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	D6.1 (a): RAWFIE Operational Platform Testing and Integration Report (a)		
Confidentiality	PU	Deliverable type[1]	R
Deliverable File	D6.1	Date	31.05.2016
Approval Status[2]	WP Leader, 1st Reviewer, 2nd Reviewer	Version	1.0
Contact Person	Philippe Dallemagne	Organization	CSEM
Phone		E-Mail	Philippe.Dallemagne@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
Philippe Dallemagne	CSEM	Philippe.dallemagne@csem.ch
Nikolaos Pringouris	HAI	Priggouris.nikolaos@haicorp.com
Jason Ramapuram	HES-SO	Jason-emmanuel.ramapuram@hesge.ch
Marcel Heckel	Fraunhofer	marcel.heckel@ivi.fraunhofer.de
Giovanni Tusa	IES	g.tusa@iessolutions.eu
Kostas Kolomvatsos	UoA	kostasks@di.uoa.gr
Miltiadis Kyriakakos	UoA	miltos@di.uoa.gr
Kiriakos Georgouleas	HAI	Georgouleas.kiriakos@haicorp.com
Vasil Kumanov	Epsilon Bulgaria	Vasil.kumanov@epsilon-bulgaria.com
Kakia Panagidi	UoA	kakiap@di.uoa.gr
Ricardo Martins	MST	Rasm@oceanscan-mst.com

REVIEWERS TABLE

Name	Company	E-Mail
Marcel Heckel	Fraunhofer	
Giovanni Tusa	IES	g.tusa@iessolutions.eu

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	01.03.2016	Template	All
0.2	17.03.2016	TOC / Initial version	All
0.3	21.04.2016	Revised ToC	All
0.4	09.05.2016	Version for editing	All
0.5	19.05.2016	Consolidated version	All
0.6	26.05.2016	Version for internal review	All
1.0	31.05.2016	Final version	All

Abstract:

The objective of this deliverable is to report about the interface tests, the verification tests and to present the integration results and all the technicalities required for the consolidation of the several components (software and hardware) of the RAWFIE architecture in a unified platform. Enhancements of the RAWFIE operational platform based on the outcomes of the testing procedures are also listed in this deliverable. The document is released as a live document in three phases/cycles according to the roadmap.

This deliverable is based on the results of the following tasks: T6.1 and T6.2 on the basis of the work done in WP5, and on the verification tests planning presented in D4.3.

Keywords: Integration, **i**nterface tests, verification tests,

Part II: Table of Contents-

List of Figures

List of Tables

Error! Reference source not found.

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. 95.

Part III: Executive Summary

The objective of this deliverable is to report about the results obtained during the tests of the component interfaces and of the integration. It presents the identified enhancements of the RAWFIE operational platform based on the aforementioned results. The document is an evolutive document delivered in three phases/cycles according to the roadmap.

Chapter 1 presents the scope of the document, some definitions and abbreviations together with the relation to other RAWFIE deliverables. Chapter 2 describes the interface and verification tests performed on the RAWFIE components and system. Preliminarily, it presents the approach and methodology used for describing, performing and reporting the tests and integration verification. Based on the results obtained from the previous steps, the roadmap followed by the RAWFIE project is impacted and the subsequent modifications and improvements are listed in Chapter 3. Further customisations are briefly mentioned in Chapter 4. A conclusion is drawn in Chapter 5.

Part IV: Main Section

1 Introduction

1.1 Scope of D6.1

The scope of this document is to report about the integration of all components developed in Tasks 5.1, 5.2 and 5.3, as well as their combined testing and customization. Specifically, this undertakes the consolidation of several RAWFIE components in the three tiers of the architecture.

This deliverable presents:

- The results of the integration of RAWFIE components in a unified platform and the verification of the RAWFIE system (verifying the functionalities of the several integrated components), in complement to the test and verification of individual components that is supposed to be done in WP5;
- The integration activities (required technicalities, tests) that had been done and the current results, mainly the activities that have been done to get a running system for the 1st review of the project, (i.e. obtained during the first development cycle);
- Technical issues and consolidation of the several RAWFIE components;
- Recording of the interface and verification tests and steps for supporting improvement on the RAWFIE operational platform (such as enhancements that need to be considered for the next iteration and the corresponding development plan).

Eventually, this document will report the results of the tests done at each testbed site according to the integration, deployment and testing plans defined in WP2 after the completion of each development cycle. These will include the analysis of the failures, errors, user feedback and comments to modify and improve the respective RAWFIE components and integrated system in the subsequent development cycles.

1.2 Definitions

This document makes use of a number of specific terms, which should be understood as defined below:

- Verification of a system is the task of determining that the system is built according to its specifications (functionalities according to requirements and design specifications);
- Validation is the process of determining that the system actually fulfills the purpose for which it was intended (according to the users needs);
- 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 users needs and / or performances in the field against realistic scenarios).

1.3 Relation to other deliverables

The work performed in WP6 is based on the outcome 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.

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 D4.3, with some modifications that will be highlighted in the following of the document.

D6.1 provides feedback to WP5 (based on the results of the integration tests to be taken into account in D5.3 and D5.4) for revisiting and improving the implementation of components and their interaction in the global architecture. These results are also exploited by WP3 for revising/extending the defined requirements and WP4 for revising the architecture in subsequent iterations.

Although it is coarse grain, D2.2 is used for checking the completeness of D6.1 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). D6.1 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.

D6.1 refers explicitly to the Verification tests defined in D4.3 (section 5.1) for the component testing at a high level. Nevertheless, in D6.1, the structure of the test descriptions has been slightly revised to reflect the actual emphasis of the integration process on the interfaces, dependencies and interactions between components. D6.1 deals with, and presents, the interface testing results and the high-level testing results, following the templates shown respectively in Table 1 and Table 2.

2 Integration & Testing

2.1 Approach

The objective of this activity is to produce an end-to-end operational prototype of the RAWFIE platform that is used in testing pilots in the context of this specific task and, ultimately in test cases selected through the open calls. The approach taken for the integration follows the roadmap defined in D2.2. The integration process started at the very beginning of the project inception and in its associated description of work, in which numerous design choices have guided the initial steps of the project execution. From the start, the architecture, shown in Figure 1, was progressively defined and refined leading to a number of interfaces, pre-conditions, dependencies, etc. Figure 1 provides an overview of the involved components and their interconnections. Each arrow represents an interaction point between two components, reified by interfaces in the implementation. Each interface is an elementary part of the integration, which, as for every component, is tested in pre-defined scenario.

The evaluation of the component performance and of the conformance to its definition has been first obtained by individually testing each of them independently in specific conditions, exercising their interfaces. The components defined in WP4 and developed in WP5 have also been progressively integrated into a coherent, complete and self-standing system.

As a result, the RAWFIE system, as integrated during the first phase of the project, was tested and checked against the requirements gathered in WP3. Concurrently, other test results have been obtained by:

- Use of simulations, simulated data resembling real data etc.
- Exercise of individual component interfaces
- Exercise of interfaces and components once combined, in intra-tier & inter-tier integration tests
- Integration activities performed during experimentations using the prototype
- Remote control testing activities
- Etc.

D6.1 describes the results of such tests, in particular those done during internal review of the platform and experimentation of the first prototype.

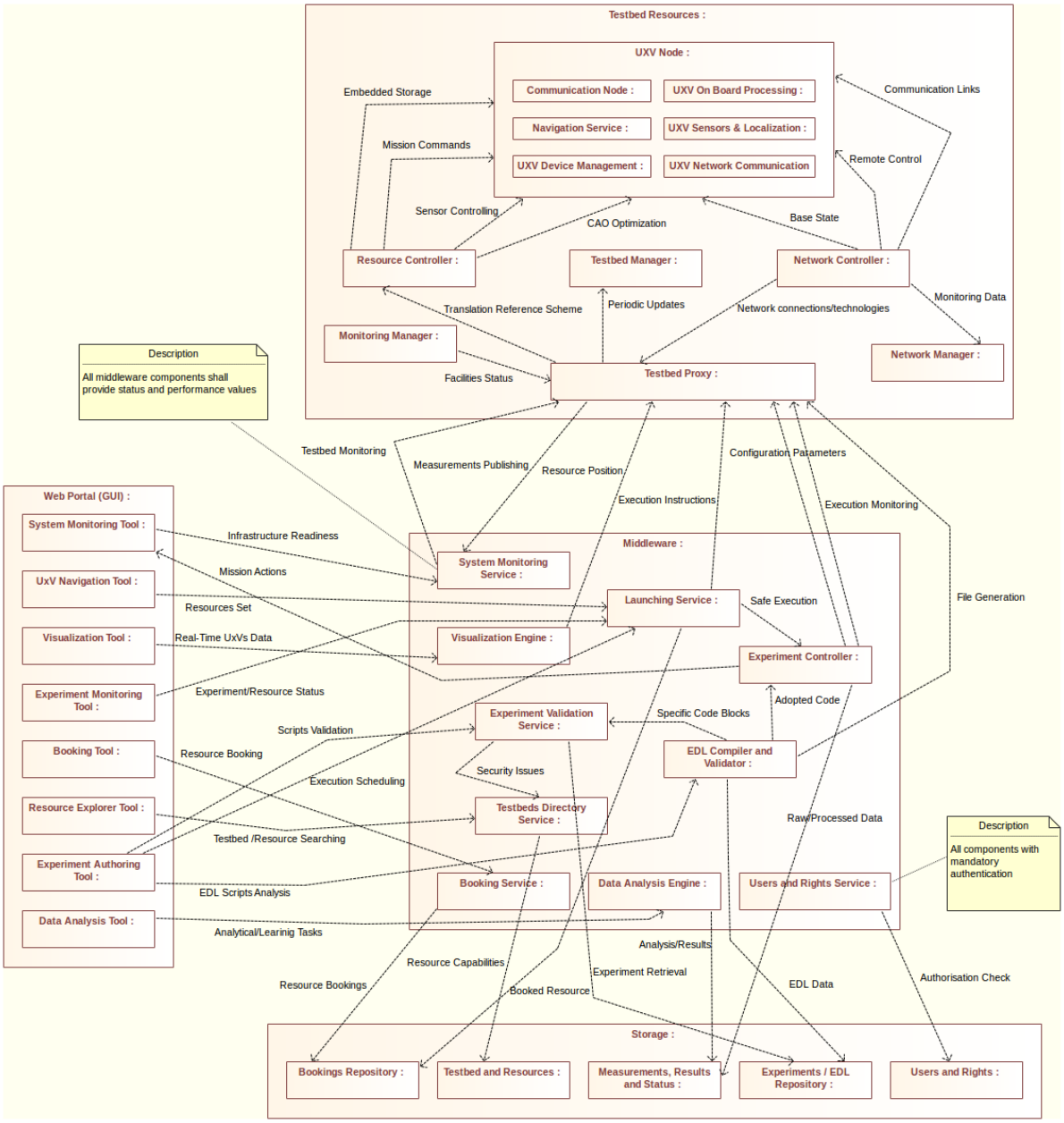

Figure 1: RAWFIE architecture (first iteration)

The following aspects represent the assets and basis, shared by all partners and work-packages, for guaranteeing efficient and flexible research and development activities across the different project cycles:

- Integration is performed at a data and interface level and by using existing tools and mechanisms (standard data representations and models, REST, AVRO, Kafka, etc.), allowing for a convenient and efficient decoupling of the components.
- In addition to provide a status of the integration, D6.1 is a testimony of the progressive yet effective installation of a systematic approach allowing for the testing of the integrated RAWFIE system at any point during the execution of the project. This includes non-regression procedures and assessments, versioning, quality-control procedure, etc.

For example, The VT – VE communication is first tested by creating stubs in the VE that provide information about the missing components and modules. They create a dummy data, that the VE can forward to the VT and the VT can display. After implementing all of the modules of the platform, these stubs are replaced by the real functions and the process is repeated again. In that way we can define whether the problem is on the VE/VT side, or it is in the other modules. The results from the different tests are observed with different tools like http requests logger, database explorers, kafka listener and others. By using them we can additionally check if the requests between the VT and VE are as specified. In case that problems of any type arise, the system is updated and the tests are executed again until there are no more problems. The same procedure is performed also for the rest of the communication to and from the VE – with the database and with other modules through kafka. The obtained results are successful for the first iteration of the platform.

Additionally non-developers perform black box testing on the VT. This includes trying the complete functionality of the VT by executing every possible functionality of the VT without knowledge of the underlying architecture, of the prerequisites of the platform and any other requirement, that if not present, could lead to problems with the platform. The successful execution of these tests guarantees tha t the VT is developed simply and intuitively and adds an additional layer of security that all bugs are fixed. This test was also successful

System integration has a prerequisite that all internal integration activities and unit tests of individual components is complete. At every phase, the successful intra-tier integration of the various subcomponents is ensured before the initiation of the required procedures to complete the inter-tier integration. System integration is tightly related to the testing and refinement tasks that will result in the different releases of the system.

Before committing the modifications and ultimately delivering the RAWFIE system to the evaluators and the customers, a number of typical situations, implemented under the form of reference scenarios described in D4.3, are systematically (re)played. Their outputs are examined to check if the functions and non-functional properties are still valid and/or within the specifications.

During the next cycle, the RAWFIE components as well as the whole system will be modified and improved according to new or updated requirements, specifications and bug fixes. The RAWFIE system will also be customised to meet the requirements of applications and customer preferences.

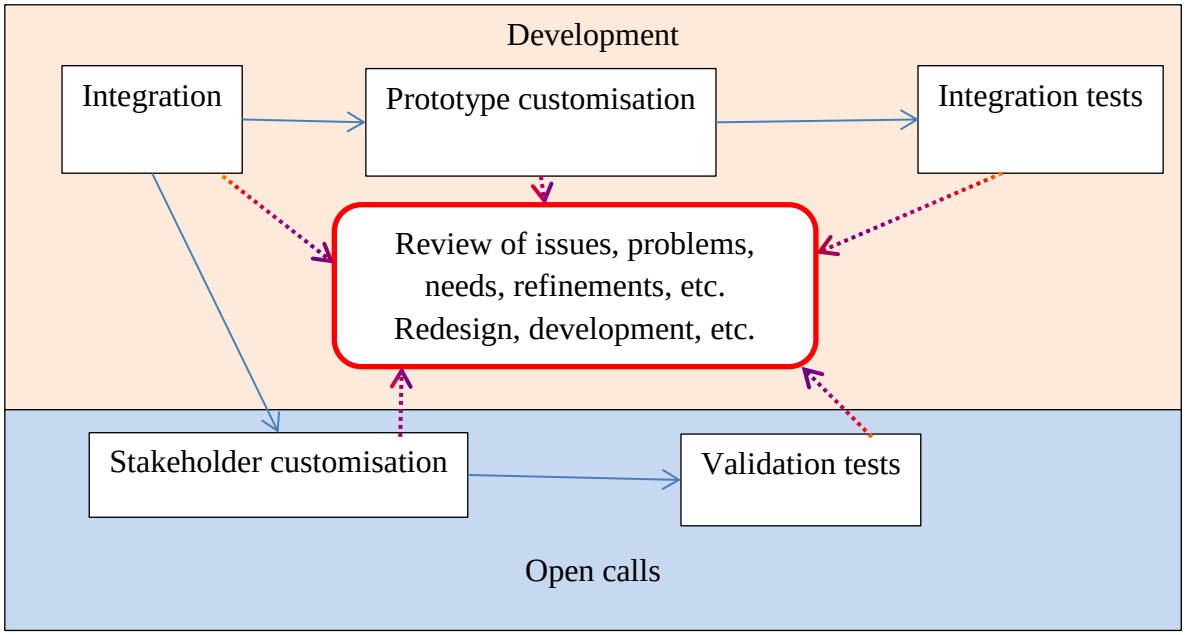

Figure 2: Integration, tests and validation process

The fact that the RAWFIE applications are highly dynamic and involve various testbeds and UxVs of different natures at different places make the systematic verification and non-regression tests complex and highly challenging. To cope with all that and the number of UxVs that must be present and operational in such RAWFIE reference scenarios, simulators can be used instead of real entities and actual deployments, in which numerous parameters are used to allow for a variety of UxV natures, behaviours, characteristics and configurations.

Ultimately, through the funded (Open Calls) and non-funded experiments several external experimenters should access and test the RAWFIE operational platform prototypes. These tests should highlight any internal processes and modules of the prototypes that need further refinements or improvements to reach to the best forthcoming interoperability of the RAWFIE infrastructure modules; in parallel, the customisations may take place, which will exercise the customisation mechanisms. All the involved technical partners analyse the user comments and adapt the respective RAWFIE components accordingly in the subsequent development cycle. The final outcome of the procedures and the participants' efforts in this task will be a stable RAWFIE platform that will be exploited from every interested experimenter.

2.2 Methodology

Integration testing includes activities where individual software modules are combined and tested as a group. It precedes validation testing and generally applies tests defined in an integration test plan to aggregates or groups of unit-tested modules with the aim to deliver as its output an integrated system ready for validation testing.

Integration activities follow the individual / unit testing activities performed (mainly in the context of WP5) on the various components defined in the architecture deliverables (WP4

D4.1 and D4.2), and are based on the integration testing plan (verification scenarios) defined in D4.3. They aim to provide sufficient proof of correctness of functionality for combinations of components at both platform and testbed tier and identify possible bugs and inefficiencies in the foreseen workflow of RAWFIE platform services usage and the testbed processes. The methodology adopted in RAWFIE for integration testing generally follows a bottom up approach, in the sense that integration activities are performed initially pairwise with test cases involving 2 components that directly communicate either synchronously or asynchronously (via message bus) and then proceeding with more extensive test scenarios involving interactions of multiple components that implement part or complete RAWFIE workflows.

The integration tests involved the following major categories:

1. **Testing of components interfaces (black box testing)**: This kind of black box testing should be performed for all components implemented in the 1st iteration cycle that provide an interface (via a REST or SOAP / RPC API) or are capable to send/receive data from Message Bus.
   An interaction matrix has been created (see Table 4) which provides a quick reference of all the interacting components (including the type of interaction) independent of the tier they exist. Based on this matrix a detailed report was compiled (see section 2.4) which elaborates on the exact interface or message exchange that was tested during integration activities.
2. **Execution/Testing of verification scenarios (1st level of white box testing)**: This step involved the execution of all the applicable (since some components were not considered for the 1st iteration) verification scenarios defined mainly in D4.3 section 5.1. Although these verification scenarios aim mainly to verify individual components' functionality in most cases, they have as pre-requisite the existence of other components (tools or services). Therefore, despite the individual component testing performed during implementation activities in WP5, the (re)execution of all these verification scenarios was deemed necessary.
3. **Execution of end to end scenarios (1st level of system testing)**: This step involved the execution of scenarios that address multiple components in all tiers and verify the behavior of the system for its expected 'real' usage (i.e. the Booking of resources and consequent execution and completion of an experiment). No such tests were prescribed/foreseen for integration testing activities during the first iteration cycle. As a consequence, this step will be done in the next cycles. It is however mentioned at this point because it is an important part of the methodology, which should not be overlooked.

Note: Performance tests and tests involving non-functional aspects of the RAWFIE system were not considered as part of the integration activities and will not be included in the present report.

Note: Because some components were not present in the first iteration, to be able to complete the integration testing activities mentioned above certain assumptions/simplifications were

made in order to meet the prerequisites needed in each test scenario. These assumptions mainly have to do with:

- The pre-existence of certain data in the RAWFIE database due to the fact that the tool/service that was responsible for inserting/updating these data was not implemented or partly implemented
- The fact that a limited number of components involved in the core experiment workflow were not considered for implementation in the 1st iteration cycle. Thus these components (involving mainly interactions via message bus) had either to be skipped during integration testing or considered to provide a default functionality

More precise information on the assumptions/simplifications made will be provided on a per test case basis in sections 2.4 and 2.5 that provide details on the testing activities.

2.2.1 Test framework

Integration of components is performed in stages:

1. *Intra-tier*: addressing activities needed to integrate and test components in the same tier (e.g. front-end, middle-tier, testbed);
2. *Inter-tier*: addressing activities needed to integrate and test components belonging to 2 different tiers;
3. *System wide*: addressing activities needed for verifying end to end interaction flows (all tiers, end-to-end integration).

Inter-tier and Intra-tier stages involved both interface testing and functional (white box) testing while the System wide stage focused only on functional aspects.

In order to allow for a common and concise way of representing the results of all kind of integration tests, two templates were used, that are shown in Table 1 and Table 2:

Table 1: template for reporting interface test results

Component: <***Component Name***>		Conducted by: <**Partner ID**>		Date: **Feb 2016**	Test Category: **Interface testing**
Preconditions		Describe any general precondition that must be present (if any)			
	Related Component	**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	<Component Name>	R	<Method Name>	Not applicable	E.g. component does not yet exists
2	<Component Name>	M-c	<Message Name>	Partial success	Message was consumed by Resource controller since Experiment Controller does not yet exists Message successfully received by receiving component
		M-c	<Message Name>	Not tested	E.g. functionality not yet supported
3	Message Bus	M-p	<Message Name>	Success	E.g. connection to database succeeded Retrieval/update/insert of information succeeded
4	<Component Name>	JDBC	<Method Name>	Fail	Describe reason of failure e.g. connection to database fail

Regarding the above template:

- For message oriented communications (where the message bus acts as intermediate) since we have producers and consumers, in the interface template we depict both of them using the convention M-c, M-p so that it is clear that the producing component sends to MessageBus and the consuming component receives the message
- For other types of synchronous interactions like REST, SOAP/ RPC, JDBC etc. it is obvious that the interface template will refer to component that initiates the communication (caller).
- Allowed status include: *Success / Partial success / Fail / Not tested / Not applicable*
- *Success* status is highlighted in green color, *Partial Success* in orange, while *Not tested / Not* applicable are identified in grey

Generally we include information regarding interactions with the message bus by both producers and consumers components. Interface of type M-p (that is the case the component acts as producer) should not include any related component (or only "Message Bus"). 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 will be received by multiple consumers and this interaction is shown 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.

The rationale of not specifying a related component when type of communication is M-p is that this kind of communication is quite loosely coupled and in general it is not easy for the producing component to know which target component will consume the message. There can be one or many components but there is no reason i.e. to create 10 rows in the producer component because the message will be consumed by 10 components.

This information is shown to the related component that acts as consumer (has type M-c).

In the case of interface testing that refers to communication between components, there are no steps here, but only *Success*, *Partial success*, *Fail* or *Not tested* with a possible remark.

Table 2: template for reporting integration scenarios test results (example adapted from D4.3 test case)

Test ID**: MB02**	Conducted by**:** **<Partner ID>**	Date**: Feb 2016**	Test Category**: Verification Tests (middle tier)**
Hardware Configuration	See section 2.3.1		
Software Configuration	See section 2.3.1		
Test Name:	***Receive resource booking notification***		
Preconditions	• The user must have a registered email account belonging to the federation • A long-term selection must be scheduled as launching selection with the previous resource booking		
Related Requirements	(may not be present in integration tests)		
Tools Used	*list any special or extra tools used beside code tests*		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Book any resource in order to carry on a certain experiment in the near future	Reservation data entries are added to the DB	Success / Partial success / Failed / Not tested / Not applicable	*list here any divergence from initial foreseen action*
2	Wait till the established date and time to be launched	-		
3	Verify that user has received the corresponding notification regarding the booking information and experiment prepared	An email is send to the user		
4				

Regarding the above template:

- HW and SW configuration may refer to RAWFIE Platform and/or testbeds. For the platform case a common configuration was used in all integration activities which is listed in section 2.3.1. For the testbeds and the UxV devices information can be found in section 2.3.4.
- The field related to requirements may be omitted in this first iteration report. The rationale is that integration tests generally are component level specific activities. However, during the integration period (January – February 2016) the only available requirements were the ones of D4.1 which were mainly high level system requirements that aim to outline the overall behavior, services and performance characteristics that the RAWFIE platform architecture should adhere to.
- Although the *action* field usually refers to a step that must be user initiated in certain cases (to better illustrate the flow of activities) it is possible to include there activities that are performed by a component (once or on a periodic basis) as a result of previous resultField *expected result* might include a single or multiple outcome(s). In the latter case the outcomes should be numbered accordingly in order to easily distinguish them

- In the verification test, we use the nearly the same status labels *Success / Partial success / Failed / Not tested / Not Applicable* (keeping in mind that partial success can apply only in situation where a single step entails multiple results).

This addresses the verification of the component and system beyond the syntactical and static analysis of the correct combination and matching of inter-component interfaces, initial requirements and pre-conditions.

2.3 Integration environment setup (UoA)

This section describes the environment (depicted in Figure 3) used for the integration of the RAWFIE components and sub-systems and the subsequent testing. This may include 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.

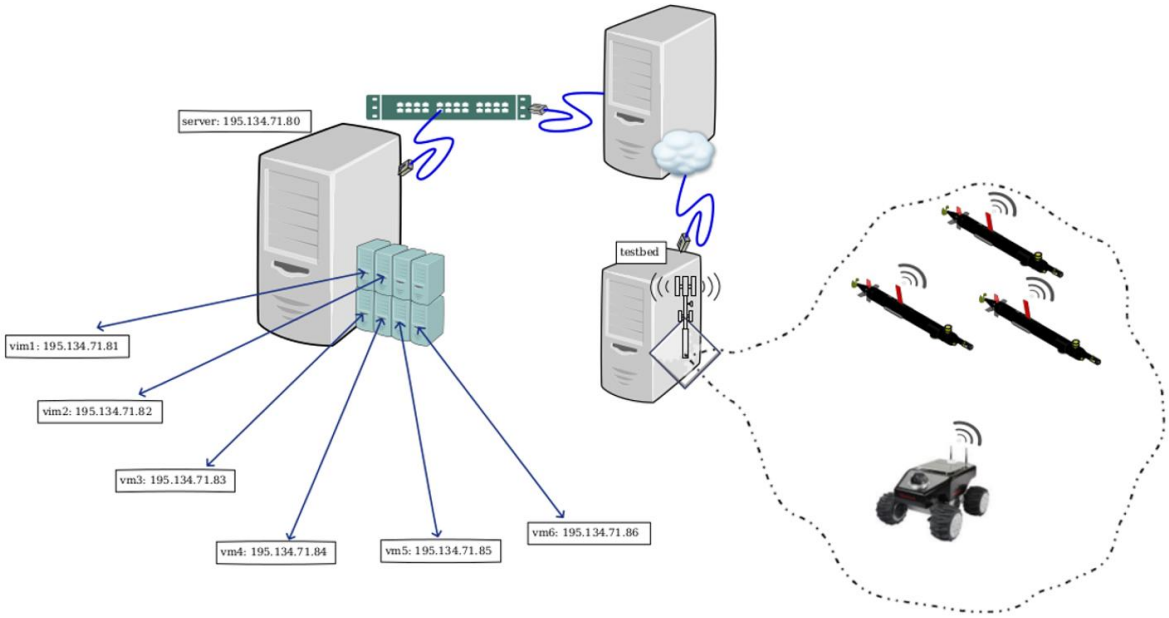

Figure 3: 1st RAWFIE environment integration

2.3.1 ICT infrastructure (UoA)

Server Hardware Configuration (HW Configuration)

The RAWFIE platform infrastructure environment is based on rack-mount servers based on dual Xeon E5-2603v2 2011 processors and equipped with RAID 1 SATA HDDs. Large amount of RAM memory (16 GB or more) supports the virtualization of RAWFIE services.

The infrastructure for the first development phase of the RAWFIE platform is built on six virtual machines (VMs). The main software for all the VMs is Ubuntu 14.04.3 LTS. The access of the VMs is done through the SSH protocol at port 22. Users participating in the development phase have access in the VMs with the same account as these have been set up in an LDAP server.

Server Software Configuration (SW Configuration)

The six VMs run additional software as described below:

- VM1
  - **Postgresql 9.4.5:** PostgreSQL is a powerful, open source object-relational database system. This is where the main database of the RAWFIE platform is setup. It includes the Postgresql user configuration for programmatical access to the repository.
  - **PostGIS:** PostGIS is a spatial database extender for PostgreSQL object-relational database. It adds support for geographic objects allowing location queries to be run in SQL.
- VM2
  - **JVM environment: Java 8 Oracle, where Tomcat8 server runs**
  - **Tomcat8:** Servlet and Web App Container providing the execution environment for the following RAWFIE services:
    - **Web Portal:** This is the main portal of the RAWFIE platform and is a Java Servlet based application
    - **Testbed Directory Service:** A RESTful web service providing the software interfaces for getting access to information about Testbeds and Resources from the PostgreSQL database.
    - **Experiment Authoring Tool:** provides several modules used for the definition and authoring of experiments,
    - **Launching Service:** is responsible for initiating *StartExperiment* requests either manually or on a scheduled basis. In the 1st iteration only manual experiment initiation will be available.
    - **Visualisation Tool:** is a web based application integrated into the RAWFIE web portal in order to support visualization of predefined data from EDL and visualisation of a real time data from UxVs.
- VM3
  - **Icingaweb2:** Icinga is a scalable and extensible monitoring system. A local postgresql has been used for the ease of Icinga installation.
  - **JNRPE:** JNRPE is designed to allow the execution of Nagios plugins based on Java for monitoring local resources on remote machines.
  - **Tomcat 7:** Tomcat is used for the MkLivestatusApiProxy and SystemMonitoringService application.
  - **Java 8:** Java is used for both JNRPE and Tomcat.
- VM4
  - **Geoserver2.8.1:** GeoServer is a Java-based software server that allows users to view and edit geospatial data.
  - **Tomcat8:** This is needed for the Geoserver application and hosts the following service:
  - Visualisation Engine: is responsible for receiving the data from the movement of the UxVs, updating, converting and adjusting it and sending

it to the visualisation tool so that it can be presented to the user and hosts the following service:

- Visualisation Engine: is responsible for receiving the data from the movement of the UxVs, updating, converting and adjusting it and sending it to the visualisation tool so that it can be presented to the user.

- **Java 7 open jdk:** This is used for both Tomcat and Geoserver support

- VM5
  - **Confluent-platform-2.11.5:** The Confluent Platform is a stream data platform that provides access to the RAWFIE Message Bus. Confluent platform is expected to offer all components (Apache Kafka broker, Apache Kafka clients, Schema Registry) needed to realise a scalable, high throughput communication bus between components. Confluent is a collection of services, tools, and guidelines for making all of RAWFIE's data available as real-time streams.
  - **Docker:** Docker allows you to package an application with all of its dependencies. It is mainly used to provide local UxVs software simulators providing sensor measurements in order to stress the Kafka installation.
  - **Java 8**
- VM6
  - **Phppgadmin:** A web-based GUI for accessing the *rawfie_db* at VM1

2.3.2 Data repositories

A PostgresSQL DB was installed in VM1 with the name *rawfie_db*. The schema has been described in the D5.1 and is consistent to the specification of the RAWFIE data model design. For the first development period the following tables were used by RAWFIE component. These tables offered information to the respective components about users, experiments, reservation status and measurements. The following table contains the name of the entity that is defined in RAWFIE data model, the components that utilize this information and the status of usage (if each component was used in the first development period or not). More details for the repositories and their attributes can be found in section 2.3 - "RAWFIE Data Model Design" of deliverable D5.1.

Table 3: Usage status of Rawfie components

Entity	Rawfie Components	Used
User	Web Portal, LDAP client, Users and Rights Service, Visualisation Engine	Not
VT_Settings	Visualisation Engine	Not
Experiment	Booking Service, Experiment Monitoring Tool, Launching service, Visualisation Engine	Yes
Experiment Execution	Experiment Monitoring Tool, Launching Service, Visualisation Engine	Yes
ExperimentLog	Testbed Manager	Not
ExperimentStatus	Visualisation Engine	Yes

Algorithms	Experiment Validation Service, EDL Compiler	Yes
EDLScript	Experiment Authoring Tool, Experiment Validation Service, EDL Compiler, Visualisation Engine	Yes
Reservation	Booking Service, Experiment Monitoring Tool	Yes
ReservationItem	Booking Service, Experiment Monitoring Tool, Launching Service, Visualisation Engine	Yes
Testbed	Testbed Directory Service, Resource Explorer Tool, Testbed Manager	Yes
Resource	Testbed Directory Service, Resource Explorer Tool, Visualisation Engine	Yes
ExperimentResourceConfig	Testbed Manager	No
ConfigParameters	Testbed Manager	No
Message	Resource Controller	No
Sensor	Experiment Validation Service, EDL Compiler, Visualisation Engine	Yes
Health_status_lut	System Monitoring Tool/Service	Yes
Connection	Experiment Validation Service, EDL Compiler	Yes

Data used for integration was mainly inserted manually by issuing SQL inserts, as several tools to this via the RAWFIE Web Portal will be implemented in the next interaction phase.

2.3.3 Message Bus data format

The data model on the message bus is a key element for the integration at all levels and interoperability of component instances. The schema of all messages was defined via AVRO schemas described in section 4.3 1 of deliverable D5.1 that were generated out of Java classes. The schemas are managed in a GIT repository, so that all developers can access and use them in their components to implement the integration of the message bus.

2.3.4 Testbeds and configurations

This section describes the testbeds used for the real life tests and the integration between testbeds (UxV and associated infrastructure) and the cloud (services and UI tools). It describes what integration activities were carried out so far.

A first configuration involved UxV simulators: MST on-board software DUNE was used to simulate MST vehicles and the GAZEBO simulator used within the ROS users community to simulate Robotnik's vehicles. This first step was necessary to test the interaction of RAWFIE components without the need of actual robots.

The efforts have been directed to test the interface developed with the common frame for RAWFIE, especially the message bus and the customized messages, commands and data format in general. Eventually, the following items were developed, tested, and integrated:

- Message definition and serialization using the Apache Avro data serialization system
- Publishing/subscribing messages to/from the Kafka message bus
- Reachability of nodes and services
- The Kafka-Robot adapter to support Robotnik's robots

- The OceanScan Proxy service to support MST vehicles

A second step consisted of replicating this working configuration in the real robots, in particular Robotnik's, as no modifications were made to the on-board software of MST vehicles and the OceanScan Proxy treats simulated and real vehicles indistinguishably as both have the exact same API. Robotonik's integration effort involved using Kafka Python and Python confluent-schema-registry stack. MST integration effort involved using Kafka Java 0.8.2.1, Kafka Scala 2.10, and Confluent 1.0. Both integration efforts used Apache Avro 1.7.7

The temporary infrastructure of the Porto testbed, whose network topology is depicted in Figure 4, comprised the following components:

- One Robotnik's SummitXL UGV equipped with temperature and pressure sensors, laser scanner, and cameras. This robot connected to the testbed infrastructure using a auxiliary 2.4 GHz 802.11n radio deployed specifically for the integration tests. This asset is represented as "UGV 0" in the network topology diagram.
- Two Light Autonomous Underwater Vehicles (LAUVs) equipped with Conductivity, Temperature, Rhodamine Dye, Chlorophyll, Phycocyanin, Phycoerythrin, and Fluorescein sensors; active dual frequency sonar and high definition camera. Communication with the Manta gateway was performed using a 2.4 GHz 802.11n radio link and 25 kHz acoustic modem. These assets are represented as "AUV 0" and "AUV 1" in the network topology diagram.
- One Durius Autonomous Surface Vehicle (ASV) equipped with camera. Communication with the Manta gateway was performed using a 2.4 GHz 802.11n radio link. This asset is represented as "ASV 0" in the network topology diagram.
- One Manta gateway with WHOI Micromodem Acoustic Modem and one 2.4 GHz 802.11n radio with an omnidirectional antenna. This asset is represented as "GW" in the network topology diagram.
- One 2.4 GHz 802.11n radio with builtin 90º sector antenna, connected to the MST network infrastructure and to the Internet through a firewall. These assets are represented in the network topology diagram as "LAN-GW", "LAN", and "Firewall" respectively.

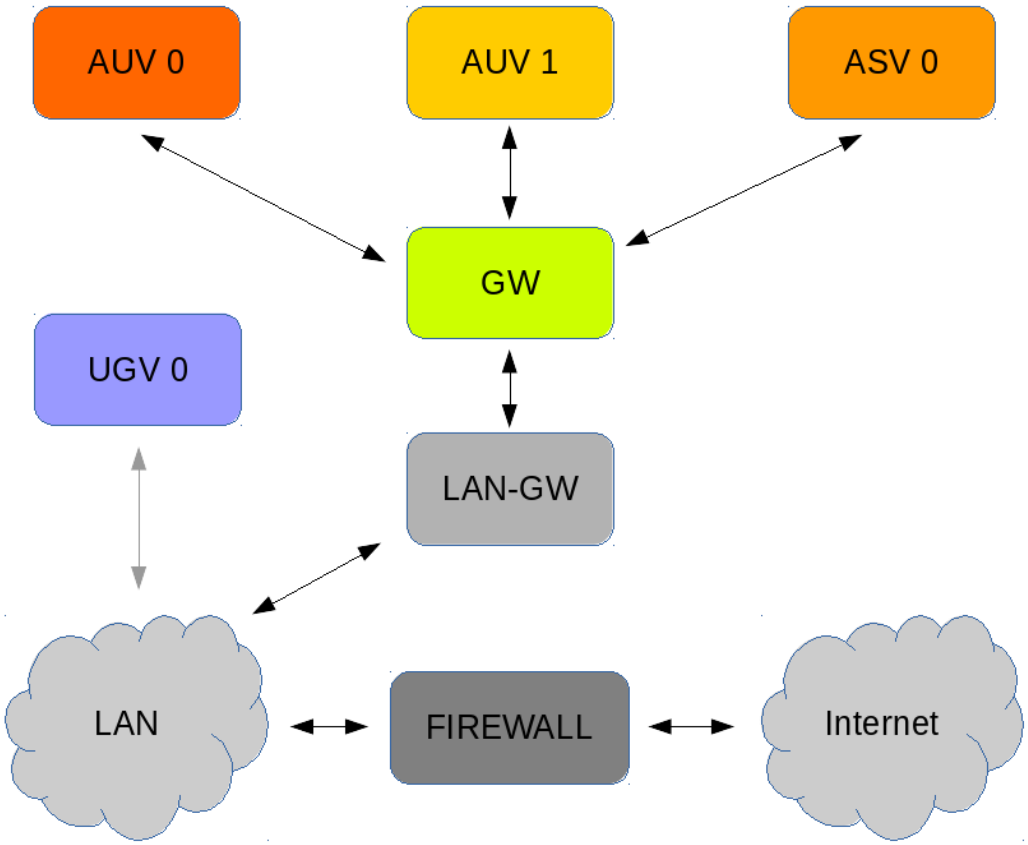

Figure 4: Architecture of the UUV+UGV setup

2.4 Integration Test Results

This paragraph provides details on the testing activities performed on components that have been grouped into specific hardware and software configurations.

The list of components that were integrated and for which the interfaces between components were tested is given in Table 4. In Table 4, 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. In other words, the cell represents what component in the respective row is calling the interface of the component that is specified in the respective column: “the row item calls (or triggers) the item in the column”.

Table 4: interface interaction matrix

r	Web Portal	Wiki	Resource Explorer Tool	Booking Tool	Experiment Authoring Tool	Experiment Monitoring Tool	System Monitoring Tool	UxV Navigation 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	Visualisation Engine	Master Data Repository	Users & Rights Repository	Measurements Repository	Results Repository	Testbed Manager	Monitoring Manager	Network Controller	Resource Controller	Navigation Service	UxV node	UxV - Network communication	UxV – Sensors & Localization	UxV – On board storage	UxV – On board processing	UxV – Device management	UxV – Proximity component
Web Portal																						O														
Wiki																																				
Resource Explorer Tool				R															R																	
Booking Tool														R					R																	
Experiment Authoring Tool											O	O			R							O														
Experiment Monitoring Tool															R			R	R			O														
System Monitoring Tool																		R																		
UxV Navigation Tool																														M	M	M				
Visualization Tool																					O									M	M	M		M		
Data Analysis Tool																	M																			
EDL Compiler & Validator					O																															
Experiment Validation Service					O																															
Users & Rights Service																						O														
Booking Service																						O														
Launching Service					R							R				M						O														
Experiment Controller															M						M					M										
Data Analysis Engine										M															O											
System Monitoring Service																																				
Testbeds Directory Service																						O														
Accounting Service																																				
Visualisation Engine										O						M						O			O											
Master Data Repository					O																O															
Users & Rights Repository																																				
Measurements Repository																																				
Results Repository																					O															
Testbed Manager																	M																			
Monitoring Manager																																				
Network Controller																																				
Resource Controller																									M							M	M	M		
Navigation Service																																				
UxV node								M	M																M			M								
UxV - Network communication								M	M																			M								
UxV – Sensors & Localization								M	M																			M								
UxV – On board storage																																				
UxV – On board processing									M																											
UxV – Device management																																				
UxV – Proximity component																																				

MessageBus	M
Rest	R
SOAP	S
Other	O
Success	
Partial Success	
Fail	
Not Tested	
Not applicable	

Table 5 - 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

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.

Figure 2 shows the complete architecture devised the first phase of the project. The components enclosed in the area have been prototyped, integrated and tested.

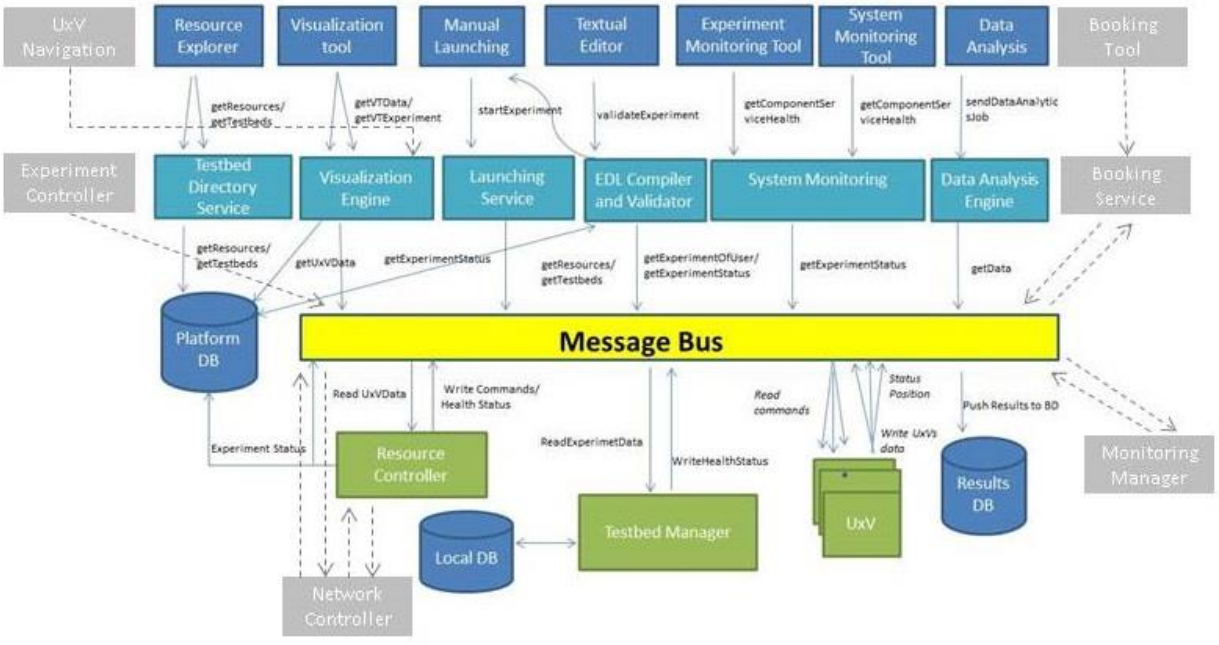

Figure 5: RAWFIE architecture (first version) and current integration coverage

2.4.1 Front-end integration

In the front-end tier, the following components were integrated:

- Web Portal
  - Login via User and Rights Service
  - Tools mentioned below where integrated into the website.
- The System Monitor Tool:
  - Displayed successfully the status of the testbed and the servers of the cloud environment. Status information was delivered by the System Monitoring Service
- Visualisation Tool
  - Checking that the interaction from VT delivers the expected results – listing the available experiments, starting the visualisation of an experiment, showing the data of the UxVs on the map and showing the movement and the sensor values being updated in real time
- Resource Explorer Tool
  - Displayed successfully the resource data delivered by the Testbed Directory Service
- Data Analysis Tool
  - Displays the UI to write manual ML job (or use predefined one)
    - This is currently in progress
  - Access to results database [via graphite UI] to show results of previous experiments
- Experiment Authoring Tool (Textual EDL Editor)
  - Loading, editing and saving of EDL scripts worked
  - EDL Compiler and Validator integration worked: Highlighted syntactical and semantic errors
  - Manual launching not implemented. Launching of scripts done via internal API.

Table 6: Test of the Web portal interfaces

Component***: Web Portal***		Conducted by**: Fraunhofer**	Date**: Feb 2016**		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

Table 7: Test of the Resource explorer interfaces

Component: ***Resource Explorer***		Conducted by: **Fraunhofer**		Date: **Feb 2016**		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	getResources	Success		More filtering criteria for the selection of resources/UxVs may be useful in a subsequent iteration
2			getAllResources	Success		Got all resources/UxVs
3			getTestbeds	Success		More filtering criteria for the selection of testbeds may be useful in a subsequent iteration
4			getAllTestbeds	Success		Got all testbeds
5	Booking Tool	HTTP	Redirect to page	Not tested		Booking Tool was not implemented

Table 8: Test of the System Monitoring Tool interfaces

Component: ***System Monitoring Tool***		Conducted by: **Fraunhofer**		Date: **Feb 2016**		Test Category: **Interface testing**
Preconditions		System Monitoring Service collected some data				
Related Component		**Type**	**Message or API Call**	**Status**		**Remarks/comments**
1	Testbeds Directory Service	REST	getComponentServiceHealths	Success		Got all health statuses

Table 9: Test of the Visualisation Tool interfaces

Component: ***Visualisation Tool***		Conducted by: **Epsilon**		Date: **Feb 2016**		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	Websocket	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 10: Test of the Data Analysis Tool interfaces

Component: ***Data Analysis Tool***	Conducted by: **HESSO**	Date: **Feb 2016**	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		

	Related Component	Type	Message or API Call	Status	Remarks/comments
1	Data Analysis Engine	M-c	buildJob()	Not tested	Working on the interfacing of the UI with the Bus stream.
2	Results Database	REST	render()	Success	Graphite is able to be queried via REST and plots results
3	Data Analysis Engine	M-p		Success	Send the Analytics jobs to the Data Analysis Engine through the Kafka message bus

Table 11: Test of the Experiment Authoring Tool interfaces

Component: ***Experiment Authoring Tool***	Conducted by: **UoA**	Date: **Feb 2016**	Test Category: **Interface testing**
Preconditions	Users are entered in the RAWFIE Web Portal		

	Related Component	Type	Message or API Call	Status	Remarks/comments
1	Textual and Visual editors	-	-	Success	Textual and visual editors are smoothly incorporated in the RAWFIE Web Portal
2	Launching service	REST	manualStart	Success	Launching tool is correctly informed about the ID of the experiment that will be executed
3	Experiment validation service	-	-	Success	Compilation and validation are smoothly executed in the authoring tool

Missing Components

The following components are not yet implemented and they were not tested:

- Experiment Monitoring Tool
- Booking Tool
- UxV Navigation Tool

They will be implemented in the next implementation iteration.

Table 12: Interface test of the Booking Tool

Component***: Booking Tool***		Conducted by**:**		Date**:**		Test Category**: interface testing**
Preconditions		• User must be logged in • UxV resources must be present in a testbed and advertised to the platform (browseable by the resource explorer tool) • Booking Service must be up and running • Testbed Directory Service must be up and running				
	Related Component	**Type**	**Message or API Call**	**Status**	**Remarks/comments**	
1	Booking Service	R	addBooking	Not tested	Booking Tool not implemented	
2		R	editBooking	Not tested	Booking Tool not implemented	
3		R	deleteBooking	Not tested	Booking Tool not implemented	
4		R	getBookings	Not tested	Booking Tool not implemented	
5		R	getBooking	Not tested	Booking Tool not implemented	
6	Testbed Directory Service	R	getResources	Not tested	Booking Tool not implemented	

2.4.2 Middle tier integration

In the front-end tier, the following components were implemented and integrated:

- System Monitoring Servcie:
  - Status data from the cloud servers and testbeds where collected successfully
- Testbed Directory Service
  - Data from the Master Data repository war accessible via the service
- EDL Compiler and Validator:
  - Validated scripts: Delivered error messages for incorrect ones
  - Compiled Scripts for later execution
- User & Rights Service
  - Checking of Login credentials loaded from the User & Rights repository workd
  - Checking of roles/rights not tested with other components
- Data Analysis Engine
  - Receive a job description from the Data Analysis Tool and build a job that can be passed to Spark.
  - Provides a mechanism to return result status to the Data Analysis Tool.
- Launching Service
  - Manual Start of an experiment
  - Generation of ExperimentStartRequest, ExperimentCancelRequest JSON messages and communication with MessageBus
- Visualisation Engine
  - Checking the communication with the database – for reading and writing sensor data, list of experiments, users etc. to and from the database

- Checking the communication with kafka for commucation with the other modules – obtaining real time data of the movement of the UxVs like position and sensor data
- Checking the communication between the VT and VE – all requests from VT should be handled properly and the results from kafka or the database, should be sent back to the VT

Table 13: Test of the Testbed Directory Service interfaces

Component: ***Testbed Directory Service***		Conducted by: **IES**		Date: **Feb 2016**	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 RepositoryHandler class, to support the searchTestbed() REST API (get details about a specific testbed)
8			fetchTestbeds	Success	Operation performed by a RepositoryHandler class, to support the getTestbeds() REST API (get details about the specified testbeds)
9			fetchResource	Success	Operation performed by a RepositoryHandler class, to support the searchResource() REST API (get details of a specific resource from a specific testbed)
10			fetchResourcesTestbed	Success	Operation performed by a RepositoryHandler class, to support the getResources() REST API (to get details of all resources from a specific testbed)
11			fetchResourcesAvailable	Success	Operation performed by a RepositoryHandler class, to support the getAvailableResources() REST API (get details of all resources which are AVAILABLE for booking tests from a specific testbed)

Table 14: Test of the Visualisation Engine interfaces

Component: ***Visualisation Engine***		Conducted by: **Epsilon**		Date: **Feb 2016**	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 (PostgreSQL database)	JDBC	SQL	Success	Get Experiment Status
	Message Bus		ExperimentStart(Exec Id, Script) Read UxVStatus Read UxVActual Position Read UxVCommands	Success	Real data from the devices
2	Visualisation Tool	Websocket	startExperiment	Success	Connect to the visualisation engine and retrieve all the information about an experiment and get data for the movement of the UxVs
3			stopExperiment	Success	Stop the visualisation of an experiment
4			getExperiments	Success	List all available experiment for the user
5			getExperimentDetails	Success	Get the details for an experiment that the user wants to visualise
6	Experiment Controller	M-c	getGoTo	Success	Get the Goto Commands, Experiment Controller is not yet implemented so we consume the message from the Resource Controller
7	UxV Node	M-c	getUxVData	Partial Success	Get the location and sensor data from the UxVs. Not all sensor data is implemented yet.

Table 15: Test of the Data Analysis Engine interfaces

Component: ***Data Analysis Engine***		Conducted by: **HESSO**		Date: **Feb 2016**	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.			
	Related Component	**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	Data Analysis Tool	M-c	sendJob()	Not tested	Working on the interfacing of the UI with the Bus stream.
2	Schema Registry	R	/subjects	Success	Successfully iterate over all schemas

Table 16: Test of the Launching service interfaces

Component: ***Launching Service***		Conducted by: **HAI**		Date: **Feb 2016**	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 (ExperimentStartRequest topic, ExperimentCancelRequest topic)			
	Related Component	**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	Experiment Validation Service	R	validateExperiment	Not tested	Experiment Validation Service does not yet exists
2	Experiment Controller	M-p	ExperimentStartRequest	Success	Message was sent successfully to Message Bus. However, it was consumed and handled by Resource controller since Experiment Controller does not yet exists
3		M-p	ExperimentCancelRequest	Success	Message was sent successfully to Message Bus However, there is no component yet implemented to consume and handle the message
4	Master Data Repository	JPA/JDBC	Database Interaction	Success	Connection to database succeeded Retrieval/update/insert of information succeeded

2.4.2.1 Missing components

The following components are not yet implemented and they were not tested nor integrated. They will be considered for integration and test in the next implementation iteration. Nevertheless, interface tests have been defined, as reflected in the tables below.

- Booking Service
- Experiment Controller

Table 17: Test of the Booking Service interfaces

Component: ***Booking Service***		Conducted by:		Date:	Test Category: **interface testing**
Preconditions		• User must be logged in • UxV resource info must be present in a Master Data Repository			
	Related Component	**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	Master Data Repository	JPA/JDBC	Database call (insert)	Not tested	Booking Service not implemented
2		JPA/JDBC	Database call (update)	Not tested	Booking Service not implemented
3		JPA/JDBC	Database call (delete)	Not tested	Booking Service not implemented

2.4.3 Testbed integration

The test of the interfaces of the different testbed components concerns:

- The Tesbed Manager

- Interface with the System Monitoring Service is ok,
- Implemented interfaces with the Experiment Controller are ok, although they may be improved,
- Others are not implemented yet.

- The Resource Controller
  - Implemented interfaces with the Message bus are ok.

Table 18: Test of the Tesbed Manager interfaces

Component: ***Testbed Manager***		Conducted by: **HAI**		Date: **February 2016**		Test Category: **interface testing**
Preconditions		• Apache Kafka 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	ExperimentStatus	Not tested		Resource Controller does not produce ExperimentStatus message yet
3	UxV Node	M-c	UxVHealthStatus	Not tested		UxV Node does not produce UxVHealthStatus message yet
4	Experiment Controller	M-c	ExperimentStart	Success		Experiment Controller does not yet exists - message sent from Launching Service
5		M-c	ExperimentStop	Not tested		Experiment Controller does not yet exists – message not yet implemented
6		M-c	ExperimentCancel	Success		Experiment Controller does not yet exists - message sent from Launching Service

Table 19: Test of the Resource Controller interfaces

Component: ***Resource Controller***		Conducted by: **CERTH**		Date: **Feb 2016**		Test Category: **interface testing**
Preconditions		• Apache Kafka properly configured, up and running Related components must be up and running				
Related Component		**Type**	**Message or API Call**	**Status**		**Remarks/comments**
1	Message Bus	M-p	WriteHealthStatus	Not tested		Send and receive real-time information to resources
		M-p	WriteUxVCommands	Success		Send and receive real-time information to resources
		M-p	WriteExperimentStatus	Not tested		Resource Controller does not write Experiment status yet
		M-c	ReadUxVStatus	Not tested		Resource Controller does not read UxV status yet
		M-c	ReadUxVLocation	Success		Resource Controller is able to read the actual position of the vehicles

Regarding the UxV's, the following components were integrated:

- UxV Node
  - Message bus adaptor working
  - Robots accepting waypoints and commands.
  - Robots publishing localization and odometry
- UxV Sensor&Localization
  - Interface to sensors working
  - Publishing values and identifying the sensor

Table 20: Test of the UxV Node interfaces

Component***:UxV Node***	Conducted by**: Robotnik, MST**	Date**: Feb 2016**	Test Category**: interface testing**

Preconditions	• A server running the Confluent platform
	• Robotnik's 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	Partial Success	Tested with success by MST
3			Abort	Partial Success	Tested with success by MST
4			Location	Success	Without GPS specifying an origin of coordinates is needed.
5	Visualization Tool	M-c	Location	Partial Success	Visualization indoors needs revision to offer a descriptive environment
6	Data Analytics	M-c	SensorReadingScalar	Partial Success	Tested Temperature, Salinity, Conductivity, and SoundSpeed with success
7			Current	Partial Success	Tested with success by MST
8			Voltage	Partial Success	Tested with success by MST
9			StorageUsage	Partial Success	Tested with success by MST
10			FuelUsage	Partial Success	Tested with success by MST
11			CpuUsage	Partial Success	Tested with success by MST
12			SensorInfo	Partial Success	Tested with success by MST

2.4.4 Inter-tier integration

Components belonging to different tiers may communicate also through the Message-bus or other external means.

Table 21: Test of the EDL Compiler and Validator interfaces

Component***: EDL Compiler and Validator***		Conducted by**: UoA**		Date**: Feb 2016**	Test Category**: Interface testing**
Preconditions		Users are entered in the RAWFIE Web Portal			
Related Component		**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	Textual and Visual editors	-	-	Success	Textual and visual editors smoothly communicate with the validator

2.4.5 End-to End Integration

Table 22 shows an end-to-end integration scenario with the supported functionalities of the first implementation cycle. The steps of this integration offer the means for experiment authoring, deployment, execution and data analysis. The same scenario was successfully performed on UxV simulators, three UUVs of MST and on one UGV vehicle of Robotnik.

Table 22: Test of the interfaces involved in end-to-end integration

Component: **ALL**			Conducted by**: Partners**	Date**: February 2016**		Test Category**: interface testing end-to-end**
Preconditions			• Apache Kafka properly configured, up and running • Related components must be up and running • 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) • Users are entered in the User & Rights Repository			
Component		**Related Components**	**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	Web Portal	User & Rights Repository	LDAP	Lookup	Success	Experimenter logins through the web portal
2	Resource Explorer	Testbeds Directory Service	REST	getAllResources getAllTestbeds	Success	The Experimenter checks available Testbeds and Resources
3	Experiment Authoring Tool	Textual and Visual editors		-	Success	Experimenter writes, validates and launches an experiment.
		Launching service	REST	manualStart	Success	
		Experiment validation service		-	Success	
4	Resource Controller	UxV Node	Message Bus	WriteUxVCommands,, ReadUxVLocation	Success	Resource Controlelr starts an experiment. RC sends commands to UxVs and receives real-time information
5	UxV Node	Resource Controller	Message Bus	ReadUxVCommands,, WriteUxVLocation	Success	GPS coordinates accuracy and threshold for next waypoint needs to be configured
6	Visualization Tool	Visulization Engine	Websocket	startExperiment, stopExpement,, getExperiments, getExperimentDetails	Success	Experimenter sees information on runnning experiment (e.g. resources waypoints) through the Web Portal
		Resource Controller UxV Node	Message Bus	getGoTo	Success	
7	Data Analysis Tool	Data Analysis Engine	Message Bus	-	Success	Experimenter performs outlier detection through the data analytics tools Send the Analytics jobs to the Data Analysis Engine through the Kafka message bus

2.5 Verification scenarios results

In this section, the results of the executed verification scenarios of D4.3 (chapter 5) are explained. The template table, given and explained in section 2.2.1, was extended to better visualise the scenario steps and the results of them.

2.5.1 Web Portal (Graphical User Interface)

2.5.1.1 Web Portal

Table 23: Verification test of the Web Portal - Login/ Logout

Test ID: **WP01**	Conducted by: **Fraunhofer**	Date: **Feb 2016**	Test Category: **Verification Tests (front end tier)**
Hardware Configuration	See section 2.3.1		
Software Configuration	See section 2.3.1		
Test Name:	***Web Portal - Login/ Logout***		
Preconditions	• User entered in the User & Rights repository		
Related Requirements			
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 24: Verification test of the Web Portal – Language selection

Test ID: **WP02**	Conducted by: **Fraunhofer**	Date: **Feb 2016**	Test Category: **Verification Tests (front end tier)**
Hardware Configuration	See section 2.3.1		
Software Configuration	See section 2.3.1		
Test Name:	***Web Portal – Language selection***		
Preconditions	• Translation available		
Related Requirements			
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	Partial success	Language is changed, but only a few text are translated (missing translations)

2.5.1.2 *System Monitoring Tool*

Table 25: Verification test of the Visualisation of system and UxV health status

Test ID: **SMT01**	Conducted by: **Fraunhofer**	Date: **Feb 2016**	Test Category: **Verification Tests (front end tier)**
Hardware Configuration	See section 2.3.1		
Software Configuration	See section 2.3.1		
Test Name:	***Visualisation of system and UxV health status***		
Preconditions	• connection to the System Monitoring Service (may not be necessary if System Monitoring Service collects all necessary data anyway) • administrative knowledge about the system state needed on user side (to check results)		
Related Requirements			
Tools Used	• Browser		

Step	Action	Expected Result	Status	Remarks
1	user opens System Monitoring Tool in the Web Portal	the System Monitoring Tool displays views with status of, middleware components, testbeds components, UxVs components	Partial success	Servers and Testbeds displayed. UxVs did not send status information (to be implemented)

2.5.1.3 *Resource Explorer Tool*

Table 26: Verification test of the Browse testbeds and UxVs and start booking

Test ID: **RET01**	Conducted by: **Fraunhofer**	Date: **Feb 2016**	Test Category: **Verification Tests (front end tier)**
Hardware Configuration	See section 2.3.1		
Software Configuration	See section 2.3.1		
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			
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 selects a testbed	Resource Explorer Tool displays all testbed details and a list of available UxVs	Success	
3	user selects a UxV	Resource Explorer Tool displays all UxVs details	Success	
4	user starts booking		Not tested	Not implemented

2.5.1.4 *Experiment Monitoring Tool*

Table 27: Verification test of the Visualisation of experiment status

Test ID: **EMT01**	Conducted by: **Fraunhofer**	Date: **Feb 2016**	Test Category: **Verification Tests (front end tier)**
Hardware Configuration	-		
Software Configuration	-		
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			
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	Not tested	Not implemented
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)	Not tested	Not implemented
4	user starts booking		Not tested	Not implemented

2.5.1.5 *Booking Tool*

Table 28: Verification test of the Visualisation of booking status

Test ID: **BT01**	Conducted by:	Date:	Test Category: **Verification Tests (middle tier)**
Hardware Configuration	-		
Software Configuration	-		
Test Name:	***Visualisation of booking status***		
Preconditions	• connection to the Booking Service ok • user opened Booking Tool though the Resource Explorer Tool (selected UxVs as parameter)		
Related Requirements			
Tools Used			

Step	Action	Expected Result	Status	Remarks
1	user opens Booking Tool though the Resource Explorer Tool (selected UxVs as parameter)	Navigation to Booking Page	Not Tested	Involved components not implemented
2	Booking Tool displays a calendar view with the dates where the UxVs are already reserved	The reserved dates should completely reflect all reservations.	Not Tested	Involved components not implemented

Table 29: Verification test of the Booking on free date

Test ID**: BT02**	Conducted by**:**	Date**:**	Test Category**: Verification Tests (front end tier)**
Hardware Configuration	-		
Software Configuration	-		
Test Name:	***Booking on free date***		
Preconditions	• connection to the Booking Service ok • user opened Booking Tool though the Resource Explorer Tool (selected UxVs as parameter) • The selected resource should not be booked (for the given interval)		
Related Requirements			
Tools Used			

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	User selects "New booking" from the UI	Booking Tool shows booking form	Not Tested	Involved components not implemented
2	User enters data (name, time, comments) and a date where no reservation exist and submits the form	A Booking Request is initiated to the Booking Service	Not Tested	Involved components not implemented
3	Booking service process the request	1. a checks for conflicts is performed 2. The new booking should be persistently saved in the DB	Not Tested	Involved components not implemented
4	Booking tool refresh	The resource is displayed with a status BOOKED	Not Tested	Involved components not implemented

Table 30: Verification test of the Booking on reserved date

Test ID: **BT03**	Conducted by:	Date:	Test Category: **Verification Tests (front end tier)**
Hardware Configuration	-		
Software Configuration	-		
Test Name:	***Booking on reserved date***		
Preconditions	• connection to the Booking Service ok • user opened Booking Tool though the Resource Explorer Tool (selected UxVs as parameter) • The selected resource should already be booked		
Related Requirements			
Tools Used			

Step	Action	Expected Result	Status	Remarks
1	User selects "New booking" from the UI	Booking Tool shows booking form	Not Tested	Involved components not implemented
2	User enters data (name, time, comments) and a date where already reservations exist and submits the form	A Booking Request is initiated to the Booking Service	Not Tested	Involved components not implemented
3	Booking service process the request	1. a checks for conflicts is performed 2. No data is saved in the DB 3. An appropriate response message is returned that there are already reservations	Not Tested	Involved components not implemented
4	Booking tool refresh	No information exists for the resource in the Booking Tool	Not Tested	Involved components not implemented

2.5.1.6 Data Analysis Tool, engine and results DB

Table 31: Verification test of the ability of the Analysis Engine to query message bus streams & schemas from the schema registry

Test ID**: PT-DAA-E-001**	Conducted by**: HESSO**	Date**: Feb 2016**	Test Category**: Verification Tests (front end tier)**
Hardware Configuration	• Spark Master [8 core / 16 gb ram] • Spark Slave [8 core / 16gb ram] • Spark Slave [8 core / 16gb ram] • 3 node Zookeer setup [collocated on spark] • 1x Name node | 1 x Data node		
Software Configuration	• Spark 1.6 • Graphite 0.9 • Confluent 2.01		
Test Name:	***Analysis Engine will be able to query message bus streams & schemas from the schema registry***		
Preconditions	• Working message bus • Working schema registry • Working Data Analysis Tool		
Related Requirements			
Tools Used			

Step	Action	Expected Result	Status	Remarks
1	User deploys job (currently via CLI, but in the future via web UI)	DAE checks if job is a pre-existing jar, else compiles a new one	Success	
2	DAE verifies schema from registry and starts a spark job that acquires data from the message bus	The job is successfully build and uploaded to the job server	Success	

Table 32: Verification test of the ability of the Analysis Engine to receive messages from the Analysis Tool

Test ID**: PT-DAA-E-002**	Conducted by**: HESSO**	Date**: Feb 2016**	Test Category**: Verification Tests (front end tier)**
Hardware Configuration	• Spark Master [8 core / 16 gb ram] • Spark Slave [8 core / 16gb ram] • Spark Slave [8 core / 16gb ram] • 3 node Zookeer setup [collocated on spark] • 1x Name node | 1 x Data node		
Software Configuration	• Spark 1.6 • Graphite 0.9 • Confluent 2.01		
Test Name:	***Analysis Engine will be able to receive messages from the Analysis Tool***		
Preconditions	• Working message bus • Working schema registry • Working Data Analysis Tool		
Related Requirements	PT-DAE-001		
Tools Used			

Step	Action	Expected Result	Status	Remarks
1	User builds a job on the Data Analysis Tool	Job is successfully checked for errors	Not tested	Data Analysis tool job selection process not implemented
2	Data Analysis Engine receives job via	The job is	Not tested	Data pipeline between the

	message bus and builds a job	successfully compiled (or an error returned)		UI and the DAE is currently in construction
3	Data Analysis Engine builds job and sends data to Spark	The job is converted to a JAR and uploaded via REST to the Spark job server	Success	

Table 33: Verification test of the ability of the Analysis Engine to write data to the results database

Test ID: **PT-DAA-E-003**	Conducted by: **HESSO**	Date: **Feb 2016**	Test Category: **Verification Tests (front end tier)**
Hardware Configuration	• Spark Master [8 core / 16 gb ram] • Spark Slave [8 core / 16gb ram] • Spark Slave [8 core / 16gb ram] • 3 node Zookeer setup [collocated on spark] • 1x Name node | 1 x Data node		
Software Configuration	• Spark 1.6 • Graphite 0.9 • Confluent 2.01		
Test Name:	***Analysis Engine will be able to write data to the results database***		
Preconditions	• Working message bus • Working schema registry • Working Data Analysis Engine • Working Graphite Instance		
Related Requirements			
Tools Used			

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	User builds a job and the jar is uploaded to the spark job server	Job is uploaded successfully and the job server registers the job in spark	Success	
2	Spark Engine sends results to the Graphite instance as it processes the data	Graphite displays a runtime stream of processed data	Success	

Table 34: Verification test of the provision of an interface to the Analysis Engine by the Analysis Tool

Test ID: **PT-DAA-T-001**	Conducted by: **HESSO**	Date: **Feb 2016**	Test Category: **Verification Tests (front end tier)**
Hardware Configuration	• Spark Master [8 core / 16 gb ram] • Spark Slave [8 core / 16gb ram] • Spark Slave [8 core / 16gb ram] • 3 node Zokeer setup [collocated on spark] • 1x Name node | 1 x Data node		
Software Configuration	• Spark 1.6 • Graphite 0.9 • Confluent 2.01		
Test Name:	***Analysis Tool will provide an interface to the Analysis Engine (DAE)***		
Preconditions	• Working message bus • Working schema registry • Working Data Analysis Tool		
Related Requirements	PT-DAA-T-002		
Tools Used			

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	User logs in to the web portal	Login successful	Success	
2	DAT queries available schemas from Schema Registry	All schemas are returned successfully	Success	
3	DAT allows user to select the data they want to work with as well as the machine learning algorithm and hyper-parameters	Job is sent via message bus to the DAE	Not tested	DAT UI is still under development

Table 35: Verification test of the ability of the Analysis Tool to query available data schemas

Test ID**: PT-DAA-T-002**	Conducted by**: HESSO**	Date**: Feb 2016**	Test Category**: Verification Tests (front end tier)**
Hardware Configuration	• Spark Master [8 core / 16 gb ram] • Spark Slave [8 core / 16gb ram] • Spark Slave [8 core / 16gb ram] • 3 node Zookeer setup [collocated on spark] • 1x Name node | 1 x Data node		
Software Configuration	• Spark 1.6 • Graphite 0.9 • Confluent 2.01		
Test Name:	***Analysis Tool will be able to query available data schemas***		
Preconditions	• Working message bus • Working schema registry • Working Data Analysis Tool		
Related Requirements			
Tools Used			

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	User logs in to the web portal	Login successful	Success	
2	DAT queries available schemas from Schema Registry	All schemas are returned successfully	Success	

Table 36: Verification test of the ability of the Analysis Tool to read results from the results database

Test ID**: PT-DAA-T-003**	Conducted by**: HESSO**	Date**: Feb 2016**	Test Category**: Verification Tests (front end tier)**
Hardware Configuration	• Spark Master [8 core / 16 gb ram] • Spark Slave [8 core / 16gb ram] • Spark Slave [8 core / 16gb ram] • 3 node Zzookeer setup [collocated on spark] • 1x Name node | 1 x Data node		
Software Configuration	• Spark 1.6 • Graphite 0.9 • Confluent 2.01		
Test Name:	***Analysis Tool will be able to read results from the results database***		
Preconditions	• Working message bus • Working schema registry • Working Data Analysis Tool • Working results database [graphite]		
Related Requirements	PT-DAA-T-001		
Tools Used			

Step	Action	Expected Result	Status	Remarks
1	User logs in to the web portal	Login successful	Success	
2	User builds job	Job successfully built (or error) and sent to DAE	Not tested	Message transfer pipeline from DAT to DAE is not yet implemented
3	Results are shown in results tab	Job results are shown as they are processed via graphite UI	Success	

2.5.1.7 Experiment authoring tool

Table 37: Verification test of the in-Textual Editor Experiments definition

Test ID**: EAT01**	Conducted by**: UoA**	Date**: Feb 2016**	Test Category**: Verification Tests (front end tier – middle tier)**
Hardware Configuration	-		
Software Configuration	• Xtext Server (pre-defined configuration) • ACE Editor (pre-defined 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	• Browser		

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	

Table 38: Verification test of the Textual Editor Experiments Update

Test ID**: EAT02**	Conducted by**: UoA**	Date**: Feb 2016**	Test Category**: Verification Tests (front end tier – middle tier)**
Hardware Configuration	-		
Software Configuration	• Xtext Server (pre-defined configuration) • ACE Editor (pre-defined 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	• Browser		

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	Not Tested	
3	Makes changes in the experiment workflow	The experiment is updated	Success	It was tested by inserting manually an experiment 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	

Table 39: Verification test of the in-Visual Editor Experiments Define

Test ID: **EAT03**	Conducted by: **UoA**	Date: **Feb 2016**	Test Category: **Verification Tests (front end tier – middle tier)**
Hardware Configuration	-		
Software Configuration	• Xtext Server (pre-defined configuration) • ACE Editor (pre-defined 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	• Browser		

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	Partial success	The visual editor is not completely implemented
3	Create an experiment by utilizing the available tools	The experimenter can defined waypoints and experiment information by clicking and designing in the visual editor	Not tested	
4	Define erroneous commands	The authoring tool responds with error messages and indication for correcting the error	Not tested	
5	Save the experiment	The experiment is stored in the database and specific files are produced to be adopted by the remaining RAWFIE components	Not tested	

Table 40: Verification test of the in-Visual Editor Experiments Update

Test ID: **EAT04**	Conducted by: **UoA**	Date: **Feb 2016**	Test Category: **Verification Tests (front end tier – middle tier)**
Hardware Configuration	-		
Software Configuration	• Xtext Server (pre-defined configuration) • ACE Editor (pre-defined 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	• Browser		

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	Not tested	
3	Makes changes in the experiment workflow	The experiment is updated	Not tested	
4	Save the experiment	The experiment is stored in the database and specific files are produced to be adopted by the remaining RAWFIE components	Not tested	

Table 41: Verification test of the Editor switching

Test ID: **EAT05**	Conducted by: **UoA**	Date: **Feb 2016**	Test Category: **Verification Tests (front end tier – middle tier)**
Hardware Configuration	-		
Software Configuration	• Xtext Server (pre-defined configuration) • ACE Editor (pre-defined 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	• Browser		

Step	Action	Expected Result	Status	Remarks
1	Access to the editors through the RAWFIE Web Portal	Redirection to the editors interface	Success	
2	Create an experiment	Experiment is presented in the editors	Partial success	The visual editor is not fully functioning
3	Switch to the alternative editor and make changes	The experiment is updated	Not tested	
4	Save the experiment	The experiment is stored in the database and specific files are produced to be	Not tested	

		adopted by the remaining RAWFIE components		

Table 42: Verification test of the experiment Launchings

Test ID**: EAT06**	Conducted by**: UoA**	Date**: Feb 2016**	Test Category**: Verification Tests (front end tier – middle tier)**
Hardware Configuration	-		
Software Configuration	• Xtext Server (pre-defined configuration) • ACE Editor (pre-defined 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	• Browser		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Access to the authoring tool through the RAWFIE Web Portal	Redirection to the editors 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	

2.5.1.8 EDL Compiler and Validator

Table 43: Verification test of the Experiments compilation

Test ID**: ECV01**	Conducted by**: UoA**	Date**: Feb 2016**	Test Category**: Verification Tests (front end tier – middle tier)**
Hardware Configuration	-		
Software Configuration	• Xtext Server (pre-defined configuration) • ACE Editor (pre-defined configuration)		
Test Name:	***Compile Experiments***		
Preconditions	• User entered in the RAWFIE Portal		
Related Requirements	PT-CPV-001, PT-CPV-002, PT-CPV-003, PT-CPV-004, PT-EXV-S-001, PT-EXV-S-002, PT-EXV-S-003		
Tools Used	• Browser		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Access to the authoring tool through the RAWFIE Web Portal	Redirection to the editors interface	Success	
2	Write a simple experiment	The experiment workflow is presented in the available editors	Partial Success	The visual editor is not fully functioning
3	Compile the experiment	The necessary files required by the remaining RAWFIE components are produced	Success	

Table 44: Verification test of the Experiments validation

Test ID**: ECV02**	Conducted by**: UoA**	Date**: Feb 2016**	Test Category**: Verification Tests (front end tier – middle tier)**
Hardware Configuration	-		
Software Configuration	• Xtext Server (pre-defined configuration) • ACE Editor (pre-defined configuration)		
Test Name:	***Validate Experiments***		
Preconditions	• User entered in the RAWFIE Portal		
Related Requirements	PT-CPV-001, PT-CPV-002, PT-CPV-003, PT-CPV-004, PT-EXV-S-001, PT-EXV-S-002, PT-EXV-S-003		
Tools Used	• Browser		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Access to the authoring tool through the RAWFIE Web Portal	Redirection to the editors interface	Success	
2	Write a simple experiment	The experiment workflow is presented in the available editors	Partial Success	The visual editor is not fully functioning
3	Validate the experiment	Validation is performed and error / warning messages are presented in the editors	Success	

2.5.1.9 *UxV Navigation Tool*

Table 45: Verification test of the UxV navigation tool access and produced instructions validation

Test ID**: UxVNT01**	Conducted by**: TBD**	Date**: Feb 2016**	Test Category**: Verification Tests (front end tier – middle tier)**
Hardware Configuration	-		
Software Configuration			
Test Name:	***Validate Experiments***		
Preconditions	• Requires Web Portal to be functioning and accessible		
Related Requirements			
Tools Used	• Browser		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Access the UxV Navigation Tool through the portal	Ability to navigate the swarm	Not tested	Access the UxV navigation tool and validate the produced instructions
2	Validate the produced instructions Validate the schema of the JSON output file Validate the data format of the JSON output file Validate the size of the JSON output file	All validation successful. The output data should be accessible and compatible with the required format	Not tested	This component will provide to the user the ability to remotely navigate a squad of UxVs. Through a user friendly interface, the experimenter will specify the required details of the experiment, providing information regarding the number of the vehicles, the number of the units etc.

2.5.1.10 *Visualization Tool*

Table 46: Verification test of the User request handling

Test ID**: VIS01**	Conducted by**: Epsilon**	Date**: Feb 2016**	Test Category**: Verification Tests (front end)**
Hardware Configuration	See section 2.3.1		
Software Configuration	See section 2.3.1		
Test Name:	***User request handling***		
Preconditions	• Requires visualization tool to be functioning & accessible. • Requires visualization engine to be functioning & accessible.		
Related Requirements			
Tools Used	• Browser		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	User sends a predefined websocket request via the visualization tool	The visualization tool forwards it to the visualization engine	Success	
2	Handle the response from the visualization engine	The response is visualized on the user screen	Success	

Table 47: Verification test of the Geospatial data handling

Test ID: **VIS02**	Conducted by: **Epsilon**	Date: **Feb 2016**	Test Category: **Verification Tests (front end)**
Hardware Configuration	See section 2.3.1		
Software Configuration	See section 2.3.1		
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			
Tools Used	• Browser		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Acquire predefined geospatial data (WMS, WFS) via the message bus	Data is properly received in the correct format at the VE	Success	
2	Modify the data to be suited for the VT and send it via websocket to VT	VT renders the data and plots it on the screen	Success	

Table 48: Verification test of the Geospatial data modification

Test ID: **VIS03**	Conducted by: **Epsilon**	Date: **Feb 2016**	Test Category: **Verification Tests (front end)**
Hardware Configuration	See section 2.3.1		
Software Configuration	See section 2.3.1		
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			
Tools Used	• Browser		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Acquire predefined geospatial data (WMS, WFS) via the message bus	Data is properly received in the correct format at the VE	Success	
2	Add a layer of information data and send it to the VT	VT plots the data and the layer properly	Not tested	This feature is not available yet

Table 49: Verification test of the Experiment Controller communication

Test ID: **VIS04**	Conducted by: **Epsilon**	Date: **Feb 2016**	Test Category: **Verification Tests (front end)**
Hardware Configuration	See section 2.3.1		
Software Configuration	See section 2.3.1		
Test Name:	***Experiment Controller communication***		
Preconditions	• Requires experiment controller to be functioning & accessible. • Requires visualization engine to be functioning & accessible.		
Related Requirements			
Tools Used			

Step	Action	Expected Result	Status	Remarks
1	Receive a message that the experiment has started from the Experiment Controller	The visualization tool starts the experiment	Not tested	The concept changed. Now the VT requests this information and cannot interact with such message from the Experiment Controller
2	Receive a message that the experiment has stopped from the Experiment Controller	The VT stops the experiment	Not tested	The concept changed. Now the VT requests this information and cannot interact with such message from the Experiment Controller

Table 50: Verification test of the Visualization Tool Interaction

Test ID: **VIS05**	Conducted by: **Epsilon**	Date: **Feb 2016**	Test Category: **Verification Tests (front end)**
Hardware Configuration	See section 2.3.1		
Software Configuration	See section 2.3.1		
Test Name:	***Visualization Tool Interaction***		
Preconditions	• Requires visualization tool to be functioning & accessible. • Requires visualization engine to be functioning & accessible.		
Related Requirements			
Tools Used	• Browser		

Step	Action	Expected Result	Status	Remarks
1	Enable/Disable different features of the visualization tool (e.g. show/hide speed web widget)	The user sees the updated plot (show/hide speed web widget)	Success	

Table 51: Verification test of the Camera interaction

Test ID: **VIS06**	Conducted by: **Epsilon**	Date: **Feb 2016**	Test Category: **Verification Tests (front end)**
Hardware Configuration	See section 2.3.1		
Software Configuration	See section 2.3.1		
Test Name:	***Camera interaction***		
Preconditions	• Requires visualization tool to be functioning & accessible. • Requires visualization engine to be functioning & accessible. • Requires Experiment controller to be functioning & accessible.		
Related Requirements			
Tools Used	• Browser		

Step	Action	Expected Result	Status	Remarks
1	Retrieve with the visualization engine quasi real time data from one UxV, processes it and send it to the visualization tool	The VT plots the data properly	Success	
2	Change the camera view for the scenario	Data camera is adjusted	Success	

2.5.2 Communication and storage components

2.5.2.1 Testbeds directory service

Table 52: Verification test of the resource Retrieval from testbed facility

Test ID**: TD01**	Conducted by**: IES**	Date**: Feb 2016**	Test Category**: Verification Tests (Middle Tier)**
Hardware Configuration			
Software Configuration	Testbed Directory Service deployed in a RAWFIE server (with the Apache Tomcat Servlet Container) Access to the PostgreSQL server granted		
Test Name:	***Retrieve resources from 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 the ID of the testbed he is looking for (in case of the getResources() API), and about the resource he is looking for (in case of the searchResourceAPI())		
Related Requirements			
Tools Used	SOAP UI Client		

Step	Action	Expected Result	Status	Remarks
1.a	The input JSON request is prepared, specifying the testbed identifier	The Testbed Directory Service gives back a JSON response message, containing details about all resources belonging to the specified testbed	Success	If no testbed or resources are found for any particular reason, or an error occurs, the calling component should be notified and should react accordingly. Specific error and notification messages are going to be compiled in the next iteration so that the calling component (e.g. the Resource Explorer Tool) will, in turn, provide them to the end users in a graphical and user friendly way
2.a	The getResources() REST API is called from the SOAP UI Client Tool, providing the prepared JSON request in input			
1.b	The input JSON request is prepared, specifying the testbed identifier and the identifier of the resource	The Testbed Directory Service gives back a JSON response message, containing detailed information about the specific resource belonging to the specified testbed	Success	If no testbed or resource is found for any particular reason, or an error occur, the calling component should be notified and should react accordingly. Specific error and notification messages are going to be compiled in the next iteration so that the calling component (e.g. the Resource Explorer Tool) will, in turn, provide them to the end users in a graphical and user friendly way
2.b	The searchResource() REST API is called from the SOAP UI Client Tool, providing the prepared JSON request in input			

Table 53: Verification test of the Addition of a new testbed facility to the RAWFIE federation

Test ID**: TD02**	Conducted by**: IES**	Date**: Feb 2016**	Test Category**: Verification Tests (Middle Tier)**
Hardware Configuration			
Software Configuration	Testbed Directory Service deployed in a RAWFIE server (with the Apache Tomcat Servlet Container) Access to the PostgreSQL server granted		
Test Name:	***Add new testbed facility to the RAWFIE federation***		
Preconditions	Access to the PostgreSQL server must be granted for the Testbed Directory Service When preparing the test, the test executor should know as much information as possible about the testbed to be inserted, and according to the information required by the platform		
Related Requirements			
Tools Used	SOAP UI Client		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	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 getTestbeds() or searchTestbed() REST API (see **TD04** in the following)	Success	If it is not possible to insert the new testbed for any particular reason (e.g. mal formatted JSON request), the calling component should be notified about the error occurred, and should react accordingly. Specific error and notification messages are going to be compiled in the next iteration so that the calling component (e.g. the Resource Explorer Tool) will, in turn, provide them to the end users in a graphical and user friendly way
2	The createTestbed() REST API is called from the SOAP UI Client Tool, specifying the testbed information in the input JSON request			

Table 54: Verification test of the Registration of a new UxV node into a testbed facility

Test ID**: TD03**	Conducted by**: IES**	Date**: Feb 2016**	Test Category**: Verification Tests (Middle Tier)**
Hardware Configuration			
Software Configuration	Testbed Directory Service deployed in a RAWFIE server (with the Apache Tomcat Servlet Container) Access to the PostgreSQL server granted		
Test Name:	***Register new 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 as much information as possible about the new resource to be added, the related testbed, and according to the information required by the platform		
Related Requirements			
Tools Used	SOAP UI Client		

Step	Action	Expected Result	Status	Remarks
1	The input JSON request is prepared, with the information about the new resource to be added (and 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 getResources() or searchResource() REST API (see **TD01** above)	Success	If it is not possible to insert the new resource (UxV node) for any particular reason (e.g. malformatted JSON request), the calling component should be notified about the error occurred, and should react accordingly. Specific error and notification messages are going to be compiled in the next iteration so that the calling component (e.g. the Resource Explorer Tool) will, in turn, provide them to the end users in a graphical and user friendly way
2	The createResource() REST API is called from the SOAP UI Client Tool, specifying the needed information in the provided input JSON request			

Table 55: Verification test of the Retrieval of testbed information and belonging resources

Test ID**: TD04**	Conducted by**: IES**	Date**: Feb 2016**	Test Category**: Verification Tests (Middle Tier)**
Hardware Configuration			
Software Configuration	Testbed Directory Service deployed in a RAWFIE server (with the Apache Tomcat Servlet Container) Access to the PostgreSQL server granted		
Test Name:	***Retrieve testbed information and belonging resources***		
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 is looking for, in case only information of resources form a specific testbed is required		
Related Requirements			
Tools Used	SOAP UI Client		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1.a	The getTestbeds() REST API is called from the SOAP UI Client Tool, without any specific testbed information (null JSON input request)	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	If no testbeds are found for any particular reason, or an error occurs, the calling component should be notified and should react accordingly. Specific error and notification messages are going to be compiled in the next iteration so that the calling component (e.g. the Resource Explorer Tool) will, in turn, provide them to the end users in a graphical and user friendly way
1.b	The input JSON request is prepared, with the information about the identifier of the testbed we are requesting information	The Testbed Directory Service gives back a JSON response message, containing details about the testbed and all registered resources belonging to it	Success	If no testbed is found for any particular reason, or an error occurs, the calling component should be notified and should react accordingly. Specific error and notification messages are going to be compiled in the next iteration so that the calling component (e.g. the Resource Explorer Tool) will, in turn, provide them to the end users in a graphical and user friendly way
2.b	The searchTestbed() REST API is called from the SOAP UI Client Tool, specifying the needed information in the provided input JSON request			

2.5.2.2 Users and Rights Service

Table 56: Verification test of the Visualisation of experiment status

Test ID**: URS01**	Conducted by**: Fraunhofer**	Date**: Feb 2016**	Test Category**: Verification Tests (middle tier)**
Hardware Configuration	See section 2.3.1		
Software Configuration	See section 2.3.1		
Test Name:	***Visualisation of experiment status***		
Preconditions	• Valid user name and password known		
Related Requirements	•		
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 return failure	Success	
2	valid user name and password sent to the Users & Rights Service	Users & Rights Service return failure	Success	

Table 57: Verification test of the user rights checks

Test ID**: URS02**	Conducted by**: Fraunhofer**	Date**: Feb 2016**	Test Category**: Verification Tests (middle tier)**
Hardware Configuration	See section 2.3.1		
Software Configuration	See section 2.3.1		
Test Name:	***Check user rights***		
Preconditions	• Valid user rights known		
Related Requirements	•		
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	

2.5.2.3 Launching Service

Table 58: Verification test of the short term launching

Test ID**: LS01**	Conducted by**: HAI**	Date**: Feb 2016**	Test Category**: Verification Tests (middle tier)**
Hardware Configuration	See section 2.3.1		
Software Configuration	See section 2.3.1		
Test Name:	***Short term launching***		
Preconditions	• Requires the Web portal to be accessible. • Requires the Launching tool to be accessible. • Requires the Message Bus and the experiment controller to be accessible. • The master data repository should contain reservations for the user.		
Related Requirements			
Tools Used			

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	User selects an already defined experiment	Experiment info is loaded to UI	Success	
2	User initiates manual start via the web UI for the select experiment	manualStart is called on the Launching Service, checking if no executionId already exists for the experiment	Success	
2-1		If no execution ID exists: 1. Launching service generates an ExperimentStartRequest to the Message Bus. 2. An executionId is generated that uniquely identifies the running experiment 3. ExperimentStartRequest is consumed by the ExperimentController	Success	ExperimentController is not implemented therefore the message is consumed directly by the ResourceController in the trestbed tier
2-2		If an execution ID already exists: 1. Launching service considers the experiment already running and returns an error message 2. No further action	Success	

Table 59: Verification test of long term launching

Test ID: **LS02**	Conducted by: **HAI**	Date: -	Test Category: **Verification Tests (middle tier)**
Hardware Configuration	See section 2.3.1		
Software Configuration	See section 2.3.1		
Test Name:	***Long term launching***		
Preconditions	• Requires the Web portal to be accessible. • Requires the Launching tool to be accessible. • Requires the Message Bus and the experiment controller to be accessible. • The master data repository should contain reservations for the user. • The master data repository should contain experiments scheduled for a feature time • The platform launching scheduler must be running		
Related Requirements			
Tools Used			

Step	Action	Expected Result	Status	Remarks
1	Initiate the checking process of the booking repository (via Platform Scheduler trigger)	An experiment is identified in the DB	Not Tested	launching scheduler does not yet exists
2		sheduledStart is called by the Launching Service	Not Tested	Method not yet implemented
2-1		If no execution ID exists: 4. Launching service generates an ExperimentStartRequest to the Message Bus. 5. An executionId is generated that uniquely identifies the new experiment 6. ExperimentStartRequest is consumed by the ExperimentController	Not Tested	
2-2		If an execution ID already exists: 3. Launching service considers the experiment already running and returns an error message 4. No further action	Not Tested	

2.5.3 Testbed control, monitoring and analysis components

2.5.3.1 Experiment Controller

Table 60: Verification test of Experiment Controller connection

Test ID**: EC01**	Conducted by**: CERTH**	Date**: -**	Test Category**: Verification Tests (middle tier)**
Hardware Configuration	See section 2.3.1		
Software Configuration	See section 2.3.1		
Test Name:	***Connection Test***		
Preconditions	• Requires web portal to be functioning and accessible. • Register an experiment (Testbed manager) • Send Network Requirements (Testbed manager) • Send basic instructions to the Resource Controller • Transmit simulated or real results back to the Experiment Monitoring Tool		
Related Requirements			
Tools Used			

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Register an experiment (Testbed manager)	Successful registration	Not Tested	
2	Send Network Requirements (Testbed manager)	Network requirements met, acknowledged by the Testbed Controller	Not Tested	
3	Send basic instructions to the Resource Controller	Instructions acknowledged by the Resource Manager (resources are available)	Not Tested	The EC transmits to the resource controller the instructions he received from the Web Portal.
4	Transmit simulated or real results back to the Experiment Monitoring Tool	Results successfully received by the Experiment Monitoring Tool	Not Tested	

Table 61: Verification test of Experiment Controller workflow

Test ID: **EC02**	Conducted by: **CERTH**	Date: -	Test Category: **Verification Tests (middle tier)**
Hardware Configuration	See section 2.3.1		
Software Configuration	See section 2.3.1		
Test Name:	***Execute experiment workflow***		
Preconditions	• The experimenter have already created the script for the experiment of interest • The chosen resource must be completely available and ready to use		
Related Requirements			
Tools Used			

Step	Action	Expected Result	Status	Remarks
1	The experimenter forwards the script to the Experiment Controller in order to start or barely execute the next action of the resource mission	Successful forwarding and start of execution	Not Tested	
2	The instructions are forwarded to the corresponding testbed facility	Testbed facility received the instructions correctly	Not Tested	
3	The resource receives the new set of instructions as generated from the script for overriding the experiment workflow	The resource overrides its current experiment according to the new instructions	Not Tested	The execution of the experiment happens just as the experimenter defined it in the EDL script and the action was successfully performed.
4			Not Tested	

2.5.3.2 Monitoring Manager

Table 62: Verification test of Monitoring Activity

Test ID: **MM01**	Conducted by: **CSEM**	Date: -	Test Category: **Verification Tests (middle tier)**
Hardware Configuration	See section 2.3.1		
Software Configuration	See section 2.3.1		
Test Name:	***Check Monitoring Activity***		
Preconditions	• Requires the resource controller to be accessible. • Requires the network controller to be accessible. • Requires the data tier to be accessible.		
Related Requirements			
Tools Used			

Step	Action	Expected Result	Status	Remarks
1			Not Tested	The experiment should smoothly start and the appropriate RAWFIE components should be initiated.

2.5.3.3 Network Controller

Table 63: Verification test of network interface switching due to connectivity problems

Test ID**: NC01**	Conducted by**: CSEM**	Date**: -**	Test Category**: Verification Tests (middle tier)**
Hardware Configuration	See section 2.3.1		
Software Configuration	See section 2.3.1		
Test Name:	***Switch network interface due to connectivity problem***		
Preconditions	• Requires the Testbed Manager to be accessible		
Related Requirements			
Tools Used			

Step	Action	Expected Result	Status	Remarks
1	The Network Controller 'checks' the connectivity of the resources through the Resource Controller.	The Resource Controller informs the Network Controller for malfunctions in the network connectivity of the resources.	Not Tested	
2	The Network Controller receives the incoming messages from the Resource Controller.	The appropriate network interface is selected.	Not Tested	The Network Controller identifies problems in the connectivity and triggers the Resource Controller to force the change of the network interface.

2.5.3.4 Resource Controller and Navigation Service

Table 64: Verification test of Connection and of Accuracy validation of the given Instructions

Test ID**: RC01**	Conducted by**: CERTH**	Date**: -**	Test Category**: Verification Tests (middle tier)**
Hardware Configuration	See section 2.3.1		
Software Configuration	See section 2.3.1		
Test Name:	***Connection Test and Validation of the Accuracy of the Given Instructions***		
Preconditions	• The proxy should be connected to the testbed • Requires the UxV to be ready to operating (e.g. en route). • Requires the UxV to be reachable by any communication mean.		
Related Requirements			
Tools Used			

Step	Action	Expected Result	Status	Remarks
1	Receive instructions from the Experiment Controller	Instructions received	Not Tested	
2	Validate the Obstacle Avoidance Mechanism using known simulated scenarios	Validation Status available	Not Tested	
3	Validation of the Collision Avoidance Mechanism using known simulated scenarios	Validation Status available		
4	Send basic instructions to the UxVs through the Testbed Manager so as to perform UxV01- UxV05 tests	The UxV follows the instruction correctly, in order and timely, according to the specified parameters.		
5	Transmit the results back to the Experiment Controller			

2.5.4 Testbed resources

2.5.4.1 Testbed Manager

Note: TM01, TM02, TM03 are obsolete.

Table 65: Verification test of Testbed health status

Test ID: **TM04**	Conducted by: **HAI**	Date: **Feb 2016**	Test Category: **Verification Tests (Testbed tier)**
Hardware Configuration Details	Testbed site x86 PC		
Software Configuration Details	Java installed, RAWFIE Testbed Manager installed (Java application), Apache Kafka in RAWFIE platform accessible		
Test Name:	***Check Testbed health status***		
Preconditions	• Requires middle tier to be accessible (System Monitoring Service) • Initial Testbed Manager configuration: o CPU usage WARNING > 50%, CRITICAL >90% o Memory usage WARNING > 50%, CRITICAL >90% o Disk usage WARNING > 50%, CRITICAL >90% o Frequency of sending messages 30 sec		
Related Requirements			
Tools Used			

Step	Action	Expected Result	Status	Remarks
1	Testbed Manager started	1. Testbed manager successfully initialized 2. Testbed Manager checks periodically CPU load, memory and disk usage	Success	
2	Testbed 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 Manager status. Initial status OK	Success	
4	Artificially increase CPU or Memory usage	Status message sent to the message bus (TBC)	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 Manager 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 Manager status. Status changes back to OK	Success	Close extra running applications

Note: The following tests are obsolete, although performed, due to implementation changes. They are mentioned for reference and will be either updated or removed in next iteration.

Table 66: Verification test of status of the experiments

Test ID: **TM02**	Conducted by:	Date:	Test Category: **Verification Tests (Testbed tier)**
Hardware Configuration Details	Testbed site x86 PC		
Software Configuration Details	Java installed, RAWFIE Testbed Manager installed (Java application), Apache Kafka in RAWFIE platform accessible		
Test Name:	***Checks the status of the experiments***		
Preconditions	• Requires middle tier to be accessible • Requires the experiment controller to be accessible • ~~Requires Data Tier to be accessible~~ (Obsolete: Data Tier is accessible only from Middle Tier components)		
Related Requirements			
Tools Used			

Step	Action	Expected Result	Status	Remarks
1	Experiment Controller sends a request to Testbed Manager	Request message properly received from Testbed Manager	Not Applicable	This step has been omitted as it has been replaced by sending the experiment status periodically from Testbed Manager without a previous request
2	Testbed Manager checks locally the status of the experiments	-		
3	Sends a list with the experiments and their status to Experiment Controller	The list of experiments is properly received from Experiment Controller	Not Applicable	(replaced by step 4)
4	Sends the experiment status to Experiment Controller			

Table 67: Verification test of the Management of the experiments without middle-tier connection

Test ID: **TM03**	Conducted by:	Date:	Test Category: **Verification Tests (Testbed tier)**
Hardware Configuration Details	Testbed site x86 PC		
Software Configuration Details	Java installed, RAWFIE Testbed Manager installed (Java application), Apache Kafka in RAWFIE platform accessible		
Test Name:	***Manage the experiments without middle-tier connection***		
Preconditions	• Testbed loses the connection with the middle tier		
Related Requirements			
Tools Used			

Step	Action	Expected Result	Status	Remarks
1	Testbed Manager checks the status of the experiments	Testbed Manager properly receives ExperimentStatus message	Not tesred	
2	Testbed Manager informs Resource Controller for “emergency” situation and pause experiments		Not Applicable	
3	Resource Controller sends a response		Not Applicable	

2.5.4.2 UxV Node

The UxV node and related components are interacting with the other Rawfie component through the Message Bus.

Table 68: Verification test of UxV Return to base

Test ID: **UxV01**	Conducted by: **MST**	Date: **Feb 2016**	Test Category: **Verification Tests (Testbed tier)**
Hardware Configuration	rawfie.mst.auv-1, rawfie.mst.auv-2, rawfie.mst.asv-1		
Software Configuration	OceanScan Proxy 2016.02		
Test Name:	***Return to base***		
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	Resource controller reachable		
Tools Used	Neptus Command & Control Software		

Step	Action	Expected Result	Status	Remarks
1	Establish the communication with the UxV	Communication established	Success	
2	Establish a secure control session	Secured control session established	Partial Success	At this point only network level security is used (i.e., WPA2)
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	
5	Close the secure control session.	The UxV is home after a safe return. Connection closed	Partial Success	See remark on step 2

Table 69: Verification test of the ability of the UxV to follow a route

Test ID: **UxV02**	Conducted by: **MST**		Date: **Feb 2016**	Test Category: **Verification Tests (testbed tier)**
Hardware Configuration	rawfie.mst.auv-1, rawfie.mst.auv-2, rawfie.mst.asv-1			
Software Configuration	OceanScan Proxy 2016.02			
Test Name:	***Follow a route***			
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	Resource controller reachable			
Tools Used	Neptus Command & Control Software			
Step	**Action**	**Expected Result**	**Status**	**Remarks**
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 70: Verification test of Acquire sensor samples

Test ID: **UxV03**	Conducted by: **MST**	Date: **Feb 2016**	Test Category: **Verification Tests (Testbed tier)**
Hardware Configuration	rawfie.mst.auv-1, rawfie.mst.auv-2, rawfie.mst.asv-1		
Software Configuration	OceanScan Proxy 2016.02		
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			
Tools Used	Neptus Command & Control Software		

Step	Action	Expected Result	Status	Remarks
1	Establish the communication with the UxV	Communication established	Success	
2	Establish a secure control session (if not done already)	Secured control session established	Partial Success	At this point only network level security is used (i.e., WPA2)
3	Send the acquisition commands	Commands received and executed	Partial Success	At this point the UxVs are always acquiring data from all sensors
4	Store sensor samples and, if possible, transmit them via the data communication system	Samples stored and, if possible, transmitted	Success	
5	If opened specifically for the matter of the test, close the secure control session.	Sensor samples have acquired correctly and are stored in the UxV memory or in the experiment database. Connection closed	Partial Success	See remark on step 2

Table 71: Verification test of Fidelity to commands

Test ID: **UxV04**	Conducted by: **MST**	Date: **Feb 2016**	Test Category: **Verification Tests (Testbed tier)**
Hardware Configuration	rawfie.mst.auv-1, rawfie.mst.auv-2, rawfie.mst.asv-1		
Software Configuration	OceanScan Proxy 2016.02		
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			
Tools Used	Neptus Command & Control Software		

Step	Action	Expected Result	Status	Remarks
1	Establish the communication with the UxV	Communication established	Success	
2	Establish a secure control session (if not done already)	Secured control session established	Partial Success	At this point only network level security is used (i.e., WPA2)
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 behaviour of the UxV,	Status recorded	Success	
6	If opened specifically for the matter of the test, close the secure control session.	Sensor samples have acquired correctly and are stored in the UxV memory or in the experiment database. Connection closed	Partial Success	See remark on step 2

2.5.4.3 UxV Network Communication

Table 72: Verification test of Continuous communication

Test ID: **UxV06**	Conducted by: **MST**	Date: **Feb 2016**	Test Category: **Verification Tests (Testbed tier)**
Hardware Configuration	rawfie.mst.auv-1, rawfie.mst.auv-2, rawfie.mst.asv-1		
Software Configuration	OceanScan Proxy 2016.02		
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			
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	The UxV is “home” (to be defined, since it may depend on the type of UxV, the running experiment, the host testbed) after a safe return. “Home” may be an attribute of the UxV.
3	Close the communication session.	Communication closed	Success	

Table 73: Verification test of Secure communication

Test ID: **UxV07**	Conducted by: **MST**	Date: **Feb 2016**	Test Category: **Verification Tests (Testbed tier)**
Hardware Configuration	rawfie.mst.auv-1, rawfie.mst.auv-2, rawfie.mst.asv-1		
Software Configuration	OceanScan Proxy 2016.02		
Test Name:	***Secure 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			
Tools Used	Neptus Command & Control Software		

Step	Action	Expected Result	Status	Remarks
1	Establish the communication with the UxV	Communication established	Success	
2	Establish a secure control session (if not done already)	Secured control session established	Partial Success	At this point only network level security is used (i.e., WPA2)
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	The end to end communication between the UxV and the ground control is established, secured and maintained.
5	Close the secure control session.	Connection closed	Partial Success	See remark on step 2

Table 74: Verification test of Real-time communication

Test ID: **UxV08**	Conducted by: **MST**	Date: **Feb 2016**	Test Category: **Verification Tests (Testbed tier)**
Hardware Configuration	rawfie.mst.auv-1, rawfie.mst.auv-2, rawfie.mst.asv-1		
Software Configuration	OceanScan Proxy 2016.02		
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 (e.g. en route). - Requires the UxV to be reachable by any communication mean.		
Related Requirements			
Tools Used	Neptus Command & Control Software		

Step	Action	Expected Result	Status	Remarks
1	Establish the communication with the UxV	Communication established	Success	
2	Establish a secure control session (if not done already)	Secured control session established	Partial Success	At this point only network level security is used (i.e., WPA2)
3	Send safe commands and measure the temporal characteristics of the communication (e.g. response time, synchronisation 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
4	Close the secure control session.	Connection closed	Partial Success	See remark on step 2

Table 75: Verification test of Resume communication and data transfer

Test ID**: UxV09**	Conducted by**:**	Date**:**	Test Category**:** **Verification Tests (Testbed tier)**
Hardware Configuration	See section 2.3.1		
Software Configuration	See section 2.3.1		
Test Name:	***Resume communication and data transfer***		
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			
Tools Used			

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Establish the communication with the UxV	Communication established	Not Tested	
2	Start a transaction.	Transaction started	Not Tested	
3	Interrupt the communication at the low-level (e.g. disconnect the antenna)	Communication is interrupted, the transaction is not complete.	Not Tested	The UxV detects the communication interruption and the re-establishment of the communication link and resume the interrupted transaction (may be by restarting it).
4	Re-establish the communication low level means	The transaction resumes and completes	Not Tested	
5	Close the communication session.	Connection closed	Not Tested	

Table 76: Verification test of UxV Device Management

Test ID: **UxV10**	Conducted by:	Date:	Test Category: **Verification Tests (Testbed tier)**
Hardware Configuration	See section 2.3.1		
Software Configuration	See section 2.3.1		
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			
Tools Used			

Step	Action	Expected Result	Status	Remarks
1	Establish the communication with the UxV	Communication established	Not Tested	
2	Establish a secure control session (if not done already)	Secured control session established	Not Tested	
3	Send the return to base command	Command received and applied	Not Tested	
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	Not Tested	
5	Close the secure control session.	The UxV is home after a safe return. Connection closed	Not Tested	

Table 77: Verification test of the UxV connection

Test ID: UxV11	Conducted by: MST	Date: **Feb 2016**	Test Category**: Verification Tests (testbed tier)**
Hardware Configuration	rawfie.mst.auv-1, rawfie.mst.auv-2, rawfie.mst.asv-1		
Software Configuration	OceanScan Proxy 2016.02		
Test Name:	***Connection Test***		
Preconditions	UxV-Node launched		
Related Requirement	Message bus working		
Tools Used	OceanScan Proxy 2016.02 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 78: Verification test of Sensor Data Acquisition 1

Test ID: **UxV12**	Conducted by: **MST**	Date: **Feb 2016**	Test Category: **Verification Tests (Testbed tier)**
Hardware Configuration	rawfie.mst.auv-1, rawfie.mst.auv-2, rawfie.mst.asv-1		
Software Configuration	OceanScan Proxy 2016.02		
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			
Tools Used	Neptus Command & Control Software		

Step	Action	Expected Result	Status	Remarks
1	Establish the communication with the UxV	Communication established	Success	
2	Establish a secure control session (if not done already)	Secured control session established	Partial Success	At this point only network level security is used (i.e., WPA2)
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
5	Close the secure control session.	The UxV is home after a safe return. Connection closed	Partial Success	See remark on step 2

Table 79: Verification test of Sensor Data Acquisition 2

Test ID: **UxV13**	Conducted by: **MST**	Date: **Feb 2016**	Test Category: **Verification Tests (Testbed tier)**
Hardware Configuration	rawfie.mst.auv-1, rawfie.mst.auv-2, rawfie.mst.asv-1		
Software Configuration	OceanScan Proxy 2016.02		
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			
Tools Used	Neptus Command & Control Software		

Step	Action	Expected Result	Status	Remarks
1	Establish the communication with the UxV	Communication established	Success	
2	Establish a secure control session (if not done already)	Secured control session established	Partial Success	At this point only network level security is used (i.e., WPA2)
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.
6	Close the secure control session.	The UxV is home after a safe return. Connection closed	Partial Success	See remark on step 2

Table 80: Verification test of Data Storage

Test ID: **UxV14**	Conducted by: **MST**	Date: **Feb 2016**	Test Category: **Verification Tests (Testbed tier)**
Hardware Configuration	rawfie.mst.auv-1, rawfie.mst.auv-2, rawfie.mst.asv-1		
Software Configuration	OceanScan Proxy 2016.02		
Test Name:	***Data Storage***		
Preconditions	- UxV is in operation state and the parent UxV node has been launched. - Sensor node is functional		
Related Requirements			
Tools Used	Neptus Command & Control Software		

Step	Action	Expected Result	Status	Remarks
1	Establish the communication with the UxV	Communication established	Success	
2	Establish a secure control session (if not done already)	Secured control session established	Partial Success	At this point only network level security is used (i.e., WPA2)
3	A request for storing certain data is done	Command received and data is stored locally	Partial Success	At this point no such command exists and the UxVs will store all data
4	After a mission given, data storage in the system is checked.	Data was correctly stored and kept.	Success	The data is stored and identified in the robot system
5	Close the secure control session.	The UxV is home after a safe return. Connection closed	Partial Success	See remark on step 2

Table 81: Verification test of Waypoints Processed

Test ID: **UxV15**	Conducted by: **MST**	Date: **Feb 2016**	Test Category: **Verification Tests (Testbed tier)**
Hardware Configuration	rawfie.mst.auv-1, rawfie.mst.auv-2, rawfie.mst.asv-1		
Software Configuration	OceanScan Proxy 2016.02		
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			
Tools Used	Neptus Command & Control		

Step	Action	Expected Result	Status	Remarks
1	Establish the communication with the UxV	Communication established	Success	
2	Establish a secure control session (if not done already)	Secured control session established	Not Tested	At this point only network level security is used (i.e., WPA2)
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.	Partial Success	The UxVs used in this test are not equipped with collision avoidance sensors.
5	Iterate step 4 until assessment is complete	UxV stops, informs and recalculate its route to next waypoint if an unexpected obstacle is found.		
6	Close the secure control session.	The UxV is home after a safe return. Connection closed	Partial Success	See remark on step 2

3 Roadmap

The results obtained during the experimentations and the specific tests are analysed to identify and characterise the improvements and fixes to be brought to the prototype implementation (second iteration). Furthermore, possible customizations are also briefly mentioned.

3.1 Deviations

No major deviation from the initial plan has been required, implemented or identified from the integration standpoint. However, the Testbed Proxy component has been removed from the RAWFIE architecture, which slightly impacted a number of tests.

Some components and tests have not yet been performed, which deviates from the D4.3 test planning. Additionally, changes have been brought in the verification template used in D4.3

to report the observations and results in the same place; a template for the specific test of the interface has been created and used for the corresponding test report)

3.2 Suggested modifications and improvements

3.2.1 Modifications and improvements to the RAWFIE system

Eventually, a limited number of major modifications have been collected at this point, although the design and the initial implementation steps led to numerous adjustments, design fixes, etc. This was particularly the case for the data model, the usage of the message bus, the definition of the AVRO schemas and the geographical coordinate system(s). After this stabilisation phase, the remaining modifications are listed below:

- Web Portal
  - Better integration of the EDL editor and the Visualisation Tool (currently done via iframe)
- System Monitoring Tool
  - Better structured view: Categories and filters functions instead of a plain table

Since not all components have been implemented or tested, further modifications are to be expected in the next development iterations.

The improvements of verified components that have been identified and to be implemented during the next cycles are:

- Web Portal
  - Implement User management GUI
  - Language is changed, but only a few texts are translated (add translations)
- Launching Service
  - Improve the feedback returned to the callers of the Launching Service API by adding an appropriate text field in the returned response (currently in case of error there is no indication on what exactly went wrong)
- Resource Explorer Tool
  - should implement a search functionality
  - More filtering criteria for the selection of resources/UxVs may be useful in a subsequent iteration
  - More filtering criteria for the selection of testbeds may be useful in a subsequent iteration
- EDL Visual Editor
  - Complete and fix the visual editor features
  -
- System Monitoring Service
  - Also monitor UxV status
- System Monitoring Tool
  - Servers and Testbeds displayed, but UxVs did not send status information (to be implemented)
- Master Data Repository

    - Creation of additional history tables for certain tables of the RAWFIE data model in order to have better auditing of all actions related mainly to experiment execution and resource reservation (i.e. currently only the last status of an executing experiment is available).
- Visualisation engine
    - Get the location and sensor data from the UxVs. Implement the support for all sensor data.
- Testbed manager
    - Experiment Controller does not yet exists - message sent from Launching Service.
- UxV node
    - Visualization indoors needs revision to offer a descriptive environment,
    - Only temperature measurement was tested. Add more sensor interfaces.
    - Threshold to accept local position as the waypoint needs to be carefully tuned (in particular when following a route).
    - To modify the architecture of the Publishers regarding ROS-Rawfie adaptor to make each publisher match an specified ros standard message, in case future partners can make use of them

4 Suggested Customizations

This paragraph aims at listing the expected customization mechanisms foreseen for supporting the following objectives:

- Adapt to a specific application or usage;
- Adapt to specific regulations;
- Adapt to specific environment;
- Etc.

Customization is not “improvement or refinements”, but the adaptation or personalization of the system as it is to a specific purpose, usage or environment. The customization is done by RAWFIE stakeholders and not by the project consortium. However, the project consortium defines and implement the customisation mechanisms. Customisation does not directly address issues, problems, failures, functional or non-functional gaps, etc. but customisation may allow for selecting different options helping in solving them.

To achieve such objectives, it is possible to do:

- Customisation through parametrization
- Customisation through component customization (affects only internal interfaces of components or component implementation)
- Customisation through component recombination (different components are used instead of the initial ones, leading to potential interface redefinitions)

Note that customisation is a static process, which does not change once performed. Further customisation is required to change, which is not supposed to be performed in real-time.

4.1 Component customizations

Many customisations are expected to occur during the project, in particular when adapting the RAWFIE system to the needs and aims of applications developed in the projects selected in the context of the Open calls.

The customisation of components by any authorised stakeholder could be made possible by defining a generic interface exposed by any RAWFIE component for the support of plug-ins (registration, authentication, activation, etc.) that would have access to private component-specific interfaces. Such interfaces would be publicly described (structure, parameters, semantics) but would be only accessible by duly registered plug-ins. This proposal is currently under study and it has not been implemented yet.

4.2 General Platform & testbed Customizations

The above mechanisms allow for the customisation of most aspect of the RAWFIE platform. Other needs requiring the exploration of further customisation mechanisms have not been yet identified.

4.3 UxVs Customizations

UxVs are probably the most varying element of the applications targeted by RAWFIE. They can be of three different natures at least (ground, aerial, water surface and more), for which the characteristics can be very different from one model to another. Since the general architecture of UxV varies from one manufacturer or UxV family, it is only possible to take into account its external behaviour and physical characteristics, in particular in the form of requirements. Most of these requirements have been identified and described in WP3 deliverables and D4.4. The verification of the UxV component has been done on the basis of these requirements, which define the typical behaviour and characteristics of a RAWFIE UxV.

As a matter of fact, the customisation of the RAWFIE UxVs has been done by the two UxV manufacturers that are members of the RAWFIE consortium (for ground and water surface vehicles), exclusively for allowing the support for the integration with the RAWFIE ecosystem. As this point, no customisation is provided by RAWFIE to easily customise the UxV component to application specific needs. For the time being, the objective is to list the detected improvements, fixes and new features to be brought to the already integrated system.

Further customisation can be done in two ways: by using UxV-dedicated values for some of the component parameters or by “plugging” additional components to the RAWFIE components that are linked to the UxV (i.e. components in direct connection with the UxV through the “UxV adaptor” and components that consume or produce UxV understandable data through the message bus, e.g. in other RAWFIE tiers).

The proposed customisation approach will be experimented by UxV manufacturers, for example during the projects selected in the context of the Open Calls.

5 Conclusion

Generally, the integration and the resulting RAWFIE prototype followed the plan, giving satisfactory results, in line with the expectations. Some components had to be modified and corrections and required features have been identified. In addition, the integration process is based on an appropriate test and verification methodology and framework, allowing the teams for focusing on the technical work.

The choices made during the proposal phase and the early stages of the project proved relevant and effective. The integration done during the first development cycle was successful for most of the implemented components, the interfaces were appropriate, the data model and architecture were easily updated, even if it was not. In the rare cases that led to longer discussions, the approach taken allowed for focusing on the questions to be debated (for example on the geographical coordinate system), instead of the constraints and idiosyncrasies of the implementation.

Indeed, the implementation is still in a very early stage some components are not yet available and others are missing some required functionalities. As a consequence, the integration could only be partially done. Nevertheless, the parts that were integrated worked as expected. The most important task for the next iteration period is to complete and improve the system so that it provides all the basic functionalities that are obligatory to create and execute experiments.

Every new feature that is implemented should be tested through integration tests for compatibility and reliability with the other modules. This includes defining steps for each integration test and executing them by the developers. Each step should be observed for compliance with and deviations from the specifications and marked down. In case of unconformities, the software should be updated and the integration tests should be executed again. The current features that are implemented, have followed these steps and have ended with success for most of them; should a failure have been observed, it is noted and taken into account for correction during the next development cycle.

Part V: 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 82 are considered in the context of the integration.

Table 82: 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/