Road-, Air- and Water-based Future Internet Experimentation

Project Acronym:	RAWFIE		
Contract Number:	645220		
Starting date:	Jan 1st 2015	Ending date:	Dec 31st 2018

Deliverable Number and Title	**D6.5: RAWFIE Operational Platform Testing and Integration Report (c)**		
Confidentiality	PU	**Deliverable type**[1]	R
Deliverable File	D6.5	**Date**	14.11.2018
Approval Status[2]	WP Leader, 1st Reviewer, 2nd Reviewer	**Version**	1.1
Contact Person	Damien Piguet	**Organization**	CSEM
Phone		**E-Mail**	Damien.Piguet@csem.ch

[1] Deliverable type: P(Prototype), R (Report), O (Other)
[2] Approval Status: WP leader, 1st Reviewer, 2nd Reviewer, Advisory Board

AUTHORS TABLE

Name	Company	E-Mail
Kakia Panagidi	UoA	kakiap@di.uoa.gr
Kostas Kolomvatsos	UoA	kostasks@di.uoa.gr
Vasil Kumanov	Aberon	Vasil.kumanov@aberon.bg
Marcel Heckel	Fraunhofer	marcel.heckel@ivi.fraunhofer.de
Kiriakos Georgouleas	HAI	Georgouleas.Kiriakos@haicorp.com
Nikolaos Priggouris	HAI	PRIGGOURIS.Nikolaos@haicorp.com
Jason Ramapuran	HES-SO	jason-emmanuel.ramapuram@hesge.ch
Philippe Dallemagne	CSEM	Philippe.dallemagne@csem.ch
Damien Piguet	CSEM	Damien.Piguet@csem.ch
Giovanni Tusa	IES	g.tusa@iessolutions.eu
Miquel Cantero	ROBOTNIK	mcantero@robotnik.es
Ricardo Martins	MST	rasm@oceanscan-mst.com

REVIEWERS TABLE

Name	Company	E-Mail
Giovanni Tusa	IES	g.tusa@iessolutions.eu
Kakia Panagidi	UOA	kakiap@di.uoa.gr

DISTRIBUTION

Name / Role	Company	Level of confidentiality[3]	Type of deliverable
Consortium		PU	R

[3] Deliverable Distribution: PU (Public, can be distributed to everyone), CO (Confidential, for use by consortium members only), RE (Restricted, available to a group specified by the Project Advisory Board).

CHANGE HISTORY

Version	Date	Reason for Change	Pages/Sections Affected
0.1	01.06.2018	First draft from previous iteration and D4.9 c.	All
0.2	June – October 2018	Continuous and collaborative update following the tests executions	All
0.3	01.11.2018	Version for internal review	All
1.0	05.11.2018	Final version	All
1.1	14.11.2018	Wrapped-up for submission	All

Abstract:

The objective of this deliverable is to report about the integration and testing of the RAWFIE system at the end of the third and last development cycle. It presents the status of the interface tests and the verification tests as well as of the integration results. The document is the third release over the three phases/cycles defined in the RAWFIE project. The tests reported in this document were executed during the current iteration if they were not successful at the previous one or if they were newly defined in D4.9.

This deliverable is based on the results of the tasks T6.1 and T6.2, on the work done in WP5, and on the verification tests planning presented in D4.9.

Keywords: Integration, interface tests, verification tests, roadmap

Table of Contents-

<u>List of Figures</u>

List of Tables

The following table gives the abbreviations used across the RAWFIE projects in the documents and deliverables.

Table 1: Common abbreviations

Abbreviation	Meaning
3D	three-dimensional space
ACL	Access Control List
AGL	Above Ground Level
AHRS	Attitude and Heading Reference System
AJAX	Asynchronous JavaScript and XML
AM	Aggregate Manager (of SFA)
AP	Access Point
API	Application Programming Interface
API	Application programming interface
AT	Aerial Testbed
AUV	Autonomous underwater vehicle
B-VLOS	Beyond Visual Line Of Sight
CA	Certification Authority
CAA	Civil Aviation Authority
CAO	Cognitive Adaptive Optimization
CBNR	Chemical Biological Nuclear Radiological
CEP	Circular Error Probability
CPU	Central Processing Unit
CSR	Certificate Signing Request
DETEC	Department of the Environment, Transport, Energy and Communication
DGCA	Directorate General of Civil Aviation
DoA	Description of Actions
EASA	European Aviation Safety Agency
EC	Experiment Controller
ECC	Error Correction Code
ECV	EDL Compiler & Validator
EDL	Experiment Description Language
EDL	Experiment Description Language
EER	Experiment and EDL Repository
EU	European Union
E-VLOS	Extended Visual Line Of Sight
EVS	Experiment Validation Service
FIRE	Future Internet Research & Experimentation
FOCA	Federal Office of Civil Aviation
FPS	Frames Per Second
FPV	First Person View
GAA	German Aviation Act
GIS	Geographic Information System
GNSS	Global Navigation Satellite System
GPIO	General Purpose Input/Output
GPS	Global Positioning System
GUI	Graphical user interface
HD	High Definition
HTTP	Hypertext Transfer Protocol
HW	Hardware

IAA	Irish Aviation Authority
IaaS	Infrastructure as a Service
IDE	Integrated Development Environment
IDE	integrated development environment
IFR	Instrument Flight Rules
IP	Internet Protocol
ISO	International Standards Organization
JDBC	Java Database Connectivity
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
KPI	Key Performance Indicator
LBL	Long Baseline
LDAP	Lightweight Directory Access Protocol
LS	Launching Service
MEMS	MicroElectroMechanical System
MM	Monitoring Manager
MSO	Multi Swarm Optimization
MT	Maritime Testbed
MOM	Message Oriented Middleware
MVC	Model View Controller
NAT	Network Address Translation
NC	Network Controller
NF	Non Functional
ODBC	Open Database Connectivity
OEDL	OMF EDL
OMF	cOntrol and Management Framework
OMF	Orbit Management Framework
OML	ORBIT Measurement Library
OS	Operating System
OTA	Over The Air
P2P	Point to Point
PSO	Particle Swarm Optimization
PTZ	Pan Tilt Zoom
RC	Resource Controller
RC	Resource Controller
RE	Requirement Engineering
REST	Representational state transfer
RIA	Research and Innovation Action
ROS	Robot Operating System
ROV	Remotely Operated Vehicle
RPA	Remotely Piloted Aircraft
RPAS	Remotely Piloted Aircraft System
RPS	Remotely Piloted Station
RSpec	SFA Resource Specification
SaaS	Software as a Service
SAML	Security Assertion Markup Language
SFA	Slice-based Federation Architecture
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SQL	Simple Query Language
SSO	Single-Sign-On
SVN	Apache Subversion
TM	Testbed Manager

TMS	Testbed Manager Suite
TP	Testbed Proxy
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
UI	User Interface
UML	Unified Modelling Language
USV	Unmanned Surface Vehicle
UUV	Unmanned Underwater Vehicle
UxV	Unmanned aerial/ground/surface/underwater Vehicle
VE	Visualization Engine
VT	Vehicular Testbed
VT	Visualization Tool
WCS	Web Coverage Service
WFS	Web Feature Service
WMS	Web Map Service
WPS	Web Processing Service
WSDL	Web Services Description Language
XMPP	Extensible Messaging and Presence Protocol

Table 2 gives the notations commonly used across the present document.

Table 2: Notations

Notation	**Description**
D*X.Y*	Deliverable *X.Y* from the DoW
MS*X*	Milestone *X* from the DoW
WP*X*	Work package *X* from the DoW
OC*X*	Open Call *X*
A*X.Y*	Activity number *Y* in Phase *X*
DL*X.Y*	Deadline number *Y* in Phase *X*
M*X*	Project month number *X*

A glossary is located at the end of this document in Annex, p. 163.

Part I: Executive Summary

The objective of this deliverable is to report on the integration level obtained for the RAWFIE platform after the third development iteration and to give the results obtained during the tests of the interfaces of the RAWFIE components and of their integration into a unified and operational system. It presents the status of the interface tests and the verification tests as well as of the integration results, including the technicalities required for the ensuring that the platform can be used by third parties. The document also lists the principles and procedures related to the integration of third parties (UxV providers, experimenters) to the platform.

The document is organised into 4 parts. The second part (Part II) is the main section, which is structured into two Chapters. Chapter 1 presents the scope of the document, some definitions and abbreviations together with the relation to other RAWFIE deliverables. Chapter 2 describes the various aspects of the integration and testing of the RAWFIE system. It describes the approach and methodology used for describing, performing and reporting the tests and integration verification. It is followed by the integration with external entities (mainly SFA), the integration setup and the results of the tests of the interface and the verification tests performed on the RAWFIE components and system. To make sure that the current RAWFIE system meets the basic performance requirements, a section presents the measured performance of the kafka message bus in different setups. A conclusion is drawn in Part III to assess the overall maturity of the platform in the last iteration of its development.

Annexes are in Part IV of the report.

<u>Part II: Main Section</u>

1 Introduction

1.1 Scope of D6.5

The scope of this document is to present the final results of the tests of the operational platform, together with the status of the component's integration after the 3rd and last project development iteration cycle.

1.2 Definitions

This document makes use of a number of specific terms, which the RAWFIE team understands as defined below:

- **Verification** of a system is the task of determining that the system is built according to its specifications (functionalities developed according to requirements and design specifications);
- **Validation** is the process of determining that the system actually fulfils the purpose for which it was developed (according to the specification);
- **Evaluation** reflects the acceptance of the system by the end users and its performance in the field, which eventually translates into usefulness (always according to user needs and / or performances in the field against realistic scenarios).

1.3 Relation to other deliverables

The work performed in WP6 relies on the outcomes of WP3 and WP4, as well as on WP5 activities, which performed the development and integration of components, according to the roadmap described in D2.2.

D6.5 is an update of D6.3. From a programmatic point of view, it provides a feedback to WP8 Open calls in the form of an assessment of the system readiness for its operation by end users for the identification of final corrections needed.

D6.5 refers to D4.8 and D4.9 (and their earlier iterations) for many aspects, including the architectural concepts, the data model and the components interactions. The testing of the components interfaces and their integration is based on the architecture and design deliverables of WP4, and specifically on the verification scenarios and planning presented in deliverable D4.9. Modifications from the abovementioned scenarios and planning, when present, will be highlighted in the rest of the document.

In spite of its coarse granularity, D2.2 forms the basis for checking the completeness of D6.5 coverage. D2.2 specifies the different rounds of development and the objectives in terms of function, environment, etc. which directly defines the boundaries of the prototype integration or related tasks (see sections 3.3 to 3.10). D6.5 reports on the integration steps and the verification of components once combined with the rest of the RAWFIE system, before the submission of this system to the validation process.

D6.5 refers explicitly to the Verification scenarios defined in D4.3, D4.6 and D4.9 (section 5.1) for the component testing at a high level, which gives emphasis to the integration process and therefore on the interfaces, dependencies and interactions between components. D6.5 reflects this emphasis, focusing on the results of the integration process and on the interfaces, dependencies and interactions between components. D6.5 deals with, and presents, the interface testing results and the high-level testing results, according to verification templates found in D4.6 and D4.9.

As D6.5 is an iteration of D6.3, some verification tests that did not produce completely successful results at the time of writing D6.3 were re-executed for the current iteration and their results are given in this deliverable. Some other tests or parts of tests were removed because they are no longer relevant due to architectural changes. This is clearly indicated beside all concerned test items.

2 Integration & Testing

2.1 Approach

The objective of the Integration & Testing activities, whose results are presented in this deliverable, is to produce the third version of the end-to-end operational prototype of the RAWFIE platform. Following the time-plan defined for Phase 2 of the Integration & Testing roadmap (D2.2), the results reported in this deliverable reflect the integration and testing work carried out by project's partners during the 3rd technical iteration.

Since the approach does not substantially differ from what described in deliverable D6.3 (Integration & Testing during the 2nd iteration), the reader is also invited to refer to Section 2 of D6.3 and its predecessor D6.1 for further details.

As a result of the 2nd Integration & Testing iteration, some suggestions for modifications and improvements to RAWFIE components and interfaces were derived. These suggestions, together with the outcomes of the implementation activities from WP5, and the third version of the requirements from D3.3, have triggered modifications and improvements in the design of components' functionalities and interfaces, being used as inputs for the third version of the RAWFIE architecture (D4.7) and components' specification (D4.8). In turn, the new version of the components' design, was used for defining new interface tests and verification scenarios, or for updating the existing ones in D4.9. D4.9 is therefore the main reference document for the integration and verification tests reported in this deliverable.

2.2 Methodology

Integration testing includes activities where the different software components of the system are combined and tested as a group, to verify both the communication interfaces and end-to-end workflows and functionalities. The reader is invited to refer also to D6.1, Section 2, where further details of the methodology are explained. Here we highlight that, for the purposes of integration testing, the following tests categories are considered in the integration and verification plan (D4.6, D4.9) and, as a consequence, in the present deliverable:

- **Testing of components interfaces:** this kind of tests are performed for all implemented components that provide a software interface to other components (via a REST or SOAP / RPC API) or are capable to send/receive data from Message Bus. As an example of the communication interfaces that need to be verified during system components' integration, following
- **Figure** 1 and **Figure 2**, taken from the D4.8, provide an overview of the several interactions (through different communication technologies) between Frontend Tier components and Middle Tier components, and between Middle Tier components and other system components, respectively.
- **Execution/Testing of verification scenarios**: This involves the execution of all the verification scenarios defined in D4.9, Section 5.1 and can comprise tests whose aim is mainly to verify individual components' functionality – although in most cases they have as prerequisite the existence of other components – as well as end to end scenarios, where several system components are involved

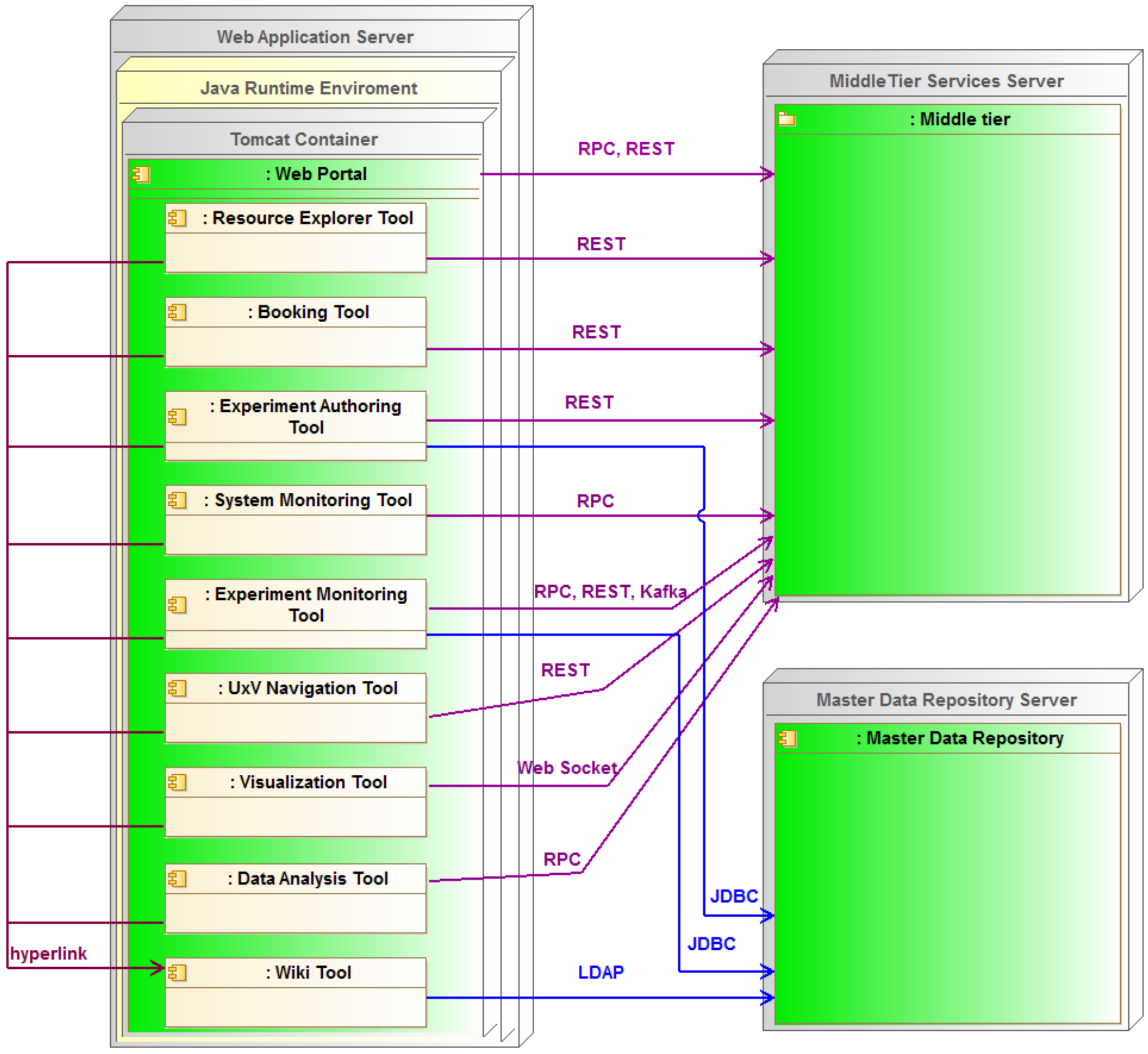


Figure 1: Overview of software interfaces provided by Middle Tier Services and the Master Database, and used by Frontend Tier module

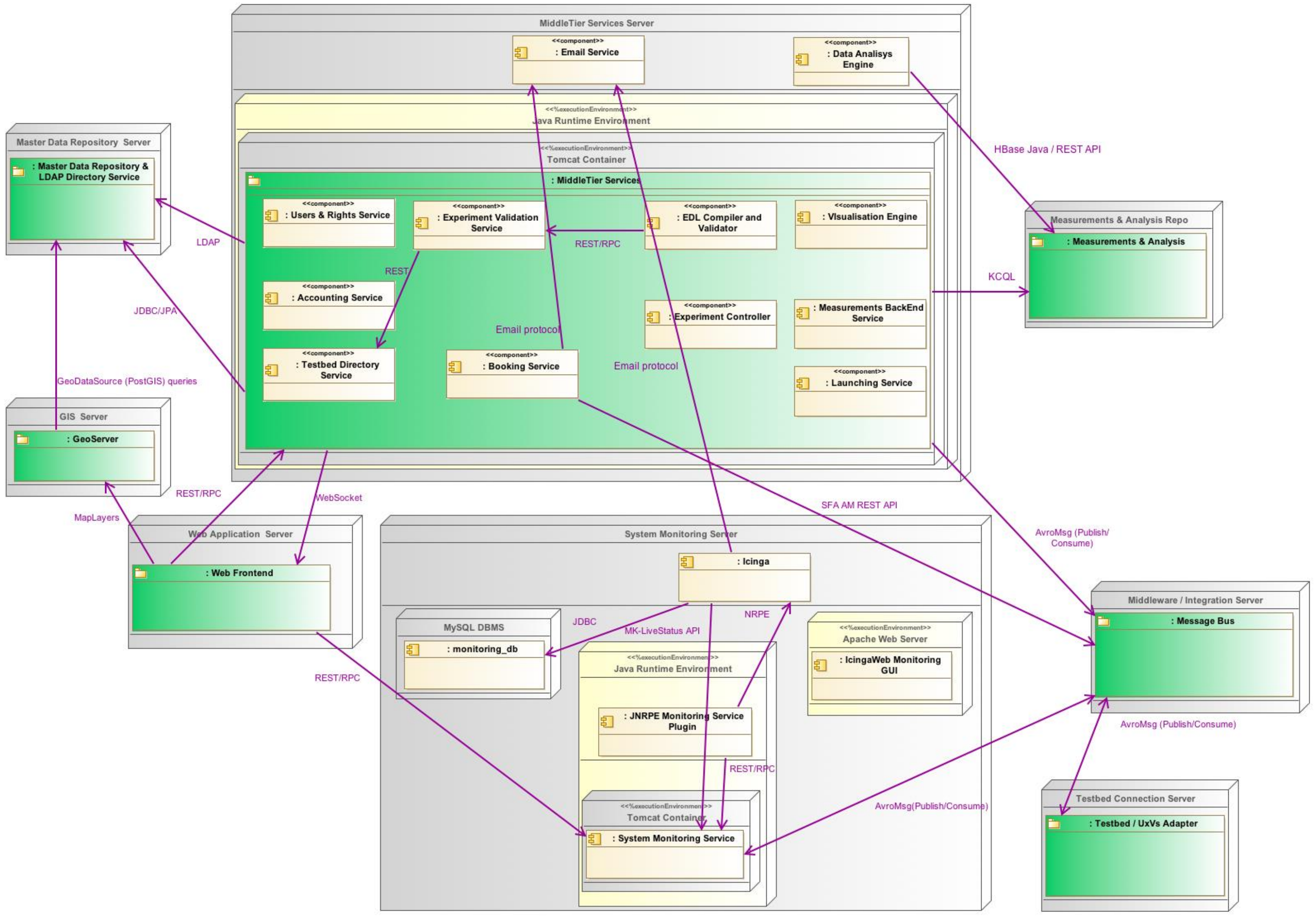

MiddleTier Services Server
<<component>> : Email Service
<<component>> : Data Analisys Engine
<<executionEnvironment>> Java Runtime Environment
<<executionEnvironment>> Tomcat Container
: MiddleTier Services
<<component>> : Users & Rights Service
<<component>> : Experiment Validation Service
<<component>> : EDL Compiler and Validator
<<component>> : Visualisation Engine
<<component>> : Accounting Service
<<component>> : Experiment Controller
: Measurements BackEnd Service
<<component>> : Testbed Directory Service
<<component>> : Booking Service
<<component>> : Launching Service
REST/RPC
REST
Email protocol
Email protocol
Master Data Repository Server
: Master Data Repository & LDAP Directory Service
LDAP
JDBC/JPA
GeoDataSource (PostGIS) queries
GIS Server
: GeoServer
REST/RPC
MapLayers
WebSocket
Web Application Server
: Web Frontend
REST/RPC
HBase Java / REST API
Measurements & Analysis Repo
: Measurements & Analysis
KCQL
SFA AM REST API
AvroMsg (Publish/ Consume)
System Monitoring Server
: Icinga
JDBC
MK-LiveStatus API
NRPE
MySQL DBMS
: monitoring_db
<<executionEnvironment>> Java Runtime Environment
: JNRPE Monitoring Service Plugin
REST/RPC
<<executionEnvironment>> Tomcat Container
: System Monitoring Service
<<executionEnvironment>> Apache Web Server
: IcingaWeb Monitoring GUI
AvroMsg(Publish/Consume)
Middleware / Integration Server
: Message Bus
AvroMsg (Publish/Consume)
Testbed Connection Server
: Testbed / UxVs Adapter

Figure 2: Overview of software interfaces between Middle Tier components, and between Middle Tier components and other system components

2.2.1 Tests reporting format

Results of the verification tests are reported using two different reporting templates, for interfaces testing and for the verification scenarios, respectively. These templates are described in Section 2.2.1 of deliverable D6.1.

2.3 Integration of external components

The integration of new tools and services for the extension of the experimentation capabilities, can be easily realised thanks to the open architecture of RAWFIE, based on a mix of SOA principles (therefore the availability of RPC and REST API) and the decoupling of components and functionalities through the asynchronous communication via the Message Bus.

Specific technical constraints are defined for the integration of new vehicles, testbeds and experimenters, which have been described in the D4.8 in the form of technical guidelines for third parties. In the following subsection 2.3.3 the actual processed to be followed for new testbeds and UxVs integration from both the technical and operational standpoint are reported in details, also considering what has been already done in the open calls cycles for new software, hardware and experimenters integration.

In general, integration procedures for newcomers are available on the project Redmine/Wiki tool in the Work Package 8 section, which is accessible to newcomers unlike the other work package sections which are restricted to the consortium. Software examples are available in Gitlab space shared only with the experimenters.

2.3.1 Interoperability with external SFA clients through the SFA Aggregation Manager

From the technical standpoint, interoperability with external SFA clients is realised through the implementation of a modified version of the SFA Aggregation Manager (AM) at Testbed level, and its integration with existing RAWFIE components. The modified SFA Aggregation Manager is provided in the context of the SAM proposal, who joined the project after the 1st Open Call. It is therefore part of the SAM software module, which will be deployed on each connected Testbed in order to handle, among the others, the reservation process of the respective resources. Please also refer to D4.7 and D4.8 for more details about the components and functionalities of SAM software module.

The following are the main integration scenarios that realise the SFA principles:

- **Adding/Editing/Deleting of resources**. This action will always be performed through the Testbed Manager admin UI. In this scenario the RAWFIE Testbed Manager component will act as the gateway to the SFA Aggregation Manager, since it will forward the modification requests to both the SFA Aggregation Manager using the provided REST API (for updating the local Triple Store DB) and to the

Testbed Directory Service through its REST API, for updating the same information in the centralised Master Data Repository of RAWFIE

- **Listing / searching of resources**. This action can be performed through the RAWFIE platform as well as through external SFA enabled clients / GUI (e.g. MySlice). In the former case, the RAWFIE Resource Explorer Tool and, in turn, the Testbed Directory Service components will be used to search and visualise all or specific UxV resources in the given Testbed. In the latter case, external SFA clients will directly call the SFA Aggregation Manager through the provided REST API. The SFA AM will in turn perform semantic queries to the local Triple Store DB.
- **Booking requests.** This action can be performed through the RAWFIE platform as well as through external SFA enabled clients / GUI (e.g. MySlice). In the former case, the RAWFIE Booking Tool will forward the booking request, through the Booking Service, to the SFA Aggregation Manager using the provided REST API and to the RAWFIE Master Data Repository, so that all repositories will be synchronised. In the latter case, external SFA clients will directly call the SFA Aggregation Manager through the provided REST API and the SFA AM will in turn perform the booking of resources in the local Triple Store DB. The Booking Service will also periodically synchronise itself with the SFA Aggregation Manager, in order to ensure consistency between the reservations made using the SFA interface (and therefore the content of the Triple Store DB), and the ones made using the RAWFIE Booking Tool (Master Data Repository).

2.3.2 Integration of RAWFIE "newcomers"

RAWFIE aims to create a federation of different testbeds that will work together to make their resources available under a common framework. Specifically, it aims at delivering a unique, mixed experimentation environment across the space and technology dimensions. RAWFIE integrates numerous testbeds for experimenting in vehicular (road), aerial and maritime environments. Vehicular Testbeds (VT) will deal with Unmanned Ground Vehicles (UGVs) while Aerial Testbeds (AT) and Maritime Testbeds (MT) will deal with Unmanned Aerial Vehicles (UAVs) and Unmanned Surface Vehicles (USVs), respectively. All these items are managed by a central controlling entity, which will be programmed per case and fully overview/drive the operation of the respective mechanisms (e.g., auto-pilots, remote controlled ground vehicles).

In terms of integration, different actors (UxV providers, experimenters) follow different processes in order to join RAWFIE federation based on their needs. Each actor receives specific guidelines that ensure the correction remote operation of the RAWFIE platform.

2.3.2.1 Integration of a new testbed

RAWFIE searches for improvements in terms of new facilities (testbeds) that could host experiments and devices. First of all newcomers must specify:

- What type of testbed is, i.e. indoor or outdoor
- What type of devices it can host, i.e UAVs, USVs and UGVs

In the RAWFIE project testbeds can host more than one type of devices.

2.3.2.1.1 Requirements

The next step for the testbed providers is to ensure testbeds' compliance with the RAWFIE hard requirements in order to host devices and experiments. Each facility should provide closely monitored and controlled environments and should be able to:

- receive, inspect, assemble/fix and store UxVs
- provide emergency services (i.e., crash, fire or rescue) and recovery processes
- define minimum experimentation time
- have the appropriate equipment, both ground-based and mobile, to monitor and control vehicles, including
    - Radar facilities or other kinds of equipment (e.g. cameras) for tracking and surveillance
    - Telemetry facilities such as antennas, receivers, display instrumentation systems
    - Command uplink and optical tracking facilities
    - Premier digital photographic and video services including operation of still cameras, high speed and video systems for Range Safety support, surveillance, and post-launch analysis (e.g. failure analysis)
        - a Person responsible in the field is needed with Visual Line of Sight (VLOS) during experiments' execution
    - High bandwidth for supporting experiments with swarm of devices
- If a facility is dedicated to UAVs, then:
    - The altitude must be more than 50 meters and below of 150m
    - Must be away from populated areas
    - Must provide remote pilot with VLOS, which shall be located at not more than 200m
    - Must provide geofenced area with anti-collision systems

2.3.2.1.2 RAWFIE Technical Support

When the hard requirements listed before are fulfilled, and testbed facility joins the federation, then a contact point form the technical team of RAWFIE is assigned to the newcomers (testbed responsible/operators). Regular skype calls between the contact points and the new beneficiaries are established once-per-week for resolving questions and efficiently overview the testbed integration. RAWFIE team provides the testbed operators with a manual for using the web portal and a software package that contains all testbed software components (downloadable from the RAWFIE tickets and activities' tracking tool, based on Redmine, in the WP8 section). The Testbed Manager component provides a GUI for the configuration of the testbed and the insertion of the testbed vehicles (screenshots are given in deliverable D5.5, section 4.5.1). Testbed operators have access and control to the following aspects from the RAWFIE Portal:

- Define their preferable dates and times when experimenters can run experiments
- Accept/Reject a booked experiment in their testbeds
- Overview the experiments that will be conducted using their testbeds
- Visualize a running experiment and cancel if it necessary

All UxVs are operating in RAWFIE Virtual Private Network. RAWFIE provides all the necessary certificates in order to establish a VPN inside the infrastructure.

While the former idea within the project was to allow testbed owners to develop their own version of the testbed components (as it is currently written in the D4.8), subsequently the consortium decided that this will be forbidden, in order to increase compliance, safety and simplicity. Therefore testbeds' owners must use the software developed and provided by RAWFIE within their testbeds.

As long as the project is running, the contact point that will support testbeds representatives is the project coordinator UOA. Once the project is completed, the primary contact point will be established by the organisation that takes over the RAWFIE platform according to the federation policy established in Work Package 2 and deliverable D2.3 – Federation Policy.

2.3.2.1.3 Training

New testbed managers must participate in webinar organized by UoA about RAWFIE ethics requirements. Ethics requirements are detailed, and the strategies to mitigate the risk explained. The main focus of the webinar is the dual-use requirement, but the misuse requirements are also covered and instructions about how to deal with each of them are given.

2.3.2.1.4 Integrated testbeds

During the project lifecycle, six (6) testbeds were integrated in RAWFIE. These testbeds are operational and equipped with different types of devices.

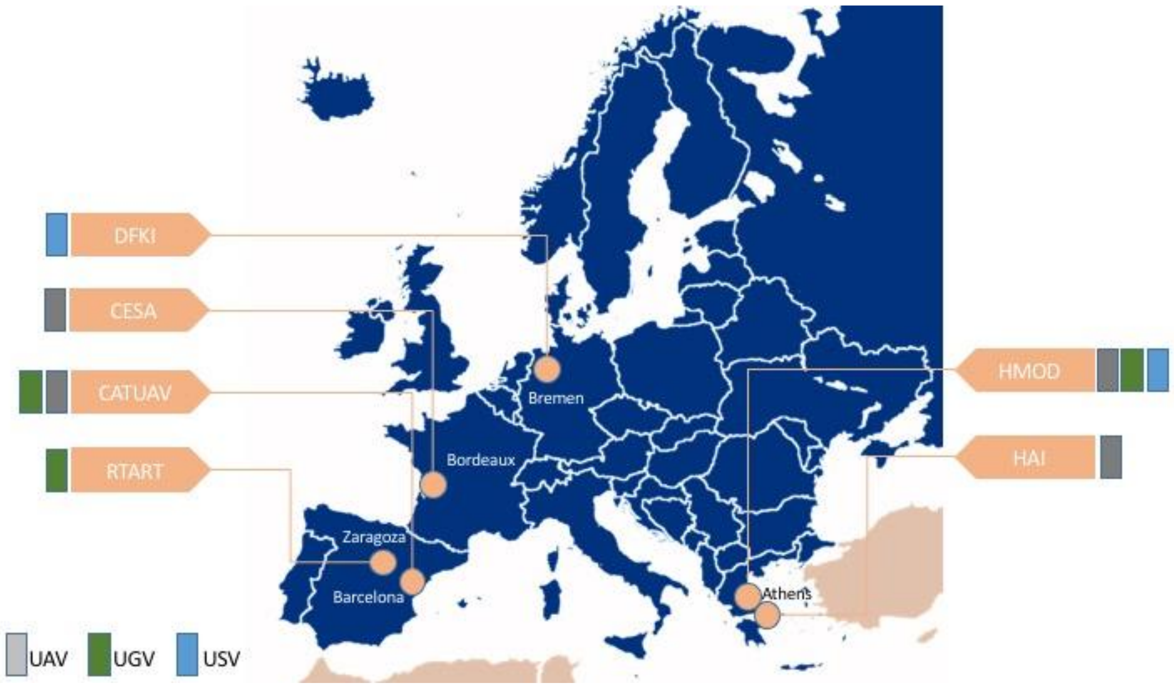


Figure 3: Testbeds Distribution

2.3.2.2 Integration of new vehicles

The basic idea behind the RAWFIE effort is the automated, remote operation of a large number of robotic devices for the purpose of assessing the performance of different technologies in the networking, sensing and mobile/autonomic application domains. RAWFIE considers three kinds of vehicles; UGVs, USVs and UAVs. The project aims to feature a significant number of UxV nodes in order to establish an extended test infrastructure for RAWFIE related experimentation purposes. All these items will be managed by a central controlling entity which will be programmed per case and fully overview/drive the operation of the respective mechanisms (e.g., auto-pilots, remote controlled ground vehicles). Internet connectivity will be extended to the mobile units to enable remote programming (over-the-air), control and data collection.

2.3.2.2.1 Requirements

RAWFIE promotes the experimentation under different technologies of devices (UxVs) that are equipped with different sensors, cameras and network interfaces. The following requirements have been defined on D3.3 to secure the interoperability with the RAWFIE platform, control units and testbeds:

- Compliance of UxVs to RAWFIE specification and interfaces
  - to be able to operate in a RAWFIE Tesbed, a RAWFIE UxV interacts with the other Testbed entities (proxy, controllers, other UxV's). As such the UxV shall conform to the RAWFIE global architecture and conceptual components defined in D4.8
- Each UxV shall have a unique Identification code
- Each UxV shall be able to operate autonomously
- Each UxV node shall ensure a minimum autonomy of 15-30 minutes (UXV-NOD-002/D3.3)
- Each UxV node shall ensure payloadshall be able to carry additional payload equipment of at least 0.5 to 1 kg in weight. (UXV-NOD-002 /D3.3)
- UxVs shall provide the capability of taking manual remote control of the UxVs(UXV-NET-001/D3.3)
- UxV network interface management:
  - each UxV shall be able to manage (detect/configure/control/use) the network interfaces installed, during the setup and execution of a mission (UXV-INT-014/D3.3)
- UxV communication interoperability with RAWFIE (incoming/outgoing):
  - each UxV shall be able to receive/send and decode/encode the incoming/outgoing messages from the testbed and deliver them to the relevant on-board component.
- Each UxV node shall tag location and timing capability to each sensor readings (SSL2)
- UxV location and sensor data shall be made available to the experimenter
- UxVs shall be capable to revert to a safe mode

2.3.2.2.2 RAWFIE Support

When the requirements are fulfilled and a new UxV joins the federation, then a contact point form the technical team of RAWFIE is assigned to the newcomer. Regular skype calls between the contact points and the new beneficiaries are established once-per-week for resolving questions and efficiently overview the integration of the UxV in the testbeds.

RAWFIE team provides access to the Gitlab that is created in the project. Examples for the UxV on-board software to interact with the message bus are shared with the third parties. When the integration with Message Bus is completed and tested, the device can be delivered to the testbed. For UAVs a flight insurance for the devices is needed (for ROC2 and ROC3 devices these insurances are provided by the coordinator). When the devices are delivered to the testbeds, a series of validation scenarios – the Operational Safety Scenarios described in D4.9 - are conducted in order to ensure the safe operational behaviour of the UxVs. For this purpose, one or more nodes of the same type and manufacturer will be always verified.

The main failsafe topics addressed by the emergency scenarios are listed below[4]:

- Communication link failsafe
- Battery/fuel failsafe
- GCS[5] failsafe (related to failure in Resource Controller, Testbed Manager, etc.)
- Geofencing issues (Testbed Boundary breaching)
- Localization issues
- Collision issues

For each of these main topics identified, specific Operational Safety Scenarios have been defined by the consortium and are described in Section 6.6 of deliverable D4.9. Those tests ensure that any new vehicle complies with the platform rules, that it is properly interfaced with its testbed through the Message Bus and that it understands the minimal set of commands related to its category. A checklist that summarises the whole new vehicle integration procedure with pointers to the necessary information is available on RAWFIE Wiki tool.

As long as the project is running, the contact point for support is the project coordinator UOA. Once the project is completed, the primary contact point will be established by the organisation that takes over the RAWFIE platform according to the federation policy established in Work Package 2 and deliverable F2.3 – Federation Policy.

2.3.2.2.3 Training

New UxVs manufacturers must participate in webinar organized by UoA about RAWFIE ethics requirements as described in 2.3.3.1.3.

2.3.2.2.4 Integrated UxVs

Below all the devices delivered to RAWFIE testbeds and allocated in different countries are listed.

UxV\Testbeds	Type	HMOD	HAI	Catuav	CESA	RTART	DFKI	Total
PLADYPOS	USV	3					7	10
FLEXUS	USV	10						10
NIRIIS	USV	3					7	10

[4] It must be noted that the failsafe topics addressed by the emergency scenarios are considered in the context of the RAWFIE system. Most UxVs (especially UAVs) provide inherent failsafe mechanisms related to most of these topics. These mechanisms should be regarded as an extra safety umbrella in case the RAWFIE specific ones fail

[5] GCS= Ground Control Station

VENAC	UAV	2	6	4				12
DOGMA	Fixed Wings	2	4	2	2			10
FIBLE	UGV	5		2		3		10
ITCROWD	UAV	4		4	4			12
Total		29	10	12	6	3	14	74

2.3.2.3 Experimenters

Experimenters of testbed and UxV resources can be categorized in three main groups based on the experiments' type:

- Experimentation of UxVs hardware components:
  - Integration of new hardware to the vehicles of an existing testbed is required by most of the experimenters. New hardware is represented by new sensors, alternative communication interfaces for networking-related experiments, supplementary computer, etc.
- Experimentation of UxVs software:
  - Experimenters need to test network algorithms based on different allocation of devices in space. This type of experiments handle UxVs as Access Points that are enhanced with mobility.
- Experimentation with dynamic re-routing of UxVs:
  - Experimenters need to monitor the camera or sensor feedback of UxVs in space and re-locate them dynamically based on events (like fire detection) or telemetry statistics.

Once experimenters have a clear picture on which category of experiments they want to run, they should get in contact with the RAWFIE federation to be assisted for the preparation of the experiment.

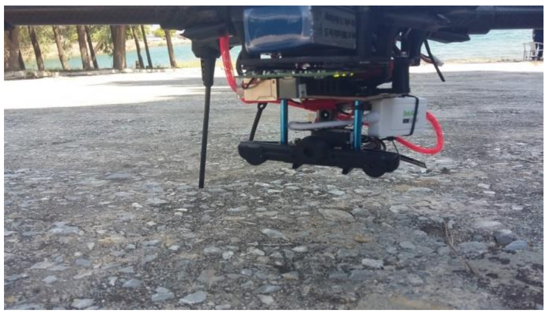
New experimenters shall clarify the experiments or the problem they would like to solve to the responsible contact point. Then together they would define

- In which Testbed they would like to run the experiments
- With what type of devices
- What will be the hardware mounted (if it is needed)

Afterwards, an account is created in order to access web portal and administration tools: like Redmine, Gitlab and Owncloud. Inside Owncloud, experimenters will find a folder which contains a tutorial about the use of the platform.

In case that experimenters need to consume or produce messages from/to the Message bus (for instance if they integrate their own hardware to RAWFIE UxV’s), specific guidelines are given together with two examples of adapters based on different technologies (Java, Python) which are available in gitlab. For safety reason all the adapters that are developed by experimenters are tested prior to a simulated testbed of UAVs or USVs.

For hardware integration such as extra sensors or communications means, specific dimensions and operating guidelines shall be submitted to the RAWFIE platform manager who will advise the best integration solution. For instance, the image below represents the integration of a snapdragon computer on a RAWFIE UAV.



Experimenters can book their resources in the permitted timetable of RAWFIE booking service and start write their experiments. Each experiment is documented in advance, and experimental resources booked, through an on-line system and cannot be conducted until the proposed experiment has been approved by the ‘Ethics Committee’. The launching day experimenter can either launch manually the devices or schedule the launching time beforehand. In case of UAVs, qualified pilots must be supplied by the testbed operators and/or the University of Athens.

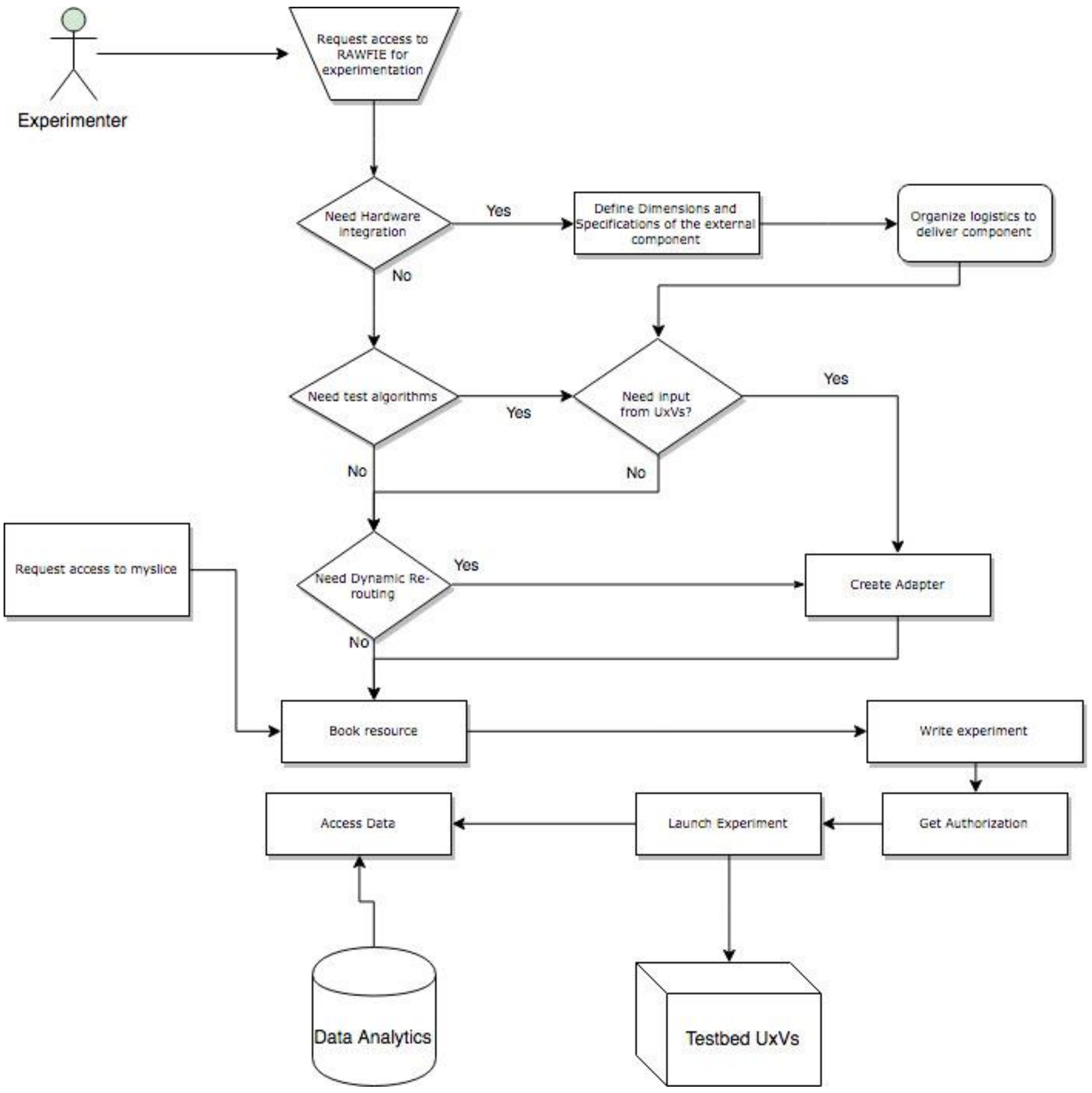


Figure 4: Flowchart of a new experiment

2.4 Integration environment

This section describes the environment used for the integration of the RAWFIE components and sub-systems and the subsequent testing. A high level overview is depicted in Figure 5. The integration environment includes the information, communication and computing infrastructure (servers, networks, etc.), the configuration (component settings, credentials, etc.) and data repositories, the testbeds used for testing and all other external services.

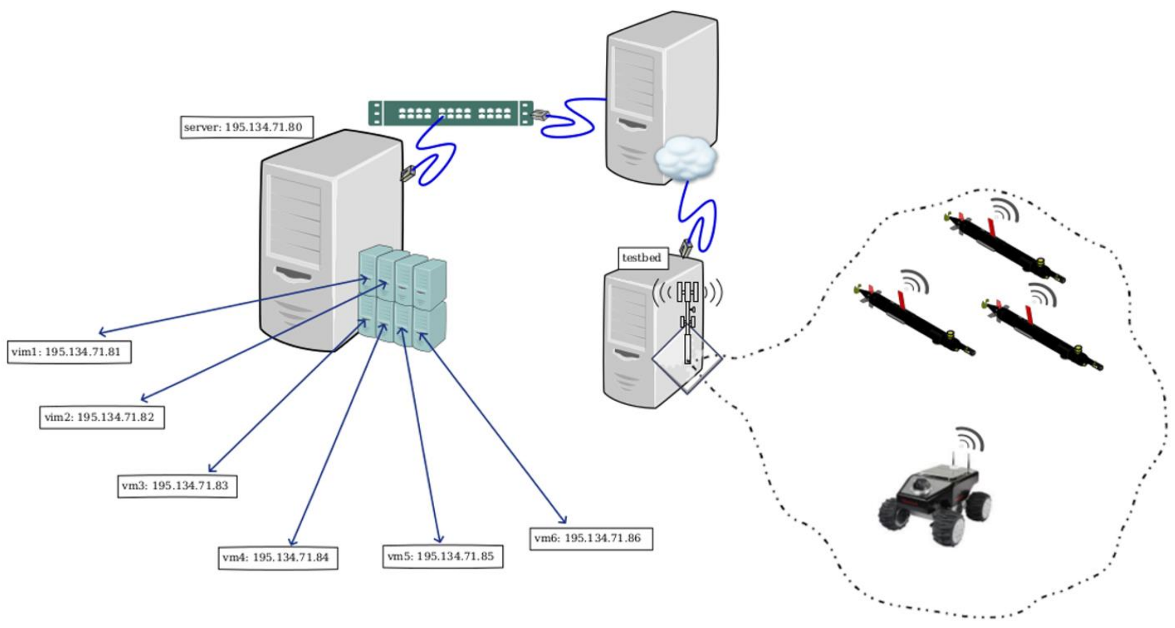


Figure 5: RAWFIE environment integration

2.4.1 Development Lifecycle of RAWFIE Tools and Services

A clone infrastructure of the production RAWFIE platform infrastructure described in D5.3, was created for development, integration and testing purposes, therefore for facilitating continuous integration and resolving of errors. This environment is illustrated in Figure 6Figure 6.

The messages from the online RAWFIE platform (production environment) are mirrored to the development environment in order to test all services with real data. The mirroring procedure is also used in the opposite direction, for software code updates and for upgrading services: when a service / software is stable enough it is moved to the online platform.

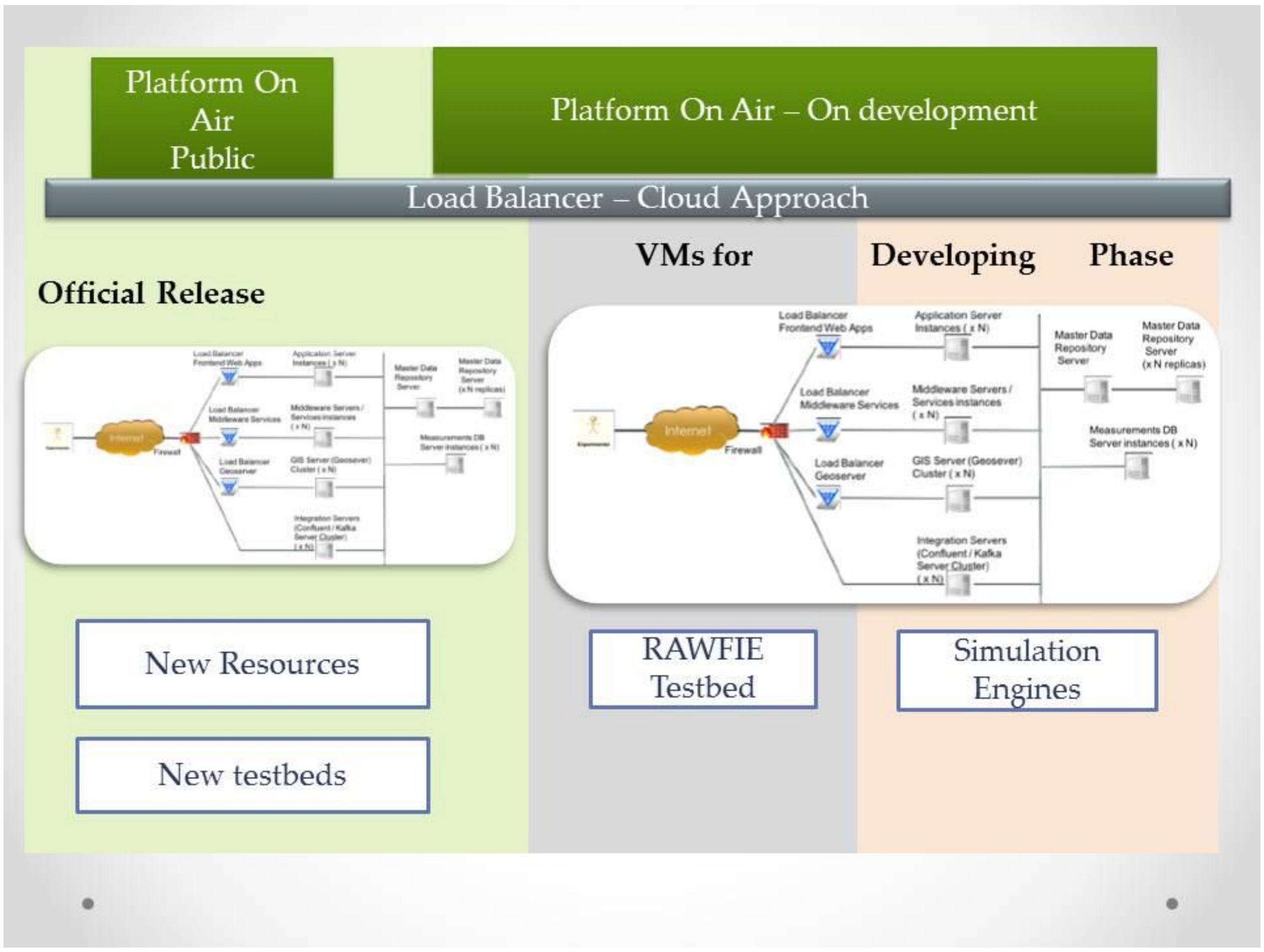


Figure 6: RAWFIE clones for the development infrastructure

According to the DoA, the first Milestone related to the development cycles was defined in M18 on which the 1st release of the platform was released. In order to outline a structured development process while maximizing the productivity and reducing possible bugs (that could be exposed to the experimenters), the RAWFIE consortium agreed in the creation of two identical environments: production and development. The production environment is the online platform that external users and experimenters can reach the RAWFIE functionalities via Internet. The development environment consists of servers and services used for updates in coding and upgrading the services without affecting the rest of the infrastructure.

2.4.2 Data repositories

The data model defined in D4.7 can be broken down into four major components:

1. Persistent Storage of Message BUS / Measurements DB: this will be done by Kafka Connect duplicating all messages on the BUS to HBASE (which is in turn backed by Hadoop).
2. Analysis Results DB: this database will contain the results for the data analysis tasks and is currently backed by a time series database called Whisper
3. Master Data DB: this will house traditional SQL type data and is implemented using PostgreSQL.

4. Users & Rights Repository: uses a LDAP repository, as LDAP is a de facto standard for user management. It stores all user related data (name, organisation, address, password) and group memberships (roles based access control). The selected implementation is OpenDJ

2.4.3 Tools & techniques for integration

RAWFIE uses a number of collaboration tools providing an integration friendly environment for development and deployment, such as Git, Docker and Redmine (see Figure 7).

In addition, Hadoop and HIVE are used as the connectors between the messages and the data storage of experimenters, which provides an efficient decoupling that is convenient for integration. An automatic data chunking is implemented in an experiment-specific (or experimenter-specific) directory on the HDFS storage. Such directory is created with the initiation of an experiment.

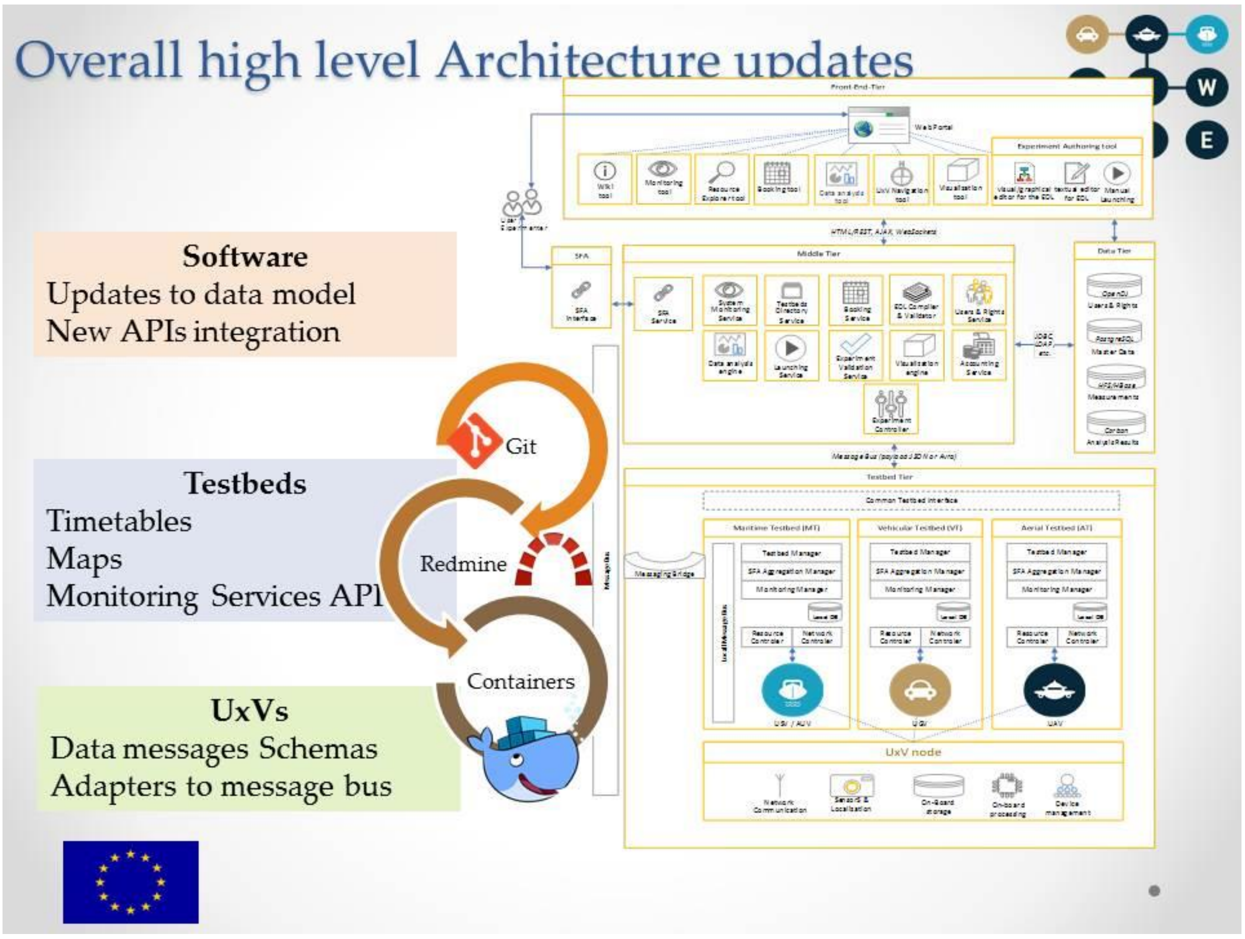


Figure 7: Tools for integration

Several tools are being used in order to facilitate continuous reporting and the integration of the software tools in a common environment. Redmine is used for issue-tracking tool. It contains information related to the project work packages and the relevant actions. A Git platform was installed with Gitlab environment for all partners to work concurrently by using branching. All software is uploaded so that partners can create branches for their specific development needs and features.

Another feature that is used for the integration is the creation of machine image boxes in order to provide to testbed operators “black boxes” with the RAWFIE required services pre-installed and pre-configured. RAWFIE components are installed in Vagrant image boxes, which are used for quick deployment of the RAWFIE system by the developers and testers. Docker is in use for the automation of installation for the simulators of USVs and UGVs.

2.4.4 Message Bus

The message bus is an essential integration tool. RAWFIE uses the Kafka message bus for interconnecting the components, for data exchange, ordering and persistency, for reliability and robustness.

The Kafka mirroring feature is used for creating the replica of an existing cluster, for example, for the replication of an active data centre into a passive data centre. Kafka provides a mirror maker tool for mirroring the source cluster into target cluster. This feature is used to allow for the replication of an exploitation environment to a site dedicated to development, test or maintenance.

The following diagram depicts the mirroring tool placement in architectural form:

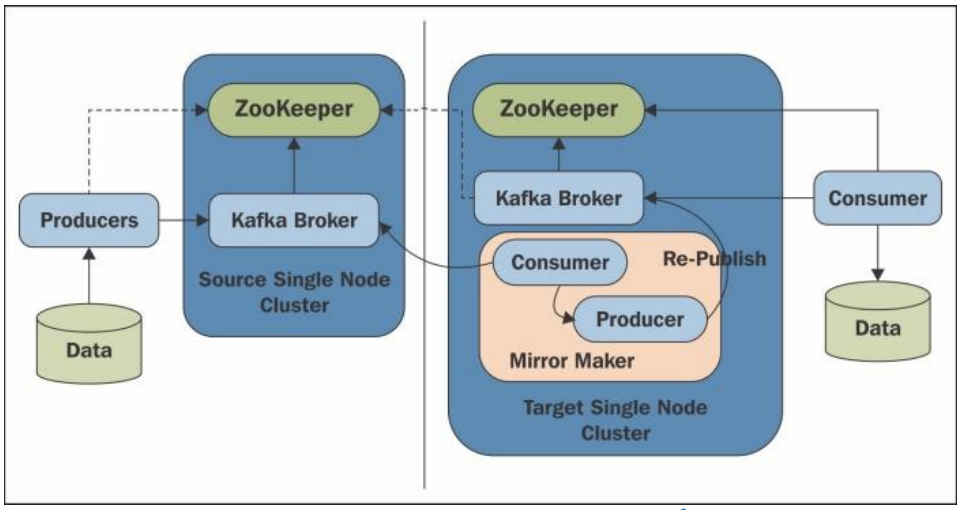


Figure 8: Mirroring architecture[6]

In contrast of replication processes, mirroring provides duplication of data across the testbeds. The advantages of mirroring are multiple like when a single connection is down, the possibility of longer clients connection/session times (depending on the location of the testbeds), and legislation (some data can be collected in a country while some other data should not).

6 https://www.packtpub.com/mapt/book/big_data_and_business_intelligence/9781782167938/4/ch04lvl1sec20/cluster-mirroring-in-kafka

2.4.5 Integration of new UxVs

WP4 deliverables provide technical guidelines for new UxVs integration in the platform. As specified in D4.8, UxV providers need to implement an “UxV Node” software module. This module is the software adaptor for RAWFIE, which will make the integrated UxV able to send measurements data, and to receive information and commands in standard format, mainly as JSON messages based on AVRO schemas. The RAWFIE “UxV Node” module also implements Apache Kafka Publishers and Consumers software, for the communication with other RAWFIE components.

2.4.6 Integration of new Testbeds

Besides providing the needed equipment for network connectivity, Testbeds owners need to deploy on premises the following RAWFIE software components:

- At least two local **Apache Kafka message bus servers**, for redundancy and high availability: these nodes realise the communication of the UxVs in the given Testbed, with other RAWFIE components
- **Testbed Manager**: provides the software interface to store UxVs related information to the Local DB, to the Master Data Repository through the Testbed Directory Service and to the Triple Store DB through the SFA Aggregate Manager (see D4.4, D4.5, D4.7, D4.8 for detailed information on the design and interactions of these components)
- **Triple Store DB and SFA Aggregate Manager**: the SFA AM provides, through a REST API, advertising functionalities based on semantic searches on the local Triple Store. The same REST API is used for editing or adding new resources, to store local resources (UxVs) information in the Triple Store DB
- **Resource Controller** (optional): provides resources controlling capabilities according to custom algorithms developed within the RAWFIE project
- **Monitoring Manager**: provides Testbed side connection to the System Monitoring services and the related Frontend tools.

These elements are distributed using Vagrant virtual machines. Several Vagrant[7] virtual machine image boxes provide testbed operators with an environment bundled with all the RAWFIE components and the required software for these components to function properly. These images include all the testbed services, such as the Testbed Manager, the Resource Controller, the Message Bus broker, etc.

The distribution of these boxes to our testbed operators has two main benefits. First, we save time from building from scratch every time the required software environment to perform tests. Secondly, the distribution of ready-to-go images ensures that there will be no problems to our testers, due to software incompatibilities. In addition, with every upcoming upgrade to the RAWFIE components everything will continue to work properly.

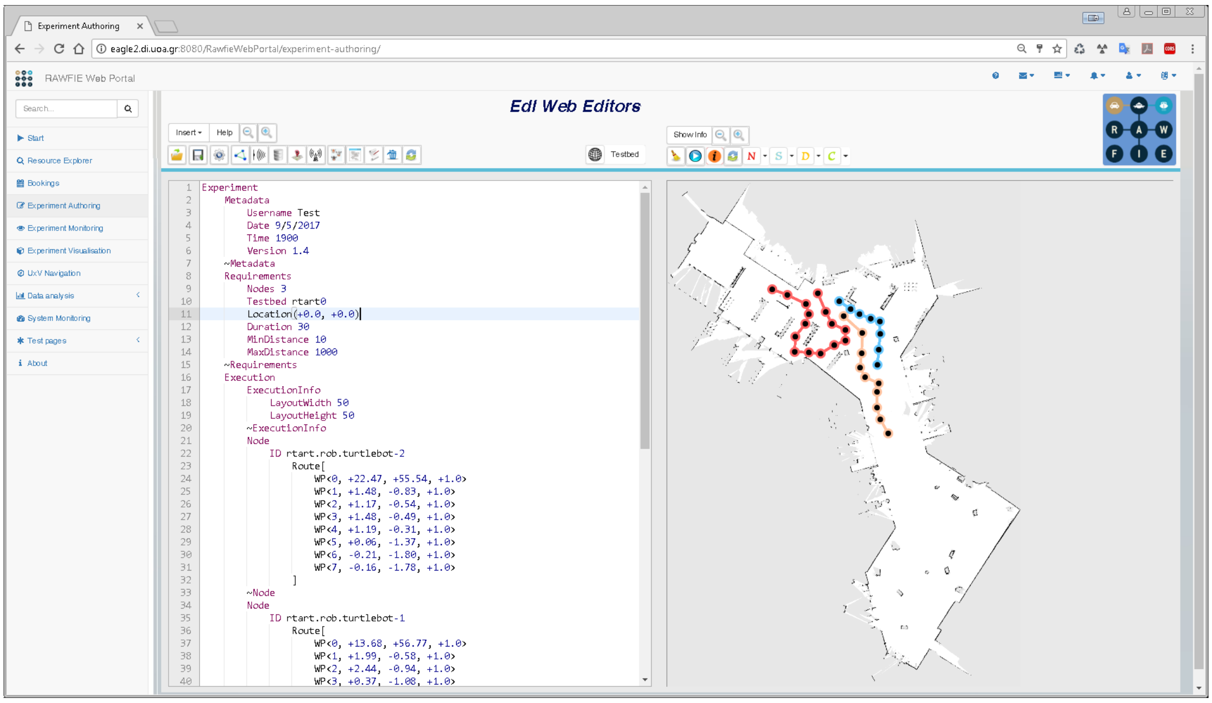
[7] https://www.vagrantup.com/

The process to integrate devices and testbeds in RAWFIE platform is straightforward:

1. Testbeds provide information registered in RAWFIE database like location, name of the testbed, polygon of area or indoor map (if the testbeds is indoor)
2. RAWFIE provides to testbed operator a VM for being installed in a local server
3. VPN certificates created for the testbed and VPN connection
4. Testbed operator registers via Testbed Manager the devices in the database
5. Trainings for the devices delivered in testbed
6. Testbed is up and running

Although the delivery of the devices to testbeds coming from 1st Open Call is ongoing, some testbeds have started the integration process to the RAWFIE platform.

The first testbed ready for the integration was an indoor testbed providing experiments for UGVs in several rooms. Starting from the kick off meeting in Athens for the Open Calls, 1 people from the University of Zaragoza provided an infrastructure for monitoring the possible area of experiments. The Wi-Fi coverage was established and tested to all the areas. The next thing was the installation of a local RAWFIE server. The credentials for the VPN network was sent to the testbed and a Virtual image of machine embedding of the required aforementioned services was sent to the testbed. The indoor maps were created by a lidar-embedded sensor on the devices and sent for their integration to RAWFIE geoserver in order to be used by the Experiment Authoring Tool and the Visualization tool (illustrated in Figure 9). The devices were made compatible with the Message bus by implementing a kafka consumer and producer, available in the VPN network. The integration was completed with a training session delivered by the manufacturer of the devices (UGVs) to the testbed owners.



```
Experiment
    Metadata
        Username Test
        Date 9/5/2017
        Time 1900
        Version 1.4
    ~Metadata
    Requirements
        Nodes 3
        Testbed rtart0
        Location(+0.0, +0.0)
        Duration 30
        MinDistance 10
        MaxDistance 1000
    ~Requirements
    Execution
        ExecutionInfo
            LayoutWidth 50
            LayoutHeight 50
        ~ExecutionInfo
        Node
            ID rtart.rob.turtlebot-2
                Route[
                    WP<0, +22.47, +55.54, +1.0>
                    WP<1, +1.48, -0.83, +1.0>
                    WP<2, +1.17, -0.54, +1.0>
                    WP<3, +1.48, -0.49, +1.0>
                    WP<4, +1.19, -0.31, +1.0>
                    WP<5, +0.06, -1.37, +1.0>
                    WP<6, -0.21, -1.80, +1.0>
                    WP<7, -0.16, -1.78, +1.0>
                ]
        ~Node
        Node
            ID rtart.rob.turtlebot-1
                Route[
                    WP<0, +13.68, +56.77, +1.0>
                    WP<1, +1.99, -0.58, +1.0>
                    WP<2, +2.44, -0.94, +1.0>
                    WP<3, +0.37, -1.08, +1.0>
```

Figure 9: Experimental area of the University of Zaragoza displayed in the Experiment Authoring Tool and the Visualization tool

2.5 Results of the Integration Test

This section provides an overview of the software interfaces between the various SW modules developed within RAWFIE. It includes front-end components as well as modules implemented at middle tier, testbed and UxV tiers. The table below provides additional information about the type of interfaces that exist between each pairs of components. The level of implementation/testing is depicted with appropriate colouring and represents the situation at the end of the 3rd development iteration.

In Table 1 each cell represents an interface that was tested. This cell is used by the two components at the cross lines: each client component, or caller of one or many services interfaces, is represented in the rows, while the called component or service interface/s is represented in the columns.

Table 1: interface interaction matrix

Row =[accesses]=> Column	Web Portal	Wiki	Resource Explorer Tool	Booking Tool	Experiment Authoring Tool	Experiment Monitoring Tool	System Monitoring Tool	Visualization Tool	Data Analysis Tool	EDL Compiler & Validator	Experiment Validation Service	Users & Rights Service	Booking Service	Launching Service	Experiment Controller	Data Analysis Engine	System Monitoring Service	Testbeds Directory Service	Accounting Service	Visualization Engine	Master Data Repository	Users & Rights Repository	Measurements Repository	Results Repository	Testbed Manager	Monitoring Manager	Network Controller	Resource Controller	Aggregate Manager (SFA)	UxV node	UxV Proximity	UxV - Network communicatio	UxV – Sensors & Localization	UxV – On board storage	UxV – On board processing	UxV – Device management	Schema Registry
Web Portal	■											R										L															
Wiki		■																				L															
Resource Explorer Tool			■	R														R																			
Booking Tool				■								R	R								O																
Experiment Authoring Tool					■					O	O			R							J																
Experiment Monitoring Tool			R		R	■								R			R				J																
System Monitoring Tool							■										R																				
Visualization Tool								■												O										M		M	M		M		
Data Analysis Tool									■							M,R								R													
EDL Compiler & Validator										■	O										J																
Experiment Validation Service											■										J																
Users & Rights Service												■									J	L															
Booking Service												R	■								O								R								
Launching Service												R		■	M-p						O							M-p									
Experiment Controller														M-c	■					M-p	O				M-p			M-p									
Data Analysis Engine									R							■							O	R,O													R
System Monitoring Service	R																■								M-c					M-c							
Testbeds Directory Service																		■			J																
Accounting Service																			■		J																
Visualization Engine															M-c					■	J									M-c							
Master Data Repository																					■																
Users & Rights Repository																						■															
Measurements Repository																							■														
Results Repository																								■													
Testbed Manager															M-c		M-p	R							■		M-c	M-c	R								
Monitoring Manager																	M-p									■				M-c							
Network Controller																									M-p		■	M				M-c					M
Resource Controller															M-c										M-p		M	■		M		M	M				
Aggregate Manager (SFA)																													■								
UxV node								M												M-p						M-p		M		■	I	I	I	I	I	I	M
UxV Proximity																														I	■						
UxV - Network communication								M																			M-p	M		I	I	■					
UxV – Sensors & Localization								M																				M		I			■				
UxV – On board storage																														I				■			
UxV – On board processing								M																						I					■		
UxV – Device management																														I						■	
Schema Registry																																					■

Type	Description
M-c	Message bus consumer (receives messages from the message bus)
M-p	Message bus producer (sends messages to the message bus)
REST or R	REST (via HTTP) web service
LDPA or L	LDPA
JDBC or J	JDBC
JPA	Java Persistence API
I	UxV internal: UxV OS dependent

Table 2: Interface types used in interface testing

Type	Description
M-c	Message bus consumer (receives messages from the message bus)
M-p	Message bus producer (sends messages to the message bus)
REST or R	REST (via HTTP) web service
SOAP or S	SOAP web service
LDPA or L	LDPA
JDBC or J	JDBC
JPA	Java Persistence API
I	UxV internal: UxV OS dependent

Note: for interface of type M-p, a related component is not included (or only "Message Bus" is mentioned). This is for example the case when the component acts as producer. The rationale behind this is that the producer of an Avro message just sends to the Bus agnostic of which will receive it. This message may be received by multiple consumers and this interaction will be depicted in the interface table of each receiver component including information for the exact producer. Therefore, there is no need to replicate this for the producer by including several similar rows.

2.5.1 Front-end integration

In the front-end tier, the integration activities included:

- Integration of *User and Rights Service* with the Web Portal as the main authorization mechanism for gaining access to the RAWFIE platform
- The following tools were integrated and became accessible via the web portal:
    - Wiki Tool
    - Resource Explorer Tool
    - Booking Tool
    - Experiment Authoring Tool
    - Experiment Monitoring Tool
    - System Monitor Tool
    - Visualisation Tool
    - Data Analysis Tool

Details on the interface testing activities performed for each front-end tool mentioned above are provided in the tables that follow.

Table 3: Test of the Web portal interfaces

Component: ***Web Portal***		Conducted by: **Fraunhofer**		Date: **May 2018**	Test Category: **Interface testing**
Preconditions		Users are entered in the User & Rights Repository Wiki Tool has some help pages			
Related Component		**Type**[8]	**Message or API Call**	**Status**	**Remarks/comments**
1	User & Rights Repository	LDAP	Lookup	Success	Lookup user with the given password from the login page worked
2	Wiki Tool	Other	HTTP open web page	Success	Open web page in the Wiki Tool containing help for the current page.

Table 4: Test of the Wiki Tool interfaces

Component: ***Wiki Tool***		Conducted by: **Fraunhofer**		Date: **May 2018**	Test Category: **Interface testing**
Preconditions		Users are entered in the User & Rights Repository			
Related Component		**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	User & Rights Repository	LDAP	Lookup	Success	Lookup user with the given password from the login page worked

[8] Type refers to how the component interacts/interfaces with related component. For example if the component produces a message intended to be received by the related component the type should be M-p (acts as producer) while if it consumes a message type should be M-c.

Table 5: Test of the Resource explorer interfaces

Component: ***Resource Explorer***		Conducted by: **Fraunhofer**		Date: **May 2018**	Test Category: **Interface testing**
Preconditions		Resources are entered in the Master Data repository			
Related Component		**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	Testbeds Directory Service	REST	searchResource	Success	Search resource by resource id passig a JSON in input
2			getAllResources	Success	Got all resources/UxVs
3			searchTestbed	Success	Search testbed by testbed id passing a JSON in input
4			getAllTestbeds	Success	Got all testbeds
5			getResources	Success	Got all resources/UxVs for a specific testbed id passing a JSON in input
6			testbed/identifier//{id}	Success	Testbed by testbed id
7			testbed/name/{name}	Success	Testbed by testbed name
8			testbeds?param1=value1 &param2=value2&param3=value3	Success	Testbeds by search parameters
9			resource/identifier/{id}	Success	Resource by resource id
10			resource/name/{name}	Success	Resource by resource name
11			resources?param1=value1 &param2=value2&param3=value3&param4=value4	Success	Resources by search parameters
12			testbeds/uav	Success	Testbeds supporting UAV
13			testbeds/usv	Success	Testbeds supporting UGV
14			testbeds/ugv	Success	Testbeds supporting USV
15			Testbeds/auv	Success	Testbeds supporting AUV
5	Booking Tool	HTTP	Redirect to booking page of testbed	Success	Booking Tool opens the booking page of the related testbed

Table 6: Test of the Booking Tool interfaces

Component: ***Booking Tool***		Conducted by: **HAI**		Date: **Feb 2018**	Test Category: **interface testing**
Preconditions		• User must be logged in • UxV resources must be present in a testbed and advertised to the platform (browsable by the resource explorer tool) • Booking Service must be up and running • User & Rights Service must be up and running			
	Related Component	**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	Booking Service	R	addReservation	Success	
2		R	editReservation	Success	
3		R	deleteReservation	Success	
4		R	getReservations	Success	
5		R	getReservation	Success	
6		R	checkForConflictingReservations	Success	
7		R	approveBooking	Success	
8		R	rejectBooking	Success	
9	User & Rights Service	R	checkLogin	Success	Used to ensure that user of tool is authorized
10		R	checkTestbedRoles	Success	Used during approveBooking/ rejectBooking
11	Master Data Repository	JPA/JDBC	JPQL and/or JPA queries	Success	used to retrieve reservation & resource information for display in calendar view

Table 7: Test of the Experiment Authoring Tool interfaces

Component: ***Experiment Authoring Tool***		Conducted by: **UoA**		Date: **Feb 2017**	Test Category: **Interface testing**
Preconditions		Users are entered in the RAWFIE Web Portal			
Related Component		**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	Launching service	REST	manualStart	Success	Launching tool is correctly informed about the ID of the experiment that will be executed
2		REST	schedule	Success	Schedule launch button correctly sends the necessary info in the launching tool
3	EDL Compiler & Validator	Other	-	Success	The compiler & validator is correctly adopted when needed
4	Experiment validation service	Other	HTTP requests	Success	Compilation and validation are smoothly executed in the authoring tool
5	Master Data Repository	JDBC	JDBC-SQL Queries	Success	Data are correctly retrieved

Table 8: Test of the Experiment Monitoring Tool interfaces

Component: ***Experiment Monitoring Tool***		Conducted by: **Fraunhofer**		Date: **May 2018**	Test Category: **Interface testing**
Preconditions		System Monitoring Service collected some data Experiment Status is up-to-date in database			
Related Component		**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	Master Data Repository	JDBC	SQL – select experiments of user	Success	
2		JDBC	SQL – select experiment data and status	Success	
3		JDBC	SQL – select UxVs data of experiment	Success	
4	System Monitoring Service	REST	getComponentServiceHealth	Success	Health status shown
5	Launching Service	REST	cancel	Success	Status set to canceled

Table 9: Test of the System Monitoring Tool interfaces

Component: System Monitoring Tool		Conducted by: **Fraunhofer**		Date: **May 2018**	Test Category: Interface testing
Preconditions		System Monitoring Service collected some data			
Related Component		**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	System Monitoring Service	REST	getComponentServiceHealths	Success	Got all health statuses

Table 10: Test of the Visualisation Tool interfaces

Component: ***Visualisation Tool***		Conducted by: **Aberon**		Date: **Feb 2017**	Test Category: **Interface testing**
Preconditions		• User must be logged in to the portal			
	Related Component	**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	Visualisation Engine	Web-socket	startExperiment	Success	Connect to the visualisation engine and retrieve all the information about an experiment and get data for the movement of the UxVs
2			stopExperiment	Success	Stop the visualisation of an experiment
3			getExperiments	Success	List all available experiment for the user
4			getExperimentDetails	Success	Get the details for an experiment that the user wants to visualise

Table 11: Test of the Data Analysis Tool interfaces

Component*: Data Analysis Tool*		Conducted by: **HESSO**		Date**: Feb 2017**	Test Category**: Interface testing**
Preconditions		• User must be logged in • Resources must be associated with a user • Resources must be associated with an experiment • Message Bus must be up and schema registry must be accessible • Results database must be accessible • Zeppelin & Spark must be operational			
	Related Component	**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	Results Database	REST	render()	Success	Graphite is able to be queried via REST and plots results
2	Data Analysis Engine	M-p	buildJob()	Success	Send the Analytics jobs to the Data Analysis Engine through the Kafka message bus
3		REST	Send the SPARK job directly from the Zeppelin UI	Success	Message sent to Spark Directly via REST interface. This is part of Zeppelin by default and works already.

Table 12: Test of the Accounting Tool interfaces

Component*: Resource Explorer*		Conducted by: **Fraunhofer**		Date**: May 2018**	Test Category**: Interface testing**
Preconditions		• User must be logged in with "billing manager" role • Users with subscribtions, resource usages, invoices are already registered			
Related Component		**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	Accounting Service	REST	getBalance(dn)	Success	Balance of user with the ID is returned
2			getCurrentSubscriptionType(dn)	Success	Type ID of the subscription of the user is returned
3			createAccount(account, subscription)	Success	User was created in the accounting service after his first billing action (book a resource)
4			getInvoices(dn)	Success	Returned all invoices of the given user
5			getUsages(dn)	Success	Returned all usage data of the given user
			setNextSubscriptionType(dn, subscription)	Success	Set the subscription of the given user beginning with the next billing period.
6			getNextSubscriptionType(dn)	Success	Returned of the next planned subscription given user
7			getAccounts()	Success	Returned all accounts

2.5.1.1 Missing components

The UxV Navigation Tool was removed from the platform and thus not implemented. In its place the Relocator was implemented coming from the need of dynamic navigation and not as a remote-control-navigation system.

2.5.2 Middle tier integration

In the middle-tier integration, activities included testing of interfaces of the following services (with front-end tools, between them and through the message bus):

- EDL Compiler and Validator
- Experiment Validation Service
- User & Rights Service
- Booking Service
- Launching Service
- Experiment Controller
- Data Analysis Engine
- System Monitoring Service
- Testbed Directory Service
- Visualisation Engine

Details on the interface testing activities performed for each component mentioned above are provided in the tables that follow.

Table 13: Test of the EDL Compiler and Validator interfaces

Component***: EDL Compiler and Validator***		Conducted by**: UoA**		Date**: Feb 2017**	Test Category**: Interface testing**
Preconditions		Users are entered in the RAWFIE Web Portal			
Related Component		**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	Experiment validation service	Other	HTTP requests	Success	Experiments are smoothly validated
2	Master data Repository	JDBC	JDBC-SQL Queries	Success	The data are correctly retrieved

Table 14: Test of the Experiment Validation Service interfaces

Component***: Experiment Validation Service***		Conducted by**: UoA**		Date**: Feb 2017**	Test Category**: interface testing**
Preconditions		Users have entered into the RAWFIE portal.			
	Related Component	**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	Master data Repository	JDBC	JDBC-SQL Queries	Success	Data are correctly retrieved

Table 15: Test of the User & Rights Service interfaces

Component***: Users & Rights Service***		Conducted by**: Fraunhofer**		Date**: May 2018**	Test Category**: Interface testing**
Preconditions					
Related Component		**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	User & Rights repository	LDAP	bind	Success	User credential validated
2		LDAP	search	Success	Entries (users, groups etc.) listed
3		LDAP	create	Success	Entries (users, groups etc.) added
4		LDAP	modify	Success	Entries (users, groups etc.) edited

5	Master Data Repository	JDBC	SQL select testbed roles	Success	Read roles for testbeds
		JDBC	SQL edit testbed roles	Success	Edit roles for testbeds

Table 16: Test of the Booking Service interfaces

Component: ***Booking Service***		Conducted by: **HAI**		Date**: February 2017**	Test Category**: interface testing**
Preconditions		• User must be logged in • UxV resource info must be present in a Master Data Repository • User & Rights Service must be up and running • SFA Aggregate Manager must be deployed in a testbed and running			
	Related Component	**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	Master Data Repository	JPA/ JDBC	Database call (insert)	Success	
2		JPA/ JDBC	Database call (update)	Success	
3		JPA/ JDBC	Database call (delete)	Success	
4	User & Rights Service	R	checkLogin	Success	Used to ensure that user of service is authorized
5	Aggregate Manager[9] (SFA)	R	samant/allocate	Success	used to create a new reservation to the SFA
6		R	samant/delete	Success	used to delete a lease (reservation if for some reason it fails in the RAWFIE side
7		R	admin/getInfo	Success	Used to retrieve information related to all active leases (reservations) in SFA. Needed for synchronization of RAWFIE & SFA Triple Store Databases
8		R	Admin/change_state	Success	Used to change state of lease (required during approve or reject booking action)

[9] Aggregate Manager used in RAWFIE is an adapted version of the SFA Aggregate Manager implemented in the context of the SAMANT ROC1 subproject. The aggregate manager supports all SFA specific functionality but provides for the additional reservation status that are needed to support RAWFIE 2-phase Booking process

Table 17: Test of the Launching service interfaces

Component***: Launching Service***		Conducted by: **HAI**		Date**: Feb 2018**	Test Category**: interface testing**
Preconditions		• User must be logged in • An experiment must be present for a user • Resources must be associated with a user • Resources must be associated with an experiment • Message Bus must be up and configured with appropriate topics (ExperimentLaunchRequest topic, ExperimentCancelRequest topic) • Experiment Controller must be up and running			
	Related Component	**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	Experiment Controller	M-p	ExperimentLaunchRequest	Success	Message was sent successfully to Message Bus and consumed by Experiment Controller
2	Resource Controller	M-p	ExperimentCancelRequest	Success	Message was sent successfully to Message Bus
3	Master Data Repository	JPA/ JDBC	Database Interaction	Success	Connection to database succeeded Retrieval/update/insert of information succeeded
4	User & Rights Service	R	checkLogin	Success	Used to ensure that user of service is authorized

Table 18: Test of the Experiment Controller interfaces

Component: Experiment Controller		Conducted by: **CERTH**		Date: **Feb 2017**	Test Category: interface testing
Preconditions		• Message Bus must be up and configured with appropriate topics • Connection to the RAWFIE database is required • The related Resource Controller must be up and running			
	Related Component	**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	Launching Service	M-c	ExperimentLaunchRequest	Success	Message was successfully consumed by Experiment Controller
3	Master Data Repository	JDBC	Database Interaction	Success	Retrieval of the experiment Script succeeded
4		JDBC	Database Interaction	Success	Retrieval of the resources partitions ids succeeded
5		JDBC	Database Interaction	Success	Retrieval of the testbed coordination system succeeded
6		JDBC	Database Interaction	Success	Insertion/Update inside experimentlog/experiment_execution/ experiment tables succeeded
7	Resource Controller	M-p	ExperimentStartRequest	Success	Message was sent successfully to Message Bus and consumed by Resource Controller
8		M-c	ExperimentStatusMsg	Success	Message was consumed by Experiment Controller
9	Testbed Manager	M-p	ExperimentStartRequest	Success	Message was sent successfully to Message Bus and consumed by Testbed Manager
10	Visualization Engine	M-p	ExperimentStartRequest	Success	Message was sent successfully to Message Bus and consumed by Visualization Engine

Table 19: Test of the Data Analysis Engine interfaces

Component: Data Analysis Engine	Conducted by: **HESSO**	Date: **Feb 2017**	Test Category: Interface testing		
Preconditions	• User must be logged in • Resources must be associated with a user • Resources must be associated with an experiment • Message Bus must be up and schema registry must be accessible • Results database must be accessible. • Spark must be operational • Landoop Schema browser must be operational				
	Related Component	**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	Schema Registry + Schema Browser	REST	/subjects	Success	Successfully iterate over all schemas via the augmented Landoop schema browser. Selection of features can also be done here.
2	Data Analysis Tool	REST	/api/notebook	Success	Data Analysis tool utilizes Zeppelin REST api to POST data
3	Results Database	REST / Sockets	graphite.send()	Success	A simple socket based connection from Spark sends online results to the graphite instance
4	Measurements Database	M-c	hbase.read()	Not Tested	Awaiting hadoop / hbase deployment

Table 20: Test of System Monitoring Service interfaces

Component: ***System Monitoring Service***		Conducted by: **Fraunhofer**	Date: **May 2018**	Test Category: **Interface testing**	
Preconditions					
Related Component		**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	Servers (Computer)	O	various	Success	Servers health status collected
2	Testbed Manager	M-c	TestbedHealthStatus	Success	Testbed send their health status to the message bus
3		M-c	UvVHealthStatus	Success	UxV health statuses send to the message bus

Table 21: Test of the Testbed Directory Service interfaces

Component: ***Testbed Directory Service***	Conducted by: **IES**		Date: **Feb 2016, April 2017, June2018**	Test Category: **interface testing**	
Preconditions	Testbeds and Resources tables, as well as all related tables with linked information about testbeds and resources, are present in the Master Data Repository (PostgreSQL DBMS)				
Related Component		**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	Master Data Repository (PostgreSQL database)	JPA - JDBC Interaction	insertTestbed	Success	Operation performed by a RepositoryHandler class, to support the **createTestbed()** REST API
2			updateTestbed	Success	Operation performed by a RepositoryHandler class, to support the **editTestbed()** REST API
3			deleteTestbed	Success	Operation performed by a RepositoryHandler class, to support the **deleteTestbed()** REST API
4			insertResource	Success	Operation performed by a RepositoryHandler class, to support the **createResource()** REST API
5			updateResource	Success	Operation performed by a RepositoryHandler class, to support the **editResource()** REST API
6			deleteResource	Success	Operation performed by a RepositoryHandler class, to support the **deleteResource()** REST API
7			fetchTestbed	Success	Operation performed by a RepositoryQuery class, to support the **searchTestbed()** REST API (get details about a specific testbed)
8			fetchTestbeds	Success	Operation performed by a RepositoryQuery class, to support the **getTestbeds()** REST API (get details about the specified testbeds)
9			fetchResource	Success	Operation performed by a RepositoryQuery class, to support the **searchResource()** REST API (get details of a specific resource from a specific testbed)
10			fetchResourcesTestbed	Success	Operation performed by a RepositoryQuery class, to support the **getResources()** REST API (to get details of all resources from a specific testbed)
11			fetchResourcesAvailable	Success	Operation performed by a RepositoryQuery class, to support the **getAvailableResources()** REST API (get details of all resources which are AVAILABLE for booking tests from a specific testbed)
12			fetchTestbedById	Success	Operation performed by a RepositoryQuery class, to support the testbed search by id
13			fetchTestbedByName	Success	Operation performed by a RepositoryQuery class, to support the testbed search by name
14			fetchTestbedsByUAV	Success	Operation performed by a RepositoryQuery class, to support the testbed search by UAV support
15			fetchTestbedsByUGV	Success	Operation performed by a RepositoryQuery class, to support the testbed search by UGV support
16			fetchTestbedsByUSV	Success	Operation performed by a RepositoryQuery class, to support the testbed search by USV support
17			fetchTestbedsByAUV	Success	Operation performed by a RepositoryQuery class, to support the testbed search by AUV support
18			fetchTestbedsByParameters	Success	Operation performed by a RepositoryQuery class, to support the testbeds search by a combination of search criteria
19			fetchResourceById	Success	Operation performed by a RepositoryQuery class, to support the resource search by id
20			fetchResourceByName	Success	Operation performed by a RepositoryQuery class, to support the resource search by name
21			fetchResourcesByParameters	Success	Operation performed by a RepositoryQuery class, to support the resources search by a combination of search criteria

22			insertArea	Success	Operation performed by a RepositoryHandler class, to support the **createArea()** REST API
23			updateArea	Success	Operation performed by a RepositoryHandler class, to support the **editArea()** REST API
24			deleteArea	Success	Operation performed by a RepositoryHandler class, to support the **deleteArea()** REST API
25			insertSensor	Success	Operation performed by a RepositoryHandler class, to support the **createSensor()** REST API
26			updateSensor	Success	Operation performed by a RepositoryHandler class, to support the **editSensor()** REST API
27			deleteSensor	Success	Operation performed by a RepositoryHandler class, to support the **deleteSensor()** REST API
28			insertNetInterface	Success	Operation performed by a RepositoryHandler class, to support the **createNetInterface()** REST API
29			updateNetInterface	Success	Operation performed by a RepositoryHandler class, to support the **editNetInterface()** REST API
30			deleteNetInterface	Success	Operation performed by a RepositoryHandler class, to support the **deleteNetInterface()** REST API
31			associateResourceTestbed	Success	Operation performed by a RepositoryHandler class, to support the **associateResourceTestbed()** REST API
32			associateNetIfResource	Success	Operation performed by a RepositoryHandler class, to support the **associateNetInterface()** REST API

Table 22: Test of the Visualisation Engine interfaces

Component**: *Visualisation Engine***		Conducted by: **Aberon**		Date**: March 2017**	Test Category**: interface testing**
Preconditions		• User must be logged in to the portal • Measurements and Results repository should be available • Kafka should be available with the necessary topics			
Related Component		**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	Master Data Repository	JDBC	GetExperimentDetails	Success	Get Experiment Status
2	Resource Controller	M-c	getGoTo	Success	Get the Goto Commands
3	Experiment Controller	M-c	ExperimentStartRequest	Success	Get the ExperimentStartRequest from the Message bus
4	UxV Node	M-c	getUxVLocation	Success	Get the location of an UxV
5		M-c	getUxVSensorData	Success	Get the sensor data from the UxVs. Not all sensor data is implemented yet.

2.5.3 Testbed & UxV integration

At the testbed level integration, activities included testing of interfaces of the following components (between them and through the message bus with UxVs or middle-tier components):

- The Testbed Manager
- The Monitoring Manager
- The Resource Controller
- UxV node
- Network Controller
- Proximity Component
- SFA Aggregation Manager (passive component, not tested)

Details on the interface testing activities performed for each component mentioned above are provided in the tables that follow.

Table 23: Test of the Tesbed Manager interfaces

Component: ***Testbed Manager***		Conducted by: **HAI**		Date: **May 2018**	Test Category: **interface testing**
Preconditions		• Confluent platform properly configured, up and running • Related components must be up and running			
Related Component		**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	System Monitoring Service	M-p	TestbedHealthStatus	Success	System Monitoring properly consumes the message that describes the current health of the machine running the Testbed Manager
2	Resource Controller	M-c	ExperimentStatusMsg	Success	Testbed Manager properly consumes the message that described the status of an experiment from Resource Controller
3		M-p	ExperimentCancelRequest	Succes	Testbed Manager properly cancels an experiment in case of emergency situations
4	Experiment Controller	M-c	ExperimentStartRequest	Success	Testbed Manager properly consumes the message that describes the start of an experiment from Experiment Controller
5	Resource Controller	M-c	TestbedServicesHealthStatus	Success	Testbed Manager successfully consumes and presents the messages about the health status of Resource Controller
6	Network Controller	M-c	TestbedServicesHealthStatus	Success	Testbed Manager successfully consumes and presents the messages about the health status of Network Controller
7	Aggregate Manager – SFA	R	/admin/create	Success	New resources entered by Testbed Manager are properly propagated in SFA Triple Store database
8	Aggregate Manager – SFA	R	/admin/update	Success	Modifications in existing resources from Testbed Manager are properly propagated in SFA Triple Store database
9	Aggregate Manager - SFA	R	/admin/delete	Success	Removal of existing resources from Testbed Manager are properly propagated in SFA Triple Store database

Table 24: Test of the Monitoring Manager interfaces

Component*: Monitoring Manager*		Conducted by: **HAI**		Date**: May 2018**	Test Category**: interface testing**
Preconditions		• Confluent platform properly configured, up and running • Reliable Internet connection with UxVs			
Related Component		**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	UxVNode	M-c	FuelUsage	Success	Real data from UxV devices
2		M-c	CpuUsage	Success	Real data from UxV devices
3		M-c	StorageUsage	Success	Real data from UxV devices
4		M-c	Location	Success	Monitoring Manager successfully receives Location messages from UxV devices
5		M-c	Attitude	Success	Monitoring Manager successfully receives Attitude messages from UxV devices
6	System Monitoring Service	M-p	UxVHealthStatus	Success	System Monitoring properly consumes messages about current UxV health status

Table 25: Test of the Resource Controller interfaces

Component*: **Resource Controller***		Conducted by: **CERTH**		Date**: Feb 2017**	Test Category**: interface testing**
Preconditions		• Confluent platform properly configured, up and running • Experiment Controller must be up and running • Related UxV Nodes must be up and running			
Related Component		**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	UxV Node	~~M-p~~	~~WriteHealthStatus~~	~~Not tested~~	~~Send and receive real-time information to resources~~
2		M-p	WriteUxVCommands	Success	Send and receive real-time information to resources
3		M-p	WriteExperimentStatus	Success	Send real-time kafka messages regarding the status of the experiment
4		M-c	ReadUxVStatus	Success	Resource Controller reads UAVs statuses so as to successfully take-off all the aerial vehicles before the experiment initiation.
5		M-c	Location	Success	Resource Controller is able to read the actual position of the vehicles
6	Experiment Controller	M-c	ExperimentStartRequest	Success	Resource Controller successfully receives and parses the experiment to be executed
7		M-p	ExperimentStatusMsg	Success	Message was sent successfully to Message Bus
8	Launching Service	M-c	ExperimentCancelRequest	Success	Resource Controller successfully receives and executes cancel requests
9	Testbed Manager	M-c	ExperimentCancelRequest	Success	Resource Controller successfully receives and executes cancel requests
		M-p	ExperimentStatusMsg	Success	Resource Controller successfully publishes status messages regarding any change in the progress of the received experiment

Table 26: Test of the UxV Node interfaces

Component:***UxV Node***	Conducted by: **Robotnik, MST**	Date**: Feb 2017**	Test Category**: interface testing**		
Preconditions	• A server running the Confluent platform • UxV manufacturer's (e.g. Robotnik) specific preconditions: ○ The necessary topics should be already registered ○ A server running the Confluent platform should be available with the necessary topics ○ Input from the resource controller ○ Reliable Internet connection				
	Related Component	**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	Resource Controller	M-c	Goto	Success	GPS coordinates accuracy and threshold for next waypoint needs to be configured
2			KeepStation	Success	Tested with success by MST; Ground vehicles are accepting this command as no waypoint commanded
3			Abort	Success	Tested with success
4			Location	Success	Without GPS specifying an origin of coordinates is needed. For indoor scenarios Cartesian coordinates are given with standard goto message
5	Visualization Tool	M-p	Location message	Success	Visualization indoors is now using specific images created with mapping tools normally using 2D scans
6	Visualization Engine	M-p	Location message	Success	Get the location of an UxV
7		M-p	SensorReadingScalar	Success	Get the sensor data from the UxVs. Not all sensor data is implemented yet.
8		M-p	UxVStatus	Not tested	
9	Data Analytics	M-p	SensorReadingScalar	Success	Tested Salinity, Conductivity, and SoundSpeed with water vehicles. Temperature measurements from both water and ground vehicles
10			Current	Success	Tested with success by MST
11			Voltage	Success	Tested with success by MST
12			StorageUsage	Success	Tested with success by MST
13			FuelUsage	Success	Tested with success by MST
14			CpuUsage	Success	Tested with success by MST
15			SensorInfo	Success	Tested with success by MST
16	Monitoring manager	M-p	FuelUsage	Success	Real data from the devices
17			CpuUsage	Success	Real data from the devices
18			StorageUsage	Success	Real data from the devices
19	Schema Registry	M	CachedSchemaRegistryClient	Success	Get the schema registry

Table 27: Test of the Network Controller Interfaces

Component*:* ***Monitoring Manager***		Conducted by**: CSEM**		Date**: July 2018**	Test Category**: interface testing**
Preconditions		• Confluent platform properly configured, up and running • Testbed components running • Reliable Internet connection with UxVs			
Related Component		**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	Network Controller	M-c	ExperimentStartRequest	Success	
2		M-c	ExperimentStatusMsg	Success	
3		M-p	GlobalNetwPerf	Success	
4		M-c	Location	Success	
5		M-p	NetwPerfUxV	Success	
6		M-c	NetwReportingPeriod	Success	
7		M-p	TestbedServicesHealthStatus	Success	

Remark: command *NetwSelectIf* removed. Network interface selection is done either internally within the UxVs or decided from the information available in *GlobalNetwPerf* messages.

Table 28: Test of the Proximity Component interfaces

Component*:* ***Monitoring Manager***		Conducted by**: MST**		Date**: November 2017**	Test Category**: interface testing**
Preconditions		• Proximity component linked to a UxV serial port interface and powered on • UxV connected to the testbed			
Related Component		**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	Proximity Component	M-c	ProxyConnectData	Success	Message used for test purpose only
2		UART	HCI interface	Success	UART/HCI interface between the proximity component and the UxV

SFA aggregation manager (untested)

The SFA aggregation manager is a passive component that does not call any external module. Rather, it is called via REST API by the Booking Service and/or the Testbed Manager. Therefore, in what concerns interface testing the SFA aggregation manager is viewed as a black box that is called by the 2 aforementioned RAWFIE components. Both of them already provide information on the possible interactions in their interface test tables.

2.6 Verification scenarios results

2.6.1 Frontend Tier

The verification of the Front-end tier mainly consists testing the Web Portal GUI elements.

2.6.1.1 *Web Portal*

Table 29: Verification test of the Web Portal - Login/ Logout

Test ID: **WP01**	Conducted by: **Fraunhofer**		Date: **May 2018**	Test Category: **Verification Tests (front end tier)**
Hardware Configuration	See section 2.4			
Software Configuration	See section 2.4			
Test Name:	***Web Portal - Login/ Logout***			
Preconditions	• User entered in the User & Rights repository			
Related Requirements	PT-WEB-P-001, PT-WEB-P-002			
Tools Used	• Browser			
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	user opens RAWFIE any web page	redirect to login page, login form displayed	Success	
2	user enters invalid credentials and submits the form	error message displayed	Success	
3	user enters valid credentials and submits the form	redirect to start page	Success	
4	user press the logout button	redirect to login page, login form displayed, logout message displayed	Success	

Table 30: Verification test of the Web Portal – Language selection

Test ID: **WP02**	Conducted by: **Fraunhofer**		Date: **May 2018**	Test Category: **Verification Tests (front end tier)**
Hardware Configuration	See section 2.4			
Software Configuration	See section 2.4			
Test Name:	***Web Portal – Language selection***			
Preconditions	• Translation available			
Related Requirements	PT-WEB-P-001			
Tools Used	• Browser			
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	user opens RAWFIE any web page	web page with language selection displayed,	Success	
2	user changes the language	web page displayed in the selected language	Success	

Table 31: Verification test of the Web Portal – User management

Test ID: **WP03**	Conducted by: **Fraunhofer**	Date: **May 2018**	Test Category: **Verification Tests (front end tier)**
Hardware Configuration	See section 2.4		
Software Configuration	See section 2.4		
Test Name:	***Web Portal – User management***		
Preconditions	• Admin login available • No pending registration request		
Related Requirements	PT-WEB-P-002		
Tools Used	Browser		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Browser 1: login as administrator and open user management page	management page displayed	Success	
2	Browser 1: Navigate to registration requests page	No registration request displayed	Success	Registration request where integrated into the user list page. No separate page.
3	Browser 2: Open register form, fill in form (login credentials, personal data, etc.) and submit	Registration request stored and confirmation shown to the user.	Success	
4	Browser 2:Try to login with the submitted login credentials	Login failed. Display message that user is looked	Success	
5	Browser 1: Reload registration requests page	The new registration request is show	Success	
6	Browser 1: Accept the new user	The new user is now unlooked	Success	
7	Browser 2: Try to login with the submitted login credentials	Login successful.	Success	
8	Browser 1: Navigate to the user list and delete the new user	User deleted	Success	
9	Browser 2: Logout and try to login with the submitted login credentials	Login failed. Show invalid credentials messages	Success	

2.6.1.2 *Wiki Tool*

Table 32: Verification test of the Wiki Tool – Component Help

Test ID: **WT01**	Conducted by: **Fraunhofer**	Date: **May 2018**	Test Category: **Verification Tests (front end tier)**
Hardware Configuration			
Software Configuration			
Test Name:	***Wiki Tool – Component help***		
Preconditions	• Help pages added to the Wiki		
Related Requirements	PT-WIK-001, PT-WIK-003		
Tools Used	Browser		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Login to the Web Portal and open Resource Explorer	Resource Explorer page displayed	Success	
2	Click on the Help icon	Wiki Tool opened with the article about Resource Explorer	Success	
3	Change display language in the Wiki	Wiki article displayed in another language	Success	
4	Repeat step 2 of other pages (like Visualization Tool, Booking tool, etc.)	Wiki Tool opened with the article about other tools	Success	

Table 33: Verification test of the Wiki Tool – Editing

Test ID: **WT02**	Conducted by: **Fraunhofer**	Date: **May 2018**	Test Category: **Verification Tests (front end tier)**
Hardware Configuration	See section 2.4		
Software Configuration	See section 2.4		
Test Name:	***Wiki Tool – Editing***		
Preconditions	• User for Wiki management defined		
Related Requirements	PT-WIK-001, PT-WIK-002, PT-WIK-004		
Tools Used	Browser		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Login to the Web Portal as normal experimenter and open a page in the Wiki Tool	Wiki page displayed	Success	
2	Try to edit the page	Editing not possible due to missing rights	Success	
3	Login as administrator and assign the Wiki manager right to the user	The user has now the Wiki manager right	Success	
4	Login as the first user and open a page in the Wiki Tool	Wiki page displayed	Success	
5	Try to edit the page	Editing allowed and changes are saved	Success	

2.6.1.3 *Resource Explorer Tool*

Table 34: Verification test of the Browse testbeds and UxVs and start booking

Test ID**: RET01**	Conducted by**: Fraunhofer**	Date**: May 2018**	Test Category**: Verification Tests (front end tier)**
Hardware Configuration			
Software Configuration			
Test Name:	***Browse testbeds and UxVs and start booking***		
Preconditions	• connection to the Testbeds Directory Service OK • data about testbeds and UxVs available		
Related Requirements	PT-REE-T-001, PT-REE-T-003, PT-REE-T-004		
Tools Used	Browser		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	user opens Resource Explorer Tool in the Web Portal	Resource Explorer Tool displays a view with all available testbeds	Success	
2	user set some filter parameters too find a testbed fitting to its needs	Resource Explorer Tool displays only the testbeds fitting to the filter	Success	
3	user selects a testbed	Resource Explorer Tool displays all testbed details and a list of available UxVs	Success	
4	user selects a UxV	Resource Explorer Tool displays all UxVs details	Success	
5	user starts booking	Booking Tool opened with the selected resources	Success	It was agreed to open the testbed for booking. Not the UxV.

2.6.1.4 *Booking Tool*

Table 35: Verification test of the Booking Tool Calendar View and its display options

Test ID: **BT01**	Conducted by: **HAI**	Date**: June 2018**	Test Category**: Verification Tests (web tier)**
Hardware Configuration	-		
Software Configuration	-		
Test Name:	***Booking Tool Calendar View and display options***		
Preconditions	• connection to the Booking Service ok • user has logged in the web portal • reservations of different status exist in the Master DB		
Related Requirements	PT-BOO-T-001, PT-BOO-T-003, PT-BOO-T-006, PT-BOO-T-010 PT-BOO-T-015, PT-BOO-T-016, PT-BOO-S-008		
Tools Used			

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Click of Bookings menu item	Navigation to Booking Tool (Calendar View)	Success	
		Calendar view displays by default the present week with all defined bookings	Success	
2	Switch Calendar display to display week, month, day interval via the appropriate options	Calendar view changes to present the selected interval with all defined bookings	Success	
3	Navigate back and forth in time via the provided navigation buttons (for every selection made in step 2)	Calendar view changes to previous or future date time intervals and displays even past reservations	Success	
4	Verify by inspection of existing reservations that only reservations of certain status are visible in the Calendar View	Reservation of status PENDING, OK or REJECTED should only be displayed	Success	
5	While in Calendar view, switch between different testbeds by changing selection in the corresponding combo box	Reservations only for the selected testbeds are available	Success	new step added in D4.9
6	(Repeat action in step 5)	While selecting different testbeds verify also that the displayed Calendar timeslots adhere to the testbed	Success	new step added in D4.9

		operational hours as defined in the Testbed DB table		
7	Check filtering of calendar displayed events by setting/modifying the filter textbox and clicking the apply button	Based on filter options certain booking events may become visible or invisible	Success	new step added in D4.9

Table 36: Verification test of the Booking Tool Calendar View Interactions

Test ID: **BT02**	Conducted by: **HAI**	Date: **June 2018**	Test Category: **Verification Tests (web tier)**	
Hardware Configuration	-			
Software Configuration	-			
Test Name:	***Booking Tool Calendar View Interactions***			
Preconditions	• connection to the Booking Service ok • user has logged in the web portal • reservations of different status exist in the Master DB			
Related Requirements	PT-BOO-T-001, PT-BOO-T-003, PT-BOO-T-005, PT-BOO-T-006, PT-BOO-T-016, PT-BOO-S-002, PT-BOO-S-004			
Tools Used				
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Click on an empty calendar timeslot (result should depend on the relevance of the timeslot to the present time)	If click occurs on a past timeslot a popup warning is displayed	Success	
		If click occurs on a future timeslot the "Create Reservation" window opens	Success	
2	Click on an existing reservation (result should depend on the relevance of the reservation to the present time)	If click occurs on a past reservation the "Edit Reservation" window opens but no further actions are offered to the user	Success	
3	(see also test BT04)	If click occurs on a future reservation the "Edit Reservation" window opens and the user can perform certain actions on the reservation. Displayed actions depend on user role and reservation status	Success	
4	verify the displayed color for each reservation (click existing reservations)	Coloring of reservation should differ based on the reservation status (shown in the Edit Reservation window)	Success	
5	Perform steps 1-3 after selecting different testbeds in the provided drop down list	Verify that when a testbed is selected in the corresponding Calendar view drop down box then only resources from this	Success	new step added in D4.9

		specific testbed are displayed in all popup windows (Create/Edit/View reservations)		
7	verify the time options available during reservation edit/create	The time steps for begin and end time should not fall outside the testbed defined operation hours	Success	new step added in D4.9

Table 37: Verification test of the Booking Tool Create Reservation

Test ID: **BT03**	Conducted by**: HAI**	Date**: June 2018**	Test Category**: Verification Tests (web tier)**
Hardware Configuration	See section 2.4		
Software Configuration	See section 2.4		
Test Name:	***Booking Tool Create Reservation***		
Preconditions	• connection to the Booking Service ok • user has logged in the web portal • user has clicked on an empty future timeslot		
Related Requirements	PT-BOO-T-001, PT-BOO-T-003, PT-BOO-T-004, PT-BOO-T-009 PT-BOO-T-010, PT-BOO-T-011, PT-BOO-T-017, PT-BOO-S-006		
Tools Used			

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	User edits the field of the “Create Reservation” form so that no time overlapping with other reservation exists and presses the OK button (no conflicts scenario)	Reservation is created and displayed in the Calendar View. Reservation is put in PENDING state	Success	
2	User edits the field of the “Create Reservation” form so that a time overlapping with other reservation exists and presses the OK button (possible conflict scenario)	If no common resources exist with the overlapping reservation then the new reservation is created and displayed in the Calendar View. Reservation is put in PENDING state	Success	
		If common resources exist with the overlapping reservation then the new reservation is not created and a warning message is displayed	Success	Result may depend on status of pre-existing reservation

Table 38: Verification test of the Booking Tool Edit Reservation Actions

Test ID: **BT04**	Conducted by: **HAI**	Date: **June 2018**	Test Category: **Verification Tests (web tier)**
Hardware Configuration	See section 2.4		
Software Configuration	See section 2.4		
Test Name:	***Booking Tool Edit Reservation Actions***		
Preconditions	• connection to the Booking Service ok • user has logged in the web portal • user has clicked on an existing future reservation		
Related Requirements	PT-BOO-T-003, PT-BOO-T-005, PT-BOO-T-007, PT-BOO-T-008 PT-BOO-T-010, PT-BOO-T-011, PT-BOO-T-013, PT-BOO-T-014 PT-BOO-S-006, PT-NF-002		
Tools Used			

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	The actions available to the Edit Reservation window depend on the: • status of reservation • user • role of the user			

	status=PENDING
	user= owner of reservation
	role= EXPERIMENTER
		Actions available:
		OK, CANCEL DELETE
			Success	

	status=OK
	user= owner of reservation
	role= EXPERIMENTER
		Actions available:
		OK, CANCEL DELETE
			Success	

	status=REJECTED
	user= owner of reservation
	role= EXPERIMENTER
		Actions available:
		OK, CANCEL DELETE
			Success	

	status=PENDING
	user= owner of reservation
	role= TESTBED_OP
		Actions available:
		OK, CANCEL, DELETE, APPROVE, REJECT
			Success	

	status=PENDING
	user= not owner of reservation
	role= TESTBED_OP
		Actions available:
		CANCEL, APPROVE, REJECT
			Success	

	status=OK
	user= owner of reservation
	role= TESTBED_OP
		Actions available:
		CANCEL, DELETE, REJECT
			Success	

	status=OK
	user= not owner of reservation
	role= TESTBED_OP
		Actions available:
		CANCEL, REJECT
			Success	

	status=REJECTED
	user= owner of reservation
	role= TESTBED_OP
		Actions available:
		CANCEL, DELETE, APPROVE
			Success	

	status= REJECTED
	user= not owner of reservation
	role= TESTBED_OP
		Actions available:
		CANCEL, APPROVE
			Success	

	user= not owner of reservation	No actions available	Success	
2	Owner of reservation performs changes to the reservation and presses OK button	If the changes do NOT introduce conflicts in both timeslots and selected resources then the reservation is successfully updated and the UI refreshed to display the changes	Success	
		If the changes do introduce	Success	

		conflicts in both timeslots and selected resources then a warning message appears and no further action is performed		
3	Owner of reservation presses DELETE button	If reservation does not refer to a currently running experiment then it is put in a CANCELLED state and removed from the UI	Success	
4	User with TESTBED_OP role presses APPROVE button	If no resource conflicts with already created reservation exists then reservation status becomes OK and color changes appropriately in the Calendar view	Success	
5	User with TESTBED_OP role presses REJECT button	reservation status becomes REJECTED and color changes appropriately in the Calendar view	Success	

Table 39: Verification test of the Booking Tool SFA integration

Test ID: **BT05**	Conducted by**: HAI**	Date**: July 2018**	Test Category**: Verification Tests (web tier)**
Hardware Configuration	-		
Software Configuration	-		
Test Name:	***Booking Tool SFA Integration***		
Preconditions	• connection to the Booking Service ok • connection to the SFA Aggregate Manager ok • user has logged in the web portal • user has clicked on an empty future timeslot		
Related Requirements	TB-AGG-001, TB-AGG-002, TB-AGG-004, TB-AGG-005, PT-BOO-T-002		
Tools Used			

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Replicate all steps defined in BT03 (creation of the reservation)	Verify by the SFA UI (i.e. MySlice) that there exists a reservation for the involved resources in the Aggregate Manager data store	Success	
2	Replicate steps 3 & 4 of BT04	Verify the status of reservation is also updated in Aggregate Manager	Success	
3	Perform a reservation of resources from the MySlice interface`	After refreshing the calendar view, verify that a reservation exists for these resources	Success	

2.6.1.5 *Experiment Authoring Tool*

Table 40: Verification test of the in-Textual Editor Experiments definition

Test ID: **EAT01**	Conducted by**: UoA**		Date**: April 2017**	Test Category**: Verification Tests (front end tier – middle tier)**
Hardware Configuration	See section 2.4			
Software Configuration	See section 2.4			
Test Name:	***Define Experiments in the Textual Editor***			
Preconditions	• User entered in the RAWFIE Portal			
Related Requirements	PT-EXA-T-001, PT-EXA-T-002, PT-EXA-T-003, PT-EXA-T-004, PT-EXA-T-005, PT-EXA-T-006, PT-EXA-T-007, PT-EXA-T-008, PT-EXA-T-009, PT-EXA-T-010, PT-EXA-T-011, PT-EXA-T-012, PT-EXA-T-013, PT-EXA-T-014, PT-EXA-T-015, PT-EXA-T-016, PT-EXV-S-002			
Tools Used	• RAWFIE Web Portal • RAWFIE Textual Editor			
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Access to the Textual Editor through the RAWFIE Web Portal	Redirection to the Textual Editor interface	Success	The redirection was smoothly concluded
2	Write an experiment	Experiment is presented in the editor	Success	The experiment was presented in the editor
3	Utilize code completion, content assist and compilation	The editor responds with specific drop down lists, messages, etc.	Success	All the functionalities were smoothly concluded
4	Define erroneous commands in the experiment workflow	The editor responds with error messages and indication for correcting the error	Success	All the erroneous commands were correctly identified
5	Save the experiment	The experiment is stored in the database and specific files are produced to be adopted by the remaining RAWFIE components	Success	The experiment is correctly stored in the database

Table 41: Verification test of the Textual Editor Experiments Update

Test ID: **EAT02**	Conducted by**: UoA**	Date**: April 2017**	Test Category**: Verification Tests (front end tier – middle tier)**
Hardware Configuration	See section 2.4		
Software Configuration	See section 2.4		
Test Name:	***Update Experiments in the Textual Editor***		
Preconditions	• User entered in the RAWFIE Portal		
Related Requirements	PT-EXA-T-001, PT-EXA-T-002, PT-EXA-T-003, PT-EXA-T-004, PT-EXA-T-005, PT-EXA-T-007, PT-EXA-T-008, PT-EXA-T-009, PT-EXA-T-010, PT-EXA-T-011, PT-EXA-T-012, PT-EXA-T-013, PT-EXA-T-014, PT-EXA-T-015, PT-EXA-T-016		
Tools Used	• RAWFIE Web Portal • RAWFIE Textual Editor		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Access to the Textual Editor through the RAWFIE Web Portal	Redirection to the Textual Editor interface	Success	The redirection was smoothly concluded
2	Open an already defined experiment	Experiment is presented in the editor	Success	The experiment was presented in the editor
3	Makes changes in the experiment workflow	The experiment is updated	Success	All changes were depicted in the editor
4	Save the experiment	The experiment is stored in the database and specific files are produced to be adopted by the remaining RAWFIE components	Success	The experiment was successfully stored in the database

Table 42: Verification test of the in-Visual Editor Experiments Define

Test ID: **EAT03**	Conducted by**: UoA**	Date**: April 2017**	Test Category**: Verification Tests (front end tier – middle tier)**
Hardware Configuration	See section 2.4		
Software Configuration	See section 2.4		
Test Name:	***Define Experiments in the Visual Editor***		
Preconditions	• User entered in the RAWFIE Portal		
Related Requirements	PT-EXA-T-001, PT-EXA-T-002, PT-EXA-T-003, PT-EXA-T-004, PT-EXA-T-005, PT-EXA-T-007, PT-EXA-T-008, PT-EXA-T-009, PT-EXA-T-010, PT-EXA-T-011, PT-EXA-T-012, PT-EXA-T-013, PT-EXA-T-014, PT-EXA-T-015, PT-EXA-T-016		
Tools Used	• RAWFIE Web Portal • RAWFIE Visual Editor		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Access to the Visual Editor through the RAWFIE Web Portal	Redirection to the Visual Editor interface	Success	The editor was correctly depicted in the portal
2	Access the available toolbar	Specific windows are presented	Success	The user can have easy access in the toolbar
3	Create an experiment by utilizing the available tools	The experimenter can define waypoints and experiment information by clicking and designing in the visual editor	Success	The experiment was easily defined by the user
4	Define erroneous commands	The authoring tool responds with error messages and indication for correcting the error	Success	Erroneous commands were correctly identified in the editors
5	Save the experiment	The experiment is stored in the database and specific files are produced to be adopted by the remaining RAWFIE components	Success	The experiment was correctly stored in the database

Table 43: Verification test of the in-Visual Editor Experiments Update

Test ID: **EAT04**	Conducted by**: UoA**		Date**: April 2017**	Test Category**: Verification Tests (front end tier – middle tier)**
Hardware Configuration	See section 2.4			
Software Configuration	See section 2.4			
Test Name:	***Update Experiments in the Visual Editor***			
Preconditions	• User entered in the RAWFIE Portal			
Related Requirements	PT-EXA-T-001, PT-EXA-T-002, PT-EXA-T-003, PT-EXA-T-004, PT-EXA-T-005, PT-EXA-T-007, PT-EXA-T-008, PT-EXA-T-009, PT-EXA-T-010, PT-EXA-T-011, PT-EXA-T-012, PT-EXA-T-013, PT-EXA-T-014, PT-EXA-T-015, PT-EXA-T-016			
Tools Used	• RAWFIE Web Portal • RAWFIE Visual Editor			
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Access to the Visual Editor through the RAWFIE Web Portal	Redirection to the Visual Editor interface	Success	The editor was correctly depicted in the portal
2	Open an already defined experiment	Experiment is presented in the editor	Success	The user can easily open an already stored experiment
3	Makes changes in the experiment workflow	The experiment is updated	Success	The user can easily make changes in both editors
4	Save the experiment	The experiment is stored in the database and specific files are produced to be adopted by the remaining RAWFIE components	Success	The experiment was correctly stored

Table 44: Verification test of the Editor switching

Test ID: **EAT05**	Conducted by**: UoA (test modified in D4.9)**		Date: **October 2017**	Test Category**: Verification Tests (front end tier – middle tier)**
Hardware Configuration	See section 2.4			
Software Configuration	section 2.4			
Test Name:	***Switch between the Editors***			
Preconditions	• User entered in the RAWFIE Portal			
Related Requirements	PT-EXA-T-001, PT-EXA-T-002, PT-EXA-T-003, PT-EXA-T-004, PT-EXA-T-005, PT-EXA-T-008, PT-EXA-T-009, PT-EXA-T-010, PT-EXA-T-011, PT-EXA-T-012, PT-EXA-T-013, PT-EXA-T-015, PT-EXA-T-017			
Tools Used	• RAWFIE Web Portal • RAWFIE Textual Editor • RAWFIE Visual Editor			
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Access to the editors through the RAWFIE Web Portal	Redirection to the editor interface	Success	The editors were smoothly opened
2	Create an experiment	Experiment is presented in the editor interface	Success	The user created an expriment in the textual

				editor and synchronized the editors
3	Switch to the alternative editor and make changes	The experiment is updated	Success	Both editors are always showing the same experiment definition at any time – The user can make cases in both editors - The synchronization was correct
4	Save the experiment	The experiment is stored in the database and specific files are produced to be adopted by the remaining RAWFIE components	Success	The experiment was correctly stored in the database

Table 45: Verification test of the experiment Launchings

Test ID: **EAT06**	Conducted by**: UoA**	Date**: April 2017**	Test Category**: Verification Tests (front end tier – middle tier)**
Hardware Configuration	See section 2.4		
Software Configuration	See section 2.4		
Test Name:	***Launch experiments***		
Preconditions	• User entered in the RAWFIE Portal		
Related Requirements	PT-EXA-T-001, PT-EXA-T-002, PT-EXA-T-003, PT-EXA-T-004, PT-EXA-T-005, PT-EXA-T-008, PT-EXA-T-009, PT-EXA-T-010, PT-EXA-T-011, PT-EXA-T-012, PT-EXA-T-013, PT-EXA-T-015		
Tools Used	• RAWFIE Web Portal • RAWFIE Textual - Visual Editors • RAWFIE Launching Tool		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Access to the authoring tool through the RAWFIE Web Portal	Redirection to the editors interface	Success	The authoring tool opens smoothly
2	Select an experiment	A drop down list of the available experiments is appeared and the experimenter has the opportunity to select one	Success	The experiment can be selected and opened
3	Start the experiment execution	The launching service is informed with the experiment ID and the execution starts	Success	After clicking in the appropriate button, the required information was transferred to the launching service

Table 46: Verification test of the experiment Launchings

Test ID: **EAT07**	Conducted by**: UoA (new test in D4.9)**	Date**: October 2017**	Test Category**: Verification Tests (front end tier – middle tier)**
Hardware Configuration	-		
Software Configuration	•		
Test Name:	***Launch (scheduled) experiments***		
Preconditions	• User entered in the RAWFIE Portal		
Related Requirements	PT-EXA-T-001, PT-EXA-T-002, PT-EXA-T-003, PT-EXA-T-004, PT-EXA-T-005, PT-EXA-T-008, PT-EXA-T-009, PT-EXA-T-010, PT-EXA-T-011, PT-EXA-T-012, PT-EXA-T-013, PT-EXA-T-015		
Tools Used	• RAWFIE Web Portal • RAWFIE Textual - Visual Editors • RAWFIE Launching Tool		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Access to the authoring tool through the RAWFIE Web Portal	Redirection to the editor interface	Success	The authoring tool opens smoothly
2	Select the scheduled launching tool	A drop-down list of the available experiments is appeared and the experimenter has the opportunity to select one	Success	The experiment can be selected and opened
3	Define the experiment execution	The launching service is informed with the experiment ID and the execution is planned	Success	After clicking in the appropriate button, the required information was transferred to the launching service (scheduled launching)

2.6.1.6 *Experiment Monitoring Tool*

Table 47: Verification test of the Visualisation of experiment status

Test ID: **EMT01**	Conducted by: **Fraunhofer**	Date: **May 2018**	Test Category: **Verification Tests (front end tier)**
Hardware Configuration	See section 2.4		
Software Configuration	See section 2.4		
Test Name:	***Visualisation of experiment status***		
Preconditions	• connection to the Launching Service ok • knowledge about the experiments state needed on user side (to check results)		
Related Requirements	PT-EXM-T-001, PT-EXM-T-002		
Tools Used	• Browser		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	user opens Experiment Monitoring Tool in the Web Portal	Experiment Monitoring Tool displays a view with all experiments of the current user (ordered by date descending). The list also contains a sort summary of the experiments state	Success	
2	user selects a experiment	Experiment Monitoring Tool displays all experiment details (date / timespan; related testbed; list of used UxVs; execution state ; link to the used EDL)	Success	Additionally health status and review status are shown

Table 48: Verification test of the canceling of experiments

Test ID: **EMT02**	Conducted by: **Fraunhofer**	Date: **May 2018**	Test Category: **Verification Tests (front end tier)**
Hardware Configuration	-		
Software Configuration	-		
Test Name:	***Cancel of experiment***		
Preconditions	• Experiments running		
Related Requirements	PT-EXM-T-003, PT-EXP-C-001, PT-LAU-S-010, PT-LAU-S-012, TB-MAN-005		
Tools Used	•		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	user opens Experiment Monitoring Tool in the Web Portal	Experiment Monitoring Tool displays a view with all experiments of the current user	Success	
2	user selects an experiment	Experiment Monitoring Tool displays all experiment details and the option to cancel it	Success	
3	User clicks the cancel button	Cancellation request is sent. User is informed about the ongoing cancellation	Success	
4	User watches further the experiment status	Experiment status is set to “cancelled” when the cancellation is complete	Success	

2.6.1.7 System Monitoring Tool

Table 49: Verification test of the Visualisation of system and UxV health status

Test ID: **SMT01**		Conducted by: **Fraunhofer**	Date: **May 2018**	Test Category: **Verification Tests (front end tier)**
Hardware Configuration				
Software Configuration				
Test Name:		***Visualisation of system and UxV health status***		
Preconditions		• connection to the System Monitoring Service • administrative knowledge about the system state needed on user side (to check results)		
Related Requirements		PT-SYM-T-001, PT-SYM-T-002, PT-SYM-T-004, PT-SYM-T-005, PT-SYM-S-007		
Tools Used		• Browser		
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	user opens System Monitoring Tool in the Web Portal	the System Monitoring Tool displays a view with severity indication and textual information of middleware components, testbeds components, UxVs components	Success	
2	User sets some sorting (name, server/testbed/UxV) and filter options to see the services he is interested in.	Monitoring Tool filters and sorts the data accordingly	Success	
3	User watches the web site for a while	Displayed data is updated automatically (e.g. last update time)	Success	
4	User manually triggers a change in a health status of a component	Displayed health status of the component should change	Success	

Table 50: Verification test of the Filtering based on roles

Test ID: **SMT02**		Conducted by: **Fraunhofer**	Date: **May 2018**	Test Category: **Verification Tests (front end tier)**
Hardware Configuration				
Software Configuration				
Test Name:		***Filtering based on roles***		
Preconditions		• connection to the System Monitoring Service • administrative knowledge about the system state needed on user side (to check results)		
Related Requirements		PT-SYM-T-003		
Tools Used		• Browser		
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	User with admin rights logs in	Logged in	Success	
2	User opens System Monitoring Tool in the Web Portal	User sees all servers, testbeds an UxVs	Success	
2	User with experimenter rights logs in	Logged in	Success	
3	User opens System Monitoring Tool in the Web Portal	User sees only testbeds an UxVs	Success	

Table 51: Verification test of the Administrative Monitoring View

Test ID: **SMT03**		Conducted by: **Fraunhofer**	Date: **May 2018**	Test Category: **Verification Tests (front end tier)**
Hardware Configuration				
Software Configuration				
Test Name:		***Administrative Monitoring View***		
Preconditions		• connection to the System Monitoring Service • administrative knowledge about the system state needed on user side (to check results)		
Related Requirements		PT-SYM-T-001, PT-SYM-T-004, PT-SYM-T-005, PT-SYM-T-006, PT-SYM-S-007, PT-SYM-S-009		
Tools Used		• Browser • SSH client		
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	user opens the Icinga Web application	Icinga Web shows the dashboard with the status information	Success	
2	User watches the web site for a while	Displayed data is updated automatically (e.g. last update time)	Success	
3	User manually triggers a change in a health status of a component	Displayed health status of the component should change	Success	
4	User opens detail page of a service	Details of the service are shown	Success	
5	User opens history page of a service	History with health status changes of the service are shown	Success	

(See also tests for System Monitoring Service)

2.6.1.8 *Visualisation Tool*

Table 52: Verification test of the User request handling

Test ID: **VIS01**		Conducted by**: Aberon (test modified in D4.9)**	Date**: June 2018**	Test Category**: Verification Tests (front end)**
Hardware Configuration				
Software Configuration				
Test Name:		***User request handling***		
Preconditions		● Requires visualization tool to be functioning & accessible. ● Requires visualization engine to be functioning & accessible.		
Related Requirements		PT-VIS-E-001, PT-VIS-E-003, PT-VIS-E-005, PT-EXP-C-002, PT-EXP-C-003, PT-EXP-C-004, PT-EXP-C-006, PT-EXP-C-007, PT-EXP-C-008, PT-EXP-C-009, PT-VIS-T-001, PT-VIS-T-002, PT-VIS-T-003, PT-VIS-T-004, PT-VIS-T-005, PT-VIS-T-006, PT-VIS-T-007		
Tools Used		●		
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	A first user starts one of the experiments from the experiment list	The visualization tool forwards it to the visualization engine	Success	
2	the visualisation engine starts the visualisation of the first experiment and forwards the data to the first user	The map is loaded and the experiment is visualized on the first user's screen	Success	
3	A second user starts visualizing another experiment from another computer	The visualization tool forwards it to the visualization engine	Success	
4	the visualisation engine starts the visualisation of the second experiment and forwards the data to the second user	The map is loaded and the experiment is visualized on the second user's screen	Success	

Table 53: Verification test of the Geospatial data handling

Test ID: **VIS02**		Conducted by: **Aberon (test modified in D4.9)**	Date: **June 2018**	Test Category: **Verification Tests (front end)**
Hardware Configuration				
Software Configuration				
Test Name:		***Geospatial data handling***		
Preconditions		● Requires visualization tool to be functioning & accessible. ● Requires visualization engine to be functioning & accessible. ● Requires message bus to be functioning & accessible.		
Related Requirements		PT-VIS-E-001, PT-VIS-E-002, PT-VIS-E-003, PT-VIS-E-004, PT-EXP-C-002, PT-EXP-C-003, PT-EXP-C-004, PT-EXP-C-006, PT-EXP-C-007, PT-EXP-C-008, PT-EXP-C-009, PT-VIS-T-001, PT-VIS-T-002, PT-VIS-T-003, PT-VIS-T-004, PT-VIS-T-005, PT-VIS-T-006, PT-VIS-T-007		
Tools Used		●		
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	User starts an already running experiment	Request is forwarded to the VE	Success	
2	The VE sends the data for the experiment in the correct format to the VT	VT presents the data for the experiment in layers to the user	Success	

Table 54: Verification test of the Geospatial data modification

Test ID: **VIS03**		Conducted by: **Aberon (test modified in D4.9)**	Date: **June 2018**	Test Category: **Verification Tests (front end)**
Hardware Configuration				
Software Configuration				
Test Name:		***Geospatial data modification***		
Preconditions		● Requires visualization tool to be functioning & accessible. ● Requires visualization engine to be functioning & accessible. ● Requires message bus to be functioning & accessible.		
Related Requirements		PT-VIS-E-001, PT-VIS-E-003, PT-EXP-C-002, PT-EXP-C-003, PT-EXP-C-004, PT-EXP-C-006, PT-EXP-C-007, PT-EXP-C-008, PT-EXP-C-009, PT-VIS-T-001, PT-VIS-T-002, PT-VIS-T-003, PT-VIS-T-004, PT-VIS-T-005, PT-VIS-T-006, PT-VIS-T-007		
Tools Used		● Browser		
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	User starts an already running experiment	Data is visualized properly to the user	Success	
2	User turns off a layer with data	VT hides the data from this layer from the user	Success	
3	User turns on a layer with data from the experiment	VT requests this data from the VE, receives it and shows it to the user in the proper layer	Success	

Table 55: Verification test of the Experiment Controller communication

Test ID: **VIS04**		Conducted by: **Aberon (test modified in D4.9)**	Date: **June 2018**	Test Category: **Verification Tests (front end)**
Hardware Configuration				
Software Configuration				
Test Name:		***Experiment Controller communication***		
Preconditions		● Requires experiment controller to be functioning & accessible. ● Requires visualization engine to be functioning & accessible.		
Related Requirements		PT-VIS-E-001, PT-VIS-E-003, PT-EXP-C-002, PT-EXP-C-003, PT-EXP-C-004, PT-EXP-C-006, PT-EXP-C-007, PT-EXP-C-008, PT-EXP-C-009, PT-VIS-T-001, PT-VIS-T-002, PT-VIS-T-007		
Tools Used				
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	The user starts an experiment	The message is forwarded to the visualisation engine	Success	
2	Receive a message that the experiment has started from the Experiment Controller	The visualization tool starts the experiment and loads the map	Success	
3	Receive a message that the experiment has stopped from the Experiment Controller	The VT stops the experiment and the user gets a notification about that event	Success	

Table 56: Verification test of the Visualization Tool Interaction

Test ID: **VIS05**		Conducted by: **Aberon (test modified in D4.9)**	Date: **June 2018**	Test Category: **Verification Tests (front end)**
Hardware Configuration				
Software Configuration				
Test Name:		***Visualization Tool Interaction***		
Preconditions		● Requires visualization tool to be functioning & accessible. ● Requires visualization engine to be functioning & accessible.		
Related Requirements		PT-VIS-E-001, PT-VIS-E-003, PT-VIS-T-001, PT-VIS-T-002, PT-VIS-T-003, PT-VIS-T-004, PT-VIS-T-005, PT-VIS-T-006, PT-VIS-T-007		
Tools Used		●		
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Enable different features of the visualization tool (e.g. show/hide speed web widget)	The user sees the updated plot (show speed web widget)	Success	
2	Disable a feature (e.g. speed web widget)	The widget is removed from the screen	Success	

Table 57: Verification test of the Indoor maps

Test ID: **VIS06**	Conducted by: **Aberon (test modified in D4.9)**		Date: **June 2018**	Test Category: **Verification Tests (front end)**
Hardware Configuration				
Software Configuration				
Test Name:	***Indoor maps interaction***			
Preconditions	● Requires visualization tool to be functioning & accessible. ● Requires visualization engine to be functioning & accessible. ● Requires Experiment controller to be functioning & accessible. ● Requires an indoor map to be loaded in the GeoServer			
Related Requirements	PT-VIS-E-001, PT-VIS-E-003, PT-VIS-T-001, PT-VIS-T-002, PT-VIS-T-003, PT-VIS-T-004, PT-VIS-T-005, PT-VIS-T-006, PT-VIS-T-007, PT-VIS-T-008			
Tools Used	●			

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Open the visualization tool, list all experiments	All experiments owned by the user are displayed	Success	
2	Start an experiment with indoor maps	An experiment is loaded, the indoor map is loaded from the GeoServer and is shown on the screen	Success	
3	A UxV moves	The data from the VE is received and plotted on the screen	Success	

2.6.1.9 *Data Analysis Tool*

Table 22: Verification test of starting a data analysis task on the DAE via the DAT

Test ID**: DAT01**	Conducted by**: HES-SO**	Date**:**	Test Category**: Verification Tests (front end)**
Hardware Configuration			
Software Configuration			
Test Name:	***Start a data analysis task on the DAE via the DAT***		
Preconditions	● Requires the message bus to be functioning and accessible ● Requires the schema registry to be functioning and accessible ● Requires the Zeppelin notebook interface of the DAT to be functioning and accessible ● Requires result repository to be functioning and accessible		
Related Requirements	PT-DAA-T-001, PT-DAA-T-003, PT-DAA-T-005, PT-DAA-T-006, PT-DAA-T-007, PT-DAA-T-008		
Tools Used	●		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Authorized user logs into the web portal and clicks on the schema registry tab of the Data Analysis Tool GUI embedded into the web portal	Login successful, successfully reaches the schema registry GUI tab of the Data Analysis Tool GUI embedded into the web portal	Success	
2	User selects the topics and fields corresponding to streaming data currently present on the message bus to perform an analysis task on, then clicks on the "create Zeppelin notebook" button once the desired elements have been selected.	A Zeppelin notebook has been successfully created, and is already populated with the topics and fields selected by the user.	Success	
3	User designs an analysis task in the notebook relying on Spark and starts it within the notebook.	The job has been successfully started. The process should be visible through the spark master UI of the Data Analysis Tool. Additionally, if the streaming results are published to the time series database (result repository), the results should be visible on the Grafana dashboard (part the Data Analysis Tool).	Success	

Table 22: Verification test of retrieving data from the message bus

Test ID: **DAT02**		Conducted by**: HES-SO**	Date**:**	Test Category**: Verification Tests (front end)**
Hardware Configuration				
Software Configuration				
Test Name:		***Retrieve data from the message bus***		
Preconditions		● Requires the message bus to be functioning and accessible ● Requires the schema registry to be functioning and accessible ● Requires result repository to be functioning and accessible		
Related Requirements		PT-DAA-T-00, PT-DAA-T-006, PT-DAA-T-007, PT-DAA-T-008		
Tools Used		●		
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Authorized user logs into the web portal and clicks on the schema registry tab of the Data Analysis Tool GUI embedded into the web portal	Login successful, successfully reaches the schema registry GUI tab of the Data Analysis Tool GUI embedded into the web portal	Success	
2	User selects the topics and fields corresponding to streaming data currently present on the message bus to perform an analysis task on, then clicks on the “create Zeppelin notebook” button once the desired elements have been selected.	A Zeppelin notebook has been successfully created, and is already populated with the topics and fields selected by the user.	Success	
3	User designs a streaming analysis task in the notebook to be performed on data from the message bus and starts it within the notebook.	The data is successfully retrieved and the analysis task therefore can process it and display the results on the Grafana dashboard.	Success	

Table 22: Verification test of ending a running job

Test ID: **DAT03**		Conducted by**: HES-SO**	Date**:**	Test Category**: Verification Tests (front end)**
Hardware Configuration				
Software Configuration				
Test Name:		***End a running job***		
Preconditions		● Requires the message bus to be functioning and accessible ● Requires the schema registry to be functioning and accessible ● Requires the Zeppelin notebook interface of the DAT to be functioning and accessible ● Requires result repository to be functioning and accessible		
Related Requirements		PT-DAA-T-004, PT-DAA-T-003, PT-DAA-T-005, PT-DAA-T-006, PT-DAA-T-007		
Tools Used		●		
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Authorized user logs into the web portal and clicks on the schema registry tab of the Data Analysis Tool GUI embedded into the web portal	Login successful, successfully reaches the schema registry GUI tab of the Data Analysis Tool GUI embedded into the web portal	Success	
2	User selects the topics and fields corresponding to streaming data currently present on the message bus to perform an analysis task on, then clicks on the "create Zeppelin notebook" button once the desired elements have been selected.	A Zeppelin notebook has been successfully created, and is already populated with the topics and fields selected by the user.	Success	
3	User designs an streaming analysis task in the notebook to be performed on data from the message bus and starts it within the notebook.	The data is successfully retrieved and the analysis task therefore can process it and display the results on the Grafana dashboard.	Success	
4	User stops the running job within the Zeppelin notebook	The job has been successfully stopped (results stopped being sent to the dashboard)	Success	

Table 22: Verification test of accessing past results

Test ID**: DAT04**	Conducted by**: HES-SO**	Date**:**	Test Category**: Verification Tests (front end)**
Hardware Configuration			
Software Configuration			
Test Name:	***Access past results***		
Preconditions	● Requires the message bus to be functioning and accessible ● Requires the schema registry to be functioning and accessible ● Requires the Zeppelin notebook interface of the DAT to be functioning and accessible ● Requires result repository to be functioning and accessible		
Related Requirements	PT-DAA-T-002, PT-DAA-T-005, PT-DAA-T-007, PT-DAA-T-008		
Tools Used	●		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Authorized user logs into the web portal and clicks on the results repository tab of the Data Analysis Tool GUI embedded into the web portal	Login successful, successfully reaches results repository GUI (Grafana dashboard) tab of the Data Analysis Tool GUI embedded into the web portal	Success	
2	User uses the Grafana dashboard interface to display results of previous time steps.	The dashboard allows such browsing and displays the past results of the associated experiment (associated to a metric) correctly	Success	
3	User accesses the data persistently stored on Grafana's underlying time series database vias CLI.	The data is correctly accessed.	In progress	The data storage is done via HDFS, which can be accessed through the DAT.

2.6.2 Middle Tier (Services and Communication components)

2.6.2.1 Testbed Directory Service

Table 58: Verification test of the resources information retrieval and resources search

Test ID**: TD01**	Conducted by**: IES**	Date: **February 2017, June 2018**	Test Category**: Verification Tests (Middle Tier)**
Hardware Configuration			
Software Configuration			
Test Name:	***Retrieve resources information and search for specific resources***		
Preconditions	Access to the PostgreSQL server must be granted for the Testbed Directory Service When preparing the test, the test executor should know either the ID of the resource he/she is looking for, or other parameters according to the criteria he/she is using for selecting specific resources		
Related Requirements	PT-DIR-S-003, PT-DIR-S-004, PT-DIR-S-006		
Tools Used			

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1.a	The input request is prepared, specifying in input the testbed identifier and the resource status (for the ***/request/getResourcesByStatus()*** REST interface), nothing in case the ***/request/getAllResources()*** REST interface is used	No error occurred. The Testbed Directory Service gives back a JSON response message, containing details about all resources in a specific status (e.g. Booked, Released, Sleep_mode), or all resources in case the *getAllResources()* interface is used	Success	Addition of the *getResourcesByStatus* method according to the last requirements iteration and the subsequent updated component design in the D4.9
2.a	The ***/request/getAllResources()* (without parameters) or *request/ getResourcesByStatus()*** REST interfaces can be called from the SOAP UI Client Tool.			
1.b	The /***request/resource/identifier/{id}*** REST interface is called (from the browser or using a tool like SOAP UI), specifying the id of a specific resource	No error occurred. The Testbed Directory Service gives back a JSON response message, containing detailed information about the resource (or the list of resources) matching the search criteria	Success	
2.b	The /***request/resource/name/{name}*** REST interface is called (from the browser or using a tool like SOAP UI), specifying the name of a specific resource			
3.b	The ***/request/resources?param1=value1&param2=value2&param3=value3&param4=value4*** REST interface is called (from the browser or using a tool like SOAP UI), with one or more query parameters according to the selected search criteria, that is, a combination of one or more of the following 4 possible search parameters: • *resource_status* • *resource_status_message* • *resource_type* • *health*			
4.b	The /***request/resources/testbedid/{id}*** REST interface is called (from the browser or using a tool like SOAP UI), specifying the id of the Testbed we would like to get resources from			

Table 59: Verification tests for adding, editing or removing a testbed facility

Test ID: **TD02**	Conducted by**: IES**	Date**: February 2017, June 2018**	Test Category**: Verification Tests (Middle Tier)**
Hardware Configuration	See section 2.4		
Software Configuration	See section 2.4		
Test Name:	***Add / Edit / Delete a testbed facility***		
Preconditions	Access to the PostgreSQL server must be granted for the Testbed Directory Service When preparing the test for the testbed registration case, the test executor should know the information about the testbed to be inserted. In case of a testbed deletion, the testbed id must be known in advance		
Related Requirements	PT-DIR-S-005		
Tools Used	SOAP UI		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1.a	The input JSON request is prepared, with the information about the new testbed to be added	No error occurred. And the information about the new testbed is from now on available in the Master Data Repository, as it can be verified by using the ***getAllTestbeds()*** or other REST interfaces for Testbeds searches (see **TD04**)	Success	
2.a	***The /request/createTestbed()*** REST interface is called from the SOAP UI Client Tool, specifying the testbed information in the input JSON request			
1.b	The input JSON request is prepared, with the information about the testbed whose information is to be updated	No error occurred. And the updated testbed information is from now on available in the Master Data Repository, as it can be verified by using the ***getAllTestbeds()*** or other REST interfaces for Testbeds searches (see **TD04**)	Success	Added in D6.5
2.b	***The /request/editTestbed()*** REST interface is called from the SOAP UI Client Tool, specifying the testbed information in the input JSON request			
1.c	The input JSON message request is prepared, with the unique id of the testbed facility to be deleted	No error occurred. And the information about the deleted testbed (and related resources) is not available anymore in the Master Data Repository, as it can be verified by using the ***getAllTestbeds()*** or other REST interfaces (see **TD04** in the following)	Success	
2.c	The ***/request/deleteTestbed()*** REST interface is called from the SOAP UI Client Tool, specifying the information about the testbed to be deleted in the provided input JSON request		Success	

Table 60: Verification test of the registration or removal of a new UxV node into a testbed facility

Test ID: **TD03**	Conducted by: **IES**		Date: **February 2017, June 2018**	Test Category: **Verification Tests (Middle Tier)**
Hardware Configuration	See section 2.4			
Software Configuration	See section 2.4			
Test Name:	***Register / Edit / Delete an UxV node into a testbed facility***			
Preconditions	Access to the PostgreSQL server must be granted for the Testbed Directory Service. When preparing the test, the test executor should know either the ID/name of the resource and testbed he/she is looking for, or the list criteria for selecting specific resources			
Related Requirements	PT-DIR-S-007			
Tools Used	SOAP UI			
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1.a	The input JSON message request is prepared, with all information about the new resource to be added (and the unique id of the testbed facility it belongs to)	No error occurred. And the information about the new resource (UxV node) is from now on available in the Master Data Repository, as it can be verified by using the ***getAllResources()*** or other REST API for Resources searches (see previous tests **TD01**)	Success	
2.a	The ***/request/createResource()*** REST interface is called from the SOAP UI Client Tool, specifying the information about the resource to be added in the provided input JSON request			
1.b	The input JSON request is prepared, with the information about the resource whose information is to be updated (and the unique id of the testbed facility it belongs to)	No error occurred. And the updated resource information (UxV node) is from now on available in the Master Data Repository, as it can be verified by using the ***getAllResources()*** or other REST API for Resources searches (see previous tests **TD01**)	Success	Added in D6.5
2.b	***The /request/editResource()*** REST interface is called from the SOAP UI Client Tool, specifying the resource information in the input JSON request			
1.c	The input JSON message request is prepared, with the unique id of the resource to be deleted and of the testbed facility it belongs to	No error occurred. And the resource (UxV node) is not available anymore in the Master Data Repository, as it can be verified by using the ***getAllResources()*** or other REST API (see previous tests **TD01**)	Success	
2.c	The ***/request/deleteResource()*** REST interface is called from the SOAP UI Client Tool, specifying the information about the resource to be deleted in the provided input JSON request			

Table 61: Verification test of the testbeds information retrieval and testbeds search

Test ID: **TD04**	Conducted by: **IES**	Date: **April 2017, June 2018**	Test Category: **Verification Tests (Middle Tier)**	
Hardware Configuration	See section 2.4			
Software Configuration	See section 2.4			
Test Name:	***Retrieve testbed information and search for specific testbeds***			
Preconditions	Access to the PostgreSQL server must be granted for the Testbed Directory Service When preparing the test, the test executor should know the ID of the testbed he/she is looking for, or it can just provide one or a set of search criteria			
Related Requirements	PT-DIR-S-001, PT-DIR-S-002, PT-DIR-S-006			
Tools Used				
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1.a	The /***request/getAllTestbeds()*** REST interface is called from the SOAP UI Client Tool, without any specific testbed information (null JSON input request)	No error occurred. The Testbed Directory Service gives back a JSON response message, containing details about all registered testbeds and all resources belonging to each of them	🟩 Success	
1.b	The ***/request/testbed/identifier/{id}*** REST interface is called from the Browser, specifying the id of a specific testbed	No error occurred. The Testbed Directory Service gives back a JSON response message, containing details about the available testbeds conforming to the search criteria	🟩 Success	
2.b	The /***request/testbed/name/{name}*** REST interface is called, specifying the name of a specific testbed			
3.b	The /***request/testbeds?param1=value1&param2=value2*** REST interface is called, with one or more query parameters according to the selected search criteria, that is, a combination of one or both the following 2 search parameters: • *health* • *testbedstatusmessage*		🟩 Success	
4.b	The /***request/testbed/uav*** REST interface is called, looking for all testbeds supporting UAV resources		🟩 Success	
5.b	The /***request/testbed/ugv*** REST interface is called, looking for all testbeds supporting UGV resources		🟩 Success	
6.b	The /***request/testbed/usv*** REST interface is called, looking for all testbeds supporting USV resources		🟩 Success	

7.b	The /***request/testbed/auv*** REST interface is called, looking for all testbeds supporting AUV resources		Success	

Test ID**: TD05**	Conducted by**: IES**	Date**: June 2018**	Test Category**: Verification Tests (Middle Tier)**
Hardware Configuration	See section 2.4		
Software Configuration	See section 2.4		
Test Name:	***Register / Edit / Delete a Testbed Area***		
Preconditions	Access to the PostgreSQL server must be granted for the Testbed Directory Service.		
Related Requirements			
Tools Used	SOAP UI		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1.a	The input JSON message request is prepared, with all information about the new testbed area to be added (and the unique id of the testbed facility it belongs to)	No error occurred. And the information about the new Area is from now on available in the Master Data Repository	Success	Added in D6.5
2.a	The ***/request/createArea()*** REST interface is called from the SOAP UI Client Tool, specifying the information about the testbed area to be added in the provided input JSON request			
1.b	The input JSON request is prepared, with the information about the testbed area whose information is to be updated (and the unique id of the testbed facility it belongs to)	No error occurred. And the updated testbed area information is from now on available in the Master Data Repository	Success	Added in D6.5
2.b	***The /request/editArea()*** REST interface is called from the SOAP UI Client Tool, specifying the testbed Area information in the input JSON request			
1.c	The input JSON message request is prepared, with the name of the resource to be deleted and the id of the testbed facility it belongs to	No error occurred. And the testbed area is not available anymore in the Master Data Repository	Success	Added in D6.5
2.c	The ***/request/deleteArea()*** REST interface is called from the SOAP UI Client Tool, specifying the information about the testbed area to be deleted in the provided input JSON request			

Test ID: **TD05**	Conducted by**: IES**	Date**: June 2018**	Test Category**: Verification Tests (Middle Tier)**
Hardware Configuration	See section 2.4		
Software Configuration	See section 2.4		
Test Name:	***Register / Edit / Delete a UxV Sensor***		
Preconditions	Access to the PostgreSQL server must be granted for the Testbed Directory Service.		
Related Requirements			
Tools Used	SOAP UI		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1.a	The input JSON message request is prepared, with all information about the new sensor to be added (and the unique id of the resource it belongs to)	No error occurred. And the information about the new sensor is from now on available in the Master Data Repository	Success	Added in D6.5
2.a	The ***/request/createSensor()*** REST interface is called from the SOAP UI Client Tool, specifying the information about the sensor to be added in the provided input JSON request			
1.b	The input JSON request is prepared, with the information about the sensor whose information is to be updated (and the unique id of the resource it belongs to)	No error occurred. And the updated sensor information is from now on available in the Master Data Repository	Success	Added in D6.5
2.b	***The /request/editSensor()*** REST interface is called from the SOAP UI Client Tool, specifying the sensor information in the input JSON request			
1.c	The input JSON message request is prepared, with the name of the sensor to be deleted and the id of the resource (UxV) it belongs to	No error occurred. And the sensor is not available anymore in the Master Data Repository	Success	Added in D6.5
2.c	The ***/request/deleteResource()*** REST interface is called from the SOAP UI Client Tool, specifying the information about the resource to be deleted in the provided input JSON request			

Test ID: **TD06**		Conducted by**: IES**	Date**: June 2018**	Test Category**: Verification Tests (Middle Tier)**
Hardware Configuration		See section 2.4		
Software Configuration		See section 2.4		
Test Name:		***Register / Edit / Delete a Network Interface***		
Preconditions		Access to the PostgreSQL server must be granted for the Testbed Directory Service.		
Related Requirements				
Tools Used		SOAP UI		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1.a	The input JSON message request is prepared, with all information about the new UxV network interface to be added	No error occurred. And the information about the new network interface is from now on available in the Master Data Repository	Success	Added in D6.5
2.a	The ***/request/createNetInterface()*** REST interface is called from the SOAP UI Client Tool, specifying the information about the resource to be added in the provided input JSON request			
1.b	The input JSON request is prepared, with the information about the net interface whose information is to be updated	No error occurred. And the updated net interface information is from now on available in the Master Data Repository)	Success	Added in D6.5
2.b	***The /request/editNetInterface()*** REST interface is called from the SOAP UI Client Tool, specifying the net interface information in the input JSON request			
1.c	The input JSON message request is prepared, with the unique id of the network interface to be deleted	No error occurred. And the net interface is not available anymore in the Master Data Repository	Success	Added in D6.5
2.c	The ***/request/deleteNetInterface()*** REST interface is called from the SOAP UI Client Tool, specifying the information about the net interface (id) to be deleted in the provided input JSON request			

2.6.2.2 EDL Compiler and Validator

Table 62: Verification test of the in-Textual Editor Experiments definition

Test ID: **EAT01**	Conducted by**: UoA**	Date**: April 2017**	Test Category**: Verification Tests (front end tier – middle tier)**
Hardware Configuration	-		
Software Configuration			
Test Name:	***Define Experiments in the Textual Editor***		
Preconditions	• User entered in the RAWFIE Portal		
Related Requirements	PT-EXA-T-001, PT-EXA-T-002, PT-EXA-T-003, PT-EXA-T-004, PT-EXA-T-005, PT-EXA-T-008, PT-EXA-T-009, PT-EXA-T-010, PT-EXA-T-011, PT-EXA-T-012, PT-EXA-T-013, PT-EXA-T-015		
Tools Used	• RAWFIE Web Portal • RAWFIE Textual Editor		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Access to the Textual Editor through the RAWFIE Web Portal	Redirection to the Textual Editor interface	Success	
2	Write an experiment	Experiment is presented in the editor	Success	
3	Utilize code completion, content assist and compilation	The editor responds with specific drop down lists, messages, etc.	Success	
4	Define erroneous commands in the experiment workflow	The editor responds with error messages and indication for correcting the error	Success	
5	Save the experiment	The experiment is stored in the database and specific files are produced to be adopted by the remaining RAWFIE components	Success	The experiment was correctly stored

Table 63: Verification test of the Textual Editor Experiments Update

Test ID: **EAT02**	Conducted by**: UoA**	Date**: April 2017**	Test Category**: Verification Tests (front end tier – middle tier)**
Hardware Configuration	-		
Software Configuration			
Test Name:	***Update Experiments in the Textual Editor***		
Preconditions	• User entered in the RAWFIE Portal		
Related Requirements	PT-EXA-T-001, PT-EXA-T-002, PT-EXA-T-003, PT-EXA-T-004, PT-EXA-T-005, PT-EXA-T-008, PT-EXA-T-009, PT-EXA-T-010, PT-EXA-T-011, PT-EXA-T-012, PT-EXA-T-013, PT-EXA-T-015		
Tools Used	• RAWFIE Web Portal • RAWFIE Textual Editor		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Access to the Textual Editor through the RAWFIE Web Portal	Redirection to the Textual Editor interface	Success	
2	Open an already defined experiment	Experiment is presented in the editor	Success	
3	Makes changes in the experiment workflow	The experiment is updated	Success	
4	Save the experiment	The experiment is stored in the database and specific files are produced to be adopted by the remaining RAWFIE components	Success	The experiment was correctly stored

Table 64: Verification test of the in-Visual Editor Experiments Define

Test ID: **EAT03**		Conducted by: **UoA**	Date: **April 2017**	Test Category: **Verification Tests (front end tier – middle tier)**
Hardware Configuration		-		
Software Configuration		•		
Test Name:		***Define Experiments in the Visual Editor***		
Preconditions		• User entered in the RAWFIE Portal		
Related Requirements		PT-EXA-T-001, PT-EXA-T-002, PT-EXA-T-003, PT-EXA-T-004, PT-EXA-T-005, PT-EXA-T-008, PT-EXA-T-009, PT-EXA-T-010, PT-EXA-T-011, PT-EXA-T-012, PT-EXA-T-013, PT-EXA-T-015		
Tools Used		• RAWFIE Web Portal • RAWFIE Visual Editor		
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Access to the Visual Editor through the RAWFIE Web Portal	Redirection to the Visual Editor interface	Success	
2	Access the available toolbar	Specific windows are presented	Success	
3	Create an experiment by utilizing the available tools	The experimenter can define waypoints and experiment information by clicking and designing in the visual editor	Success	
4	Define erroneous commands	The authoring tool responds with error messages and indication for correcting the error	Success	
5	Save the experiment	The experiment is stored in the database and specific files are produced to be adopted by the remaining RAWFIE components	Success	

Table 65: Verification test of the in-Visual Editor Experiments Update

Test ID: **EAT04**		Conducted by**: UoA**	Date**: April 2017**	Test Category**: Verification Tests (front end tier – middle tier)**
Hardware Configuration		-		
Software Configuration				
Test Name:		***Update Experiments in the Visual Editor***		
Preconditions		• User entered in the RAWFIE Portal		
Related Requirements		PT-EXA-T-001, PT-EXA-T-002, PT-EXA-T-003, PT-EXA-T-004, PT-EXA-T-005, PT-EXA-T-008, PT-EXA-T-009, PT-EXA-T-010, PT-EXA-T-011, PT-EXA-T-012, PT-EXA-T-013, PT-EXA-T-015		
Tools Used		• RAWFIE Web Portal • RAWFIE Visual Editor		
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Access to the Visual Editor through the RAWFIE Web Portal	Redirection to the Visual Editor interface	Success	
2	Open an already defined experiment	Experiment is presented in the editor	Success	
3	Makes changes in the experiment workflow	The experiment is updated	Success	
4	Save the experiment	The experiment is stored in the database and specific files are produced to be adopted by the remaining RAWFIE components	Success	The experiment was correclty stored

Table 66: Verification test of the Editor switching

Test ID: **EAT05**		Conducted by**: UoA**	Date**: April 2017**	Test Category**: Verification Tests (front end tier – middle tier)**
Hardware Configuration		-		
Software Configuration		•		
Test Name:		***Switch between the Editors***		
Preconditions		• User entered in the RAWFIE Portal		
Related Requirements		PT-EXA-T-001, PT-EXA-T-002, PT-EXA-T-003, PT-EXA-T-004, PT-EXA-T-005, PT-EXA-T-008, PT-EXA-T-009, PT-EXA-T-010, PT-EXA-T-011, PT-EXA-T-012, PT-EXA-T-013, PT-EXA-T-015		
Tools Used		• RAWFIE Web Portal • RAWFIE Textual Editor • RAWFIE Visual Editor		
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Access to the editors through the RAWFIE Web Portal	Redirection to the editor interface	Success	
2	Create an experiment	Experiment is presented in the editor interface	Success	
3	Switch to the alternative editor and make changes	The experiment is updated	Success	The synchronization is performed automaticaly while both editors are visible in the portal
4	Save the experiment	The experiment is	Success	

		stored in the database and specific files are produced to be adopted by the remaining RAWFIE components		

Table 67: Verification test of the experiment Launchings

Test ID: **EAT06**	Conducted by**: UoA**	Date**: April 2017**	Test Category**: Verification Tests (front end tier – middle tier)**
Hardware Configuration	-		
Software Configuration	•		
Test Name:	***Launch experiments***		
Preconditions	• User entered in the RAWFIE Portal		
Related Requirements	PT-EXA-T-001, PT-EXA-T-002, PT-EXA-T-003, PT-EXA-T-004, PT-EXA-T-005, PT-EXA-T-008, PT-EXA-T-009, PT-EXA-T-010, PT-EXA-T-011, PT-EXA-T-012, PT-EXA-T-013, PT-EXA-T-015		
Tools Used	• RAWFIE Web Portal • RAWFIE Textual - Visual Editors • RAWFIE Launching Tool		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Access to the authoring tool through the RAWFIE Web Portal	Redirection to the editor interface	Success	
2	Select an experiment	A drop-down list of the available experiments is appeared and the experimenter has the opportunity to select one	Success	
3	Start the experiment execution	The launching service is informed with the experiment ID and the execution starts	Success	The launching service was correctly informed with the experiment information

Table 68: Verification test of the experiment Launchings

Test ID: **EAT07**	Conducted by**: UoA**	Date**: April 2017**	Test Category**: Verification Tests (front end tier – middle tier)**
Hardware Configuration	-		
Software Configuration	•		
Test Name:	***Launch (scheduled) experiments***		
Preconditions	• User entered in the RAWFIE Portal		
Related Requirements	PT-EXA-T-001, PT-EXA-T-002, PT-EXA-T-003, PT-EXA-T-004, PT-EXA-T-005, PT-EXA-T-008, PT-EXA-T-009, PT-EXA-T-010, PT-EXA-T-011, PT-EXA-T-012, PT-EXA-T-013, PT-EXA-T-015		
Tools Used	• RAWFIE Web Portal • RAWFIE Textual - Visual Editors • RAWFIE Launching Tool		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Access to the authoring tool through the	Redirection to the	Success	

	RAWFIE Web Portal	editor interface		
2	Select the scheduled launching tool	A drop-down list of the available experiments is appeared and the experimenter has the opportunity to select one	Success	
3	Define the experiment execution	The launching service is informed with the experiment ID and the execution is planned	Success	The launching service was correctly informed with the experiment information

2.6.2.3 *Users & Rights Service*

Table 69: Verification test of the Users & Rights Service login checking

Test ID: **URS01**	Conducted by: **Fraunhofer**		Date: **May 2018**	Test Category: **Verification Tests (middle tier)**
Hardware Configuration	See section 2.4			
Software Configuration	See section 2.4			
Test Name:	***Login checking***			
Preconditions	• Valid user name and password known			
Related Requirements	• PT-USR-S-001			
Tools Used	• SOAPUI REST client			
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	invalid user name and password sent to the Users & Rights Service	Users & Rights Service returns failure	Success	
2	valid user name and password sent to the Users & Rights Service	Users & Rights Service returns OK	Success	

Table 70: Verification test of the Users & Rights Service roles/rights checking

Test ID: **URS02**	Conducted by: **Fraunhofer**		Date: **May 2018**	Test Category: **Verification Tests (middle tier)**
Hardware Configuration				
Software Configuration				
Test Name:	***Roles/rights checking***			
Preconditions	• Valid user rights known			
Related Requirements	PT-USR-S-002			
Tools Used	• SOAPUI REST client			
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	user ID and available required rights sent to the Users & Rights Service	Users & Rights Service return true	Success	
2	user ID and not available required rights sent to the Users & Rights Service	Users & Rights Service return false	Success	

Table 71: Verification test of the user rights checks

Test ID: **URS03**		Conducted by: **Fraunhofer**	Date**: May 2018**	Test Category**: Verification Tests (middle tier)**
Hardware Configuration				
Software Configuration				
Test Name:		***Adding and editing user data***		
Preconditions		• New user does not exist		
Related Requirements		PT-USR-S-002		
Tools Used		• SOAPUI REST client		
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	New user data (personal data and roles) sent to the Users & Rights Service	Users & Rights Service creates the new user and returns true	Success	
2	Request user data of new user	Users & Rights Service return the data. It should be equal to the data of step 1	Success	
3	Edited user data (personal data and roles) sent to the Users & Rights Service	Users & Rights Service saves the user data and returns true	Success	
4	Request user data of the user	Users & Rights Service return the data. It should be equal to the data of step 3	Success	

2.6.2.4 Booking Service

The Booking Service is tightly coupled with the Booking Tool component. Therefore, the verification tests described for the Booking Tool should also be considered during Booking Service functionality verification activities. Verification tests of the component focus around testing and ensuring the correctness of each provided method.

All Test Procedures BS01, BS02, BS03, BS04, BS05, BS06, BS07, BS08 remain unchanged compared to what was defined in the previous version of the deliverable (D6.3). However due to regression testing, test procedures were rerun on June 2018.

Table 72: Verification test of Booking Service add reservation functionality

Test ID: **BS01**	Conducted by: **HAI**	Date: **June 2018**	Test Category: **Verification Tests (middle tier)**
Hardware Configuration	See section 2.4		
Software Configuration	See section 2.4		
Test Name:	***Booking Service add reservation functionality***		
Preconditions	• Master DB is prepopulated with reservations of different status and timeslots (involved tables are: Reservation, Resource Reservation) • User initiating the call is a valid experimenter		
Related Requirements	PT-BOO-S-001 (user level booking), PT-BOO-S-002, PT-BOO-S-004 PT-BOO-S-005, PT-BOO-S-007, PT-BOO-S-012		
Tools Used	Maven, Java test client, HttpRequestor Firefox plugin Booking Tool UI		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Call addReservation() providing a datetime interval that has passed	response should be returned with a proper failure message	Success	
2	Call addReservation() providing a datetime interval in the future (NO conflict in requested resources with existing reservation at the same time)	Appropriate MasterDB tables are updated (new reservation in status=PENDING)	Success	
		If email sending is enabled then email is send to both the creator and the testbed operator of the reserved resources	Success	
		The returned response contains the newly created reservationId and the reservation status	Success	
3	Call addReservation() providing a datetime interval in the future conflict in requested resources with existing reservation at the same time)	response should be returned with a proper failure message	Success	

Table 73: Verification test of Booking Service edit reservation functionality

Test ID: **BS02**	Conducted by: **HAI**	Date: **June 2018**	Test Category: **Verification Tests (middle tier)**
Hardware Configuration	See section 2.4		
Software Configuration	See section 2.4		
Test Name:	***Booking Service edit reservation functionality***		
Preconditions	• Master DB is prepopulated with reservations of different status and timeslots (involved tables are: Reservation, Resource_Reservation) • User initiating the call is a valid experimenter		
Related Requirements	PT-BOO-S-002, PT-BOO-S-005, PT-BOO-S-007, PT-BOO-S-013		
Tools Used	Booking Tool UI		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Call editReservation() providing appropriate ReservationData which should include the reservationId (the call should include credentials about the user initiating it)	If provided user credentials do not match with the ones of the reservation owner then a proper failure message is returned	Success	
		If existing reservation status!= PENDING then no update should be possible and a proper failure message is returned	Success	
		If time related changes refer to an interval in the past then a proper failure message is returned	Success	
	(If status= PENDING & user credential match)	If overlaps with existing reservation are introduced and resources conflicts are detected then a proper failure message is returned	Success	
	(If status= PENDING & user credential match)	If no resources conflicts are detected the changes are accepted and the corresponding DB tables updated	Success	
2	Repeat step 1 with different kind of changes related to timeslots and resource selection	Ensure that expected results are respected as described in step 1	Success	Success of reservation edit depends on whether overlaps introduce conflicts according to the steps described in step 1

Table 74: Verification test of Booking Service approve reservation functionality

Test ID: **BS03**	Conducted by: **HAI**	Date: **June 2018**	Test Category: **Verification Tests (middle tier)**
Hardware Configuration	See section 2.4		
Software Configuration	See section 2.4		
Test Name:	***Booking Service approve reservation functionality***		
Preconditions	• Master DB is prepopulated with reservations of different status and timeslots (involved tables are: Reservation, Resource_Reservation)		
Related Requirements	PT-BOO-S-002, PT-BOO-S-005, PT-BOO-S-007, PT-BOO-S-013, PT-NF-002		
Tools Used	Maven, Java test client, HttpRequestor Firefox plugin Booking Tool UI		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Call approveReservation() (the call should include credentials about the user initiating it)	If provided credentials do not match with an authorized platform user then a proper failure message is returned	Success	
		If provided credentials do not refer to an authorized platform user with role=TESTBED_OP then a proper failure message is returned	Success	
		If reservationId refers to a reservation with status !=PENDING then a proper failure message is returned	Success	
		If reservationId refers to a past reservation then then a proper failure message is returned	Success	
		If conflicts are detected with any other APPROVED reservation then then a proper failure message is returned	Success	
2	(If status= PENDING & caller=TESTBED_OP & no conflicts detected	Status change is accepted and corresponding DB tables updated	Success	
		An email is send to the owner of the reservation	Success	
		A ReservationStatusMsg is send to Message bus	Success	

Table 75: Verification test of Booking Service reject reservation functionality

Test ID: **BS04**		Conducted by: **HAI**		Date: **June 2018**	Test Category: **Verification Tests (middle tier)**
Hardware Configuration		See section 2.4			
Software Configuration		See section 2.4			
Test Name:		***Booking Service reject reservation functionality***			
Preconditions		• Master DB is prepopulated with reservations of different status and timeslots (involved tables are: Reservation, Resource_Reservation)			
Related Requirements		PT-BOO-S-002, PT-BOO-S-005, PT-BOO-S-007, PT-BOO-S-013, PT-NF-002			
Tools Used		Maven, Java test client, HttpRequestor Firefox plugin Booking Tool UI			
Step	**Action**		**Expected Result**	**Status**	**Remarks**
1	Call approveReservation() (the call should include credentials about the user initiating it)		If provided credentials do not match with an authorized platform user then a proper failure message is returned	Success	
			If provided credentials do not refer to an authorized platform user with role=TESTBED_OP then a proper failure message is returned	Success	
			If reservationId refers to a reservation with status !=PENDING or APPROVED then a proper failure message is returned	Success	
			If reservationId refers to a past reservation then then a proper failure message is returned	Success	
2	(If status= PENDING & caller=TESTBED_OP		Status change is accepted and corresponding DB tables updated	Success	
			An email is send to the owner of the reservation	Success	
			A ReservationStatusMsg is send to Message bus	Success	

Table 76: Verification test of Booking Service delete reservation functionality

Test ID: **BS05**	Conducted by: **HAI**	Date: **June 2018**	Test Category: **Verification Tests (middle tier)**
Hardware Configuration	See section 2.4		
Software Configuration	See section 2.4		
Test Name:	***Booking Service delete reservation functionality***		
Preconditions	• Master DB is prepopulated with reservations of different status and timeslots (involved tables are: Reservation, Resource_Reservation)		
Related Requirements	PT-BOO-S-002, PT-BOO-S-005, PT-BOO-S-007, PT-NF-002		
Tools Used	Maven, Java test client, HttpRequestor Firefox plugin Booking Tool UI		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Call deleteReservation() (the call should include credentials about the user initiating it)	If provided credentials do not match with an authorized platform user then a proper failure message is returned	Success	
		If reservationId refers to a past reservation then a proper failure message is returned	Success	
		If reservationId refers to a reservation with resources involved in a currently running experiment a proper failure message is returned	Success	
		If none of the above then status change to CANCELLED	Success	

Table 77: Verification test of Booking Service retrieve reservation(s) functionality

Test ID: **BS06**	Conducted by: **HAI**	Date: **June 2018**	Test Category: **Verification Tests (middle tier)**
Hardware Configuration	See section 2.4		
Software Configuration	See section 2.4		
Test Name:	***Booking Service retrieve reservation(s) functionality***		
Preconditions	• Master DB is prepopulated with reservations of different status and timeslots (involved tables are: Reservation, Resource_Reservation)		
Related Requirements	PT-BOO-S-002, PT-BOO-S-008		
Tools Used	HttpRequestor Firefox plugin		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Call getReservation() providing a reservationId	Inspect response and ensure data is inline with the information stored in the MasterDB	Success	
2	Call getReservations() providing appropriate search criteria (time, user etc.)	Inspect response and ensure data is in line with the information stored in the MasterDB	Success	

Table 78: Verification test of Booking Service check for conflicts functionality

Test ID: **BS07**		Conducted by**: HAI**	Date**: June 2018**	Test Category**: Verification Tests (middle tier)**
Hardware Configuration		See section 2.4		
Software Configuration		See section 2.4		
Test Name:		***Booking Service check for conflicts functionality***		
Preconditions		• Master DB is prepopulated with reservations of different status and timeslots (involved tables are: Reservation, Resource_Reservation)		
Related Requirements		PT-BOO-S-002, PT-BOO-S-006, PT-BOO-S-012		
Tools Used		HttpRequestor Firefox plugin		
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Call checkForConflictingReservations() providing proper reservation data info	Returns true or false depending on whether resource conflicts are detected for time overlapping with pre-existing in the MasterDB reservations	Success	
2	Call getReservations() providing appropriate search criteria (time, user etc.)	Inspect response and ensure data is in line with the information stored in the MasterDB	Success	

Table 79: Verification test of Booking Service simultaneous reservations support

Test ID: **BS08**		Conducted by**: HAI**	Date**: June 2018**	Test Category**: Verification Tests (middle tier)**
Hardware Configuration		See section 2.4		
Software Configuration		See section 2.4		
Test Name:		***Booking Service simultaneous reservations support***		
Preconditions		• Master DB is prepopulated with reservations of different status and timeslots (involved tables are: Reservation, Resource_Reservation)		
Related Requirements		PT-BOO-S-002, PT-BOO-S-010		
Tools Used		soapUI		
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Multiple calls of Booking Service addReservation() method (execute BS01 multiple times simultaneously from different clients)	Ensure that all requests are processed and multiple reservations are created in the MasterDB	Success	

2.6.2.5 Launching Service

- All Test Procedures LS01, LS02, LS03, LS04 did not change since the previous version of the deliverable (D6.3). However due to regression testing, test procedures were rerun on June 2018.

Table 80: Verification test of the Launching Service manual start (short term launching)

Test ID**: LS01**	Conducted by**: HAI**	Date**: June 2018**	Test Category**:** **Verification Tests (middle tier)**
Hardware Configuration	See section 2.4		
Software Configuration	See section 2.4		
Test Name:	***Experiment short term launching***		
Preconditions	• Requires the Message Bus and the experiment controller to be accessible. • The master data repository should contain reservations for the user and for a defined experiment (involved tables are Experiment Experiment_Execution., Reservation, Reservation_item)		
Related Requirements	PT-LAU-S-001, PT-LAU-S-003, PT-LAU-S-004, PT-LAU-S-005, PT-LAU-S-007 PT-LAU-S-008, PT-LAU-S-009 (by design), PT-LAU-S-012, PT-LAU-S-013 (by design), PT-LAU-S-015		
Tools Used	Experiment Authoring Tool UI Maven, Java test client, HttpRequestor Firefox plugin		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	User call manualStart() providing an experiment Id	if experimentId is not present in the MasterDB then a proper failure message is returned	Success	
		If supplied user credentials do not match an authorized user then a proper failure message is returned	Success	
		If supplied user credentials match an authorized user but refer to booked resources of another user then a proper failure message is returned	Success	
2	(case experimentId exists)	if an executionId already exists and refers to a running experiment (status=Ongoing) then a proper failure message is returned	Success	
3	(case no executionId exists or exists for an status!=Ongoing)	Launching service generates an ExperimentStartRequest to the Message Bus (targeting the Experiment Controller).	Success	
		Master DB tables are properly updated (tables Experiment_Execution, Reservation_item)	Success	
		LaunchingServiceActionResp json message is returned containing the generated executionId and the status of the experiment	Success	

Table 81: Verification test of the Launching Service schedule (long term launching)

Test ID**: LS02**	Conducted by**: HAI**	Date**: June 2018**	Test Category**:** **Verification Tests (middle tier)**
Hardware Configuration	See section 2.4		
Software Configuration	See section 2.4		

Test Name:	***Experiment long term launching***
Preconditions	• Requires the Message Bus and the experiment controller to be accessible. • The master data repository should contain reservations for the user and for a defined experiment (involved tables are Experiment Experiment_Execution., Reservation, Reservation_item) • The platform launching scheduler must be running
Related Requirements	PT-LAU-S-002, PT-LAU-S-003, PT-LAU-S-004, PT-LAU-S-005, PT-LAU-S-007 PT-LAU-S-008, PT-LAU-S-009 (by design), PT-LAU-S-011, PT-LAU-S-012 PT-LAU-S-013 (by design), PT-LAU-S-014, PT-LAU-S-015
Tools Used	Maven, Java test client, HttpRequestor Firefox plugin

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	User call schedule() providing experimentId, startDate, endDate	if experimentId is not present in the MasterDB then a proper failure message is returned	Success	
		If supplied user credentials do not match an authorized user then a proper failure message is returned	Success	
		If supplied user credentials match an authorized user but refer to booked resources of another user then a proper failure message is returned	Success	
		If startDate or, endDate refer to past time then a proper failure message is returned	Success	
		If startDate or endDate are not contained within the timeslot defined for the associated reservation then a proper failure message is returned	Success	
		if an executionId already exists and refers to a running experiment (status=Ongoing) then a proper failure message is returned	Success	
2	Scheduling part (case all preconditions are met)	Launching Scheduler is called and a job is added to be launched at the specified startDate	Success	
		The user (owner) of the experiment and the testbed operator are informed by an appropriate notification (email)	Success	
		Master DB tables are properly updated (tables Experiment_Execution, Reservation_item). The status of the experiment should be BOOKED	Success	
		LaunchingServiceActionResp json message is returned containing the generated executionId and the status of the experiment	Success	
3	Execution part (check Launching Service activity when startDate arrives)	Master DB tables are properly updated (tables Experiment_Execution, Reservation_item)The status of the experiment changes to ONGOING	Success	
		Launching service generates an ExperimentStartRequest to the Message Bus (targeting the Experiment Controller).	Success	
		Scheduled job (for the executionId) is removed from scheduler	Success	

Table 82: Verification test of the Launching Service cancellation request

Test ID: **LS03**	Conducted by: **HAI**		Date: **June 2018**	Test Category: **Verification Tests (middle tier)**
Hardware Configuration	See section 2.4			
Software Configuration	See section 2.4			
Test Name:	***Experiment cancellation request***			
Preconditions	• Requires the Message Bus and the experiment controller to be accessible. • The master data repository should contain reservations for the user and for a defined experiment (involved tables are Experiment Experiment_Execution., Reservation, Reservation_item) • An experiment should be schedule for a future time			
Related Requirements	PT-LAU-S-009 (by design), PT-LAU-S-010, PT-LAU-S-012, PT-LAU-S-013 (by design)			
Tools Used	Maven, Java test client, HttpRequestor Firefox plugin			

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	User call cancellation() providing an executionId	if executionId is not present in the MasterDB then a proper failure message is returned	Success	
		If supplied user credentials do not match an authorized user then a proper failure message is returned	Success	
		If supplied user credentials match an authorized user but refer to an experiment of another experimenter then a proper failure message is returned (Exception to this rule if credentials refer to a testbed operator or administrator)	Success	
2	(case executionId exists)	If the experiment is already running (status= ONGOING) then cancellation is not possible and a proper failure message is returned	Success	
		If no schedule job is found in Launching scheduler then a proper failure message is returned	Success	
3	(executionId exists and the execution is still in the scheduler)	Job is removed from the scheduler	Success	
		Master DB tables are properly updated (tables Experiment_Execution, Reservation_item). The status of the experiment changes to CANCELLED	Success	
		LaunchingServiceActionResp json message is returned containing with the executionId, status= CANCELLED and empty message field	Success	
		The user (owner) of the experiment and the testbed operator are informed by an appropriate notification (email)	Success	

Table 83: Verification test of Launching Service simultaneous launching capability

Test ID: **LS04**		Conducted by: **HAI**	Date: **June 2018**	Test Category: **Verification Tests (middle tier)**
Hardware Configuration		See section 2.4		
Software Configuration		See section 2.4		
Test Name:		***Launching Service simultaneous launching capability***		
Preconditions		• Master DB is prepopulated with reservations of different status and timeslots (involved tables are: Reservation, Resource_Reservation)		
Related Requirements		PT-LAU-S-006, PT-LAU-S-011		
Tools Used		soapUI		
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Multiple calls of Launching Service schedule() method (execute LS01 multiple times simultaneously from different clients)	Ensure that all requests are processed multiple experiments executions exist in the Job Scheduler	Success	

2.6.2.6 *Visualisation Engine*

Table 84: Visualisation engine user request handling

Test ID: **VE01**		Conducted by: **Aberon**	Date: **April 2017**	Test Category: **Verification Tests (middle tier)**
Hardware Configuration		See section 2.4		
Software Configuration		See section 2.4		
Test Name:		***User request handling***		
Preconditions		• Requires visualization tool and visualization engine to function and be accessible		
Related Requirements		VIS01, VIS02		
Tools Used				
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Visualization engine receive through websocket request from visualization tool	The visualization engine handles the request	Success	
2	Visualization engine sends through websocket the response	Visualization tool receives response	Success	

Table 85: Visualization engine geospatial data modification

Test ID: **VE02**		Conducted by: **Aberon**	Date: **April 2017**	Test Category: **Verification Tests (middle tier)**
Hardware Configuration		See section 2.4		
Software Configuration		See section 2.4		
Test Name:		***Geospatial data modification test***		
Preconditions		• Requires visualization tool and visualization engine to function and be accessible		
Related Requirements		VIS01,VIS02		
Tools Used				
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Visualization engine receive through the message bus	The visualization engine handles the request	Success	
2	Visualization engine update data in database	Data is properly stored in the database for future retrieval	Success	

Table 86: Visualization engine camera interaction

Test ID: **VE03**		Conducted by: **Aberon**	Date:	Test Category: **Verification Tests (middle tier)**
Hardware Configuration				
Software Configuration				
Test Name:		***Geo Data Test***		
Preconditions		• Requires visualization tool and visualization engine to function and be accessible		
Related Requirements		VIS01,VIS02		
Tools Used				
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	visualisation engine receives through the message bus data from the visualisation tool	The visualization engine handles the request	Success	
2	Visualization engine updates data in database	Data is properly stored in the database for future retrieval	Success	

Table 58: Visualization engine indoor map handling

Test ID: **VE04**	Conducted by**: Aberon**	Date**:**	Test Category**: Verification Tests (middle tier)**
Hardware Configuration			
Software Configuration			
Test Name:	***Indoor map test***		
Preconditions	● Requires visualization tool and visualization engine to function and be accessible and an indoor map to be loaded in the GeoServer and experiment using indoor map		
Related Requirements	VIS01, VIS02		
Tools Used			

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	visualisation engine receives a request from the visualisation tool to start an experiment that needs indoor map	the visualisation engine loads needed data from the db	Success	
2	Visualization engine receives data from an UxV	Visualisation engine updates this data and forwards it to the VE	Success	

2.6.2.7 *Data Analysis Engine*

Table 58: Verification test of accepting analysis tasks defined through the Data Analysis Tool

Test ID**: DAE01**	Conducted by**: HESSO**	Date**:**	Test Category**: Verification Tests (front end)**
Hardware Configuration			
Software Configuration			
Test Name:	***Accept analysis tasks defined through the Data Analysis Tool***		
Preconditions	● Requires the Zeppelin notebook interface of the DAT to be functioning and accessible ● Requires result repository to be functioning and accessible		
Related Requirements	PT-DAA-S-001, PT-DAA-S-002		
Tools Used	●		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Authorized user logs into the web portal and clicks on the Zeppelin notebook GUI tab of the Data Analysis Tool GUI embedded into the web portal	Login successful, successfully reaches the Zeppelin notebook GUI tab of the Data Analysis Tool GUI embedded into the web portal	Success	
2	User designs a spectrum of data analysis tasks in the notebook relying on various interpreters (e.g. Spark, Python, etc.). For a given task, the user starts it in its respective notebook. A tasks can be defined using predefined built-in algorithms or via procedures that the user would have designed from scratch within the interface.	The task has been successfully started (statement for a given task). The results (again, for a given task) are visible through the Grafana dashboard.	Success	

Table 58: Verification test of scales properly to the addition of workers

Test ID: **DAE02**	Conducted by**: HESSO**	Date**:**	Test Category**: Verification Tests (front end)**
Hardware Configuration			
Software Configuration			
Test Name:	***Scales properly to the addition of workers***		
Preconditions	● Requires the Zeppelin notebook interface of the DAT to be functioning and accessible ● Requires result repository to be functioning and accessible		
Related Requirements	PT-DAA-S-003, PT-DAA-S-004, PT-DAA-T-004		
Tools Used	●		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Administrator designs and starts an analysis task via the Data Analysis Tool Zeppelin notebook GUI (see DAE01) under a given cluster configuration.	The task has been successfully started, results are visible on the Grafana dashboard (for streaming tasks).	Success	
2	Administrator stops running task.	The task has been successfully stopped.	Success	
3	Administrator increases the number of workers in the Spark cluster and launches the same task.	The task has been successfully started, results are visible on the Grafana dashboard (for streaming tasks). The results are similar to the results of the previous task.	Success	

2.6.2.8 System Monitoring Service

Table 87: Verification test of the System Monitoring

Test ID: **SYMS01**		Conducted by: **Fraunhofer**	Date: **May 2018**	Test Category: **Verification Tests (middle tier)**
Hardware Configuration		See section 2.4		
Software Configuration		See section 2.4		
Test Name:		***System Monitoring***		
Preconditions		•		
Related Requirements		PT-SYM-S-001, PT-SYM-S-002, PT-SYM-S-006		
Tools Used		Browser		
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Service polls the computes of the middle tier for their status	Computes return their health status to the service	Success	
2	Service listen to status messages on the message bus	Testbed component sent automatically status information on the message bus. Messages received by the service	Success	
3	System Monitory Tool request status information	Service collects the information and returns it	Success	

Table 88: Verification test of the System Monitoring Problem Notifications

Test ID: **SYMS02**	Conducted by: **Fraunhofer**	Date**: May 2018**	Test Category**: Verification Tests (middle tier)**
Hardware Configuration			
Software Configuration			
Test Name:	***System Monitoring Problem Notifications***		
Preconditions	• Notification receivers are configured ○ Administrators ○ User register for notifications if certain components are down • Status information is collected • connection System Monitoring Service and Tool • administrative knowledge about the system state needed on user side (to check results) • administrative access to a server to shutdown a server		
Related Requirements	PT-SYM-T-001, PT-SYM-S-002, PT-SYM-S-003, PT-SYM-S-004, PT-SYM-S-008		
Tools Used	• Email client • Browser • SSH client		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	User shuts down a server of RAWFIE	Error notifications (e.g. email) should be sent by the System Monitoring Service to the administrators and other registered users	Success	
	user opens System Monitoring Tool in the Web Portal	the System Monitoring Tool request the data from the Service and displays the server in critical state	Success	
	User restarts the server of RAWFIE	A recovery notification (e.g. email) should be sent by the System Monitoring Service to the administrators	Success	
	user opens System Monitoring Tool in the Web Portal	the System Monitoring Tool request the data from the Service and displays the server in OK state	Success	

Table 89: Verification test of sending notification on planned downtime

Test ID: **SYMS03**		Conducted by: **Fraunhofer**	Date: **May 2018**	Test Category: **Verification Tests (middle tier)**
Hardware Configuration				
Software Configuration				
Test Name:		***Sending notification on planned downtime***		
Preconditions		• Notification receivers are configured		
Related Requirements		PT-SYM-S-005		
Tools Used		• Browser • Email client		
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	User marks a service with downtime start	A notification (e.g. email) should be sent by the System Monitoring Service to the administrators.	Success	
1	User marks a service with downtime end	A notification (e.g. email) should be sent by the System Monitoring Service to the administrators.	Success	

2.6.2.9 *Accounting Service*

Table 90: Verification test of the accounting data collection

Test ID: **ACCS01**	Conducted by: **Fraunhofer**	Date: **May 2018**	Test Category: **Verification Tests (middle tier)**
Hardware Configuration			
Software Configuration			
Test Name:	***Accounting data collection***		
Preconditions	• Accounting data is empty for the used user • Experimenter 1 and experimenter 2 have different active cost models		
Related Requirements	PT-ACC-S-001, PT-ACC-S-002, PT-ACC-S-003, PT-ACC-S-004, PT-ACC-S-005, PT-ACC-S-006		
Tools Used	Browser		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Experiment of experimenter 1 is completed. Notifications sent on the message bus.	Accounting received the event and computes the charge for the experiments based on the active cost model of experimenter 1	Success	Done via database triggers and periodical database checks, not via message bus
2	Experiment of experimenter 2 is completed. Notifications sent on the message bus.	Accounting received the event and computes the charge for the experiments based on the active cost model of experimenter 2 (should be different to experimenter 1)	Success	Done via database triggers and periodical database checks, not via message bus
3	Billing period ends	Bill is created and sent to the both experimenters	Success	

Table 91: Verification test of the account charging

Test ID: **ACCS02**	Conducted by: **Fraunhofer**	Date: **May 2018**	Test Category: **Verification Tests (middle tier)**
Hardware Configuration			
Software Configuration			
Test Name:	***Account charging***		
Preconditions	• User has an external payment system account		
Related Requirements			
Tools Used			

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	User opens the user profile page in the Web Portal and klicks on account charging. .	Redirect to payment system selection and the to the external payment system	Not tested	No payment system connected. Account charging can be done by a user with billing manager role manually via the Web Portal (Accounting Tool)
2	User executes the payment	Payment is added to the account balance	Not tested	

2.6.2.10 Experiment Controller

The Experiment Controller component requirement not addressed by the tests specified below is

- PT-EXP-C-001 “Cancellation of running experiments should be possible”.

Justification:
The cancellation of an ongoing experiment is possible through direct communication between Experiment Monitoring Tool (see 2.6.1.6 paragraph) and the Resource Controller.

Table 92 Verification test of experiment forwarding

Test ID**: EC01**	Conducted by**: CERTH**	Date: **September 2018**	Test Category**: Verification Tests (middle tier)**	
Hardware Configuration	-			
Software Configuration	-			
Test Name:	***Forward experiment details to the corresponding testbed***			
Preconditions	• Requires the Message Bus to be accessible • Requires the corresponding instance Resource Controller to be up and running • Requires the entries on the corresponding tables in the Master DB to be appropriately filled in.			
Related Requirements	PT-EXP-C-002			
Tools Used				
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Send an ExperimentLaunchRequest type of message	Experiment Controller properly consumes the message.	Success	
		Interaction with the Master DB to retrieve all the required information. During this procedure, the following fields are properly retrieved: • EDL script • Vehicles canonical names • Partitions IDs of all the involved vehicles • Coordinate system	Success	Direct access to Master Data Repository (PostgreSQL database) , not via message bus
		An ExperimentStartRequest type of message is dispatched to the Kafka message bus.	Success	

Table 93 Verification test of handling status updates of a running experiment

Test ID: **EC02**	Conducted by: **CERTH**	Date: **September 2018**	Test Category: **Verification Tests (middle tier)**	
Hardware Configuration	-			
Software Configuration	-			
Test Name:	***Status updates of a running experiment***			
Preconditions	• Requires the Message Bus to be accessible • Requires the corresponding instance Resource Controller to be up and running • Requires the entries on the corresponding tables in the Master DB to be appropriately filled in.			
Related Requirements	PT-EXP-C-006, PT-EXP-C-007, PT-EXP-C-008, PT-EXP-C-009			
Tools Used				
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Send an ExperimentStatusMsgtype type of message regarding a running experiment	Experiment Controller properly consumes the message and updates the following tables inside Master DB: • experimentlog • experiment_execution • experiment	Success	

Table 94 Verification test of supporting experiments execution in multiple testbeds

Test ID: **EC03**	Conducted by: **CERTH**		Date:	Test Category: **Verification Tests (middle tier)**
Hardware Configuration	-			
Software Configuration	-			
Test Name:	***Support execution of experiments in multiple testbeds – Parallel execution***			
Preconditions	• Requires the Message Bus to be accessible • Requires the corresponding instance Resource Controller to be up and running • Requires the entries on the corresponding tables in the Master DB to be appropriately filled in. • Requires that multiple testbeds are connected to the RAWFIE platform			
1Related Requirements	PT-EXP-C-003, PT-EXP-C-004			
Tools Used				
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Send an ExperimentLaunchRequest type of message for testbed A	Experiment Controller properly consumes the message and dispatch an ExperimentStartRequest type of message.	Success	
		An instance of the Resource Controller, launched for testbed A, successfully receives the requested experiment.	Success	
2	Send an ExperimentLaunchRequest type of message for testbed B	While the first experiment is executed, Experiment Controller properly consumes the new message and dispatch an ExperimentStartRequest type of message.	Success	
		An instance of the Resource Controller, launched for testbed B, successfully receives the requested experiment.	Success	
3	Update Master DB with information coming from both the running experiments	During the execution of all the experiments, Experiment Controller receives distinct status messages of each experiment and properly updates the corresponding fields inside the Master DB.	Success	

2.6.3 Testbed Tier

This section presents the test of the Testbeds and Resources control components.

2.6.3.1 Monitoring Manager

Monitoring Manager is tightly coupled with Testbed Manager coexisting in the same application running at testbed level enabling the user to have a close look at computing and UxV resources utilization.

The Monitoring Manager component requirement not addressed by the tests specified below is

- TB-MOM-005: Testing of this requirement presumes that other services with well-defined interfaces like Weather conditions service are available to make verification procedures feasible.

Test procedure MM01 is an updated version of that defined in D6.3 with extra steps added. Test procedure MM02 is almost identical to Test Manager's procedure TM03 in D6.3 which has been moved to Monitoring Manager section for better cohesion of monitoring activities.

Table 95: Verification test of UxV health status

Test ID**: MM01**		Conducted by**: HAI**	Date**: May 2018**	Test Category**: Verification Tests (middle tier)**
Hardware Configuration				
Software Configuration				
Test Name:		***Check UxV health status***		
Preconditions		• Requires the Message Bus to be accessible • Requires the network controller to be accessible. • Requires the System Monitoring Service to be accessible • Initial UxV status configuration: ○ Fuel usage WARNING < 50%, CRITICAL < 15% ○ CPU usage WARNING > 50%, CRITICAL > 85% ○ Storage usage WARNING > 50%, CRITICAL > 85%		
Related Requirements		TB-MOM-001, TB-MOM-003, TB-MOM004, PT-SYM-S-002, UXV-NOD-001, TB-UVG-001		
Tools Used				
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Monitoring Manager receives periodically messages from UxVs related to resources utilization (FuelUsage, CpuUsage, Storage Usage) from the message bus.	Monitoring Manager properly consumes the messages and displays the result in Monitoring Manager's User Interface	Success	Network Controller is not required
2	Monitoring Manager calculates an overall UxV status upon predefined criteria for the above received messages	UxV status is displayed in Monitoring Manager's User Interface	Success	
3	Monitoring Manager periodically transmits a message describing the UxV Status to the Message Bus	System Monitoring Service receives and displays the current status for each UxV	Success	

Table 96: Verification test of testbed health status

Test ID: **MM02**		Conducted by: **HAI**	Date: **May 2018**	Test Category: **Verification Tests (Testbed tier)**
Hardware Configuration Details				
Software Configuration Details				
Test Name:		***Check Testbed health status***		
Preconditions		• Requires middle tier to be accessible (System Monitoring Service) • Initial Testbed health status configuration: ○ CPU usage WARNING > 50%, CRITICAL >85% ○ Memory usage WARNING > 50%, CRITICAL >85% ○ Disk usage WARNING > 50%, CRITICAL >85% ○ Frequency of sending messages 30 sec		
Related Requirements		TB-MOM-002, TB-MOM-003, TB-MOM004, PT-SYM-S-002		
Tools Used				
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Monitoring Manager started	1. Monitoring manager successfully initialized 2. Monitoring Manager checks periodically CPU load, memory and disk usage	Success	
2	Monitoring manager processing (status assessment)	3. A TestbedHealthStatus message is created containing an overall assessment (OK, WARNING, CRITICAL) for the usage metrics monitored 4. The message is sent to the Message bus	Success	
3	Check System Monitoring Service UI display at Middle Tier	Display of Testbed health status. Initial status OK	Success	
4	Artificially increase CPU or Memory usage	Status message sent to the message bus	Success	i.e. by opening or running additional resource intensive applications in the machine where Testbed Manager is installed
5	Recheck System monitoring Service UI display at Middle Tier	Display of Testbed health status. Status changes to WARNING or CRITICAL	Success	
6	Decrease CPU or Memory usage and recheck System monitoring Service UI display at Middle Tier	Display of Testbed health status. Status changes back to OK	Success	Close extra running applications

2.6.3.2 *Network Controller*

Table 97: Verification test of network interface listing

Test ID**: NC01**	Conducted by**: CSEM**		Date**:**	Test Category**: Verification Tests (middle tier)**
Hardware Configuration				
Software Configuration				
Test Name:	***Communications interface listing and management***			
Preconditions	• Requires the Testbed Manager and data base to be active			
Related Requirements	TB-NEC-001, TB-NEC-002			
Tools Used	Message Bus and Network Controller (debug mode) logs.			
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	The Network Controller ‘lists’ the available communication interfaces (as resources) through the RAWFIE testbed database.	The Data base is accessible and the network interface information tables are filled.	Success	

Table 98: Verification test of network interface management

Test ID: NC02		Conducted by**: CSEM**	Date**:**	Test Category**:** **Verification Tests (middle tier)**
Hardware Configuration				
Software Configuration				
Test Name:		***Management of the network interfaces***		
Preconditions		• Requires the Testbed to be operational (in particular: Message Bus, UxV) • UxV availability		
Related Requirements		TB-NEC-002, TB-NEC-003, TB-NEC-004, TB-NEC-005, TB-NEC-006, UXV-INT-013		
Tools Used		Message Bus and components logs		
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	One UxV is activated but stays static. The message bus is available, the Network Controller is running. A dummy experiment is started so that there is traffic between the UxV and the testbed.		Success	
2	The UxV sends regular Network Interface performance messages to the Message Bus on topic NetwPerfUxv which contains reports on link quality, latency, etc…	The Network Controller reads the network interface performance message, records and analyses network performance data.	Success	Topic *NetwPerfUxv*
3	In parallel, the Network Controller monitors the network performance from the testbed infrastructure wherever available.	The Network Controller uses OS tools to assess link quality, latency, etc…	Success	Tested with latency
4	The Network Controller regularly publishes a high-level network performance indicator	For each link, a value between 0 (no link) and 5 (excellent link) is given. In addition some textual information on the network performance accompanies the indicator	Success	Topic *GlobalNetwPerf*
5	The performance of the primary communication interface is artificially reduced, for instance by shadowing.	The Network Controller notices the performance degradation and suggest a network interface change on topic NetwSelectIf.	Success	Performance degradation noticed and signalled.
6	The UxV switches to the secondary communication interface.	There is no or minimal (< 10s) communication break between the UxV and the testbed.	Not tested	No UxV with more than one interface available at testing time.

2.6.3.3 *Resource Controller*

Table 99 Verification test of starting/cancelling an experiment

Test ID: **RC01**	Conducted by: **CERTH**	Date:	Test Category: **Verification Tests (testbed tier)**
Hardware Configuration	-		
Software Configuration	-		
Test Name:	***Start/Cancel an experiment***		
Preconditions	• Requires the Message Bus to be accessible. • Requires Experiment Controller to be up and running.		
Related Requirements	TB-REC-001, TB-REC-002, TB-REC-006, TB-REC-007		
Tools Used			

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Send an ExperimentStartRequest type of message	Resource Controller properly consumes the message (filtering out all the messages that do not belong to the specific testbed) and initiates the command and control loop.	Success	At this point, Resource Controller assumes that the devices are ready to operate
		An experiment status update is dispatched.	Success	A running instance of the Experiment Controller is needed in order to catch this status update
2	Send an ExperimentCancelRequest type of message	Resource Controller properly consumes the message (filtering out all the messages that do not belong to the specific testbed) and dispatches abort commands to all the operational UxVs.	Success	After the abort commands, Resource Controller dispatches RTL messages in each one of the involved devices.
		An experiment status update is dispatched.	Success	A running instance of the Experiment Controller is needed in order to catch this status update

Table 100 Verification test of the command the control loop

Test ID: **RC02**		Conducted by**: CERTH**	Date**:**		Test Category**: Verification Tests (testbed tier)**
Hardware Configuration		-			
Software Configuration		-			
Test Name:		***Check functionality of command and control loop.***			
Preconditions		• Requires the Message Bus to be accessible. • Requires Experiment Controller to be up and running. • Requires all the involving UxVs to be operational.			
Related Requirements		TB-REC-003, TB-REC-004,TB-REC-005, TB-REC-007, UXV-NOD-001, UXV-SEN-004			
Tools Used					
Step	**Action**	**Expected Result**	**Status**		**Remarks**
1	Resource Controller sends a set of waypoints to all the involved UxVs	Each one of the involved UxV receives and proceeds to the commanded waypoint.	Success		
2	UxV continuously sends actual location	Resource Controller receives actual position and checks if the UxV has reached the previously transmitted waypoint (within a pre-specified radius of tolerance).	Success		
		Resource Controller sends the new set of waypoints, when all the operational UxVs have reached their current waypoints.	Success		If there is no other set of waypoints the experiment is considered COMPLETED and an appropriate ExperimentStatusMsg is dispatched to the Kafka message bus

2.6.3.4 *UxV Proximity component*

Table 101: Verification test of Proximity component Backup communication

Test ID: **UxP01**		Conducted by**: CSEM**	Date**: April 2017**	Test Category**: Verification Tests (UxV tier)**
Hardware Configuration		UxV with Proximity component (CSEM WiseNode)		
Software Configuration		UxV Embedded OS + CSEM WiseNET		
Test Name:		***Backup communication***		
Preconditions		• UxV are equipped with the Proximity component		
Related Requirements		PT-GEN-001, PT-P-001, PT-P-003, PT-A-001, PT-A-003, PT-A-004, PT-A-005, PT-A-006, PT-A-007, ,PT-A-009, ,PT-A-014, PT-A-016, PT-B-001, PT-L-002, PT-E-002, PT-E-003, TB-G-004, TB-G-006, TB-I-001, TB-G-013, TB-D-001, UXV-PRX-001, UXV-PRX-002, UXV-PRX-004		
Tools Used				
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	The UxVs are booked, the experiment is programmed and started.		Success	Tested in another context
2	The UxVs lose the connection with the primary RAWFIE communication system	The Proximity communication system takes over	Success	Tested during neighbor discovery demonstration
3	The UxVs act autonomously, following the loaded mission instructions, logging all motion parameters, exchanging information across the swarm	The UxV use the Proximity communication system.	Success	Tested during neighbor discovery demonstration
4	The UxVs come back and the logged information is analysed	The communication statistics exhibits low packet error rate and low latency	Success	Tested during neighbor discovery demonstration

Table 102: Verification test of UxV retrieval using the communication system of the Proximity component

Test ID**: UxP02**		Conducted by**: CSEM**	Date**: April 2017**	Test Category**: Verification Tests (UxV tier)**
Hardware Configuration		UxV with Proximity component (CSEM WiseNode)		
Software Configuration		UxV Embedded OS + CSEM WiseNET		
Test Name:		***UxV retrieval***		
Preconditions		• UxV are equipped with the Proximity component		
Related Requirements		PT-GEN-001, PT-P-001, PT-P-003, PT-A-001, PT-A-003, PT-A-004, PT-A-005, PT-A-006, PT-A-007, ,PT-A-009, ,PT-A-014, PT-A-016, PT-B-001, PT-L-002, PT-E-002, PT-E-003, TB-G-004, TB-G-006, TB-I-001, TB-G-013, TB-D-001, UXV-PRX-001, UXV-PRX-003, UXV-PRX-006		
Tools Used				
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	The UxVs are booked, the experiment is programmed and started.		Success	Tested in another context
2	The UxVs perform their mission and one of them exhausts its main power source		Success	Tested during neighbor discovery demonstration
3	The other UxVs uses the Proximity component communication systems to communicate and locate the stopped UxV	The connection is established with the stopped UxV and the collected information allows for locating it	Success	Tested during neighbor discovery demonstration
4	The other UxVs transmit the location and status of the stopped UxV to the RAWFIE resource manager		Success	Tested during neighbor discovery demonstration

Table 103: Verification test of Swarm motion using the Proximity component

Test ID**: UxP03**		Conducted by**: CSEM**	Date**: April 2017**	Test Category**: Verification Tests (UxV tier)**
Hardware Configuration		UxV with Proximity component (CSEM WiseNode)		
Software Configuration		UxV Embedded OS + CSEM WiseNET		
Test Name:		***Swarm motion***		
Preconditions		• UxV are equipped with the Proximity component. • Acceptable margin for the relative location of UxV is defined depending on the type of UxV and the scenario dynamics.		
Related Requirements		PT-GEN-001, PT-P-001, PT-P-003, PT-A-001, PT-A-003, PT-A-004, PT-A-005, PT-A-006, PT-A-007, ,PT-A-009, ,PT-A-014, PT-A-016, PT-B-001, PT-L-002, PT-E-002, PT-E-003, TB-G-004, TB-G-006, TB-I-001, TB-G-013, TB-D-001		
Tools Used				
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	The UxVs are booked, the experiment is programmed and started.		Success	Tested in another context
2	The UxVs perform their mission moving in a coordinated fashion		Not tested	Not implemented
3	The UxVs log all position		Not tested	Not implemented
4	The UxVs come back and the logged information is analysed	The UxV relative locations were within the acceptable margin	Not tested	Not implemented

2.6.3.5 Testbed Manager

Test procedures related to verifying Testbed Manager correct behaviour and adherence to requirements defined in D3.3 are provided in this section. Following the last test plan update (D4.9), the following modifications have been brought to the tests:

Test procedures TM01 and TM04 have been updated with extra steps added.

TM03 of D6.3 has been moved to Monitoring Manager section as MM02. TM02 of D6.3 has been eliminated based on the assumption that the actions specified in this test will be handled by proper Message Bus configuration.

TM02, TM03 and TM05 presented here are new.

Table 104: Verification test of experiment handling from testbed manager

Test ID: **TM01**	Conducted by**: HAI**	Date**: May 2018**	Test Category**: Verification Tests (Testbed tier)**	
Hardware Configuration Details				
Software Configuration Details				
Test Name:	***Testbed Manager Experiment Handling***			
Preconditions	• Requires middle tier to be accessible (Experiment Controller Service) • Requires the resource controller to be accessible • Requires local PostgreSQL Server accessible			
Related Requirements	TB-MAN-005, TB-MAN-004, TB-MAN-001, TB-MAN-007, TB-MAN-010			
Tools Used				
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Start Testbed Manager	Testbed manager successfully initialized Successful connection to the local (testbed site) database server	Success	
2	Testbed Manager receives an ExperimentStartRequest message from Message Bus	A new experiment is registered in the local database. Testbed Manager rejects experiments not intended for this testbed	Success	
3	Testbed Manager receives ExperimentStatusMsg messages from Message Bus	ExperimentStatusMsg messages are periodically transmitted from Resource Controller providing the current status of the experiment. Upon reception of a final state message the experiment is registered either as completed, failed or cancelled in the experiments history log in the local database	Success	
4	Testbed Manager sends an ExperimentCancelRequest message to the Message Bus	Resource controller receives the message and initiates all necessary actions to safely stop all UxV resources. The experiment is registered as cancelled in the experiments history log in the local database	Success	
5	User selects to see the experiments executed in the testbed	Information about the experiments executed in the testbed is retrieved from the local database (experiments log) and shown in the relevant window	Success	

Table 105: Verification test for creating and updating a testbed in the master database

Test ID: **TM02**	Conducted by: **HAI**	Date: **May 2018**	Test Category: **Verification Tests (Testbed tier)**
Hardware Configuration Details			
Software Configuration Details			
Test Name:	***Register and update a testbed in master RAWFIE database***		
Preconditions	• Requires Testbed Directory Service		
Related Requirements	TB-MAN-001, TB-MAN-007, PT-GEN-R-004, PT-DIR-S-005, PT-REE-T-001, PT-REE-T-002		
Tools Used			

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	User starts Testbed Manager application in testbed site	Testbed manager successfully initialized Successful connection to the local (testbed site) database server	Success	
2	Upon entering the application for the first time the user doesn't find any valid testbed data. The user creates a new testbed by editing the appropriate user interface window	A new testbed is created in the master database using the REST call defined in Testbed Directory Service's API (/request/createTestbed). The new testbed is displayed in Resource Explorer Tool	Success	
3	The user updates the testbed data by editing the appropriate user interface window	Testbed data are updated in the master database using the REST call defined in Testbed Directory Service's API (/request/editTestbed). The updated testbed is displayed in Resource Explorer Tool	Success	

Table 106: Verification test for creating, updating and deleting a testbed area in the master database

Test ID: **TM03**	Conducted by: **HAI**	Date: **May 2018**	Test Category: **Verification Tests (Testbed tier)**	
Hardware Configuration Details				
Software Configuration Details				
Test Name:	***Register, update and delete a testbed area in master RAWFIE database***			
Preconditions	• Requires Testbed Directory Service			
Related Requirements	TB-MAN-001, TB-MAN-007, PT-GEN-R-004, PT-DIR-S-005, PT-REE-T-001, PT-REE-T-002			
Tools Used				
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	User starts Testbed Manager application in testbed site	Testbed manager successfully initialized Successful connection to the local (testbed site) database server	Success	
2	The user creates a new testbed area by editing the appropriate user interface window	A new testbed area is created in the master database using the REST call defined in Testbed Directory Service's API (/request/createArea). The new testbed area is displayed in Resource Explorer Tool	Success	
3	The user updates an existing testbed area by editing the appropriate user interface window	The testbed area is updated in the master database using the REST call defined in Testbed Directory Service's API (/request/editArea). The updated testbed area is displayed in Resource Explorer Tool	Success	
4	The user deletes an existing testbed area	The testbed area is deleted from the master database using the REST call defined in Testbed Directory Service's API (/request/deleteArea). The testbed area now is not present in Resource Explorer Tool	Success	

Table 107: Verification test of creating, updating and deleting a resource in the master database

Test ID: **TM04**	Conducted by: **HAI**		Date: **May 2018**	Test Category: **Verification Tests (Testbed tier)**
Hardware Configuration Details				
Software Configuration Details				
Test Name:	***Register, update and delete a resource in master RAWFIE database***			
Preconditions	• Requires Testbed Directory Service			
Related Requirements	TB-MAN-002, TB-MAN-006, TB-MAN-007, PT-GEN-R-004, PT-DIR-S-007, PT-REE-T-002			
Tools Used				
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	User starts Testbed Manager application in testbed site	Testbed manager successfully initialized Successful connection to the local (testbed site) database server	Success	
2	The user creates a new UxV resource by editing the appropriate user interface window	A new resource is created in the master database using the REST call defined in Testbed Directory Service's API (/request/createResource). The new resource is displayed in Resource Explorer Tool	Success	
3	The user updates an existing UxV resource by editing the appropriate user interface window	The resource is updated in the master database using the REST call defined in Testbed Directory Service's API (/request/editResource). The updated resource is displayed in Resource Explorer Tool	Success	
4	The user deletes an existing UxV resource	The resource is deleted from the master database using the REST call defined in Testbed Directory Service's API (/request/deleteResource). The resource now is not present in Resource Explorer Tool	Success	

Table 108: Verification test for creating, updating and deleting a sensor in the master database

Test ID: **TM05**	Conducted by: **HAI**	Date: **May 2018**	Test Category: **Verification Tests (Testbed tier)**
Hardware Configuration Details			
Software Configuration Details			
Test Name:	***Register, update and delete a sensor in master RAWFIE database***		
Preconditions	• Requires Testbed Directory Service		
Related Requirements	TB-MAN-002, TB-MAN-006, TB-MAN-007, PT-GEN-R-004, PT-DIR-S-007, PT-REE-T-003		
Tools Used			

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	User starts Testbed Manager application in testbed site	Testbed manager successfully initialized Successful connection to the local (testbed site) database server	Success	
2	The user creates a new sensor by editing the appropriate user interface window	A new sensor is created in the master database using the REST call defined in Testbed Directory Service's API (/request/createSensor). The new sensor is displayed in Resource Explorer Tool	Success	
3	The user updates an existing sensor by editing the appropriate user interface window	The sensor data are updated in the master database using the REST call defined in Testbed Directory Service's API (/request/editSensor). The updated sensor is displayed in Resource Explorer Tool	Success	
4	The user deletes an existing sensor	The sensor is deleted from the master database using the REST call defined in Testbed Directory Service's API (/request/deleteSensor). The sensor now is not present in Resource Explorer Tool	Success	

Table 109: Verification test for creating, updating and deleting a network interface in the master database

Test ID: **TM06**	Conducted by**: HAI**	Date**: May 2018**	Test Category**: Verification Tests (Testbed tier)**
Hardware Configuration Details			
Software Configuration Details			
Test Name:	***Register, update and delete a network interface in master RAWFIE database***		
Preconditions	• Requires Testbed Directory Service		
Related Requirements	TB-MAN-002, TB-MAN-006, TB-MAN-007, PT-GEN-R-004, PT-DIR-S-007, PT-REE-T-003		
Tools Used			

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	User starts Testbed Manager application in testbed site	Testbed manager successfully initialized Successful connection to the local (testbed site) database server	Success	
2	The user creates a new network interface by editing the appropriate user interface window	A new network interface is created in the master database using the REST call defined in Testbed Directory Service's API (/request/createNetInterface). The new network interface is displayed in Resource Explorer Tool	Success	
3	The user updates and existing new network interface by editing the appropriate user interface window	The network interface data are updated in the master database using the REST call defined in Testbed Directory Service's API (/request/editNetInterface). The updated network interface is displayed in Resource Explorer Tool	Success	
4	The user deletes an existing new network interface	The network interface is deleted from the master database using the REST call defined in Testbed Directory Service's API (/request/deleteNetInterface). The network interface now is not present in Resource Explorer Tool	Success	

Table 110: Verification test for assigning a network interface to a resource in the master database

Test ID: **TM07**	Conducted by: **HAI**	Date: **May 2018**	Test Category: **Verification Tests (Testbed tier)**
Hardware Configuration Details			
Software Configuration Details			
Test Name:	***Associate a network interface with a resource in master RAWFIE database***		
Preconditions	• Requires Testbed Directory Service		
Related Requirements	TB-MAN-002, TB-MAN-006, TB-MAN-007, PT-GEN-R-004, PT-DIR-S-007, PT-REE-T-003		
Tools Used			

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	User starts Testbed Manager application in testbed site	Testbed manager successfully initialized Successful connection to the local (testbed site) database server	Success	
2	The user assigns a network interface to an existing resource by editing the appropriate user interface window	A new network interface/resource association is created in the master database using the REST call defined in Testbed Directory Service's API (/request/associateNetIfResource). The new network interface for the resource is displayed in Resource Explorer Tool	Success	
3	The user deletes the network interface assigned to a resource	The network interface/resource association is deleted from the master database using the REST call defined in Testbed Directory Service's API (/request/deleteNetIfResource). The information about assigned network interfaces to the resource is updated in Resource Explorer Tool	Success	

Table 111: Verification test of Aggregate Manager create, update and delete operations

Test ID: **TM08**	Conducted by: **HAI**	Date: **May 2018**	Test Category: **Verification Tests (Testbed tier)**
Hardware Configuration Details			
Software Configuration Details			
Test Name:	***Register, update and delete a resource in SFA Aggregate Manager triple store database***		
Preconditions	• Requires Aggregate Manager REST API		
Related Requirements	TB-AGG-001, TB-AGG-002, TB-AGG-003, TB-AGG-004, TB-AGG-005, TB-MAN-002, TB-MAN-007		
Tools Used			

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	User starts Testbed Manager application in testbed site	Testbed manager successfully initialized Successful connection to the local (testbed site) database server	Success	
2	The user creates a new UxV resource by editing the appropriate user interface window	A new resource is created in the triple-store database using a POST REST call defined in Aggregate Manager's API. The new resource is accessible from MySlice API	Success	
3	The user updates and existing UxV resource by editing the appropriate user interface window	The resource is updated in the triple store database using a PUT REST call defined in Aggregate Manager's API. The updated resource is accessible from MySlice API	Success	
4	The user deletes an existing UxV resource	The resource is deleted from triple-store database using a DELETE REST call defined in Aggregate Manager's API. The resource now is not present in MySlice API	Success	

Table 112: Verification test of services running at testbed

Test ID**: TM09**	Conducted by**: HAI**	Date**: May 2018**	Test Category**: Verification Tests (Testbed tier)**
Hardware Configuration Details			
Software Configuration Details			
Test Name:	***Check the status of all services running at testbed level***		
Preconditions	• Requires middle tier to be accessible (Experiment Controller Service) • Requires the resource controller to be accessible • Requires local PostgreSQL Server accessible		
Related Requirements	TB-MAN-009, TB-MAN-007		
Tools Used			

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	User starts Testbed Manager application in testbed site	Testbed manager successfully initialized Successful connection to the local (testbed site) database server	Success	
2	Testbed manager receives periodical status messages from Resource Controller and Network Manager in the Message Bus		Success	
3	User is able to see the availability of the components that run at testbed level by selecting the appropriate user interface window	Show current status of components running at testbed level	Success	

Table 113: Verification test of testbed statistics display

Test ID: **TM10**		Conducted by: **HAI**	Date: **May 2018**	Test Category: **Verification Tests (Testbed tier)**
Hardware Configuration Details				
Software Configuration Details				
Test Name:		***Display testbed statistics***		
Preconditions		• Requires the Message Bus to be accessible • Requires middle tier to be accessible (Experiment Controller Service) • Requires local PostgreSQL Server accessible		
Related Requirements		TB-MAN-009, TB-MAN-007		
Tools Used				
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	User starts Testbed Manager application in testbed site	Testbed manager successfully initialized Successful connection to the local (testbed site) database server	Success	
2	The user selects to see statistical information related to testbed usage by selecting the appropriate user interface window	Statistical information about testbed alive time, number of experiments completed/failed/cancelled and information about time utilization and participation in experiments per resource is displayed	Success	
3	A new experiment is executed in the testbed	See TM01 above	Success	
4	The user selects to see statistical information related to testbed usage by selecting the appropriate user interface window	Statistical information has been updated	Success	

2.6.3.6 UxV Node

All tests related to the establishment of a secure connection from the UxVs to the testbed and Message Bus were removed due to an architectural change: RAWFIE security is implemented by VPN, which makes the use of secure connections inside the VPN redundant.

Table 114: Verification test of UxV Return to base

Test ID: **UxV01**		Conducted by**: Rob, UoA, Certh**	Date**: 15/12/16**	Test Category**: Verification Tests (Testbed tier)**
Hardware Configuration		RobSim-SummitXL, Laser scan, IMU, camera)		
Software Configuration		RobSim-VirtualBox VM(ROS, Ubuntu 14.04,Gazebo)		
Test Name:		***Return to base***		
Preconditions		- Requires the RAWFIE system to be operational (e.g. Resource controller reachable) - Requires the mission to be defined and running. - Requires the UxV to be ready to operating (e.g. en route). - Requires the UxV to be reachable by any communication mean.		
Related Requirements		PT-EXA-T-008, PT-NAV-T-001, PT-NAV-T-002, PT-NAV-T-003, PT-VIS-T-001, TB-REC-001, TB-REC-004, UXV-NET-009, UXV-SEN-003, UXV-SEN-005, UXV-PRC-001, UXV-MGT-002 ,UXV-PRC-003,UXV-PRC-005, UXV-MGT-006, UXV-NOD-001,UXV-SEN-004, TB-UVG-001		
Tools Used		Network, Servers, Personal Computer, Skype		
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Establish the communication with the UxV	Communication established	OK	
3	Send the return to base command	Return to base command received	OK	It is treated as a waypoint to the origin
4	If the UxV is not autonomous, instruct it with the necessary waypoint or guidance information, possibly until the end of the test	Further optional instructions for returning home received, Confirmation of the UxV at home	OK	Either with provided waypoint for path planning or just one waypoint

Test ID: **UxV01**		Conducted by: **MST**	Date: **Feb 2017**	Test Category: **Verification Tests (Testbed tier)**
Hardware Configuration		AUV-0, AUV-1, and ASV-0 (as described in D6.1 and D6.2)		
Software Configuration		OceanScan-MST IMC/RAWFIE Translator (as described in D4.5)		
Test Name:		***Return to base***		
Preconditions		- Requires the RAWFIE system to be operational (e.g. Resource controller reachable) - Requires the mission to be defined and running. - Requires the UxV to be ready to operating (e.g. en route). - Requires the UxV to be reachable by any communication mean.		
Related Requirements		PT-EXA-T-008, PT-NAV-T-001, PT-NAV-T-002, PT-NAV-T-003, PT-VIS-T-001, TB-REC-001, TB-REC-004, UXV-NET-009, UXV-SEN-003, UXV-SEN-005, UXV-PRC-001, UXV-MGT-002 ,UXV-PRC-003,UXV-PRC-005, UXV-MGT-006, UXV-NOD-001,UXV-SEN-004, TB-UVG-001		
Tools Used		Neptus Command & Control Software		
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Establish the communication with the UxV	Communication established	Success	
3	Send the return to base command	Return to base command received	Success	
4	If the UxV is not autonomous, instruct it with the necessary waypoint or guidance information, possibly until the end of the test	Further optional instructions for returning home received, Confirmation of the UxV at home	Success	

Table 115: Verification test of the ability of the UxV to follow a route

Test ID: **UxV02**		Conducted by**: Rob, UoA, Certh**	Date**: 15/12/16**	Test Category**: Verification Tests (testbed tier)**
Hardware Configuration		RobSim-SummitXL, Laser scan, IMU, camera)		
Software Configuration		RobSim-VirtualBox VM(ROS, Ubuntu 14.04,Gazebo)		
Test Name:		***Follow a route***		
Preconditions		- Requires the RAWFIE system to be operational (e.g. Resource controller reachable) - Requires the mission to be defined and running. - Requires the UxV to be ready to operating (e.g. en route). - Requires the UxV to be reachable by any communication mean.		
Related Requirements		PT-EXA-T-008, PT-NAV-T-001, PT-NAV-T-002, PT-NAV-T-004, PT-VIS-T-001, TB-REC-001, TB-REC-004, UXV-NET-009, UXV-SEN-003, UXV-SEN-004, UXV-SEN-005, UXV-PRC-001, UXV-NOD-001, TB-UVG-001, UXV-INT-007, UXV-INT-008, UXV-INT-009, UXV-INT-010, UXV-INT-011		
Tools Used		Network, Servers, Personal Computer, Skype		
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Resource controller computes mission and send waypoint	Robot proceeds to the specified point,	Success	Care to choose reachable waypoints
2	Robot continuously sends actual location	RC receives position and check if WP have been reached	Success	
3	RC sends next point	Robot receives and proceed to next point	Success	Reached target location with desired location must be checked carefully by RC

Test ID: **UxV02**	Conducted by: **MST**	Date: **Feb 2017**	Test Category: **Verification Tests (testbed tier)**
Hardware Configuration	AUV-0, AUV-1, and ASV-0 (as described in D6.1 and D6.2)		
Software Configuration	OceanScan-MST IMC/RAWFIE Translator (as described in D4.5)		
Test Name:	***Follow a route***		
Preconditions	- Requires the RAWFIE system to be operational (e.g. Resource controller reachable) - Requires the mission to be defined and running. - Requires the UxV to be ready to operating (e.g. en route). - Requires the UxV to be reachable by any communication mean.		
Related Requirements	PT-EXA-T-008, PT-NAV-T-001, PT-NAV-T-002, PT-NAV-T-004, PT-VIS-T-001, TB-REC-001, TB-REC-004, UXV-NET-009, UXV-SEN-003, UXV-SEN-004, UXV-SEN-005, UXV-PRC-001, UXV-NOD-001, TB-UVG-001, UXV-INT-007, UXV-INT-008, UXV-INT-009, UXV-INT-010, UXV-INT-011		
Tools Used	Neptus Command & Control Software		

Step	**Action**	**Expected Result**	**Status**	**Step**
1	Resource controller computes mission and send waypoint	Robot proceeds to the specified point,	Success	
2	Robot continuously sends actual location	RC receives position and check if WP have been reached	Success	
3	RC sends next point	Robot receives and proceed to next point	Success	

Table 116: Verification test of Acquire sensor samples

Test ID: **UxV03**	Conducted by: **Rob, UoA, Certh**	Date: **15/12/16**	Test Category: **Verification Tests (Testbed tier)**
Hardware Configuration	RobSim-SummitXL, Laser scan, IMU, camera)		
Software Configuration	RobSim-VirtualBox VM(ROS, Ubuntu 14.04,Gazebo)		
Test Name:	***Acquire sensor samples***		
Preconditions	- Requires the RAWFIE system to be operational - Requires the mission to be defined and running. - Requires the UxV to be ready to operating (e.g. en route). - Requires the UxV to be reachable by any communication mean.		
Related Requirements	PT-NF-001, UXV-SEN-004, UXV-SEN-005, UXV-STO-001, UXV-STO-002, UXV-NET-006, PT-VIS-T-003, TB-MAN-004, UXV-STO-001, UXV-STO-002, UXV-STO-003, UXV-STO-004, UXV-SEN-001, UXV-SEN-002, UXV-SEN-003, UXV-SEN-005, UXV-MGT-001, UXV-NOD-001, UXV-MGT-006-TB-UVG-001, UXV-INT-012		
Tools Used	Network, Servers, Personal Computer, Skype		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Establish the communication with the UxV	Communication established	Success	
3	Send the acquisition commands	Commands received and executed	Success	Set of commands to be completed
4	Store sensor samples and, if possible, transmit them via the data communication system	Samples stored and, if possible, transmitted	Success	

Test ID: **UxV03**		Conducted by: **MST**	Date: **Feb 2017**	Test Category: **Verification Tests (Testbed tier)**
Hardware Configuration		AUV-0, AUV-1, and ASV-0 (as described in D6.1 and D6.2)		
Software Configuration		OceanScan-MST IMC/RAWFIE Translator (as described in D4.5)		
Test Name:		***Acquire sensor samples***		
Preconditions		- Requires the RAWFIE system to be operational - Requires the mission to be defined and running. - Requires the UxV to be ready to operating (e.g. en route). - Requires the UxV to be reachable by any communication mean.		
Related Requirements		PT-NF-001, UXV-SEN-004, UXV-SEN-005, UXV-STO-001, UXV-STO-002, UXV-NET-006, PT-VIS-T-003, TB-MAN-004, UXV-STO-001, UXV-STO-002, UXV-STO-003, UXV-STO-004, UXV-SEN-001, UXV-SEN-002, UXV-SEN-003, UXV-SEN-005, UXV-MGT-001, UXV-NOD-001, UXV-MGT-006-TB-UVG-001, UXV-INT-012		
Tools Used		Neptus Command & Control Software		
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Establish the communication with the UxV	Communication established	Success	
3	Send the acquisition commands	Commands received and executed	Success	Output of sensors is controlled via the SensorPublishControl message.
4	Store sensor samples and, if possible, transmit them via the data communication system	Samples stored and, if possible, transmitted	Success	

Table 117: Verification test of Fidelity to commands

Test ID**: UxV04**	Conducted by**: MST**	Date**: Feb 2017**	Test Category**: Verification Tests (Testbed tier)**
Hardware Configuration	AUV-0, AUV-1, and ASV-0 (as described in D6.1 and D6.2)		
Software Configuration	OceanScan-MST IMC/RAWFIE Translator (as described in D4.5)		
Test Name:	***Fidelity to commands***		
Preconditions	- Requires the RAWFIE system to be operational - Requires the mission to be defined and running. - Requires the UxV to be ready to operating (e.g. en route). - Requires the UxV to be reachable by any communication mean.		
Related Requirements	UXV-NET-006, PT-NF-001, TB-MOM-003, TB-MAN-004, UXV-STO-001, UXV-STO-002, UXV-STO-003, UXV-STO-004,, TB-UVG-001, UXV-NOD-001, UXV-PRC-003, UXV-PRC-005		
Tools Used	Neptus Command & Control Software		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Establish the communication with the UxV	Communication established	Success	
3	Send repeatedly pre-defined sets of commands, covering the full range of possible UxV actions,	Commands received and executed	Success	
4	Check the conformance of the undertaken actions and corrections (if necessary) to the commands,	Undertaken actions in conformance to the commands	Success	
5	Record all fine grained status of the UxV over the duration of the test, to be able to reconstruct the behavior of the UxV,	Status recorded	Success	

Table 118: Verification test of Continuous communication

Test ID: **UxV05**	Conducted by**: Rob, UoA, Certh**	Date**: 15/12/16**	Test Category**: Verification Tests (Testbed tier)**
Hardware Configuration	RobSim-SummitXL, Laser scan, IMU, camera)		
Software Configuration	RobSim-VirtualBox VM(ROS, Ubuntu 14.04,Gazebo)		
Test Name:	***Continuous communication***		
Preconditions	- Requires the RAWFIE system to be operational - Requires the mission to be defined and running. - Requires the UxV to be ready to operating. - Requires the UxV to be reachable by any communication mean.		
Related Requirements	UXV-NET-006, TB-MOM-003, UXV-STO-004, UXV-MGT-006, TB-UVG-001		
Tools Used	Network, Servers, Personal Computer, Skype		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Establish the communication with the UxV	Communication established	OK	
2	Exchange a predefined set of commands and data.	Commands and data correctly exchanged	OK	Location, Attitude, LaserScan tested
3	Close the communication session.	Communication closed	OK	

Test ID**: UxV05**	Conducted by**: MST**	Date**: Feb 2017**	Test Category**: Verification Tests (Testbed tier)**	
Hardware Configuration	AUV-0, AUV-1, and ASV-0 (as described in D6.1 and D6.2)			
Software Configuration	OceanScan-MST IMC/RAWFIE Translator (as described in D4.5)			
Test Name:	***Continuous communication***			
Preconditions	• Requires the RAWFIE system to be operational • Requires the mission to be defined and running. • Requires the UxV to be ready to operating. • Requires the UxV to be reachable by any communication mean.			
Related Requirements	UXV-NET-006, TB-MOM-003, UXV-STO-004, UXV-MGT-006, TB-UVG-001			
Tools Used	Neptus Command & Control Software			
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Establish the communication with the UxV	Communication established	Success	
2	Exchange a predefined set of commands and data.	Commands and data correctly exchanged	Success	
3	Close the communication session.	Communication closed	Success	

Table 119: Verification test of Continuous communication

Test ID**: UxV06**	Conducted by**: MST**	Date**: Feb 2017**	Test Category**: Verification Tests (Testbed tier)**	
Hardware Configuration	AUV-0, AUV-1, and ASV-0 (as described in D6.1 and D6.2)			
Software Configuration	OceanScan-MST IMC/RAWFIE Translator (as described in D4.5)			
Test Name:	***Continuous communication***			
Preconditions	- Requires the RAWFIE system to be operational - Requires the UxV to be ready to operating. - Requires the UxV to be reachable by any communication mean.			
Related Requirements	UXV-NET-006, PT-NF-001, TB-MOM-003, UXV-STO-004, UXV-MGT-006,TB-UVG-001			
Tools Used	Neptus Command & Control Software			
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Establish the communication with the UxV	Communication established	Success	
3	Check communication parameters	Communication parameters and status are correct and matching	Success	
4	Exchange a pre-defined set of commands and data,	Commands and data correctly exchanged	Success	
5	Close the communication session.	Communication closed	Success	

Table 120: Verification test of Secure communication

Test ID**: UxV07**	Conducted by**: MST**		Date**: Feb 2017**	Test Category**: Verification Tests (Testbed tier)**
Hardware Configuration	AUV-0, AUV-1, and ASV-0 (as described in D6.1 and D6.2)			
Software Configuration	OceanScan-MST IMC/RAWFIE Translator (as described in D4.5)			
Test Name:	***Secure communication***			
Preconditions	- Requires the RAWFIE system to be operational - Requires the mission to be defined and running. - Requires the UxV to be ready to operating (e.g. en route). - Requires the UxV to be reachable by any communication mean.			
Related Requirements	UXV-NET-006, PT-NF-001, TB-MOM-003, UXV-STO-004, TB-UVG-001, UXV-NOD-001, UXV-PRC-003, UXV-PRC-005, UXV-MGT-006			
Tools Used	Neptus Command & Control Software			

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Establish the communication with the UxV	Communication established	Success	
3	Send safe commands and measure the temporal characteristics of the communication (e.g. response time, synchronization of reception across a swarm of UxV (coordinated group of UxV), etc.).	Real-time constraints applicable to the exchanged commands are met or mismatches are detected	Success	The time of flight of messages is greater when the producer registers with the message bus, sometimes reaching more than 10 seconds. This latency is perfectly tolerated by MST vehicles

Table 121: Verification test of Real-time communication

Test ID: **UxV08**	Conducted by**: MST**		Date**: Feb 2017**	Test Category**: Verification Tests (Testbed tier)**
Hardware Configuration	AUV-0, AUV-1, and ASV-0 (as described in D6.1 and D6.2)			
Software Configuration	OceanScan-MST IMC/RAWFIE Translator (as described in D4.5)			
Test Name:	***Real-time communication***			
Preconditions	- Requires the RAWFIE system to be operational - Requires the mission to be defined and running. - Requires the UxV to be ready to operating. - Requires the UxV to be reachable (at least sporadically) by any communication mean.			
Related Requirements	UXV-NET-006, TB-MOM-003, TB-MAN-004, UXV-STO-001, UXV-STO-002, UXV-STO-003, UXV-STO-004, TB-UVG-001, UXV-MGT-003, UXV-MGT-006			
Tools Used	Neptus Command & Control Software			

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Establish the communication with the UxV	Communication established	Success	
2	Start a transaction.	Transaction started	Success	
3	Interrupt the communication at the low-level (e.g. disconnect the antenna)	Communication is interrupted, the transaction is not complete.	Success	
4	Re-establish the communication low level means	The transaction resumes and completes	Success	
5	Close the communication session.	Connection closed	Success	

Table 122: Verification test of UxV Device Management

Test ID: **UxV09**	Conducted by**: Rob**		Date**: 20/04/2017**	Test Category**: Verification Tests (Testbed tier)**
Hardware Configuration	Summit XL			
Software Configuration	ROS Indigo, Ubuntu 14.04			
Test Name:	***UxV Device Management***			
Preconditions	• ~~Requires the RAWFIE system to be operational~~ • ~~Requires the mission to be defined and running.~~ • Requires the UxV to be ready to operating (e.g. en route). • Requires the UxV to be reachable by any communication mean.			
Related Requirements	UXV-NET-006, PT-NF-001, TB-MOM-003, TB-MAN-004, UXV-STO-001, UXV-STO-002,UXV-STO-003, UXV-STO-004, UXV-MGT-006			
Tools Used	Secured Remote Desktop Application			

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Establish the communication with the UxV	Communication established	Success	Internal tool for maintenance
2	Establish a secure control session (if not done already)	Secured control session established	Success	
3	Send device management commands	Command received and applied	-	Full control of embedded robot computer
4	Check and log the status of the device	Device has responded to the commands according to the specification	Success	
5	Close the secure control session.	The UxV is home after a safe return. Connection closed	Success	

Test ID: **UxV09**	Conducted by**: MST**		Date**: Feb 2017**	Test Category**: Verification Tests (Testbed tier)**
Hardware Configuration	AUV-0, AUV-1, and ASV-0 (as described in D6.1 and D6.2)			
Software Configuration	OceanScan-MST IMC/RAWFIE Translator (as described in D4.5)			
Test Name:	***UxV Device Management***			
Preconditions	• Requires the RAWFIE system to be operational • Requires the mission to be defined and running. • Requires the UxV to be ready to operating (e.g. en route). • Requires the UxV to be reachable by any communication mean.			
Related Requirements	UXV-NET-006, PT-NF-001, TB-MOM-003, TB-MAN-004, UXV-STO-001, UXV-STO-002,UXV-STO-003, UXV-STO-004, UXV-MGT-006			
Tools Used	Neptus Command & Control Software			

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Establish the communication with the UxV	Communication established	Success	
3	Send device management commands	Command received and applied	Success	
4	Check and log the status of the device	Device has responded to the commands according to the specification	Success	

Table 123: Verification test of the UxV connection

Test ID: **UxV10**	Conducted by: **Rob, UoA, Certh**	Date: 27/2/2017	Test Category**: Verification Tests (testbed tier)**

Hardware Configuration	Summit XL
Software Configuration	Ros Indigo, Ubuntu 14.04
Test Name:	**UxV Connection Test**
Preconditions	UxV-Node launched, Message bus working
Related Requirement	UXV-NET-006, TB-MOM-003, UXV-STO-004
Tools Used	**Robot, Porto MST Facilities Network, PC**

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Kafka Subscriber is called from another machine	Topic is shown with UxV information being published	Success	
2	Kafka Publisher is called with a valid waypoint	Robot proceeds to the specified point	Success	

Test ID: **UxV10**	Conducted by: **MST**	Date: **Feb 2017**	Test Category**: Verification Tests (testbed tier)**

Hardware Configuration	AUV-0, AUV-1, and ASV-0 (as described in D6.1 and D6.2)
Software Configuration	OceanScan-MST IMC/RAWFIE Translator (as described in D4.5)
Test Name:	**UxV Connection Test**
Preconditions	UxV-Node launched, Message bus working
Related Requirement	UXV-NET-006, TB-MOM-003, UXV-STO-004
Tools Used	OceanScan-MST IMC/RAWFIE Translator (as described in D4.5) Testsuit

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Kafka Subscriber is called from another machine	Topic is shown with UxV information being published	Success	
2	Kafka Publisher is called with a valid waypoint	Robot proceeds to the specified point	Success	

Table 124: Verification test of Sensor Data Acquisition 1

Test ID: **UxV11**	Conducted by: **Rob, UoA, Certh**		Date: 27/2/2017	Test Category: **Verification Tests (Testbed tier)**
Hardware Configuration	Summit XL			
Software Configuration	Ros Indigo, Ubuntu 14.04			
Test Name:	***Sensor Data Acquisition 1***			
Preconditions	- UxV is in operation state and the parent UxV node has been launched - Network Communication is also fully functional			
Related Requirements	UXV-NET-006, PT-NF-001, TB-MOM-003, TB-MAN-004, UXV-STO-001, UXV-STO-002,UXV-STO-003, UXV-STO-004, UXV-SEN-004, UXV-MGT-001, UXV-MGT-006			
Tools Used	**Robot, Porto MST Facilities Network, PC**			
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Establish the communication with the UxV	Communication established	Success	
3	Acquire sensor data	Data acquired (every sensor works as specified)	Success	
4	Send acquired data	Data received	Success	

Test ID: **UxV11**	Conducted by: **MST**		Date: **Feb 2017**	Test Category: **Verification Tests (Testbed tier)**
Hardware Configuration	AUV-0, AUV-1, and ASV-0 (as described in D6.1 and D6.2)			
Software Configuration	OceanScan-MST IMC/RAWFIE Translator (as described in D4.5)			
Test Name:	***Sensor Data Acquisition 1***			
Preconditions	- UxV is in operation state and the parent UxV node has been launched - Network Communication is also fully functional			
Related Requirements	UXV-NET-006, PT-NF-001, TB-MOM-003, TB-MAN-004, UXV-STO-001, UXV-STO-002,UXV-STO-003, UXV-STO-004, UXV-SEN-004, UXV-MGT-001, UXV-MGT-006			
Tools Used	Neptus Command & Control Software			
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Establish the communication with the UxV	Communication established	Success	
3	Acquire sensor data	Data acquired (every sensor works as specified)	Success	Individual sensor data is tested
4	Send acquired data	Data received	Success	Provides data gathered by each sensor placed on the robot. Data streamed of every sensor is tested individually

Table 125: Verification test of Sensor Data Acquisition 2

Test ID**: UxV12**	Conducted by**: Rob, UoA, Certh**	Date: 27/2/2017	Test Category**: Verification Tests (Testbed tier)**	
Hardware Configuration	Summit XL			
Software Configuration	Ros Indigo, Ubuntu 14.04			
Test Name:	***Sensor Data Acquisition 2***			
Preconditions	- UxV is in operation state and the parent UxV node has been launched - Network Communication is also fully functional			
Related Requirements	UXV-NET-006, PT-NF-001, TB-MOM-003, TB-MAN-004, UXV-STO-001, UXV-STO-002,UXV-STO-003, UXV-STO-004, UXV-SEN-004, UXV-MGT-001, UXV-MGT-006			
Tools Used	**Robot, Porto MST Facilities Network, PC**			
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Establish the communication with the UxV	Communication established	Success	
3	Instruct the robot to move to a known location	Robot at the specific location	Success	
4	Acquire current location data	Location data acquired (location sensor works as specified)	Success	
5	Send acquired location data	Data received	Success	

Test ID**: UxV12**	Conducted by**: MST**	Date**: Feb 2017**	Test Category**: Verification Tests (Testbed tier)**
Hardware Configuration	AUV-0, AUV-1, and ASV-0 (as described in D6.1 and D6.2)		
Software Configuration	OceanScan-MST IMC/RAWFIE Translator (as described in D4.5)		
Test Name:	***Sensor Data Acquisition 2***		
Preconditions	- UxV is in operation state and the parent UxV node has been launched - Network Communication is also fully functional		
Related Requirements	UXV-NET-006, PT-NF-001, TB-MOM-003, TB-MAN-004, UXV-STO-001, UXV-STO-002,UXV-STO-003, UXV-STO-004, UXV-SEN-004, UXV-MGT-001, UXV-MGT-006		
Tools Used	Neptus Command & Control Software		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Establish the communication with the UxV	Communication established	Success	
3	Instruct the robot to move to a know location	Robot at the specific location	Success	Robot is moved to a precisely located point and a comparison is done later
4	Acquire current location data	Location data acquired (location sensor works as specified)	Success	Localization of the robot is tested.
5	Send acquired location data	Data received	Success	Provides data about the location of the robot. Location is compared to known location.

Test ID: **UxV13**		Conducted by: **MST**	Date: **Feb 2017**	Test Category: **Verification Tests (Testbed tier)**
Hardware Configuration		AUV-0, AUV-1, and ASV-0 (as described in D6.1 and D6.2)		
Software Configuration		OceanScan-MST IMC/RAWFIE Translator (as described in D4.5)		
Test Name:		***Data Storage***		
Preconditions		- UxV is in operation state and the parent UxV node has been launched. - Sensor node is functional		
Related Requirements		UXV-NET-006, TB-MAN-004, UXV-STO-001, UXV-STO-002,UXV-STO-003, UXV-STO-004, TB-MAN-004, UXV-STO-001, UXV-STO-002, UXV-STO-003, UXV-STO-004, UXV-STO-005, UXV-MGT-006		
Tools Used		Neptus Command & Control Software		
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Establish the communication with the UxV	Communication established	Success	
3	A request for storing certain data is done	Command received and data is stored locally	Not tested	The UxVs store all data, thus store command not needed
4	After a given mission, data storage in the system is checked.	Data was correctly stored and kept.	Success	The data is stored and identified in the robot system

Table 126: Verification test of Waypoints Processed

Test ID: **UxV14**		Conducted by: **Rob, UoA, Certh**	Date: **15/12/16**	Test Category: **Verification Tests (Testbed tier)**
Hardware Configuration		RobSim-SummitXL, Laser scan, IMU, camera)		
Software Configuration		RobSim-VirtualBox VM(ROS, Ubuntu 14.04,Gazebo)		
Test Name:		***Waypoints Processed***		
Preconditions		- UxV is in operation state and the UxV parent node has been launched. - Sensor node is functional, network communication is functional		
Related Requirements		UXV-NET-006, TB-MAN-004, UXV-STO-001, UXV-STO-002,UXV-STO-003, UXV-STO-004, UXV-SEN-004, UXV-MGT-006		
Tools Used		Network, Servers, Personal Computer, Skype		
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Establish the communication with the UxV	Communication established	OK	
3	Waypoints are sent to the UxV	UxV receives and processes the waypoints	OK	
4	The calculated route is applied to the UxV	The actual trajectory matches the route calculated by the navigation.	OK	
5	Iterate step 4 until assessment is complete	UxV stops, informs and recalculate its route to next waypoint if an unexpected obstacle is found.	OK	Recalculation is done internally by UxV node

Test ID: **UxV14**		Conducted by: **MST**	Date: **Feb 2017**	Test Category: **Verification Tests (Testbed tier)**
Hardware Configuration		AUV-0, AUV-1, and ASV-0 (as described in D6.1 and D6.2)		
Software Configuration		OceanScan-MST IMC/RAWFIE Translator (as described in D4.5)		
Test Name:		***Waypoints Processed***		
Preconditions		- UxV is in operation state and the UxV parent node has been launched. - Sensor node is functional, network communication is functional		
Related Requirements		UXV-NET-006, TB-MAN-004, UXV-STO-001, UXV-STO-002,UXV-STO-003, UXV-STO-004, UXV-SEN-004, UXV-MGT-006		
Tools Used		Neptus Command & Control Software		
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Establish the communication with the UxV	Communication established	Success	
3	Waypoints are sent to the UxV	UxV receives and processes the waypoints	Success	Semi-autonomous mission is tested. The UxV has to process a set of waypoints and move to each waypoint in sequence. The UxV processes the data.
4	The calculated route is applied to the UxV	The actual trajectory matches the route calculated by the navigation.	Success	
5	Iterate step 4 until assessment is complete	UxV stops, informs and recalculate its route to next waypoint if an unexpected obstacle is found.	Not Tested	The UxVs used in this test are not equipped with obstacle avoidance systems.

2.7 Benchmarking of different Message Bus topologies and configurations

2.7.1 Purpose

The message bus is a key element of the RAWFIE system, both from the point of view of the features and of the performance. Benchmarking kafka on reference platforms will give valuable and reliable indications for the dimensioning of the RAWFIE system so that, in similar conditions, it can increase the chances formeeting the time constraintsduring most of the experimentation execution.

2.7.2 Scenarios and setup

The detailed description of the test setup, kafka configuration and other hardware and software parameters are given in section 3.2.4 of deliverable D4.7. The next paragraphs give the most important aspects of the considered scenarios. Scenario A corresponds to a Single centralised Apache Kafka Broker. The scenario B corresponds to Multiple Apache Kafka

Brokers with the same topics on each different Testbed. The scenario C corresponds to the Multiple Apache Kafka Brokers with different topics per testbed.

For scenario A, a Kafka cluster with 4 nodes was created. All VMs were running in 2GB RAM. Every VM was running a producer and a consumer. Jconsole was used for collecting metrics and exporting them.

For scenario B and C a cluster of 5 computers with 3 Kafka nodes and 3 Zookeeper instances were used. Acting as the simulated Testbed environments 2 Virtual Machines each in a different network were connected to the internet with a regular ADSL connection. In scenarios B and C, all the messages were sent in the VPN network as was established in all testbeds for security reasons.

For Scenario A the metrics described in the following were collected. This is the complete result set for 1000 records. All messages were sent to one topic from the same remote machine (i.e., running on a different country than the Kafka server). The consumer and producer run on separate threads. Each dispatched record contains a timestamp that can be used to measure the round-trip time (RTT). Two scenarios were tested:

a) burst produce/consume: the producer dispatches a burst of 1000 records back to back to the message bus and the elapsed time is recorded (TX). The consumer reads those 1000 records from the message as soon as they are available and the elapsed time is recorded (RX). In this scenario we try to measure the latency characteristics of records that are not used for automatic control of UxVs (i.e., payload sensor data, basic telemetry) and therefore will not trigger any reply.
b) synchronous produce/consume: the producer dispatches one record to the message bus and the elapsed time is recorded (TX) it then waits for the consumer to read the record from the message bus and this elapsed time is recorded (RX). In this scenario we try to measure the latency characteristics of records that may trigger a reply (i.e. waypoint references).

For scenarios B and C, Kafka metrics from the TotalTimeMs family were collected. Each virtual machine was running one Kafka broker and in the case of the third scenario one Zookeeper instance. In each scenario, we had two producers sending 50 messages per second and ten consumers running locally in every VM, emulating the traffic in a Testbed environment where UxV devices performing the produce and consume operations pointed to their local broker. For Scenario C scenario we also had the Apache Kafka Mirror Maker tool performing the mirroring from the virtual machines broker to the cluster located in the UoA premises.

Field trials were also performed for scenario B. For the field trials a UAV mission and USVs mission were used where devices where handled specifically by RAWFIE implementing Scenario B as message infrastructure. The field trials were performed in a testbed with high network complexity where the UxVs were connected either by 4G network or WiFi to a testbed operator server which in turn route the produced messages in the Kafka cluster in a different geographical area. Consuming messages required the opposite path. However the performance penalty despite the network difficulties was small. Every participating node in the experiments was clock synchronized via NTP servers. TotalTimeMs is the total time taken to service a request (be it a produce, fetch-consumer, or fetch-follower request) from Jconsole. The TotalTimeMs measurement itself is the sum of four metrics:

- queue: time spent waiting in the request queue
- local: time spent being processed by leader
- remote: time spent waiting for follower response (only when requests.required.acks=-1)
- tresponse: time to send the response

2.7.3 Results

Table 127 summarises the execution performance of kafka in the two metrics in the scenario A. The test runs over more than 100s and 20s respectively.

Table 127: Sync and Burst cased tested in scenario A

	Sync Test (TX/RX) | 1000 records	Burst Test (TX/RX) | 1000 records
Subscribed Topics	1	1
Elapsed Time	113226 ms	21662 ms
Schema Initialization	8 ms	11 ms
Kafka Producer Initialization	3 ms	3 ms
Kafka Consumer Initialization	5266 ms	5075 ms
Kafka Consumer Shutdown	0 ms	611 ms

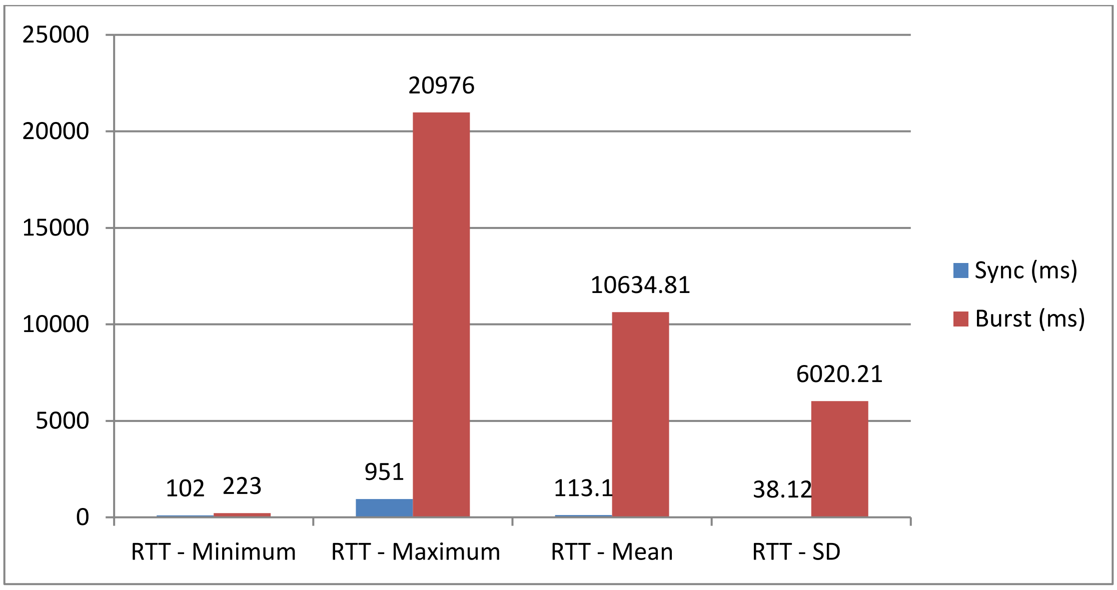


Figure 10: Round Trip Time metrics in scenario A

Note: Y axis is duration in millisecond.

In the burst test, which results are displayed in Figure 10, the producer does not wait for the consumer to complete. The Round Trip Time is measured using the timestamp in the transmitted/received record. The interpretation of the observed phenomenon is that the first dispatched messages takes longer to return to the consumer than the next dispatched messages. This is usually due to on-demand resource allocation, routing, queue establishment, handshaking, etc. to which kafka may be also sensitive.

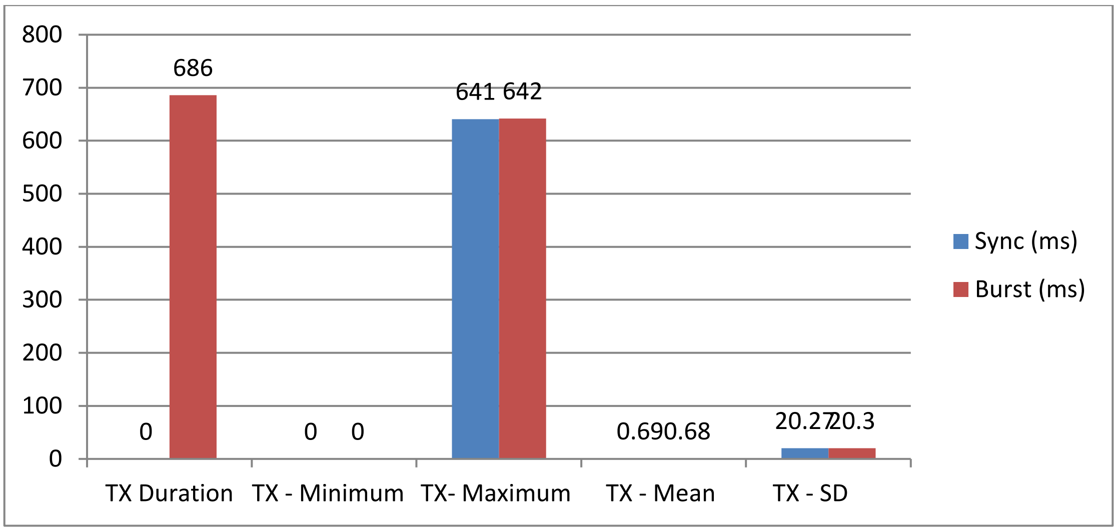


Figure 11: TX metrics in Scenario A

Note: Y axis is always duration in millisecond.

The TX duration on Figure 11 is the time it takes to pass the message to the Kafka infrastructure. Only the producer side is accounted for.

We used the produce and fetch-consumer measurements in each scenario and the results are shown bellow

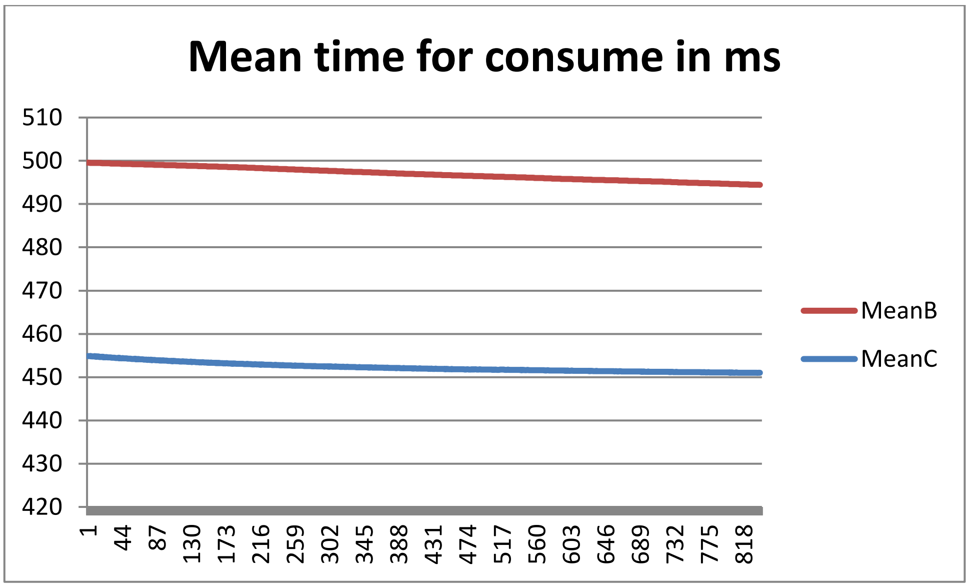


Figure 12: Mean Time for consuming messages in Scenarios B and C

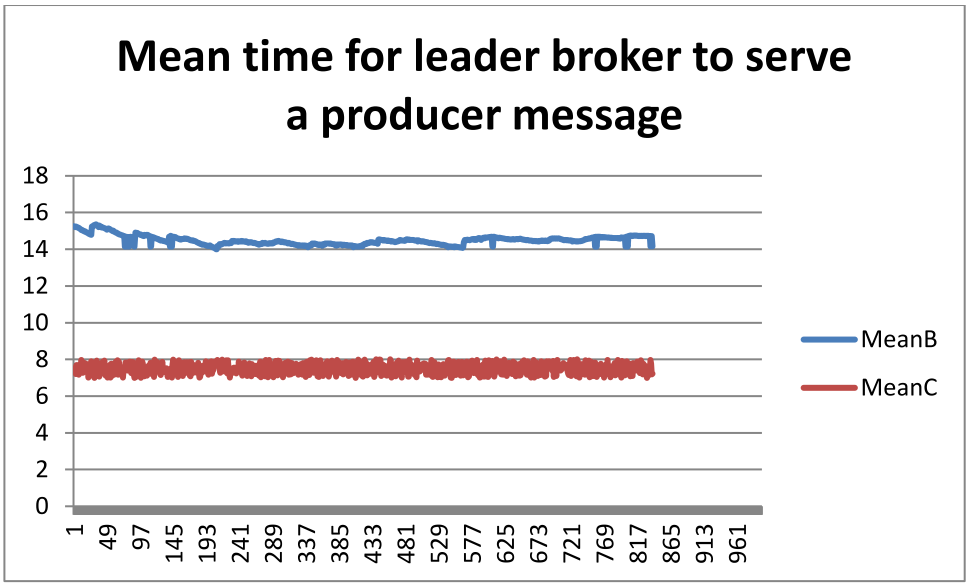


Figure 13: Mean Time for leader broker to serve messages in Scenarios B and C

Figure 12 shows the results of the consumer measurements from the time that a consumer sends a request to consume from a partition in the Kafka broker until it's request is serviced

Figure 13 shows the results of the producer measurements from the time a producer sends a produce request to the time the leader broker in the UoA Kafka cluster send a response that the produce request was completed.

From the figures above we can notice that the time for serving a produced message is lower in scenario C than in the related values in scenarios A and B. This was expected because the broker in its testbed is assigned to handle a bunch of messages produced and consumed by a small number of the devices. The small amount of partitions enhances the handling of the messages between the entities.

On both scenarios B and C, a load balancing mechanism was applied for serving the messages requests in local and in global layer. The messages were served in the logical boundaries of a server and delays from road trips were obviated. Scenario C was further enhanced by avoiding the repartitioning, which is an action that can lead to errors during the delivery of the messages. Every testbed broker handles topics different from the others. Partitions of topics in other testbeds are not affected by adding or removing devices or even a whole testbed.

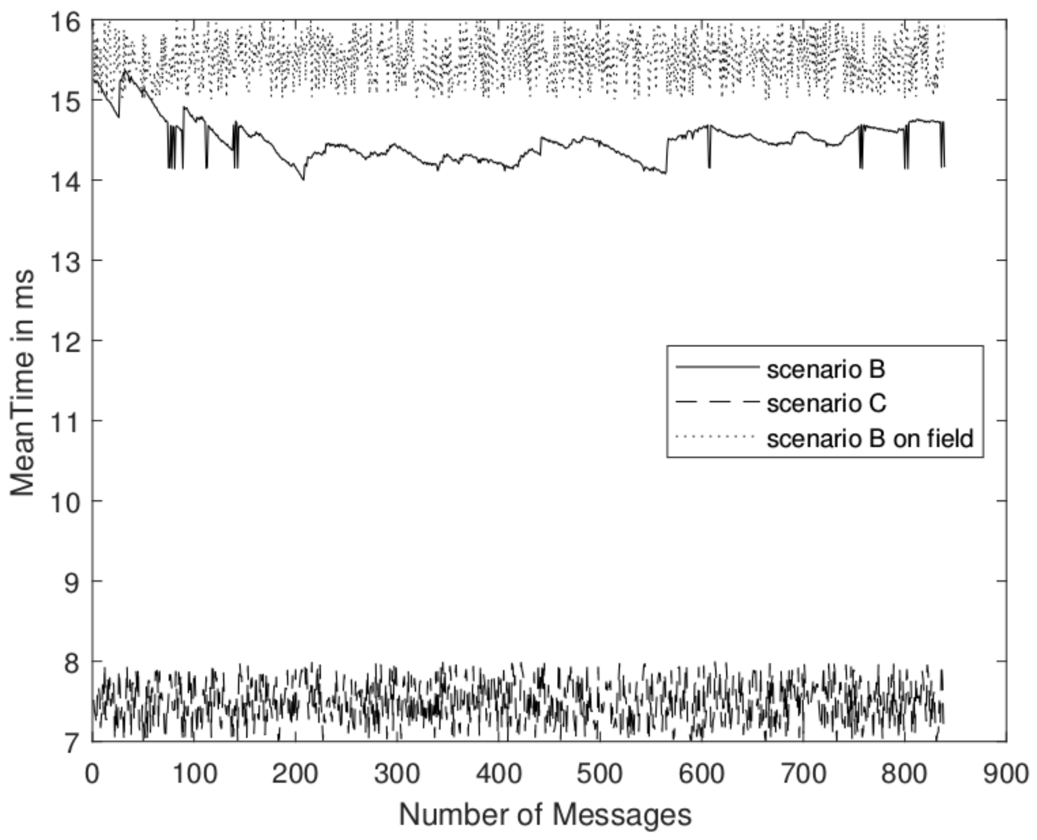


Figure 14: Mean Time for leader broker to serve messages in Scenarios B and C

Figure 14 summarizes the Mean time for a broker to serve a producer message for scenarios B,C and the field trials on scenario B. We can notice that the time for serving a produced message is lower in scenario C than in the related values in scenarios A and B and the field trials. This was expected because the broker in its testbed is assigned to handle a bunch of messages produced and consumed by a small number of the devices. The small amount of partitions enhances the handling of the messages between the entities.

2.7.4 Discussion

It is apparent from the aforementioned metrics that Scenario A with a centralized broker approach is not an effective solution for meeting the RAWFIE needs, as it was expected. The high number of messages exchanged in parallel executions of experiments can lead to system delays as shown in figure 5. The mean time in RTT for a burst of 1000 records was measured close to 10,6 seconds. However, the performance of system was improved with the use of cluster architecture. Ultimately, The tests for scenario A were designed to assess the expected performance of Kafka when using MST producer/consumer implementations. Due to the slow dynamics of MST watercrafts and the resilience to latency of our on-board the results were deemed acceptable.

On both scenarios B and C, a load balancing mechanism was applied for serving the messages requests in local and in global layer. The messages were served in the logical boundaries of a server and delays from road trips were obviated. Scenario C was further enhanced by avoiding the repartitioning, which is an action that can lead to errors during the delivery of the messages. Every testbed broker handles topics different from the others. Partitions of topics in other testbeds are not affected by adding or removing devices or even a whole testbed. Each testbed is a micro-system that controls and knows only the devices in it. This concludes that a local broker needs the half time (8 ms from 16 ms) for serving a produced message as shown in figure 8. This achievement was the reason for migrating from scenario A to scenario C as the main infrastructure for RAWFIE message bus.

2.8 Deviations with respect to D6.1, D6.3 and D4.9

This deliverable shows a near exhaustive coverage of the latest platform requirements and all tests specified in D4.9 have been executed. Almost all tests have been executed with success. Minor deviations are due to minor features not anymore relevant, for instance the suppression of a command when the related task is executed spontaneously like in the on-board storage case.

Also, a few features have been tested in different context or scenario than originally planned for convenience reason. This is described in the remark column of the concerned test results.

Part III: Conclusion & Roadmap

The RAWFIE integration process is mature enough for the operation of the platform. All requirements are covered by the implementation and tested successfully with only a few minor deviations, and the correct interaction of the numerous platform components has been demonstrated.

The platform will of course continue to evolve in the future to address special needs arising from its users thanks to its design based on popular and easy to use interfaces which favours evolution.

Part IV: Annex

Annex A Glossary

The RAWFIE glossary consists of generic terms, contributed by all partners, used across the entire RAWFIE project.

A

Accounting Service

RAWFIE component. Component that keeps track of resources usage by individual users.

Aggregate Manager

Slice Federation Architecture (SFA) term. The Aggregate Manager API is the interface by which experimenters discover, reserve and control resources at resource providers.

Avro

Apache Avro: a remote procedure call and data serialization framework

B

Booking Service

RAWFIE component. The Booking Service manages bookings of resources by registering data to appropriate database tables.

Booking Tool

RAWFIE component. The Booking tool will provide the appropriate Web UI interface for the experimenter to discover available resources and reserve them for a specified period.

C

Common Testbed Interface

RAWFIE component. The set of software and hardware functionalities each Testbed provider should ensure, for the communication with Middle Tier software components of RAWFIE, therefore for the integration with the RAWFIE platform

Component

A reusable entity that provides a set of functionalities (or data) semantically related. A component may encapsulate one or more modules (see definition) and should provide a well defined API for interaction

D

Data Analysis Engine

RAWFIE component. The Data Analysis Engine enables the execution of data processing jobs by sending requests to a processing engine which will perform the computations specified when the analytical task was defined through the Data Analysis Tool to be transmitted to the processing engine for execution.

Data Analysis Tool

RAWFIE component. The Data Analysis Tool enables the user to browse available data sources for subject to analytical treatment as well as previous analysis tasks' outcomes.

E

EDL Compiler & Validator

RAWFIE component. The EDL validator will be responsible for performing syntactic and semantic analysis on the provided EDL scripts.

Experiment Authoring Tool

RAWFIE component. This component is actually a collection of tools for defining experiments and authoring EDL scripts through RAWFIE web portal. It will provide features to handle resource requirements/configuration, location/topology information, task description etc.

Experiment Controller

RAWFIE component. The Experiment Controller is a service placed in the Middle tier and is responsible to monitor the smooth execution of each experiment. The main task of the experiment controller is the monitoring of the experiment execution while acting as ‘broker’ between the experimenter and the resources.

Experiment Monitoring Tool

RAWFIE component. Shows the status of experiments and of the resources used by experiments.

Experiment Validation Service

RAWFIE component. The Experiment Validation Service will be responsible to validate every experiment as far as execution issues concern.

M

Master Data Repository

RAWFIE component. Repository that stores all main entities that are needed in the RAWFIE platforms. Is an SQL-database

Measurements Repository

RAWFIE component. Stores the raw measurements from the experiments

Message Bus

Also known as Message Oriented Middleware. A message bus is supports sending and receiving messages between distributed systems. It is used in RAWFIE across all tiers to enable asynchronous, event-based messaging between heterogeneous components. Implements the Publish/Subscribe paradigm.

Module

A set of code packages within one software product that provides a special functionality

Monitoring Manager

RAWFIE component. Monitors the status of the testbed and the UxVs belonging to it, at functional level, e.g. the 'health of the devices' and current activity.

N

Network Controller

Manages the network connections and the switching between different technologies in the testbed in order to offer seamless connectivity in the operations of the system.

L

Launching Service

RAWFIE component. The Launching Service is responsible for handling requests for starting or cancellation of experiments.

R

Resource Controller

RAWFIE component. The Resource Controller can be considered as a cloud robot and automation system and ensures the safe and accurate guidance of the UxVs.

Resource Explorer Tool

RAWFIE component. The experimenter can discover and select available testbeds as well as resources/UxVs inside a testbed with this tool. Administrators can manage the data.

Results Repository

RAWFIE component. Stores the results of data analyses.

Resource Specification (RSpec)

SFA term. This is the means that the SFA uses for describing resources, resource requests, and reservations (declaring which resources a user wants on each Aggregate).

S

Schema Registry

A schema registry is a central service where data schemas are uploaded to. As an added benefit each schema has versions with it can convert allowable formats to other ones (e.g.: float to double) It maintains schemas for the data transferred and keeps revisions to be able to upgrade the definitions as with the simple field conversion. Used in RAWFIE for messages on the message bus.

Service

A component that is running in the system, providing specific functionalities and accessible via a well known interface.

Slice Federation Architecture (SFA)

SFA is the de facto standard for testbed federation and is a secure, distributed and scalable narrow waist of functionality for federating heterogeneous testbeds.

Subsystem

A collection of components providing a subset of the system functionalities.

System

A collection of subsystems and/or individual components representing the provided software solution as a whole.

System Monitoring Service

RAWFIE component. Checks readiness of main components and ensure that all critical software modules will perform at optimum levels. Predefined notification are triggered whenever the corresponding conditions are met, or whenever thresholds are reached

System Monitoring Tool

RAWFIE component. Shows the status and the readiness of the various RAWFIE services and testbed

T

Testbed

A testbed is a platform for conducting rigorous, transparent, and replicable testing of scientific theories, computational tools, and new technologies.

In the context of RAWFIE, a testbed or testbed facility is a physical building or area where UxVs can move around to execute some experiments. In addition, the UxVs are stored in or near the testbed.

Testbeds Directory Service

RAWFIE component. Represents a registry service of the middleware tier where all the integrated testbeds and resources accessible from the federated facilities are listed, belonging to the RAWFIE federation.

Testbed Manager

RAWFIE component. Contains accumulated information about the UxVs resources and the experiments of each one of the federation testbeds.

Tool

A GUI implementation to do a special thing, e.g. the “Resource Explorer tool” to search for a resource

U

Users & Rights Repository

RAWFIE component. Management of users and their roles. Is a directory services (LDAP).

Users & Rights Service

RAWFIE component. Manages all the users, roles and rights in the system.

UxV

The generic term for unmanned vehicle. In RAWFIE, it can be either:

USV -	Unmanned Surface vehicle.
UAV -	Unmanned Aerial vehicle.
UGV -	Unmanned Ground vehicle.
UUV -	Unmanned Underwater vehicle.

UxV Navigation Tool

RAWFIE component. This component will provide to the user the ability to (near) real-time remotely navigate a squad of UxVs.

UxV node

RAWFIE component. A single UxV node. The UxV is a complete mobile system that interacts with the other Testbed entities. It can be remotely controlled or able to act and move autonomously.

V

Visualisation Engine

RAWFIE component. Used for providing the necessary information to the Visualisation tool, to communicate with the other components, to handle geospatial data, to retrieve data for experiments from the database, to load and store user settings and to forward them to the visualisation tool.

Visualisation Tool

RAWFIE component. Visualisation of an ongoing experiment as well as visualisation of experiments that are already finished

W

Web Portal

RAWFIE component. The central user interface that provides access to most of the RAWFIE tools/services and available documentation.

Wiki Tool

RAWFIE component. Provides documentation and tutorials to the users of the platform.

Annex B Requirements

The requirements listed in Table 128: Requirements considered for the integration are considered in the context of the integration.

Table 128: Requirements considered for the integration

PT-WEB-P-001	A web portal interface shall be provided to the users of the platform to access almost all main functionalities.
PT-BOO-T-003	Booking Tool should delegate all its actions related to Booking of a resource to the Booking Service
PT-BOO-T-004	Booking Tool may also interact with the Testbeds Directory Service in order to retrieve information on unallocated testbed resources
PT-REE-T-004	Link to the Booking Tool should be provided
PT-EXM-T-003	Cancellation of running experiments should be possible via Web Portal
PT-VIS-T-002	A 3D visualization should be available for the tracking of all moving resources
PT-VIS-T-004	The Visualisation Tool shall provide access to information / features associated to each UxV device on the geographic map
PT-DAA-T-001	Analysis tool will provide interface to data engine.
PT-DAA-T-002	Analysis tool will provide ability to query available data schemas
PT-DAA-T-003	Analysis tool will be able to read results from Results Database
PT-DAA-E-001	Analysis Engine will be able to query message bus streams
PT-DAA-E-001	Analysis Engine will be able to receive messages from Analysis Tool
PT-DAA-E-002	Analysis Engine will be able to write data to the Results Database
PT-DIR-S-007	The Testbed Directory Service shall provide the possibility to register new resources belonging to a specific testbed in the RAWFIE platform, as well as to unregister (delete) resources
PT-CPV-001	A tool for translating EDL into user directives shall be provided
PT-CPV-002	An experimenter should have the opportunity to use a code generation engine
PT-CPV-003	Experiments defined via EDL shall be validated after their authoring
PT-CPV-004	The compiler and validator should communicate with the authoring tool in order to transfer error indications and hints for solving them
PT-BOO-S-006	Booking Service should be able to compute and return feedback on conflicting bookings for a provided booking request
PT-LAU-S-001	Launching Service should support short-term or manual launching of an experiment initiated directly by an experimenter
PT-VIS-E-001	The Visualization Engine shall handle the communication with the Message Bus, for the information that will be coming from the UxVs
PT-EXP-C-002	RAWFIE platform shall allow experimenters to remotely navigate UxVs.
PT-EXP-C-006	The Experiment Controller shall support receiving feedback at regular intervals from all testbed facilities about the progress of the experiment in this time interval

PT-EXP-C-008	The Experiment Controller shall be able to continuously feed the front-end tier (Experiment Monitoring Tool) giving the experimenter a clear view of the experiment workflow as a whole
PT-EXA-T-001	Experiment Description Language (EDL) shall be used as a language for the definition of experiment scenarios
PT-EXA-T-002	The EDL shall allow the definition of all necessary requirements for an experiment
PT-EXA-T-003	For each defined experiment specific metadata, i.e. name, version, date and description shall be defined.
PT-EXA-T-004	An experimenter shall be able to provide initial conditions and/or configuration parameters for an experiment
PT-EXA-T-005	An experimenter shall be able to manage/guide the available booked resources during experiment authoring
PT-EXA-T-008	An experimenter shall be able to provide navigation or movement directives during experiment authoring
PT-EXA-T-009	An experimenter should be able to create groups of UxVs resources, for which specific directives will apply.
PT-EXA-T-010	A textual editor shall be provided for the authoring of RAWFIE experiments
PT-EXA-T-011	A visual/graphical editor shall be provided for the authoring of RAWFIE experiments
PT-EXA-T-012	Platform shall allow saving, editing and/or deletion of an experiment defined via EDL
PT-EXA-T-013	The visual editor should allow the definition of movement and location waypoints from a map
PT-EXA-T-015	Validation of EDL script should be possible prior to or during saving
PT-EXV-S-001	RAWFIE shall provide a validator to constantly check experiment scenarios during runtime
PT-EXV-S-002	The validation service should perform syntactic checking
PT-EXV-S-003	The validation service should perform semantic checking
TB-MOM-004	Testbed monitoring manager should be able to transmit the current status to the System Monitoring Service.
TB-REC-003	The Resource Controller shall receive location messages from the vehicles at regular intervals
TB-REC-005	For the experiment accomplishment the Resource Controller shall operate in close coordination with the Experiment Controller
TB-MAN-005	Testbed Manager shall be periodically informed about the status of all running experiments in the testbed
UXV-NET-006	UxV communication interoperability with RAWFIE (incoming)
UXV-NET-007	UxV communication interoperability with RAWFIE (outgoing)
UXV-SEN-005	UxVs should sent a notification to the Resource Controller when they reach the desired location

References

[1]	Xtext: https://eclipse.org/Xtext/index.html
[3]	OpenLayers: http://openlayers.org/
D4.3	Pilot Experimentation, Scenarios for Validation and Testing (a)
D4.4	High Level Design and Specification of RAWFIE Architecture (b)
D4.5	Design and Specification of RAWFIE Components (b)
D4.6	Pilot Experimentation Scenarios for Validation and Testing (b)
D4.8	Design and Specification of RAWFIE Components (c)
D4.9	Pilot Experimentation Scenarios for Validation and Testing (c)
D5.3	Development of RAWFIE Components (b)
D6.1	RAWFIE Operational Platform Testing and Integration Report (a)
D6.2	RAWFIE Platform Validation (a)
D6.3	RAWFIE Operational Platform Testing and Integration Report (b)