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	D4.8 - Design and Specification of RAWFIE Components(c)		
Confidentiality	PU	Deliverable type[1]	R
Deliverable File	D4.8	Date	30.06.2017
Approval Status[2]	1st and 2nd Reviewer, WP Leader	Version	1.0
	Giovanni Tusa	Organization	IES Solutions
Phone	+39 095211640	E-Mail	g.tusa@iessolutions.eu

[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
Giovanni Tusa	IES	g.tusa@iessolutions.eu
Marcel Heckel	Fraunhofer	marcel.heckel@ivi.fraunhofer.de
Nikolaos Priggouris	HAI	PRIGGOURIS.Nikolaos@haicorp.com
Kiriakos Georgouleas	HAI	Georgouleas.Kiriakos@haicorp.com
Vasil Kumanov	Aberon	vasil.kumanov@aberon.bg
Ricardo Martins	MST	rasm@oceanscan-mst.com
Damien Piguet	CSEM	damien.piguet@csem.ch
Philippe Dallemagne	CSEM	pda@csem.ch
Kostas Kolomvatsos	UoA	kostasks@di.uoa.gr
Kakia Panagidi	UoA	kakiap@di.uoa.gr
Lionel Blondè	HES-SO	lionel.blonde@hesge.ch

REVIEWERS TABLE

Name	Company	E-Mail
Kakia Panagidi	UOA	kakiap@di.uoa.gr
Kiriakos Georgouleas	HAI	Georgouleas.Kiriakos@haicorp.com

DISTRIBUTION

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

CHANGE HISTORY

Version	Date	Reason for Change	Pages/Sections Affected
0.1	2017-05-15	Start discussions and preparation of the 3rd version of the components design (IES internal)	all
0.2	2017-05-29	TOC / Initial version of contents	all
0.3	2017-06-05	First contributions	Section 4.2 (design of components)
0.4	2017-06-16	Updates to the components' functions and interfaces specifications	Sections 4.1, 4.2, 4.3
0.5	2017-06-27	Updates to the components' functions and interfaces specifications. First internal review round.	Section 4
0.6	2017-06-30	Updates to the components' functions and interfaces specifications. Updates to the global sequence diagrams	Section 4, 5
0.7	2017-07-03	Updates to Section 3. Updates to the components' functions and interfaces specifications. Updates to the global sequence diagrams. New section about security approaches. Ready for review	Section 1, 2, 3, 4, 5, 6
0.8	2017-07-06	Review completed	All
1.0	2017-07-10	Final	All

[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).

Abstract:
As a result of the progresses on tasks T4.2, T4.3 and T4.4, the final version of the Design and Specification of RAWFIE Components deliverable, for the 3rd technical iteration cycle, is released.

Built on the final updated list of architectural components and technological decisions presented in the D4.7, on the 2nd iteration of components' implementation and on the updated requirements' definition of D3.3, this report presents the final modifications to the design, and the interfaces specifications of RAWFIE platform and components.

Keywords:
design, architecture, component, interfaces, workflows, interactions, interfaces, diagrams, methods, classes, deployment, server, scenarios, physical architecture

Part II: Table of Contents-

Part II: Table of Contents- ... 5
List of Figures ... 9
List of Tables ... 11
Part III: Executive Summary ... 12
Part IV: Main Section ... 13
1 Introduction ... 13
1.1 Scope of D4.8 ... 13
1.2 Relation to other deliverables ... 13
2 Overview of changes for the third iteration of the design and specifications of RAWFIE components ... 13
3 Final deployment of the RAWFIE platform ... 14
3.1 Deploy and configuration of the Apache Kafka message bus cluster ... 15
4 Updates on the design and specification of the software components – 3rd iteration ... 17
4.1 Frontend Tier (Web Portal GUI elements) ... 17
4.1.1 Overview ... 17
4.1.2 Wiki Tool ... 17
4.1.3 Resource Explorer Tool ... 18
4.1.4 Booking Tool ... 20
4.1.5 Experiment Authoring Tool ... 24
4.1.6 Experiment Monitoring Tool ... 26
4.1.7 System Monitoring Tool ... 28
4.1.8 UxV Navigation Tool ... 30
4.1.9 Visualisation Tool ... 31
4.1.10 Data Analysis Tool ... 34
4.2 Middle Tier (Services and Communication components) ... 36
4.2.1 Overview ... 36
4.2.2 Testbed Directory Service ... 38
4.2.3 EDL Compiler and Validator ... 44
4.2.4 Experiment Validation Service ... 46
4.2.5 Users & Rights Service ... 47
4.2.6 Booking Service ... 53
4.2.7 Launching Service ... 59
4.2.8 Visualisation Engine ... 63
4.2.9 Data Analysis Engine ... 65

4.2.10 System Monitoring Service 65
4.2.11 Accounting Service 67
4.2.12 Experiment Controller 71
4.3 Testbed Tier (Testbeds and Resources control components) 75
4.3.1 Overview 75
4.3.2 Monitoring Manager 76
4.3.3 Network Controller 80
4.3.4 Resource Controller 83
4.3.5 UxV Proximity component 87
4.3.6 Testbed Manager 91
4.3.7 SFA Aggregate Manager 94
4.3.8 UxV Node 103
5 Global Sequence diagrams showing main RAWFIE processes 108
5.1 Registration of Testbed Resources 108
5.2 Booking Testbed Resources 111
5.3 System Monitoring 113
5.4 Experiment Execution and Monitoring 113
5.5 Experiment Measurements Recording 115
5.6 Authoring and Launching of an Experiment 117
5.7 Data Analysis 119
6 Security considerations 122
6.1 Network topology 122
6.2 Internal communication, encryption and authentication 123
6.3 Message bus access 125
7 Summary and Outlook 125
8 References 127
9 Annex 127
9.1 Detailed description of the API provided by RAWFIE components 127
9.1.1 Testbed Directory Service 127
9.1.2 System Monitoring Service 134
9.1.3 User & Rights Service 135
9.1.4 Booking (Reservation) Service 146
9.1.5 Launching Service 149
9.2 Abbreviations 151

9.3 Glossary 153
A 153
Accounting Service 153
Aggregate Manager 153
Avro 154
B 154
Booking Service 154
Booking Tool 154
C 154
Common Testbed Interface 154
Component 154
D 154
Data Analysis Engine 154
Data Analysis Tool 154
E 154
EDL Compiler & Validator 155
Experiment Authoring Tool 155
Experiment Controller 155
Experiment Monitoring Tool 155
Experiment Validation Service 155
M 155
Master Data Repository 155
Measurements Repository 155
Message Bus 155
Module 155
Monitoring Manager 156
N 156
Network Controller 156
L 156
Launching Service 156
R 156
Resource Controller 156
Resource Explorer Tool 156
Results Repository 156

Resource Specification (RSpec) 156
S 156
Schema Registry 156
Service 157
Slice Federation Architecture (SFA) 157
Subsystem 157
System 157
System Monitoring Service 157
System Monitoring Tool 157
T 157
Testbed 157
Testbeds Directory Service 157
Testbed Manager 158
Tool 158
U 158
Users & Rights Repository 158
Users & Rights Service 158
UxV 158
UxV Navigation Tool 158
UxV node 158
V 158
Visualisation Engine 158
Visualisation Tool 159
W 159
Web Portal 159
Wiki Tool 159

List of Figures

Figure 1: Deployment and configuration of RAWFIE Message bus.. 16
Figure 2: Web Portal – Deployment / Components Diagram.. 17
Figure 3: Resource Explorer Tool - Class diagram 19
Figure 4: Booking Tool - Class diagram.. 23
Figure 5: Class diagram of the Authoring Tool 26
Figure 6: Experiment Monitoring Tool - Class diagram 27
Figure 7: System Monitoring Tool - Class diagram 29
Figure 8: UxV Navigation Tool - Class diagram... 31
Figure 9: Visualisation Tool - Class diagram 33
Figure 10: Middle Tier Components – Deployment / Components Diagram 37
Figure 11: Testbed Directory Service class diagram 40
Figure 12: Experimenter search resources of specific type (USV).. 42
Figure 13: Platform admin registers a new Testbed 43
Figure 14: Register a new UxV resource 44
Figure 15: Class diagram for the ECV... 45
Figure 16: Class diagram for the EVS 47
Figure 17: Users & Rights Service - Class diagram 49
Figure 18: Users & Rights Service – Password-based user login.. 50
Figure 19: Users & Rights Service – Check user authorisation... 51
Figure 20: Users & Rights Service – Check user authorisation... 53
Figure 21: Booking Service - Class diagram 56
Figure 22: Booking Service - Overview 57
Figure 23: Booking Service – Add/Edit a booking... 58
Figure 24: Class diagram of the Launching Service 62
Figure 25: Visualisation Engine - Class diagram 64
Figure 26: System Monitoring Service - Class diagram 67
Figure 27: Accounting Service – Class diagram.. 69
Figure 28: Accounting Service – Register usage information 70
Figure 29: Accounting Service – Register external payment 70
Figure 30: Accounting Service – Payment via payment system.. 71
Figure 31 Experiment Controller - Class Diagram 73
Figure 32 Experiment Controller - Sequence diagram 74
Figure 33: Testbed control, analysis and monitoring– Deployment / Components Diagram 75
Figure 34: Monitoring Manager - Class diagram 78
Figure 35: Monitoring Manager sent and received Message Bus messages 79
Figure 36 Resource Controller - Class Diagram.. 85
Figure 37 - Resource Controller - Sequence diagram.. 86
Figure 38: UxV Node architecure with Proximity component. Dotted line boxes represent hardware. Continuous line boxes represent software components. 89
Figure 39: Testbed Manager - Class diagram 93
Figure 40: Testbed Manager - Experiment handling sequence diagram 94
Figure 41: Aggregate Manager architectural components.. 96
Figure 42: Aggregate Manager - Get SFA-API version sequence diagram 98
Figure 43: Aggregate Manager - retrieve resources information using REST-API sequence diagram 99

Figure 44: Aggregate Manager - retrieve resources information using XML-RPC API sequence diagram ... 99
Figure 45: Aggregate Manager - allocation of resources through REST API ... 100
Figure 46: Aggregate Manager - allocation of resources through XML-RPC API ... 101
Figure 47: Aggregate Manager - create resource sequence diagram ... 102
Figure 48: Aggregate Manager - update resource sequence diagram ... 102
Figure 49: Aggregate Manager - retrieve resource sequence diagram ... 103
Figure 50: Sequence Diagram of "Registration of Testbed Resources" proces ... 110
Figure 51: Sequence Diagram for “Booking Testbed Resources” process ... 112
Figure 52: Sequence Diagram for “Experiment Execution and Monitoring” process ... 114
Figure 53: Sequence Diagram for “Experiment Measurements Recording” process ... 116
Figure 54: Sequence diagram for “Authoring and Launching of an Experiment” ... 118
Figure 55: Sequence Diagram for the “Data Analysis in a streaming scenario” process ... 120
Figure 56: Sequence Diagram for the “Data Analysis in a batch scenario” process ... 121
Figure 57: Network topology ... 122
Figure 58: Simplified diagram for TLS handshake with server and client certificate ... 124

List of Tables

Table 1: net_interfaces testbed manager table 81
Table 2: resources_net_interfaces table of the testbed manager data base 81
Table 3: 2nd iteration dynamic data with topic names 82
Table 4: List of requirements for an UxV node to be used in RAWFIE 108
Table 5: REST methods description for the Testbed Directory Service 127

Part III: Executive Summary

The present document provides the 3rd and final version of the RAWFIE platform and components design and specification. As such, it is delivered at the beginning of the 3rd RAWFIE iteration cycle.

The report starts with some updates about the final deployment of the RAWFIE operational platform from a physical point of view. Among the other information, including information about the installation of RAWFIE components in several servers (namely the Web Application Server instances, the Middle Tier Services Server and System Monitoring Services Server instances, the GIS Server instances, the Master Data, Users & Rights Repositories, Measurements Repository and Analysis Results Repository Server instances, the Testbed Components server instances and the UxV Node Server (normally, the embedded server onboard of each UxV node)), we report in a dedicated sub-section, the last deployment and configuration principle for the Apache Kafka message bus. Each Testbed connected to RAWFIE is going to have its own Kafka Broker (and a corresponding second instance for replication and fault tolerance), that will handle, locally, the communication between its own UxVs and the Testbed components themselves. The "centralised" set of Kafka brokers will still be present at the platform level (currently deployed at UoA premises): the messages exchanged for the local communication on each Testbed will be mirrored in the centralised installation.

Basically the design of all components at the different application tiers ha been updated before starting the 3rd implementation iteration, by either adding or by modifying functionalities and software interfaces specifications. These updates are reported in Section 4 and its sub-sections, where it is also reported how the several components of the platform will be integrated and connected together in the final platform implementation, showing specific workflows through UML sequence diagrams. In the same sections we highlight, for each component and using a specific table template, how the requirements from D3.3 are mapped to software functionalities and / or software interfaces specifications.

The purpose of the adopted design and specification approach is twofold: to present and define how the final functionalities expected for the new development cycle iteration – based on what has been defined in the latest requirements' specification document (D3.3) - will be implemented, and to observe in depth the data flow and interaction between the components specified in the deliverable D4.7.

This final RAWFIE architecture and components' design document ends with a set of UML sequence diagrams, which are again slightly updated with respect to the same diagrams in D4.5, based on the new design modifications. These sequence diagrams – the so called "Global Sequence Diagrams", show how the different software components interact and how the software interfaces are used, in some of the most relevant RAWFIE processes / use cases.

Part IV: Main Section

1 Introduction

1.1 Scope of D4.8

This deliverable describes the final software and physical design of the components belonging to the RAWFIE platform architecture. It presents the concepts for the setup of the physical infrastructure of the platform, the approach for the deployment of the several applications and components by the mean of UML Deployment and components diagrams, and the software classes implementing the required functionalities for each component, starting from the updated list of requirements provided in WP3 (D3.3).

Practically it answers:

- How the requirements defined in D3.3 are translated into software design and in turn into implemented functionalities
- Overall understanding of the operational architecture (physical infrastructure, components' deployment in different servers and execution environments)
- Detailed, development oriented software design of components
- Components interfaces and interactions in some of the most relevant RAWFIE processes or use cases (using sequence diagrams)

1.2 Relation to other deliverables

The 3rd and final iteration of the design and specifications of RAWFIE components in D4.8, is elaborated based on the requirements specification presented in D3.3 – Specification of requirements for the 3rd and final RAWFIE development cycle. The final version of the high level architecture presented in D4.7 provides an input for this deliverable as a general architectural picture, where relations among components is firstly highlighted. The work in this deliverable takes also into account the outcome of 2nd iteration development activities in WP5, and will guide the work for the 3rd iteration development period which will be reported in future WP5 deliverables.

2 Overview of changes for the third iteration of the design and specifications of RAWFIE components

Below we shortly summarise the most important changes made in comparison to D4.5:

- Physical deployment (Section 3) updated with more information on the current setup of the Message Bus cluster
- Deployment diagrams in Sections 4.1.1, 4.2.1 and 4.3.1, updated
- Updated link of components' functionalities with the updated list of requirements from D3.3, for each component in Section 4

- Changes to almost all components design to reflect final design of functionalities and interfaces, with updated class and sequence diagrams, always in chapter 4
- Besides the existing components' updates, the following modifications can be highlighted:
    - Accounting Service definition updated (KillBill integration etc.)
    - Users & Rights Service interface updated
    - In the Testbed Tier, Monitoring Manager is moved inside Testbed Manager and is considered a subcomponent of it sharing the same user interface
- Global sequence diagrams, highlighting some of the most relevant RAWFIE processes / use cases updated in Section 5

3 Final deployment of the RAWFIE platform

The physical elements listed in the following contribute to the RAWFIE physical architecture. All these elements are detailed using UML Deployment diagrams in the subsequent sections of the document.

- Web Application Server
  The server instance/s and the environment where all RAWFIE Frontend tier applications run. Includes a Java Runtime Environment (JRE), and the Apache Tomcat Servlet Container, where the Web Portal components framework is deployed.
- Middle Tier Services Server
  The server instance/s and the environment where all RAWFIE Middle Tier services run. Include a Java Runtime Environment (JRE), and the Apache Tomcat Servlet Container, where the components are deployed.
- Integration Server
  The server / server instances where the Confluent platform and the Message Bus cluster are deployed. Include a Java Runtime Environment (JRE).
- GIS Server
  The server / server instances where the GIS solutions adopted in RAWFIE (Geoserver cluster) is deployed. It may include a local PostgreSQL / PostGIS database or will use the Master Data Repository one, as datasource for storing and serving geographic data as layers.
- Master Data Repository Server
  The server / server instances where the PostgreSQL / PostGIS RAWFIE database is deployed, together with the LDAP directory (OpenDJ [1]).
- Measurements and Analysis Repositories Server
  The server / server instances where the Measurements and Analysis Repositories will be deployed.
- System Monitoring Server
  The server / server instances where the System Monitoring Services and applications run (Icinga Web GUI [2], monitoring DB, JNRPE [3] plugin and System Monitoring Service).

- Testbed Server/s
  Testbed services/components will run on their dedicated HW located at the various remote testbed facilities. Most of the testbed services are expected to run as standalone processes. Components of the Testbed Server/s also include local installation of Apache Kafka brokers, for handling the communication between the connected UxVs and the specific Testbed internal components (Testbed Manager, Resource Controller, and so on).

3.1 Deploy and configuration of the Apache Kafka message bus cluster

Apache Kafka has been chosen for implementing the message bus and as serialization format of the messages on the bus under the concept of “Multiple Nodes- Multiple brokers clusters” instance. A local Message Bus (message broker) is installed within each Testbed. This way, the internal communication between the components in the testbed and the UxVs, e.g. the Resource Controller and the UxVs is be performed in a local, controlled network environment, thus reducing the impact of the network in the latency of the communication. The overall workload in the message bus is reduced (since each broker will just handle its own messages), and the local message bus system are adjusted to the needs of the Testbed itself. Messages from each local broker are mirrored to a centralized Kafka broker deployed in the Cloud, so that Middle Tier components which need to access specific messages (e.g. logs or other data for experiments’ control) can directly access the central message broker rather than each of the local ones, as depicted in Figure 1.
The use of different partitions is selected for the different UxVs in the same Testbed; this ensures that the messages of the various UxVs do not intermix, and it provides much shorter message bus queues dedicated to a particular UxV and much faster response times. A key is attached to each message in which the producer guarantees that all messages with the same key will arrive to the same partition. A topic partition is the unit of parallelism in Kafka.
To overcome the case of repartitioning in case of new testbeds, topics are created with a different name per testbed (i.e prefixing the <testbed_id>_ in each case) only for the messages related with the control of devices. Every testbed broker handles topics different from the others and all the topics are mirrored in the main cluster for redundancy purposes. The UxVs need to consider the testbed identifier in order to know where to send or from where to receive in each case (this can be part of their initial configuration when deployed in a testbed). Partitions of topics in other testbeds is not affected by adding or removing devices or even a whole testbed.

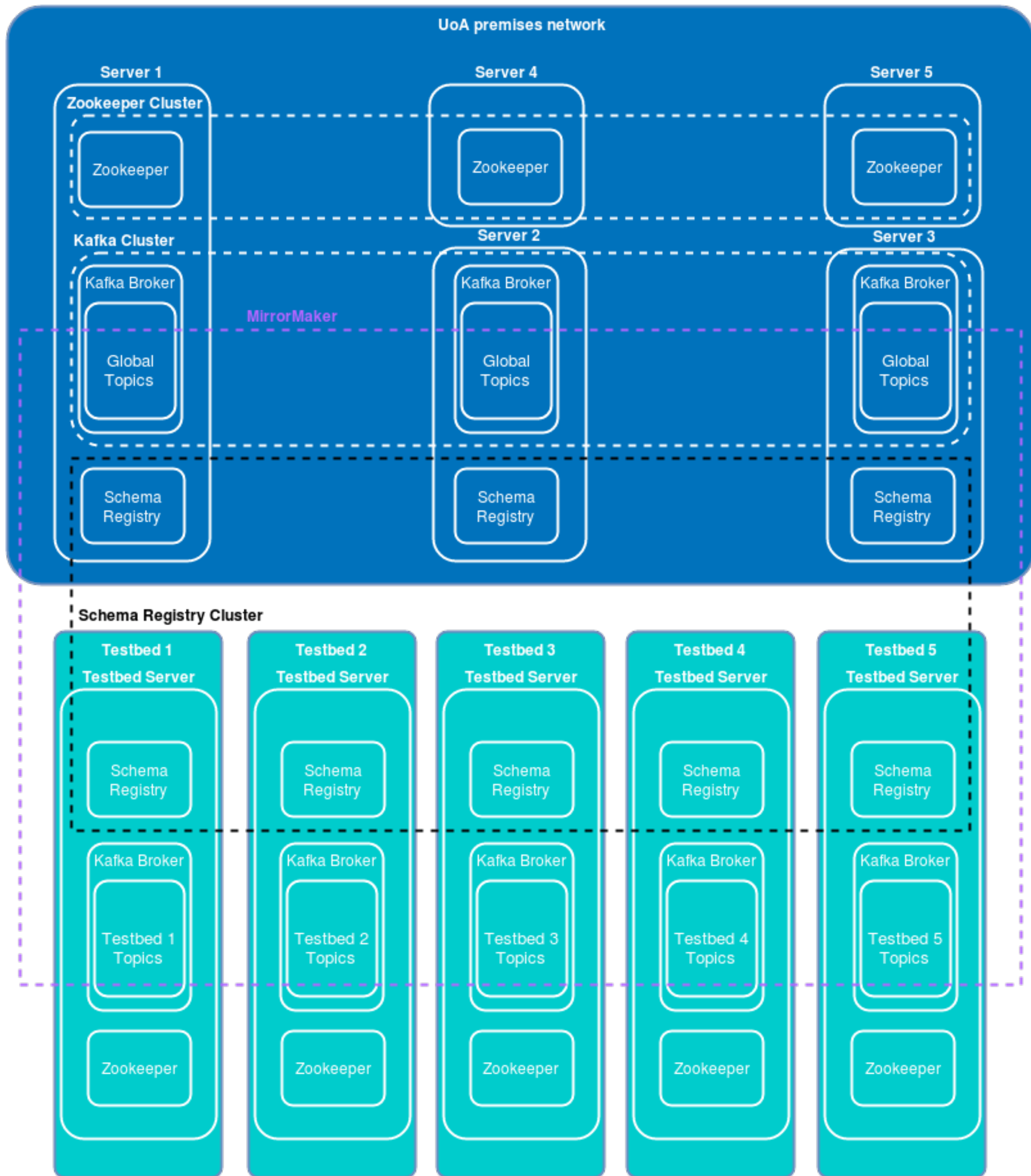

Figure 1: Deployment and configuration of RAWFIE Message bus

4 Updates on the design and specification of the software components – 3rd iteration

4.1 Frontend Tier (Web Portal GUI elements)

4.1.1 Overview

The UML Deployment Diagram of Web Portal components and their interactions with Middle Tier and Data Tier components is presented in Figure 2.

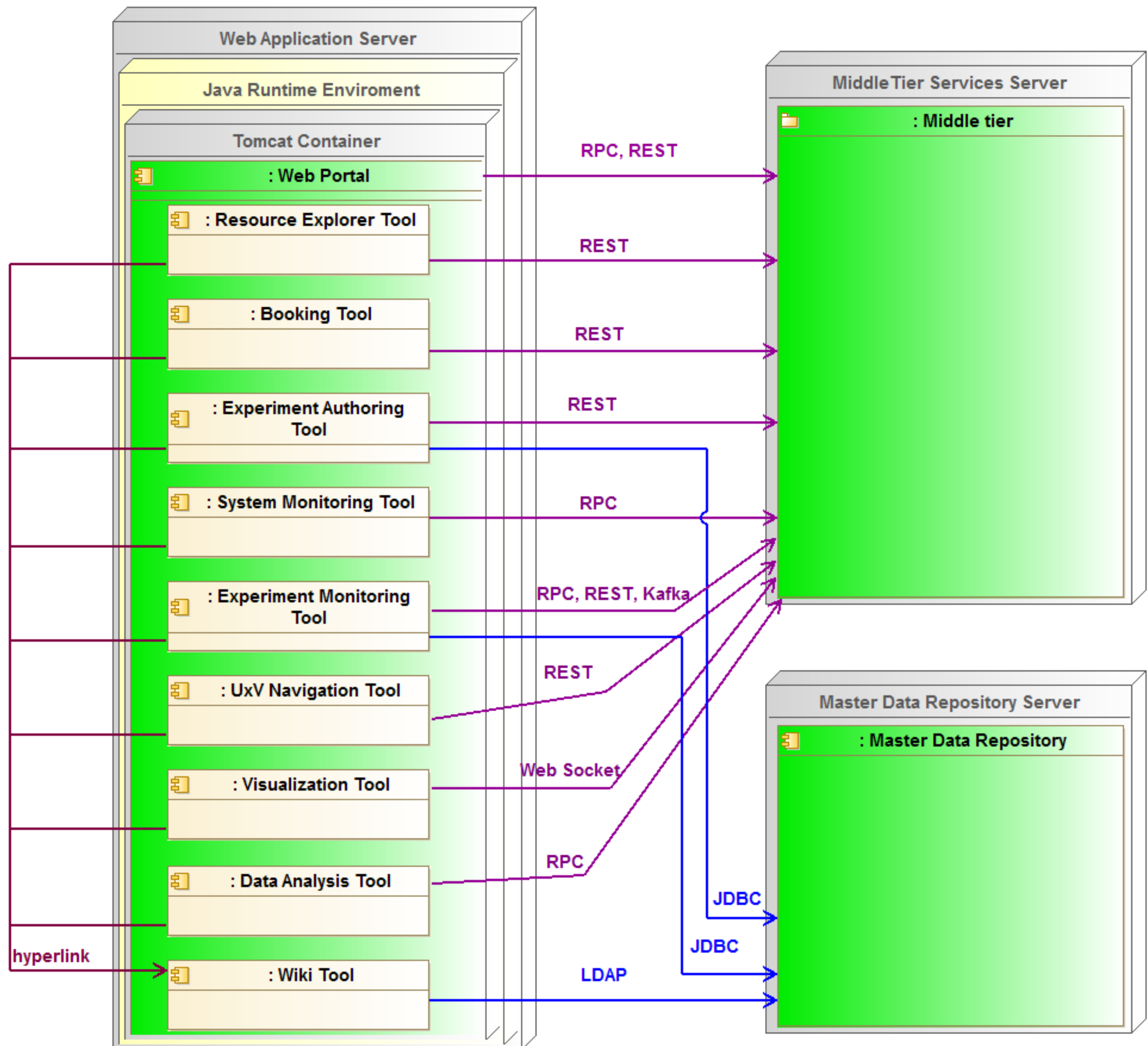

Figure 2: Web Portal – Deployment / Components Diagram

4.1.2 Wiki Tool

All kinds of documentation and tutorials relating to the RAWFIE system will be managed by the Wiki Tool.

4.1.2.1 *Component requirements as identified in D3.3*

ID (Priority)	Description	Requirement Mapping with component's functionalities
PT-WIK-001 (HIGH)	A tutorial or similar type of documentation shall be provided to the users of the platform	The Wiki Tool will be used to manage all manuals, documentation and other information (e.g. extended descriptions of testbeds and UxVs) about the RAWFIE system.
PT-WIK-002 (HIGH)	The Wiki shall use the user credentials from the User & Rights repository	The Wiki Tool will be configured to use the LDAP interface of the User & Rights repository for authentication.
PT-WIK-003 (MEDIUM)	The Wiki should support internationalization and localization	Wiki pages are provided in different languages.
PT-WIK-004	The Wiki should be easy to use and edit	A WYSIWYG editor and full text search is provided by the Wiki Tool

4.1.2.2 *Final specification of functionalities and interfaces*

The third party application "XWiki" [5] is used to realise the Wiki Tool. Beside the HTTP/HTML interface for displaying and editing, no special operations are foreseen.
Using the LDAP-Authenticator, the XWiki will access the User & Rights repository.
All tools in the Web Portal will provide hyperlinks to the wiki pages, that contain manuals or other information about the tool.

4.1.3 Resource Explorer Tool

Via the Resource Explorer Tool, the experimenter can discover and select available testbeds as well as resources inside a Testbed that she/he will utilize to build future experiments.

4.1.3.1 *Component requirements as identified in D3.3*

ID (Priority)	Description	Requirement Mapping with component's functionalities
PT-REE-T-001 (HIGH)	The UI interface shall illustrate testbed and UxV information of the RAWFIE federation that the experimenters should take advantage of	The pages of the IndexPage, ResourcePage and TestbedPage visualise the information
PT-REE-T-002 (LOW)	Registration of testbeds and UxVs may be possible via the Web Portal	EditTestbed and EditResourcePage can be used to add and edit testbeds and UxVs
PT-REE-T-003 (MEDIUM)	RAWFIE platform should provide a Resource Discovery tool for fine-grained resource searches	The SearchPage will provide a search form and results list

PT-REE-T-004 (MEDIUM)	Link to the Booking Tool should be provided	ResourcePage will provide a link to the Booking Tool, so that the current UxV can be booked.

4.1.3.2 *Final specification of functionalities and interfaces*

The Resource Explorer Tool provides several web pages to interact with the Testbed Directory Service. A search page is provided to let the user search for resources that meet his requirements. Specific details of Testbeds and UxVs could be viewed on the details web pages. Adding and editing of Testbeds and UxVs could be done via the edit web pages.

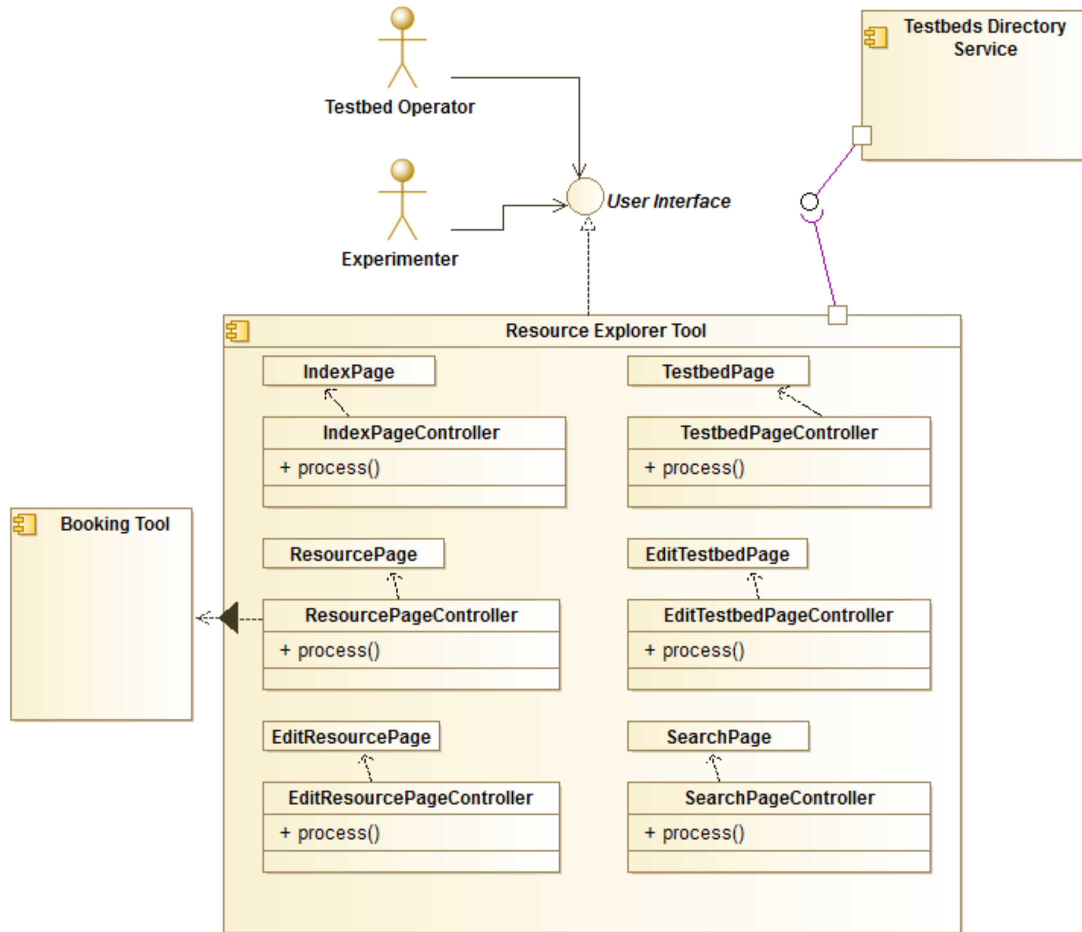

Figure 3: Resource Explorer Tool - Class diagram

The Resource Explorer Tool mainly interacts with the Testbed Directory Service. Please see the section about the Testbed Directory Service for a more detailed description. Additionally, the ResourcePageController can call the BookingTool to directly book the selected resources.

Provided Interfaces

- Web portal GUI:
  Used by the Experimenters to find appropriate Testbeds and UxVs.

Required Interfaces

- Testbed Directory Service Interface:
  Read the resource data for visualisation
- Booking Tool:
  Redirect user (in the browser) to the booking tool, to start booking of the selected resources

4.1.4 Booking Tool

The booking tool provides the front-end that allows a potential user/experimenter to reserve resources to selected Testbeds for a specified period (slice) of time. Booking of resources by the experimenter is a prerequisite in order to be able to assign them later on to an authored experiment (experiment level reservation) and proceed with launching of the actual experiment. In the following section and throughout this document, the terms booking and reservation should be considered interchangeable.

4.1.4.1 *Component requirements as identified in D3.3*

ID (Priority)	**Description**	**Requirement Mapping with component's functionalities**
PT-BOO-T-001 (HIGH)	Booking Tool should allow booking of resources at the experimenter level for a specified period and for selected resources	Mapped by design (*CalendarViewPage* will provide initial selection of time slices while *CreateBookingPage* allows for selection of resources (see also PT-BOO-T-004)
PT-BOO-T-002 (HIGH)	Booking Tool functionality shall be compatible with the SFA architecture and the notion of slices reservations	Aggregate Manager Rest API is contacted during every booking request See also Booking Service section
PT-BOO-T-003 (HIGH)	Booking Tool should delegate all its actions related to Booking of a resource to the Booking Service	Mapped By design (see class diagram)
PT-BOO-T-004 (HIGH)	Booking Tool shall also interact with the Testbeds Directory Service in order to retrieve information on unallocated testbed resources	Mapped by design (*CreateBookingPage* interacts with *Testbed Directory Service,* see class diagram)
PT-BOO-T-005 (HIGH)	Booking Tool should communicate with the underline services using JSON formatted messages (through an RPC or REST API)	Implementation specific (Booking Service will provide an RPC & REST interface enabling communication via Avro JSON messages)
PT-BOO-T-	Booking Tool should	Fulfilled by existence of *CalendarViewPage*

006 (HIGH)	provide appropriate functionality for viewing the reservations of a user/experimenter	
PT-BOO-T-007 (HIGH)	Booking Tool should allow editing of Reservations defined in a future time	Fulfilled by existence of *EditBookingPage*
PT-BOO-T-008 (HIGH)	Booking Tool should allow cancellation of present and future defined Reservations	Fulfilled by existence of *CancelBookingPage*
PT-BOO-T-009 (HIGH)	Booking Tool should allow creation of bookings through an intuitive UI interface	Fulfilled by existence of *CreateBookingPage*
PT-BOO-T-010 (HIGH)	Appropriate notification mechanism should be provided to the user in case status of reservation request is not directly available.	A booking (reservation) *status* field will be included in every booking response message which should be visible in the UI (*CalendarViewPage* and/or *BookingDetailsPage*) See also Booking Service section
PT-BOO-T-011 (MEDIUM)	Booking Tool may provide assistance of feedback to the potential experimenter during the booking process	Email notifications are sent to the experimenter when reservation status changes (i.e. approved or rejected). Failure of booking provides also the reason of failure.
PT-BOO-T-014 (HIGH)	Booking Tool UI interface should be protected with appropriate authorization and differentiate available actions and view based on user and its assigned role	Only authorized access is allowed. Certain actions (Booking Approve or Reject) are available only to users with a certain testbed role. Edit of Booking is allowed only to the user that created it.
PT-BOO-T-015 (HIGH)	Booking Tool should be integrated in the RAWFIE web portal.	Integration in the RAWFIE web portal was achieved during 2nd iteration integration activities
PT-BOO-T-016 (HIGH)	Booking Tool should limit reservation of resources during testbeds operational hours	Operational hours integrated in the Database schema (testbed table). The UI provides a selection for the desired testbed and displays a calendar view only for the operational hours.
PT-BOO-T-017 (HIGH)	Booking Tool should prohibit reservation of the same resource by different users at overlapping time	Booking Tool contacts Booking Service and performs some validation checks prior to booking approval.

	periods	

4.1.4.2 *Final specification of functionalities and interfaces*

The Booking Tool acts as a front-end interface to the Booking Service and provides a set of actions depending on the logged in user role (experimenter or testbed manager) as well as on the reservation status of the booking at the time of the action. The Booking Tool is integrated in the RAWFIE web portal and exposes a set of appropriate web pages that enable the following functionalities:

- View bookings in Calendar View
- View booking details (per booking)
- Create booking
- Edit booking (by experimenter or testbed manager)
- Cancel booking (by experimenter or testbed manager)
- Approve booking (by testbed manager)
- Reject booking (by testbed manager)

In the class diagram that follows the changes compared to the previous version are marked with red colour.

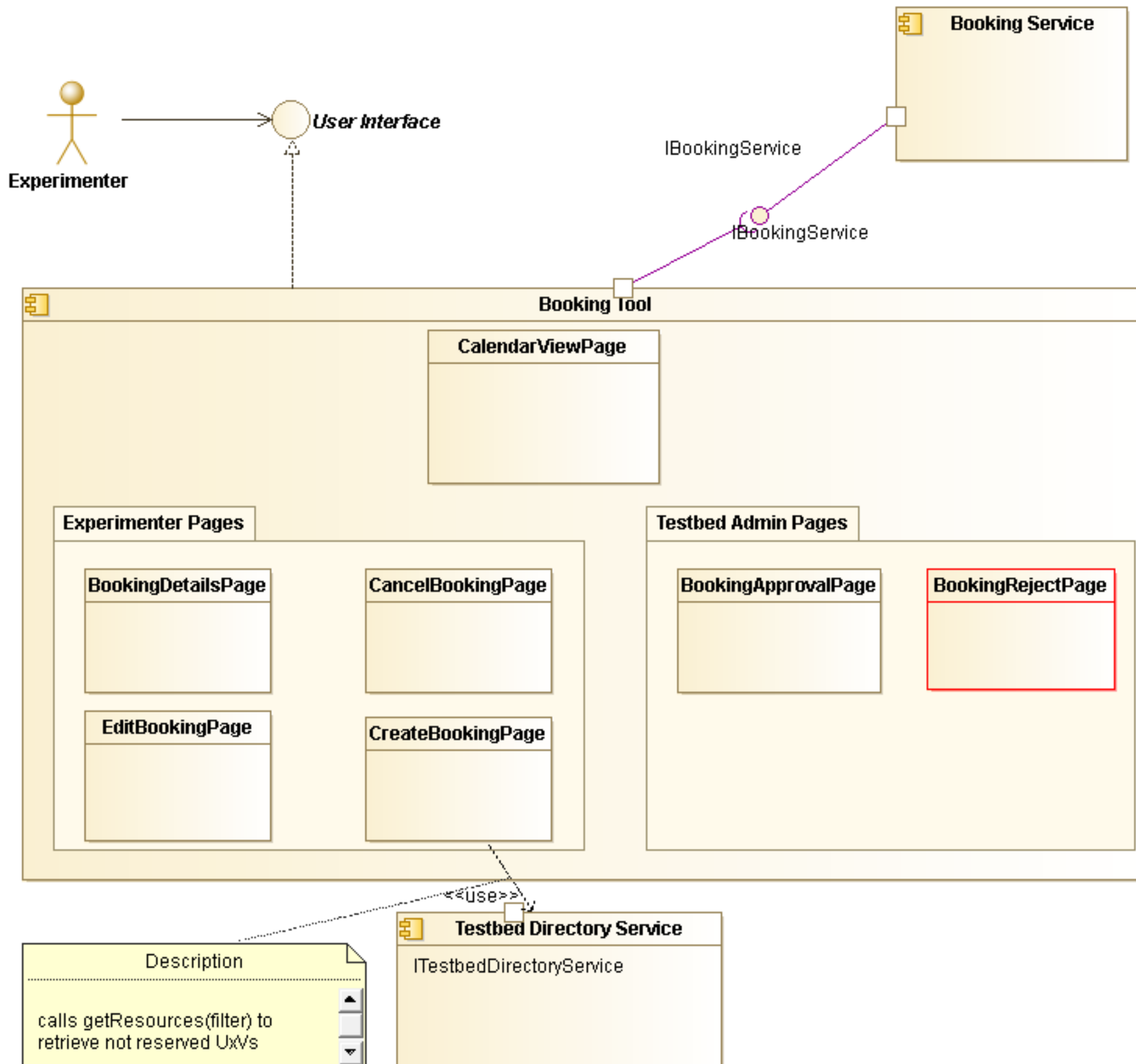

Figure 4: Booking Tool - Class diagram

Provided Interfaces

- Web portal GUI: Used by the Experimenter and/or Testbed Adminsitrotors/Operators

Required Interfaces

- Booking Service: Read the bookings for visualisation, add, edit, reject or aprove bookings
- Testbed Directory Service: to retrieve not booked resources during initial selection for creation of a new reservation

4.1.4.3 *Updated sequence diagrams*

No separate sequence diagrams provided for Booking Tool. See section 4.2.6 (Booking Service) below.

4.1.5 Experiment Authoring Tool

The Experiment Authoring Tool is responsible to provide functionalities to the experimenters that are related to the definition of experiments by using the EDL. Two editors are provided: the textual and the visual editors. These editors incorporate all the necessary functionalities as those found in typical IDEs as well as functionalities related to the compilation and validation of the defined experiments.

4.1.5.1 *Component requirements as identified in D3.3*

ID (Priority)	Description	Requirement Mapping with component's functionalities
PT-EXA-T-001 (High)	Experiment Description Language (EDL) shall be used as a language for the definition of experiment scenarios	The EDL model is already in place
PT-EXA-T-002 (High)	The EDL should allow the definition of all necessary requirements for an experiment	The EDL model offers the necessary terminology
PT-EXA-T-003 (Medium)	For each defined experiment specific metadata, i.e. name, version, date and description shall be defined	The EDL model offers the necessary terminology
PT-EXA-T-004 (High)	An experimenter shall be able to provide initial conditions and/or configuration parameters for an experiment	The EDL model offers the necessary terminology
PT-EXA-T-005 (High)	An experimenter shall be able to manage/guide the available booked resources during experiment authoring	The EDL model offers the necessary terminology
PT-EXA-T-006 (Medium)	An experimenter shall be able to define the type of information to be gathered and/or stored by UxV resource(s)	The EDL model offers the necessary terminology
PT-EXA-T-007 (Medium)	An experimenter shall be able to define the type of metrics to be gathered and/or stored during an	The EDL model will offer the necessary terminology

	experiment and/or per UxV resource	
PT-EXA-T-008 (High)	An experimenter shall be able to provide navigation or movement directives during experiment authoring	The EDL model offers the necessary terminology The Textual editor also supports this functionality
PT-EXA-T-009 (High)	An experimenter should be able to provide formation information for a group of UxVs resources	The EDL model offers the necessary terminology The Textual editor also supports this functionality
PT-EXA-T-010 (High)	A textual editor shall be provided for the authoring of RAWFIE experiments	The Textual editor is already in place
PT-EXA-T-011 (High)	A visual/graphical editor shall be provided for the authoring of RAWFIE experiments	The Visual editor is already in place
PT-EXA-T-012 (High)	Platform shall allow saving, editing and/or deletion of an experiment defined via EDL	The Textual/Visual editor offers this functionality
PT-EXA-T-013 (High)	The visual editor should allow the definition of movement and location waypoints from a map	The Visual editor offers this functionality
PT-EXA-T-014 (Medium)	During authoring of an experiment selection of resources should be limited only to the ones previously reserved from the user at the foreseen time of experiment	The Textual/Visual editor offers this functionality
PT-EXA-T-015 (High)	Validation of EDL script should be possible prior to or during saving	The Textual/Visual editor offers this functionality

4.1.5.2 *Final specification of functionalities and interfaces*

Provided Interfaces

- Web portal GUI:
  Used by the experimenter to access the Experiment Authoring Tool

Required Interfaces

The Experiment Authoring Tool requires interfaces from the following backend services:

- EDL Compiler and Validator Service: perform compilation, recognize syntactic errors and warnings and generate the appropriate code

- Experiment Validation Service: perform the experiment validation (efficient experiment execution in the respective testbed)
- Experiment and EDL Repository: request saved EDL scripts, EDL language elements, store EDL script
- Launching service: request interface to set the appropriate launching time the experiment to be performed

All the above interfaces are implemented as planned. The following figure presents the high level class diagram depicting the interfaces with other RAWFIE components.

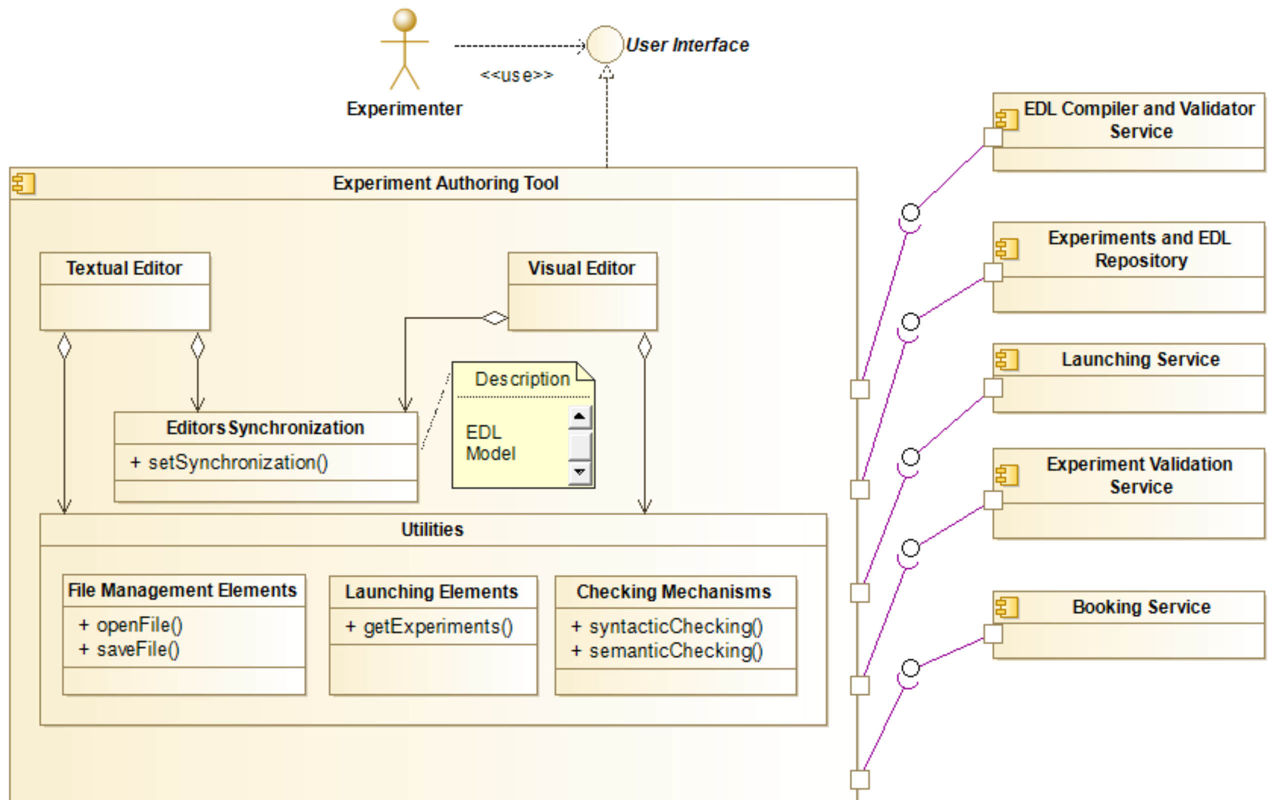

Figure 5: Class diagram of the Authoring Tool

4.1.5.3 *Updated sequence diagrams*

There are not any updates in the sequence diagrams of the authoring tool. Please refer in D4.5 for details.

4.1.6 Experiment Monitoring Tool

Experiment Monitoring Tool collects and displays the information regarding experiments and the resources used by them.

4.1.6.1 *Component requirements as identified in D3.3*

ID (Priority)	Description	Requirement Mapping with component's functionalities
PT-EXM-T-001 (HIGH)	A RAWFIE user should be able to view an overview of his/her experiments	The ExperimentSelectionPage shows all experiments of the logged-in user
PT-EXM-T-002 (MEDIUM)	Experiment Monitoring and Visualisation should be integrated	A direct integration will not be done. Nevertheless, a better linkage will be the realized. E.g. a link to directly start the visualisation from this tool.
PT-EXM-T-003 (MEDIUM)	Cancellation of running experiments should be possible via Web Portal	The ExperimentStatusPage will provide a button to cancel an experiment.

4.1.6.2 *Final specification of functionalities and interfaces*

The logged-in user can first select the experiment of interest from a list of experiments, on which he has appropriate rights. On the "ExperimentStatusPage" the status information of the selected experiment will be displayed. Also a hyper line is present, to start the visualisation of the experiment.

The ExperimentStatusManagement will collect and prepare the data for displaying in the web page. For this, it communicates with the System Monitoring Service and the Master Data repository. To start or cancel an experiment the Launching Services is called. It also listens to the message bus to perform update on special events.

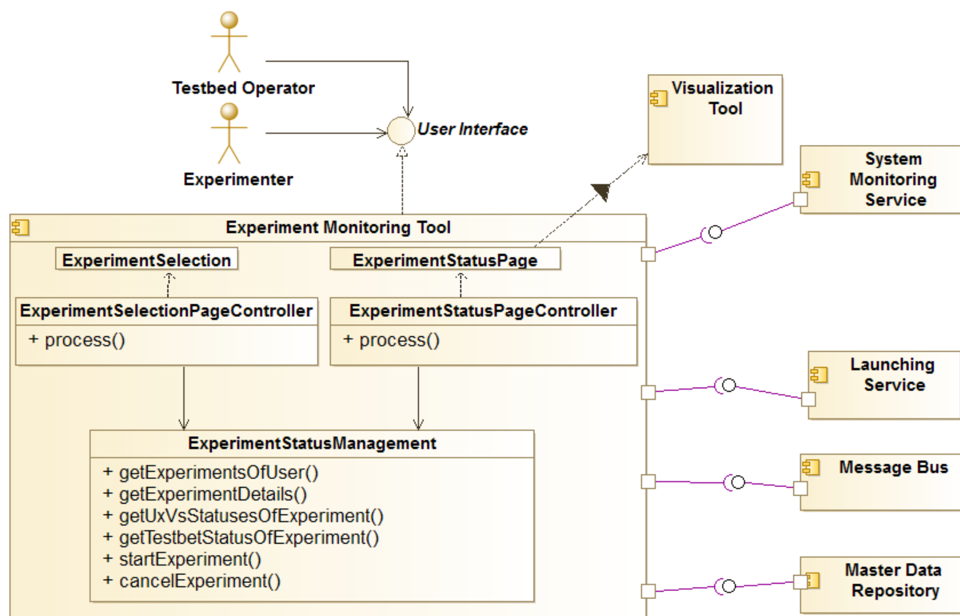

Figure 6: Experiment Monitoring Tool - Class diagram

Provided Interfaces

- Web portal GUI:
  Used by the users (Experimenter, Testbed Operator) to get status information about experiments or to cancel experiments.

Required Interfaces

- System Monitoring Service:
  Get status information about involved testbeds and UxVs.
- Master Data Repository
  Query the experiments of a user.
  Query information about the experiment status.
- Launching Service:
  To manually start or cancel an experiment the Launching Service is called to execute the necessary steps
- Message Bus: Listen to special event to trigger the status update process.

4.1.6.3 *Updated sequence diagrams*

No updates (refer to D4.5 section 4.1.6 for details)

4.1.7 System Monitoring Tool

Shows the status and the readiness of the various RAWFIE services (mainly the ones residing in the middle tier).

4.1.7.1 *Component requirements as identified in D3.3*

ID (Priority)	Description	Requirement Mapping with component's functionalities
PT-SYM-T-001 (HIGH)	Listing and/or visualisation of current system health status shall be available.	StatusDashboardPage and also the third party Icinga Web provide this functionalities
PT-SYM-T-002 (MEDIUM)	The current system health status should be grouped thematically.	StatusDashboardPage will do this in future
PT-SYM-T-003 (MEDIUM)	Filtering of the accessible component health statuses by user roles/rights should be possible.	StatusDashboardPage will do this in future
PT-SYM-T-004 (MEDIUM)	The health statuses webpage should be updated automatically.	StatusDashboardPage and also the third party Icinga Web provide this functionalities

4.1.7.2 *Final specification of functionalities and interfaces*

The System Monitoring Tool loads all its information from the System Monitoring Service and displays them in an appropriate way.

The third party application Icinga Web will display detailed status information for Platform Administrators. The simplified StatusDashboardPage will be public available to all RAWFIE users to get informed about the system state.

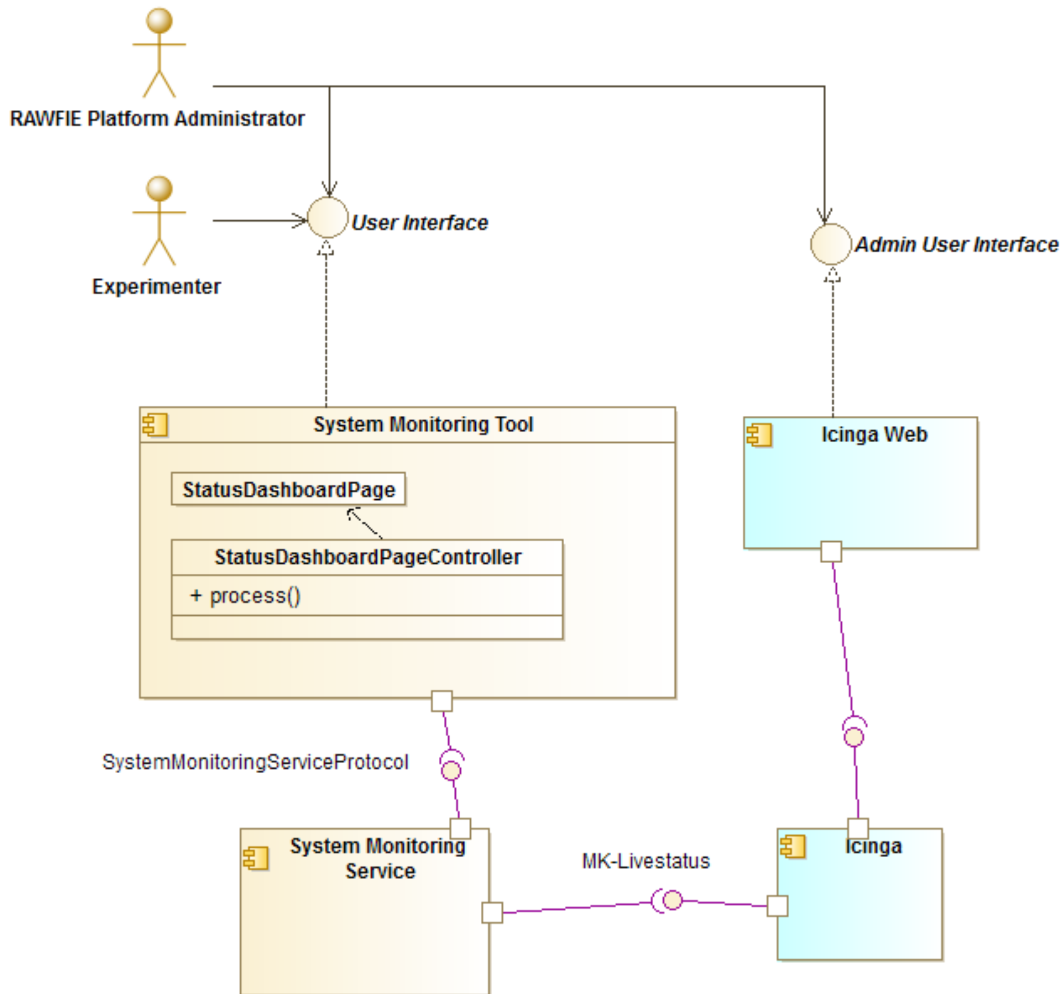

Figure 7: System Monitoring Tool - Class diagram

The System Monitoring Tool only interacts with the System Monitoring Service. Please see the section about the System Monitoring Service to get a more detailed description.

Provided Interfaces

- Web portal GUI:
  Used by the users (Experimenter, RAWFIE Platform Administrator) to get system status information
- Icinga Web:
  RAWFIE Platform Administrator uses this to get detailed system status information

Required Interfaces

- System Monitoring Service:
  Reads the system status from the middleware service for visualisation in the appropriate web pages

4.1.8 UxV Navigation Tool

This component will provide to the user the ability to remotely navigate a squad of UxVs. The UxV Navigation Tool will provide the ability to non-expert users to remotely guide a squad of robotic vehicles to perform basic navigation missions such as waypoint navigation, map construction, area surveillance and path planning.

4.1.8.1 *Component requirements as identified in D3.3*

ID (Priority)	Description	Requirement Mapping with component's functionalities
PT-NAV-T-001 (HIGH)	This component will provide to the user the ability to remotely navigate a squad of UxVs through a user friendly interface.	Instuctions_Manager will implement this in the 3rd iteration of development
PT-NAV-T-002 (HIGH)	This tool provides some basic validation of the user's instructions	Instuctions_Manager
PT-NAV-T-003 (HIGH)	UxV Navigation Tool should be available for the navigation of all moving resources. Real time navigation may be restricted by the communication technology of the UxV data transmission.	Instuctions_Manager will implement this in the 3rd iteration of development
PT-NAV-T-004 (HIGH)	UxV Navigation Tool should be available to read from the database a detailed version of the map of the available areas	Initialization will will implement this in the 3rd iteration of development

4.1.8.2 *Final specification of functionalities and interfaces*

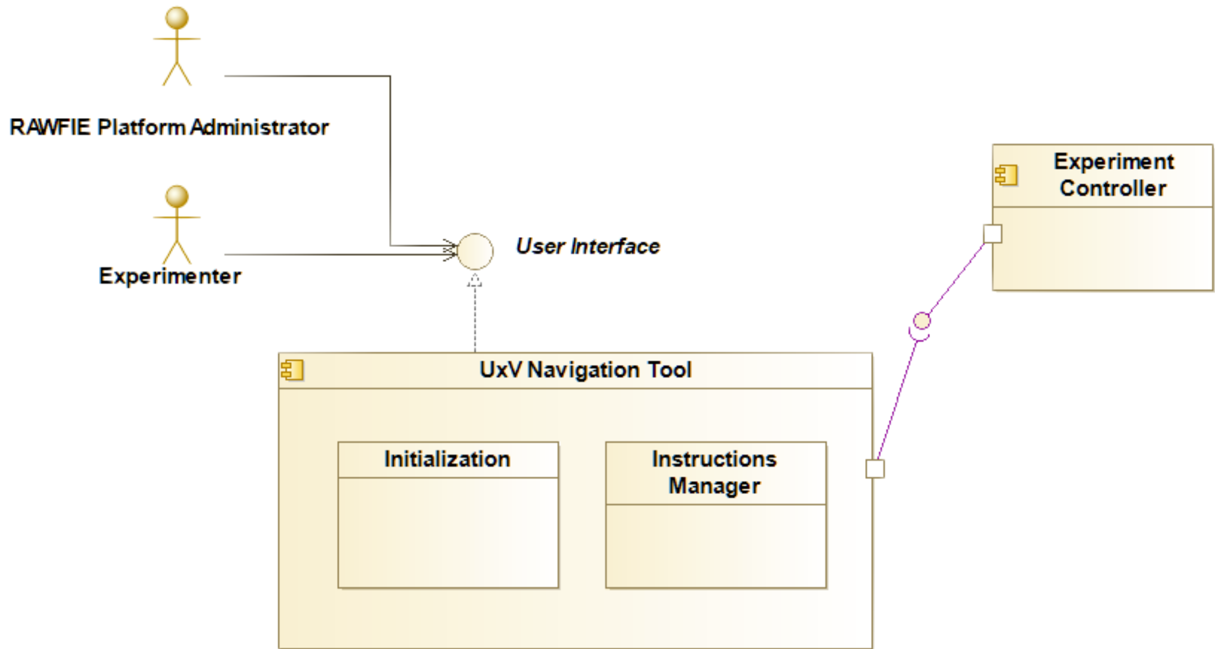

Figure 8: UxV Navigation Tool - Class diagram

Provided Interfaces

- Web portal GUI:
  Used by the users (Experimenter, Testbed Operator) to get instructions.

Required Interfaces

- Experiment Controller Interface: So as to initialize the experiment and to transfer the user's instructions
- Experiment Monitoring Tool Interface: Although there is no direct connection between these two components, the Experiment Monitoring Tool is required so as to inform the experimenter about the current status of the experiment. Additionally, Experiment Monitoring Tool is responsible for the cancellation of an experiment. Experiment Controller is responsible for transferring messages between these two components.

4.1.8.3 *Updated sequence diagrams*

No updates to the sequence diagram/s of this component. Please refer to the ones reported in D4.5.

4.1.9 Visualisation Tool

The Visualisation Tool provides visualisation of the geospatial data of a running experiment. Further, it enables the user to show and track all UxV resources and to apply additional modifications (layers, filters, etc.) to the geospatial data and to show different sensor data, GPS coordinates and others.

4.1.9.1 *Component requirements as identified in D3.3*

ID (Priority)	Description	Requirement Mapping with component's functionalities
PT-VIS-T-001 (HIGH)	The Visualisation Tool shall allow the visualisation of information about the running experiments, in tabular/graphical form	When Start Visualisation is initiated, the experiment will be visible
PT-VIS-T-002 (LOW)	A 3D visualization should be available for the tracking of all moving resources	Moving the camera provides 3D view. An option that will be provided only if 3D mapping is presented for a specific testbed
PT-VIS-T-003 (LOW)	The Visualisation Tool may allow visualisation of video streams coming from the experiment, and experiment's camera control	Launching a widget if this option is available on the UxV will show the video stream
PT-VIS-T-004 (MEDIUM)	The Visualisation Tool shall provide access to information / features associated to each UxV device on the geographic map	When clicking on a vehicle, its information/features will be visible in a widget
PT-VIS-T-005 (MEDIUM)	The Visualisation Tool shall allow organization and manipulation of multiple geographic layers	A button similar to the one for switching external providers, will give the option to switch between layers
PT-VIS-T-006 (MEDIUM)	Possibility of Adding/Removing/Updating graphical widgets should be provided	Widgets can be opened and closed with a mouse click
PT-VIS-T-007 (MEDIUM)	Possibility to display both actual and expected UxVs' route and position should be provided	When experiment is started, both routes and position are visualised

4.1.9.2 *Final specification of functionalities and interfaces*

The VT has the following tasks:

- Handle experimenter requests for manipulating the geo information data. These requests will be sent to the VE over the Websocket (WSSteam and WSData channels), but the response will be received over the GIS (Map) channel. These manipulations include

moving the map, panning, tilting, zooming etc. and also showing/hiding different layers like thermal layers, roads, obstacles and others

- Handle experimenter requests for camera manipulation. They will be handled internally without sending requests to the VE
- Handle experimenter requests for showing/hiding widgets on the screen. These widgets can represent speed of UxVs, GPS positions, different sensor data and other information. This data will be received from the VE over the websocket
- Convert the geo information data in the appropriate format for visualising by the web map library
- Plot the whole information in the browser window appropriately in an easily understandable manner in order to allow the experimenter to properly and successfully execute the experiment

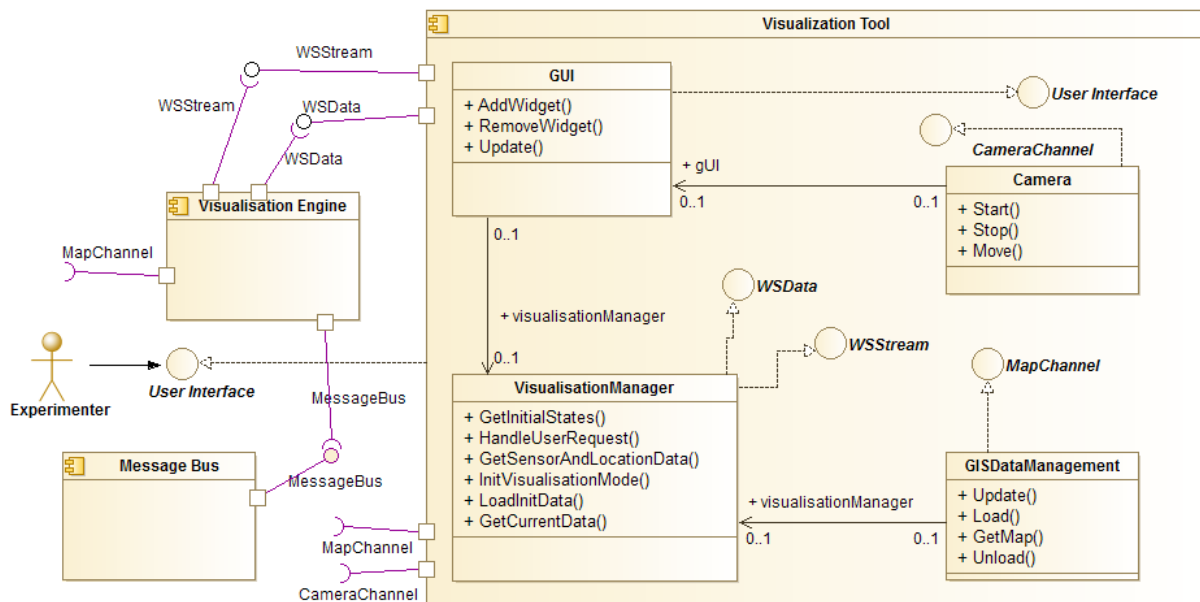

Figure 9: Visualisation Tool - Class diagram

Required interfaces

- The GIS interface is used to send geographical information in various formats like WMS, WFS, WPS and WCS from the VE to the VT. The VT requests map information over the websocket and the geo-information data is sent over the GIS interface.
- The websocket is used in both directions to retrieve information like sensor data from VE to VT or to inform the VE that the experimenter changed a layer in the VT and it needs to be reloaded from the VE.
- Video stream from a camera in order to visualise a camera input from an UxV

Provided interfaces

- The VT has interface to the experimenter through a web-browser, allowing it to receive commands from a mouse or keyboard and to manipulate the layout of the visualisation like switching on/off widgets/layers/maps etc.

4.1.9.3 *Updated sequence diagrams*

No updates (refer to D4.5 for details)

4.1.10 Data Analysis Tool

The Data Analysis Tool is the child-component of the Web Portal through which the user is able to use functionalities provided by the Data Analysis Engine (Zeppelin GUI) as well as being able to browse schemas currently present in the schema registry and select fields from a given schema for later analysis (Schema Registry GUI).
Zeppelin is an interactive notebook interface that sits on top of the Data Analysis Engine. It provides access to various interpreters such as jdbc, psql, python, and more importantly a spark interpreter directly linked to the Compute Engine subcomponent of the Data Analysis Engine. Via the Zeppelin interface, the user can create notebooks and design data analysis jobs using the spark interpreter with the Scala programming language. It also enables tables and plots embeddings, via the appropriate interpreter. The user can use several interpreters in a single notebook. In order to design data analysis jobs, a lot of functions performing analytical tasks or data structures manipulation are provided in built in packages.
Since the Data Analysis Engine enables the execution of both batch and streaming analysis jobs, data sources can differ. Batch data can be fetched from a database, provided the appropriate information is provided through the spark call in the Zeppelin notebook. Streaming data can be sequentially acquired through the message bus (Kafka), directly queryable and fetchable from within Zeppelin.
In order to enable the user to select which data to retrieve from the message bus (which Kafka topic to subscribe to), the former can use the Schema Registry GUI of the Data Analysis Tool. Via this interface, the user can browse the available schemas in the schema registry (whose presence does not guaranty that data is actually being published under it on the message bus), and select the field or fields of interest for later analysis. The user will then be redirected to a newly created Zeppelin notebook containing a flattened structure of the selected information and can then design the analysis job in the continuity of the same notebook.
The Data Analysis Tool also provides access to a dashboard (Grafana GUI) sitting on top of the Whisper time series database which enables the visualization of the results of data analysis tasks conducted on streaming data.
Finally, the Data Analysis Tool provides a view of the spark master GUI to monitor the activity of the different components of the Data Analysis Engine.

4.1.10.1 *Component requirements as identified in D3.3*

ID (Priority)	Description	Requirement Mapping with components functionalities
PT-DAA-T-001 (MEDIUM)	Analysis tool will provide interface to data engine	All the parameters selected by the user through the interface provided by the Data Analysis Tool (schemas, fields, models, etc.) will enable the Data Analysis Engine to compile this information into an analytics task.
PT-DAA-T-002 (LOW)	Analysis tool will provide access to past experiments	The Graphite dashboard will be integrated in the tool, enabling visualization of results contained in the results repository.

PT-DAA-T-003 (MEDIUM)	Analysis tool will provide ability to query message bus streams	The tool will provide the ability to query the schema registry in the message bus. The desired available schemas can then be specified as parameters in any data analysis task definition.
PT-DAA-T-004 (MEDIUM)	Analysis tool will provide interface to end running jobs	The tool will provide the ability to send a kill signal for a specified running task which will interrupt the associated task's execution.
PT-DAA-T-005 (MEDIUM)	Analysis tool will provide a simple metric selection interface, a view of the result stream and the job status tab	The tool will provide task parameter selection forms, a Graphite dashboard integration and a Spark job-tracker page integration.

4.1.10.2 *Final specification of functionalities and interfaces*

Provided interfaces:

Web Portal GUI: the Data Analysis Tool is available through the Web Portal GUI and provides its functionalities via distinct tabs:

- Schema Registry GUI: enables the user to browse the schemas available on the schema registry and select the fields of the targeted schema that the user wishes to run a data analysis job on. Upon validation, selected entities will be embedded automatically into a new notebook in Zeppelin, after which the user can design the analytics task.
- Zeppelin GUI: interactive notebook interface that enables the user to design its own data analysis experiments on the selected data. Various data analysis algorithms and data manipulation functions are provided as part of built-in packages of Zeppelin, transparently available to be used in the design of a notebook. Since each block in a notebook is tied to an interpreter, the user willing to design an analytics job from scratch without the provided functions can do so.
- Spark master GUI: enables the user/administrator to monitor Zeppelin activity, which is the Data Analysis Engine Frontend.
- Grafana/Graphite GUI: enables the user to visualize what the running streaming analysis job send to the Analysis Results Repository in real time. The user can select the time window displayed and can therefore come back to see the sent results at a specific period of time in the past.

Required interfaces:

- Data Analysis Engine
- Measurements Repository
- Analysis Results Repository
- Message Bus

4.1.10.3 *Updated sequence diagrams*

For Sequence Diagrams where the Data Analysis Tool is involved, please refer to Section 5.7.

4.2 Middle Tier (Services and Communication components)

4.2.1 Overview

Middle Tier services provide most of the business logic needed to serve the users' request coming from the Frontend Tier, to get access to the data repositories on the Data Tier, and for the interaction with the Testbed Tier software components through the Message Bus. The UML Deployment Diagram of the Middle Tier components, showing the servers and the execution environments for the deployment of Middle Tier services, together with the internal interaction between services, as well as with Web Portal components, Testbed components, the GIS Server and the Data Repositories, is shown in Figure 10 below. In the picture, the following main interactions are highlighted:

1. LDAP interface to the LDAP Directory Service, used by the Users & Rights Service
2. REST/RPC API provided by RAWFIE services
    - different components, especially the ones belonging to the Frontend Tier, use the interface provided by the Users & Rights Protocol for authentication purposes
    - all or almost all components from the Frontend Tier use the REST/RPC API provided by the corresponding services in the Middle Tier, e.g. the Booking Tool uses the one provided by the Booking Service, the Resource Explorer Tool the one provided by the Testbed Directory service, and so on
3. JDBC/JPA connection to the Master Data Repository
    - direct JDBC/JPA connection to the database is used, in the Middle Tier, by the Testbed Directory Service, the Experiment Controller, the Booking Service and the Launching Service
4. SFA AM REST API
    - in the Middle Tier, the only service interacting with the SFA Aggregate Manager API is the Booking Service
5. Message Bus
    - the Launching Service, the Visualisation Engine, the Data Analysis Engine and the Experiment Controller, among the others, interact and get access to information coming from the UxVs / the Testbed Tier, by interfacing with the Kafka Message Bus
6. Mail Service:
    - accessed by the System Monitoring Service (write), the Booking Service (read) and the Accounting Service (read/write)
7. KCQL (Kafka Connect Query Language / Protocol [15]) interface is used by the Kafka HBase Connector (Measurements Backend Service in the diagram), to push measurements from the Message Bus to the Measurements Repository
8. HBase Java / REST API used by the Data Analysis Engine to get access to the HBase tables in the Master Data Repository, for batch analysis as explained in the following of the document

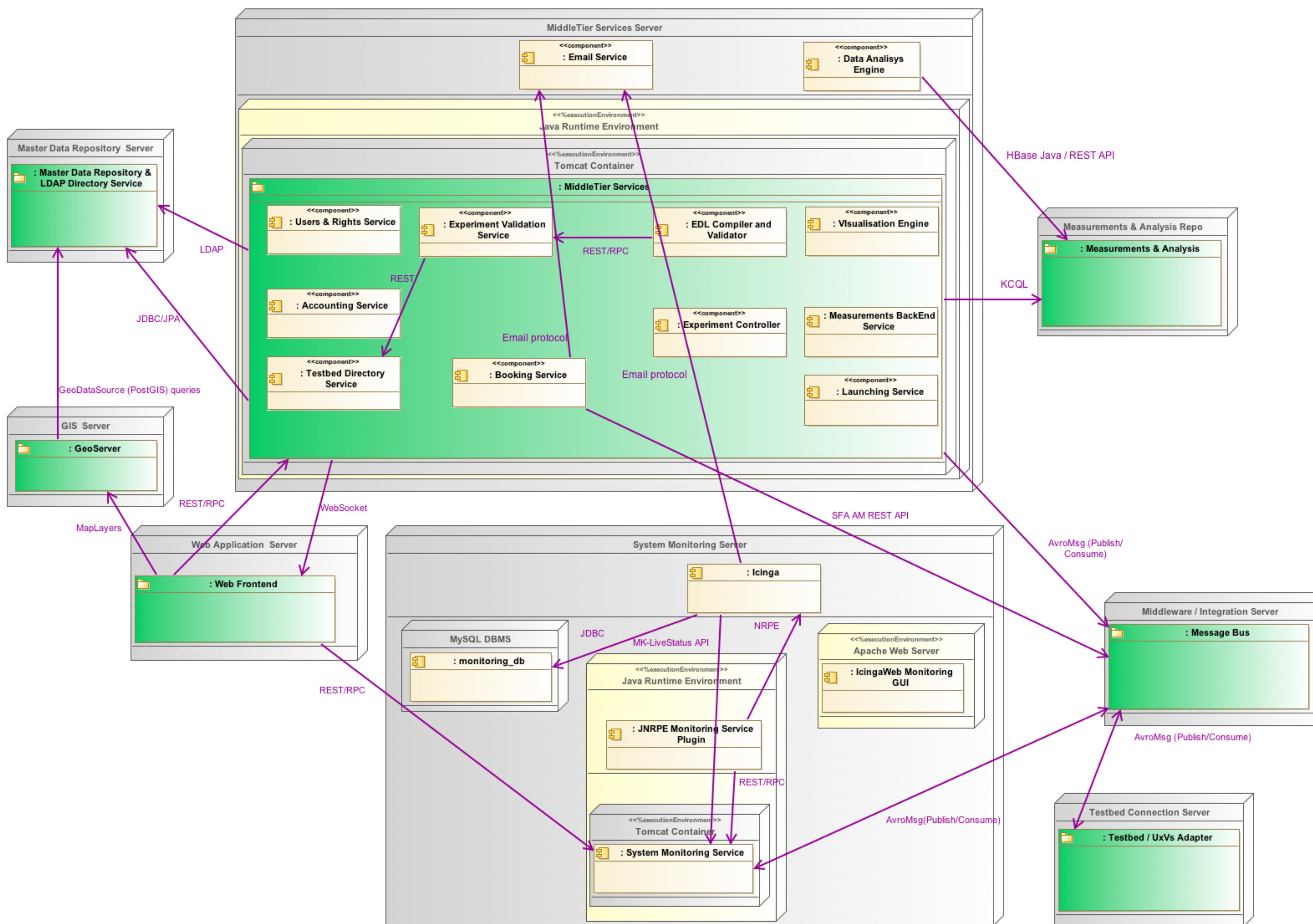

Figure 10: Middle Tier Components – Deployment / Components Diagram

4.2.2 Testbed Directory Service

4.2.2.1 *Component requirements mapping, as identified in D3.3*

ID (Priority)	Description	Requirement Mapping with component's functionalities
PT-DIR-S-001 (HIGH)	The Testbed Directory Service shall provide REST / Web Service API to access information on all Testbeds registered in RAWFIE	Implementation of the *getAllTestbeds* REST interface. Provides access to information about Testbeds registered in RAWFIE
PT-DIR-S-002 (MEDIUM)	The Testbed Directory Service should provide REST / Web Service API to access information on all Testbeds registered in RAWFIE according to predefined filters	Implementation of REST interfaces for filtering Testbeds by: • *id* • *name* • *UAV support* • *UGV support* • *USV support* • *AUV support* • *a combination of health and status values*
PT-DIR-S-003 (HIGH)	The Testbed Directory Service shall provide REST / Web Service API to access to information about available Resources (UxVs) belonging to the Testbeds registered in RAWFIE	Implementation of the *getAllResources* REST interface. Provides access to information about all Resources registered in RAWFIE. Implementation of a REST interface for getting Resources belonging to a specific Testbed (by Testbed *id*)
PT-DIR-S-004 (MEDIUM)	The Testbed Directory Service should provide REST / Web Service API to access to information on available Resources (UxVs) belonging to the Testbeds registered in RAWFIE, and according to predefined filters.	Implementation of REST interfaces for filtering Resources by: • *id* • *name* • *testbed id* • *a combination of health, status and type*
PT-DIR-S-005 (HIGH)	The Testbed Directory Service should provide the possibility to register new Testbeds in the RAWFIE platform, as well as to unregister (delete) testbeds from the platform	Implementation of the *createTestbed* REST interface. Allows the registration of a new Testbed, by providing input with Testbed information in JSON format Implementation of the *editTestbed* REST interface. Allows to update information for a given Testbed, by providing input with updated

		Testbed information in JSON format Implementation of the *deleteTestbed* and *deleteTestbedParameters* REST interface. Provide the possibility to delete a Testbed, specifying the Testbed id
PT-DIR-S-006 (MEDIUM)	Some basic query capabilities should be provided	See API associated to PT-DIR-S-002 and PT-DIR-S-004
PT-DIR-S-007 (HIGH) (from D3.2)	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	Implementation of the *createResource* REST interface. Allows the registration of a new Resource, by providing input with Resource information in JSON format Implementation of the *editResource* REST interface. Allows to update information for a given Resource, by providing input with updated Resource information in JSON format Implementation of the *deleteResource* and *deleteResourceParameters* REST interface. Provide the possibility to delete a Resource, specifying the Resource id

The Testbed Directory Service is a registry service of the middleware tier, where all the integrated testbeds and resources accessible from the RAWFIE facilities can be registered, deleted, modified or listed (filtered).

4.2.2.2 *Final specification of functionalities and interfaces*

Following Figure 11 provides an updated picture of the Testbed Directory Service class diagrams. In red are highlighted to the classes which have been modified, compared to the previous version described in D4.5.

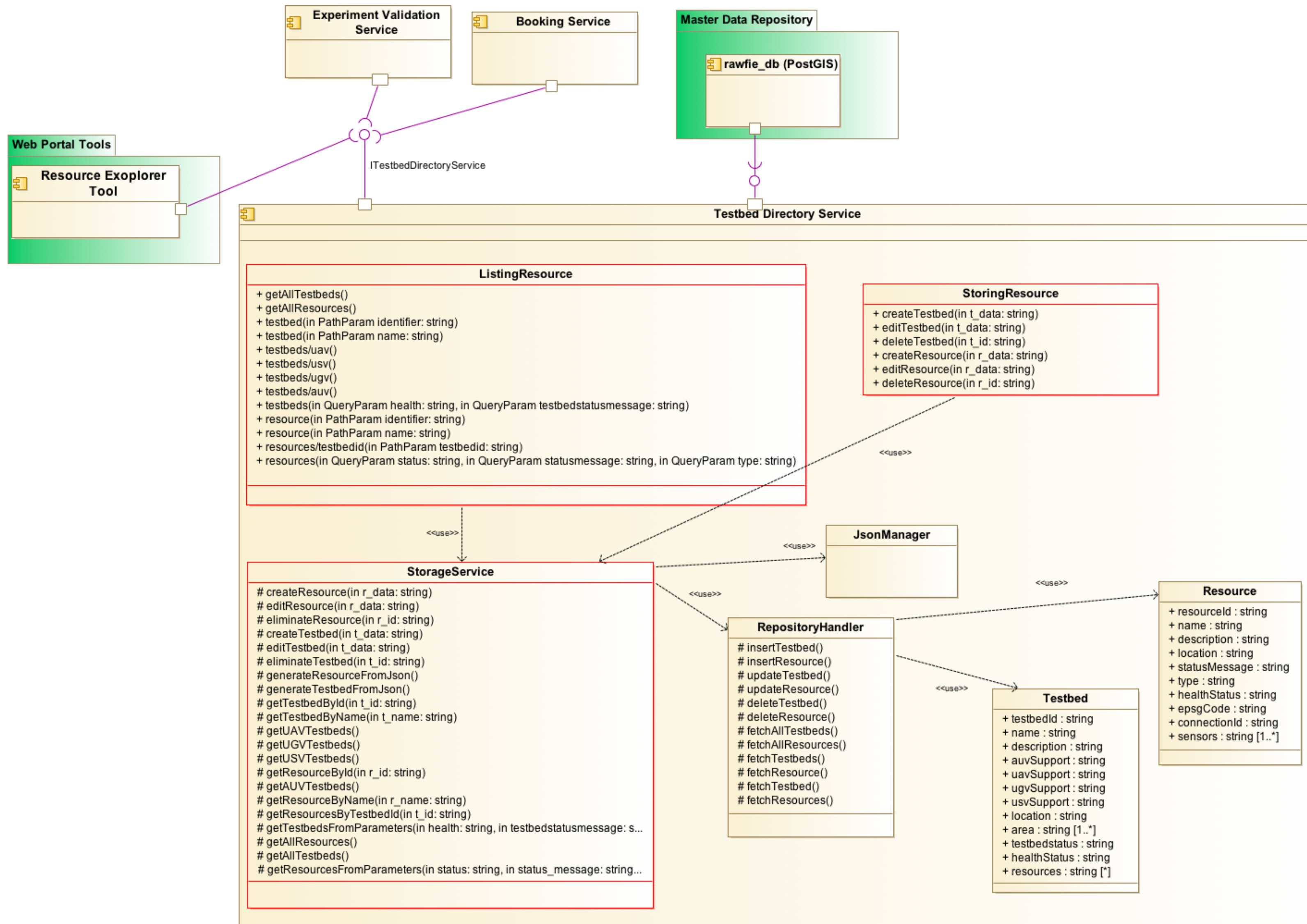

Figure 11: Testbed Directory Service class diagram

The updated list of the REST API exposed by the Testbed Directory Service is provided below. For a detailed description of operations and responsibilities of the Testbed Directory Service component, please refer to deliverable D4.5. In the Annex of the present document (section 9.1.1) a detailed description of the components' interfaces (API) is provided along with information on how to invoke them and the expected results.

Provided interfaces

- *getAllTestbeds*
- *getAllResources*
- *testbed/identifier/{id}*
- *testbed/name/{name}*
- *testbeds/auv*
- *testbeds/uav*
- *testbeds/usv*
- *testbeds/ugv*
- *testbeds?health=Val1&testbedstatusmessage=Val2*
- *resource/identifier/{id}*
- *resource/name/{name}*
- *resources/testbedid/{id}*
- *resources?resource_status=Val1&resource_status_message=Val2&resource_type=Val3&health=Val4*
- *createTestbed*
- *editTestbed*
- *deleteTestbed*
- *createResource*
- *editResource*
- *deleteResource*
- *deleteTestbedParameters/{id}*
- *deleteResourceParameters/{id}*

Required Interfaces

The Testbed Directory Service is in charge of executing the queries to the Master Data Repository, for realising CRUD (Create, Read, Update, Delete) operations. It uses JPA technology for its operations.

4.2.2.3 *Updated sequence diagrams*

Following Figure 12, Figure 13 and Figure 14 show the updated Sequence diagrams for 3 of the most relevant use cases where the Testbed Directory Service is involved: search of available resources based on specific criteria, registration of a new testbed and registration of a new UxV resource. Updates with respect to the previous version of the same diagrams (in D4.5) are again highlighted with red lines.

Search for an available resource

1. The Experimenter issues a search request by specifying the parameters relative to the specific resource information, using the Resource Explorer Tool. In our example, the Experimenter is requesting resources of type USV (encoded with resource code type =3 in the database).
2. The Resource Explorer Tool uses the REST API method implemented by the ListingResource class, passing the appropriate value of the resource type query parameter (search criteria)
3. The REST interface method uses the getResourcesFromParameters method of the StorageService class, which in turn fetches the information from the Master Data Repository, through the RepositoryHandler class, which provides the JPA (Java Persistence API) interface to the database

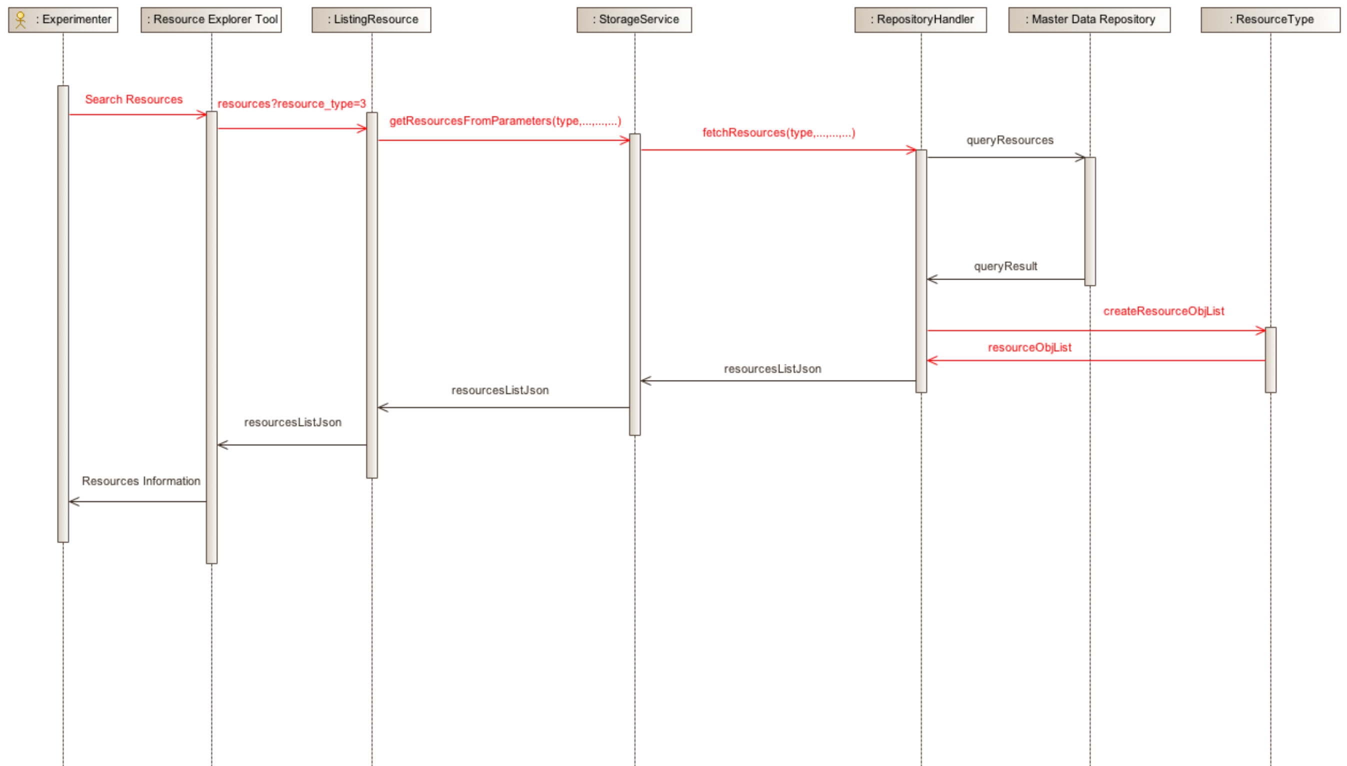

Figure 12: Experimenter search resources of specific type (USV)

Register a new Testbed in the platform

1. The Platform Administrator starts with the process of registering a new Testbed into the RAWFIE federation, after its formal approval and compliance with specific regulations.
2. In this case, the request is issued at local Testbed level through the Testbed Manager (see related component description section). The Testbed Manager calls the *createTestbed* REST interface implemented by the *StoringResource* class, by providing in input the Testbed information structure (JSON)
3. The REST interface method uses the *createTestbed* method of the *StorageService* class, which in turn inserts the information in the Master Data Repository, through the

RepositoryHandler class which provides a JPA (Java Persistence API) interface to the database, and after the same information is converted in the TestbedType object structure

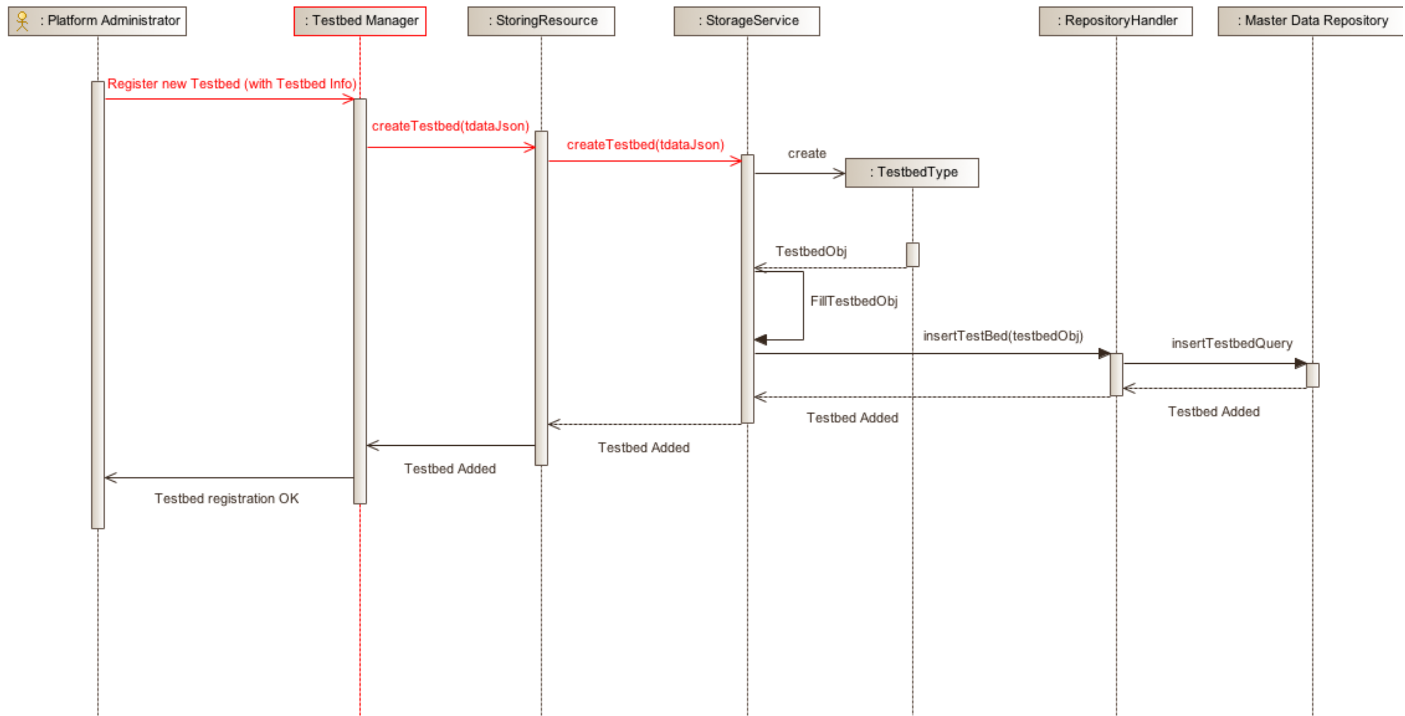

Figure 13: Platform admin registers a new Testbed

Add a new UxV device into a Testbed facility

1. The Testbed Operator starts with the process of registering a new Resource into the given testbed
2. In this case, the request is issued at local Testbed level through the Testbed Manager (see related component description section). The Testbed Manager calls the *createResource* REST interface implemented by the *StoringResource* class, by providing in input the Resource information structure including the Testbed Identifier (JSON structure)
3. The REST interface method uses the *createResource* method of the *StorageService* class, which in turn inserts the information in the Master Data Repository, through the *RepositoryHandler* class which provides a JPA (Java Persistence API) interface to the database, and after the same information is converted in the ResourceType object structure

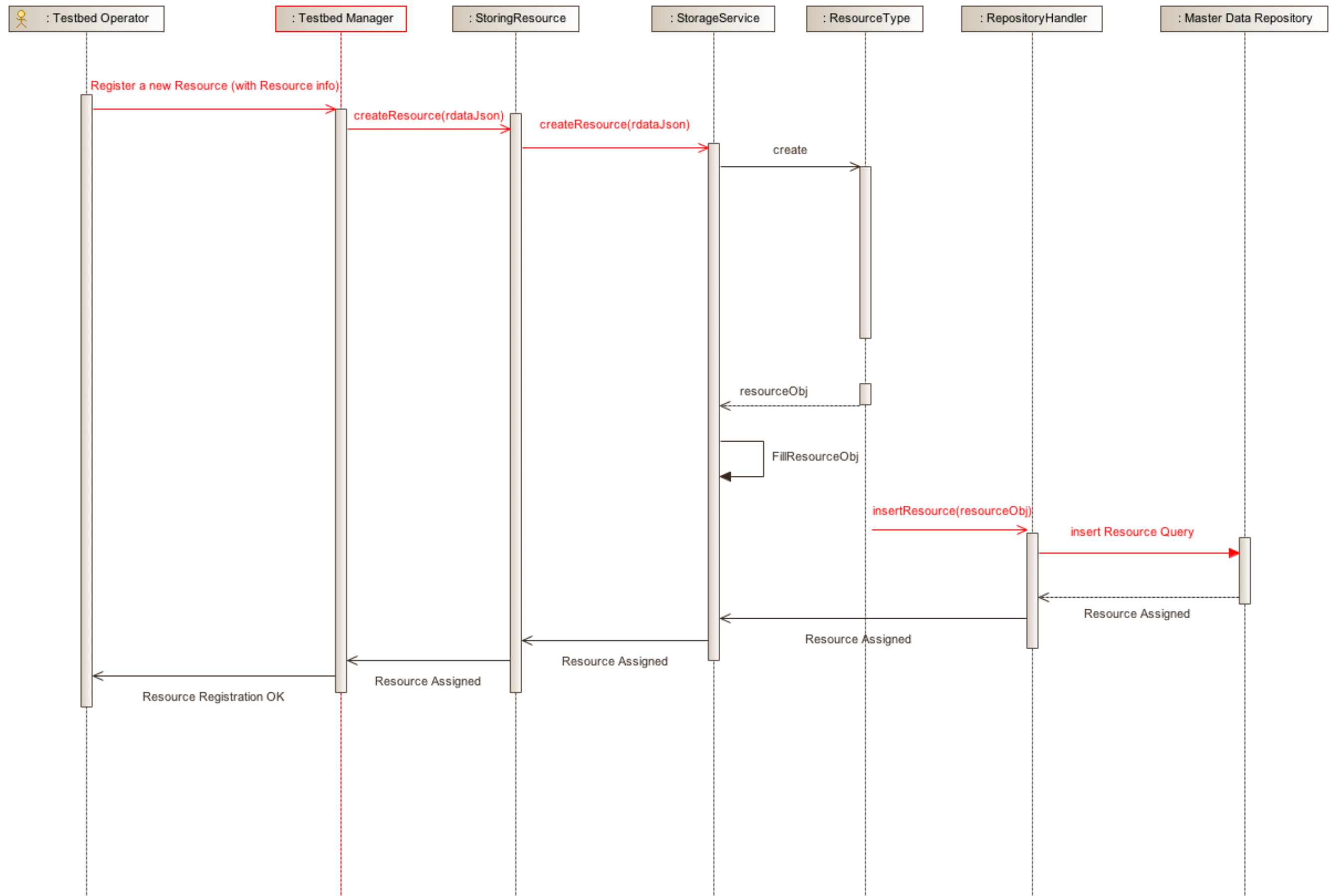

Figure 14: Register a new UxV resource

4.2.3 EDL Compiler and Validator

The EDL Compiler & Validator (ECV) is responsible for performing syntactic and semantic analysis on the provided EDL scripts. The compilation and validation will be performed on top of the proposed EDL model that is based on a specific grammar. The ECV will access the provided script and identify any syntactic and semantic errors that could jeopardize the execution of the experiment. Finally, when no errors are present, the component will have the opportunity to generate the final code to be uploaded in the UxVs.

4.2.3.1 *Component requirements as identified in D3.3*

ID (Priority)	Description	Requirement Mapping with component's functionalities
PT-CPV-001 (High)	A tool for translating EDL into user directives shall be provided	The Compiler & Validator provides this functionality.
PT-CPV-002 (High)	An experimenter should have the opportunity to use a code generation engine	The Compiler & Validator provides this functionality.
PT-CPV-003	Experiments defined via	The Compiler & Validator provides this

(High)	EDL shall be validated after their authoring	functionality.
PT-CPV-004 (High)	The compiler and validator should communicate with the authoring tool in order to transfer error indications and hints for solving them	The Compiler & Validator provides this functionality.

4.2.3.2 *Final specification of functionalities and interfaces*

The main operations are related to scripts compilation and validation and the production of the appropriate files to be adopted by the remaining components of the RAWFIE architecture. A syntactic validator accompanied by a custom validator (to cover any special needs for scripts compilation) undertake the responsibility of identifying errors and warnings. Cross link validation will be responsible to cover complex aspects of the experiments workflow. Finally, a generator will be responsible producing the final code that could be adopted by the remaining architecture.

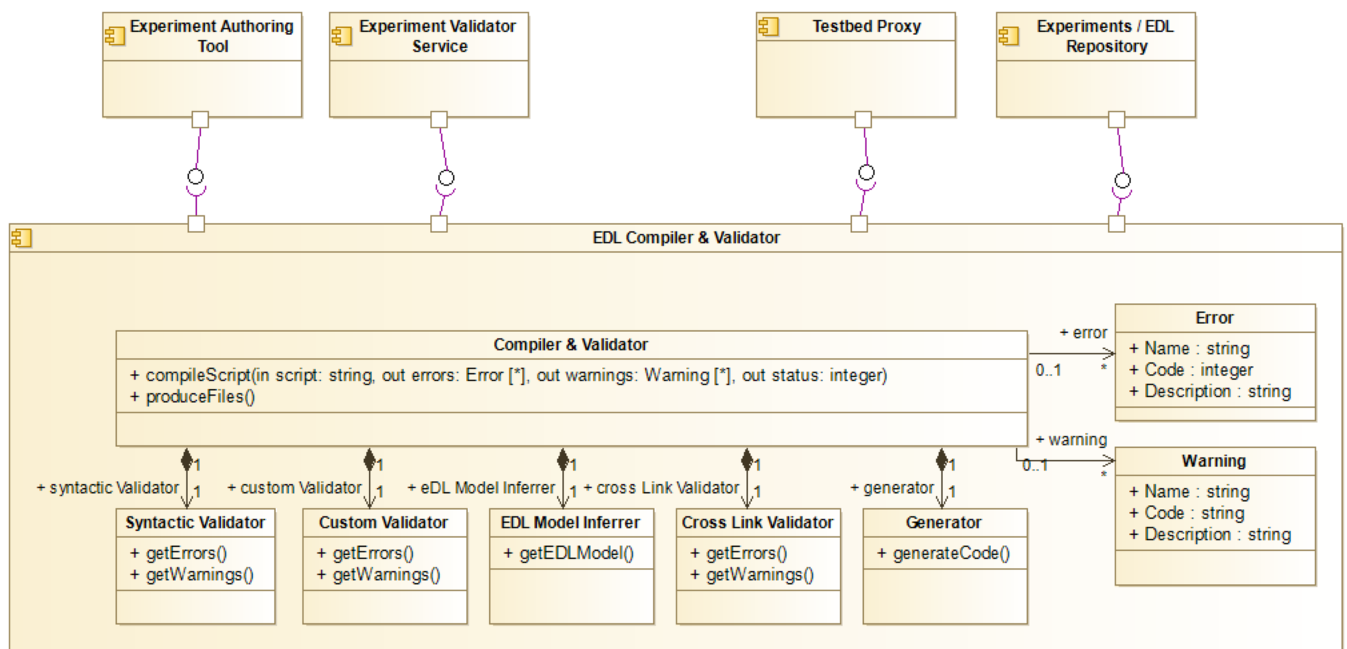

Figure 15: Class diagram for the ECV

4.2.3.3 *Updated sequence diagrams*

There are not any updates in the sequence diagrams of the ECV. Please refer in D4.5 for details.

4.2.4 Experiment Validation Service

The Experiment Validation Service (EVS) is responsible for experiments validations with regard to execution issues. Thus, the EVS will validate if each experiment can be efficiently executed in the selected Testbed.

4.2.4.1 *Component requirements as identified in D3.3*

ID (Priority)	Description	Requirement Mapping with component's functionalities
PT-EXV-S-001 (HIGH)	RAWFIE shall provide a validator to constantly check experiment scenarios during runtime	The Validator provides this functionality.
PT-EXV-S-002 (HIGH)	The validation service should perform syntactic checking	The Validator provides this functionality.
PT-EXV-S-003 (HIGH)	The validation service should perform semantic checking	The Validator provides this functionality.

4.2.4.2 *Final specification of functionalities and interfaces*

The EVS involves a simple interface accessible by other components that are responsible to initiate the validation process. An attribute named 'verbose' indicates if the service will provide extensive information in a data log related to the analytical view of the validation process. In the following picture, we present the class diagram of the discussed service.

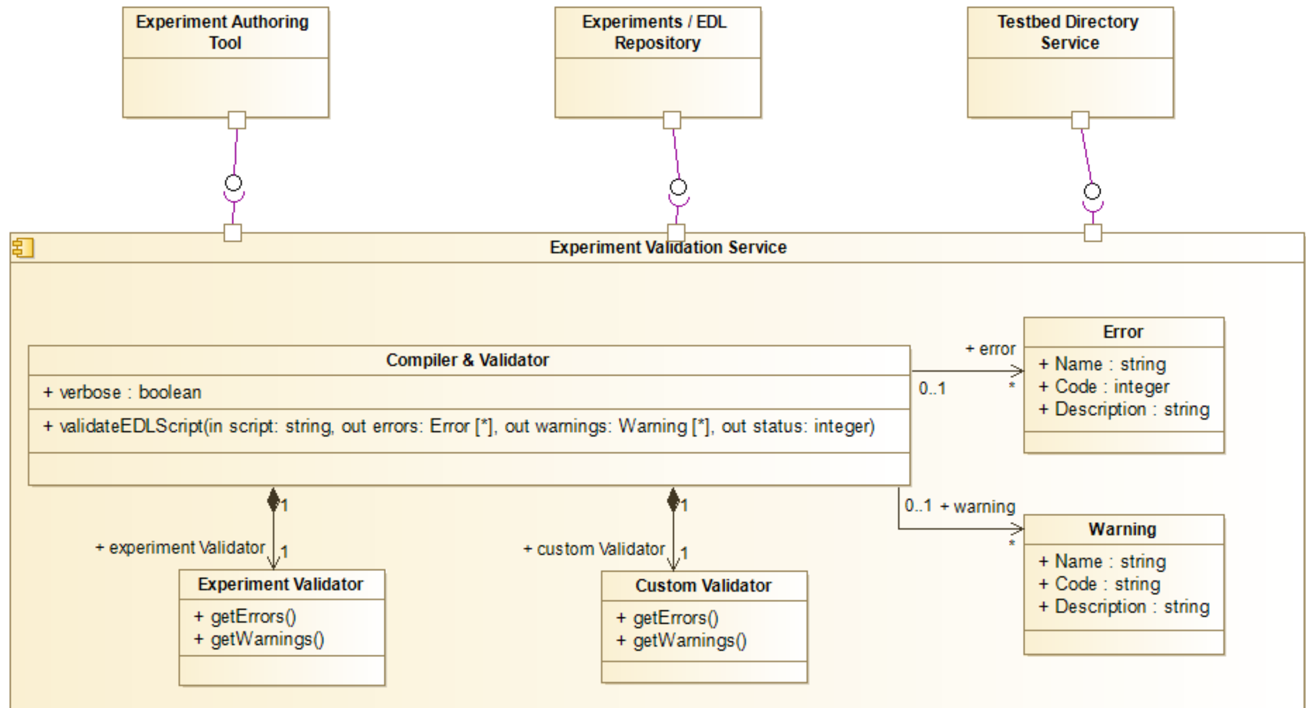

Figure 16: Class diagram for the EVS

4.2.4.3 *Updated sequence diagrams*

There are not any updates in the sequence diagrams of the EVS. Please refer in D4.5 for details.

4.2.5 Users & Rights Service

The Users & Rights Service provides authentication and authorization to all components of the system.

4.2.5.1 *Component requirements as identified in D3.3*

ID (Priority)	Description	Requirement Mapping with component's functionalities
PT-USR-S-001 (HIGH)	User login credentials checking shall be provided	The UserAndRightsServiceProtocol interface provides this function.
PT-USR-S-002 (HIGH)	RAWFIE platform shall support various roles with different privileges at every level of access.	Users & Rights Service can assign several roles to each user. The applications themselves need to know which roles indicate access privileges. They can then check if the user has the required roles.
PT-USR-S-003 (LOW)	The Users & Rights Service may provide a proxy service for web application that do not check access rights.	The ProxyService proxies a HTTP request and checks the roles.

4.2.5.2 *Final specification of functionalities and interfaces*

The Users & Rights Service is based on the Users & Rights Repository that is a LDAP server. The LDAP server stores users/component IDs and their roles (rights). Also groups of users that can have roles are supported. Components may directly access the LDAP server (using a restricted account) or via the Users & Rights Services to get advanced query and editing functions. (Hint: If possible components should use the Users & Rights Services. But if some existing software with LDAP support is used, it will be easier to use LDAP instead of adapting the software).

The authentication between the different RAWFIE components can be done via X.509 client certificates. For authorization the roles need to be checked via the Users & Rights Services or Repository.

The Users & Rights Services interface will provide functions to check credentials (in cases where a user/experimenter does not provide a client certificate, a basic user/password authorisation is possible), to read, add and edit users, to change the password of a user and to check the rights/roles of a user.

An additional ProxyService is provided for applications that do not check access rights. It proxies the HTTP request, looks for the requested URL, determines the needed roles for this URL and checks if the user of the session has the needed roles.

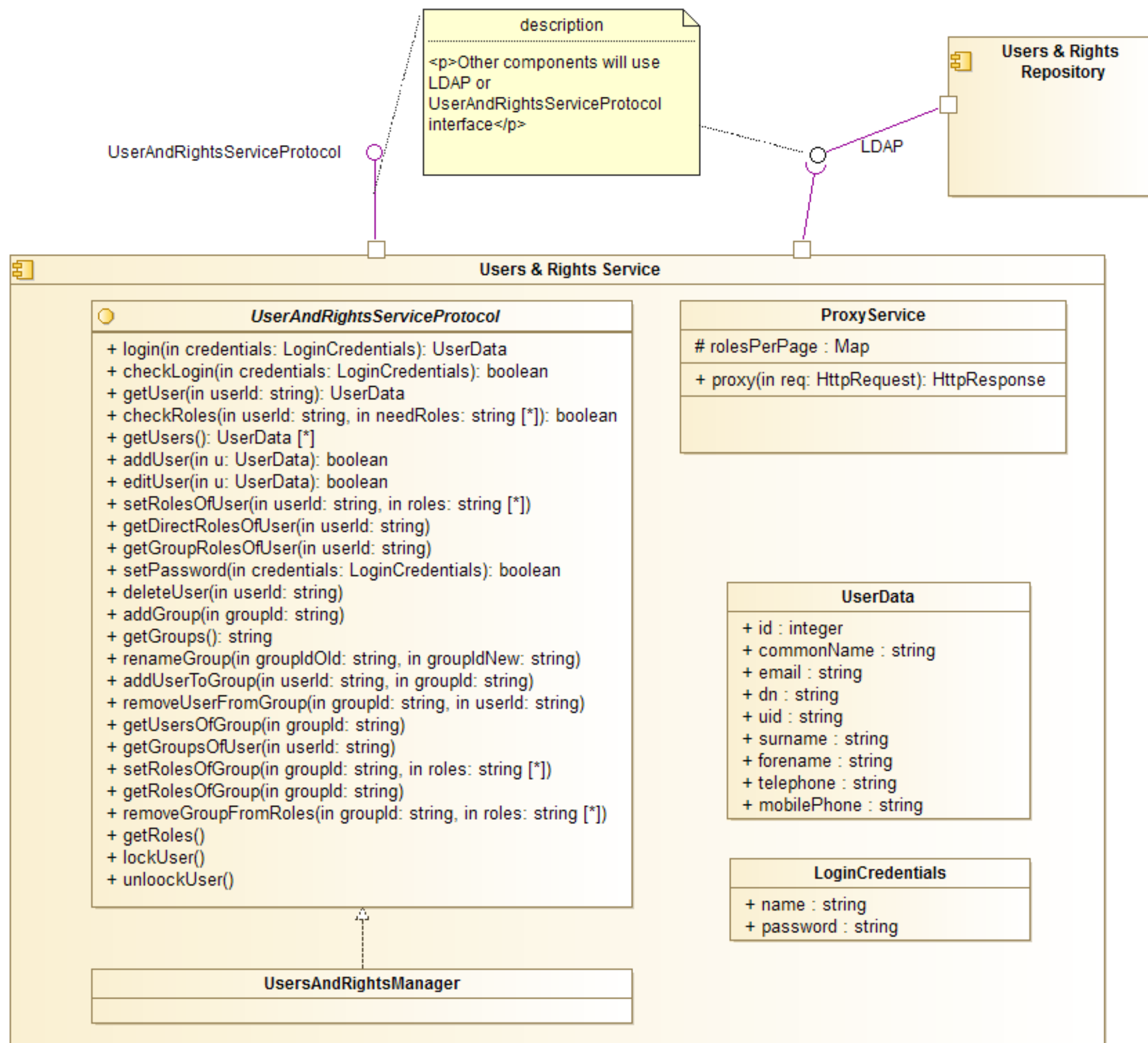

Figure 17: Users & Rights Service - Class diagram

Provided interfaces

- *UserAndRightsServiceProtocol*:
  Any other RAWFIE component (especially from the Front Tier) may access this interface to get user related information.
  UserAndRightsServiceProtocol methods are exposed both via Avro RPC or REST API.

Required interfaces

- Users & Rights Repository:
  will provide a standard *LDAP* interface for access

4.2.5.3 *Updated sequence diagrams*

Password-based user login

1. A user opens via its browser an application of the RAWFIE web page and requests a restricted resource (URL)
2. The application checks if the user is locally logged-in (e.g. via cookie for this application)
3. If not
    a. Redirect to SSO page
    b. The SSO page checks if the user is globally logged in
    c. If not
        i. The user is asked for credentials (username and password)
        ii. The SSO page sends the credentials to the User & Rights Service
        iii. The User & Rights Service checks the credentials and returns whether they are OK
    d. The SSO page redirects to the original web page (with some login token as parameter)
    e. The user requests the original web page again (with some login token as parameter)
    f. The application checks the login token and creates a user session (e.g. transmitted via a cookie)
4. Proceed with "Check user authorisation"

user (browser) : Experimenter
Some App : Web Portal
SSO : Web Portal
: Users & Rights Service
Request a restricted resource
Check if user is logged in
alt
[User not logged in (in App)]
Redirect to Login-Page
Request login page
Check if user is logged in
alt
[User not logged in globally]
Request credentials
Credentials
checkLogin(credentials:LoginCredentials)
Check credentials
true
Redirect to App (with login tocken as parameter)
Request a restricted resource (with login tocken as parameter)
Check login token and creste user seesion
ref
Check user authorisation

Figure 18: Users & Rights Service – Password-based user login

X.509 Certificate-based user login
No updates (refer to D4.5 for details)

Check user authorisation

1. After the user has logged in and has requested a restricted resource, the web application checks if user is allowed to see the resource
2. Component requests the Users & Rights Service if the given user has the specific role/right to see/edit this resource
3. The Users & Rights Service does:
    a. Check if the user exits
    b. Get groups of the user
    c. Check if the role members contain the user or one of the groups of the user
4. If ok: grant access to the user
5. If wrong: show access denied to the user.

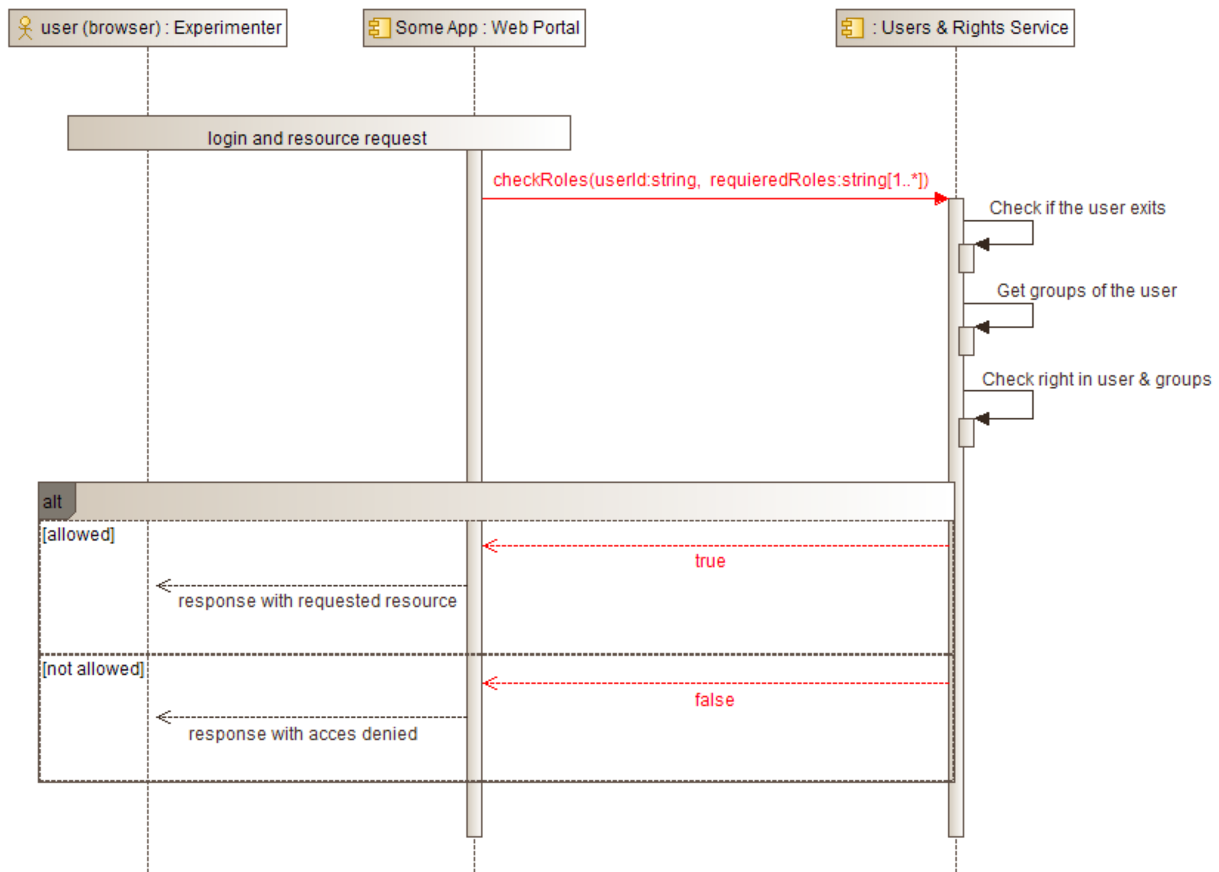

Figure 19: Users & Rights Service – Check user authorisation

Trusted and secure communication between the components
The components in RAWFIE should also use X.509 certificates to establish a trusted and secured communication between them.

1. component A calls service of another component B

2. Transport layer: SSL handshake with client and server certificates (on error close connection)
3. If there is a need to verify the authorisation
    a. checks the certificate of the component A and reads the component name out of the certificate
    b. Component B calls Users & Rights Service to check if component A or the user that has initiated the whole process has the needed roles/rights
    c. Transport layer: SSL handshake with client and server certificates (on error close connection)
    d. The Users & Rights Service
        i. checks the certificate of the component B and reads the component name out of the certificate and
        ii. checks if component B is allowed to read permissions
        iii. checks if component A or the user has the needed roles/rights
        iv. Returns the result (allowed/not allowed)
4. If allowed
    a. component B executes the service method
    b. returns the result to component A
5. If not allowed
    a. component B returns “access denied” to component A

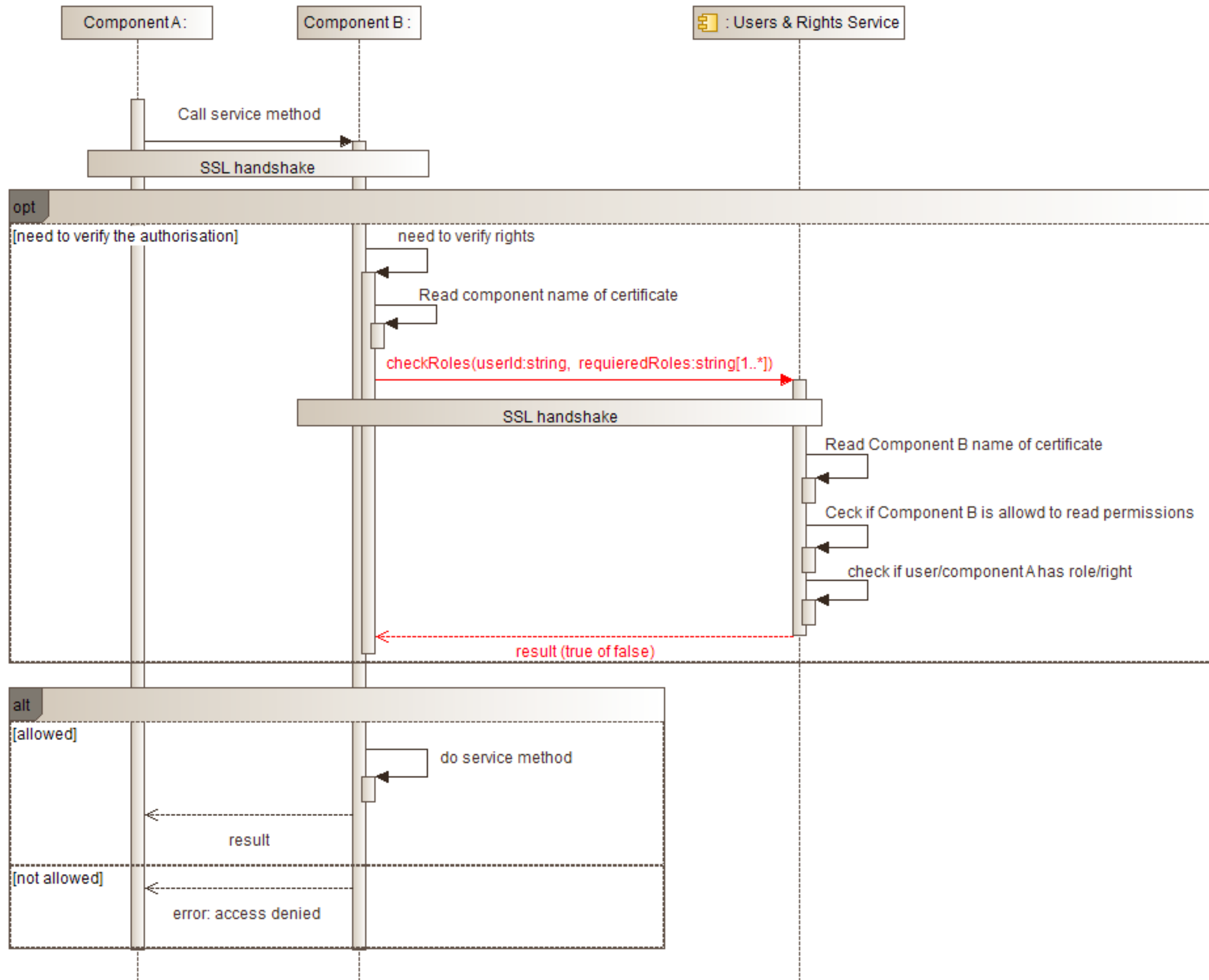

Figure 20: Users & Rights Service – Check user authorisation

4.2.6 Booking Service

The Booking Service is responsible for processing and validating all reservations requests at user or/and experiment level initiated within the RAWFIE platform. It is also responsible for handling changes of status of Booking requests and informing the interesting parties via appropriate notifications. In the 3rd iteration Booking Service will be augmented to achieve synchronization with the SFA Aggregated Manager Reservation process.

4.2.6.1 *Component requirements as identified in D3.3*

ID (Priority)	Description	Requirement Mapping with component's functionalities
PT-BOO-S-001 (HIGH)	Booking Service shall support reservations of resources at both user level and experiment level	*BookingManager* provides methods for processing both experimenter level and experiment reservations (see class diagram)
PT-BOO-S-	User level booking shall	Booking Tool can call the *IBookingService*

002 (HIGH)	be triggered by the Booking Tool via a REST API.	interface methods both via RPC or REST (see *Interactions and relationships with other components*)
PT-BOO-S-003 (HIGH)	Experiment level booking shall be triggered by the experimenter before issuing a manual or schedule launching of a validated experiment	*processExperimentLevelRequest(...)* method is provided by the *BookingManager* which should be called by other services or tools (i.e. *Experiment Authoring Tool*) prior to calling the *Launching Service*
PT-BOO-S-004 (HIGH)	Experiment level booking shall support both immediate booking as well as booking at a future time	Addressed by design and the way booking process is implemented. User booking is performed at specific timeslots in the future. Experiment booking has as prerequisite an existing user booking and refines it.
PT-BOO-S-005 (HIGH)	Booking Service shall provide all the necessary methods to manage the bookings including addition, modification and cancellation/deletion operations	*IBookingService* interface provides methods for all requested actions
PT-BOO-S-006 (HIGH)	Booking Service shall be able to compute and return feedback on conflicting bookings for a provided booking request	*checkForConflictingBooking(...)* method provides this functionality
PT-BOO-S-007 (HIGH)	Reservation Data should be persistent in order to survive service failures and be available by other services	*BookingManager* module interacts with the master data repository via JDBC/JPA in order to update/insert booking info
PT-BOO-S-008 (MEDIUM)	Historical data retrieval for Bookings/Reservations should be available on demand	All data related to reservations are stored in the master data repository and can be queried
PT-BOO-S-010 (HIGH)	Booking functionality shall be able to correctly handle simultaneous Reservations requests by end users	*IBookingService methods* will be exposed as REST and RPC services in a servlet container ensuring multithreaded support
PT-BOO-S-011 (MEDIUM)	Notification mechanisms may be provided for experiments scheduled for execution in the future.	NOT APPLICABLE Refers to execution of experiments and not to the booking process (see also launching service)
PT-BOO-S-012	Booking functionality should provide means to	B*ookingRequestChecker* will apply a set of checks on the proposed reservation ensuring

(HIGH)	ensure fairness in resource booking as well as protect for malevolent actions that a user may perform.	fairness and protection form spurious actions (see *Operations and attributes* section) Moreover, booking requests are generally put in a pending status waiting for approval by a testbed operator which introduces an extra level of protection from malevolent actions
PT-BOO-S-013 (HIGH)	All Booking Service incoming requests should contain user initiating information and delegate/contact the User & Rights service in order to perform validation	

4.2.6.2 *Final specification of functionalities and interfaces*

Provided Interfaces

- Booking Service implements the *IBookingService* interface (see class diagram) that exposes the following methods:
    - *addBooking*
    - *editBooking*
    - *checkForConflictingBookings*
    - *deleteBooking*
    - *rejectBooking*
    - *approveBooking*
    - *getBooking*
    - *getBookings*

IBookingService methods are exposed both via RPC or REST API. The interface is mainly used by the Booking Tool that provides a Web UI to manage the bookings (edit/add/approval etc.):

Required Interfaces

- Booking Service interacts with the Master Data Repository via JDBC/JPA, in order to retrieve/insert/update booking information for a registered experimenter/user of the platform.
- Booking Service acts also as a producer of booking status update messages (*BookingStatusMsg*) that are sent to the message bus and may be consumed by other interested services/modules.
- Booking Service interacts with the SFA Aggregate Manager Rest API to enable update/synchronization with the SFA reservation structures (maintained internally by the Aggregate Manager Triple Store DB.

In the figures that follow, red colour is used to highlight differences compared to the previous version of the architecture

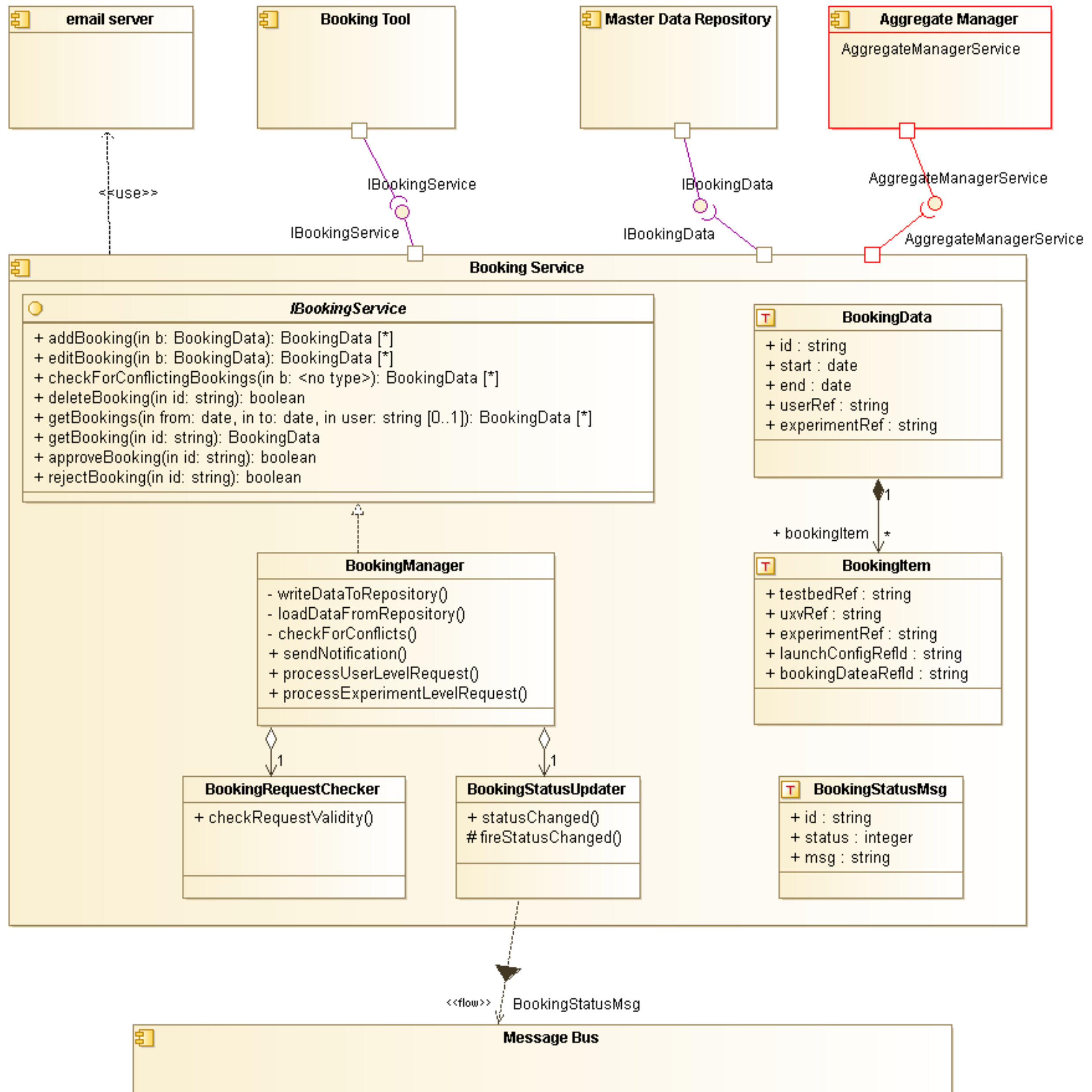

Figure 21: Booking Service - Class diagram

4.2.6.3 *Updated sequence diagrams*

Only add/edit a booking procedure is being affected by the integration/synchronization with SFA Aggregate manager reservation process. The rest sequence diagrams remain the same as described in the previous version of the deliverable (D4.5).

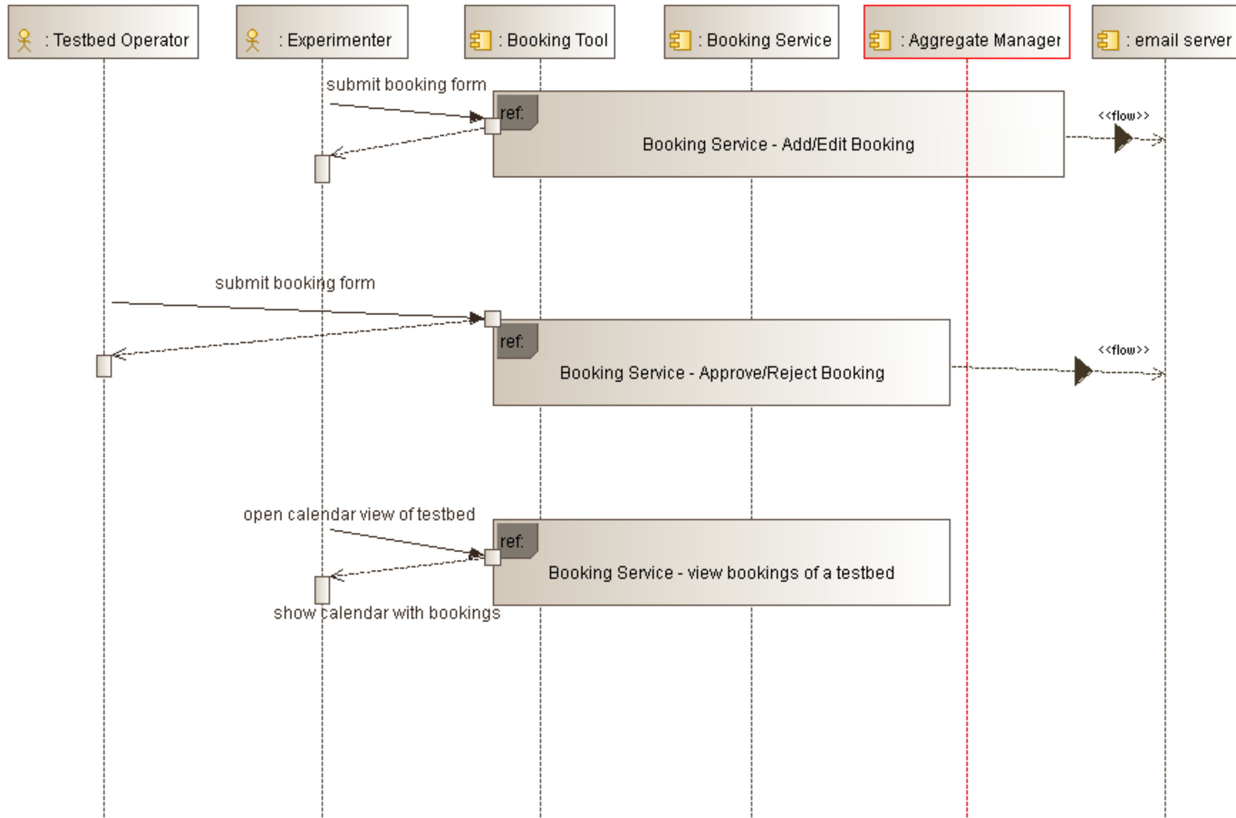

Figure 22: Booking Service - Overview

View bookings of a testbed
No updates (refer to D4.5 for details).

Add/edit a booking (experimenter)

1. Experimenter submits the form with the booking details to the *Booking Tool*
2. *Booking Tool* calls the *addBooking(...)* or *editBooking(...)* method of the *BookingManager*
3. The *BookingManager* processes the booking request:
    - reads all bookings of the given resources in the given timespan from the *Master Data Repository*
    - contacts the *BookingRequestChecker* that checks the validity of the request and whether any conflicts with existing booking are introduced
4. If conflicts are identified, they are returned to the *Booking Tool*, which shows them to the user
5. If there are no conflicts, then *BookingManager*:
    a. Calls allocate on the SFA Aggregate Manager in order to add reservation the SFA Triple Store. If a conflict is detected then we return to step 4 above otherwise the process continues
    b. Writes or updates the data repository appropriately with the new or updated booking data. The booking status is set to *PENDING* (to be approved by a testbed operator),

c. creates and sends an email message both to the experimenter (initiating the request) as well as to the registered testbed operator responsible for approving the booking later on
d. return a success message to the *Booking Tool*, which shows it to the user

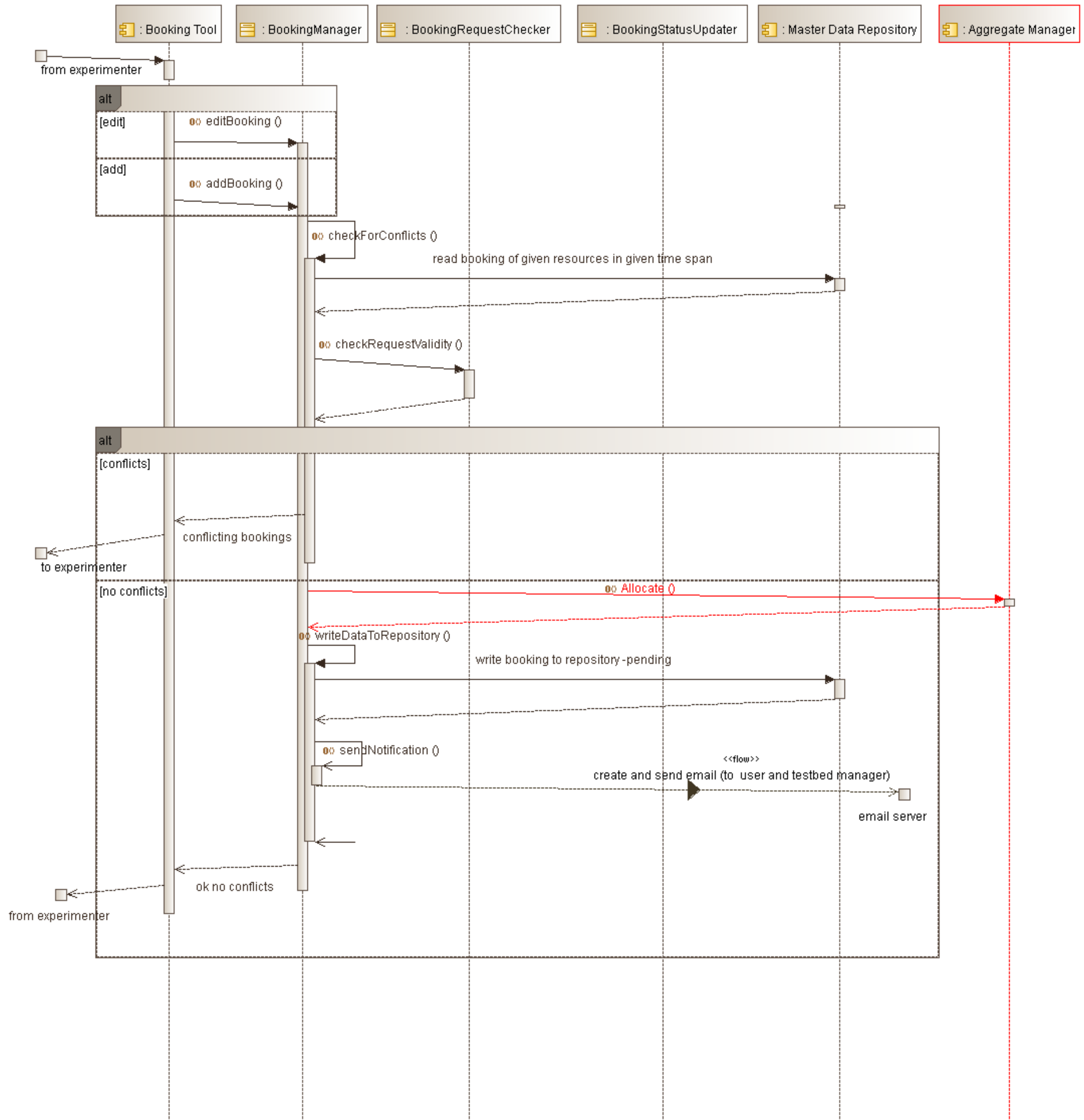

Figure 23: Booking Service – Add/Edit a booking

Approve/Reject a pending booking (testbed operator)

No updates (refer to D4.5 for details).

4.2.7 Launching Service

The Launching Service (LS) is responsible for handling requests for starting or cancellation of experiments. It supports short term and long term launching. LS will execute only authorized and approved experiments based on spatio-temporal constraints validated just prior to the actual launching.

4.2.7.1 *Component requirements as identified in D3.3*

ID (Priority)	Description	Requirement Mapping with component's functionalities
PT-LAU-S-001 (HIGH)	Launching Service shall support short-term or manual launching of an experiment initiated directly by an experimenter	*ILaunchingServiceProtocol* provides a *ManualStart(...)* method
PT-LAU-S-002 (HIGH)	Launching Service shall support long-term or scheduled launching of an experiment initiated directly by an experimenter	*ILaunchingServiceProtocol* provides a *schedule(...)* method. The LS contains an internal *ExperimentScheduler* singleton module that can be used for adding or removing an experiment for/from future launching
PT-LAU-S-003 (HIGH)	Each executing experiment shall be uniquely identified within RAWFIE ecosystem	*LaunchingServiceProtocolImpl* class provides a *generateExecutionId(...)* method that should create a unique Id to be associated with launched experiment
PT-LAU-S-004 (HIGH)	During launching it must be ensured that the experiment to be started has been validated based on spatio-temporal constraints	*LaunchingServiceProtocolImpl* class provides a *preValidate(...)* method that applies a sequence of checks prior to actual launch
PT-LAU-S-005 (HIGH)	During launching it must be ensured that the experiment to be started belongs to an authorized user of the RAWFIE platform	*LaunchingServiceProtocolImpl* class provides a *preValidate(...)*method that applies a sequence of checks prior to actual launch
PT-LAU-S-006 (HIGH)	The Launching Service shall be able to address simultaneous requests for starting an experiment	*ILaunchingServiceProtocol methods* will be exposed as REST and RPC services in a servlet container ensuring multithreaded support
PT-LAU-S-007 (HIGH)	The Launching Service shall send an appropriate message upon successful	*ExperimentStartRequest* JSON message is sent by createAndSendMessage() upon successful processing of manual or scheduled launching

	starting of an experiment	(see sequence diagrams below)
PT-LAU-S-008 (HIGH)	The Launching Service shall interact with other components or database services in order to retrieve information needed for deciding on launching an experiment	*LaunchingServiceProtocolImpl* class provides a *updateLaunchConfig(...)* method and the service interacts with the Master Data Repository for retrieving Booking and Experiment Related information
PT-LAU-S-009 (HIGH)	Interactions of the launching service with database services and/or other components should respect the RAWFIE platform boundary	By design, Launching Service interacts only with middle tier components, the message bus and the master repository. No direct communication with the testbed tier exists.
PT-LAU-S-010 (HIGH)	Launching service shall support requests for experiment cancellation	*ILaunchingServiceProtocol* provides a *cancel(...)* method and may sent an *ExperimentCancelRequest* to the ExperimentController
PT-LAU-S-012 (HIGH)	Launching service shall provide appropriate feedback to the requested entity regarding failures on fulfilling a request	All *ILaunchingServiceProtocol* methods may return a *LaunchingActionResp* structure which includes a boolean *status* field indicating success or failure and a *msg* string field that may provide details the problem
PT-LAU-S-013 (HIGH)	Launching service shall not alter or modify any information related to the actual execution of an experiment	By design, Launching Service generates and forwards only appropriate messages for initiating or cancelling an experiment. Database write operations are related only to updating/relating the *executionId* with an experiment and they do not "touch" application specific data
PT-LAU-S-014 (MEDIUM)	Notification mechanisms may be provided for experiments scheduled for execution in the future.	*ExperimentScheduler* component provides a *sendNotification(...)* event that can be triggered at a configurable interval prior to actual launching

4.2.7.2 *Final specification of functionalities and interfaces*

Provided Interfaces

- Launching Service implements the *ILaunchingService* interface (see class diagram) which provides the following methods:
    - *manualStart* (issues a launch request immediately for a given experiment)
    - *schedule* (schedules an experiment for launching at a future time)
    - *cancel* (issues a request for cancelling a running or scheduled experiment)

IBookingService methods are exposed both via RPC or REST API. The main components for calling the provided API are *Experiment Authoring Tool* & *Experiment Monitoring Tool*.

Required Interfaces

- Launching Service interacts with the Master Data Repository via JDBC/JPA (*IBookingData* interface), in order to retrieve booking related information and proceed with certain validation actions required prior to experiment launching.
- Launching Service interacts with the Master Data Repository via JDBC/JPA (*IExperimentData* interface), in order to retrieve/update/create experiment execution related information
- Launching Service produces *ExperimentLaunchRequest* and *ExperimentCancelRequest* messages that are send to Messages Bus and consumed by any other middle tier component. The main component interested for those types of messages is the *ExperimentController* module.

The figure below depicts a class diagram showing the interfaces and internal structure of the *Launching Service* (red colour is used to highlight differences compared to the previous version of the architecture.

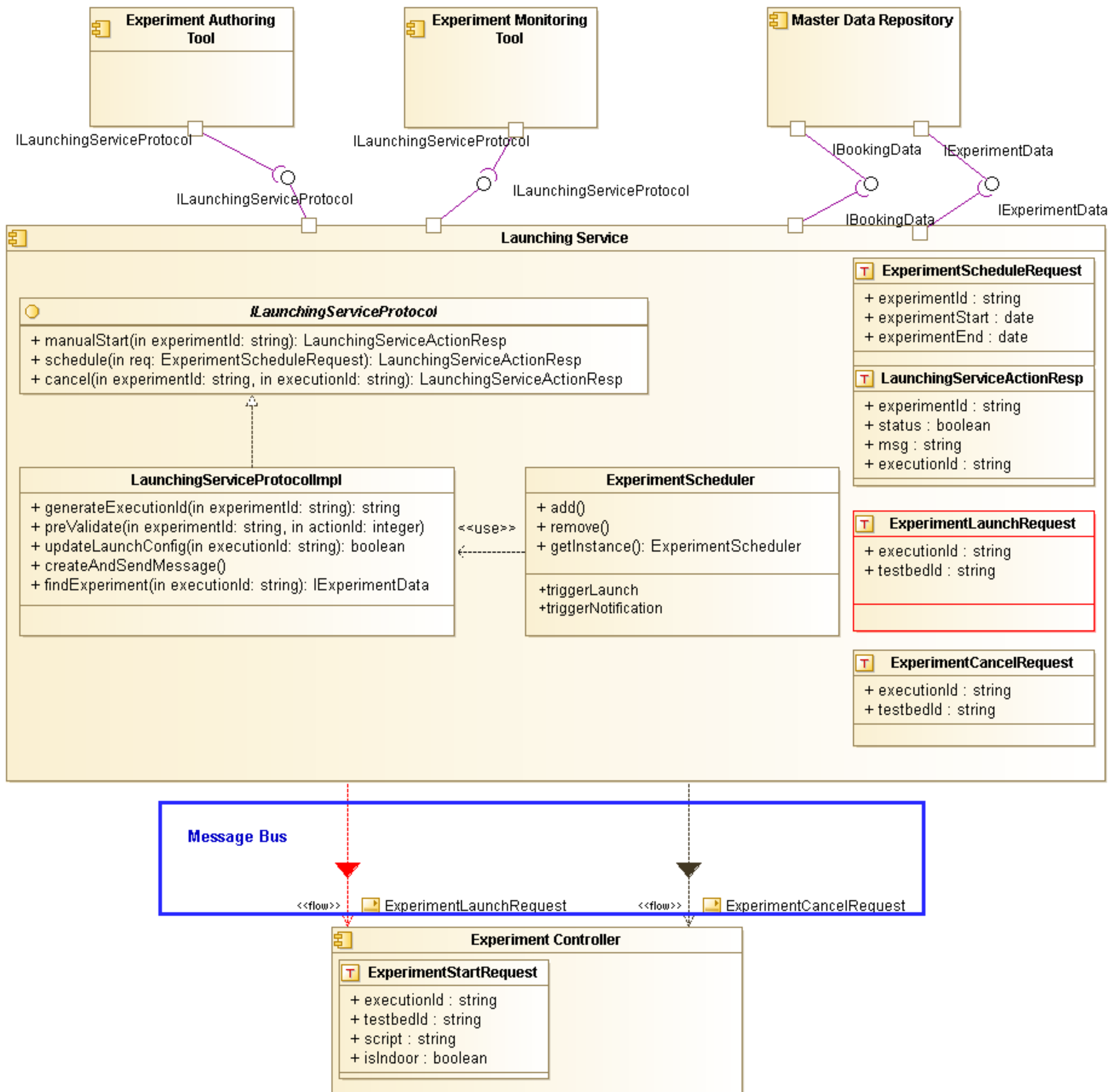

Figure 24: Class diagram of the Launching Service

4.2.7.3 *Updated sequence diagrams*

All sequence diagrams remain the same as in the previous version of the deliverable (D4.5) with the exception that message send to message bus is of type *ExperimentLaunchRequest* instead of *ExperimentStartRequest*.

Manual Launch

No updates (refer to D4.5 for details)

Scheduled Launch

No updates (refer to D4.5 for details)

Cancellation
No updates (refer to D4.5 for details).

4.2.8 Visualisation Engine

The Visualisation Engine provides the necessary back end services for geospatial data visualisation related to running experiments. It provides the required maps for area visualisation, can cache data for faster load times and finally provides a spatial database for converting and storing UxV information into geo information layers.

4.2.8.1 *Component requirements as identified in D3.3*

ID (Priority)	Description	Requirement Mapping with component's functionalities
PT-VIS-E-001 (HIGH)	The Visualisation Engine shall retrieve from the message bus all runtime experiment information needed for visualising the UxVs and/or any sensor measurement	With StartExperiment() the user automatically subscribes to topics that provide information about the experiment
PT-VIS-E-002 (LOW)	The Visualisation Engine shall provide a GIS server capable of handling geographical layers (overlays)	GeoServer will be deployed, configured and used for this task
PT-VIS-E-003 (LOW)	The Visualisation Engine may allow cache of data for faster access to the available geographic layers	GeoWebCache will provide this option given we provide our own maps of the premises
PT-VIS-E-004 (MEDIUM)	The Visualisation Engine shall provide the possibility to reply experiments using historical data	The data from the message bus will be mapped in the database and the VE will replay it upon request from the VT

4.2.8.2 *Final specification of functionalities and interfaces*

The VE takes care of different tasks:

- Manage what is going to be presented to the experimenter, which UxVs are going to be plotted, over which terrain, when etc. This information will come from the Message Bus and will be sent to the VT.
- Indicate when the experiment is started/stopped and if there are issues with the running experiments, a decision will be made how to present them to the experimenter. This

information will come from the Experiment Controller and will be sent to the VT over the websocket

- Which maps are about to be retrieved and from where. Maps may be retrieved from different providers like Google Maps, Bing Maps, OpenStreetMaps etc. and will be sent to the VT over the GIS channel
- Based on preferences, defined by the experimenter or set for an experiment, layers will be prepared on top of the main map to indicate different conditions, scenario and other important geographic information. This information will come from the VT over the websocket

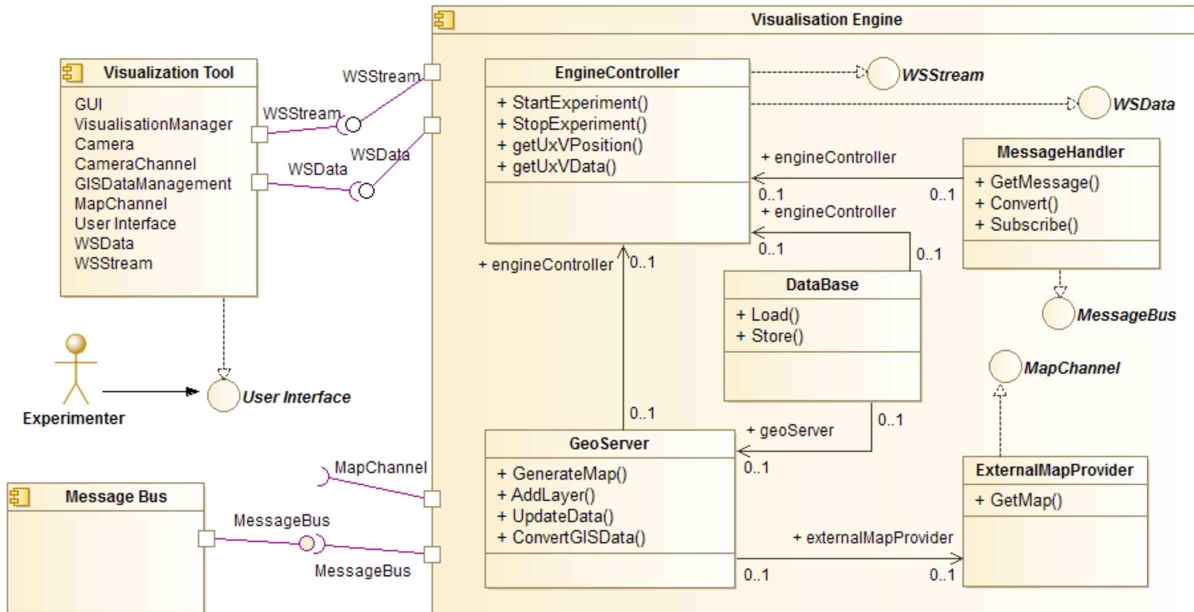

Figure 25: Visualisation Engine - Class diagram

Required interfaces:

- The VE has interface to the Message Bus for receiving updates on the UxV in the Publish/Subscribe manner.
- The VE interface to the Experiment Controller will provide information about the execution of the experiment. It will monitor for start and stop of the real experiment, as well as for cases when the connection to the UxV is lost and others.
- The external map interface is used in the VE for retrieving maps from external provider. In case the experimenter needs detailed and publically available maps, they can be received from such services over this interface.

Provided interfaces:

- The GIS interface is used to send geographical information in various formats like WMS, WFS, WPS and WCS from the VE to the VT. The VT requests map information over the websocket interface and the geo information data is sent over the GIS interface.

- The websocket interface is used in both directions to retrieve information, like sensor data from VE to VT or to inform the VE that the experimenter changed a layer in the VT and it needs to be reloaded from the VE.

4.2.8.3 *Updated sequence diagrams*

No updates (refer to D4.5 for details).

4.2.9 Data Analysis Engine

4.2.9.1 *Component requirements as identified in D3.3*

ID (Prioroty)	Description	Requirement Mapping with components functionalities
PT-DIR-S-001 (MEDIUM)	Analysis engine will support accepting analysis jobs	In order for a job to be accepted, the definition of the required user-specified parameters and models has to be performed through the Data Analysis Tool before the encapsulation.
PT-DIR-S-002 (MEDIUM)	Analysis engine will support compiling analysis jobs	Before compilation, an analysis job has to be accepted by the engine. Previously specified requirements therefore apply as well.

4.2.9.2 *Final specification of functionalities and interfaces*

No changes compared to the design reported in D4.5. Please refer to that deliverable for final design information of the Data Analysis Engine.

4.2.9.3 *Updated sequence diagrams*

Updated sequence diagrams where the Data Analysis Engine component is involved, are reported in Section 5.7 later in this document.

4.2.10 System Monitoring Service

The System Monitoring Service will check if all system components and services are running. This also includes data (if available) about the status of the Testbeds and UxVs from the Monitoring Manager of each Testbed.

4.2.10.1 *Component requirements as identified in D3.3*

ID (Priority)	Description	Requirement Mapping with component's functionalities
PT-SYM-S-001 (HIGH)	RAWFIE middle tier shall include a module to monitor the performance of the middle tier components.	The third party application Icinga will monitor the servers and the services on them
PT-SYM-S-002 (HIGH)	RAWFIE Testbeds and UxVs statuses should be monitored	The Monitoring Manager of the Testbed should send health status data on the message bus. This data will then be evaluated by the System

		Monitoring Service.
PT-SYM-S-003 (HIGH)	RAWFIE system administrators should be informed if critical, for the RAWFIE platform operation, services are down	Icinga can be configured to send emails on case of an error. SMS may also be sent, by sending an email to an SMS provider.
PT-SYM-S-004 (LOW)	User may register for notifications if certain components are down	Icinga can also be configured for this.
PT-SYM-S-005 (MEDIUM)	Notifications about planned downtimes	Icinga can also be configured for this.

4.2.10.2 *Final specification of functionalities and interfaces*

The System Monitoring Service is realized by configuring and extending the existing monitoring solution Icinga [2] (a fork of Nagios [4]). Nagios is open source and a de-facto standard software for system monitoring.

The system monitoring software Icinga has built-in functionalities to check health status of standard system, supporting actions like e.g. is server alive, does database access connections, and is memory usage too high. NRPE (Nagios Remote Plugin Executor) extension of Icinga and the JNRPE (Java NRPE) Server is used to write own plugins that collect special status information from the RAWFIE components. The plugin for RAWFIE transfers the asynchronously collected data by the System Monitirong Servcie (via Message Bus) to the Icinga server.

Icinga can also be configured to send notification (e.g. an email) to a predefined group of receivers, if servers or services are not responding or if the defined thresholds of performance indicators are exceeded.

To get health status information from Icinga for further processing in the System Monitoring Service, the MK-Livestatus API [6] extension is used.

The System Monitoring Service provides the interface “SystemMonitoringServiceProtocol”. It is used by the System Monitoring Tool and the Experiment Monitoring Tool to display the data on a web page.

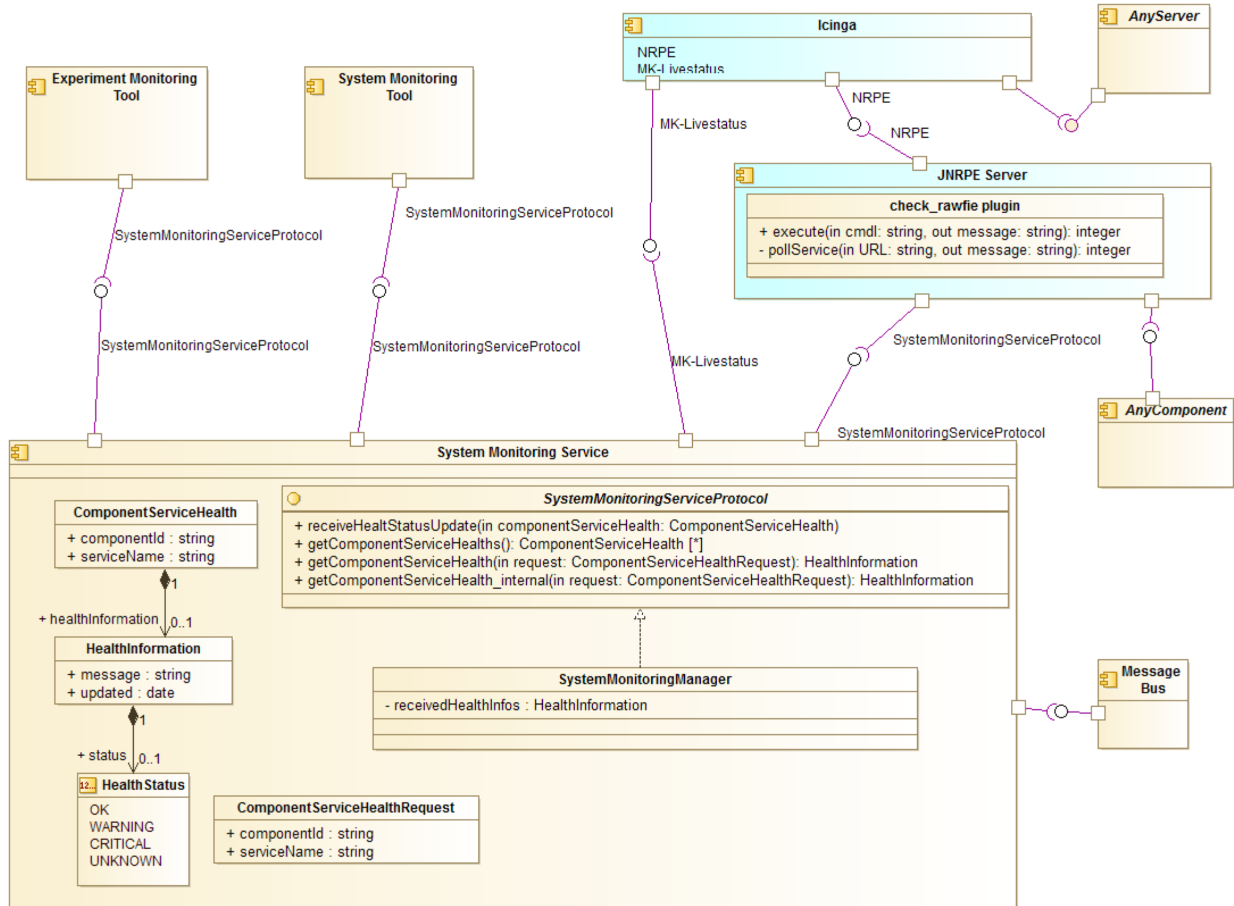

Figure 26: System Monitoring Service - Class diagram

Provided interfaces

- System Monitoring Service (SystemMonitoringServiceProtocol):
  The System Monitoring Tool display the collected data in a web page UI. Also the Experiment Monitoring Tool will show some status information about the resources belonging to an experiment.

Required Interfaces

- Health status interfaces / API exposed by other components
  All important components of the RAWFIE system are monitored via standard procedures, via special plugins/status interfaces or they send their status autonomously to the message bus.

4.2.10.3 *Updated sequence diagrams*

No updates (refer to D4.5 for details).

4.2.11 Accounting Service

Keeps track of resources usage by individual users to charge them later. The appropriate final solution for the accounting functionalities of RAWFIE is under investigation and it is strictly related to the overall business model and the specific accounting approach adopted on each

different testbed. Among the others, the Open Source solution *KillBill*[7] is being considered and analyzed.

4.2.11.1 *Component requirements as identified in D3.3*

ID (Priority)	**Description**	**Requirement Mapping with component's functionalities**
PT-ACC-S-001 (MEDIUM)	The accounting service should be capable to accept different cost models regarding RAWFIE usage on a per service basis	*KillBill* provides different "plans" for "products" where the prices could be defined on subscription base and usage base ("consumable")
PT-ACC-S-002 (MEDIUM)	The accounting service should be capable to gather statistics regarding usage of the platform by experimenters.	Data about the used resources are queried from the Master Data Repository. Notifications from the message bus about finishing, cancelling or aborting an experiment are taken into account.
PT-ACC-S-003 (MEDIUM)	The RAWFIE platform should record information related to time and type of access for a service by a user.	Such events are gathered (see PT-ACC-S-002) and sent to KillBill which records them to create invoices later
PT-ACC-S-004 (MEDIUM)	The cost model used may take into consideration the overall time of experiments executed by a user of the platform.	The "plans" of KillBill allow an adaption of the price when a specific amount of used units is exceeded.
PT-ACC-S-005 (MEDIUM)	The accounting service may support different types of charging based on the type of the experimenter (industrial, research, university etc.)	KillBill "plans" can be grouped to price lists. So price lists with discount plans can be offered to special groups of customers.
PT-ACC-S-006 (MEDIUM)	The accounting service may support predefined types of memberships regarding usage of the platform that may depend on various types of parameters	KillBill supports to have the same product in different "plans" the user may choose. So e.g. there could be a plan with a standing charge and low resource usage charges and a plan with no standing charge but high resource usage charges
PT-ACC-S-007 (MEDIUM)	The accounting service should be able to handle the addition of new services that may be incorporated in the	KillBill allows the updating of the catalogue (contains products, plans and so on) and variable rules how to apply the changes to running subscriptions.

	RAWFIE platform during time.	

4.2.11.2 *Final specification of functionalities and interfaces*

The Accounting Service will rely on *Kill Bill*[7] to manages product catalogues, subscriptions, invoices and payments. The Accounting Service reacts to events that may cause cost for the experimenter and sent appropriate messages to the KillBill server. For this, the Accounting Service listens to the Message Bus and looks for changes in the Master Data Repository.

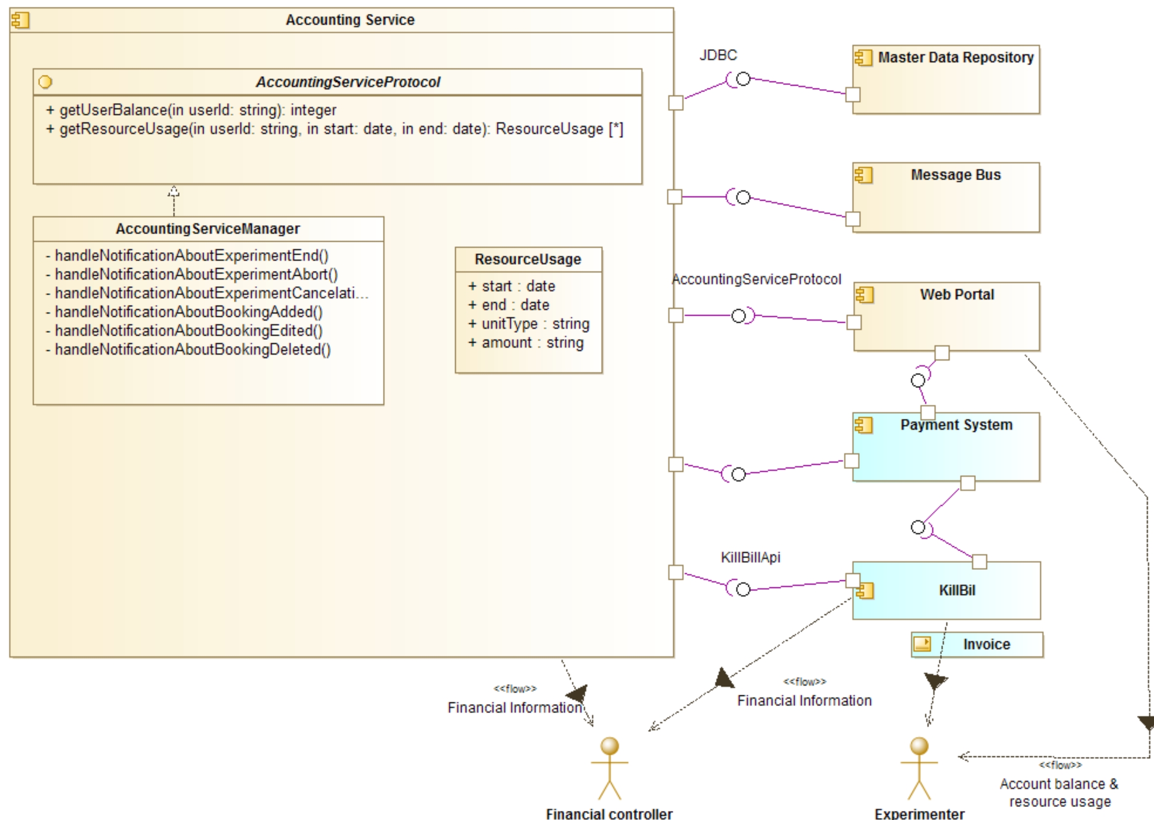

Figure 27: Accounting Service – Class diagram

Please note, that the final technological solutions (open source or not, and so on) for the Accounting Service, are still under investigation and modifications are possible.

4.2.11.3 *Updated sequence diagrams*

Register usage information

1. Through the message bus the Accounting Service receives a notification about an event, that should be charged
2. The Accounting Service requests the updated information from the Master Data repository.
3. The Accounting Service processes the loaded information (choose correct product catalogue item and amount)
4. The Accounting Service sends the catalogue item along with the amount to KillBill

5. KillBill processes the data.

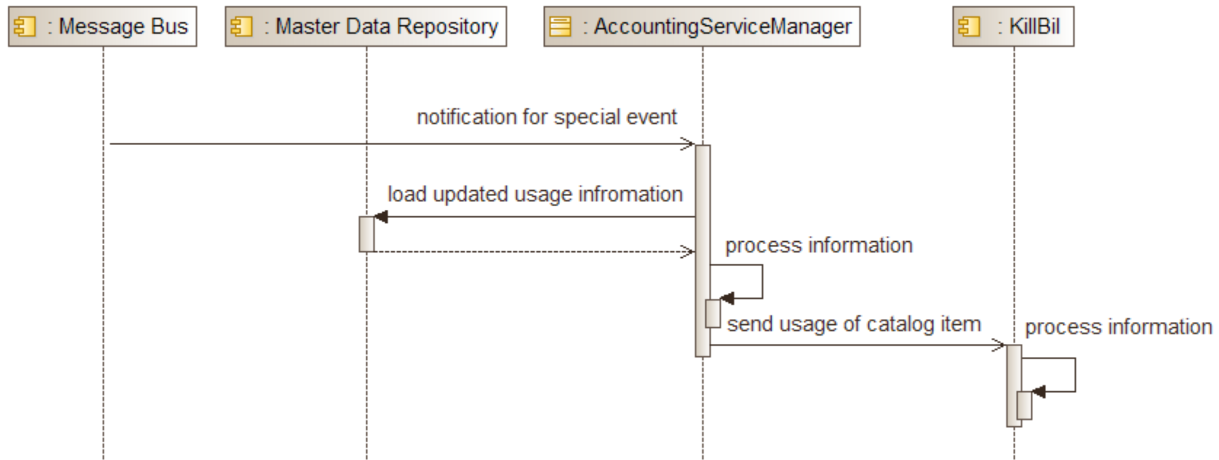

Figure 28: Accounting Service – Register usage information

Register external payment

1. The financial controller sends payment data about an external received payment (e,g, via the bank account) to the accounting service
2. The Accounting Service sends the appropriate payment notification for the account to KillBill
3. KillBill updates the account balance accordingly

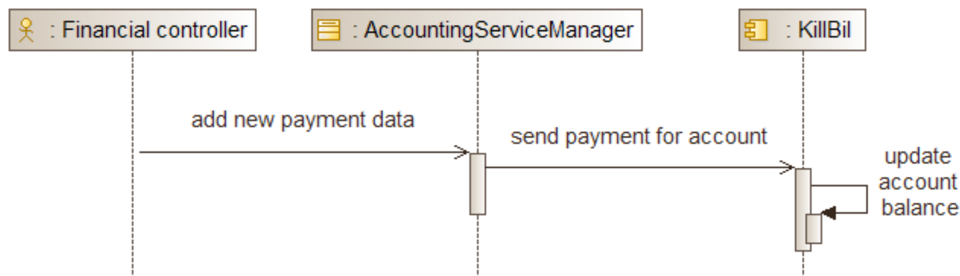

Figure 29: Accounting Service – Register external payment

Payment via a payment system

1. The experimenter starts the payment process in the Web Portal
2. The payment session is started via the AccountingService, KillBill and the external Payment System.
3. The payment session key is returned to the browser of the experimenter and he is redirected to the website of the payment system (with payment session key as parameter).
4. There the experimenter performs the payment and KillBill is notified about payment and acknowledges this.
5. The payment ends successfully and the payment system redirects the experimenter to the Web Portal.
6. In the Web Portal the experimenter can again check the payment result.

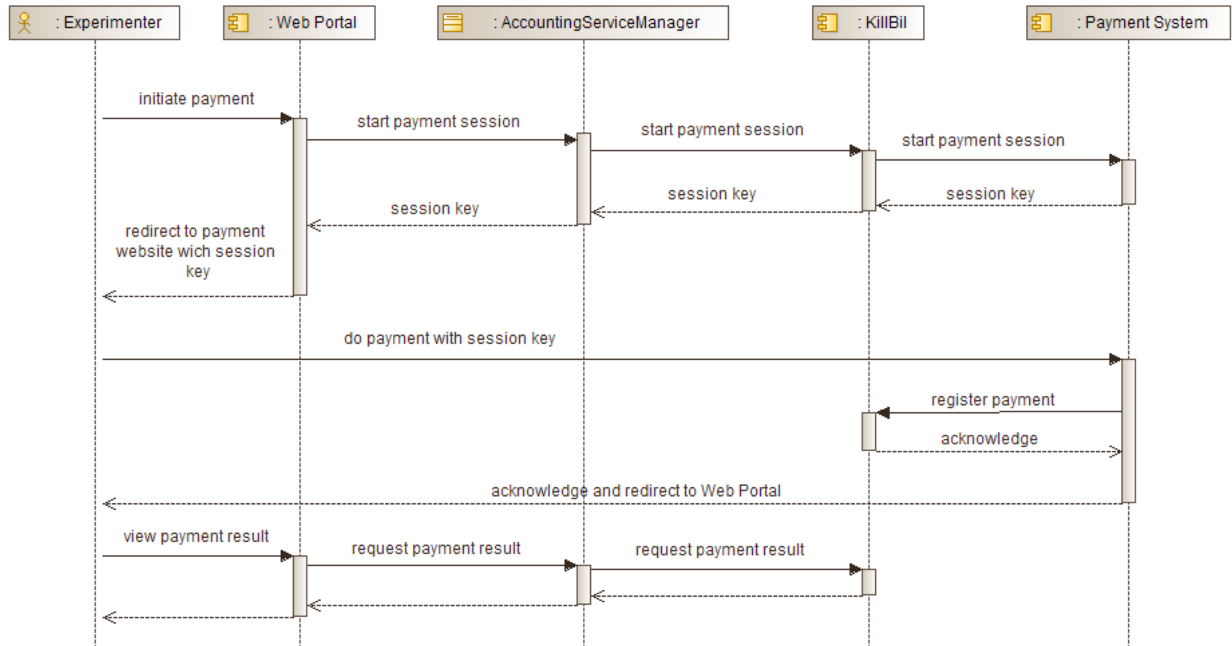

Figure 30: Accounting Service – Payment via payment system

4.2.12 Experiment Controller

The Experiment Controller (EC) is a service placed in the middle tier and is responsible to monitor the smooth execution of each experiment, acting as a 'broker' between the experimenter and the resources in (near) real time.

4.2.12.1 *Component requirements as identified in D3.3*

ID (Priority)	Description	Requirement Mapping with component's functionalities
PT-EXP-C-001 (HIGH)	Cancellation of running experiments should be possible	The cancellation of a running experiment is supported, but Experiment Controller module is bypassed. The cancellation of an ongoing experiment is possible through direct communication between Experiment Monitoring Tool (see 4.1.6 paragraph) and the Resource Controller.
PT-EXP-C-002 (MEDIUM)	Experiment Controller shall allow experimenters to remotely navigate UxVs.	Experiment Controller transfers the experimenter's desired trajectories to the corresponding Resource Controller module.
PT-EXP-C-003 (HIGH)	The Experiment Controller shall support the execution of experiments that involve multiple testbeds	The design of the Experiment Controller provides this functionality.

PT-EXP-C-004 (HIGH)	The Experiment Controller shall be able to support multiple experiments running the same time in parallel	The design of the Experiment Controller provides this functionality.
PT-EXP-C-006 (HIGH)	The Experiment Controller shall support receiving feedback at regular intervals from all testbed facilities about the progress of the experiment in this time interval	Experiment Controller is informed about the progress of the ongoing experiments through specialized status messages.
PT-EXP-C-007 (HIGH)	The Experiment Controller shall be able to override the order of instructions described in the input script while the experiment is running	Experiment Controller will support this functionality in the future.
PT-EXP-C-008 (HIGH)	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	Experiment Controller updates/inserts information inside the Master Database Repository (JDBC) about the status changes of the ongoing experiments (updated tables: experiment_execution, experiment, experimentlog).
PT-EXP-C-009 (HIGH)	The Experiment Controller shall send distinct error and warning messages in every case the experiment's state diverges from the aimed target	This kind of information is included in the distinct status messages that are annotated in the Master Database Repository (JDBC).

4.2.12.2 *Final specification of functionalities and interfaces*

The main functionalities of the Experiment Controller are:

- Transfer the instructions from the Launching Service to the Resource Controller
- Retrieve from the RAWFIE database: i) the appropriate EDL script, ii) information about the coordinate system of the testbed and iii) the partitions IDs of the involved vehicles
- Control the distributed status of the experiments
- Receive status updates for all the running experiments from the Resource Controller and keep logs about each state transition
- Update RAWFIE database tables about the outcome of the completed experiments

Required Interfaces

- Message Bus: Experiment Controller interacts with the following components through the Message Bus:
    - Launching Service: Experiment Controller consumes *ExperimentLaunchRequest* messages indicating that a new experiment has to be forwarded to the corresponding testbed facility.
    - Resource Controller, Testbed Manager and Network Controller: Experiment Controller produces *ExperimentStartRequest* that is send to Messages Bus and consumed by any testbed tier component. Additionally, Experiment Controller receives feedback (*ExperimentStatusMsg* messages) from all testbed Resource Controllers about the progress of each ongoing experiment.
- Master Database Repository (JDBC): Experiment Controller reads and stores information in the RAWFIE database:
    - Experiment Controller retrieves experiment related information (EDL script, partition IDs, coordinate system, canonical names) and proceeds with the certain preparatory actions required prior to the experiment dispatching.
    - Experiment Controller updates/inserts information about the status changes of the ongoing experiments (tables: experiment_execution, experiment, experimentlog)

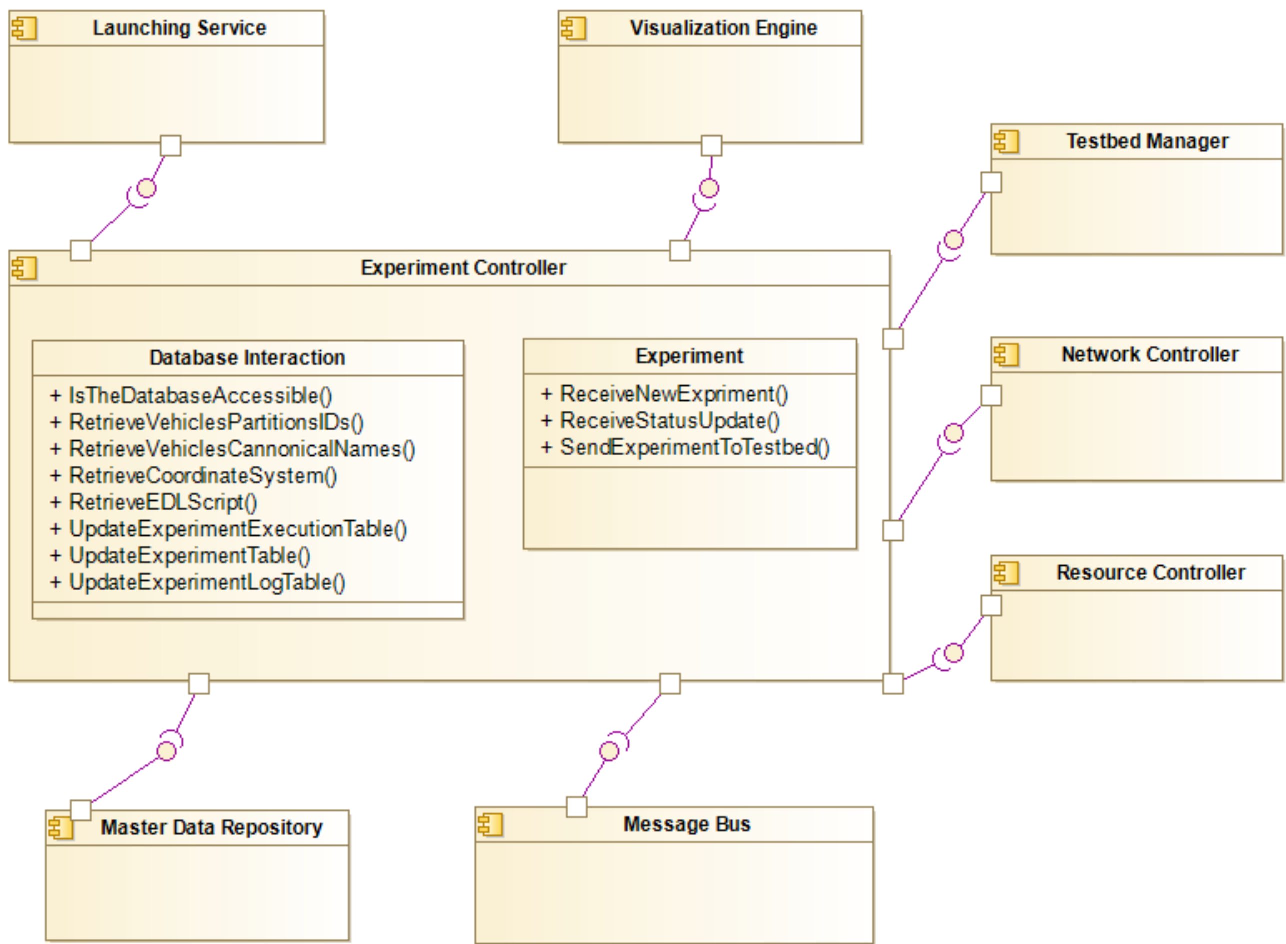

Figure 31 Experiment Controller - Class Diagram

4.2.12.3 *Updated sequence diagrams*

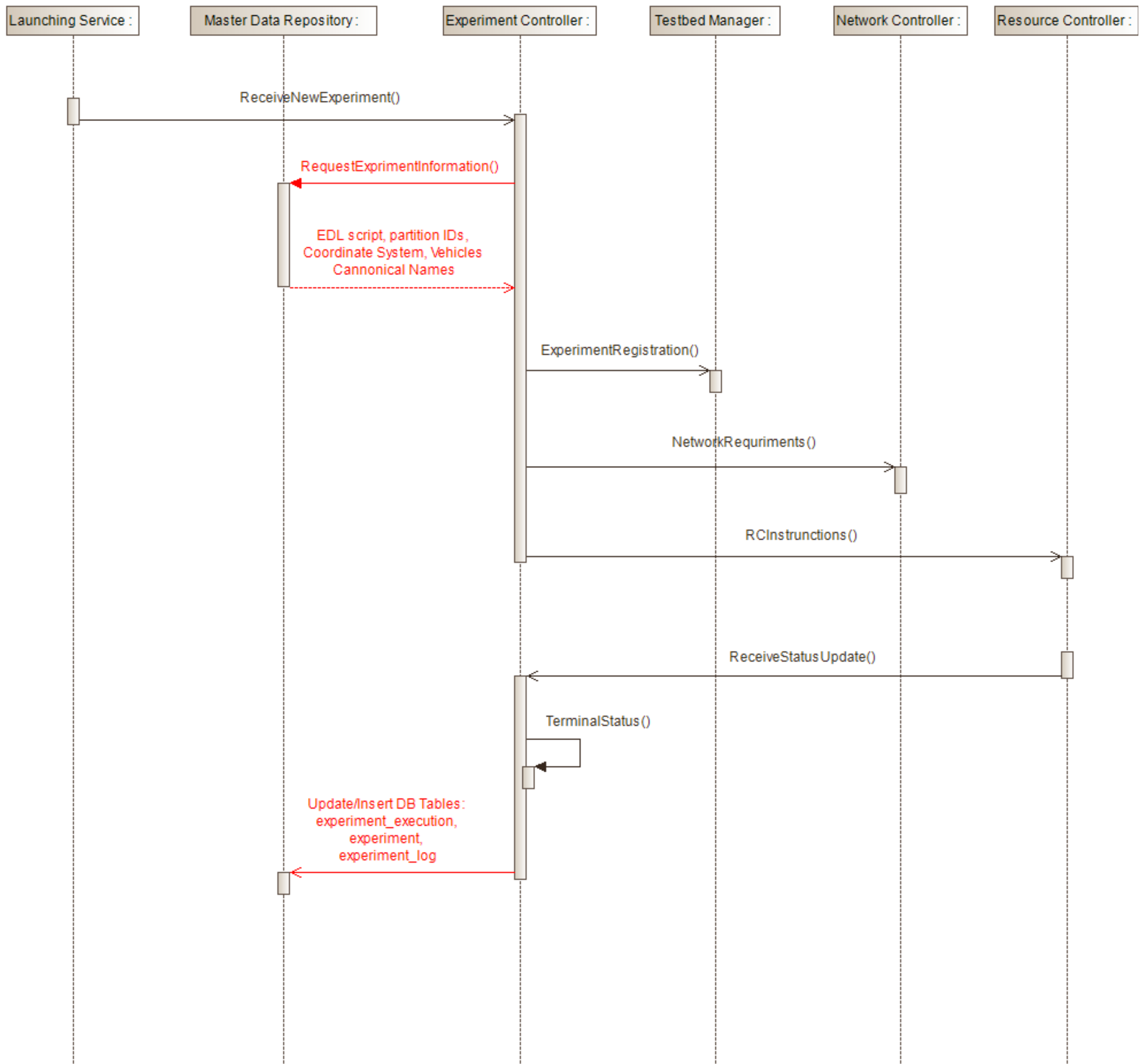

Figure 32 Experiment Controller - Sequence diagram

4.3 Testbed Tier (Testbeds and Resources control components)

4.3.1 Overview

This subsection describes the Testbed Control, monitoring and analysis components. Network Controller manages the network connections and the switching between different technologies in the testbed. Testbed Manager is responsible for the administration of the devices of each federation Testbeds as well as the operational control of all Testbed components needed for the successful execution of each experiment. Monitoring Manager is actually part of the Testbed Manager component and enables the observation of the resources usage of the testbed and UxV nodes. Aggregate Manager undertakes the responsibility to enable RAWFIE compatibility with the Slice-based Federation Architecture (SFA) that represents the de facto standard API for testbeds federation.

At testbed level the Resource Controller is the main navigation component which ensures the safe and accurate guidance of the UxVs based on the user's preferences working closely with Experiment controller that runs at platform level. The Proximity component allows members of a swarm of autonomous vehicles to discover the existence and possibly interact with each other with very low latency without depending on the RAWFIE middleware or any other ground equipment. A high level diagram showing the interactions among testbed components is presented in Figure 33. More details about the Testbed Tier components are given in the following subsections.

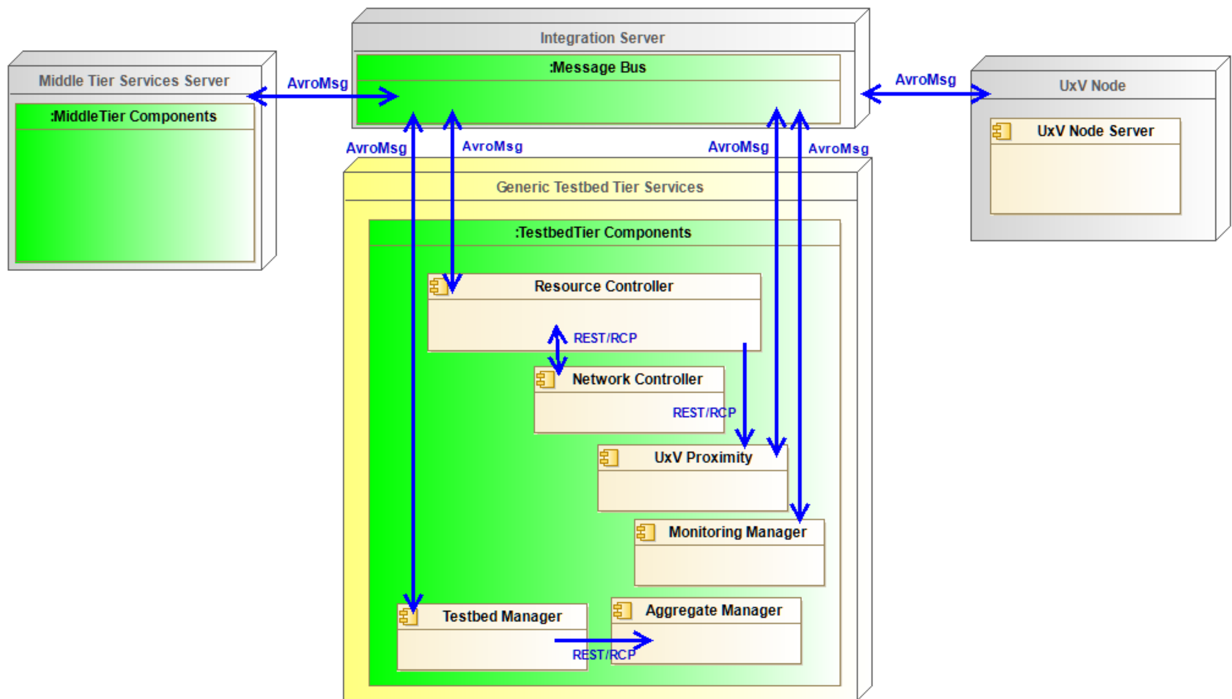

Figure 33: Testbed control, analysis and monitoring– Deployment / Components Diagram

4.3.2 Monitoring Manager

Monitoring Manager is responsible for the monitoring of the status of a testbed and the devices belonging to it, at functional level, reading information about the 'health of the devices' and their current activity.

4.3.2.1 *Component requirements as identified in D3.3*

ID (Priority)	Description	Requirement Mapping with component's functionalities
TB-MOM-001 (HIGH)	The Monitoring Manager component should be able to provide information about the capabilities of each resource node.	MonitoringController class provides startUxVConsumers() method which is responsible for enabling the consumption of Message Bus messages from each resource node. Upon reception of the message the current status of the resource is calculated
TB-MOM-002 (HIGH)	The Monitoring Manager component should collect and report current status of computing resources of the testbed facilities	HealthStatusService class provides the methods for reading the values of CPU, memory and disk usage and assessing the current testbed status
TB-MOM-003 (HIGH)	The Monitoring Manager component should store periodically all testbed information	JDBC Connectors provide all necessary functions for storing testbed and resources statuses in permanent storage
TB-MOM-004 (HIGH)	Testbed monitoring manager should be able to transmit the current status to the System Monitoring Service.	HealthStatusService class implements the fillHealthStatus() method responsible for the production of the Avro message TestbedHealthStatus and transmits it to the Message Bus
TB-MOM-005 (MEDIUM)	Monitoring Manager should be able to communicate and collect information from other services that provide important information related to the operation of testbed facility	Monitoring Manager subscribes to Message Bus topics that provide information from other testbed services

4.3.2.2 *Final specification of functionalities and interfaces*

The main functionalities of the Monitoring Manager are:

- Periodically check the current status of the available resources in the facility like battery lifetime, CPU load, free storage volume, bit error rate, etc.
- Periodically check the status of the testbed facilities computing resources.
- Store the status of the testbed characteristics and the devices in a Database/data log.
- Transmit current status information to the System Monitoring Service through the Message Bus
- Communicate with services running at testbed level like Network Controller and Resource Controller

- Display and visualize the data of the monitored resources to the testbed operators in a user friendly format. For this purpose it shares the same user interface with testbed Manager Manager

Operations and Attributes

Monitoring Manager provides all necessary classes for receiving the current level of resources for the nodes participating in experiments. After the initialization of the component MonitoringController class starts the appropriate consumers of messages from UxV nodes like current levels of fuel, CPU and storage volume upon which the current status of UxV nodes is calculated. In parallel startHealthService method starts a periodic task in which the current levels of the computing machine resources are measured and an overall status is calculated and transmitted through the Message Bus in the relevant topic TestbedHealthStatus. Monitoring Manager is equipped with the appropriate classes (HealthStatusDAO and UxVStatusDAO respectively) that are responsible for interactions and queries with the permanent storage which in this case is realized with the corresponding tables in the testbed local database. Visualization of the results is achieved with the use of displayHealthStatus() and displayUxVStatuses() methods of MonitoringView class which enable the presentation of the monitored resources in the graphical user interface of the Testbed Manager. A high level class diagram of the Monitoring Manager with its sub-components is presented in Figure 34.

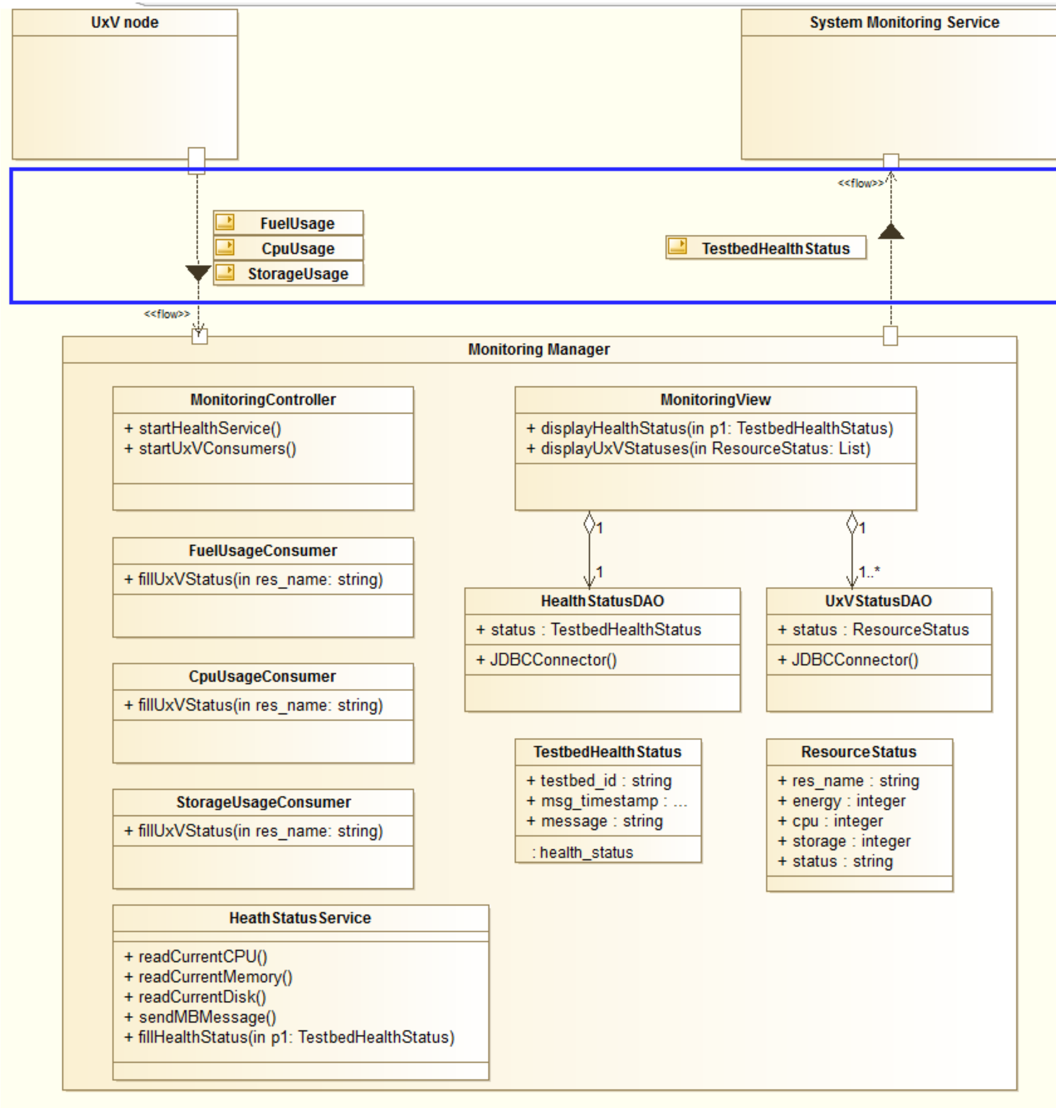

Figure 34: Monitoring Manager - Class diagram

Required Interfaces

- Message Bus: Monitoring Manager interacts with the following components through the Message Bus:
    - System Monitoring Service: Monitoring Manager periodically transmits a Testbed Health Status message which contains information about the utilization of the computing resources of the testbed
    - UxV Node: Monitoring Manager consumes messages which display resources current levels during the execution of experiments like fuel usage, CPU usage, storage usage.
- Local Database Repository (JDBC): Monitoring Manager reads and store information in the database deployed at each testbed.

4.3.2.3 *Updated sequence diagrams*

<u>Interactions and Relationships with other components</u>

The interactions of the Monitoring Manager with other components are presented in the following sequence diagram.

1. Monitoring Manager collects periodically testbed status information, fills TestbedHealthStatus message and transmits it to the Message Bus.
2. System Monitoring Service can load the data for the current status of the testbed after subscribing to the relevant Message Bus topic.
3. Monitoring Manager reads messages sent from UxVs that are related to the current levels of resources (FuelUsage, CpuUsage and StorageUsage messages)
4. Calculates their current status upon these values
5. Displays the result in the user interface.

In the diagram red colour is used to highlight the use of message bus for monitoring manager components interactions marking the difference with the previous version of the architecture.

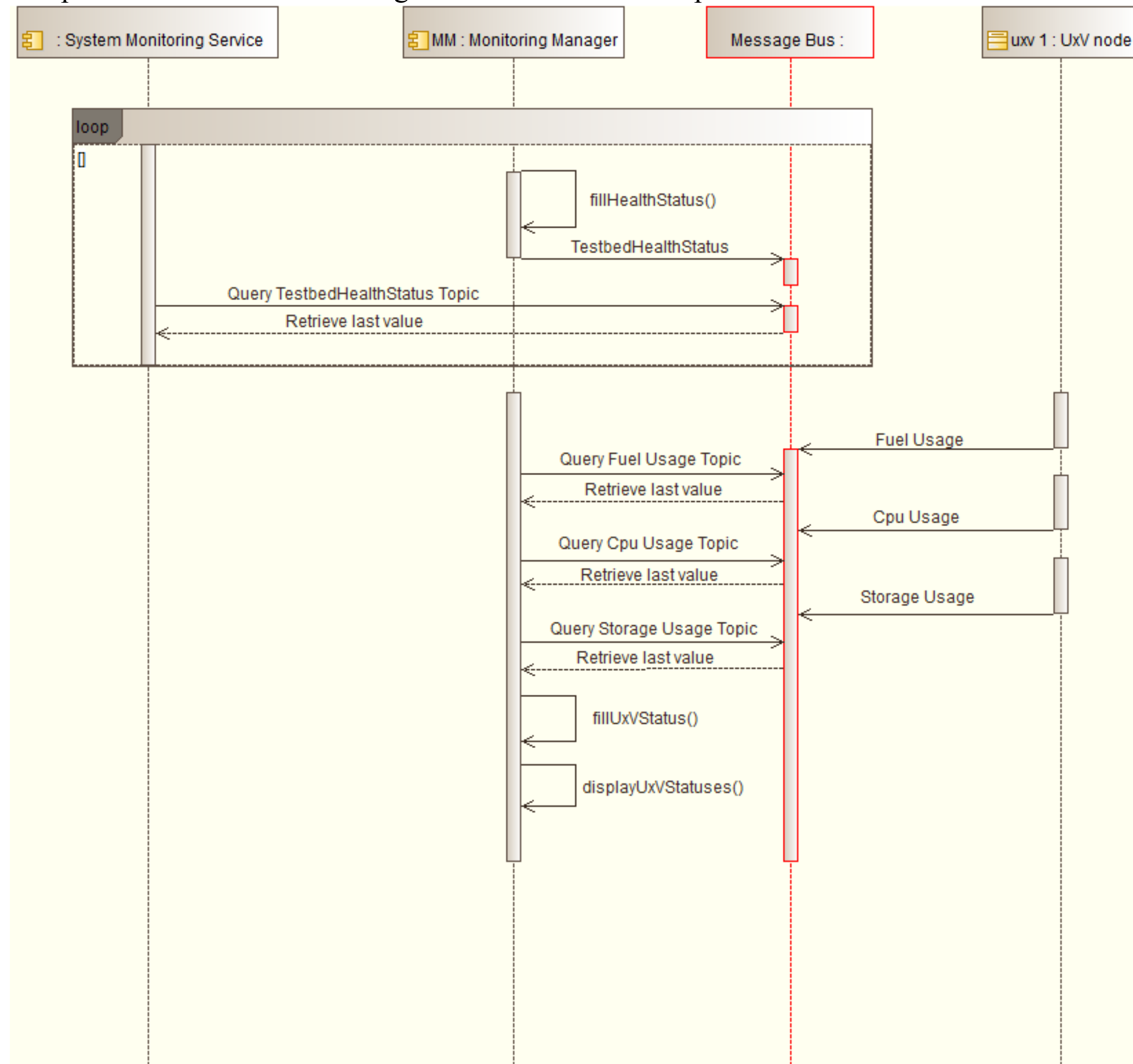

Figure 35: Monitoring Manager sent and received Message Bus messages

4.3.3 Network Controller

This paragraph describes the RAWFIE Network Controller component, starting from a brief summary of the requirements. In particular, the design alternatives and implementation trade-offs are explained so that it is possible to follow the decision path that led to the present component.

4.3.3.1 *Component requirements as identified in D3.3*

The Network Controller manages the network connections and the switching between different technologies in the testbed. For example if a problem occurs in the communication of the resource with the RC and subsequently with the Experiment Manager on the RAWFIE middleware, a fall-back interface is engaged. Through this procedure, the other networking interface/device is enabled to avoid the uncontrolled operation of the mobile unit and associated damages in the infrastructure. In addition this component is responsible for security issues. The switching alternative can be also triggered by the executed experiment.

ID (Priority)	Description	Requirement Mapping with component's functionalities
TB-NEC-001 (MEDIUM)	The RAWFIE communication resources shall be managed to offer seamless connectivity in the normal operations of the system.	The Network Controller has access to the Database where the metadata are available, as well as to the measurements of the link quality done by the UxV. Switching is automatic or based on the experience. The Network Controller is able to send commands to make the UxV switch to a specific network interface.
TB-NEC-002 (MEDIUM)	Provision of network communication resource	UXv and Testbeds offer various interfaces, whose metadata are described in the Database.
TB-NEC-003 (MEDIUM)	Alternative communication system	Same as TB-NEC-002. UxV may ultimately operate as an autonomous swarm using the Proximity component.
TB-NEC-004 (MEDIUM)	Management of the communication system	See TB-NEC-001
TB-NEC-005 (MEDIUM)	Time constraint verification and notification	The Network Controller does not verify directly that the time constraints are met. This verification can be done externally by the experiment on the basis of the measurements done by UxV as well as temporal properties of the data exchanged in the Kafka message bus.

4.3.3.2 *Final specification of functionalities and interfaces*

Static data is to be located in the testbed manager data base. A table called *net_interfaces* described all the available network interfaces, using an integer to identify them:

	net_interface_id [PK] integer	net_technology character varying(32)	vendor character varying(32)	band character varying(8)	main_frequency character varying(32)	channel_bandwidth character varying(32)	number_of_antennas integer
1	1	wifi					
2	2	3G					
*	3	zigbee					

Table 1: net_interfaces testbed manager table

A second table named resource_net_interfaces defines the matching with the network interface descriptions and the actual interfaces present on the UxVs:

	net_interface_id [PK] integer	resource_id [PK] integer	is_default boolean	interface_name character varying(12)
1	1	1	TRUE	wifi0
2	1	2	TRUE	wifi1
3	2	1		wifi0
4	3	2		wifi1
*				

Table 2: resources_net_interfaces table of the testbed manager data base

In the above table, the *resource_id* column identifies the UxV and *net_interface_id* matches the same column of the *net_interfaces table*.

Dynamic data

Dynamic data is also reworked in the second iteration with new topic names and data format definition. The dynamic data is described in Table 3 following the topic names definition below:

- NetwReportingPeriod – network performance reporting period for UxVs
- NetwSelectIf – select network interface command for UxVs
- NetwPerfUxV – UxV network connection performance reports
- GlobalNetwPerf – global network performance indicator

All fields of the records sent to the NetwPerfUxV topic are averages computed within the window given by the reporting period.

Topic	**Param**	**Desc**	**Unit**	**Type**	**Publisher**	**Subscriber (s)**
NetwReportingPeriod	period	network performance reporting period	seconds	int	Network Controller	UxVs
NetwSelectIf	iface	*net_interface_id* of table *resources_net_interfaces*	-	int	Network Controller	UxVs
NetwPerfUxV	iface	As above	-	int	UxVs	Network Controller
	bitrate	effective uplink bitrate	Kb/s	int		
	latency	average time until a post is available on the message bus	ms	int		
	rssi	As given by	dB(m)	int		

		the network interface card				
	security	Whether the message bus connection is secured or not	-	bool		
GlobalNetwPerf	resource name	Rawfie UxV identifier (canonical name)	-	string	Network Controller	Experiment Controller
	iface	interface ID as above	-	int		
	indicator	Network performance indicator	From 0 to 5	int		
	…	*Resource ID, Interface ID and performance indicator for each UxV*				
	…					
	…					

Table 3: 2nd iteration dynamic data with topic names

4.3.3.3 *Updated sequence diagrams*

The Network Controller interacts with the Resource Controller in order to acquire information from the UxVs. The Monitoring Manager can also gather statistics for the network technologies and status of the experiments.

The workflow of the Network Controller is the following:

- Wait for experiment start
    - Subscribe to Kafka topic “ExperimentStartRequest”.
- Upon experiment start:
    - Extract list of UxV’s canonical names
    - For each resource, gather list of interface from the data base. Get the default interface
    - Broadcast NetwReportingPeriod topic to set the reporting period.
- Subscribe to NetwPerfUxV topic
- Upon NetwPerfUxV message, update performance indicators, publish a GlobalNetwPerf message
- Upon request to change the interface, authorise (or not) the change and publish NetwSelectIf.

4.3.4 Resource Controller

The core component of the navigation system is the Resource Controller which ensures the safe and accurate guidance of the UxVs based on the user's preferences. Additionally, Resource Controller commands each device to switch onboard sensors on and off.

4.3.4.1.1 Component requirements as identified in D3.3

ID (Priority)	Description	Requirement Mapping with component's functionalities
TB-REC-001 (MEDIUM)	RAWFIE platform shall support a semi-autonomously way of navigation of the UxVs	Navigation_Service will be responsible for this feature.
TB-REC-002 (MEDIUM)	RAWFIE platform should be able to activate the "Emergency Scenario"	Abort messages will be dispatched to all the involving vehicles if any of the emergency conditions are met. In the future, the "Emergency Scenario" is going to be enriched with more sophisticated events/actions.
TB-REC-003 (HIGH)	The Resource Controller shall receive location messages from the vehicles at regular intervals	The Resource Controller receives location messages containing the current position of each operating UxV.
TB-REC-004 (HIGH)	The Resource Controller shall transmit the next location for the current experiment to the vehicles	Goto commands are transmitted to the corresponding topic of each testbed. The distinction between different UxVs is achieved by the partitioning on the Kafka messages.
TB-REC-005 (HIGH)	The Resource Controller shall be able to plan the next location that will be transmitted in the vehicle taking into account the locations of all UxVs that are active in that testbed	The Resource Controller transmits the new set of waypoints only when all the UxVs have reached the previously transmitted waypoints (within a pre-specified radius of tolerance).
TB-REC-006 (HIGH)	For the experiment accomplishment the Resource Controller shall operate in close coordination with the Experiment Controller	The Resource Controller informs Experiment controller about the progress of the ongoing experiment with specialized status messages.

4.3.4.2 *Final specification of functionalities and interfaces*

The main functionalities of the Resource Controller are:

- Resource Controller filters the messages from Experiment Controller and identifies the experiments that have to be executed on its own testbed.

- Resource Controller is able to parse the EDL script and identify the desirable waypoints.
- Each one of these waypoints is dispatched to the appropriate Kafka partition in order to be executed by the appropriate UxV.
- The communication protocol takes into account the location of all the UxVs in order to send the next set of waypoints.
- The Resource Controller is able to detect and identify possible safety violations. If the given instructions violate the safety constraints, the Resource Controller ignores these directions and returns appropriate warning messages to the Experiment Controller.
- Resource Controller is able to activate and deactivate the available sensors in order to control the measurements feed.
- Resource Controller sends distinct status messages informing the Experiment Controller regarding the progress of the experiment's execution.

Required Interfaces

- Resource Controller interacts with the following components through the Message Bus:
    - Experiment Controller: Resource Controller (RC) consumes *ExperimentStartRequest* messages describing details of a new experiment. Additionally, RC produces *ExperimentStatusMsg* messages to inform the Experiment Controller about the progress of the ongoing experiment
    - UxV node: RC consumes *Location_TestbedName* messages describing the current location of all involved vehicles. RC also publishes *Goto_TestBedName* and *SensorPublishControl_TestbedName* type of messages informing about the desirable locations and the activation/deactivation of available sensors, respectively. In case of emergency, RC broadcasts *Abort* messages to all the operational UxVs.
    - Experiment Monitoring Tool: Resource Controller consumes *ExperimentCancelRequest* messages indicating the cancelation of an ongoing experiment.

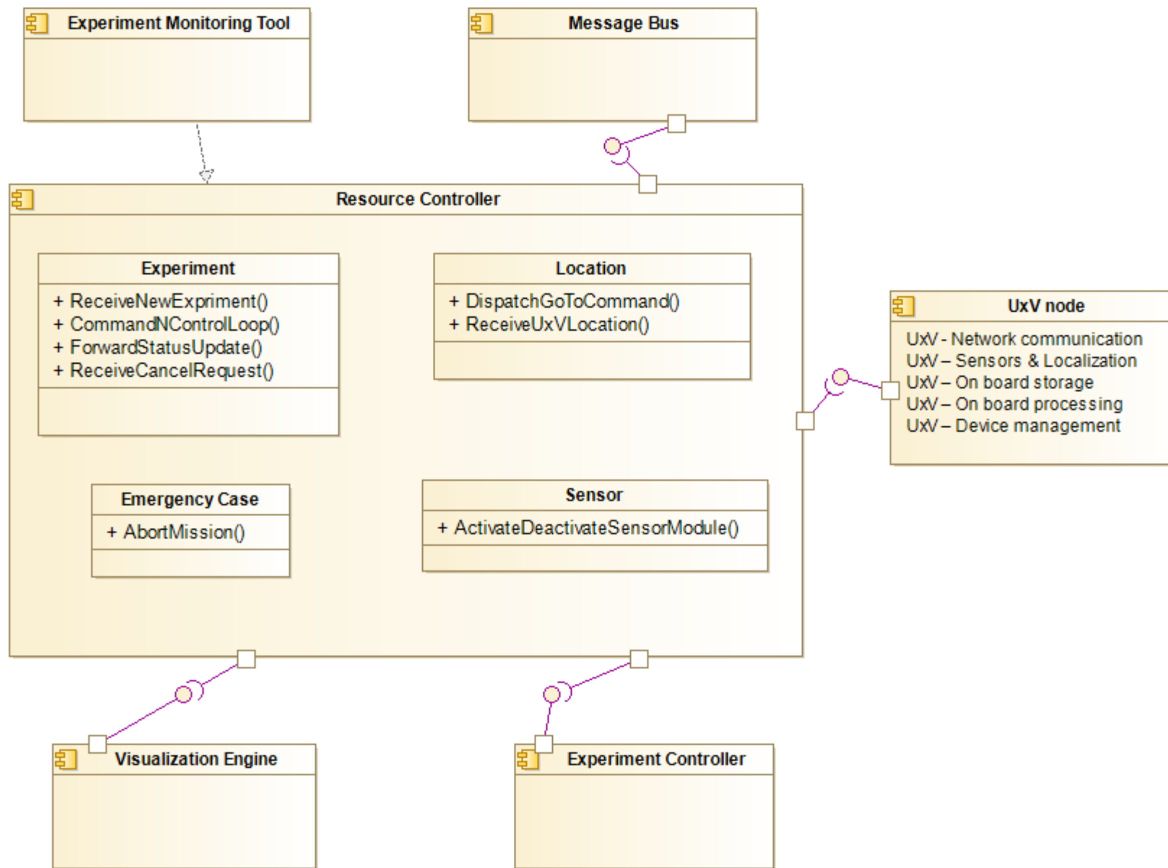

Figure 36 Resource Controller - Class Diagram

4.3.4.3 *Updated sequence diagrams*

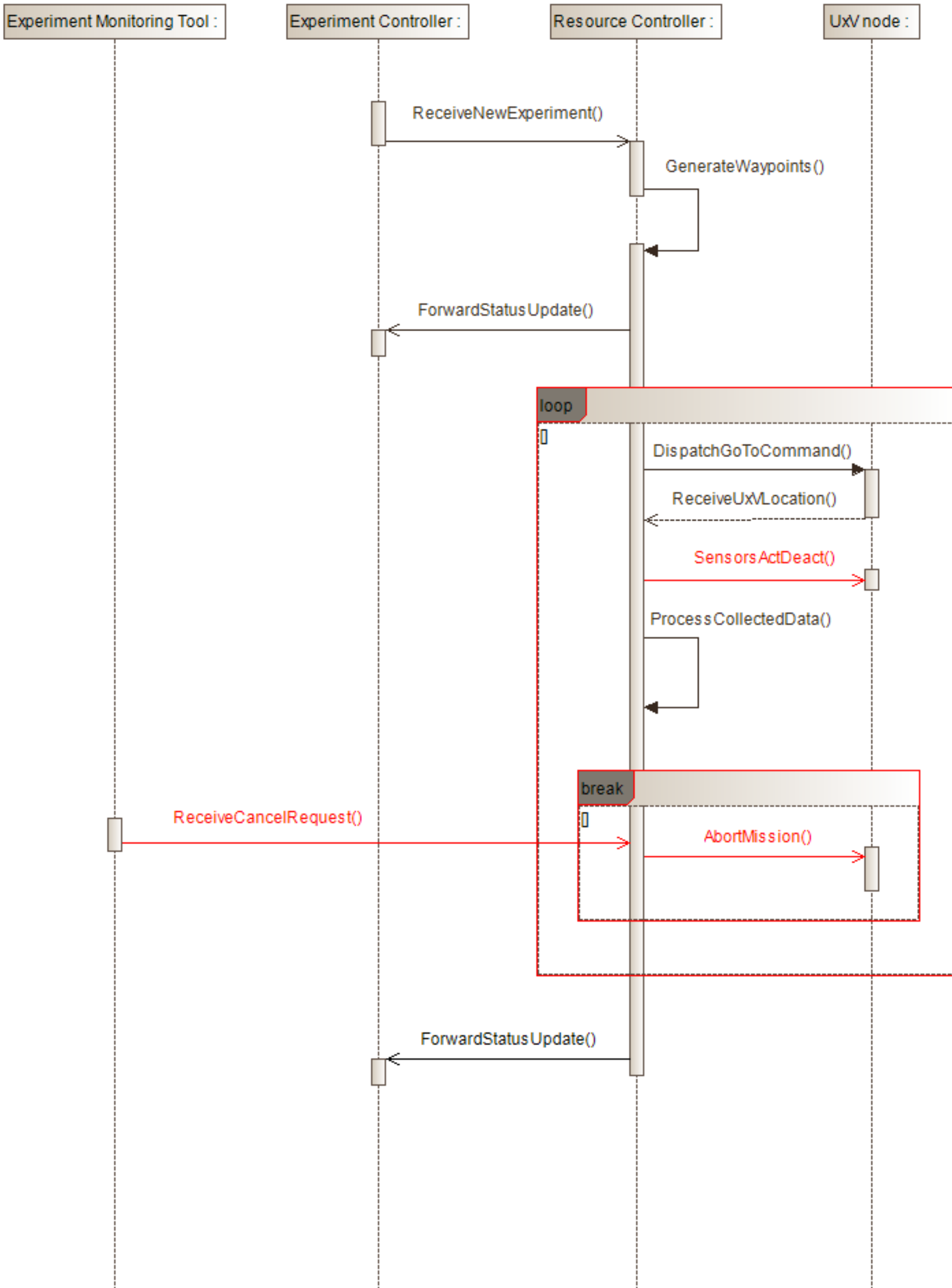

Figure 37 - Resource Controller - Sequence diagram

4.3.5 UxV Proximity component

This section presents the design of a low power wireless proximity component for the unmanned vehicles (UxV) taking part to the Rawfie platform as an element of a Testbed. The main objective of the proximity component is to allow members of a swarm of autonomous vehicles to discover the existence and possibly interact with each other with very low latency without depending on the Rawfie middleware or any other ground equipment.

4.3.5.1 *Component requirements as identified in D3.3*

The table below shows RAWFIE requirements extracted from D3.2 for which the proximity component is useful and helps satisfying.

ID (Priority)	Description	Requirement Mapping with component's functionalities
PT-NAV-T-001 (HIGH)	This component will provide to the user the ability to remotely navigate a squad of UxVs through a user friendly interface.	Neighbours detection, identification, distance estimation, collision avoidance, navigation (heading)
TB-NEC-003 (MEDIUM)	Alternative communication system	Implements the alternative communication system, relay for UxV's disconnected from the main network (communication services)
TB-NEC-004 (MEDIUM)	Management of the communication system	Signals strength measurement on the proximity radio interface, diffusion of primary radio interface link quality status (self checking, management services)
TB-REC-002 (MEDIUM)	RAWFIE platform should be able to activate the "Emergency Scenario"	Use the proximity component to find or communicate with a lost UxV (beaconing and emergency services)
TB-REC-003 (HIGH)	The Resource Controller shall receive location messages from the vehicles at regular intervals	The Proximity component exposes the perceived neighbourhood information in the report sent to the Resource controller (neighbourhood information service)
UXV-NET-002 (MEDIUM)	UxVs should be able to Synchronize their Time-References between them.	Time synchronisation using the proximity component (synchronisation service) in case of failure of the primary communication interface.
UXV-NET-004 (HIGH)	Each UxV node shall be equipped with primary and secondary communication means.	Secondary communication interface, redundancy (RAWFIE communication services)
UXV-NET-005 (MEDIUM)	UxV network interface management	The proximity component will provide a management interface (management services)
UXV-NET-	Neighbouring UxV	Neighbours detection, distance or proximity

008 (MEDIUM)	monitoring	estimation, publication of position and speed vector as well as status (Neighbour monitoring services)
UXV-NET-009 (HIGH)	Each UxV node should be able to send navigation state feedback with at least 2 Hz frequency and maximum 1 sec latency when within radio communication reach.	Navigation information publication for the neighbourhood with strict real time constraints (real-time communication services for exchanging navigation
UXV-PRC-001 (HIGH)	Each UxV shall be able to operate autonomously.	Enables autonomous operation in swarms.
UXV-PRC-002 (MEDIUM)	The UxV should provide collision avoidance mechanism	Neighbours detection, distance or proximity estimation, publication of position and speed vector, distance estimation (anti-collision service)
UXV-PRC-004 (MEDIUM)	UxVs should be able to cooperate during the execution of an experiment.	Direct, real time communication between neighbouring UxV's

4.3.5.2 *Final specification of functionalities and interfaces*

Functional architecture

The proximity component is made of two elements: the Delegate and the Head. The Proximity Delegate acts as a proxy to the proximity component for the local Proximity Component client (actually abstracted by the Delegate). It is named "Delegate" to avoid the alliterative "Proximity Proxy" combination. The following subsections describe the respective responsibilities of the Proximity elements. The Proximity Head is the active part of the component, which handles the subscription requests and publications issued by the Delegate.

Proximity Delegate

The Proximity Delegate interacts with the other UxV components and forwards their requests and data to the Proximity head over a serial line. In the downlink direction, subscriptions from the UxV Node are translated from the Rawfie middleware world (Kafka) into a more compact format accepted by the proximity component protocol. Data publications are filtered with respect to content, context and delivery specifications. In the uplink direction, the Proximity Delegate receives subscriptions from other vehicles sent-up by the Proximity Head and translates them into Rawfie middleware subscriptions by subscribing to the related topics. The same is done with uplink data which is translated and published within the primary Rawfie "world".

Proximity Head

The Proximity Head executes the Publish-Subscribe protocol, forwards uplink subscriptions (coming from other UxVs) to the Proximity Delegate and transmits publications submitted by the Proximity Delegate (thus from the UxV). The Proximity Head also manages subscriptions made

by its own UxV through the Delegate and filters incoming uplink traffic accordingly before sending it up to the delegate.

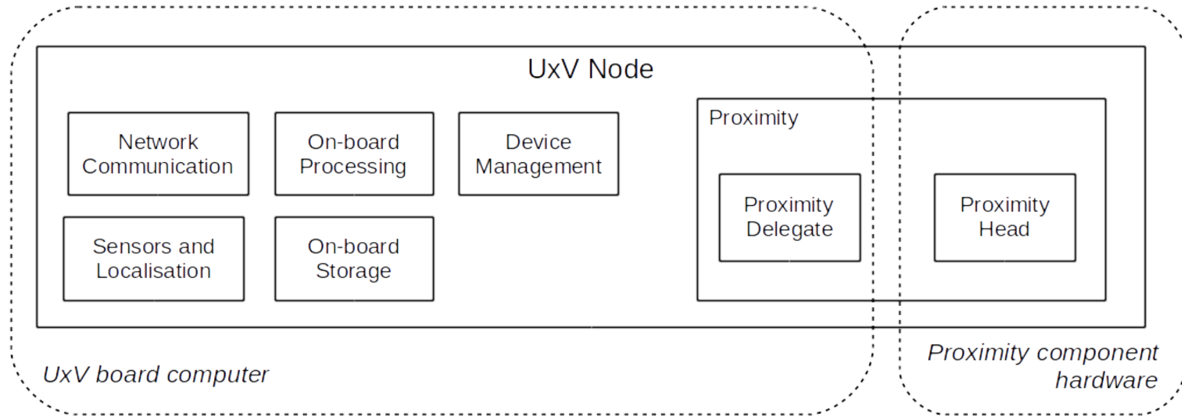

Figure 38: UxV Node architecure with Proximity component. Dotted line boxes represent hardware. Continuous line boxes represent software components.

Services

The proximity component offers a generic public-subscribe service on a secondary, local short range radio module. The topics that can be published by the proximity component will have to comply with some maximal size and throughput constraints. The interface with the UxV shall be able to describe what is acceptable to publish under which constraints at any given time.

Among the topics likely to be published by the proximity components are:

- UxV identification,
- Position,
- Speed vector,
- Perceived neighbourhood such as the signal strength received on the Proximity component radio for qualitative or quantitative distance estimation,
- Status of the internal components of the UxV, mode, etc.
- Sensor readings.

The most used topics will be published (i.e. broadcasted) spontaneously, maybe on a regular basis (cyclically). Other, possibly user defined topics will be managed through subscriptions. A proximity component shall be able to publish topics generated by the other UxV components or by itself such as:

- Neighbourhood information, statistics on appearance/disappearance
- Communication statistics (packet rates, protocol mechanisms, errors, uptime, latencies, etc.)

Additionally some wireless sensors compatible with the proximity component may be deployed off the vehicles (on the ground, on/in the water) and their data gathered by UxV's passing by.

Interfaces

The proximity component has to satisfy the following requirements regarding its interface with the UxV components in order to fulfil its functional requirements:

- Translate subscriptions received from other nodes through the proximity component radio interface and forward them to the UxV "client" of the proximity component.
- Subscribe to and receive topics published by the other UxV components.
- Access to some UxV component properties such as identifier, status, etc…
- Allows the local UxV client for subscribing to proximity component topics published by other UxVs.
- Forward data received from the proximity component radio interface to the local UxV client that has subscribed to it.

The following interface is provided by the proximity component.

- Manage and configure the proximity component parameters,
- Publish and subscribe to proximity topics (topics published by individual drones, such as status, estimated distance for a pair, UxV speed or altitude, etc.),
- Communication statistics (packet rates, protocol mechanisms, errors, uptime, latencies, etc.)
- Publish and subscribe to status of individual UxV's and groups of UxV's,
- Publish and subscribe to user defined topics
- Notification of various events using dedicated topics: appearance and disappearance, distance thresholds reached, etc.

The Proximity component requires the following interface to be integrated into a system. The management interface is made of:

- Register, unregister
- Start and stop the proximity component
- Publish topic, subscribe to topic

The filter interface could be of the type *setFilter*, *getFilter* or filters/operators can be attached to subscriptions.

The interface between the UxV and the proximity component shall be OS-agnostic.

4.3.6 Testbed Manager

The Testbed Manager is responsible for the administration of the devices of each one of the federation testbeds as well as the operational control of all testbed components needed for the successful execution of each experiment.

4.3.6.1 *Component requirements as identified in D3.3*

ID (Priority)	Description	Requirement Mapping with component's functionalities
TB-MAN-001 (HIGH)	Testbed Manager shall support permanent storage of all testbed attributes and resources attributes that belong to testbed	A local database is implemented at local level with all Testbed Manager classes having access to it
TB-MAN-002 (HIGH)	Testbed Manager shall provide information about the capabilities of each resource node	The capabilities of each resource node are presented through ShowUxVDetails() of UxV class
TB-MAN-003 (HIGH)	Testbed Manager shall check periodically the status of all other services running at testbed level	Testbed Services class implements this functionality
TB-MAN-004 (HIGH)	Testbed Manager shall contain a registration log for all the experiments executed in the testbed	JDBC Connector implemented in the Experiment class enables writing in the permanent storage (local database) for all the experiments
TB-MAN-005 (HIGH)	Testbed Manager shall be periodically informed about the status of all running experiments in the testbed	Testbed Manager receives periodically the status of all running experiments from Resource Controller through Message Bus after subscribing to ExperimentStatusMsg topic
TB-MAN-006 (MEDIUM)	Testbed Manager shall store configuration parameters for the UxVs in the relevant testbed	This functionality is provided from configureUxV() operation of the Experiment class
TB-MAN-007 (HIGH)	Testbed Manager shall implement a user interface to support the interactions between testbed operators and machines	UxV class provides the ability to add and remove resources (addNewUxV() and removeUxV())
TB-MAN-008 (HIGH)	Testbed Manager shall be capable to handle temporary interruption of communication and store data locally in case of	OBSOLETE This functionality will be inherently supported from Message Bus

	transmission failure	
TB-MAN-009 (LOW)	Testbed Manager may provide statistical data/information about testbed operation	Statistics class provides this functionality
TB-MAN-010 (HIGH)	Testbed Manager shall provide the ability to cancel an ongoing experiment in case of communication failure with the RAWFIE platform	Experiment class provides cancelExperiment() method which enables Testbed Manager to transmit an ExperimentCancelRequest message and cancel the execution of an ongoing experiment

4.3.6.2 *Final specification of functionalities and interfaces*

The main responsibilities of the Testbed Manager are:

- Provide a graphical user interface through which a user can add new testbed resources and view the capabilities of each resource node. Monitoring Manager uses the same interface for monitoring the usage of UxV and testbed resources
- Call the REST methods of the Testbed Directory Service when adding, updating or deleting resources to make consistent the content of local and master repositories
- Keep a registration log for all the experiments in the relevant testbed
- Periodically check the status of all other services running at testbed
- Store configuration parameters for the UxVs in the relevant Testbed
- Provide statistical information for the testbed usage
- Provide the required interfaces to interact with SFA Aggregation Manager

The structure of Testbed Manager is depicted in the class diagram of Figure 39, where the associations and operations of its core classes are presented. A permanent storage in the form of a local database is implemented at testbed level and classes are enabled to access it for activities where permanent storage is needed. The component contains the *UxV class* through which the testbed operator can add and delete resources and view the exact details of each resource registered in the testbed. Through the *SFA AM Driver* class these resources are transformed in SFA-compliant format and communicated to the SFA Aggregate Manager component which is responsible for the SFA-based federation of RAWFIE Testbeds.
Testbed Manager is also responsible to call the REST methods of Testbed Directory Service when creating or editing a resource or when recognizing that testbed is not registered in the Master Repository ensuring this way that the information of local and master repositories is consistent.
The Experiment class of Testbed Manager is responsible to register each new experiment locally taking the information provided from Experiment Controller after the validation of the experiment through Message Bus communication. Beyond of storing the details of the new experiment in the local database Experiment class receives the status from Resource Controller at regular intervals containing information about the experiment progress and logs locally the start and stop as well as all significant events of the experiment. The Experiment class provides also the ability to cancel an ongoing experiment from the local interface in emergency cases where RAWFIE platform is inaccessible or where abnormal behavior detected.

The Testbed Services class is responsible to collect the status of all the other components that run at testbed, inform the platform about the overall status and take the appropriate actions in cases of non-responding components. The Statistics class can present information about the utilization of the testbed or specific resources in user defined temporal intervals. Testbed Manager is designed as a desktop application equipped with Graphical User Interface and TestbedManagerView class is responsible for the visualization of all the relevant information in a user friendly format.

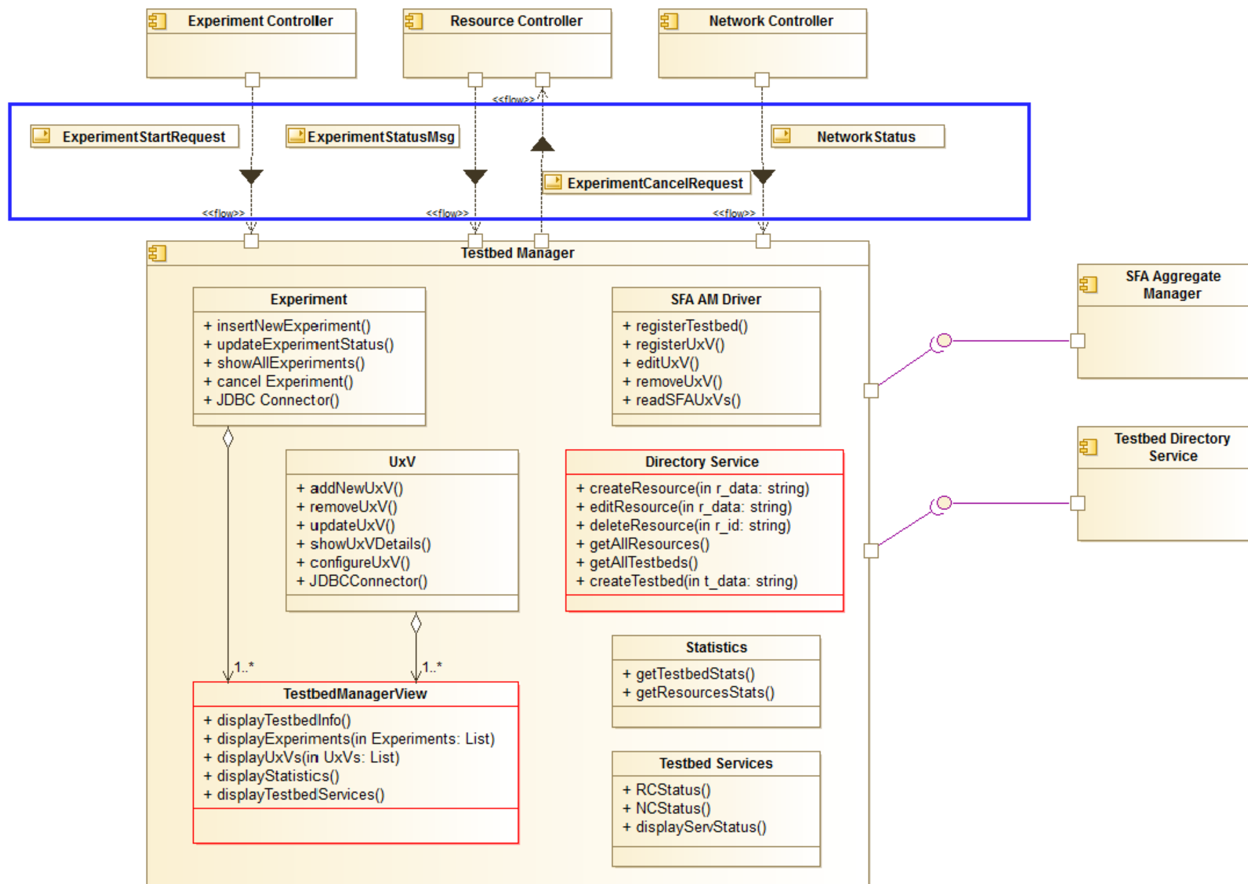

Figure 39: Testbed Manager - Class diagram

Required Interfaces

- Message Bus: Testbed Manager interacts with the following components through the Message Bus:
  - Experiment Controller: Testbed Manager consumes the messages indicating the start of a new experiment
  - Resource Controller: Testbed Manager consumes messages indicating the progress of ongoing experiments
- Local Database Repository (JDBC): Testbed Manager reads and store information in the database deployed at each testbed.
- Testbed Directory Service: Testbed Manager interacts with this component in order to synchronise the content of the Master Data Repository with the content of the local testbed database

4.3.6.3 *Updated sequence diagrams*

The sequence diagram of Figure 40 presents sequence of actions related to experiment handling at testbed level with red color used to highlight the differences from the previous version of the architecture:

1. Testbed Manager receives an ExperimentStartRequest message from platform Message Bus
2. insertNewExperiment() operation is called to write all required information about the new experiment in the local database
3. Testbed Manager receives periodically an ExperimentStatusMsg message about the current status of the experiment from Resource Controller and updates the experiment based on the last information received
4. In case needed an ExperimentCancelRequest message can be transmitted from the user interface of the application which cancels the experiment execution. This message is also consumed from the Resource Controller which is responsible for the safe return of UxV resources participating in the experiment
5. A user can see information about all the experiments that have been executed in the testbed using application's GUI

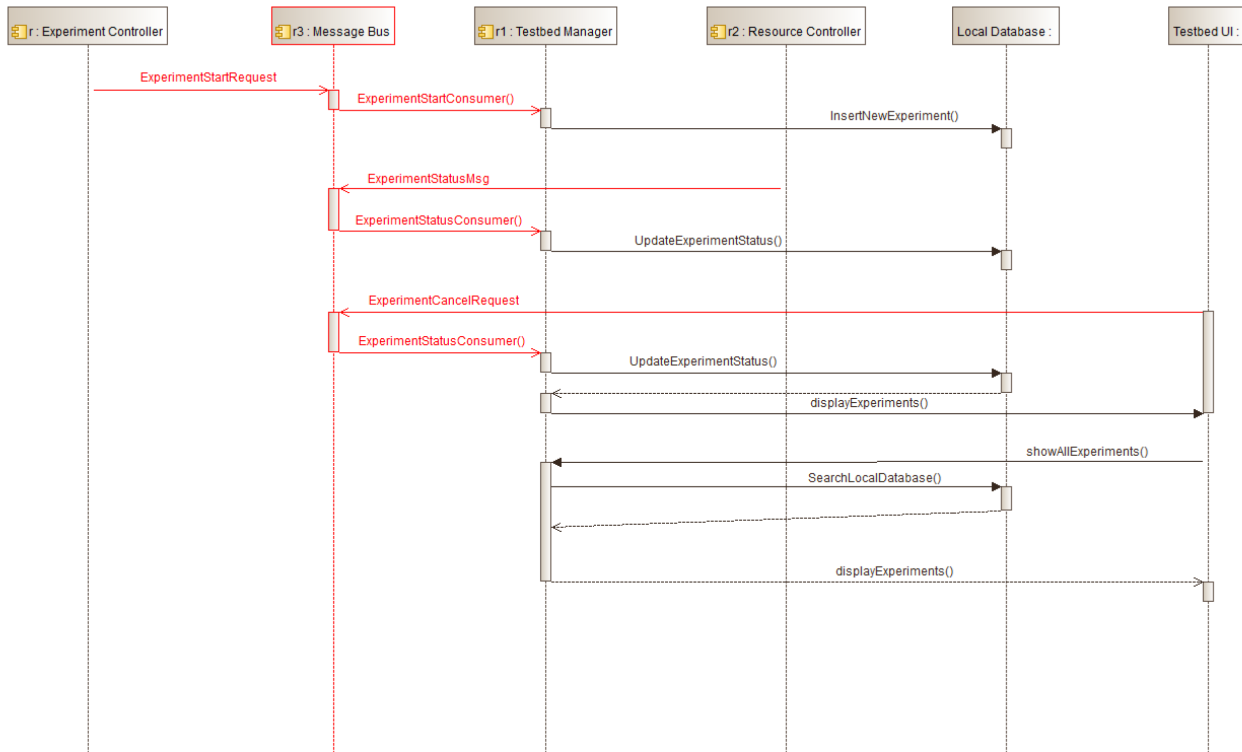

Figure 40: Testbed Manager - Experiment handling sequence diagram

4.3.7 SFA Aggregate Manager

SFA Aggregate Manager (AM) allows testbeds to securely expose their resources to the federation and enables users to reserve them by exchanging XML RSpecs files. An RSpec lists information about the resources (nodes) of each testbed formed in an XML format.

4.3.7.1 *Component requirements as identified in D3.3*

ID (Priority)	**Description**	Requirement Mapping with component's functionalities
TB-AGG-001	SFA Aggregate Manager (SAM) should provide an SFA Interface to comply with SFA based testbeds or testbed federations	It is part of the communication layer using a REST API and a XML-RPC interface
TB-AGG-002	SFA Aggregate Manager (SAM) should provide a REST API to comply with RAWFIE testbeds	A REST API is implemented to support the discovery, reservation, provision and release functionality of RAWFIE resources
TB-AGG-003	SFA Aggregate Manager (SAM) should advertise the resources of a testbed	It is part of the communication layer
TB-AGG-004	SFA Aggregate Manager (SAM) reservation process should comply with the resource reservation process of RAWFIE testbeds	Management layers is responsible for the reservations and the release of RAWFIE resources.
TB-AGG-005	SFA Aggregate Manager (SAM) should provide an interface to testbed administrators for managing RAWFIE testbeds	Supported by REST API (getinfo, create, update and remove methods)

4.3.7.2 *Final specification of functionalities and interfaces*

The SFA Aggregate Manager (SAM) architecture follows design principles which were formed by taking into consideration all the desired features that a manager framework should provide. As depicted in Figure 41, the resulted framework is divided in several fundamental architectural components, each of which possesses significant role in a specific problem area of the facility management; (i) the communication interfaces which facilitate the communication with external actors, followed by the (ii) authentication/authorization context which acts as a security intermediary between the internal system and the outside world; (iii) the management layer where most of the framework's intelligence is accumulated, including the orchestration of the supported functionality to fulfill a requested action and the required database transactions.

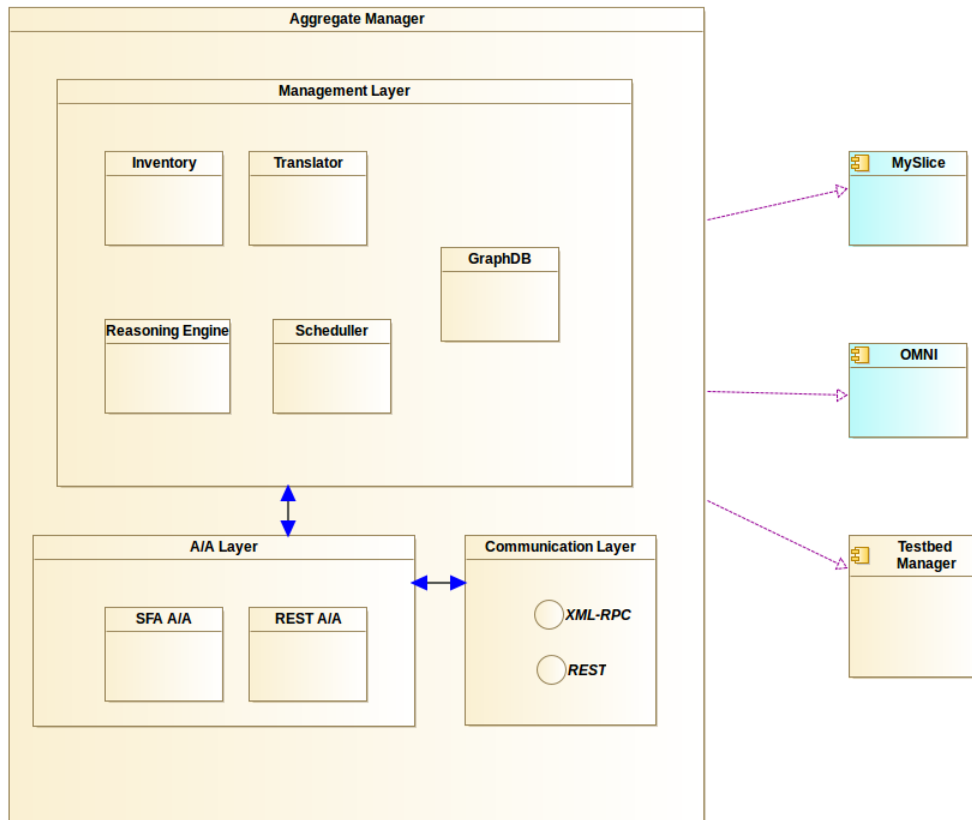

Figure 41: Aggregate Manager architectural components

Communication Layer

One essential characteristic of the presented management framework is its versatility in terms of communication interfaces or else APIs. In our implementation, two major communication interfaces are utilized. Initially the custom REST API is developed to facilitate support of the experiment's lifecycle while leveraging semantically-enriched resource descriptions. Next to that, an XML-RPC API has also been implemented supporting SFA, one of the widely-used protocols in the field of testbeds, thus enabling interoperability with existing testbed management platforms.

(Semantic aware) REST API: The REST API is tailored to support the discovery, reservation, provision and release functionality of RAWFIE resources. It leverages the OMN-based [12] resource descriptions stored in the local Semantic Graph Database, to provide the users/experimenters with semantically enriched information regarding the resources managed by the respective testbed. Thus the users are able to allocate and provision resources that correspond to their experiments' specifications, as well as release these resources when no longer used. Complementary to this functionality, this API exposes the essential administrative management methods; namely, RAWFIE resource description retrieval, creation, update and deletion are supported.

(SFA enabled) XML-RPC API: This API is exposed by the AM in order for the SAMANT platform (details about the SAMANT project and its integration in RAWFIE in the D4.7) to be interoperable with existing SFA [13] enabled provisioning tools (e.g jFED, omni) and to allow

federation with existing testbed management platforms that confront with the SFA and the GENI API v3 specification [14]. Backwards compatibility with GENI AM API v2 is also present, allowing our framework to be reachable not only by tools that are compatible with the latest API, but also by tools that are compatible with v2 AMs. Throughout the XML-RPC API, multiple arguments and returns are labelled as an RSpec18. This resource specification is the primary data structure used within the API and follows a specific set of schemas.

Authentication/Authorization Layer

The authorization and authentication system is of paramount importance to any testbed management framework and the SAMANT platform is no exception. The implemented Authentication/Authorization (A/A) module is where requests invoked in the communication layer are granted approval (or denied respectively), subject to the credentials submitted by the experimenter. Each of the APIs exposed in the communication layer, utilizes its own dedicated mechanism of handling credentials, which can be configured to authenticate the experimenters, thus determining their privileges (authorization).

The aforementioned experimenter privileges are derived through client side X.509 certificates, in all the interfaces, and are assigned to members of testbeds federated with the given testbed ("certificate authorities"). However, these certificates' structure alternates whether they are targeted at the REST or the SFA communication interface, in a way that they contain distinct attributes meant to be handled by interface-specific methods. To illustrate this specificity, the SFA X.509 certificates use an extension in order to provide a uniform resource name (URN), which enables the testbed Aggregate Manager to link an associated SFA request to a specific experiment. Following the privileges extraction, it is then designated whether the user is permitted to (i) Retrieve, (ii) Create, (iii) Modify or (iv) Release a RAWFIE (a) Resource, (b) Account or (c) Reservation. More specifically, alongside each request comes a signed XML file containing the user's privileges, guiding the A/A module to map them with the abovementioned permitted actions.

Management Layer

Translator: In order to support federation with SFA testbeds, which use legacy data formats, it is necessary the API method calls and the respective semantic descriptions to be translated into the respective SFA data models/formats. This component was initially implemented to help developers work with Open-Multinet related ontologies and was included in the OMN suite. During the SAMANT project which was submitted in the 1st Round of Open Calls initiative, it was extended to support the ontology developed specifically to fit the RAWFIE requirements. In particular, this integration of the omnlib Translator with the SAMANT platform provides support for translating locally used GENI RSpecs (structured data models) into RDF-based graphs and back. The main advantage of this approach is the automation and speed up of conversion of data that is not using RDF, while ensuring that the quality of the generated RDF data corresponds to its counterpart data in the original system.

Scheduler: The Scheduler component is the provider of the main functionality of the system, since it is the part where decisions regarding resource reservation take place, based on their availability. More specifically, when a request for booking a resource is received, it is forwarded to the Scheduler component via the Inventory Manager; firstly the Scheduler compares the requested booking's start and expiration time with those of the existing, active reservations. Afterwards, it decides based on possible timeslot conflicts and while taking into account the authorization context, whether to fulfill the request or reject it. A simple First-Come-First-Served (FCFS) policy is applied to the requests for resource reservation. However, with minor

modifications in the Scheduler component, testbeds administrators are able to define their own resource allocation policies (e.g. implementing a user role/status policy).
Inventory Manager: The manager is where all the RAWFIE related policies and resource management is concentrated. Requests regarding resource discovery, booking and reservation, resource provisioning and release (i.e. user tasks), as well as resource description retrieval, creation, update and deletion (i.e. administrative tasks) are forwarded to the Inventory Manager. This component facilitates the orchestration and coordination of the actions required to fulfill the aforementioned requests. These actions include, but are not limited to, forwarding the received GENI RSpec to the Translator and receiving the respective Semantic description (and vice-versa), consulting the Scheduler about the feasibility of booking the requested resources, manipulating proper objects which achieve compliance with the established Data Models and storing/retrieving them to/from the GraphDB triplestore, formulating responses and directing them back to the message bus. Throughout the whole process, the manager constantly addresses the authentication Layer in order to access policy-sensitive content and perform policy-sensitive tasks.
Reasoning Engine: Its main function lies in the deduction of implicit knowledge based on explicit statements stored inside the Triplestore. This extended knowledge may become permanently stored or leveraged to assert miscellaneous requests received.

4.3.7.3 *Updated sequence diagrams*

GetVersion

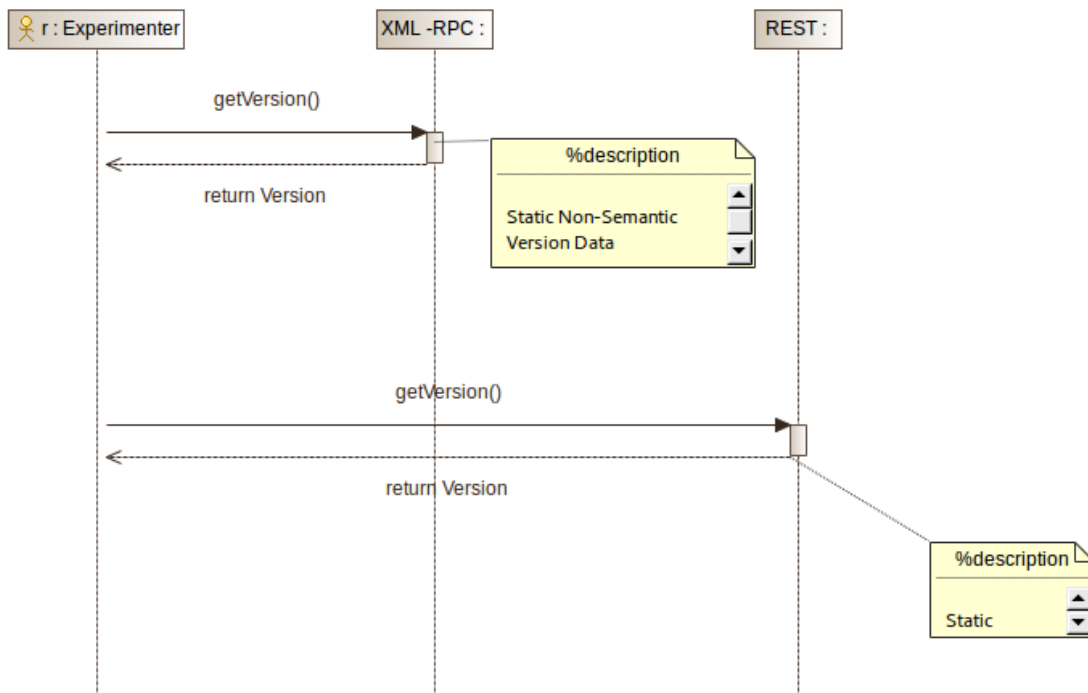

Figure 42: Aggregate Manager - Get SFA-API version sequence diagram

Figure 42 describes the process of retrieving the version of the SFA-API that current implementation of AM supports. The AM provides this functionality through the XML-RPC API where the system responds with an XLM RSPEC reply but also through the REST API where the

system responds with a semantic modelled reply. In both cases the replied version is stored in the API implementation code and no need for further interactions with the back end are necessary. There is no need for user authentication in order to perform this call.

List resources

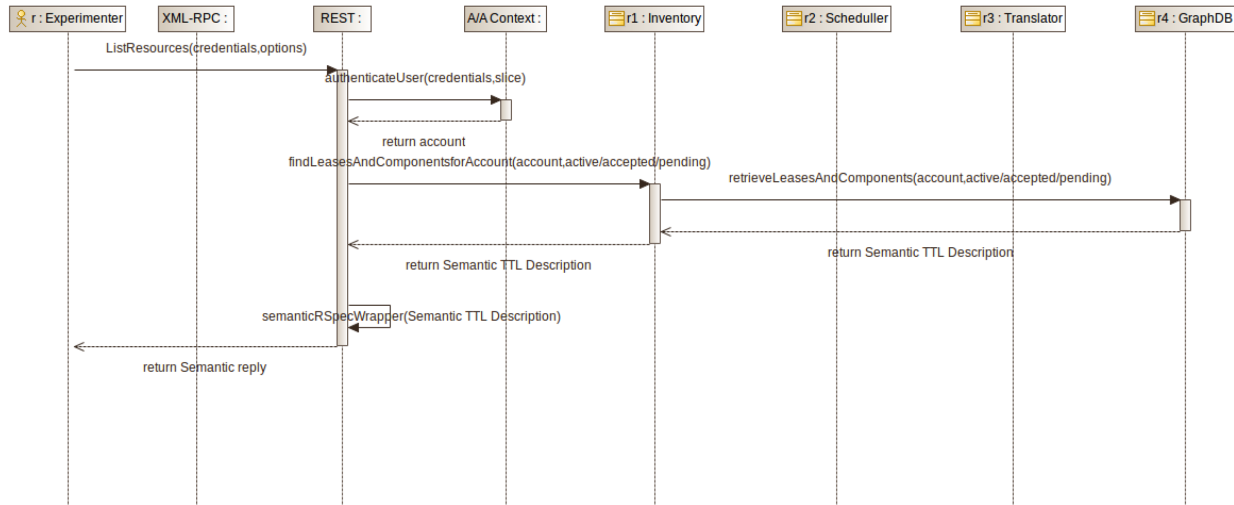

Figure 43: Aggregate Manager - retrieve resources information using REST-API sequence diagram

Figure 43 describes the process where the experimenter retrieves information about the resources of a specific testbed. The experimenter can provide criteria in the request and retrieve resources based on their availability and their allocation status. Regarding the call handled by the REST-API, first certificate based user credentials are validated in order the system to identify if the user has the respective rights to perform the information retrieval action for the specified testbed. If this is the case, the request is then forwards to the Aggregate manager that retrieves the information from the triple-store graph database. The results are then serialised based on JSON for Linking Data (JSON-LD) format and returned to the service consumer.

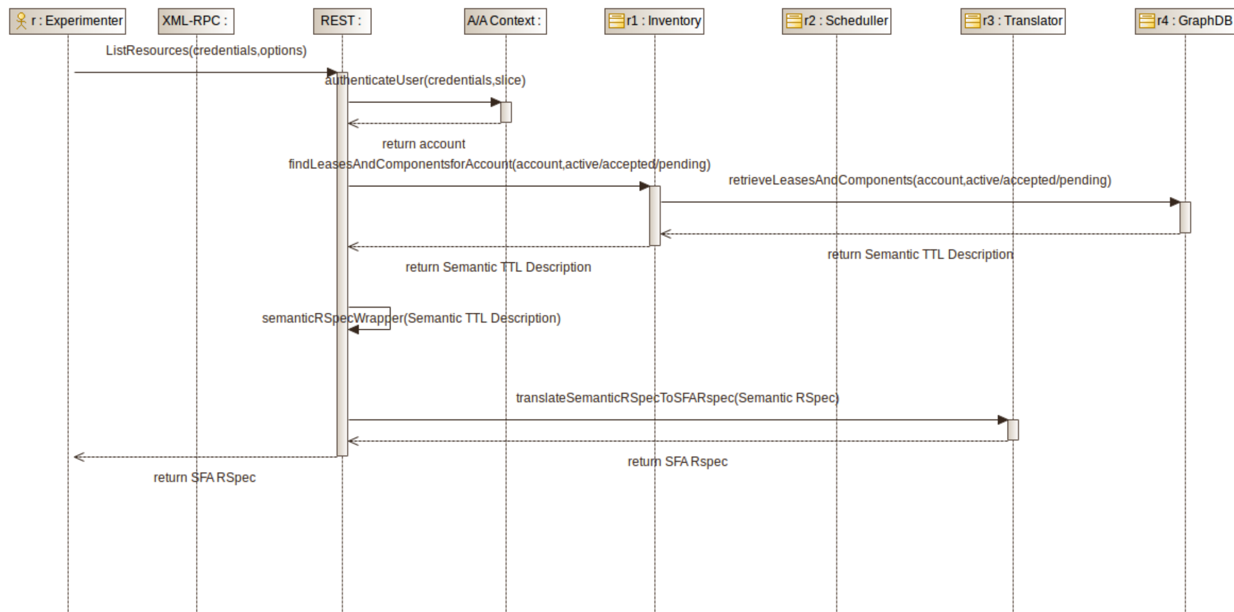

Figure 44: Aggregate Manager - retrieve resources information using XML-RPC API sequence diagram

In case the call is handled by the XML-RPC API (supporting an SFA compatible request) the overall subsequent steps are similar except from the fact that the retrieved semantic information objects are translated to the adequate RSPEC-XML format (Figure 44).

Allocate resources

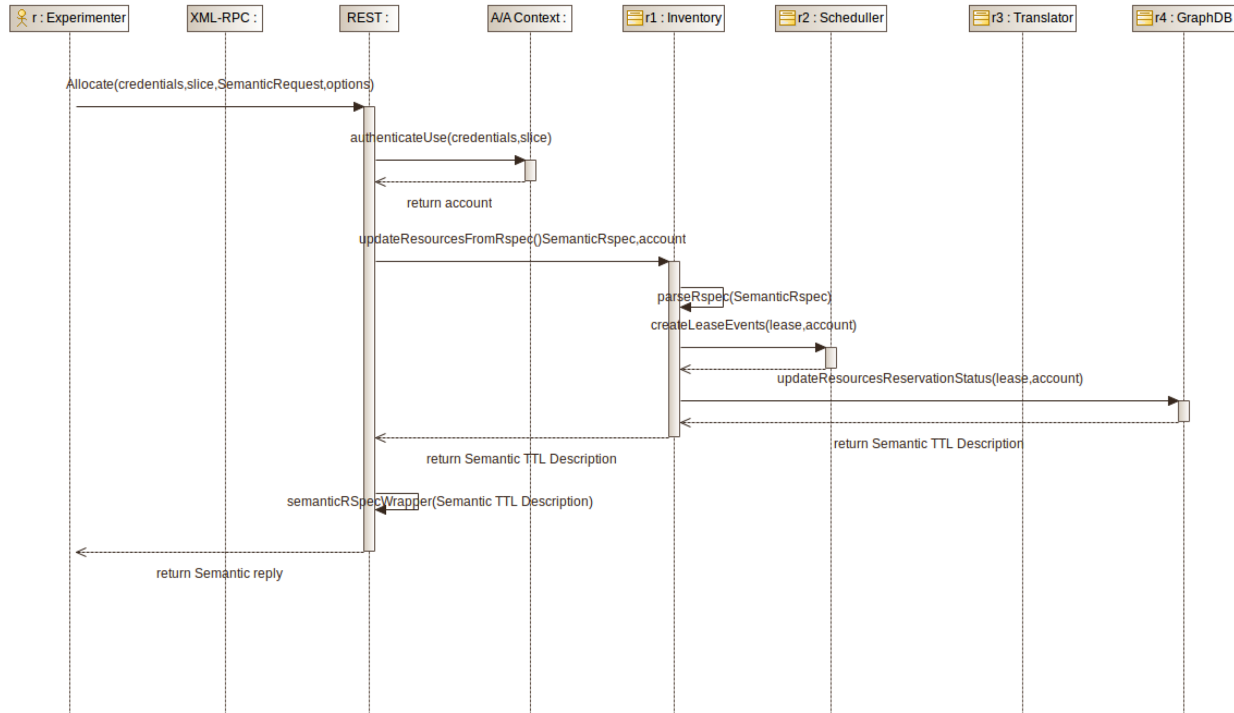

Figure 45: Aggregate Manager - allocation of resources through REST API

Figure 45 describes the process where the experimenter allocates resources. This process is feasible to be performed through the REST-API where the respective information is modelled based on semantics or through the XML-RPC where the overall call is compatible with the SFA standard. In both cases, the first step is to authenticate the service consumer and to validate the respective authorisation rights in order to perform the sliver creation process. Then the Aggregate Manager analyses the semantically described request and initially verifies, through the Scheduler component, that the requested resources are available for the issued time frame. It should be noted that Scheduler loads all active reservations, upon AM initiations, and hence no interaction with the database is necessary. If the resources are available, then the Aggregate manager updates the Graph database with the new reservation and replies to the service consumer with the descriptions of the resources along with the reservation id.

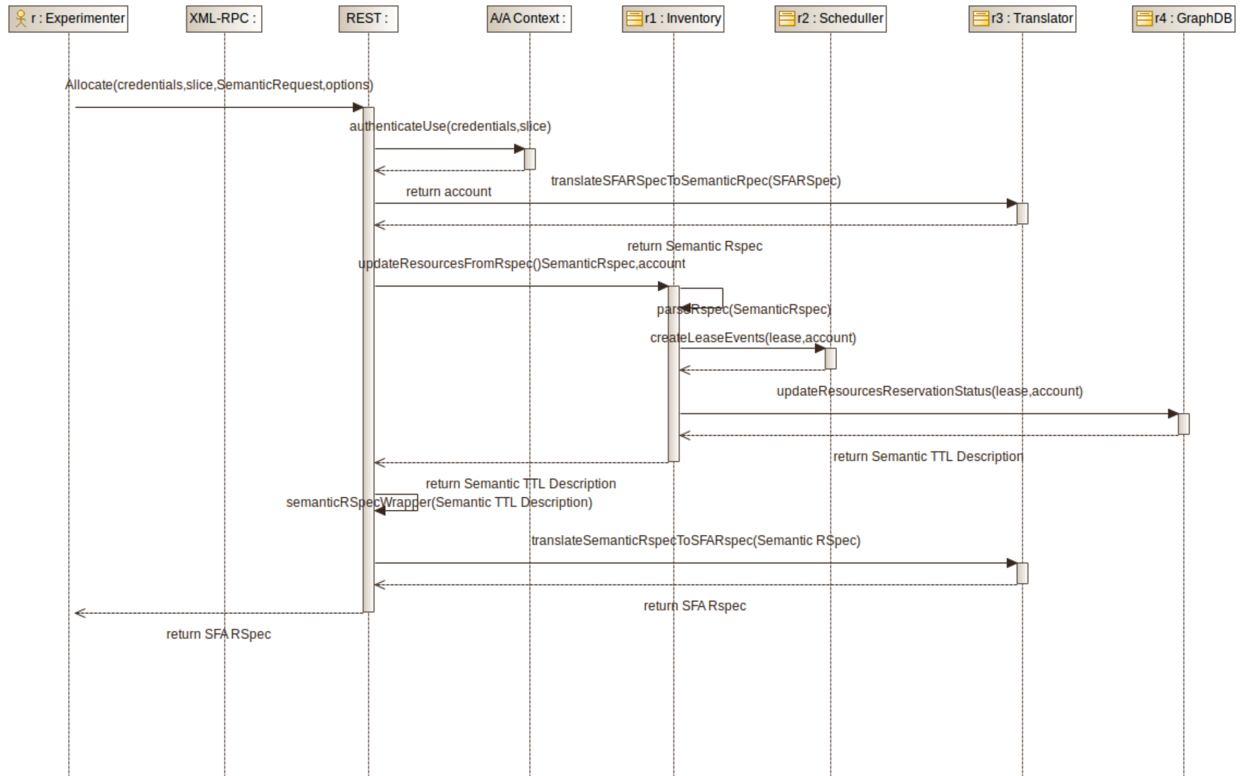

Figure 46: Aggregate Manager - allocation of resources through XML-RPC API

The handling of the request by the XML-RPC API (Figure 46) is similar with the described process but differentiates in two main points. The XML-RSPEC request is first translated to the respective OMN semantic description. Then it is feasible to be handled by the same Aggregate Manager's mechanisms as these described for the REST-API calls. In a similar manner, after a successful reservation request, AMs reply is translated to the respective XML-RPC reservation RSPEC.

Manage resources

Figure 47, Figure 48 and Figure 49 describe the process where the client creates, updates and retrieves data object describing the testbed's resources. This functionality is expected to be utilised by the testbed administrator (not the experimenter). This functionality is only available through the REST-API and all data are modelled based on OMN ontology. The call is escorted with the respective semantic descriptions of the targeted resources. After the processing of the provided certificate and the successful authentication and authorisation process the Aggregate Manager performs the necessary updates to the Graph database. Depending on the type of call (create, update, delete) the respective actions are performed. The system replies with an OMN modelled data object describing the entities that the action performed on.

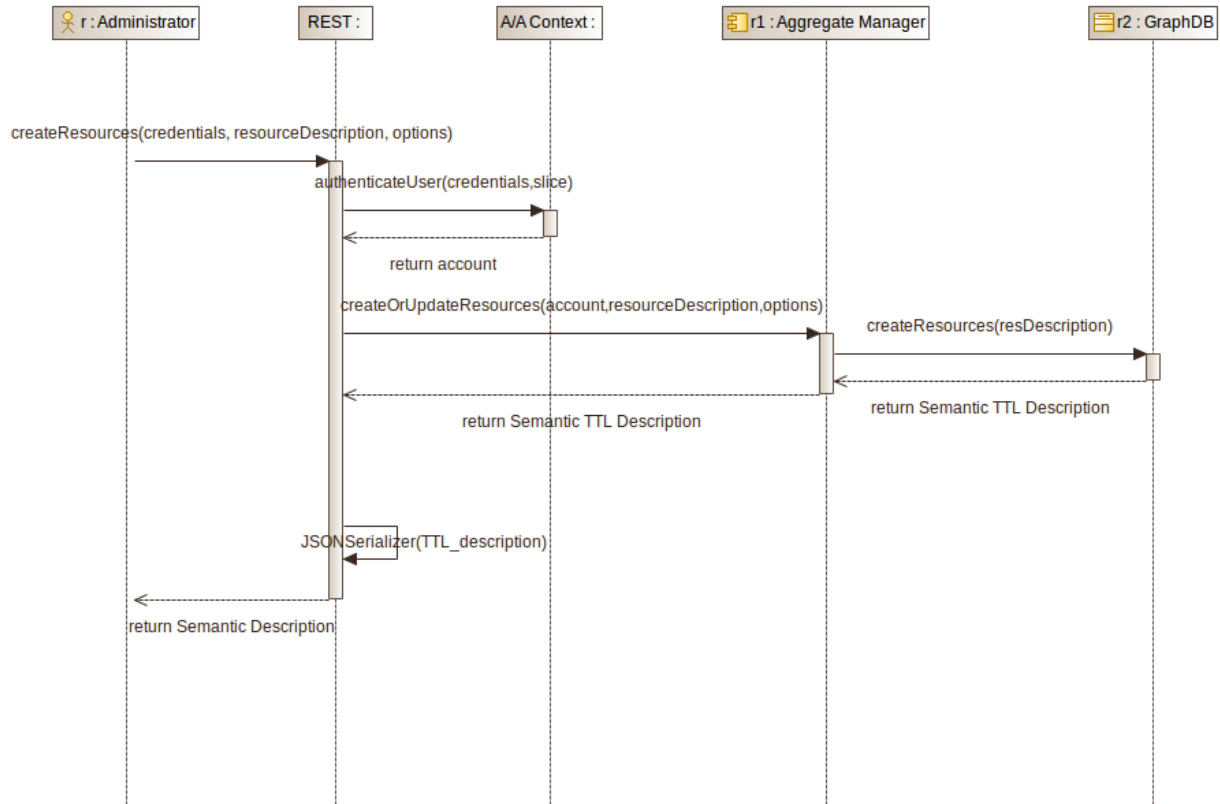

Figure 47: Aggregate Manager - create resource sequence diagram

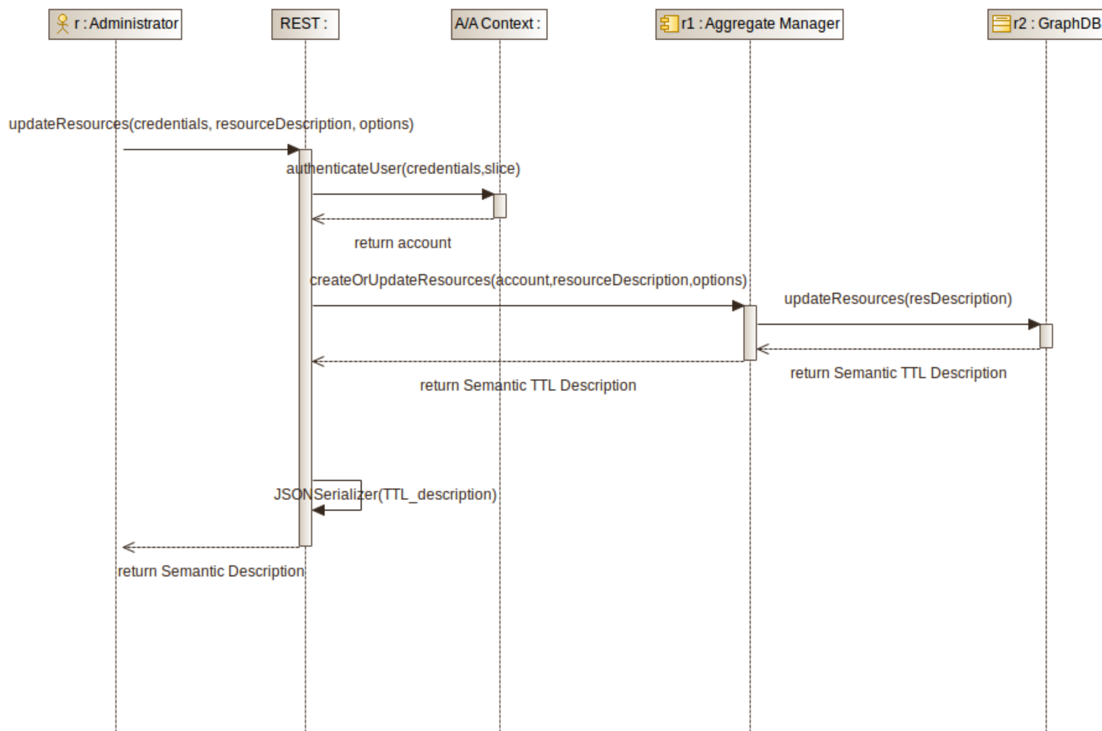

Figure 48: Aggregate Manager - update resource sequence diagram

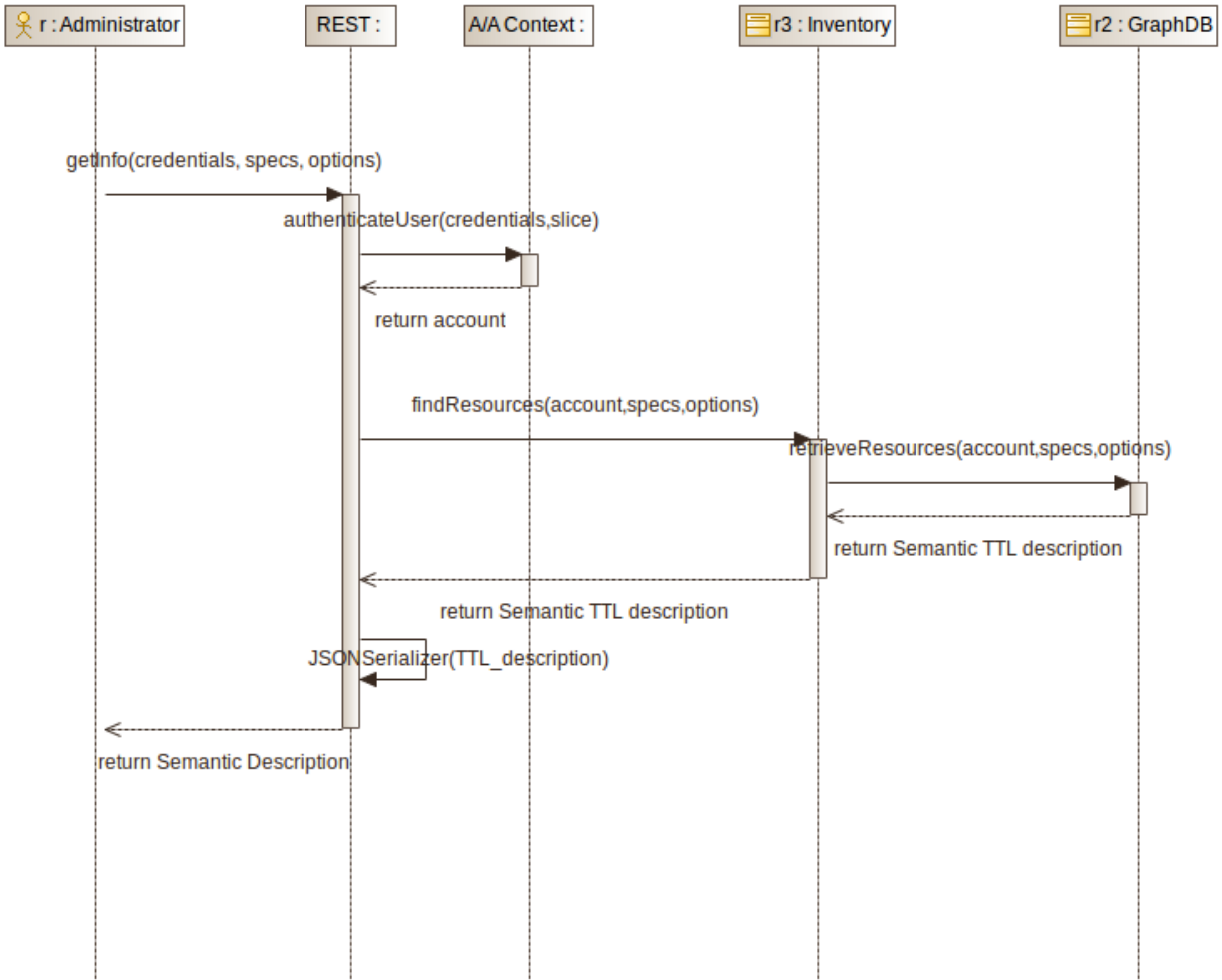

Figure 49: Aggregate Manager - retrieve resource sequence diagram

4.3.8 UxV Node

The UxV Node is a mobile system that interacts with the other Testbed entities (proxy, other UxV's). It can be remotely controlled or able to act and move autonomously, as programmed before the start-up of the experiment or as programmed during the execution of the experiment, e.g. in real-time. A UxV node interacts with the other RAWFIE components through the Message Bus, using the RAWFIE protocol defined in the remainder of the section, which conveys message according the AVRO structures (see D4.5). Examples of the implementation on two different platforms are given at the end of this section.
The basic requirements a generic UxVs Node should satisfy, in order to make it possible to use it within the RAWFIE platform (to "plug-in" within an existing Testbed, for example), have been identified by the consortium partners and are listed in Table 4.
For more detailed information on the UxV Node component specifications in RAWFIE, further to the updates provided in this and in the following subsection, please refer to section 4.3.8, 4.3.9 and 4.3.10 of D4.5.

ID (Priority)	Description	Requirement Mapping with component's functionalities
UXV-GEN-001 (HIGH)	Compliance of UxV to RAWFIE specification and interfaces	Two distinct components implement a translator between different UxV protocols and the RAWFIE UxV message format
UXV-GEN-002 (LOW)	UxV providers may provide for their supplied devices a simulator/emulator mimicking its real-world behavior and kinematics	Two simulation engines were deployed mimicking the behavior of ROBOTNIK and MST UxVs
UXV-NOD-001 (HIGH)	Each UxV shall have a unique Identification code.	Each UxV was assigned a unique identification code and canonical name. These identification tokens are used to communicate with the message bus
UXV-NOD-002 (HIGH)	Each UxV node should ensure a minimum autonomy of 15-30 minutes.	ROBOTNIK and MST UxVs have an autonomy greater than 4 hours
UXV-NOD-003 (HIGH)	Each UxV node should ensure payload.	ROBOTNIK and MST UxVs are equipped with several payload sensors
UXV-NOD-004 (MEDIUM)	Each UxV node may register the Coordination Reference System CRS it is expected to operate.	ROBOTNIK UxVs use a relative Cartesian coordinate system and MST UxVs use WGS 84 coordinates
UXV-NOD-005 (HIGH)	A proper message communication protocol should be defined for the communication between a UxV node and the testbed ground components	The RAWFIE UxV Message Protocol addresses this requirement
UXV-NOD-006 (HIGH)	All command messages received by the UxVs should be ensured that they originate from an authorized testbed component or other UxV involved in an experiment before being processed	This requirement is not currently addressed by any UxV component
UXV-INT-001 (HIGH)	All messages of the UxV Message API should contain in their header basic information about the dispatching entity.	The RAWFIE UxV Message Protocol addresses this requirement
UXV-INT-	UxV should support the	ROBOTNIK and MST protocol translators

002 (HIGH)	Goto command	address this requirement
UXV-INT-003 (MEDIUM)	UxV should support the KeepStation command	ROBOTNIK and MST protocol translators address this requirement
UXV-INT-004 (HIGH)	UxV should support the Abort command	ROBOTNIK and MST protocol translators address this requirement
UXV-INT-005 (HIGH)	UxVs should be able to advertise themselves to the RAWFIE infrastructure	ROBOTNIK and MST protocol translators address this requirement
UXV-INT-006 (HIGH)	UxVs should be able to advertise information about their sensors to the RAWFIE infrastructure	ROBOTNIK and MST protocol translators address this requirement
UXV-INT-007 (MEDIUM)	UxVs should be able to inform testbed about their CPU usage	ROBOTNIK and MST protocol translators address this requirement
UXV-INT-008 (HIGH)	UxVs should be able to inform testbed about their on-board storage	ROBOTNIK and MST protocol translators address this requirement
UXV-INT-009 (HIGH)	UxVs should be able to inform testbed about their fuel storage	ROBOTNIK and MST protocol translators address this requirement
UXV-INT-010 (HIGH)	UxVs should be able to inform testbed about their orientation (attitude)	ROBOTNIK and MST protocol translators address this requirement
UXV-INT-011 (MEDIUM)	UxVs should be able to inform testbed about their velocity and acceleration	ROBOTNIK and MST protocol translators address this requirement
UXV-INT-012 (HIGH)	UxVs shall periodically publish a digest of their scalar sensor readings	ROBOTNIK and MST protocol translators address this requirement
UXV-PRX-001 (HIGH)	Embedded UxV proximity component shall be into the UxV	MST UxVs are equipped with the UxV Proximity Component
UXV-NET-001 (MEDIUM)	Capability of taking the control of the UxVs from distance.	ROBOTNIK and MST protocol translators address this requirement
UXV-NET-002 (MEDIUM)	UxVs should be able to Synchronize their Time-References between them.	MST UxV clocks are synchronized and disciplined using GPS receivers
UXV-NET-004 (HIGH)	Each UxV node shall be equipped with primary and secondary communication means.	MST UxVs are equipped with Wi-Fi (primary communication mean) and GSM (secondary communication mean)

UXV-NET-005 (MEDIUM)	UxV network interface management	
UXV-NET-006 (MEDIUM)	UxV communication interoperability with RAWFIE (incoming)	ROBOTNIK and MST protocol translators address this requirement
UXV-NET-007 (MEDIUM)	UxV communication interoperability with RAWFIE (outgoing)	ROBOTNIK and MST protocol translators address this requirement
UXV-NET-008 (MEDIUM)	Neighboring UxV monitoring	ROBOTNIK and MST protocol translators address this requirement
UXV-NET-009 (HIGH)	Each UxV node should be able to send navigation state feedback with at least 2 Hz frequency and maximum 1 sec latency when within radio communication reach.	ROBOTNIK and MST protocol translators address this requirement
UXV-NET-010 (HIGH)	The primary communication channel of the node should support IPv4/IPv6 protocol stack.	ROBOTNIK and MST UxVs and protocol translators are currently using the IPv4 protocol stack
UXV-SEN-002 (HIGH)	Each UxV node shall be able to list the available sensors	ROBOTNIK and MST protocol translators and the RAWFIE UxV Message Protocol address this requirement
UXV-SEN-003 (HIGH)	UxV location and sensor data should be made available to the experimenter	ROBOTNIK and MST protocol translators and the RAWFIE UxV Message Protocol address this requirement
UXV-SEN-004 (HIGH)	Location sensors should be supported in each UxV unit and can be used remotely during testbed demonstrations.	ROBOTNIK and MST protocol translators and the RAWFIE UxV Message Protocol address this requirement
UXV-SEN-005 (HIGH)	UxVs should sent a notification to the Resource Controller when they reach the desired location	ROBOTNIK and MST protocol translators and the RAWFIE UxV Message Protocol address this requirement
UXV-STO-001 (HIGH)	UxVs shall be able to store data on board.	Persistent on-board storage is provided my all UxVs
UXV-STO-002 (HIGH)	UxVs shall provide a management tool of the available storage.	ROBOTNIK and MST provide software tools to manage UxV storage

UXV-STO-003 (HIGH)	UxVs shall provide an authorized access to the data management tool.	
UXV-STO-004 (HIGH)	UxVs shall provide a data log.	Logging to persistent storage is implemented in all UxVs
UXV-STO-005 (MEDIUM)	UxVs may provide an automated syncing of servers.	
UXV-PRC-001 (HIGH)	Each UxV shall be able to operate autonomously.	ROBOTNIK and MST UxVs are equipped with on-board autonomy
UXV-PRC-002 (MEDIUM)	The UxV should provide collision avoidance mechanism.	
UXV-PRC-003 (MEDIUM)	Capability of task planning of the UxVs nodes during run-time.	ROBOTNIK and MST protocol translators and the RAWFIE UxV Message Protocol address this requirement
UXV-PRC-004 (MEDIUM)	UxVs should be able to cooperate during the execution of an experiment.	
UXV-PRC-005 (HIGH)	Each UxV node shall be able to keep position while waiting for new instructions	ROBOTNIK and MST protocol translators and the RAWFIE UxV Message Protocol address this requirement
UXV-PRC-006 (MEDIUM)	UxVs shall be capable of processing sensor data in order to summarize large sensor data-sets.	The DigestLogging task of MST UxVs addresses this requirement
UXV-MGT-001 (HIGH)	UxVs shall offer on demand resources (Network, Sensor, Processing, and Controller).	
UXV-MGT-002 (HIGH)	UxV shall be capable to revert to a safe mode	ROBOTNIK and MST protocol translators and the RAWFIE UxV Message Protocol address this requirement
UXV-MGT-003 (HIGH)	UxV shall be capable to restart its internal components independently	ROBOTNIK and MST on-board software are capable of fulfilling this requirement
UXV-MGT-004 (HIGH)	UxV shall be capable to monitor the health of its components and provide appropriate health status messages to the testbed	ROBOTNIK and MST on-board software are capable of fulfilling this requirement

UXV-MGT-005 (HIGH)	UxV shall be capable to enable/disable certain internal components	ROBOTNIK and MST on-board software are capable of fulfilling this requirement
UXV-MGT-006 (HIGH)	UxV shall be capable to offer safe maintenance access for manufacturers	ROBOTNIK and MST UxVs are capable of fulfilling this requirement

Table 4: List of requirements for an UxV node to be used in RAWFIE

4.3.8.1 The RAWFIE UxV Protocol

The RAWFIE UxV Protocol was devised to abstract the differences between UxVs and expose a simple, compact, extensible, and expressive interface to monitor and control UxVs in a platform-agnostic way. New UxVs can therefore be added to the RAWFIE infrastructure by creating adapters or translators to convert UxV specific information to the RAWFIE UxV Protocol. Detailed information about the UxV protocol and different type of messages are reported in D4.5. In the following we report the structure of the SensorPublishControl message, which was not presented in the former deliverable D4.5.

- **Sensor Publish Control**

This message enables/disables publishing of specific sensor data to the message bus.

Field	Units	Description
Module	-	Canonical name of the controlled module
Quantities	-	List of quantities
Enabled	-	True to enable publishing, false otherwise

5 Global Sequence diagrams showing main RAWFIE processes

In the following sub-sections, updated sequence diagrams for some of the most relevant RAWFIE use cases, involving different software components of the RAWFIE platform, are depicted and explained.

5.1 Registration of Testbed Resources

The sequence diagram in Figure 50 shows a sample information flow with the components involved in registration of a new resource in a specific, already registered, Testbed. The registration of a new resource is considered as a triple storage process that starts from the Testbed Manager application running at testbed level and targets the master repository located in the appropriate server and SFA and local repositories located within each testbed itself.

1. Testbed Administrator opens the Testbed Manager application and enters his login credentials
2. After successful login Testbed Administrator selects the resources screen while all testbed resources and their capabilities are displayed
3. Testbed Administrator presses the New button and fills all the parameters that describe the resource

4. When the Save button is clicked Testbed Directory Service is conducted (createResource REST method) and the new resource information is stored in the Master Data Repository. Testbed Directory Service should ensure that the call is initiated by a properly authorized user with Testbed Administrator role
5. Using Aggregate Manager's SFA REST API the information for the new resource is stored in the triple-store database for SFA compliant resources located in the testbed
6. The new resource information is stored in the local repository used by Testbed Manager
7. A message about the success of the whole operation is returned to the user, In case of failure in any intermediate step all appropriate actions are performed to undo the new insertion
8. Steps about editing or deleting the resource follow the same procedure (not shown in the diagram)

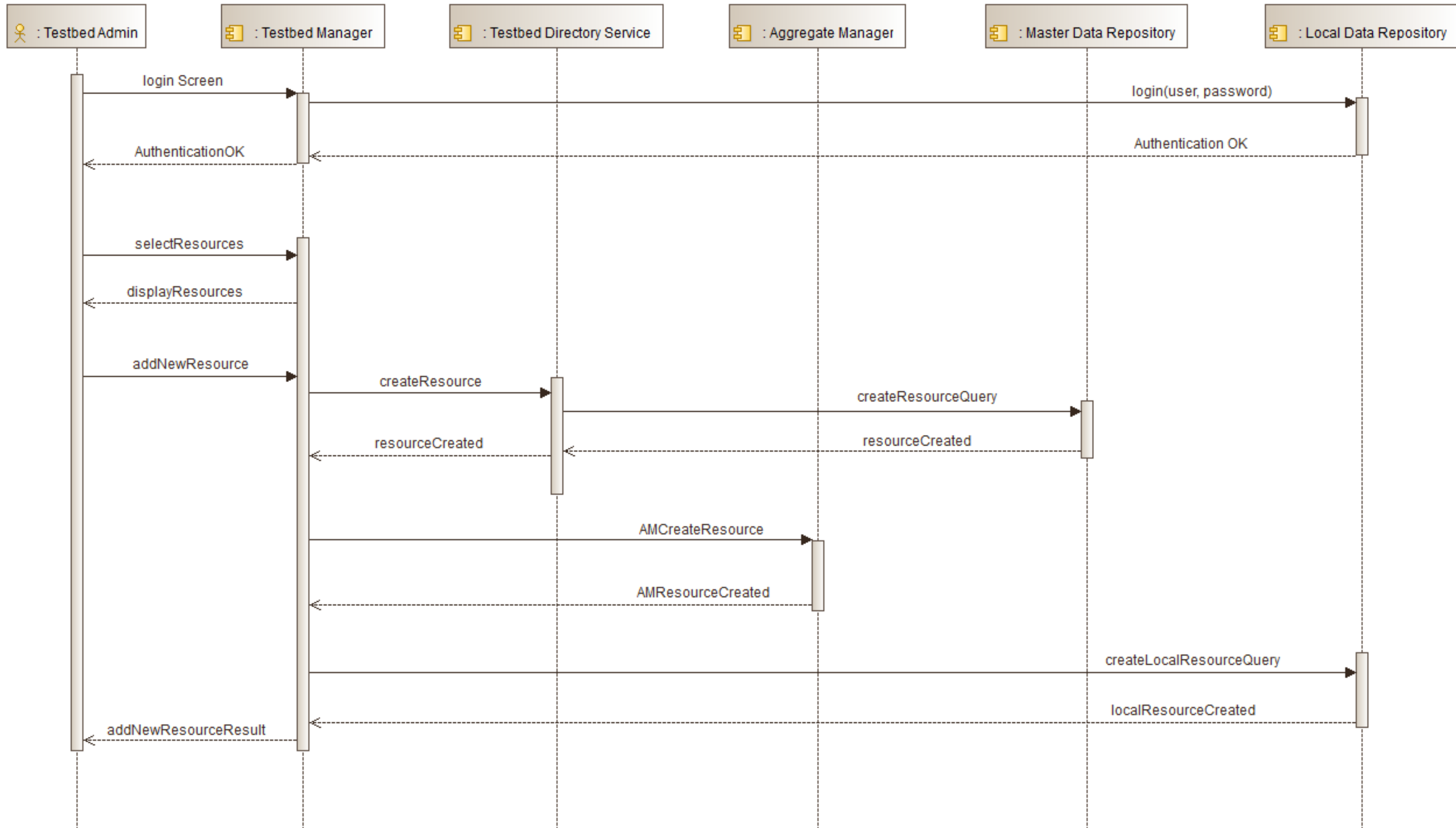

Figure 50: Sequence Diagram of "Registration of Testbed Resources" proces

5.2 Booking Testbed Resources

The sequence diagram below depicts the sequence of actions involved in a successful booking resource request. Booking resource request is implemented as a two step process requiring a confirmation form a testbed authority:

1. The Experimenter loads the CalendarView page (initial page of Booking tool)
2. The Experimenter selects a specific datetime on the CalendarView and defines the desirable time interval for booking resources (timeslot selection)
3. CreateBooking page is loaded showing the available resources for the selected period (the resource information is retrieved from the Master Data Repository indirectly via the Testbed Directory Service API)
4. Experimenter selects UxV resources and submits the booking request (addBooking() method is called on the Booking Service)
5. Allocate() is called to Aggregate Manager to ensure that reservation is also accepted by SFA. If accepted then we proceed with the steps below otherwise the process is terminated and appropriate feedback is returned to the experimenter
6. After performing the necessary authorization checks Booking Service performs its internal actions and if successful, the booking request is persisted in the database with status PENDING
7. Appropriate email notifications are sent to both the experimenter initiating the booking as well as to the testbed operator responsible for approving it
8. Following the reception of booking notification an authorized Testbed Operator loads the ApproveBooking page showing all pending booking requests
9. The Testbed Operator calls Booking Service approveBooking() method which after checking for the proper authorization performs all the internal logic required for confirming the experimenter's request
10. If no conflict or any other problem occurs, the booking request is CONFIRMED
11. Following the confirmation a BookingStatusMsg is sent to the Message Bus informing any interesting consumer component, that the booking request has changed status
12. Both Experimenter and Testbed Operator are informed by appropriate email notifications

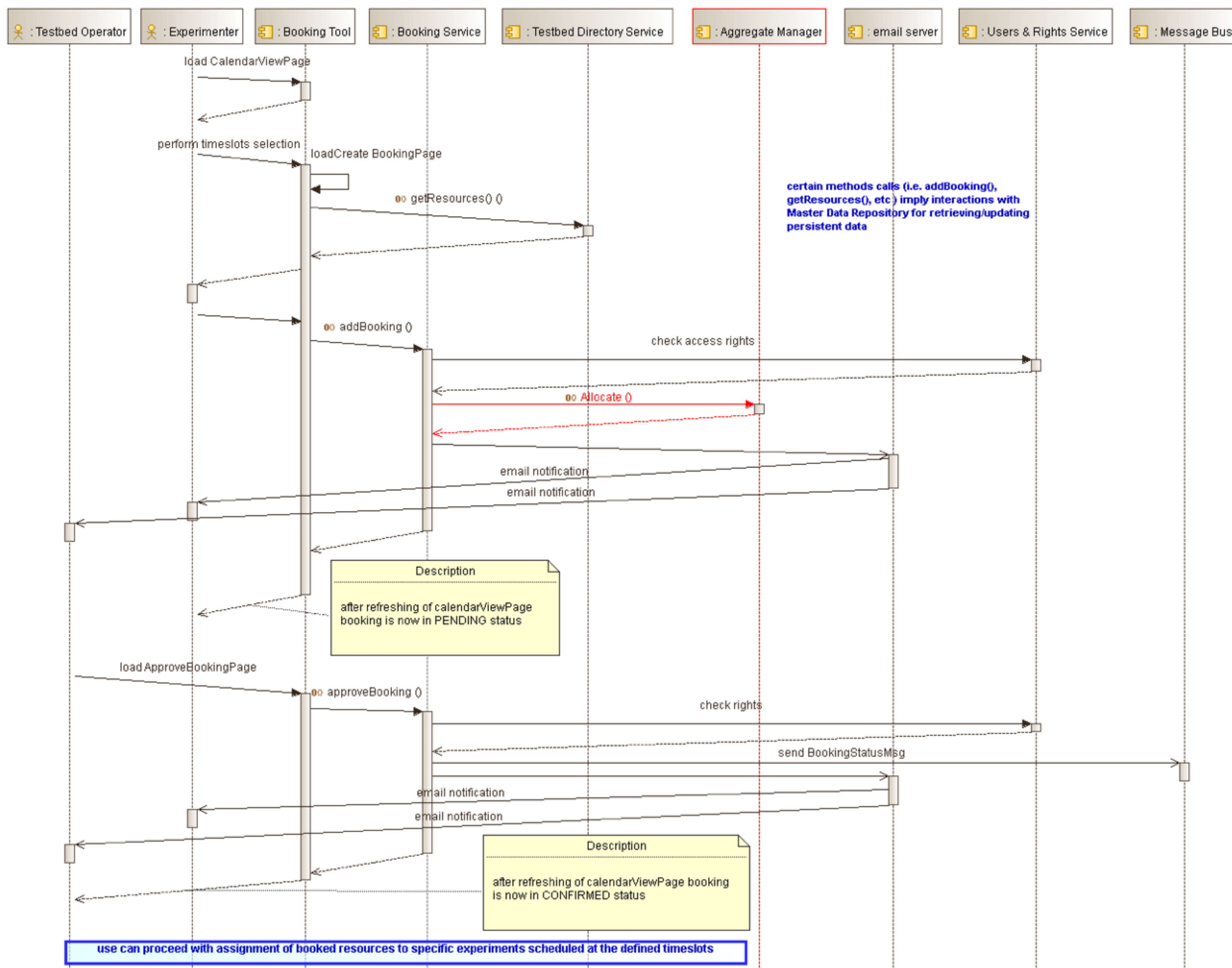

Figure 51: Sequence Diagram for “Booking Testbed Resources” process

5.3 System Monitoring

No updates (refer to D4.5 for details).

5.4 Experiment Execution and Monitoring

The sequence diagram in Figure 52 shows a sample scenario where the Experimenter opens a running experiment, gives directions, observes the results and visualises the scenario. Such scenarios could be “environmental monitoring of water canals”, “border surveillance or perimeter protection of large areas” and many others, as described in D3.1.

1. Prerequisites: the Experimenter is already logged in the system, an experiment is already running and the Experimenter and has the right to observe the experiment
2. The Experimenter starts the Experiment Monitoring Tool in order to view the experiments that (s)he booked and are available to her(him), then selects an already running experiment from the list
3. The Experimenter opens the visualisation page
4. The Visualisation Engine gets the request from the Experimenter and retrieves the available experiments for that user. The list of available experiments is presented. The experimenter chooses the experiment and starts the visualisation
5. The Visualisation Engine gets the request for the chosen experiment and subscribes to the topics that contain information about that experiment, the sensors and the UxVs that take part in the experiment
6. When the UxV sends new information from its sensors – like movement, sensors’ data, warnings, errors etc., then these messages are received by the Visualisation Engine through the Message Bus
7. The Visualisation Engine updates and converts these messages to the proper format and sends them to the Visualisation Tool, which presents them on the map and in the widgets

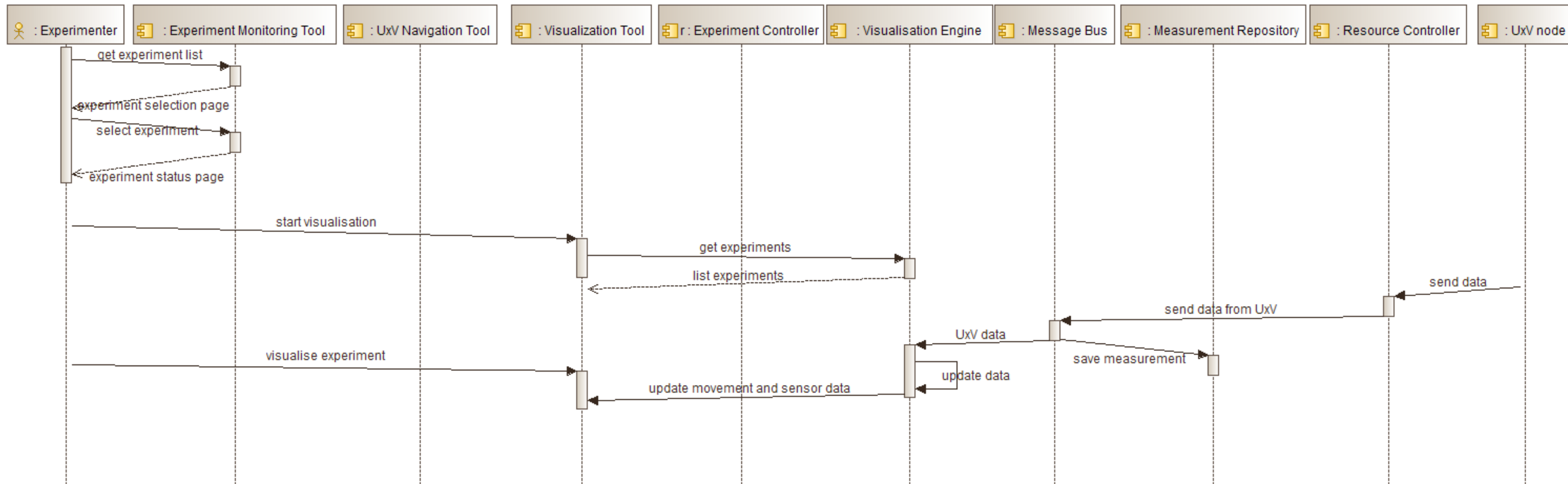

Figure 52: Sequence Diagram for “Experiment Execution and Monitoring” process

5.5 Experiment Measurements Recording

The sequence diagram in Figure 53 shows the Figure 53information flow with the components involved in the resources (UxVs) control, and in sensors measurements acquisition and storing. The following are the sequence of actions executed by the involved components:

1. The Experiment Controller, upon reception of the experiment's instructions from the Launching Service (not indicated in the picture), publishes the instructions on the dedicated Message Bus topics. Examples of instructions for experiment execution include, but are not limited to, the indication of a particular path for each given resource (UxVs), and the sensors that need to be activated for the experiment
2. The instructions are then consumed, from the same Message Bus topics, by the Resource Controller on the Testbed side
3. The Resource Controller will, in turn, publish the commands for the UxVs to the specific Message Bus topics: first the sensors activation commands are published, which are in turn consumed by the involved UxVs
4. Then the Resource Controller publishes "Next Position" commands for the UxVs, after elaborating the instructions received from the Experiment Controller and the position updates (feedback) published by the UxVs themselves, and always communicated through the Message Bus. This is a continuous, closed loop process, so that the Resource Controller may keep control of the UxVs movements (path, waypoints, and so on)
5. Together with the position updates, the UxVs also publishes all other expected sensors' measurements
6. Sensors' measurements are continuously consumed, besides the Resource Controller, by the Measurements Backend Service component (Kafka HBase Connector), which is in charge of ensuring the persistence of the same data within the selected HBase tables in the NoSQL Measurements Repository
   a. At runtime, other RAWFIE components (such as the Visualisation Engine in this example) may directly access the Measurements Repository data.

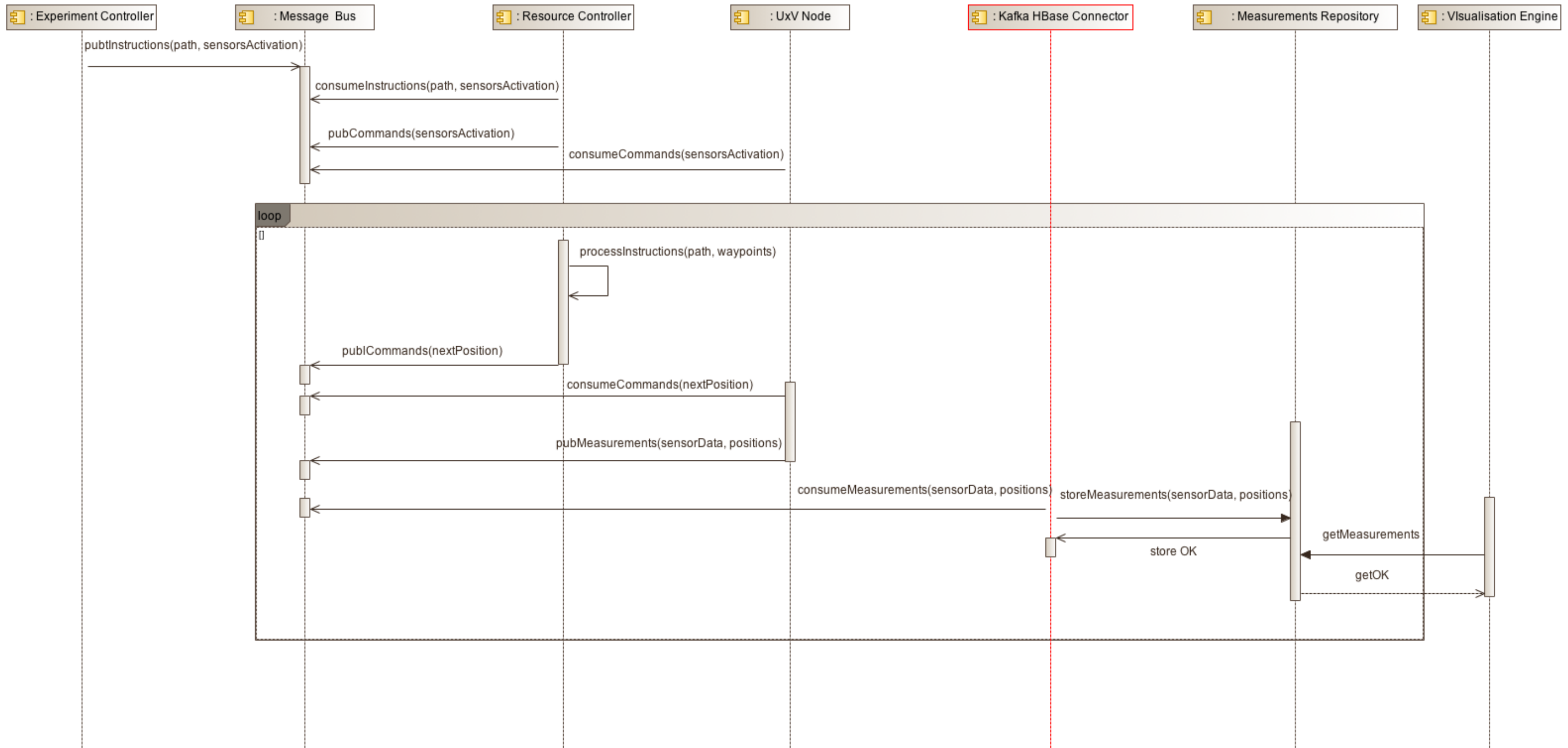

Figure 53: Sequence Diagram for “Experiment Measurements Recording” process

5.6 Authoring and Launching of an Experiment

The sequence diagram in Figure 54 shows a sample flow for authoring and launching an experiment. The adopted steps are as follows:

1. The Experiment gains access to the Authoring tool.
2. The Experiment defines the experiment and gives commands to the tool.
3. The Authoring tool performs a continuous validation process by communicating with the Compiler and Validation Tool.
4. The Compiler and Validation service communicates with the core validation service and returns the results to the Authoring tool.
5. The Authoring returns the retrieved messages to the Experimenter.
6. The Experimenter, after the definition of the experiment, produces the required files to be adopted by the remaining parts of the architecture.
7. The Authoring tool invokes the Compiler and Validation Tool and, accordingly, the required files are stored to the data repository.
8. Finally, the Experimenter selects to launch an experiment by choosing its ID (for details on the launching procedure please refer to sequence diagrams related to manual and/or schedule launching in section 4.2.7).
9. The Launching Service sends the required message to the Experiment Controller that undertakes the responsibility to control the execution process of the experiment.
10. The Experiment Controller sends the appropriate commands to the Resource Controller and, accordingly, the commands are transferred, by applying the necessary modifications to the UxV nodes.
11. A continuous communication between UxV nodes and Resource / Experiment Controller is held until the end of the execution

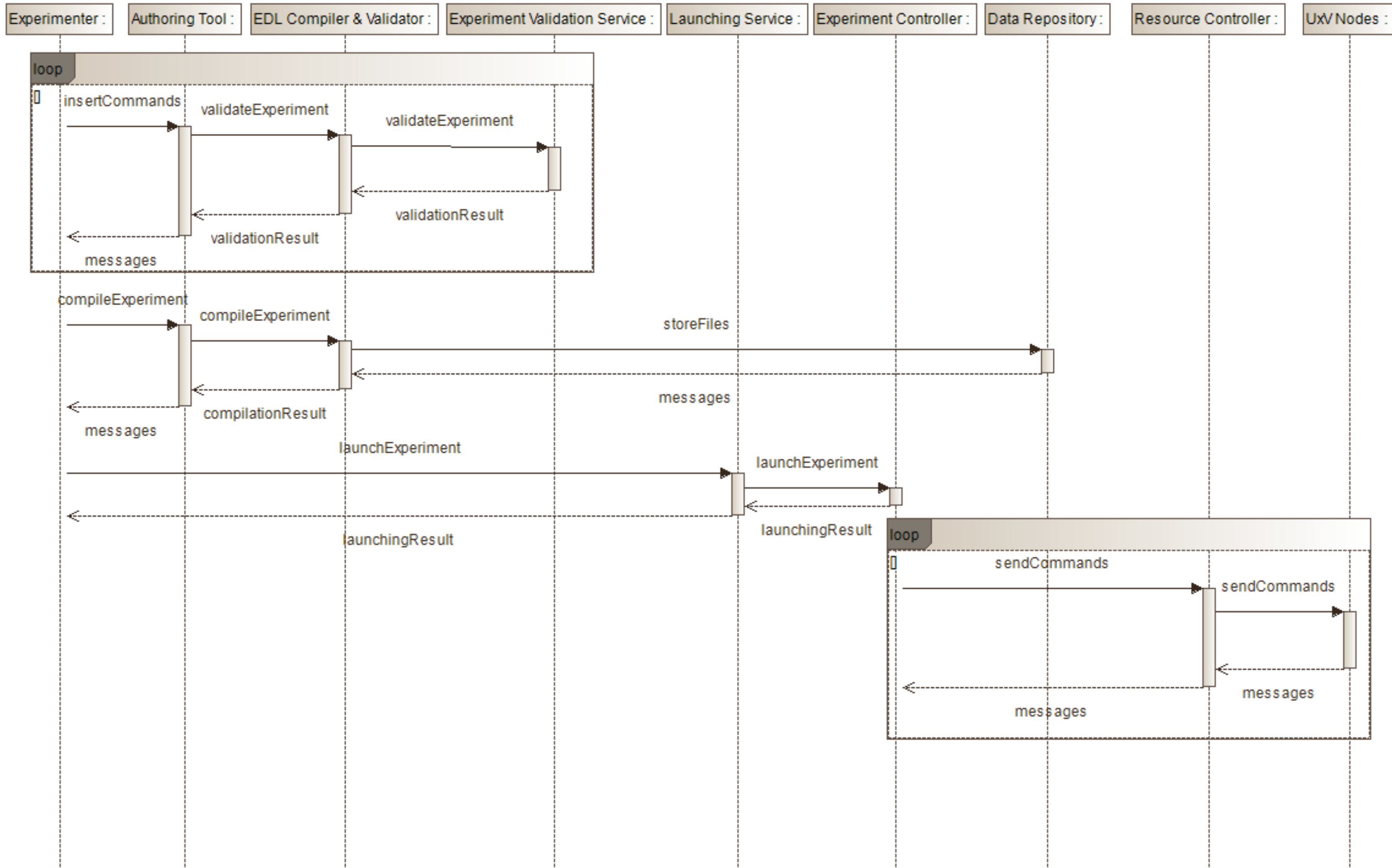

Figure 54: Sequence diagram for “Authoring and Launching of an Experiment”

5.7 Data Analysis

The sequence diagrams depicted in the following figures illustrate two distinct families of data analysis tasks which are namely the data analysis tasks performed on data streams, the streaming tasks, and the data analysis tasks performed on fixed non-time-dependent data structures, the batch tasks. Those two types both involve the Data Analysis Engine, the Data Analysis Tool, the Message Bus and the Analysis Result repository. Additionally, batch tasks are performed on data coming from the Measurement Repository whereas streaming tasks are performed on real time data coming directly from the Message Bus. In other words, the difference between the two types relies on what kind of data the task is performed on, streams or batches. If they are not interrupted, by the occurrence of an error or stopped manually by the user, streaming tasks run indefinitely. Batch tasks however end when the data structure has been covered (the number of time an algorithm pass through the fixed-size dataset is a user-defined parameter, generally called epoch). The design of both types of tasks is done in a user-initialized Zeppelin notebook, GUI accessible through the Rawfie Web Portal. The user can either create this notebook from the Zeppelin GUI or use the Schema Registry GUI to preliminarily select the topic and fields present in the Message Bus that the user wants to analyse. These GUIs are part of the Data Analysis Tool. By selecting the desired topic and fields, the user is offered the possibility to create a Zeppelin notebook, still from the Schema Registry GUI. Once selected, the user is re-directed to a freshly-created notebook in the Zeppelin GUI, populated with the topic and fields the user just selected in the previous GUI. Zeppelin is shipped with methods and functions that enable the user to easily grab data from the Message Bus via a stream abstraction provided by the Data Analysis Engine (Spark) it is built upon, and to easily pull batch data from the Measurement Repository into a dataframe abstraction. The user can now either use the provided algorithms or write his own to design the analysis task in the notebook. Finally, the user can, from within the notebook and via a provided functionality, send the analytics results to a time series database (whisper) with a dashboard for visualization purposes (streaming job) or to a classical database (batch job).

The following sequence diagram corresponds to a use-case in which a user conducts a streaming data analysis task:

1. The user defines a streaming data analysis task via the Zeppelin notebook GUI of the Data Analysis Tool. One essential step of the design consist in specifying the data source. The user can browse schemas via the Schema Registry GUI (as part of the Data Analysis Tool), select the desired topic and the subsequent fields. Once the selection is done, the user will be able to click on a notebook creation button, upon which the user will be redirected to a newly created notebook, already populated with the selected elements. The user can of course write and fill those fields directly in the notebook. As for the core of the analytics, the user can either use the packaged functions, methods and algorithms shipped with the component or design the task completely from scratch within the notebook. The interactive notebook nature of the Zeppelin GUI makes such flexibility with respect to the analytics design possible. Once the user is satisfied with the design of the analytics task, he can run the associated blocks within the notebook to request the execution of the streaming analytics task which submits it to the Data Analysis Engine.
2. The Data Analysis Tool relays the task instantiation order to the Data Analysis Engine which initiates the task.

3. The Data Analysis Engine queries the Schema Registry for the user-specified information and create a stream abstraction that is periodically updated with new entries from the Message Bus.
4. The Data Analysis Engine performs the computations defined in the analytics task sequentially on the most recently added data on the stream that is retrieved from the Message Bus via the provided schema information (topic, fields to be analyzed). The results are sequentially sent to the Analysis Results Repository as they are computed.
5. In the absence of computation error, the computation is interrupted if and only if the user sends a kill signal from the Zeppelin notebook.
6. The results can be visualised through the Data Analysis Tool which integrates the dashboard associated with the Analysis Results Repository. The user is able to visualise the results on the dashboard at any time during the computation, the task does not need to be over in order to see the results, which is crucial in streaming applications (otherwise the task would have to be interrupted to see the results).

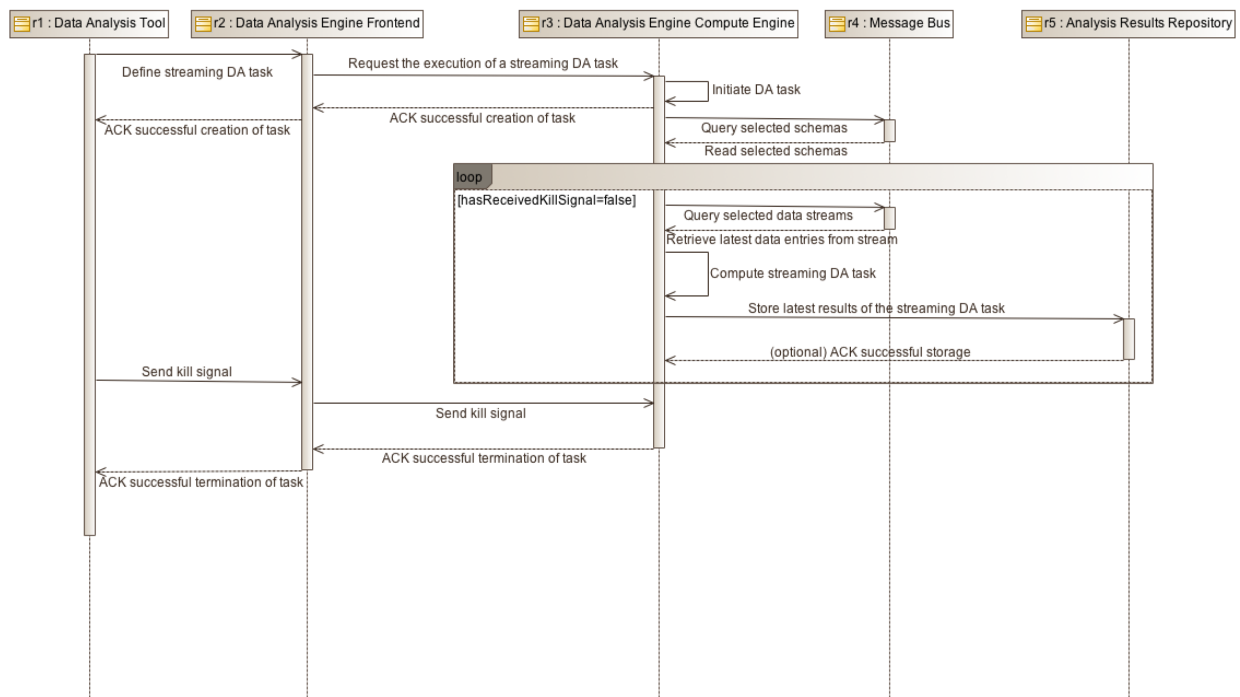

Figure 55: Sequence Diagram for the “Data Analysis in a streaming scenario” process

As mentioned in the first paragraph of this section, the second type of analysis that the user can conduct, with respect to the type of data being analysed, are batch tasks. The following sequence diagram corresponds to a use-case in which a user conducts a batch data analysis task:

1. The user defines a batch data analysis task via the Zeppelin notebook GUI of the Data Analysis Tool. One essential step of the design consist in specifying the data source. The user can browse schemas via the Schema Registry GUI (as part of the Data Analysis Tool), select the desired topic and the subsequent fields. Once the selection is done, the user will be able to click on a notebook creation button, upon which the user will be redirected to a newly created notebook, already populated with the selected elements. The

user can of course write and fill those fields directly in the notebook. Within the notebook, the user will be able to use the information (but not necessarily) to involve a Hbase table (from the Measurement Repository) as the data structure to perform computations on. As for the core of the analytics, the user can either use the packaged functions, methods and algorithms shipped with the component or design the task completely from scratch within the notebook. The interactive notebook nature of the Zeppelin GUI makes such flexibility with respect to the analytics design possible. Once the user is satisfied with the design of the analytics task, he can run the associated blocks within the notebook to request the execution of the streaming analytics task which submits it to the Data Analysis Engine.

2. The Data Analysis Tool relays the task instantiation order to the Data Analysis Engine which initiates the task.
3. The Data Analysis Engine queries the Schema Registry for the user-specified information and create a dataframe abstraction that contains all the entries present either in the specified Hbase table or any other user-specified accessible database (Measurement Repository).
4. The Data Analysis Engine performs the computations defined in the analytics task on the data present in the dataframe structure.
5. In the absence of computation error, the computation is interrupted either if the task is finished (it went over the dataframe a number of times equal to the optionally specified number of epochs) or if the user sends a kill signal from the Zeppelin notebook. It is not endless as a streaming task.
6. The results are finally stored once the analysis is complete in the Measurement Repository.

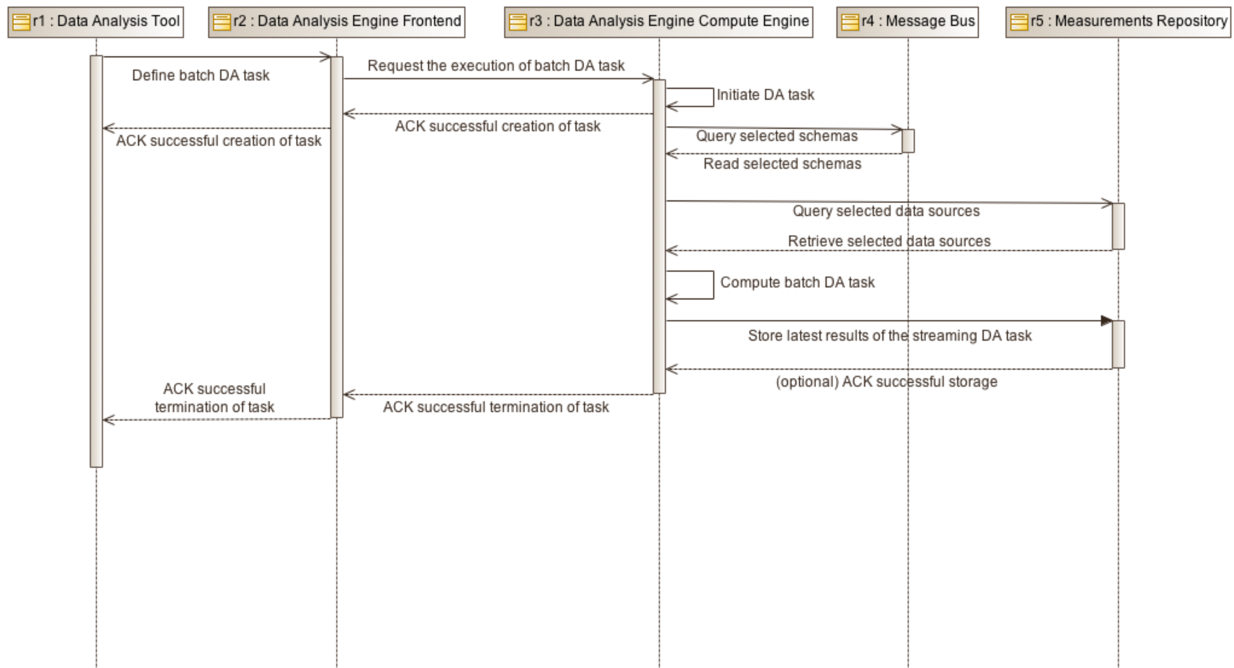

Figure 56: Sequence Diagram for the “Data Analysis in a batch scenario” process

6 Security considerations

In this chapter considerations about possible security procedures to be applied in RAWFIE, are described.

6.1 Network topology

In a rough overview of the RAWFIE network topology is given. The application server (with the platform services and web applications), database servers and message bus brokers will run with redundancies on several computers inside a computing centre. Testbed locations are connected to the RAWFIE internal network via VPN. Router configurations / VPN settings define routing restrictions (e.g. testbeds can only access the computers where the message bus broker runs). Connections to the internet are restricted by a firewall that only allows HTTPS connections to the proxy server but no other server in the RAWFIE internal network. The proxy server has X.509 server certificates installed signed by a public CA. Therefore, clients/experimenters can connect via a trusted HTTPS to the RAWFIE Web Portal. The proxy server forwards the HTTP request to the internal application server where the web applications runs. This forwarding can also include load balancing.

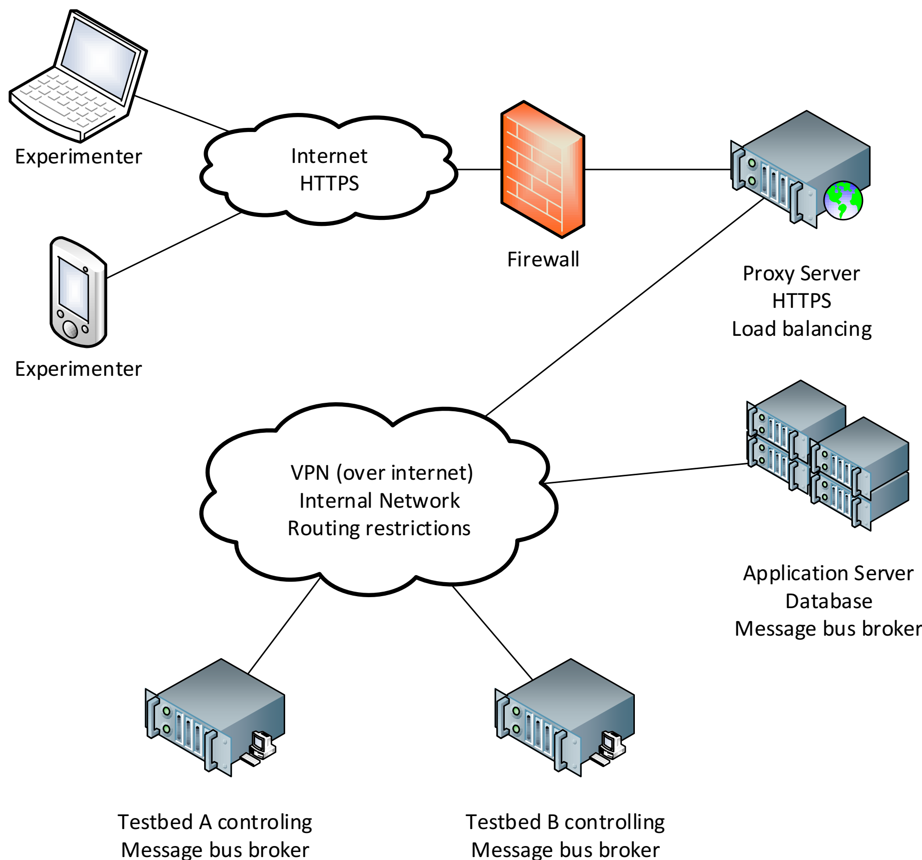

Figure 57: Network topology

6.2 Internal communication, encryption and authentication

To ensure a robust communication with integrity, the system may rely on the X.509 public key infrastructure with server and client certificates [8][9] via a SSL/TSL connection [10][11].
The RAWFIE internal communication is secured by server and client certificates. For this, an internal RAWFIE Certificate Authority (CA) may be used: a root private key and certificate are created and will be stored and managed at a secure place. Using the private key, the CA will sign the client / server certificates for each RAWFIE component (for each RAWFIE component separate private keys and certificates are created; the subject of the certificate is the component name). The CA certificate will be installed in the "trust store" of each component. Only TSL connection established using this certificates are trustworthy and are accepted by the components. Additionally, components that are allowed to call a service can be whitelisted (or have a special role in the User&Rights service) via the subject of the certificate. A simplified scheme of the TSL connection establishing is depicted in Figure 58.

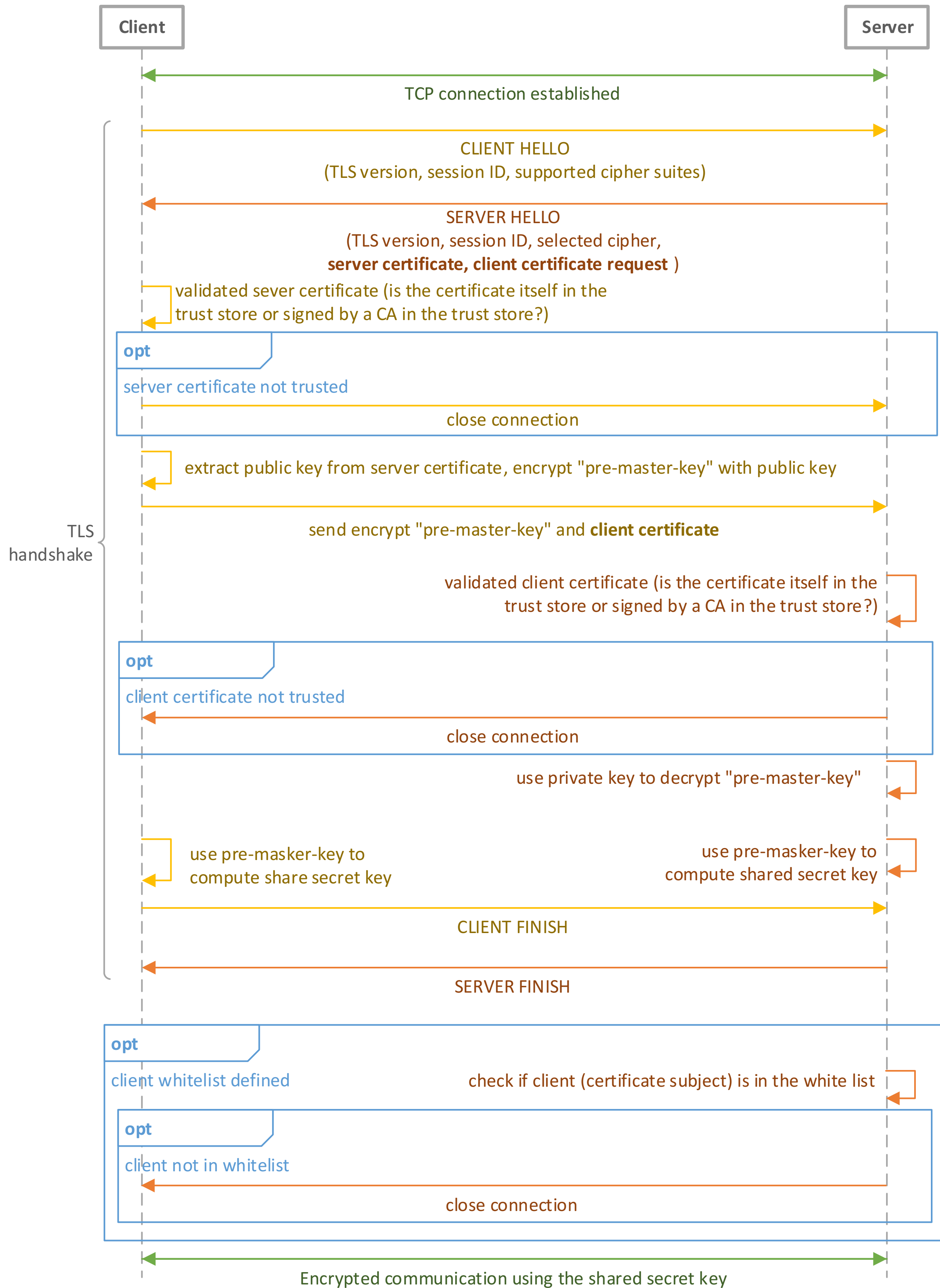

Figure 58: Simplified diagram for TLS handshake with server and client certificate

6.3 Message bus access

Access to the Message Bus may basically use the same concepts and techniques described before for the internal communication between components in RAWFIE.
As far as security issues are concerned, Apache Kafka supports the following mechanisms:

1. **Encryption of exchanged data** via SSL / TLS
    a. between clients (producers and consumers) and brokers (kafka servers)
    b. between brokers (kafka servers of the same cluster)
    c. between esternal tools and brokers (kafka servers)
2. **Authentication** of connections via SSL / TLS keys or using SASL (Simple Authentication and Security Layer) + Kerberos (for users definition) and JAAS (Java Autentication and Authorization Service)
    a. clients (producers e consumers) authentication to the brokers (kafka servers)
    b. authentication between brokers
    c. authentication of external tools to Kafka servers
    d. authentication of Kafka brokers to Zookeeper
3. **Authorization** mechanisms based on configurable **ACLs (Access Control Lists)**
    a. e.g. to control which clients (producers and consumers) can write / read to / from specific topics

Therefore, security of the exchanged data may be enabled through the creation of a TLS certificate signed by a Certificate Authority (CA), a public / private key pair for the authentication, and a proper configuration of the Apache Kafka brokers.
Authorization may be controlled on the other hand, through the configuration on each broker of specific ACL rules. Apache Kafka implements its own "Authorizer" module, which stores ACL rules for access and authorization control on specific resources. With a proper configuration of the Kafka Authorizer, it is therefore possible to allow / deny specific operations, from a specific host / client, on specific resources. This way, it will be possible to allow a specific client / host, to read and / or write (operations) from / to a topic (ACL resource).

7 Summary and Outlook

The final design and specification of RAWFIE components presented in this document provides instruction and guidelines for the physical deployment of the RAWFIE platform from a physical standpoint. It also provides instructions on how the several software components are deployed within different servers, together with the base technologies (Java, Tomcat, etc.) used for the software execution environments, in the final version of the prototype (Section 4).
The mapping of the requirements identified in D3.3, with the needed software components functionalities, was the starting point to provide a detailed design of all software components and their interfaces from a development and operational perspective (Section 4).
This detailed design will be adopted for the 3rd implementation cycle, whereas the information flows for some of the most relevant use cases highlighted in Section 5, will be the basis for verification and validation tests on components functionalities and interfaces.

Small modifications to some aspects of the architecture and components' design during the project's timeline are still possible. These may be mainly related to changes in some of the technological choices, like e.g. the specific billing solution for the accounting service.

8 References

[1] https://forgerock.org/opendj/
[2] http://www.icinga.org/
[3] http://www.jnrpe.it/
[4] http://www.nagios.org/
[5] http://www.xwiki.org/xwiki/bin/view/Main/WebHome
[6] http://mathias-kettner.com/checkmk_livestatus.html
[7] http://killbill.io/
[8] Internet Engineering Task Force, Network Working Group. *RFC 5280 - Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL)*. Online: https://tools.ietf.org/html/rfc5280
[9] ZYTRAX. *Survival guides - TLS/SSL and SSL (X.509) Certificates*. Online: http://www.zytrax.com/tech/survival/ssl.html
[10] Internet Engineering Task Force, Network Working Group. *RFC 5246 - The Transport Layer Security (TLS) Protocol Version 1.2*. Online: https://tools.ietf.org/html/rfc5246
[11] Tim Polk, Kerry McKay, Santosh Chokhani, (April 2014). *Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations*. National Institute of Standards and Technology. Online: http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r1.pdf
[12] Willner, A., Papagianni, C. A., Giatili, M., Grosso, P., Morsey, M., Al-Hazmi, Y., & Baldin, I. (2015). The Open-Multinet Upper Ontology Towards the Semantic-based Management of Federated Infrastructures. EAI Endorsed Trans. Scalable Information Systems, 2(7), e2
[13] L. Peterson, S. R. Ricci, A. Falk and J. Chase. Slice-based Federation architecture. Version 2.0 July 2010. URL: http://groups.geni.net/geni/raw-attachment/wiki/SliceFedArch/SFA2.0.pdf
[14] http://groups.geni.net/geni/wiki/GAPI_AM_API_V3
[15] http://docs.datamountaineer.com/en/latest/kcql.html

9 Annex

9.1 Detailed description of the API provided by RAWFIE components

9.1.1 Testbed Directory Service

Table 5: REST methods description for the Testbed Directory Service

Resource Path	***/request/getAllTestbeds***
Verb	GET
Consumes	None
Input example	None
Path parameter	None
Query parameters	None
Produces	application/json
Response	String with a JSON array representation of Testbeds information String with an error message in case of problems

Resource Path	***/request/getAllResources***
Verb	GET
Consumes	None
Input example	None
Path parameter	None
Query parameters	None
Produces	application/json
Response	String with a JSON array representation of Resources information String with an error message in case of problems

Resource Path	***/request/testbed/identifier/{id}***
Verb	GET
Consumes	None
Input example	None
Path parameter	*Testbed {id}*
Query parameters	None
Produces	application/json
Response	String with a JSON representation of the Testbed information String with an error message in case of problems

Resource Path	***/request/testbed/name/{name}***
Verb	GET
Consumes	None
Input example	None
Path parameter	*Testbed {name}*
Query parameters	None
Produces	application/json
Response	String with a JSON representation of the Testbed information String with an error message in case of problems

Resource Path	***/request/testbeds/uav***
Verb	GET
Consumes	None
Input example	None
Path parameter	None
Query parameters	None
Produces	application/json
Response	String with a JSON array representation of Testbeds information String with an error message in case of problems

Resource Path	***/request/testbeds/ugv***
Verb	GET
Consumes	None

Input example	None
Path parameter	None
Query parameters	None
Produces	application/json
Response	String with a JSON array representation of Testbeds information String with an error message in case of problems

Resource Path	***/request/testbeds/usv***
Verb	GET
Consumes	None
Input example	None
Path parameter	None
Query parameters	None
Produces	application/json
Response	String with a JSON array representation of Testbeds information String with an error message in case of problems

Resource Path	***/request/testbeds/auv***
Verb	GET
Consumes	None
Input example	None
Path parameter	None
Query parameters	None
Produces	application/json
Response	String with a JSON array representation of Testbeds information String with an error message in case of problems

Resource Path	***/request/testbeds?health=Val1&testbedstatusmessage=Val2***
Verb	GET
Consumes	None
Input example	None
Path parameter	None
Query parameters	*health & testbedstatusmessage values*
Produces	application/json
Response	String with a JSON array representation of Testbeds information String with an error message in case of problems

Resource Path	***/request/resource/identifier/{id}***
Verb	GET
Consumes	None
Input example	None
Path parameter	*Resource {id}*
Query parameters	None
Produces	application/json

Response	String with a JSON representation of the Resource information String with an error message in case of problems

Resource Path	***/request/resource/name/{name}***
Verb	GET
Consumes	None
Input example	None
Path parameter	*Resource {name}*
Query parameters	None
Produces	application/json
Response	String with a JSON representation of the Resource information String with an error message in case of problems

Resource Path	***/request/resources/testbedid/{id}***
Verb	GET
Consumes	None
Input example	None
Path parameter	*Testbed {id}*
Query parameters	None
Produces	application/json
Response	String with a JSON array representation of Resources information String with an error message in case of problems

Resource Path	***/request/resources?resource_status=Val1& resource_status_message=Val2 &resource_type=Val3&health=Val4***
Verb	GET
Consumes	None
Input example	None
Path parameter	None
Query parameters	*resource_status & resource_status_message & resource_type & health values*
Produces	application/json
Response	String with a JSON array representation of Resources information String with an error message in case of problems

Resource Path	***/request/createTestbed***
Verb	POST
Consumes	application/json
Input example	{ "testbedId": "IES7", "name": "CATANIA", "description": "IES internal env CT", "uavSupport": true, "ugvSupport": false,

<table>
<tr><td></td><td>"usvSupport": false,
"auvSupport": false,
"location": {
"x": 37.508940,
"y": 15.079919,
"type": "point"
},
"healthStatusLut": {
"healthStatusId": 2
},
"testbedstatusmessage": "UNKNOWN",
"simulated": true,
"indoor": true
}</td></tr>
<tr><td>Path parameter</td><td>None</td></tr>
<tr><td>Query parameters</td><td>None</td></tr>
<tr><td>Produces</td><td>None</td></tr>
<tr><td>Response</td><td>Exception message in case of problems</td></tr>
</table>

Resource Path	***/request/editTestbed***
Verb	PUT
Consumes	application/json

Input example	{
	"testbedId": "IES_1",
	"name": "CATANIA-1",
	"description": "IES internal env CT g7",
	"uavSupport": true,
	"ugvSupport": false,
	"usvSupport": false,
	"auvSupport": false,
	"location": {
	"x": 37.50894,
	"y": 15.079919,
	"type": "point"
	},
	"healthStatusLut": {
	"healthStatusId": 2
	},
	"testbedstatusmessage": "UNKNOWN",
	"simulated": true,
	"indoor": true
	}

Path parameter	None
Query parameters	None
Produces	application/json
Response	Exception message in case of problems

Resource Path	***/request/createResource***
Verb	POST
Consumes	application/json

Input example	{
	"resourceName": "Raspberry_IES_2",
	"resourceTypeLut": {
	"resourceTypeId": 2
	},
	"partition": 5,
	"connection": {
	"connectionId": "MST_wifi"
	},
	"testbed": {
	"testbedId": "IES_1"
	},
	"epsgCode": {
	"srid": 4326
	}
	}

Path parameter	None
Query parameters	None
Produces	None
Response	Exception message in case of problems

Resource Path	***/request/editResource***
Verb	PUT
Consumes	application/json

Input example	{
	"resourceId": 92,
	"resourceName": "Raspberry_IES_3",
	"resourceTypeLut": {
	"resourceTypeId": 2
	},
	"partition": 4,
	"connection": {
	"connectionId": "ROBOTNIK_wifi"
	},
	"testbed": {
	"testbedId": "IES_1"
	},
	"epsgCode": {
	"srid": 4326
	}
	}

Path parameter	None

Query parameters	None
Produces	None
Response	Exception message in case of problems

Resource Path	***/request/deleteTestbed***
Verb	DELETE
Consumes	application/json
Input example	{ "testbedId" : "IES_1" }
Path parameter	None
Query parameters	None
Produces	None
Response	Exception message in case of problems

Resource Path	***/request/deleteResource***
Verb	DELETE
Consumes	application/json
Input example	{ "resourceId" : "92" }
Path parameter	None
Query parameters	None
Produces	None
Response	Exception message in case of problems

Resource Path	***/request/deleteTestbedParameters/{id}***
Verb	DELETE
Consumes	None
Input example	None
Path parameter	*Testbed {id}*
Query parameters	None
Produces	None
Response	Exception message in case of problems

Resource Path	***/request/deleteResourceParameters/{id}***
Verb	DELETE
Consumes	None
Input example	None
Path parameter	*Resource {id}*
Query parameters	None
Produces	None

Response	Exception message in case of problems

9.1.2 System Monitoring Service

AVRO protocol definition:

```
{
  "protocol" : "SystemMonitoringServiceProtocol",
  "namespace" : "eu.rawfie.systemmonitoring.service.types",
  "types" : [ {
    "type" : "record",
    "name" : "ComponentServiceHealth",
    "fields" : [ {
      "name" : "componentId",
      "type" : "string"
    }, {
      "name" : "serviceName",
      "type" : "string"
    }, {
      "name" : "healthInformation",
      "type" : {
        "type" : "record",
        "name" : "HealthInformation",
        "fields" : [ {
          "name" : "message",
          "type" : [ "null", "string" ],
          "default" : null
        }, {
          "name" : "status",
          "type" : {
            "type" : "enum",
            "name" : "HealthStatus",
            "symbols" : [ "OK", "WARNING", "CRITICAL", "UNKNOWN" ]
          }
        }, {
          "name" : "updated",
          "type" : {
            "type" : "long",
            "CustomEncoding" : "DateAsLongEncoding"
          }
        } ]
      }
    } ]
  }, {
    "type" : "record",
    "name" : "ComponentServiceHealthRequest",
    "fields" : [ {
      "name" : "componentId",
      "type" : "string"
    }, {
      "name" : "serviceName",
      "type" : "string"
    } ]
  } ],
  "messages" : {
    "getComponentServiceHealth" : {
      "request" : [ {
        "name" : "ComponentServiceHealthRequest0",
        "type" : "ComponentServiceHealthRequest"
      } ],
      "response" : [ "null", "HealthInformation" ]
    },
    "receiveHealtStatusUpdate" : {
```

```
      "request" : [ {
        "name" : "ComponentServiceHealth0",
        "type" : "ComponentServiceHealth"
      } ],
      "response" : "null"
    },
    "getComponentServiceHealth_internal" : {
      "request" : [ {
        "name" : "ComponentServiceHealthRequest0",
        "type" : "ComponentServiceHealthRequest"
      } ],
      "response" : [ "null", "HealthInformation" ]
    },
    "getComponentServiceHealths" : {
      "request" : [ ],
      "response" : {
        "type" : "array",
        "items" : "ComponentServiceHealth",
        "java-class" : "java.util.List"
      }
    }
  }
}
```

9.1.3 User & Rights Service

AVRO protocol definition

```
{
    "protocol": "UserAndRightsServiceProtocol",
    "namespace": "eu.rawfie.users.service.types",
    "types": [
        {
            "type": "record",
            "name": "CheckTestbedRolesRequest",
            "fields": [
                {
                    "name": "userId",
                    "type": "string"
                },
                {
                    "name": "testbedId",
                    "type": "string"
                },
                {
                    "name": "requieredTestbedRoles",
                    "type": {
                        "type": "array",
                        "items": "string",
                        "java-class": "java.util.List"
                    }
                }
            ]
        },
        {
            "type": "record",
            "name": "Point",
            "namespace": "eu.rawfie.general.service.types",
            "fields": [
                {
                    "name": "latitude",
                    "type": "double"
                },
                {
```

```
                    "name": "longitude",
                    "type": "double"
                },
                {
                    "name": "altitude",
                    "type": [
                        "null",
                        "double"
                    ],
                    "default": null
                }
            ]
        },
        {
            "type": "record",
            "name": "TestbedData",
            "namespace": "eu.rawfie.general.service.types",
            "fields": [
                {
                    "name": "testbedId",
                    "type": "string"
                },
                {
                    "name": "name",
                    "type": "string"
                },
                {
                    "name": "description",
                    "type": "string"
                },
                {
                    "name": "location",
                    "type": "Point"
                },
                {
                    "name": "area",
                    "type": {
                        "type": "array",
                        "items": "Point",
                        "java-class": "java.util.List"
                    }
                },
                {
                    "name": "resourceId",
                    "type": {
                        "type": "array",
                        "items": "string",
                        "java-class": "java.util.List"
                    }
                }
            ]
        },
        {
            "type": "record",
            "name": "LoginCredentials",
            "fields": [
                {
                    "name": "name",
                    "type": "string"
                },
                {
                    "name": "password",
                    "type": "string"
                }
```

```
            ]
        },
        {
            "type": "record",
            "name": "UserData",
            "fields": [
                {
                    "name": "id",
                    "type": [
                        "null",
                        "int"
                    ],
                    "default": null
                },
                {
                    "name": "dn",
                    "type": [
                        "null",
                        "string"
                    ],
                    "default": null
                },
                {
                    "name": "uid",
                    "type": [
                        "null",
                        "string"
                    ],
                    "default": null
                },
                {
                    "name": "commonName",
                    "type": [
                        "null",
                        "string"
                    ],
                    "default": null
                },
                {
                    "name": "surname",
                    "type": [
                        "null",
                        "string"
                    ],
                    "default": null
                },
                {
                    "name": "forename",
                    "type": [
                        "null",
                        "string"
                    ],
                    "default": null
                },
                {
                    "name": "email",
                    "type": [
                        "null",
                        "string"
                    ],
                    "default": null
                },
                {
                    "name": "telephone",
```

```
                    "type": [
                        "null",
                        "string"
                    ],
                    "default": null
                },
                {
                    "name": "mobilePhone",
                    "type": [
                        "null",
                        "string"
                    ],
                    "default": null
                }
            ]
        },
        {
            "type": "enum",
            "name": "UserAndRightsServiceErrorType",
            "symbols": [
                "NOT_ALLOWED",
                "ILLEGAL_ARGUMENT",
                "UNKNOWN"
            ]
        },
        {
            "type": "error",
            "name": "UserAndRightsServiceError",
            "fields": [
                {
                    "name": "type",
                    "type": "UserAndRightsServiceErrorType"
                },
                {
                    "name": "detailMessage",
                    "type": [
                        "null",
                        "string"
                    ]
                }
            ]
        }
    ],
    "messages": {
        "getRoles": {
            "request": [ ],
            "response": {
                "type": "array",
                "items": "string",
                "java-class": "java.util.List"
            },
            "errors": [
                "UserAndRightsServiceError"
            ]
        },
        "getGroups": {
            "request": [ ],
            "response": {
                "type": "array",
                "items": "string",
                "java-class": "java.util.List"
            },
            "errors": [
                "UserAndRightsServiceError"
```

```
            ]
        },
        "getUsers": {
            "request": [ ],
            "response": {
                "type": "array",
                "items": "UserData",
                "java-class": "java.util.List"
            },
            "errors": [
                "UserAndRightsServiceError"
            ]
        },
        "setPassword": {
            "request": [
                {
                    "name": "LoginCredentials0",
                    "type": "LoginCredentials"
                }
            ],
            "response": "null",
            "errors": [
                "UserAndRightsServiceError"
            ]
        },
        "addGroup": {
            "request": [
                {
                    "name": "string0",
                    "type": "string"
                }
            ],
            "response": "null",
            "errors": [
                "UserAndRightsServiceError"
            ]
        },
        "login": {
            "request": [
                {
                    "name": "LoginCredentials0",
                    "type": "LoginCredentials"
                }
            ],
            "response": [
                "null",
                "UserData"
            ],
            "errors": [
                "UserAndRightsServiceError"
            ]
        },
        "checkLogin": {
            "request": [
                {
                    "name": "LoginCredentials0",
                    "type": "LoginCredentials"
                }
            ],
            "response": "boolean",
            "errors": [
                "UserAndRightsServiceError"
            ]
        },
```

```
        "getUser": {
            "request": [
                {
                    "name": "string0",
                    "type": "string"
                }
            ],
            "response": [
                "null",
                "UserData"
            ],
            "errors": [
                "UserAndRightsServiceError"
            ]
        },
        "checkRoles": {
            "request": [
                {
                    "name": "string0",
                    "type": "string"
                },
                {
                    "name": "array1",
                    "type": {
                        "type": "array",
                        "items": "string",
                        "java-class": "java.util.List"
                    }
                }
            ],
            "response": "boolean",
            "errors": [
                "UserAndRightsServiceError"
            ]
        },
        "addUser": {
            "request": [
                {
                    "name": "UserData0",
                    "type": "UserData"
                }
            ],
            "response": "null",
            "errors": [
                "UserAndRightsServiceError"
            ]
        },
        "editUser": {
            "request": [
                {
                    "name": "UserData0",
                    "type": "UserData"
                }
            ],
            "response": "null",
            "errors": [
                "UserAndRightsServiceError"
            ]
        },
        "setRolesOfUser": {
            "request": [
                {
                    "name": "string0",
                    "type": "string"
```

```
                },
                {
                    "name": "array1",
                    "type": {
                        "type": "array",
                        "items": "string",
                        "java-class": "java.util.List"
                    }
                }
            ],
            "response": "null",
            "errors": [
                "UserAndRightsServiceError"
            ]
        },
        "deleteUser": {
            "request": [
                {
                    "name": "string0",
                    "type": "string"
                }
            ],
            "response": "null",
            "errors": [
                "UserAndRightsServiceError"
            ]
        },
        "renameGroup": {
            "request": [
                {
                    "name": "string0",
                    "type": "string"
                },
                {
                    "name": "string1",
                    "type": "string"
                }
            ],
            "response": "null",
            "errors": [
                "UserAndRightsServiceError"
            ]
        },
        "addUserToGroup": {
            "request": [
                {
                    "name": "string0",
                    "type": "string"
                },
                {
                    "name": "string1",
                    "type": "string"
                }
            ],
            "response": "null",
            "errors": [
                "UserAndRightsServiceError"
            ]
        },
        "getUsersOfGroup": {
            "request": [
                {
                    "name": "string0",
                    "type": "string"
```

```
            }
        ],
        "response": {
            "type": "array",
            "items": "string",
            "java-class": "java.util.List"
        },
        "errors": [
            "UserAndRightsServiceError"
        ]
    },
    "getGroupsOfUser": {
        "request": [
            {
                "name": "string0",
                "type": "string"
            }
        ],
        "response": {
            "type": "array",
            "items": "string",
            "java-class": "java.util.List"
        },
        "errors": [
            "UserAndRightsServiceError"
        ]
    },
    "setRolesOfGroup": {
        "request": [
            {
                "name": "string0",
                "type": "string"
            },
            {
                "name": "array1",
                "type": {
                    "type": "array",
                    "items": "string",
                    "java-class": "java.util.List"
                }
            }
        ],
        "response": "null",
        "errors": [
            "UserAndRightsServiceError"
        ]
    },
    "getRolesOfGroup": {
        "request": [
            {
                "name": "string0",
                "type": "string"
            }
        ],
        "response": {
            "type": "array",
            "items": "string",
            "java-class": "java.util.List"
        },
        "errors": [
            "UserAndRightsServiceError"
        ]
    },
    "lockUser": {
```

```
            "request": [
                {
                    "name": "string0",
                    "type": "string"
                }
            ],
            "response": "null",
            "errors": [
                "UserAndRightsServiceError"
            ]
        },
        "unloockUser": {
            "request": [
                {
                    "name": "string0",
                    "type": "string"
                }
            ],
            "response": "null",
            "errors": [
                "UserAndRightsServiceError"
            ]
        },
        "validateUserData": {
            "request": [
                {
                    "name": "UserData0",
                    "type": "UserData"
                }
            ],
            "response": "boolean",
            "errors": [
                "UserAndRightsServiceError"
            ]
        },
        "getTestbedRoles": {
            "request": [ ],
            "response": {
                "type": "array",
                "items": "string",
                "java-class": "java.util.List"
            },
            "errors": [
                "UserAndRightsServiceError"
            ]
        },
        "getTestbeds": {
            "request": [ ],
            "response": {
                "type": "array",
                "items": "eu.rawfie.general.service.types.TestbedData",
                "java-class": "java.util.List"
            },
            "errors": [
                "UserAndRightsServiceError"
            ]
        },
        "getTestbedIds": {
            "request": [ ],
            "response": {
                "type": "array",
                "items": "string",
                "java-class": "java.util.List"
            },
```

```
            "errors": [
                "UserAndRightsServiceError"
            ]
        },
        "ping": {
            "request": [ ],
            "response": "string"
        },
        "getDirectRolesOfUser": {
            "request": [
                {
                    "name": "string0",
                    "type": "string"
                }
            ],
            "response": {
                "type": "array",
                "items": "string",
                "java-class": "java.util.List"
            },
            "errors": [
                "UserAndRightsServiceError"
            ]
        },
        "getGroupRolesOfUser": {
            "request": [
                {
                    "name": "string0",
                    "type": "string"
                }
            ],
            "response": {
                "type": "array",
                "items": "string",
                "java-class": "java.util.List"
            },
            "errors": [
                "UserAndRightsServiceError"
            ]
        },
        "removeUserFromGroup": {
            "request": [
                {
                    "name": "string0",
                    "type": "string"
                },
                {
                    "name": "string1",
                    "type": "string"
                }
            ],
            "response": "null",
            "errors": [
                "UserAndRightsServiceError"
            ]
        },
        "removeGroupFromRoles": {
            "request": [
                {
                    "name": "string0",
                    "type": "string"
                },
                {
                    "name": "array1",
```

```
                "type": {
                    "type": "array",
                    "items": "string",
                    "java-class": "java.util.List"
                }
            }
        ],
        "response": "null",
        "errors": [
            "UserAndRightsServiceError"
        ]
    },
    "getUserTestbedRoles": {
        "request": [
            {
                "name": "string0",
                "type": "string"
            }
        ],
        "response": {
            "type": "map",
            "values": {
                "type": "array",
                "items": "string",
                "java-class": "java.util.Collection"
            }
        },
        "errors": [
            "UserAndRightsServiceError"
        ]
    },
    "setUserTestbedRoles": {
        "request": [
            {
                "name": "string0",
                "type": "string"
            },
            {
                "name": "map1",
                "type": {
                    "type": "map",
                    "values": {
                        "type": "array",
                        "items": "string",
                        "java-class": "java.util.Collection"
                    }
                }
            }
        ],
        "response": "null",
        "errors": [
            "UserAndRightsServiceError"
        ]
    },
    "checkTestbedRoles": {
        "request": [
            {
                "name": "CheckTestbedRolesRequest0",
                "type": "CheckTestbedRolesRequest"
            }
        ],
        "response": "boolean",
        "errors": [
            "UserAndRightsServiceError"
```

```
            ]
        }
    }
}
```

9.1.4 Booking (Reservation) Service

AVRO protocol definition:


```
{
  "protocol" : "ReservationServiceProtocol",
  "namespace" : "eu.rawfie.reservation.service.types",
  "types" : [ {
    "type" : "record",
    "name" : "ReservationQuery",
    "fields" : [ {
      "name" : "from",
      "type" : {
        "type" : "long",
        "CustomEncoding" : "DateAsLongEncoding"
      }
    }, {
      "name" : "to",
      "type" : {
        "type" : "long",
        "CustomEncoding" : "DateAsLongEncoding"
      }
    }, {
      "name" : "userId",
      "type" : [ "null", {
        "type" : "string",
        "avro.java.string" : "String"
      } ],
      "default" : null
    } ]
  }, {
    "type" : "record",
    "name" : "ReservationStatusMsg",
    "fields" : [ {
      "name" : "reservationId",
      "type" : {
        "type" : "string",
        "avro.java.string" : "String"
      }
    }, {
      "name" : "status",
      "type" : {
        "type" : "enum",
        "name" : "ReservationStatus",
        "symbols" : [ "OK", "PENDING", "REJECTED", "CONFLICT", "DELETED" ]
      }
    }, {
      "name" : "msg",
      "type" : [ "null", {
        "type" : "string",
```

```
        "avro.java.string" : "String"
      } ],
      "default" : null
    } ]
  }, {
    "type" : "record",
    "name" : "ReservationData",
    "fields" : [ {
      "name" : "reservationId",
      "type" : {
        "type" : "string",
        "avro.java.string" : "String"
      }
    }, {
      "name" : "start",
      "type" : {
        "type" : "long",
        "CustomEncoding" : "DateAsLongEncoding"
      }
    }, {
      "name" : "end",
      "type" : {
        "type" : "long",
        "CustomEncoding" : "DateAsLongEncoding"
      }
    }, {
      "name" : "experimentRefId",
      "type" : [ "null", {
        "type" : "string",
        "avro.java.string" : "String"
      } ],
      "default" : null
    }, {
      "name" : "userRefId",
      "type" : {
        "type" : "string",
        "avro.java.string" : "String"
      }
    }, {
      "name" : "reservationItems",
      "type" : {
        "type" : "array",
        "items" : {
          "type" : "record",
          "name" : "ReservationItem",
          "fields" : [ {
            "name" : "reservationDataRefId",
            "type" : {
              "type" : "string",
              "avro.java.string" : "String"
            }
          }, {
            "name" : "experimentRefId",
            "type" : [ "null", {
              "type" : "string",
```

```
            "avro.java.string" : "String"
          } ],
          "default" : null
        }, {
          "name" : "lauchConfigRefId",
          "type" : [ "null", {
            "type" : "string",
            "avro.java.string" : "String"
          } ],
          "default" : null
        }, {
          "name" : "unqResourceId",
          "type" : {
            "type" : "string",
            "avro.java.string" : "String"
          }
        }, {
          "name" : "experimentStart",
          "type" : {
            "type" : "long",
            "CustomEncoding" : "DateAsLongEncoding"
          }
        }, {
          "name" : "experimentEnd",
          "type" : {
            "type" : "long",
            "CustomEncoding" : "DateAsLongEncoding"
          }
        } ]
      },
      "java-class" : "java.util.List"
    }
  } ]
} ],
"messages" : {
  "addReservation" : {
    "request" : [ {
      "name" : "ReservationData0",
      "type" : "ReservationData"
    } ],
    "response" : "ReservationStatusMsg"
  },
  "editReservation" : {
    "request" : [ {
      "name" : "ReservationData0",
      "type" : "ReservationData"
    } ],
    "response" : "ReservationStatusMsg"
  },
  "getReservations" : {
    "request" : [ {
      "name" : "ReservationQuery0",
      "type" : "ReservationQuery"
    } ],
    "response" : {
```

```
        "type" : "array",
        "items" : "ReservationData",
        "java-class" : "java.util.List"
      }
    },
    "getReservation" : {
      "request" : [ {
        "name" : "string0",
        "type" : "string"
      } ],
      "response" : "ReservationData"
    },
    "approveBooking" : {
      "request" : [ {
        "name" : "string0",
        "type" : "string"
      } ],
      "response" : "ReservationStatusMsg"
    },
    "rejectBooking" : {
      "request" : [ {
        "name" : "string0",
        "type" : "string"
      }, {
        "name" : "union1",
        "type" : [ "null", "string" ]
      } ],
      "response" : "ReservationStatusMsg"
    },
    "deleteReservation" : {
      "request" : [ {
        "name" : "string0",
        "type" : "string"
      } ],
      "response" : "ReservationStatusMsg"
    },
    "checkForConflictingReservations" : {
      "request" : [ {
        "name" : "ReservationData0",
        "type" : "ReservationData"
      } ],
      "response" : "boolean"
    }
  }
}
```

9.1.5 Launching Service

AVRO protocol definition:

```
{
  "protocol" : "LaunchingServiceProtocol",
  "namespace" : "eu.rawfie.launching.service.types",
  "types" : [ {
    "type" : "record",
```

```
    "name" : "ExperimentScheduleRequest",
    "namespace" : "eu.rawfie.general.service.types",
    "fields" : [ {
      "name" : "experimentId",
      "type" : {
        "type" : "string",
        "avro.java.string" : "String"
      }
    }, {
      "name" : "experimentStart",
      "type" : {
        "type" : "long",
        "CustomEncoding" : "DateAsLongEncoding"
      }
    }, {
      "name" : "experimentEnd",
      "type" : {
        "type" : "long",
        "CustomEncoding" : "DateAsLongEncoding"
      }
    } ]
  }, {
    "type" : "record",
    "name" : "LaunchingServiceActionResp",
    "fields" : [ {
      "name" : "executionId",
      "type" : "string"
    }, {
      "name" : "experimentId",
      "type" : "string"
    }, {
      "name" : "status",
      "type" : "boolean"
    }, {
      "name" : "msg",
      "type" : "string"
    } ]
  } ],
  "messages" : {
    "manualStart" : {
      "request" : [ {
        "name" : "string0",
        "type" : "string"
      } ],
      "response" : [ "null", "LaunchingServiceActionResp" ]
    },
    "schedule" : {
      "request" : [ {
        "name" : "ExperimentScheduleRequest0",
        "type" : "eu.rawfie.general.service.types.ExperimentScheduleRequest"
      } ],
      "response" : "LaunchingServiceActionResp"
    },
    "cancel" : {
      "request" : [ {
```

```
        "name" : "string0",
        "type" : "string"
      }, {
        "name" : "string1",
        "type" : "string"
      } ],
      "response" : "LaunchingServiceActionResp"
    }
  }
}
```

9.2 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
VPN	Virtual Private Network
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

9.3 Glossary

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.