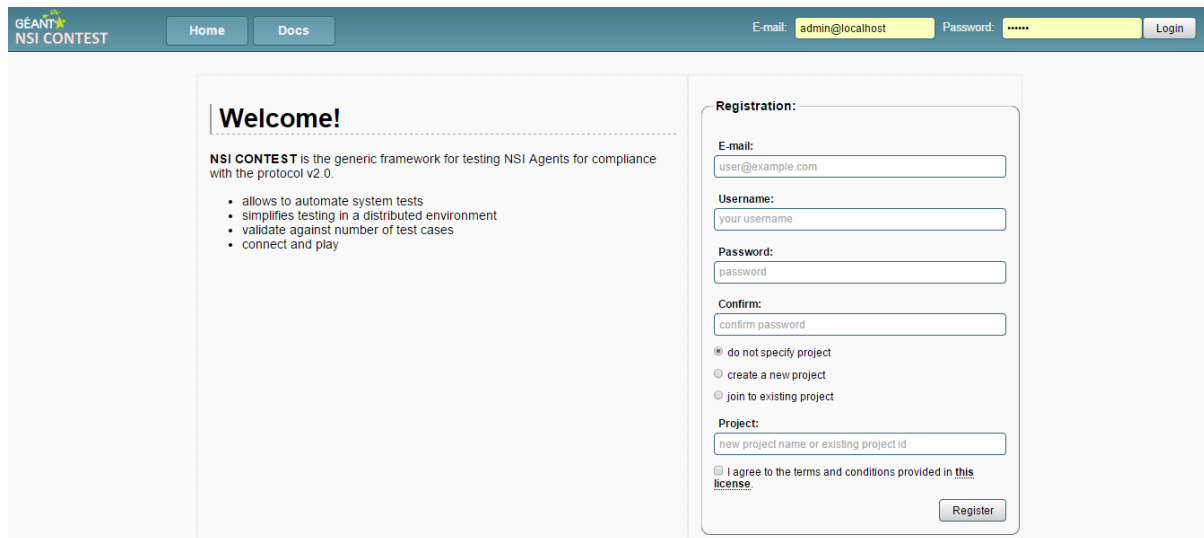


Figure 4.1: CTS web portal main page

User and project registration

The NSI CTS requires user registration in order to start using platform. This also helps to manage individual tests and project. The CTS welcome page (Figure 4.1) provides user registration and project registration/join form that allows to provide the minimum required information to start using whole platform.

During the registration process, the user is able register a new project but he or she is also able to join an existing one by providing its project id (key), created previously by someone else (e.g. from the same development team). This option can also be skipped and a project can be created or joined later after logging in to the platform.

Configuring project

A project represents a single running instance of the user agent that is executed on the user side. For the configuration process the user has to provide some information before start running test. In particular, there are two sections: the former with basic details to be updated after the project registration, the latter is a configuration section to declare the testing topology that will be used in future for all test cases (Figure 4.2).

Initially the user has to generate the testing topology for agent and then download the created files to update user agent topology with the corresponding links/connections to the testing topology. After that, the user has to upload its own user agent domain topology to the CTS. The web GUI provides the summary of the interconnections (links) between the domains represented by all agents. This feature allows to find out any inconsistent entries that may cause issues later during testing.

Notice! Clicking on topology may cause that browser open instead of download xml file that is further interpreted and viewed within the browser. In this case it might be necessary to view the source of the open xml file.

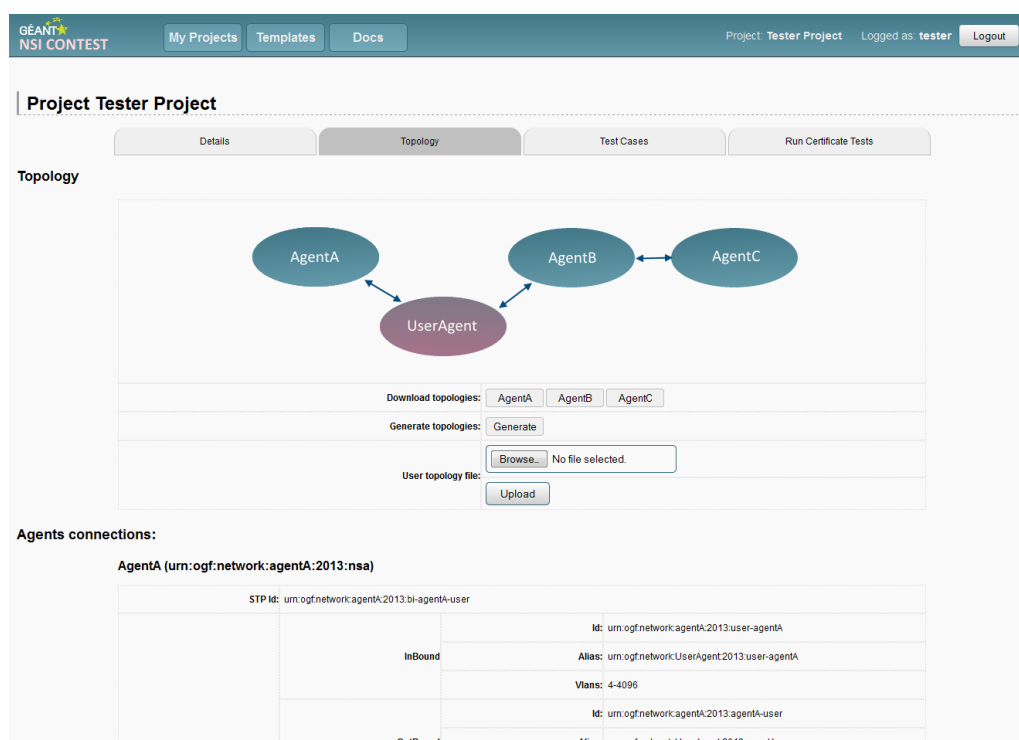
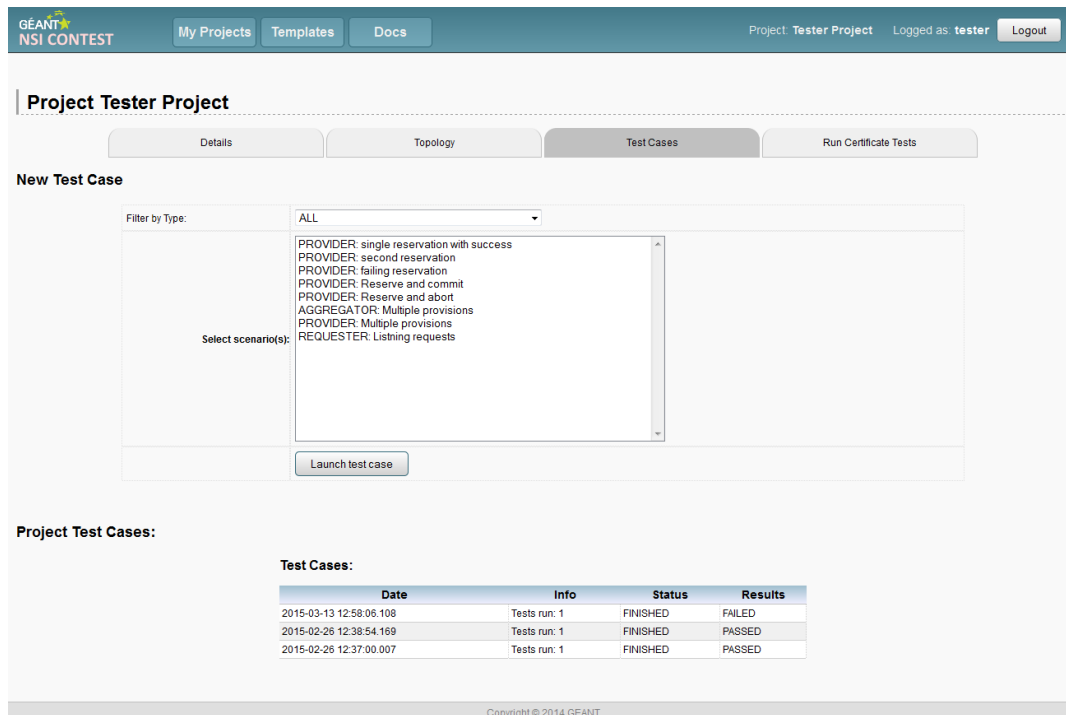


Figure 4.2: Project topology configuration

Running test cases

Creating a new test case requires selection of test predefined scenarios (Figure 4.3) that will be executed by the platform to validate the user NSI agent as a system under test. As each test is independent from any other the execution of all tests within single test case should be independent and user software should not assume any order of the executed tests. In case of running certification tests the list of selected test scenarios is fixed and cannot be changed by the user. Only CTS administrator is allowed to update the certification tests for each agent type but it will affect all projects. However the certification process simplify testing compliance with the NSI protocol for the given agent type.

After starting a new custom or certification test cases user is able to monitor and view the results for each individual executed scenario by clicking on test cases and further on their tests (Figure 4.4).



Project Tester Project

Details Topology **Test Cases** Run Certificate Tests

New Test Case

Filter by Type: ALL

Select scenario(s):

- PROVIDER: single reservation with success
- PROVIDER: second reservation
- PROVIDER: failing reservation
- PROVIDER: Reserve and commit
- PROVIDER: Reserve and abort
- AGGREGATOR: Multiple provisions
- PROVIDER: Multiple provisions
- REQUESTER: Listing requests

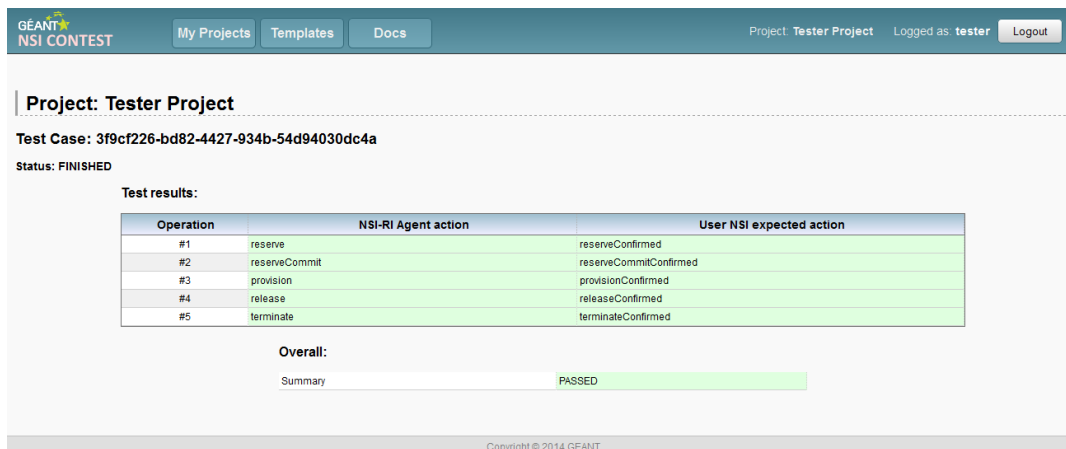
Launch test case

Project Test Cases:

Date	Info	Status	Results
2015-03-13 12:58:06.108	Tests run: 1	FINISHED	FAILED
2015-02-26 12:38:54.169	Tests run: 1	FINISHED	PASSED
2015-02-26 12:37:00.007	Tests run: 1	FINISHED	PASSED

Copyright © 2014 GÉANT

Figure 4.3: Project test cases



Project: Tester Project

Test Case: 3f9cf226-bd82-4427-934b-54d94030dc4a

Status: FINISHED

Test results:

Operation	NSI-RI Agent action	User NSI expected action
#1 reserve	reserveConfirmed	reserveConfirmed
#2 reserveCommit	reserveCommitConfirmed	reserveCommitConfirmed
#3 provision	provisionConfirmed	provisionConfirmed
#4 release	releaseConfirmed	releaseConfirmed
#5 terminate	terminateConfirmed	terminateConfirmed

Overall:

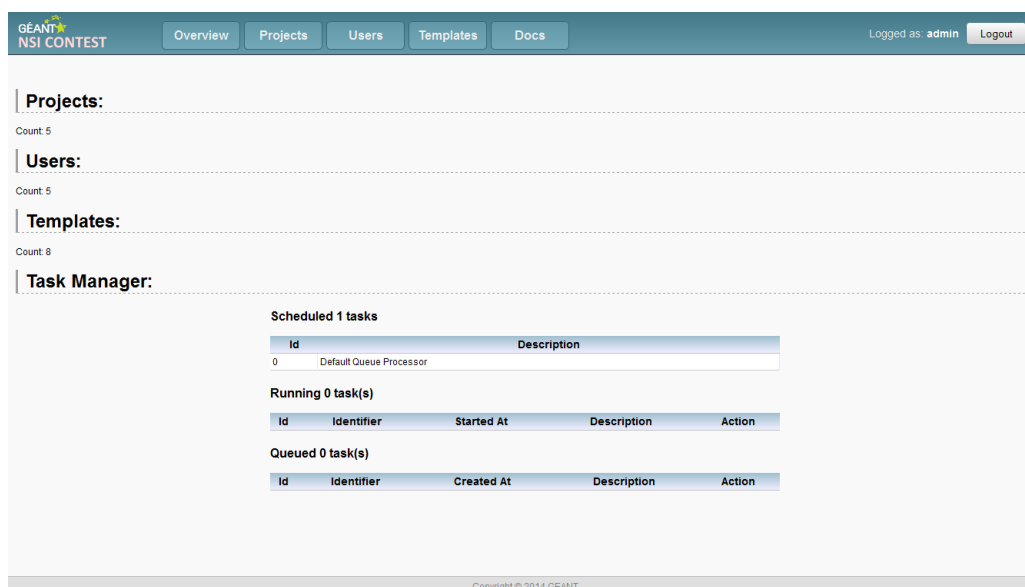
Summary PASSED

Copyright © 2014 GÉANT

Figure 4.4: Single test results

Administration panel

The administration panel (Figure 4.5) is a section of the portal that provides overview and management of users, projects and running tests. The administrator is also responsible for defining common testing scenario templates that should be prepared according to the assumptions of the testing topologies.



Overview Projects Users Templates Docs

Logged as: admin Logout

Projects:

Count: 5

Users:

Count: 5

Templates:

Count: 8

Task Manager:

Scheduled 1 tasks

Id	Description
0	Default Queue Processor

Running 0 task(s)

Id	Identifier	Started At	Description	Action
----	------------	------------	-------------	--------

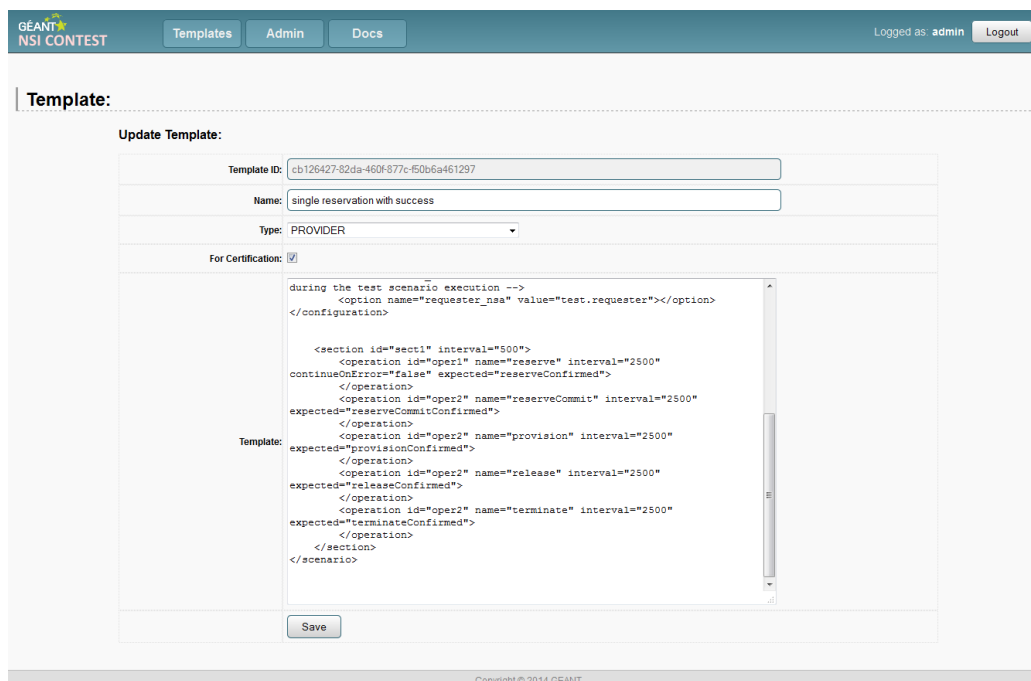
Queued 0 task(s)

Id	Identifier	Created At	Description	Action
----	------------	------------	-------------	--------

Copyright © 2014 GÉANT

Figure 4.5: CTS administration panel - overview

Creating or editing testing scenario templates consist of several information (Figure 4.6). Except the name it is important to select the NSA agent type for the defined scenario – REQUESTER, PROVIDER or AGGREGATOR. In case of adding scenario to the certification process for compliance with NSI v2 protocol “For certificate” should be selected. Changing this parameter it will cause an update for future certification test cases only.



Templates Admin Docs

Logged as: admin Logout

Template:

Update Template:

Template ID: cb126427-82da-460f-877c-f50b6a461297

Name: single reservation with success

Type: PROVIDER

For Certification: ☒

Template:

```

during the test scenario execution -->
<option name="requester_nsa" value="test.requester"></option>
</configuration>

<section id="sect1" interval="500">
  <operation id="oper1" name="reserve" interval="2500"
  continueOnError="false" expected="reserveConfirmed">
  </operation>
  <operation id="oper2" name="reserveCommit" interval="2500"
  expected="reserveCommitConfirmed">
  </operation>
  <operation id="oper2" name="provision" interval="2500"
  expected="provisionConfirmed">
  </operation>
  <operation id="oper2" name="release" interval="2500"
  expected="releaseConfirmed">
  </operation>
  <operation id="oper2" name="terminate" interval="2500"
  expected="terminateConfirmed">
  </operation>
</section>
</scenario>

```

Save

Copyright © 2014 GÉANT

Figure 4.6: Test scenario template form

1.1.2 SUT configuration and CLI commands for demonstration

The VM includes a sample NSI implementation which can be used to emulate a SUT for demonstration purposes. This software is installed in `~/sut` folder and the configuration files are available in `~/sut/etc` folder. Logging parameters can be configured in the `log4j.properties` file, while the NSI stack configuration is located in the `nsi.properties` file. This file is organized in a key-value fashion, meaning that only values can be altered. The configuration can be reloaded at any time with the “reload” command available in the CLI. The configuration parameters available for the SUT are listed in Table 4.5.

Property name	Description
ip	ip address in either canonical or domain form. "localhost" or "127.0.0.1" can be assigned. If this instance is to be accessible from Internet, an ip address assigned to an interface on deployment machine should be set
port	[1024-65535] port number on which this instance will listen on
nsa_id	literal with identifier of this instance. The value of this property is internally used as providerNSA field in CommonHeaderType. Also it determines ownership of topology (topology with nsald that matches this entry will be assigned to this instance)
service_type	describes what type of requests are supported by this NSI-RI. This should always be http://services.ogf.org/nsi/2013/07/descriptions/EVTS.A-GOLE
reserve_held_timeout	[0-360] number of seconds reservation in state HELD will wait for either commit or abort message
fail_if_response_not_delivered	[true false] true to fail operation in case asynchronous response could not be delivered. False to proceed to next state despite confirmation/failure message not being delivered
set_ero	[true false] true to always set ERO in reserve requests. This only works for multi-domain requests
requester_timeout	[0-360] number of seconds requester will be awaiting asynchronous response before timing out
log_header	[true false] true to log common header verification output
log_soap_messages	[true false] true to capture soap messages and write them to a file
log_responder	[true false] true to log responder output. Responder sends confirmation/failure and notification messages
num_threads	[0-1024] size of internal thread pool where task jobs are submitted. Higher value allows more requests to be processed simultaneously
nrm	literal with fully qualified name of the class implementing the Nrm interface. If not set default SimpleNrm implementation will be used
topologies	comma separated file names. These files should be of nml/nsi topology type that will be added NSI-RI global topology

Property name	Description
cli	[true false] true to enable command line running in foreground (if you logout instance will be shut down), or false to run in background
telnet_port	[0 1024-65535] port number for accepting telnet connections. 0 disables telnet server

Table 4.5: SUT configuration parameters

The NSI stack of the sample SUT implementation is supplied with a Command Line Interface (CLI) that can be used to inspect the current state of the NSI reservations, make new NSI requests, and perform topology and nrm management. The CLI can be accessed in two ways:

- as a regular CLI available once the SUT is initialized. This can be enabled by setting the cli property of ns.properties to true.
- as a regular CLI available by connecting with telnet client. This can be enabled by setting the telnet_port property of ns.properties to a number from 1024-65535 range. Note that only localhost connections are allowed (if you connect from a different host it will be rejected).

The SUT CLI supports five main categories of commands:

- **Utility:** allows to reload configuration, show command line help or close NSI-RI
- **Provider:** allows to interact with internal reservations directly (with omission of web service stack). This type of commands always start with the "provider" or "prov" word
- **Requester:** allows to interact with Requester (Provider client). This command can be used to send soap requests to self or other domains. Taking into consideration a large number of parameters required for reserve operation, instead of typing them all, they are read from /etc/request.properties file. This type of commands always start with the "requester" or "req" word
- **Topology:** user can view global topology (topologies uploaded to his NSI-RI) as well add new topologies from file (must be in ns/nml format). This type of commands always start with the "topology" or "topo" word
- **Nrm:** mainly used to export nrm topology into ns/nml topology file. This type of commands always start with the "nrm" word.

The full list of CLI commands are shown in Table 4.6

Command category	Commands	Description
Utility	Exit Quit	Exits application
	Reload	Reloads configuration

Command category	Commands	Description
	Help	Prints help info
Requester	All	Displays all requesters connectionId
	New	Creates a new requester
	Reserve	Sends a reserve message
	Modify [connectionID]	Sends a modify message
	Commit [connectionID]	Sends a commit message
	Abort [connectionID]	Sends a reserveAbort message
	Provision [connectionID]	Sends a provision message
	Release [connectionID]	Sends a release message
	Terminate [connectionID]	Sends a terminate message
	Query [connectionID]	Sends a query message
	Queryall qall [connectionId]	Sends query message to retrieve all reservations
	Querysync qsync [connectionId]	Calls querySync
	Queryrec qrec [connectionId]	Calls query recursive
	Querynot qnot	Calls query notification
Topology	All	Prints all topologies
	This	Prints nsald assigned to the current agent
	Clear	Removes all topologies
	Add [filename]	Adds topology from filename
Nrm	Topology	Calls getTopology on the provided NRM implementation and saves it in NML format
	New	Re-create the NRM
	Impl	Display the NRM implementation name

Table 4.6: SUT CLI commands

A new requester can be created and used for sending NSI Connection Service v2 SOAP messages from the CLI. However, given the large number of parameters in a typical reserve operation, they can be specified in a configuration file of the reservation parameters (`/etc/request.properties`). It is a key-value text file that is read every time the reserve operation is called, meaning it can be altered between reserve(modify) calls.

Property name	Description
endpoint	url where requests will be sent. Endpoints can be found in topology as <nsi:link>
provider_nsa	nsald of receiving agent. This can be found in topology as <nsi:NSA id>
reply_to	url where asynchronous responses from remote Provider should be sent. This should be the Requester service url of this agent, a combination of ip and port values from nsi.properties - http://ip:port/nsicontest/ConnectionRequester
requester_nsa	nsald of agent sending requests. Should equal to <nsi:NSA id> associated with this Requester
reservation_id	optional reservation identifier
description	literal with optional description of reservation
start_time	reservation start time in seconds, blank for immediate start
end_time	reservation end time in seconds, blank for infinite reservation
version	a nonnegative number with initial connection version
service_type	literal with requested service type. Only http://services.ogf.org/nsi/2013/07/descriptions/EVTS.A-GOLE is supported
source_stp	source STP (optionally with vlans)
dest_stp	destination STP (optionally with vlans)
ero	even number of STPs that requested path should go through
capacity	a nonnegative number of requested capacity (bps)
bidirectional	[true false] true if requesting bidirectional path
symmetric_path	[true false] true if requesting symmetric path

Table 4.7: Reserve request configuration parameters

5 Conclusions

This report has presented the architecture of the NSI Compliance Testing Suite (NSI-CTS) and the guidelines to install and use the NSI-CTS software prototype to validate the compliance of NSI Connection Service v2.0 protocol stack implementations with the standard. This document, together with the NSI-CTS prototype released in the related Virtual Machine, constitutes the final outcome of the NSI-CONTEST project.

The prototype has been presented in three demonstrations in the past year, in order to promote the usage of the NSI-CTS service in the NSI development community. The wide usage of this service has the potential to evolve the NSI activities towards a standard-driven approach, where the consistency and the interoperability of different implementations of the NSI protocol stack can be guaranteed through their validation with the common NSI Reference Implementation provided by the NSI-CTS platform.

The design and development of the NSI-CTS have been driven by requirements and continuous feedbacks collected through a strong cooperation with the OGF NSI-WG. A preliminary demonstration of the NSI-CTS concepts has been held during a conference call of the NSI team in September 2014, where the NSI developers have provided suggestions about improvements and new features of the NSI-CTS from the perspective of the final users of the platform. Two further demonstrations have been held at the SuperComputing conference in New Orleans (US) in November 2014 and at the OGF 43 meeting, held in Washington (US) in March 2015 and co-located with the GENI Engineering Conference 22.

Appendix A Sample Scenarios

This section provides some examples of test descriptor in XML format.

A.1 Requester scenarios

A.1.1 Single reservation

The following test descriptor allows to validate the procedures implemented in a SUT acting as Requester agent for a successful reservation with a single provisioning action.

```
<scenario id="1" peer="ciccio">
  <description>TESTCASE_NAME</description>

  <configuration>
    <!-- nsi_listen_port and reply_to_endpoint must be picked in accord. reply_to_endpoint
    point to fully qualified URL reachable from the SUT, whose port bit depends on nsi_listen_port
    -->

    <option name="nsi_listen_port" value="9090"></option>
    <option name="reply_to_endpoint"
value="http://127.0.0.1:9090/nsicontest/ConnectionRequester"></option>
    <!-- remote_endpoint points to the Provider port of the SUT -->
    <option name="remote_endpoint"
value="http://127.0.0.1:9091/nsicontest/ConnectionProvider"></option>
    <!-- provider_nsa is the NSA name the SUT is expected to have -->
    <option name="provider_nsa" value="test.provider"></option>
    <!-- requester_nsa is the NSA name the NSI-RI instance will have during the test
    scenario execution -->
    <option name="requester_nsa" value="test.requester"></option>
  </configuration>

  <section id="sect1" interval="500">
    <operation id="oper1" name="listen" expected="reserve" >
      <match name="startTime">2014-06-21 15:50:10.000 GMT</match>
      <match name="startTimeMin">2014-06-21 15:50:10.000 GMT</match>
      <match name="startTimeMax">2014-06-21 15:50:10.000 GMT</match>
```

```

    <match name="endTime">2014-06-21 15:55:10.000 GMT</match>
    <match name="endTimeMix">2014-06-21 15:55:10.000 GMT</match>
    <match name="endTimeMax">2014-06-21 15:55:10.000 GMT</match>
    <match name="version">0</match>
    <match name="versionMin">0</match>
    <match name="versionMax">0</match>
    <match name="bidirectional">true</match>
    <match name="symmetricPath">true</match>
    <match name="capacity">9600</match>
    <match name="capacityMin">9600</match>
    <match name="capacityMax">9600</match>
    <match
name="requestedDestStp">urn:ogf:network:test.provider:2013:topology:egress?vlan=1000-
2000</match>
        <match
name="requestedSourceStp">urn:ogf:network:test.provider:2013:topology:ingress?vlan=1000-
2000</match>
        <match name="serviceType">http://services.ogf.org/nsi/2013/07/descriptions/EVTS.A-
GOLE</match>
    </operation>

    <operation id="oper1" name="listen" expected="reserveCommit">
</operation>

    <operation id="oper1" name="listen" expected="provision">
</operation>

    <operation id="oper1" name="listen" expected="release">
</operation>

    <operation id="oper1" name="listen" expected="terminate">
</operation>
</section>
</scenario>

```

A.2 Provider scenarios

A.2.1 Single reservation

The following test descriptor allows to validate the procedures implemented in a SUT acting as Provider agent for a successful reservation with a single provisioning action.

```

<scenario id="1" peer="ciccio">
  <description>TESTCASE PA RESERV OK</description>

```

```

    <configuration>
      <!-- nsi_listen_port and reply_to_endpoint must be picked in accord. reply_to_endpoint
point to
      fully qualified URL reachable from the SUT, whose port bit depends on nsi_listen_port
-->
      <option name="nsi listen port" value="12190"></option>
      <option name="reply_to_endpoint"
value="http://127.0.0.1:12190/nsicontest/ConnectionRequester"></option>
      <!-- remote endpoint points to the Provider port of the SUT -->
      <option name="remote_endpoint"
value="http://127.0.0.1:12191/nsicontest/ConnectionProvider"></option>
      <!-- provider_nsa is the NSA name the SUT is expected to have -->
      <option name="provider_nsa" value="test.provider"></option>
      <!-- requester_nsa is the NSA name the NSI-RI instance will have during the test
scenario execution -->
      <option name="requester_nsa" value="test.requester"></option>
    </configuration>

    <section id="sect1" interval="500">
      <operation id="oper1" name="reserve" interval="2500" continueOnError="false"
expected="reserveConfirmed">
      </operation>
      <operation id="oper2" name="reserveCommit" interval="2500"
expected="reserveCommitConfirmed">
      </operation>
      <operation id="oper2" name="provision" interval="2500" expected="provisionConfirmed">
      </operation>
      <operation id="oper2" name="release" interval="2500" expected="releaseConfirmed">
      </operation>
      <operation id="oper2" name="terminate" interval="2500" expected="terminateConfirmed">
      </operation>
    </section>
  </scenario>

```

A.2.2 Single failed reservation

The following test descriptor allows to validate the procedures implemented in a SUT acting as Provider agent for a failed reservation.

```

<scenario id="2" peer="ciccio">
  <description>Failed reservation</description>

  <configuration>
    <!-- nsi_listen_port and reply_to_endpoint must be picked in accord. reply_to_endpoint
point to

```

```

    fully qualified URL reachable from the SUT, whose port bit depends on nsi_listen_port
-->
    <option name="nsi_listen_port" value="12190"></option>
    <option name="reply_to_endpoint"
value="http://127.0.0.1:12190/nsicontest/ConnectionRequester"></option>
    <!-- remote endpoint points to the Provider port of the SUT -->
    <option name="remote_endpoint"
value="http://127.0.0.1:12191/nsicontest/ConnectionProvider"></option>
    <!-- provider nsa is the NSA name the SUT is expected to have -->
    <option name="provider_nsa" value="test.provider"></option>
    <!-- requester nsa is the NSA name the NSI-RI instance will have during the test
scenario execution -->
    <option name="requester_nsa" value="test.requester"></option>
</configuration>

<section id="sect1" interval="500">
    <operation id="oper1" name="reserve" interval="2500" continueOnError="false"
expected="reserveFailed">
        <!-- no parameter is mandatory. Those which are explicitly set here will override
defaults -->
        <parameter name="version">0</parameter>
        <parameter name="bidirectional">true</parameter>
        <parameter name="symmetricPath">true</parameter>
        <parameter name="capacity">1000</parameter>
        <parameter name="requestedDestStp">urn:ogf:network:user:2013:bi-
nonexistent</parameter>
        <parameter name="requestedSourceStp">urn:ogf:network:user:2013:bi-
nonexistent</parameter>
        <parameter
name="serviceType">http://services.ogf.org/nsi/2013/07/descriptions/EVTS.A-GOLE</parameter>
    </operation>
</section>
</scenario>

```

A.2.3 Reserve abort

The following test descriptor allows to validate the procedures implemented in a SUT acting as Provider agent which receives a reservation request followed by a reserve abort request.

```

<scenario id="3" peer="ciccio">
    <description>Successful reserve with abort</description>

    <configuration>
        <!-- nsi_listen_port and reply_to_endpoint must be picked in accord. reply_to_endpoint
point to

```

```

    fully qualified URL reachable from the SUT, whose port bit depends on nsi_listen_port
-->
    <option name="nsi_listen_port" value="12190"></option>
    <option name="reply_to_endpoint"
value="http://127.0.0.1:12190/nsicontest/ConnectionRequester"></option>
    <!-- remote endpoint points to the Provider port of the SUT -->
    <option name="remote_endpoint"
value="http://127.0.0.1:12191/nsicontest/ConnectionProvider"></option>
    <!-- provider nsa is the NSA name the SUT is expected to have -->
    <option name="provider_nsa" value="test.provider"></option>
    <!-- requester nsa is the NSA name the NSI-RI instance will have during the test
scenario execution -->
    <option name="requester_nsa" value="test.requester"></option>
</configuration>

<section id="sect1" interval="500">
    <operation id="oper1" name="reserve" interval="2500" continueOnError="false"
expected="reserveConfirmed">
        <!-- no parameter is mandatory. Those which are explicitly set here will override
defaults -->
        <parameter name="version">0</parameter>
        <parameter name="bidirectional">true</parameter>
        <parameter name="symmetricPath">true</parameter>
        <parameter name="capacity">1000</parameter>
        <parameter name="requestedDestStp">urn:ogf:network:user:2013:bi-user-
nsa2</parameter>
        <parameter name="requestedSourceStp">urn:ogf:network:user:2013:bi-user-
nsa1</parameter>
        <parameter
name="serviceType">http://services.ogf.org/nsi/2013/07/descriptions/EVTS.A-GOLE</parameter>
    </operation>

    <operation id="oper2" name="reserveAbort" interval="2500"
expected="reserveAbortConfirmed">
        <!-- this operation does not support specifying parameters -->
    </operation>
</section>
</scenario>

```

A.2.4 Multiple provisions

The following test descriptor allows to validate the procedures implemented in a SUT acting as Provider agent for multiple provisioning requests within a single reservation.

```
<scenario id="4" peer="ciccio">
```



```

<description>Multiple provisions</description>

<configuration>
  <!-- nsi_listen_port and reply_to_endpoint must be picked in accord. reply_to_endpoint
point to
  fully qualified URL reachable from the SUT, whose port bit depends on nsi listen port
-->
  <option name="nsi listen port" value="12190"></option>
  <option name="reply to endpoint"
value="http://127.0.0.1:12190/nsicontest/ConnectionRequester"></option>
  <!-- remote endpoint points to the Provider port of the SUT -->
  <option name="remote_endpoint"
value="http://127.0.0.1:12191/nsicontest/ConnectionProvider"></option>
  <!-- provider_nsa is the NSA name the SUT is expected to have -->
  <option name="provider_nsa" value="test.provider"></option>
  <!-- requester_nsa is the NSA name the NSI-RI instance will have during the test
scenario execution -->
  <option name="requester_nsa" value="test.requester"></option>
</configuration>

<section id="sect1" interval="500">
  <operation id="oper1" name="reserve" interval="2500" continueOnError="false"
expected="reserveConfirmed">
    <!-- no parameter is mandatory. Those which are explicitly set here will override
defaults -->
    <parameter name="version">0</parameter>
    <parameter name="bidirectional">true</parameter>
    <parameter name="symmetricPath">true</parameter>
    <parameter name="capacity">1000</parameter>
    <parameter name="requestedDestStp">urn:ogf:network:user:2013:bi-user-
nsa2</parameter>
    <parameter name="requestedSourceStp">urn:ogf:network:user:2013:bi-user-
nsa1</parameter>
    <parameter
name="serviceType">http://services.ogf.org/nsi/2013/07/descriptions/EVTS.A-GOLE</parameter>
  </operation>

  <operation id="oper2" name="reserveCommit" interval="2500"
expected="reserveCommitConfirmed">
    <!-- this operation does not support specifying parameters -->
  </operation>

  <operation id="oper3" name="provision" interval="2500" expected="provisionConfirmed">
    <!-- this operation does not support specifying parameters -->
  </operation>

  <operation id="oper4" name="release" interval="2500" expected="releaseConfirmed">

```

```
<!-- this operation does not support specifying parameters -->
</operation>

<operation id="oper5" name="provision" interval="2500" expected="provisionConfirmed">
<!-- this operation does not support specifying parameters -->
</operation>

<operation id="oper6" name="release" interval="2500" expected="releaseConfirmed">
<!-- this operation does not support specifying parameters -->
</operation>

<operation id="oper7" name="terminate" interval="2500" expected="terminateConfirmed">
<!-- this operation does not support specifying parameters -->
</operation>

</section>
</scenario>
```

Appendix B NSI-CTS REST API for Users

This section provides full description of user interface that is intent to be used by the automatic tools. Through this REST interface it possible to manage and control tests lifecycle by external application, e.g. scripts with curl.

B.1 Calls

Every executed method requires set of headers. The list is as minimal as possible to provide simple usage of NSI-CTS API. REST interface payload accepts and produces data only in JSON format.

Header	Description
Authentication	(Required) Authentication method for every request. Only Basic method is supported with based64 encoded <login>:<password>
Content-Type	(Optional) It has to be set to application/json

The user's credentials has to be valid and exists in NSI-CTS, otherwise the service will return authentication error. It is not possible to retrieve any data without having proper rights to the requested resources.

Example:

Request:

```
GET /api/projects/[resources]
Authentication: Basic XXXXXXXXXXXXXXXX
Content-Type: application/json
```

B.1.1 Retrieve user projects

GET /api/projects		
--		<p>Body of HTTP response contains list of projects that user belongs to:</p> <pre>[{ "key": "uuid1", "name": "Project Name", "networkId": null, "usersCount": 1 }]</pre>
Parameter	Type	Description
key	UUID	The unique identifier of the project, generated by the NSI-CTS when project is created.
name	String	Name of the project
networkId	String	User NSI agent identifier. Null if this value has not been provided by user.
usersCount	Num	Number of users assigned to the project

B.1.2 Get single project information

GET /api/projects/{key}		
--		<p>Body of HTTP response contains project details:</p> <pre>{ "key": "uuid1", "name": "Project Name", "networkId": null, "usersCount": 1 }</pre>
Parameter	Type	Description

key	UUID	The unique identifier of the project, generated by the NSI-CTS when project is created.
name	String	Name of the project
networkId	String	User NSI agent identifier. Null if this value has not been provided by user.
usersCount	Num	Number of users assigned to the project

B.1.3 Get scenario templates

GET /api/templates		
--		<p>Body of HTTP response contains a list of available templates:</p> <pre>[{ "name": "Scenario Name", "type": "PROVIDER", "certification": true, "templateId": "aee40838-51ae-497d-9dfa-06839c815288" }]</pre>
Parameter	Type	Description
name	String	Name of the template
type	String	NSI Agent type that this scenario template is intended for. Values: "PROVIDER", "AGGREGATOR", "REQUESTER"
certification	Boolean	True if template scenario is used for certification
templateId	UUID	Template scenario unique identifier.

B.1.4 Get all project testcases

GET /api/projects/{key}/testcases		
--		<p>Body of HTTP response contains list of performed testcases for the specified project:</p> <pre>[{ "testCaseId": "testcaseUUID", "createdAt": timestamp, "status": "STATUS", "certification": "USERSNSITYPE", "resultStatus": "result STATUS", "testsCount": 1, "tests": null }]</pre>
Parameter	Type	Description
key	UUID	The unique identifier of the project, generated by the NSI-CTS when project is created.
testCaseId	UUID	Identifier of the testcase
createdAt	Timestamp	Timestamp when testcase has been created. Unix timestamp.
status	String	Status of the listed testcase. Values: "CREATED", "SCHEDULED", "INITIALIZING", "RUNNING", "DESTROYING", "FINISHED"
certification	String	Certification testcase for specific NSI Agent type. Values: "PROVIDER", "AGGREGATOR", "REQUESTER"
resultStatus	String	Status of the result. Values: "NA", "PASSED", "PASSED_CONDITIONALLY", "FAILED"
testsCount	Num	Number of tests executed during that testcase

B.1.5 Get single project testcase

GET /api/projects/{key}/testcases/{testCaseId}		
--		<p>Body of HTTP response contains testcase for the specified project:</p> <pre>{ "testCaseId": "testcaseUUID", "createdAt": timestamp, "status": "STATUS", "certification": "USERSITYTYPE", "resultStatus": "result STATUS", "testsCount": 1, "tests": null }</pre>
Parameter	Type	Description
key	UUID	The unique identifier of the project, generated by the NSI-CTS when project is created.
testCaseId	UUID	Identifier of the testcase
createdAt	timestamp	Timestamp when testcase has been created. Unix timestamp.
status	String	Status of the listed testcase. Values: "CREATED", "SCHEDULED", "INITIALIZING", "RUNNING", "DESTROYING", "FINISHED"
certification	String	Certification testcase for specific NSI Agent type. Values: "PROVIDER", "AGGREGATOR", "REQUESTER"
resultStatus	String	Status of the result. Values: "NA", "PASSED", "PASSED_CONDITIONALLY", "FAILED"
testsCount	num	Number of tests executed during that testcase

B.1.6 Create a new project testcase

POST /api/projects/{key}/testcases		
<pre>{ "templateIds": ["templateId1", ..., "templateIdN"] }</pre>		<p>Body of HTTP response contains testcase for the specified project:</p> <pre>{ "testCaseId": "testcaseUUID", "createdAt": timestamp, "status": "STATUS", "certification": "USERNSITYPE", "resultStatus": "result STATUS", "testsCount": 1, "tests": null }</pre>
Parameter	Type	Description
tempatelds	Array of UUID	List of templateId to be executed.
key	UUID	The unique identifier of the project, generated by the NSI-CTS when project is created.
createdAt	timestamp	Timestamp when testcase has been created. Unix timestamp.
testCaseld	UUID	Identifier of the testcase
status	String	Status of the listed testcase. Values: "CREATED"
certification	String	Certification testcase for specific NSI Agent type. Values: null
resultStatus	String	Status of the result. Values: "NA"
testsCount	num	Number of tests executed during that testcase

B.1.7 Create a new certification testcase

POST /api/projects/{key}/testcases/certificate		
<pre>{ "type": " }</pre>		<p>Body of HTTP response contains testcase for the specified project:</p> <pre>{ "testCaseId": "testcaseUUID", "createdAt": timestamp, "status": "STATUS", "certification": "USERNSITYPE", "resultStatus": "result STATUS", "testsCount": 1, "tests": null }</pre>
Parameter	Type	Description
type	String	Type of User Agent type that certification is executed for. Values: "PROVIDER", "AGGREGATOR", "REQUESTER"
key	UUID	The unique identifier of the project, generated by the NSI-CTS when project is created.
createdAt	timestamp	Timestamp when testcase has been created. Unix timestamp.
testCaseId	UUID	Identifier of the testcase
status	String	Status of the listed testcase. Values: "CREATED"
certification	String	Certification testcase for specific NSI Agent type. Values: null
resultStatus	String	Status of the result. Values: "NA"
testsCount	num	Number of tests executed during that testcase

References

[HSQLDB]	http://hsqldb.org/
[NSI-CS-2.0]	https://www.ogf.org/documents/GFD.212.pdf G. Roberts, T. Kudoh, I. Monga, J. Sobieski, J. MacAuley, C. Guok, "GFD-R-P.212 NSI-WG: NSI Connection Service v2.0", May 2014
[Spring]	https://spring.io/

Glossary

API	Application Programming Interface
CTS	Compliance Testing Suite
FSM	Finite State Machine
LSM	Lifecycle State Machine
NRM	Network Resource Manager
NSA	Network Service Agent
NSI	Network Service Interface
NSI-CS	Network Service Interface – Connection Service
NSI-RI	Network Service Interface – Reference Implementation
OGF	Open Grid Forum
PSM	Provision State Machine
REST	REpresentational State Transfer
RSM	Reservation State Machine
SOAP	Simple Object Access Protocol
STP	Service Termination Point
SUT	System Under Test
XML	eXtensible Markup Language