This project has received funding from "HORIZON 2020" the European Union's Framework Programme for research, technological development and demonstration under grant agreement no 645220

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	D4.5 - Design and Specification of RAWFIE Components		
Confidentiality	PU	Deliverable type[1]	R
Deliverable File	D4.5	Date	30.06.2016
Approval Status[2]	1st and 2nd Reviewer, WP Leader	Version	1.0
Contact Person	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
Federica Toscano	IES	f.toscano@iessolutions.eu
Kostas Kolomvatsos	UOA	kostasks@di.uoa.gr
Vasil Kumanov	EPSILON	vasil.kumanov@epsilon-bulgaria.com
Marcel Heckel	FRAUNHOFER	Marcel.Heckel@ivi.fraunhofer.de
Nikolaos Pringouris	HAI	priggouris.nikolaos@haicorp.com
Damien Piquet	CSEM	damien.piguet@csem.ch
Philippe Dallemagne	CSEM	pda@csem.ch
Lionel Blondè	HES-SO	lionel.blonde@hesge.ch
Miquel Cantero	ROBOTNIK	mcantero@robotnik.es
Ricardo Martins	MST	rasm@oceanscan-mst.com

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	2016-05-24	Start discussions and preparation of the 2nd version of the components design (IES internal)	all
0.2	2016-06-08	TOC / Initial version of contents	all
0.3	2016-06-16	First round of contributions on components design	Section 4
0.4	2016-06-22	New contributions	Sections 3 and 4
0.5	2016-06-23	Updated contents	Sections 4 and 5
0.6	2016-06-27	Updated contributions and refinements	Sections 4 and 5
0.7	2016-06-27	Version for internal review	all
0.8	2016-06-29	Version with revisions and comments	all
0.9	2016-06-30	Improvements and modifications	all
1.0	2016-07-01	Finalisation	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, a new version of the Design and Specification of RAWFIE Components deliverable, for the 2nd technical iteration cycle, is released.

Built on the updated list of architectural components and technological decisions presented in the D4.4, on the first components' prototype, and especially on the updated requirements' definition of D3.2, this report presents the updated design documentation of RAWFIE Architectural tiers, and belonging components. It moves from the conceptual perspective of the system design in the 1st iteration, to development and operational perspectives, by providing more details regarding the way the components are, or will be, implemented and deployed.

Keywords:
design, architecture, component, responsibilities, operations, attributes, workflows, interactions, relationships, interfaces, diagrams, methods, classes, deployment, server, scenarios, physical architecture, service level, scalability, high availability, failover

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 D4.5 13
1.2 Relation to other deliverables 13
2 Overview of changes for the second iteration of the design and specifications of RAWFIE components 13
3 Physical design of the overall architecture 14
3.1 Physical infrastructure 14
3.1.1 Availability, fault tolerance and scalability 15
3.1.2 Backup strategies 15
3.2 Deployment of the RAWFIE platform 16
4 Design and specification of the software components – 2nd iteration 18
4.1 Frontend Tier (Web Portal GUI elements) 18
4.1.1 Overview 18
4.1.2 Wiki Tool 19
4.1.3 Resource Explorer Tool 20
4.1.4 Booking Tool 22
4.1.5 Experiment Authoring Tool 24
4.1.6 Experiment Monitoring Tool 28
4.1.7 System Monitoring Tool 32
4.1.8 UxV Navigation Tool 34
4.1.9 Visualisation Tool 37
4.1.10 Data Analysis Tool 41
4.2 Middle Tier (Services and Communication components) 43
4.2.1 Overview 43
4.2.2 Testbed Directory Service 45
4.2.3 EDL Compiler and Validator 51
4.2.4 Experiment Validation Service 54
4.2.5 Users & Rights Service 56
4.2.6 Booking Service 63

4.2.7 Launching Service 69
4.2.8 Visualisation Engine 77
4.2.9 Data Analysis Engine 84
4.2.10 System Monitoring Service 86
4.2.11 Accounting Service (FRAU) 91
4.2.12 Experiment Controller 93
4.3 Testbed Tier (Testbeds and Resources control components) 97
4.3.1 Description 97
4.3.2 Monitoring Manager 97
4.3.3 Network Controller 100
4.3.4 Resource Controller (plus Navigation Service sub-component) 103
4.3.5 UxV Proximity component 107
4.3.6 Testbed Manager 116
4.3.7 Aggregate Manager 121
4.3.8 UxV Node 123
4.3.9 AVRO formatted messages and Kafka Schema Registry 131
4.3.10 Common Sensors for UxVs 133
5 Global Sequence diagrams showing main RAWFIE processes 135
5.1 Registration of Testbed Resources 135
5.2 Booking Testbed Resources (HAI) 137
5.3 System Monitoring 139
5.4 Experiment Execution and Monitoring 141
5.5 Experiment Measurements Recording 143
5.6 Authoring and Launching of an Experiment 145
5.7 Data Analysis 147
6 Summary and Outlook 149
7 References 150
8 Annex 150
8.1 Abbreviations 150
8.2 Glossary 152
A 152
Accounting Service 152
Aggregate Manager 153
Avro 153

B 153
Booking Service 153
Booking Tool 153
C 153
Common Testbed Interface 153
Component 153
D 153
Data Analysis Engine 153
Data Analysis Tool 153
E 154
EDL Compiler & Validator 154
Experiment Authoring Tool 154
Experiment Controller 154
Experiment Monitoring Tool 154
Experiment Validation Service 154
M 154
Master Data Repository 154
Measurements Repository 154
Message Bus 154
Module 155
Monitoring Manager 155
N 155
Network Controller 155
L 155
Launching Service 155
R 155
Resource Controller 155
Resource Explorer Tool 155
Results Repository 155
Resource Specification (RSpec) 155
S 156
Schema Registry 156
Service 156
Slice Federation Architecture (SFA) 156

Subsystem 156
System 156
System Monitoring Service 156
System Monitoring Tool 156
T 156
Testbed 156
Testbeds Directory Service 157
Testbed Manager 157
Tool 157
U 157
Users & Rights Repository 157
Users & Rights Service 157
UxV 157
UxV Navigation Tool 157
UxV node 157
V 157
Visualisation Engine 158
Visualisation Tool 158
W 158
Web Portal 158
Wiki Tool 158

List of Figures

Figure 1: Overview of the Physical RAWFIE infrastructure 15
Figure 2: Web Portal – Deployment / Components Diagram 18
Figure 3: Wiki Tool - Class diagram 19
Figure 4: Resource Explorer Tool - Class diagram 21
Figure 5: Booking Tool - Class diagram 24
Figure 6: Experiment Authoring Tool - Class diagram 26
Figure 7: Experiment Authoring Tool – Open and edit an EDL script 27
Figure 8: Experiment Authoring Tool - Create and validate an EDL experiment 28
Figure 9: Experiment Monitoring Tool - Class diagram 29
Figure 10: Experiment Monitoring Tool – Select experiment 30
Figure 11: Experiment Monitoring Tool – View experiment details 31
Figure 12: Experiment Monitoring Tool – Cancel experiment 31
Figure 13: System Monitoring Tool - Class diagram 33
Figure 14: UxV navigation tool – Class diagram 35
Figure 15: UxV navigation tool – High level sequence diagram 36
Figure 16: Visualisation Tool - Class diagram 38
Figure 17: Visualisation Tool - Sequence diagram 40
Figure 18: Data Analysis Tool – Class diagram 42
Figure 19: Data Analysis Tool – Sequence diagram involving the Data Analysis Tool in the case of a stream analytic task. 43
Figure 20: Middle Tier Components – Deployment / Components Diagram 44
Figure 21: Testbed Directory Service – Class diagram 48
Figure 22: Search Resource internal Sequence diagram 49
Figure 23: Testbed Directory Service – Register a new testbed in the platform 50
Figure 24: Testbed Directory Service - Add a new UxV device into a Testbed facility 51
Figure 25: Experiment Compiler – Class diagram 53
Figure 26: Experiment Compiler - Sequence diagram 54
Figure 27: Experiment Validator – Class diagram 55
Figure 28: Experiment Validator - Sequence diagram 56
Figure 29: Users & Rights Service - Class diagram 57
Figure 30: Users & Rights Service – Password-based user login 58
Figure 31: Users & Rights Service – X.509 Certificate-based user login 59
Figure 32: Users & Rights Service – Check user authorisation 60
Figure 33: Users & Rights Service – Check user authorisation 62
Figure 34: Booking Service - Class diagram 65
Figure 35: Booking Service - Overview 66
Figure 36: Booking Service – View bookings of a testbed 67
Figure 37: Booking Service – Add/Edit a booking 68
Figure 38: Booking Service – Approve/Reject booking 69
Figure 39: Launching service – Class diagram 73
Figure 40: Experiment launching Service Overview - Sequence diagram. 74
Figure 41: Experiment Launching Service - Manual Launch Sequence diagram. 75
Figure 42: Experiment Launching Service - Scheduled Launch Sequence diagram. 76
Figure 43: Experiment Launching Service - Cancellation Sequence diagram 77
Figure 44: Visualisation Engine - Class diagram 79

Figure 45: Visualisation Engine - Overview 80
Figure 46: Visualisation Engine – Start an experiment visualisation 81
Figure 47: Visualisation Engine - Position Update 82
Figure 48: Visualisation Engine - Update Status of an Experiment 82
Figure 49: Visualisation Engine – Stop an experiment visualisation 83
Figure 50: Visualisation Engine - Replay an experiment 83
Figure 51: Data Analysis Engine - Class diagram 85
Figure 52: Data Analysis Engine – streaming analytic task 85
Figure 53: System Monitoring Service - Class diagram 88
Figure 54: System Monitoring Service – Checking procedure 89
Figure 55: System Monitoring Service – Received health status via message bus 90
Figure 56: System Monitoring Service – View health statuses 90
Figure 57: Accounting Service – Class diagram 92
Figure 58: Experiment Controller - Class diagram 95
Figure 59: Experiment Controller – Sequence diagram 96
Figure 60: Testbed control, analysis and monitoring– Deployment / Components Diagram 97
Figure 61: Monitoring Manager – High level class diagram 99
Figure 62: Monitoring Manager – Monitoring sequence diagram 100
Figure 63 - Network Controller Class Diagram 102
Figure 64 – Starts New Experiment Connection Provisioning 102
Figure 65 - Change Connection during an experiment 103
Figure 66: Resource Controller – Class diagram 105
Figure 67: Resource Controller – Sequence diagram 106
Figure 68: UxV Proximity component class diagram 110
Figure 69: UxV Proximity Deployment diagram 110
Figure 70: Sequence diagram - topic subscription at the Proximity component 114
Figure 71: Proximity component subscription reception and data publication sequence diagram 115
Figure 72: data reception at the proximity component sequence diagram 116
Figure 73: Testbed Manager – Class Diagram 119
Figure 74:Testbed Manager experiment handling sequence diagram 120
Figure 75: Testbed components monitoring sequence diagram 121
Figure 76: Aggregate Manager – Class Diagram 122
Figure 77: Aggregate Manager- get the list of available resources sequence diagram 122
Figure 78 – Aggregate Manager allocate resources sequence diagram 123
Figure 79: Sequence Diagram for “Registration of Testbed Resources” process 136
Figure 80: Sequence Diagram for “Booking Resource” process 138
Figure 81: Sequence Diagram for “System Monitoring” process 140
Figure 82: Sequence Diagram for “Experiment Execution and Monitoring” process 142
Figure 83: Sequence Diagram for “Experiment Measurements Recording” process 144
Figure 84: Sequence Diagram for “Authoring and Launching of an Experiment” process 146
Figure 85: Sequence Diagram for the “Data Analysis in a streaming scenario” process 148
Figure 86: Sequence Diagram for the “Data Analysis in a batch scenario” process 148

List of Tables

Table 1: List of requirements for an UxV node to be used in RAWFIE 124
Table 2: Summary of UxVs functions 126

Part III: Executive Summary

The present document is the second in a series of three documents about RAWFIE platform components design and specification. As such, it is delivered at the beginning of the second RAWFIE development iteration cycle.

The report starts with the concepts for the physical infrastructure of RAWFIE. To this end, a traditional cloud based approach is described together with possible backup strategies, to address the needs of scalability, high availability and fault tolerance of RAWFIE services.

Further, D4.5 consists of an updated design specification of all components at the different application tiers. The purpose of the adopted design and specification approach is twofold: to present and define how the new functionalities expected for the new development cycle iteration - especially the ones highlighted in the latest requirements' specification document (D3.2) - will be implemented, and to observe in depth the data flow and interaction between the components specified in the deliverable D4.4. As in the previous iteration of the same document (D4.2) UML is used as design and modelling approach.
While in deliverable D4.2, the design of the several RAWFIE components was presented from a conceptual perspective, now the focus is on the development and operational perspectives. For most of the components and where already possible, the provided UML class diagrams are therefore more accurate, with details about the actual intended implementation of interfaces, methods and attributes (development perspective).
More, the physical design of the system and the components, and their interactions, is provided by the mean of UML Deployment and Components diagram (operational perspective), where the following main RAWFIE servers can be identified:

- Web Application Server (Frontend components)
- Middle Tier Services Server (Middle Tier components and most of the business logic)
- System Monitoring Services Server (all the components related to the envisaged system monitoring strategies)
- GIS Server (the server that will be in charge of handling and serving geographical maps and layers)
- Master Data and Users & Rights Repositories Server
- Measurements Repository Server
- Analysis Results Repository Server
- Testbed Components Server
- UxV Node Server (normally, the embedded server onboard of each UxV nodes)

This architectural design document ends with a set of UML sequence diagrams, showing how several different software components interact and the interfaces are used, in some of the most relevant RAWFIE processes / use cases.

Part IV: Main Section

1 Introduction

1.1 Scope of D4.5

This deliverable describes the 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 within WP3.

Practically it answers:

- How the requirements defined in D3.2 are translated into software design and, as a consequence, 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 processes / use cases (using sequence diagrams)

1.2 Relation to other deliverables

The design of components in D4.5 is elaborated based on the requirements specification presented in D3.2 – Specification of requirements for the second RAWFIE development cycle. High level architecture presented in D4.4 provides an input for this deliverable as a general architectural picture, where relations among components have firstly been identified. The work in this deliverable takes also into account the outcome of 1st iteration development activities presented in D5.1 and will guide the work for the 2nd iteration development period which will be presented in D5.2.
D4.5 is actually expected to provide a deeper architecture analysis and further detailed descriptions of components and the whole operational architecture. Therefore, the main intention of this deliverable is to design software classes for the components and illustrate their interfaces, operations and responsibilities, as well as to specify the way the components will be deployed.

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

This chapter shortly summarises the most important changes made in comparison to D4.2

- Physical infrastructure (new Section 3) and physical design (in Section 4) added

- Global sequence diagrams, highlighting some of the most relevant RAWFIE processes / use cases (Section 5, this updates the previous, global sequence diagrams presented at the beginning of D4.2)
- Changes to basically all components design, moving from a conceptual perspective to a more operational / development oriented design, with updated class diagrams and components' specific sequence diagrams
- Link of components' functionalities with the updated list of requirements from D3.2, for each component
- Besides the existing components' updates, the following modifications can be highlighted:
    - New Accounting service added
    - New Wiki Tool added
    - New Aggregate Manager (SFA related) added
    - Testbed Proxy removed
    - In the Tetsbed Tier, UxVs section completely re-elaborated, in order to provide specifications and guidelines, especially to external UxVs owners, on how new UxVs can be integrated in RAWFIE

3 Physical design of the overall architecture

This section describes the global RAWFIE architecture from a physical design standpoint.
First, a concept of the RAWFIE physical infrastructure is presented, which aims to ensure certain quality of service levels in terms of High Availability (HA) of the platform to minimise the possibility of a service downtime, scalability to serve increasing numbers of concurrent requests that can be served with an acceptable response time, and backup / failover strategies.
Before describing the updated version of the individual software design for each one of the components, formerly presented in deliverable D4.2 – Design and Specification of RAWFIE Components (a), we also present the physical design of the platform using Physical UML diagrams (Deployment and Components diagrams). This way, we specify the different types of server instances where the components of the RAWFIE tiers will be deployed, the base technologies adopted and the software execution environments. Through deployment diagrams, we will also provide a first idea of the main interactions between the components of the different tiers.

3.1 Physical infrastructure

The physical RAWFIE platform infrastructure, as depicted in Figure 1, is inspired by a traditional cloud oriented approach, where different server instances, or cluster of servers and services hosted in the cloud, are used in order to fulfill the high availability, scalability and failover objectives and strategies.

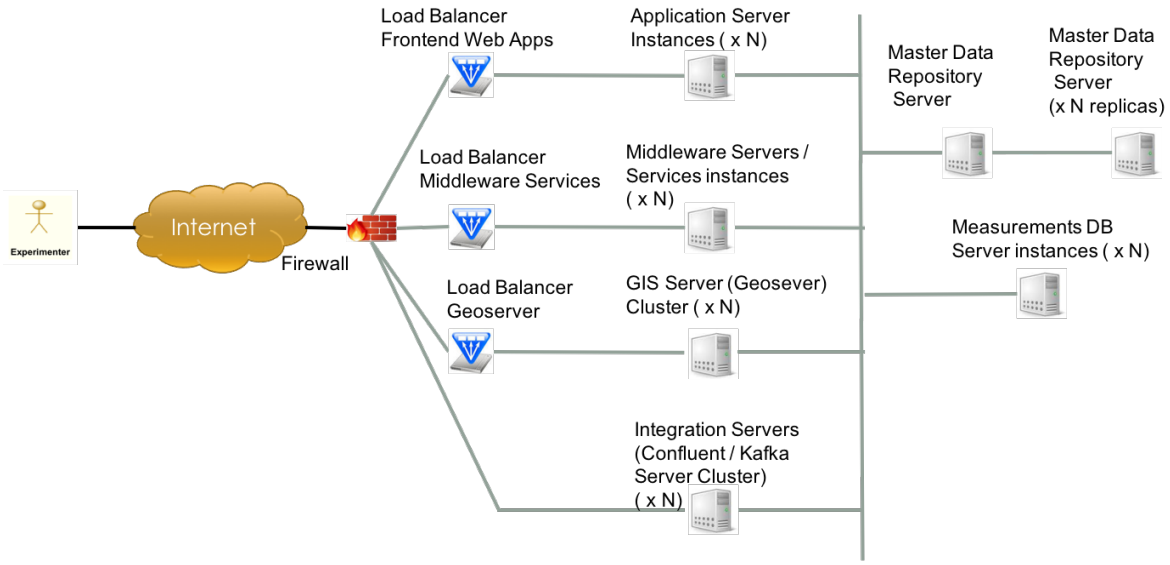

Figure 1: Overview of the Physical RAWFIE infrastructure

3.1.1 Availability, fault tolerance and scalability

The capability of the platform to minimise the service downtimes is ensured, according to the concepts highlighted in Figure 1, by the use of multiple parallel instances of each of the servers where the services will run in the different tiers: Application Server for the Web Apps (Frontend tier components), the Middleware Server (Middle Tier components), as well as the GIS Server (running Geoserver [1]) and the Integration Server (running Confluent [2] and Apache Kafka Message Bus [3]). This setup should ensure that at least some of the server instances for each service type, will always available, in case of failures.

In this configuration, the different data repositories (especially the Master Data and Measurements Data Repositories, see also deliverable D4.4 for reference), are also expected to be replicated. In the case of the Master Data Repository relational database, a master instance will support writes, and the replicas will be used to support huge volumes of read requests. In the case of the NoSQL [4] solution adopted for the Measurements Data Repository, by setting up different instances for both writes and reads.

The continuous availability of the services and data repositories after a fault, is either automatically ensured by the remaining servers instances, or manually recovered in a matter of minutes, in case there is the need to wake up a sleeping instance of a failover server.

As far as the scalability is concerned, Load Balancers components are deployed for the Web App, the Middleware and the GIS Servers, in order to split the load between several server instances in case of increase in the processing demand. Also, of special importance will be the configuration of a cluster of Apache Kafka message brokers, to address the high number of messages that will need to be served for controlling the UxV devices, and for acquiring sensors and position measurements, thus ensuring acceptable performance of the whole system, and safety of the equipment, during the execution of one or more concurrent experiments.

Scalability at the data repository level is ensured by splitting (balancing) the incoming requests to the different server instances, as well.

3.1.2 Backup strategies

Common and freely available backup solutions may be used, like:

- the Backup Manager application available on any Linux OS (the reference OS for the RAWFIE platform deployment)
- the rsync daemon and some configured cronjobs for setting up the time schedule of the different backups (always available in any Linux OS distribution)

Some of the possible examples of scheduled backup strategies are:

- Full scheduled backup of PostgreSQL database/s every night at 00:00, to a dedicated backup server
- Full scheduled backup of other repositories every night at 01:00, to a dedicated backup server
- Full scheduled backup of local disks of each server (including running software services) every day at 02:00, to a dedicated backup server

3.2 Deployment of the RAWFIE platform

The physical elements listed in the following contribute to the RAWFIE physical architecture. All these elements will be 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 [5], 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 [6]).
- 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 [7], monitoring DB, JNRPE [8] plugin and System Monitoring Service).

- Testbed
  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

4 Design and specification of the software components – 2nd iteration

4.1 Frontend Tier (Web Portal GUI elements)

4.1.1 Overview

Web Portal represents the central user interface that provides access and links to most of the RAWFIE tools/services enabling end-users to interact with the platform through the use of a web browser. Web Portal is also responsible for the management of users, access rights and login credentials. A UML Deployment Diagram of Web Portal components and their interactions with Middle Tier and Data Tier components is presented in Figure 2.**Error! Reference source not found.**

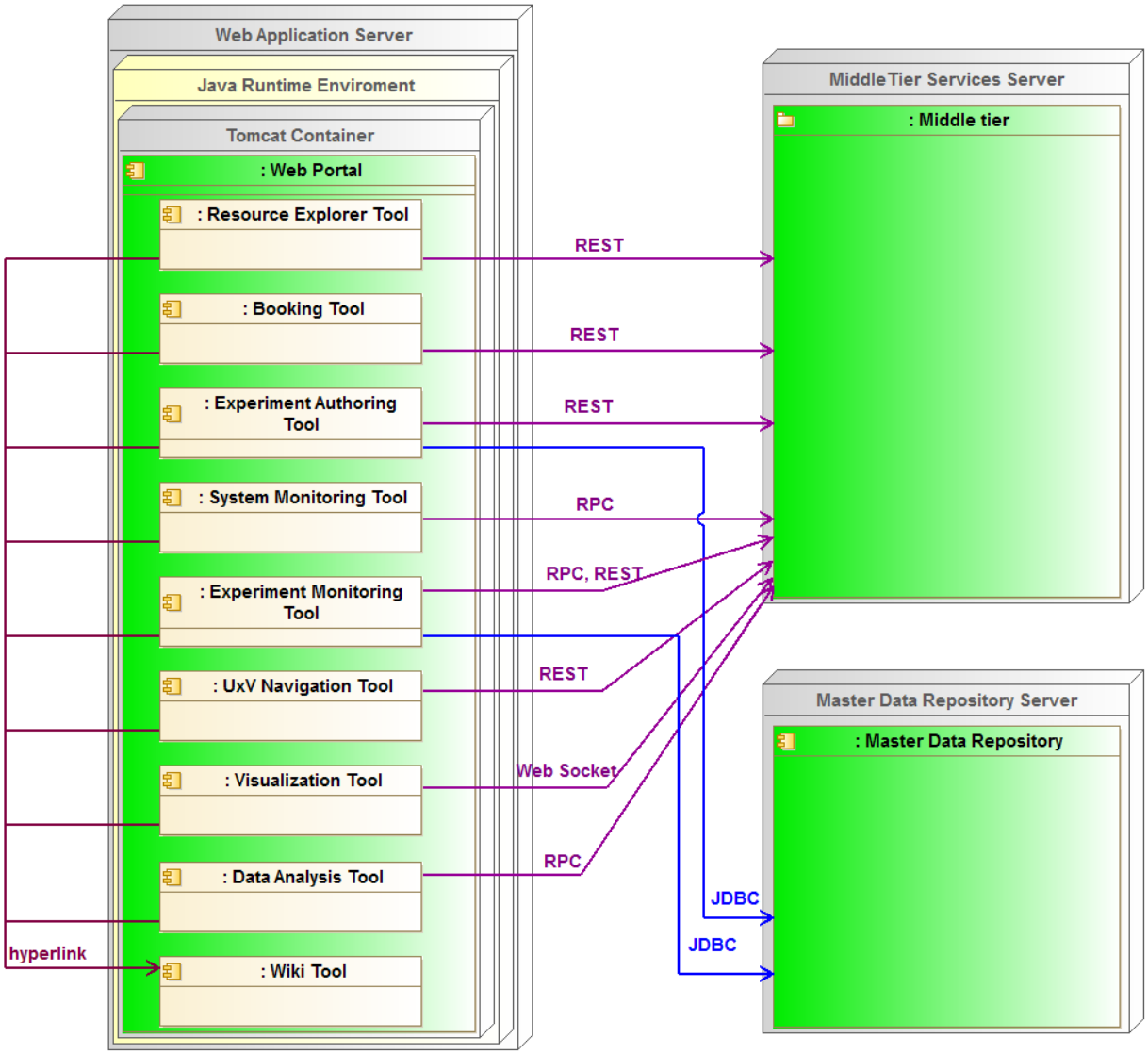

Figure 2: Web Portal – Deployment / Components Diagram

4.1.2 Wiki Tool

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

Component requirements as identified in D3.2

ID (Priority)	Description	Requirement Mapping with component's functionalities
PT-WEB-P-003 (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.

Responsibilities

- Manage information about
  - Manuals (e.g. of RAWFIE tools)
  - Technical documentation (e.g. architecture of RAWFIE or technical requirements of UxVs)
  - Other information (e.g. extended descriptions of testbeds and UxVs)
- Provide static links to content pages (for linking by other tools)
- Allow editing by authorised users

Operations and attributes

A third party application will be used to realise the Wiki Tool (e.g. MediaWiki[4]). Beside the HTTP/HTML interface for displaying, no special operations are foreseen.

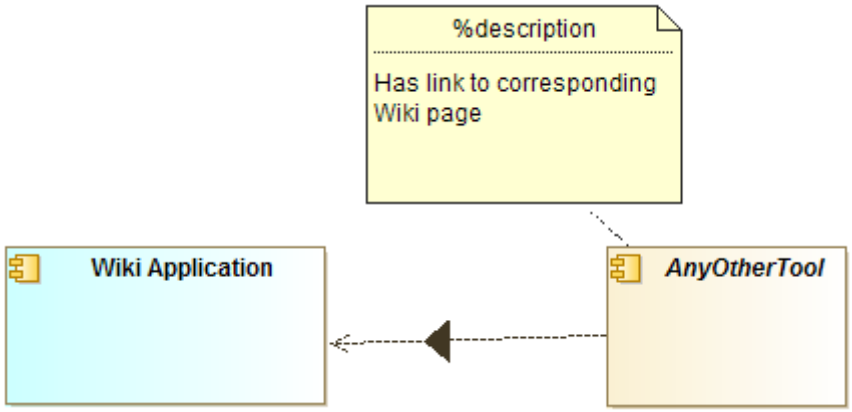

Figure 3: Wiki Tool - Class diagram

Interactions and relationships with other components

[4] https://www.mediawiki.org

Provided Interfaces

- HTTP/HTML interface for displaying and linking of pages by other tools

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.

Component requirements as identified in D3.2

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.

Responsibilities

The main responsibilities of the Resource Explorer Tool are:

- Visualise Data from the Testbeds Directory Service
- Provide ability to search and select available resources inside a testbed
- Add and edit testbeds and UxVs

Operations and attributes

The Resource Explorer Tool provides several web pages to interact with the Testbeds 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.

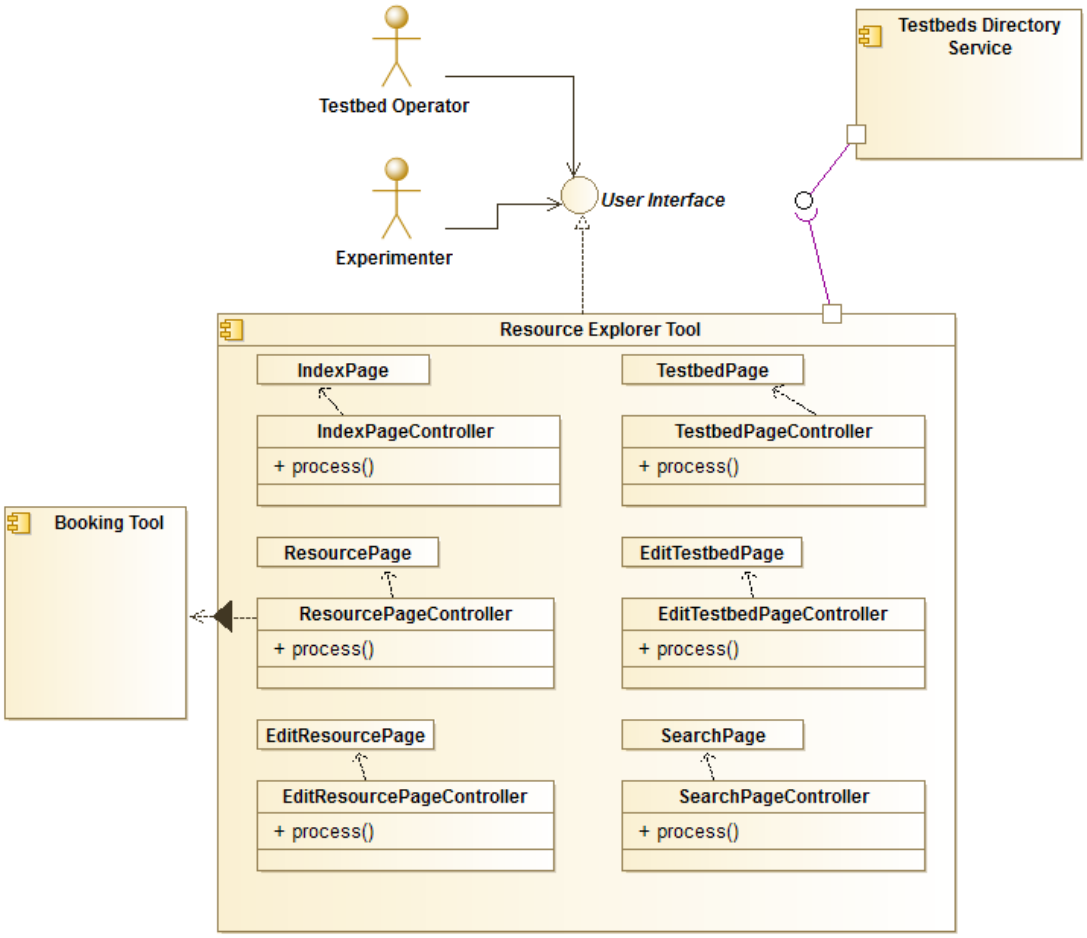

Figure 4: Resource Explorer Tool - Class diagram

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

Interactions and relationships with other components

Provided Interfaces

- Web portal GUI:
  Used by the Experimenters to find appropriate Testbeds and UxVs. Used by the Testbed Operators to maintain (add, edit, delete) data about Testbeds and UxVs.

Required Interfaces

- Testbeds Directory Service Interface:
  Read the resource data for visualisation, add and edit data
- 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.

Component requirements as identified in D3.2

ID (Priority)	Description	Requirement Mapping
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 myslice architecture and the notion of slices reservations	Not addressed - It will be investigated in the next iteration
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 interface enabling communication via Avro JSON messages)
PT-BOO-T-006 (HIGH)	Booking Tool should provide appropriate functionality for viewing the reservations of a user/experimenter	Fulfilled by existence of *CalendarViewPage*
PT-BOO-T-007 (HIGH)	Booking Tool should allow editing of existing Reservations	Fulfilled by existence of *EditBookingPage*
PT-BOO-T-008 (HIGH)	Booking Tool should allow cancellation of existing 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	A booking (reservation) *status* field will be included in every booking

	the user in case status of reservation request is not directly available.	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	It will not be addressed in this iteration and will be investigated in the next iteration cycle
PT-BOO-T-012 (HIGH)	Booking functionality should provide means to ensure fairness in resource booking as well as protect for malevolent actions that a user may perform.	Not applicable - it should be considered by the Booking Service

Responsibilities

The main responsibilities of the Booking Tool are:

- To provide intuitive overview visualization (i.e. via a calendar like view) of existing user reservations (bookings) in the RAWFIE platform
- To allow for initiation of reservation requests for one or multiple resources in a specific period of time (in one or more testbeds)
- To allow modification (editing) of existing reservations
- To allow cancellation (removal) of existing reservations
- To provide details on a selected reservation including its status
- To allow approval or rejection of Bookings request by a testbed administrator role

Operations and attributes

The Booking Tool provides a set of web pages, available to all registered users and potential experimenters, allowing the visualization (via a calendar view) of all the reservations for a specific testbed and its UxVs resources. Details of a booking could be viewed in an extra page. The tool also provides the ability to add a new booking, as well as edit or remove existing ones.
There are also pages restricted to testbed administrator that can be used for approving or rejecting a booking request initiated by a platform user.
Booking of a resource(s) generally creates a request for the Booking Service. The request may not return a synchronous response concluding the acceptance or rejection of the booking but rather put the request in a pending status until a platform administrator, decides whether the request can be fulfilled or not. Therefore, all booking responses are expected to return an appropriate status, which should be visible by the user either in the calendar view or in the booking details view.
Since the booking tool will delegate all its actions to the Booking Service (see related section) for further processing, in this section we present a simple UML class like diagram showing mainly the available pages and the interfaces with other RAWFIE elements. The reader should refer to the corresponding Booking Service section for more detailed information on how the RAWFIE user level booking process works.

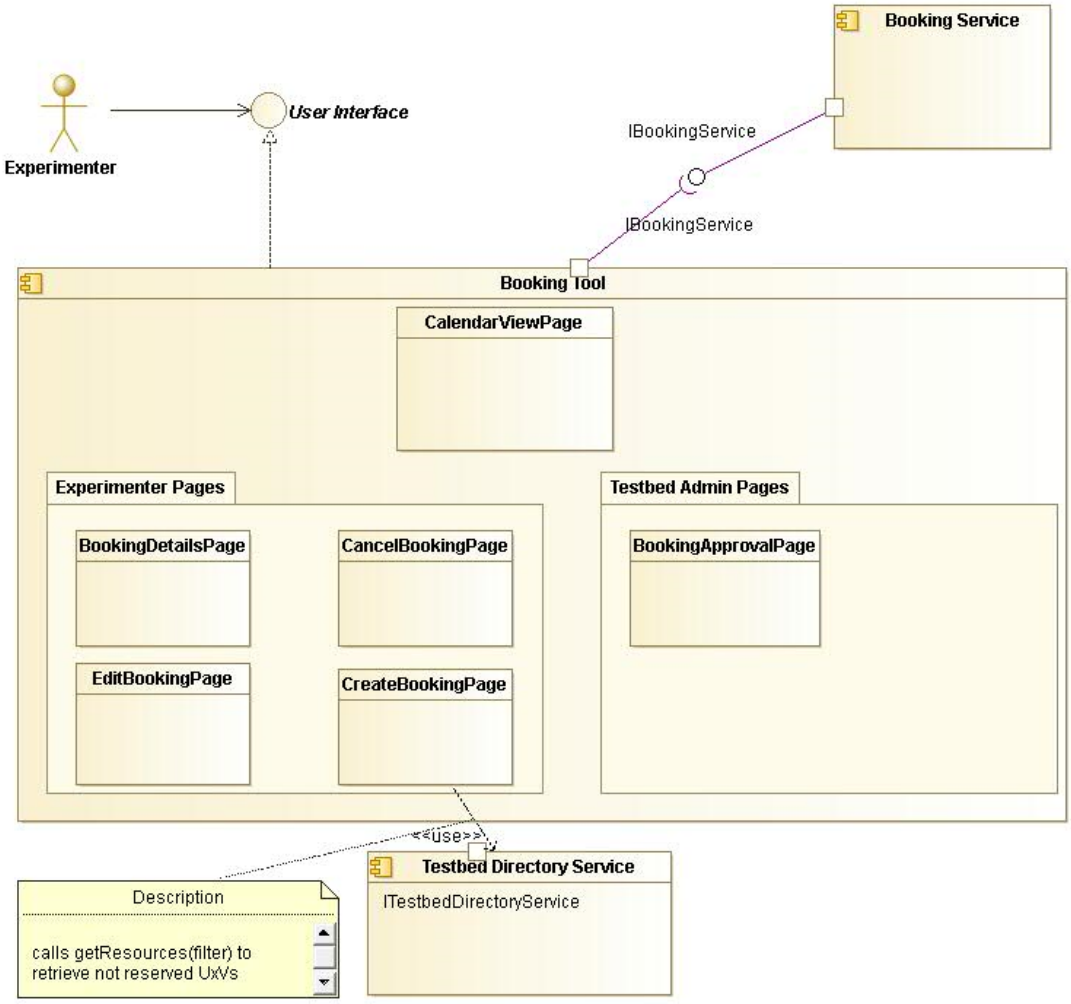

Figure 5: Booking Tool - Class diagram

Interactions and relationships with other components

Provided Interfaces

- Web portal GUI: Used by the users (Experimenter)

Required Interfaces

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

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.

Component requirements as identified in D3.2

ID (Priority)	Description	Requirement Mapping
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 experiment and/or per UxV resource	The EDL model will offer the necessary terminology
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

Responsibilities

The main responsibilities of the Experiment Authoring Tool are:

- Support for experiment definition through the Experiment Definition Language (EDL).
- Provision of a textual EDL editor.
- Provision of a visual EDL editor.
- Support for the textual and visual editor synchronisation.
- Support of typical file management commands like saving, opening, etc.
- Provision of hooks to the compiler, the validator and the experiment launcher.
- Short-term launching to start an experiment manually

Operations and attributes

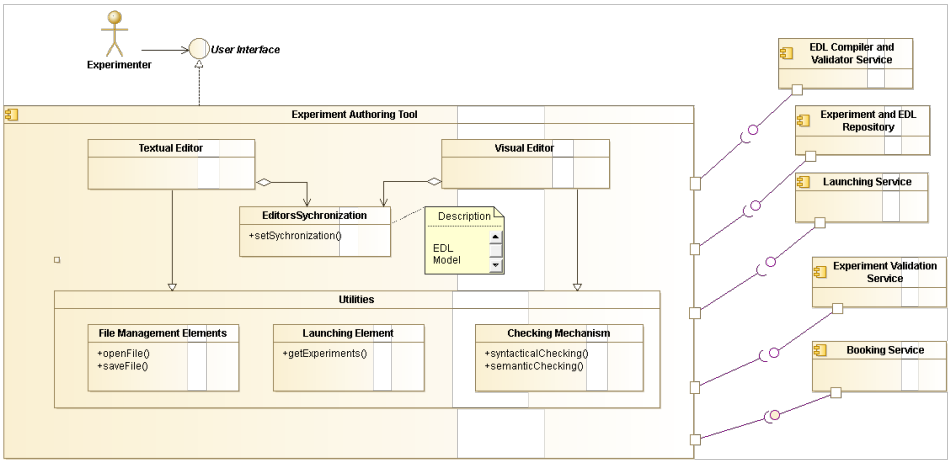

Figure 6: Experiment Authoring Tool - Class diagram

Open and edit an EDL script

1. User accesses the Experiment Authoring Tool through the web GUI
2. User clicks on the File Management Element to open a saved EDL script
3. User uses the textual editor to edit and update the script
4. Through File Management Element user saves the modifications to EDL repository

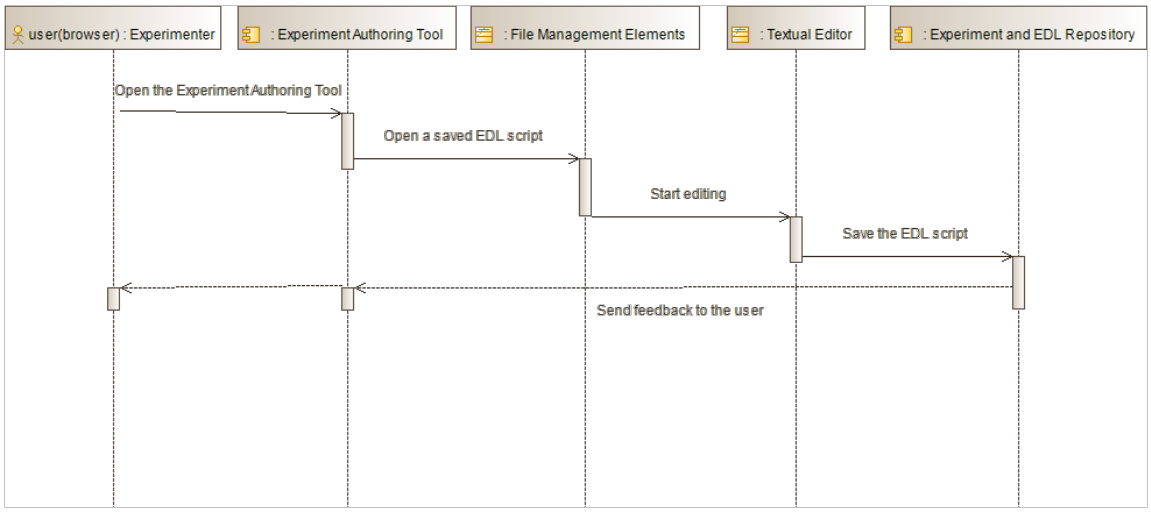

Figure 7: Experiment Authoring Tool – Open and edit an EDL script

Create and validate an EDL experiment

1. User opens the Textual and Visual editors to define an EDL experiment
2. User starts writing an experiment by using the EDL specific commands
3. Both editors will be synchronized
4. The compilation of the EDL script is performed by using the EDL Compiler and Validator service
5. The compilation results (errors and warnings) are displayed to the user through the editors
6. User corrects the errors
7. The user validates the experiment by using the Experiment Validation Service
8. The Experiment Validation Service returns through the editors the respective errors and warnings
9. The user corrects the errors
10. The compilation and validation is an iterative process that ends when all the errors being corrected
11. The experiment script is saved to the Experiment and EDL repository
12. The user launches the experiment manually through the launching element
13. The launching element triggers the respective launching service

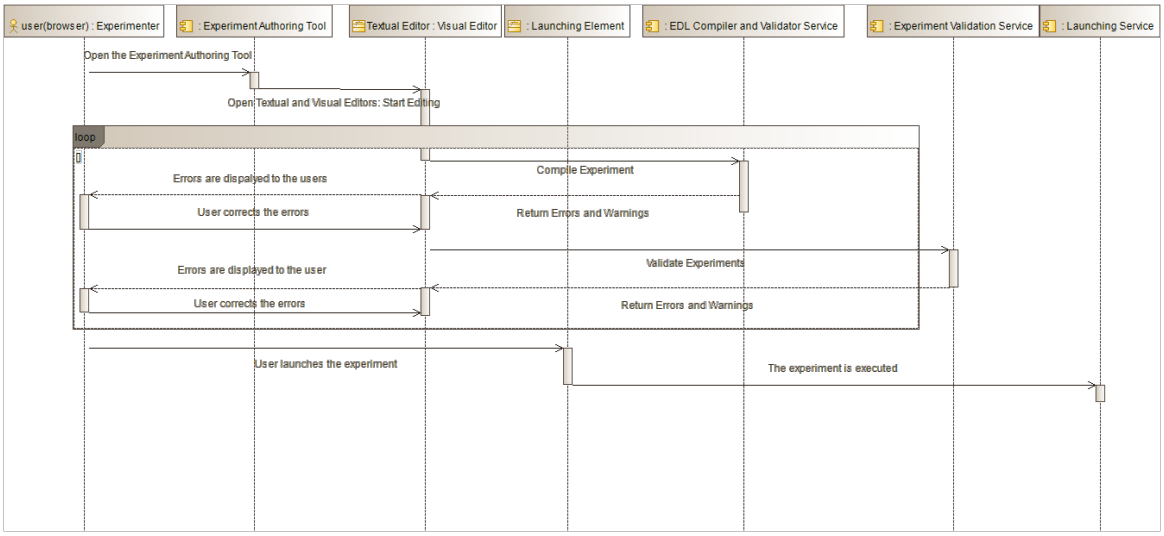

Figure 8: Experiment Authoring Tool - Create and validate an EDL experiment

Interactions and relationships with other components

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

4.1.6 Experiment Monitoring Tool

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

Component requirements as identified in D3.2

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	*To be defined during the second iteration*

	integrated	
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.

Responsibilities

The main responsibilities of the Experiment Monitoring Tool are:

- Show status of experiments (filtered by user rights)
- Show status of resources (filtered by experiments & user rights)
- Stopping/cancelling an experiment
-

Operations and attributes

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.

The ExperimentMonitoringController will collect and prepare the data for displaying in the web page. For this, it communicates with the System Monitoring Service, the Experiment Controller, the Experiments & EDL Repository and the Measurements, Results & Status Repository.

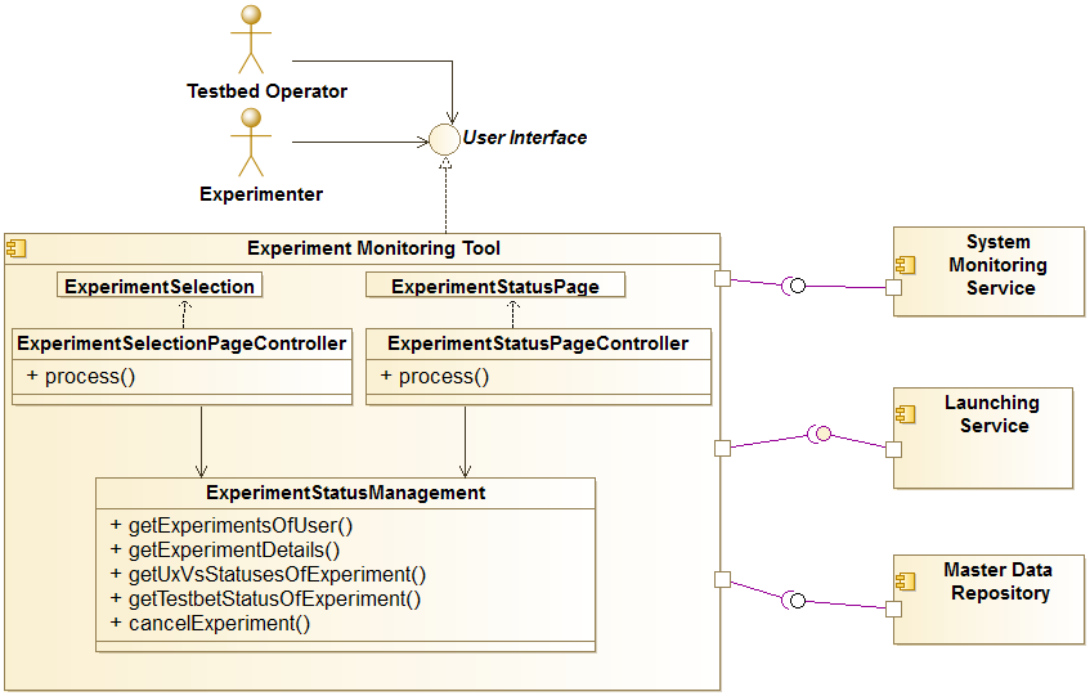

Figure 9: Experiment Monitoring Tool - Class diagram

Select experiment

1. The user opens the experiment selection page at the Experiment Monitoring Tool
2. The ExperimentSelectionPageController asks ExperimentStatusManagement for the list of experiments of the current user
3. ExperimentStatusManagement loads the experiment list from the Master Data Repository

4. The ExperimentSelectionPageController updates the ExperimentSelectionPage with the list
5. ExperimentSelectionPage is shown to the user from where he selects an experiment to get more details
6. The user selects an experiment and the status is loaded (see sequence "Get experiment status")
7. The ExperimentStatusPage is shown to the user.

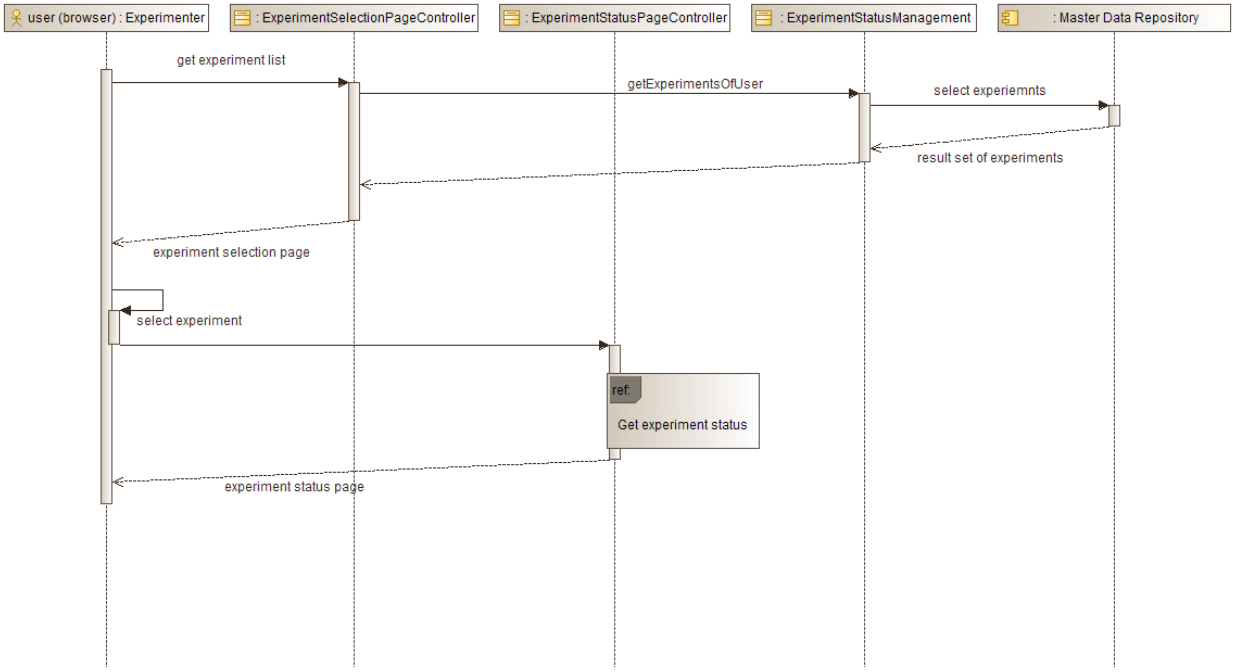

Figure 10: Experiment Monitoring Tool – Select experiment

Get experiment status

1. The user requests the status of the experiment
2. The ExperimentStatusPageController requests the data from the ExperimentStatusManagement
3. ExperimentStatusManagement queries the experiment details
    a. Static details of the experiment are loaded from the Master Data Repository
    b. Status of the experiment are loaded from the Master Data Repository
4. The UxVs statuses are queried from the System Monitoring Service
5. The Testbed statuses are queried from the System Monitoring Service
6. ExperimentStatusPage is updated with the collected data and shown to the user

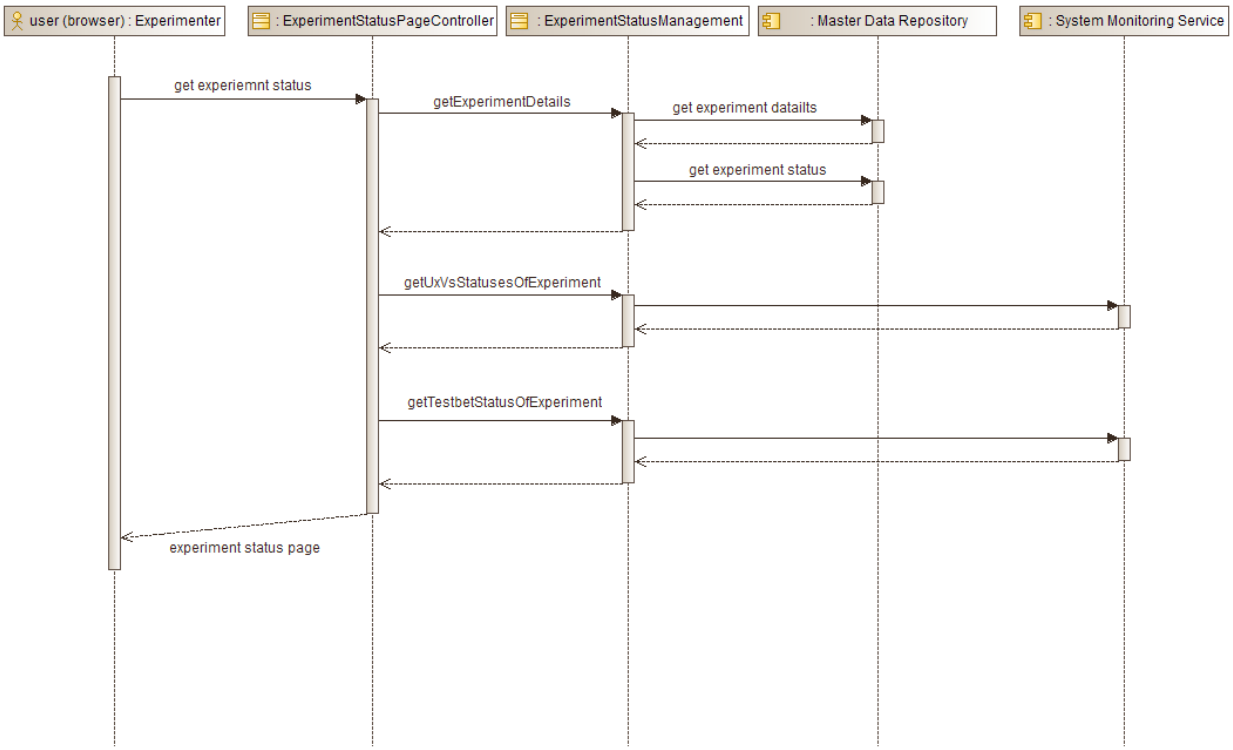

Figure 11: Experiment Monitoring Tool – View experiment details

Cancel experiment

1. On the experiment status page, the user clicks on the "Cancel" button
2. The ExperimentStatusPageController calls the ExperimentStatusManagement
3. ExperimentStatusManagement forwards the request to the Launching Service that executes the necessary steps to cancel the experiment

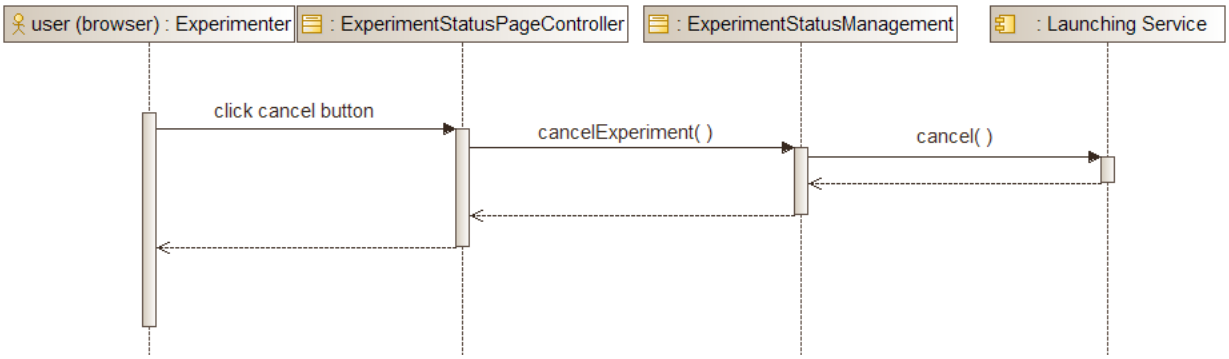

Figure 12: Experiment Monitoring Tool – Cancel experiment

Interactions and relationships with other components

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.
- Experiment Controller:
  To cancel an experiment the Experiment Controller is called to execute the necessary steps.

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)

Component requirements as identified in D3.2

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

Responsibilities

The main responsibilities of the System Monitoring Tool are:

- Show detailed status of RAWFIE system infrastructure (for administration)
  - Highlight potential problems
- Show simplified status dashboard (for normal users)
- Configure System Monitoring Service (for administration)
  - Monitoring parameters
  - Notifications on potential problems (e.g. via email)

Operations and attributes

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.

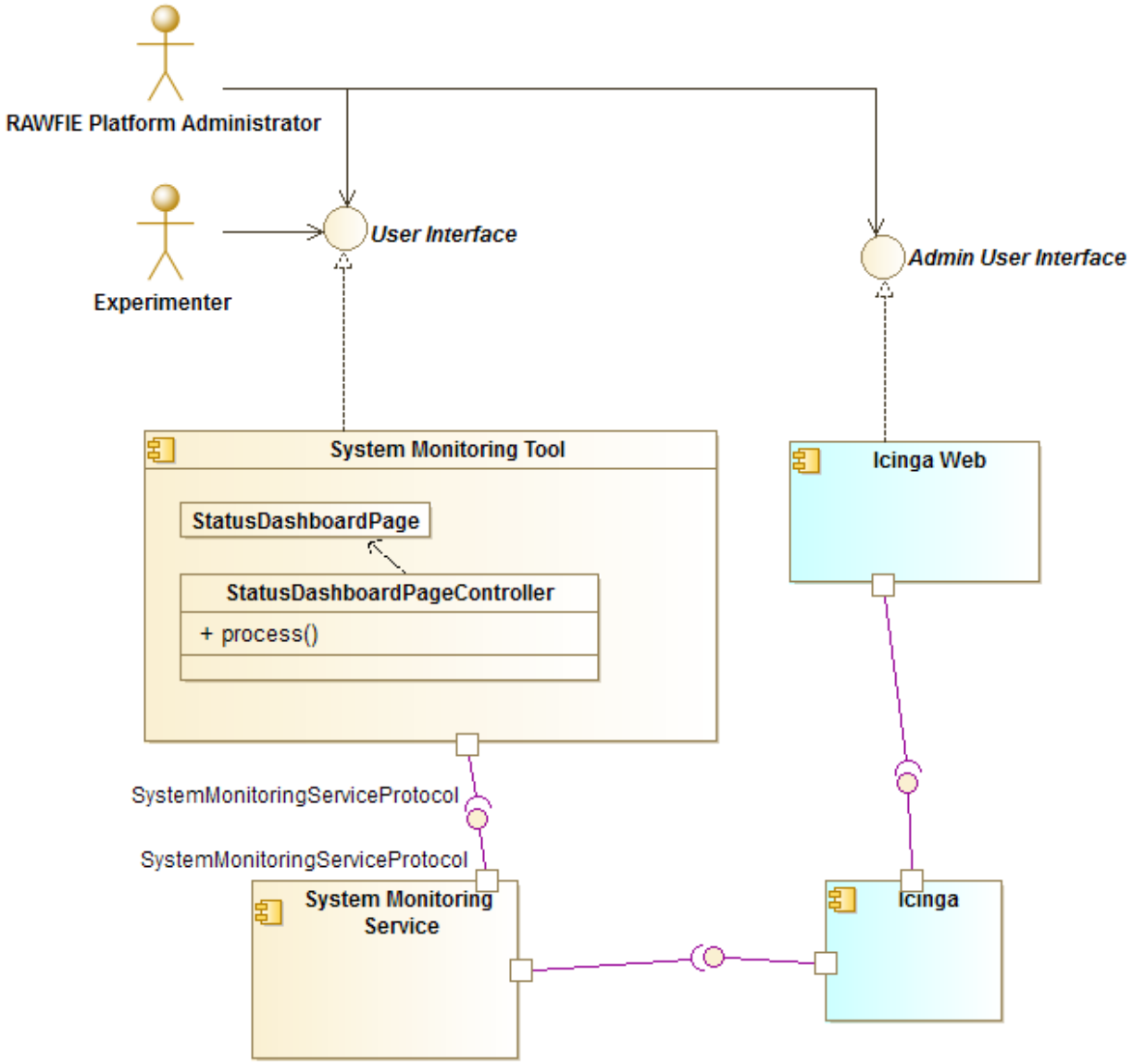

Figure 13: System Monitoring Tool - Class diagram

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

Interactions and relationships with other components

Provided Interfaces

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

Component requirements as identified in D3.2

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 do at the second integration 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 at the second integration 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 do at the second integration of development

Responsibilities

The main responsibilities of the UxV Navigation Tool are:

- Guides the vehicles using a turn based navigation mechanism
- Collects data from their equipped sensors.

Operations and attributes

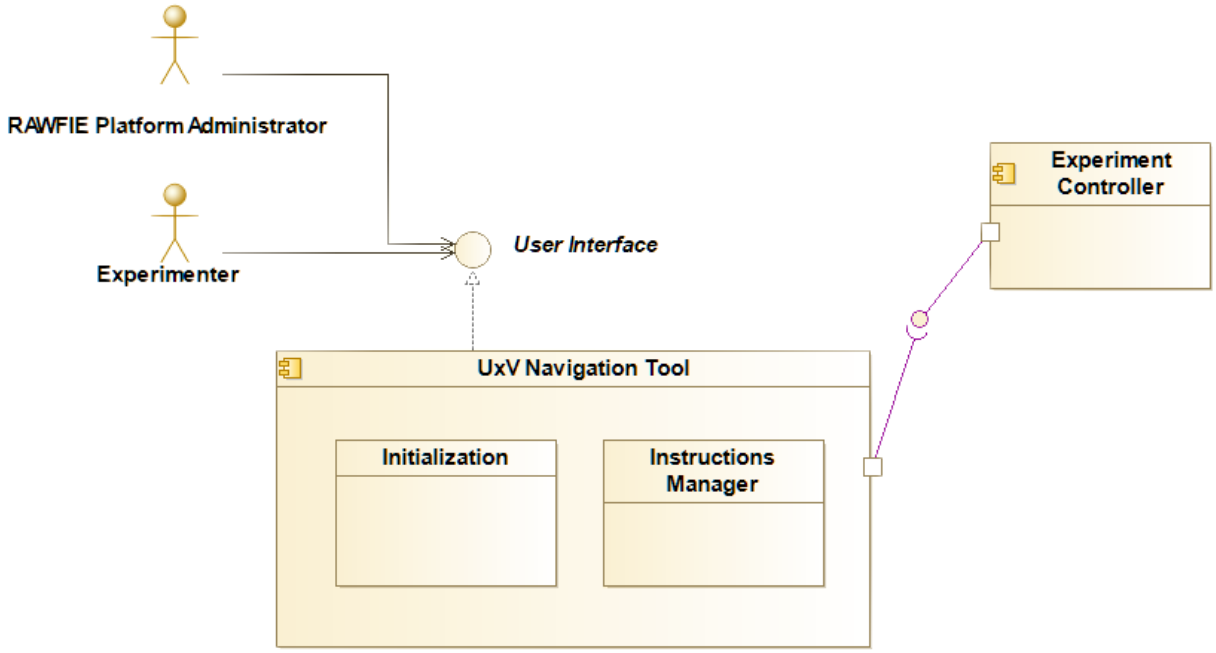

Figure 14: UxV navigation tool – Class diagram

Initialization of the Experiment:

1. Through the user interface, the experimenter or RAWFIE administrator specifies the required details of the experiment, providing information regarding the number of the vehicles, the type of the units as well as information regarding the required sensors.
2. The initialization class prepares an appropriate file and informs the Experiment Controller about the requirements.
3. The Experiment Controller interacts with the UxV Navigation Tool and informs the component about the availability of the equipment and the feasibility of the experiment.

Remote Control

After the initialization of the experiment, the virtual controller will allow the experimenter to guide the vehicles using a turn based navigation mechanism and to collect data from their equipped sensors.

1. Through the provided interfaces, users specify the next desired location for each unit.
2. The instructions manager class translates these instructions into a JSON file and transmits this file to the Experiment Controller.
3. When all the vehicles reach their desired position, the UxV Navigation Tool is ready to accept a new set of instructions.

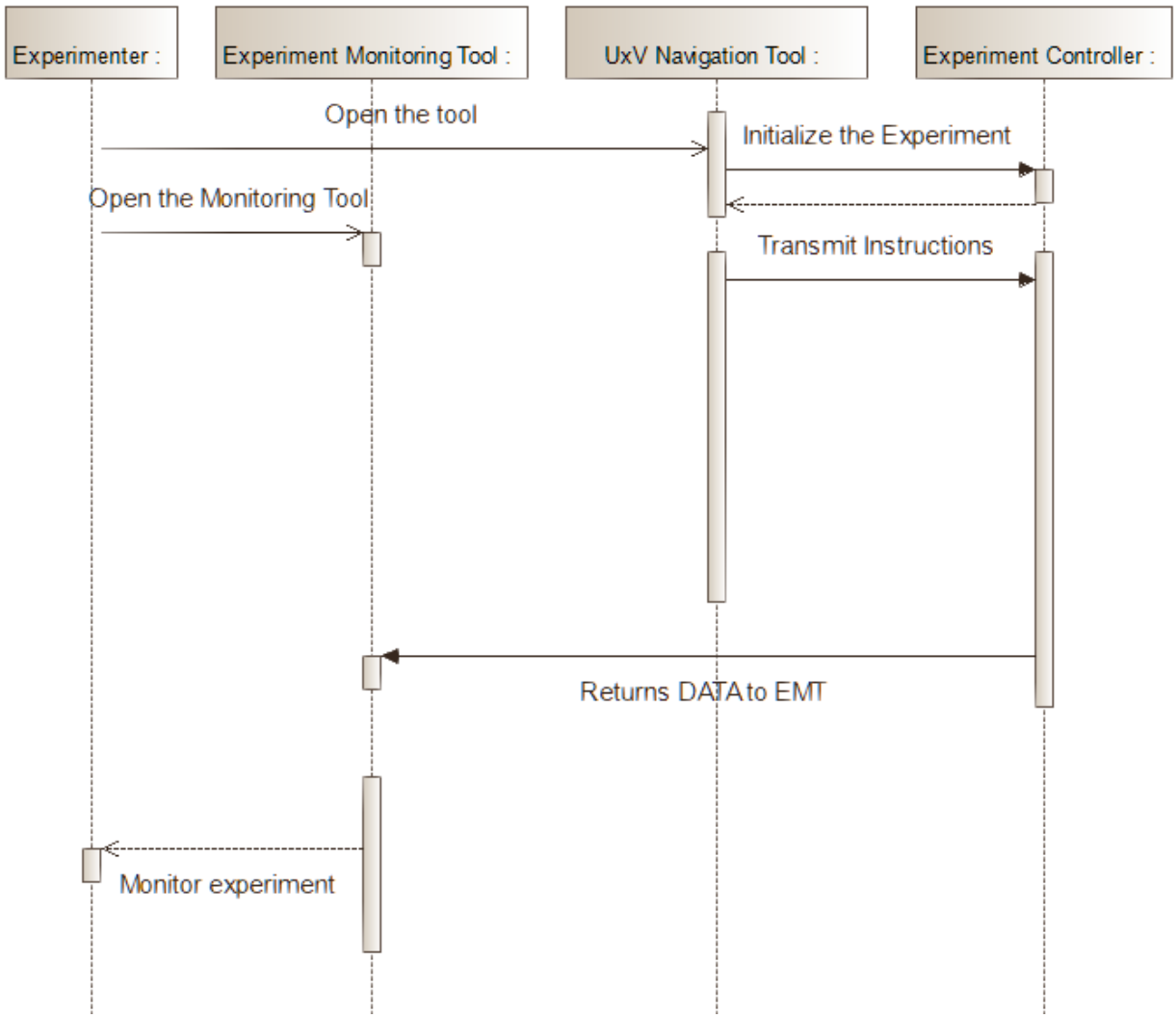

Figure 15: UxV navigation tool – High level sequence diagram

Interactions and relationships with other components

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.

4.1.9 Visualisation Tool

The Visualisation Tool provides visualisation of the geospatial data of a running experiment. Additionally 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.

Component requirements as identified in D3.2

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
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

Responsibilities

The main responsibilities of the visualisation tool are:

- Visualise a running experiment by presenting the UxVs on a map and show their movement
- Provide an option to add additional layers on top of the current map in order to show/hide/highlight different information during the execution of the experiment
- Provide an option to add/remove different widgets with information about the experiment, the UxVs, the landscape and others
- Plot a summary when the experiment is over, showing statistics about the experiment

- Change camera position by setting different point of view, angle or camera movement

Operations and attributes

In this part of the description an abstract high level class diagram of the visualisation tool (VT) is presented that will be in the front end tier, and will work closely with visualisation engine (VE) that will reside in the middle tier.

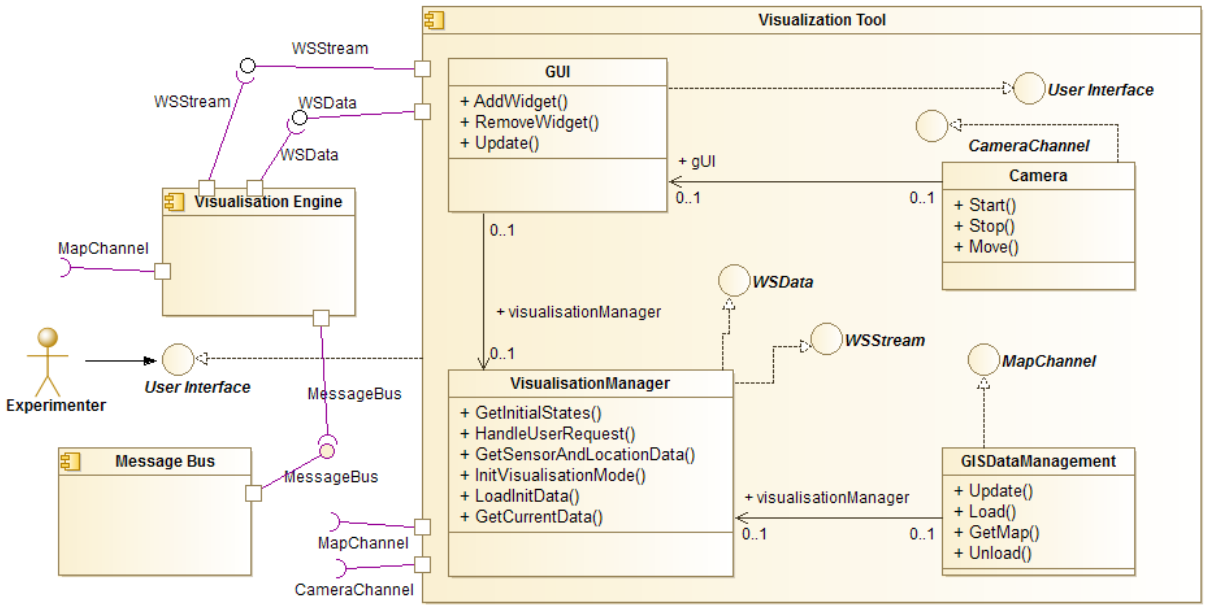

Figure 16: Visualisation Tool - Class diagram

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

The sequence diagrams below provide information about the data flow and the interaction between the different components.

Start the Visualisation Tool:

1. The experimenter chooses from the web portal to start the VT
2. The VT registers at the VE and is ready to receive information about a running or past experiment
3. The experimenter can choose to start the VT or to not use it during an experiment and decide to visualize it later on.

User updates the desired location of the UxV

1. The Experiment Controller sends the updated waypoints to the Engine Controller
2. The waypoints are converted to layers by the GISServer and the Database and the new layer is sent to the VT
3. The new layer is plotted by the Renderer
4. The experimenter can see on the map what (s)he defines as next position of the UxV in the UxV Navigation Tool and how the UxV will get there by visualising the waypoints.

The experimenter adds/removes/updates widgets

1. The experimenter directly edits the widgets in the browser window. This generates a request to the GUI to update the widgets
2. The information about the new widgets is sent to the renderer, which plots them on the screen
3. The user can adjust the information on the screen based, on the requirements and the current scenario.

The experimenter changes the position of camera

1. The experimenter updates the position of the camera directly in the browser window, which generates a request in the Camera
2. The Camera updates the positions and sends the new parameters to the Renderer, which then adjusts the positon of the experiment's camera.
3. The user can then match the view of the VT to hers/his requirements.

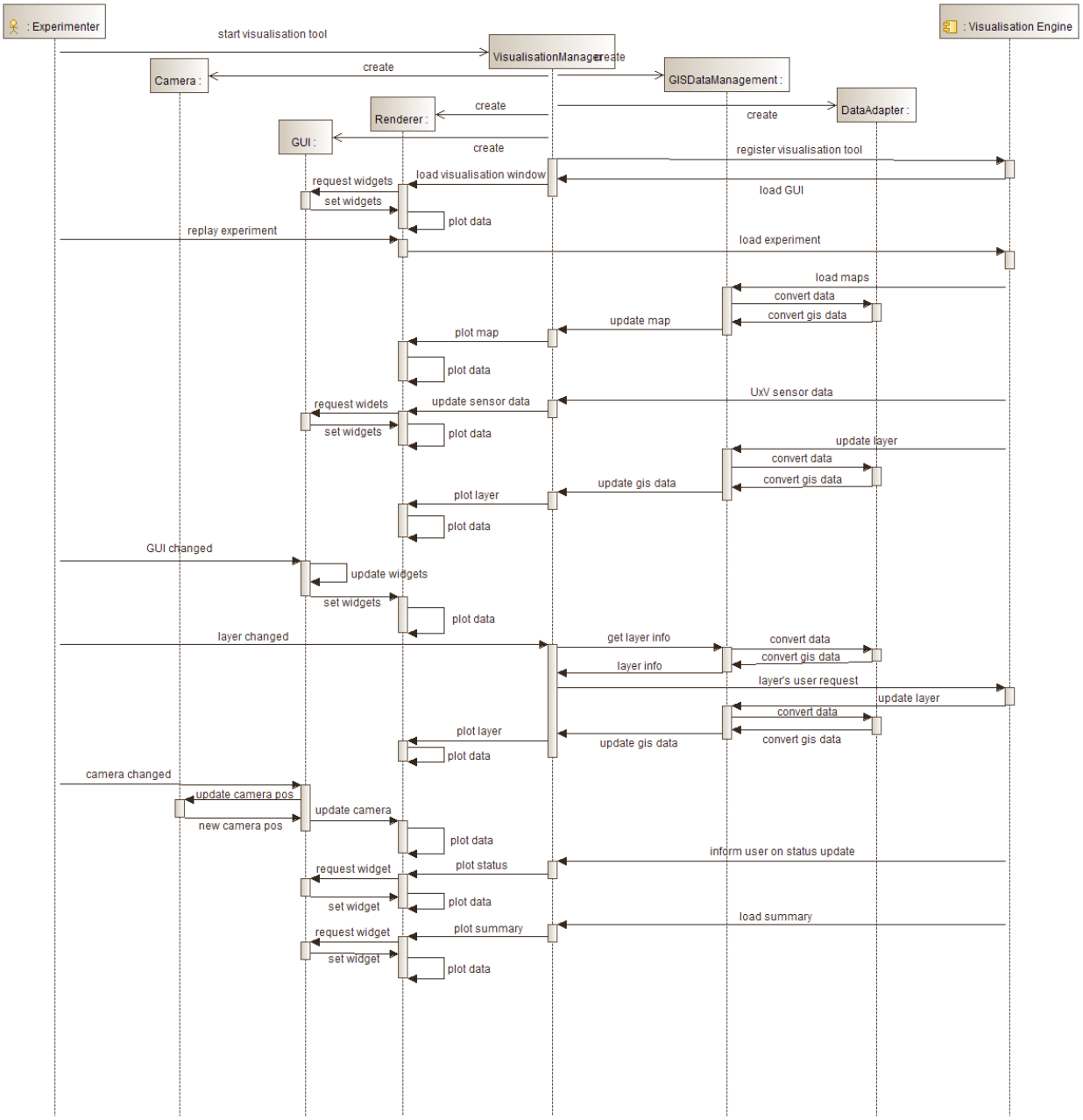

Figure 17: Visualisation Tool - Sequence diagram

Interactions and relationships with other components

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.

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.

4.1.10 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 as well as being able to access data sets stored in the different data repositories. By having access to the available data through the Data Analysis Tool, the user can browse the available data and select the tables/data structures on which he wants his analytics subroutines to be executed. The latter are also designed through the Data Analysis Tool since it acts like a user interface of the Data Analysis Engine. Through it the user not only gains access to the data coming from the data source/sources, but also enables access and selection of results from previously executed jobs, which can afterwards be involved in other data analysis jobs.

Component requirements as identified in D3.2

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.

Responsibilities

The main responsibilities of the Data Analysis Tool are:

- Browse data and results originating from previous analytics tasks
- Select data subject to further analysis
- Enable the design of data analysis jobs to be relayed to the Data Analysis Engine which will perform the execution of the data analytics tasks specified.

Operations and attributes

The class diagram of Data Analysis Tool is presented in Figure 18.

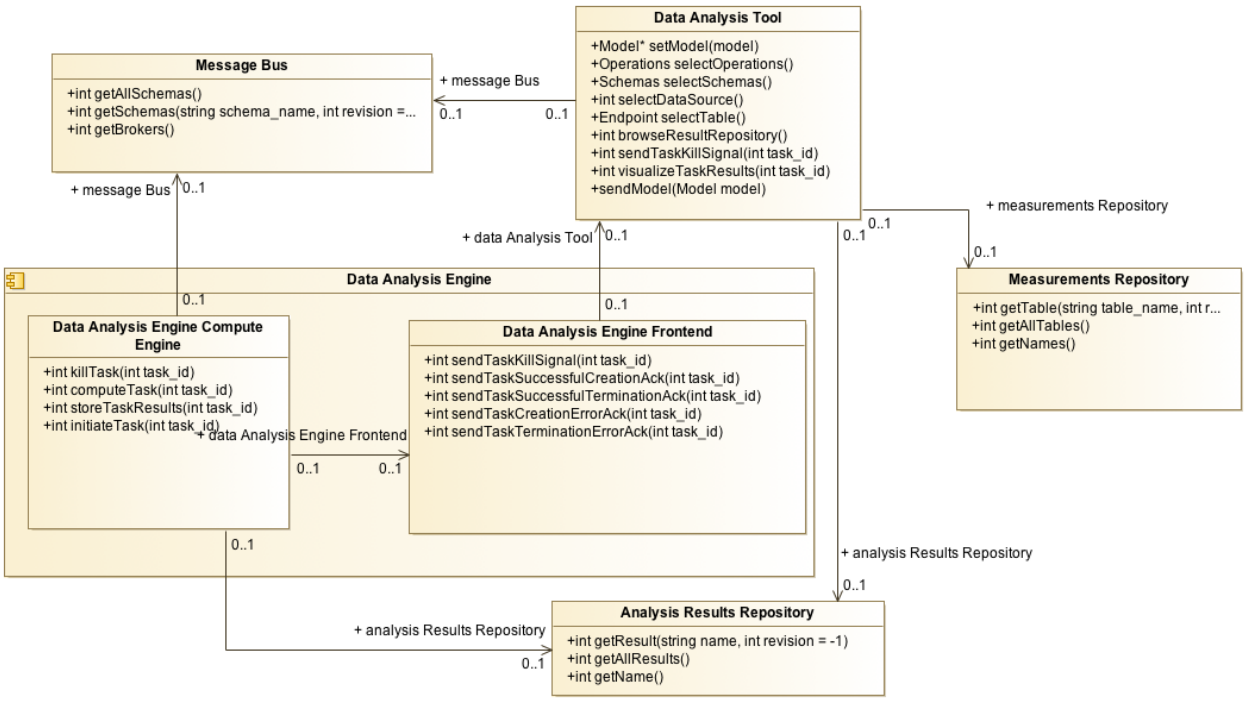

Figure 18: Data Analysis Tool – Class diagram

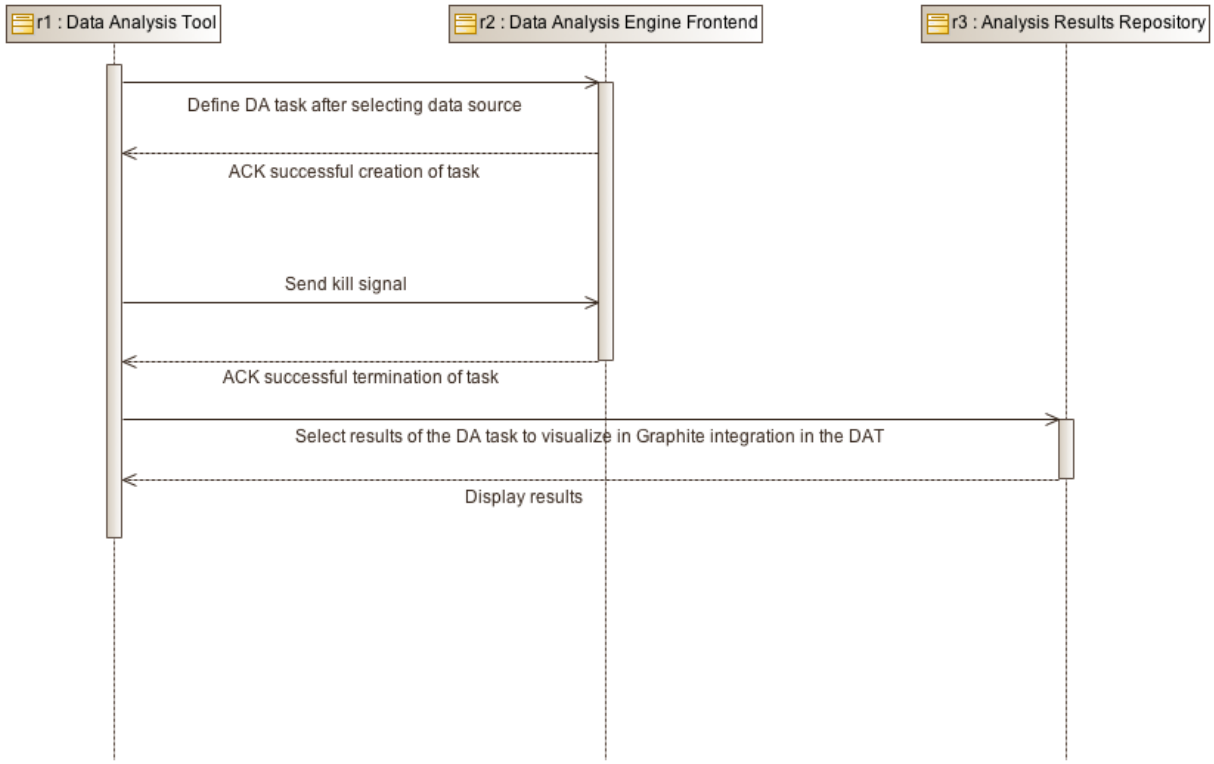

Figure 19: Data Analysis Tool – Sequence diagram involving the Data Analysis Tool in the case of a stream analytic task.

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 Diagrams of the Middle Tier components, showing the servers and the execution environments for the deployment of Middle Tier services, together with their interaction with the Web Portal components, the Testbed components, the GIS Server, and the Data Repositories, is shown in Figure 20 below.

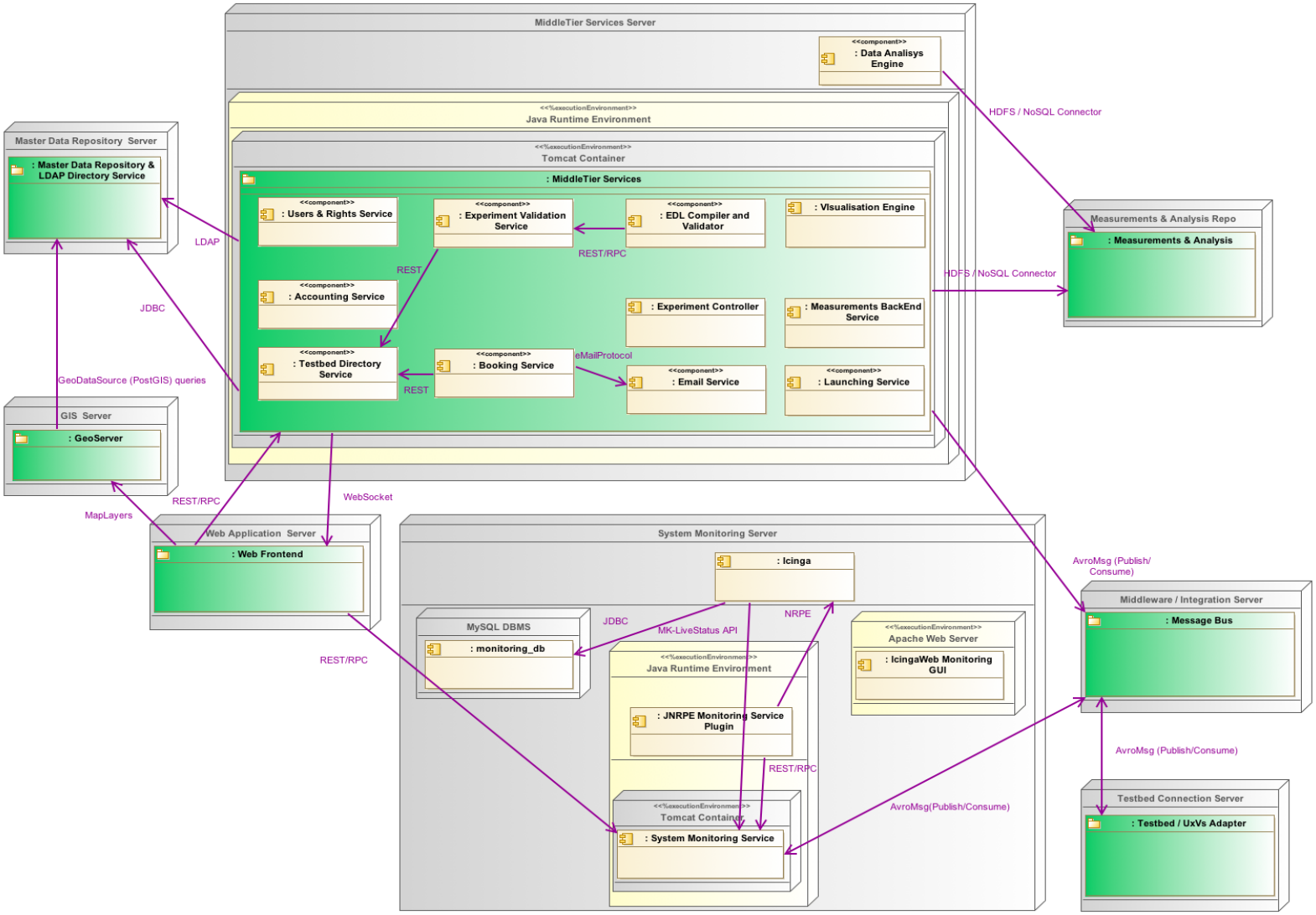

Figure 20: Middle Tier Components – Deployment / Components Diagram

4.2.2 Testbed Directory Service

Component requirements mapping, as identified in D3.2

ID (Priority)	Description	Requirement Mapping with component's functionalities
PT-DIR-S-001 (HIGH)	The Testbed Directory Service shall provide access to information on all Testbeds registered in RAWFIE	getTestbeds REST interface provides access to information about testbeds registered in RAWFIE
PT-DIR-S-002 (MEDIUM)	The Testbed Directory Service should provide access to information on all Testbeds registered in RAWFIE according to predefined filters	getTestbeds REST interface provides access to information about a list of registered testbeds by their ids getTestbeds REST interface will provide access to information about a list of testbeds by other predefined search criteria such as: • available resources type/s • location
PT-DIR-S-003 (HIGH)	The Testbed Directory Service shall provide access to information about available resources (UxVs) belonging to the testbeds registered in RAWFIE	getResources REST interface provides access to information about: • all resources • resources belonging to a list of testbeds (by testbed ids)
PT-DIR-S-004 (MEDIUM)	The Testbed Directory Service should provide access to information on available resources (UxVs) belonging to the testbeds registered in RAWFIE, and according to predefined filters	getResources REST interface will provide access to: • information about a list of resources by other predefined search criteria and capabilities such as: o resources type/s o resources status o sensor / measurements types available o other to be still defined The interface will require to define the search criteria and search value as input parameters
PT-DIR-S-005 (HIGH)	The Testbed Directory Service shoud provide the possibility to register new testbeds in the RAWFIE platform,as well as to unregister	createTestbed REST interface allows the registration of a new Testbed, by providing in input the Testbeds information structure deleteTestbed REST interface provides the possibility to delete a Testbed, by its testbed

	(delete) testbeds from the platform	id
PT-DIR-S-006 (MEDIUM)	Some basic query capabilities should be provided	searchResource REST interface will provide the possibility to search for a resource or a list of resources according to given predefined search criteria such as: • testbed id • resources type/s • resources status • sensor / measurements types available • keywords in description • other to be still defined searchtestbed REST interface will provide the possibility to search for a testbed or a list of testbeds according to given predefined search criteria such as: • testbed ids • available resources type/s • keywords in description • other to be still defined
PT-DIR-S-007 (HIGH)	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	createResource REST interface allows the registration of a new Resource, by providing in input the Testbed id and the resource information structure deleteResource REST interface provides the possibility to delete a resource

The Testbed Directory Service is basically 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.

Responsibilities

The main responsibilities of the Testbeds Directory Service are:

- Provide access to the information (about testbeds and associated resources), contained in the corresponding Master Data Repository, to other components
- Provide the available testbeds list and their status (free, booked, in use, and so on).
- Show the available resources within a given testbed and at their status (free, booked, in use, not operational, and so on)
- Provide the description and characteristics of the testbeds (name, location, available resources)

- Provide the description and characteristics of each resource from a testbed (name, type of resource such as USV, UGV, UAV, and supported sensors)
- Provide the testbeds and resources capabilities in terms of available technologies, sensors, and corresponding tests
- Execute queries to the Master Data Repository, based on searching capabilities through specific filters
- Allow updates of resources and testbeds information

The functionalities provided by the component, which take into account the list of requirements from deliverable D3.2, are highlighted in the following.
A UML class diagram is used to specify the provided and required interfaces to and from other RAWFIE components, as well as the most important operations and attributes of the Testbed Directory Service classes.

Operations and attributes

With the aim to represent the structure of the Testbed Directory Service component, a class diagram is provided below, showing the associations, operations and responsibilities of the classes. The *ListingResource* and *StoringResource* classes implement the required interface (REST API), in charge of receiving all the incoming requests from the Web Portal components, e.g. from the Resource Explorer Tool.
The above classes make use of the *StorageService* class, which basically creates instances of required testbeds and/or resources, to manipulate them later on and produce the necessary input in order to build the queries to the repository. In turn, it calls the methods provided by the *RepositoryHandler* class. This latest one, is the class responsible of realising the actual communication with the Master Data Repository (i.e., the PostgreSQL/PostGIS database), through the implementation of a JPA (Java Persistence API) interface. So it is used for final information retrieval, update and registration of information on testbeds and resources.
All internal classes make use of the *TestbedType* and *ResourceType* Java objects, for manipulating the structure of testbeds and resources.

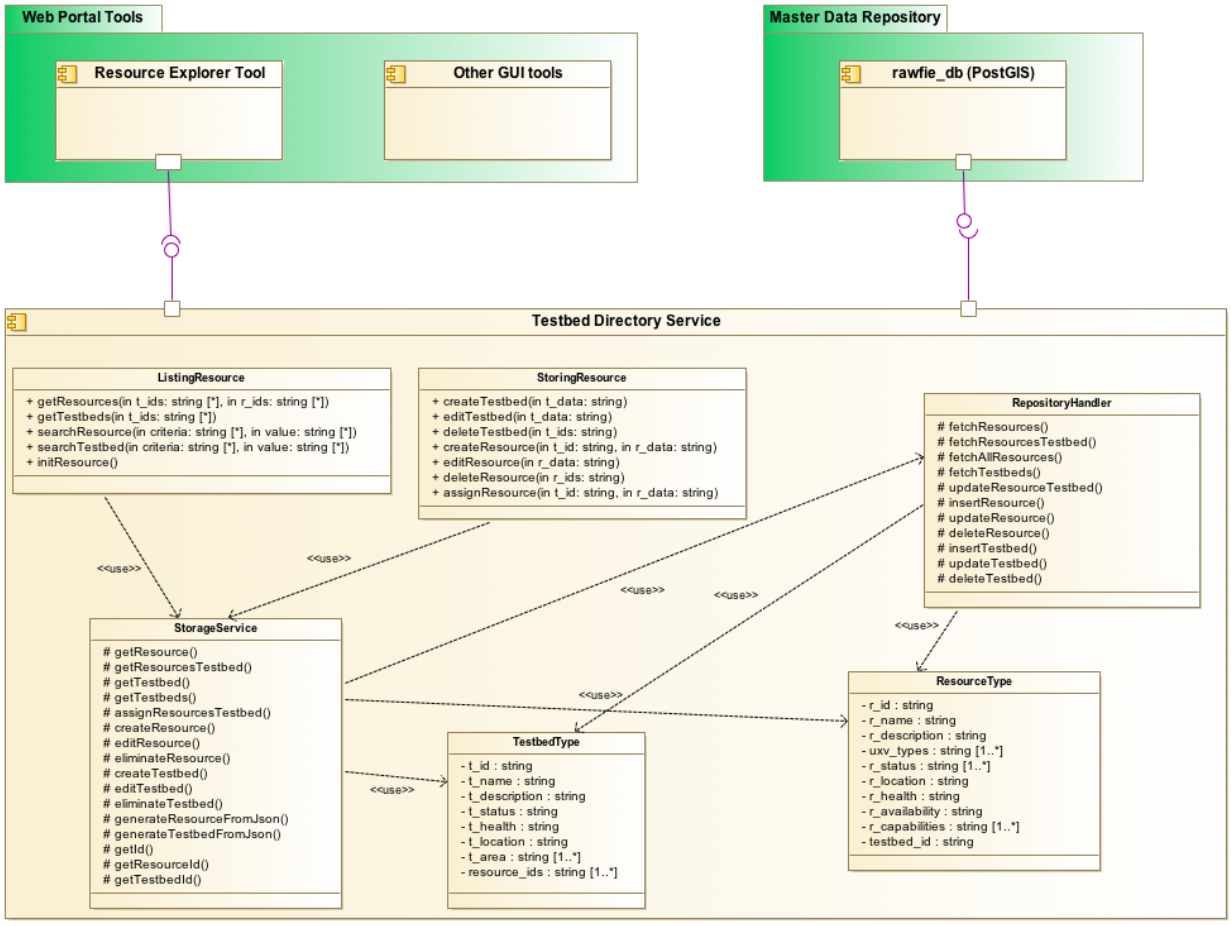

Figure 21: Testbed Directory Service – Class diagram

Below some examples of the internal interactions between Testbed Directory Service classes / modules are shown, by the mean of UML sequence diagrams.

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
2. In this case, the *searchResource* REST interface, implemented by the *ListingResource* class is called, by providing in input the search criteria and values (e.g. UxVs type, sensors types)
3. The REST interface method uses the *getResource* method of the *StorageService* class, which in turn fetches the information from the Master Data Repository, through the *RepositoryHandler* class, providing a JPA (Java Persistence API) interface to the database

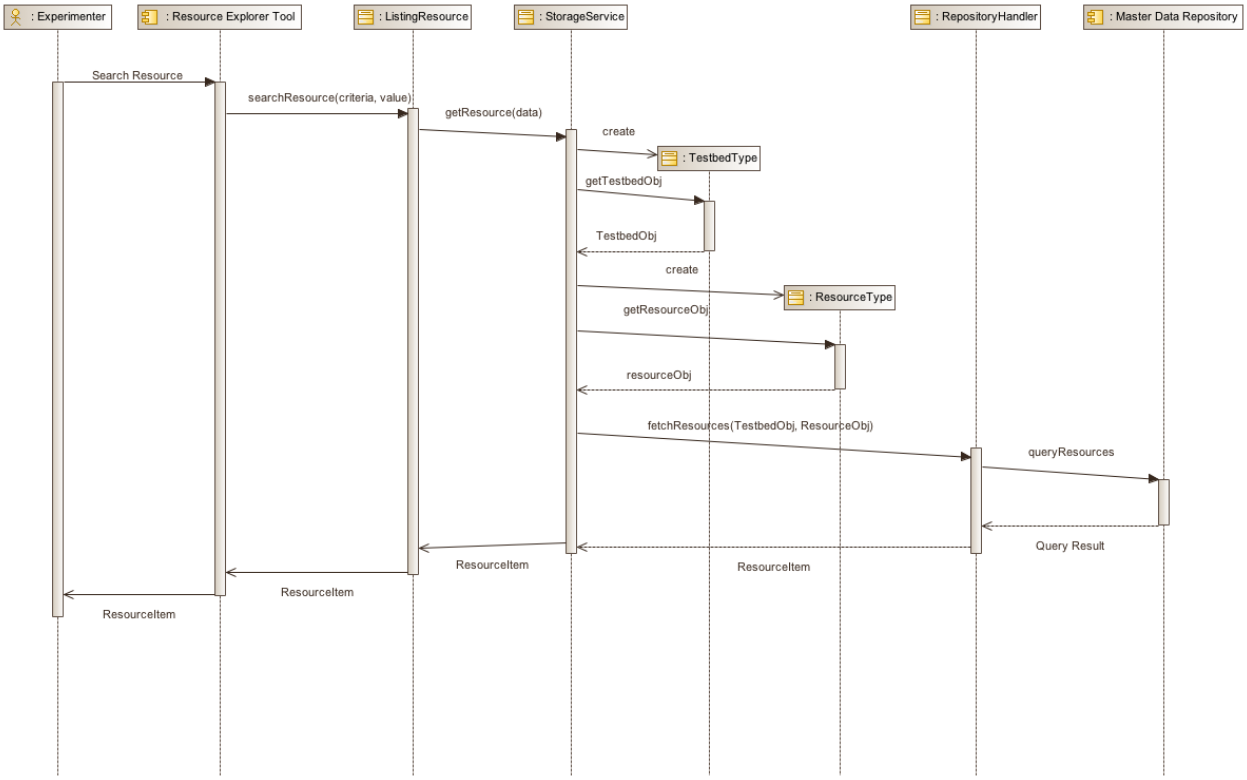

Figure 22: Search Resource internal Sequence diagram

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 *createTestbed* REST interface, implemented by the *StoringResource* class is called, by providing in input the Testbed information structure
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, providing a JPA (Java Persistence API) interface to the database

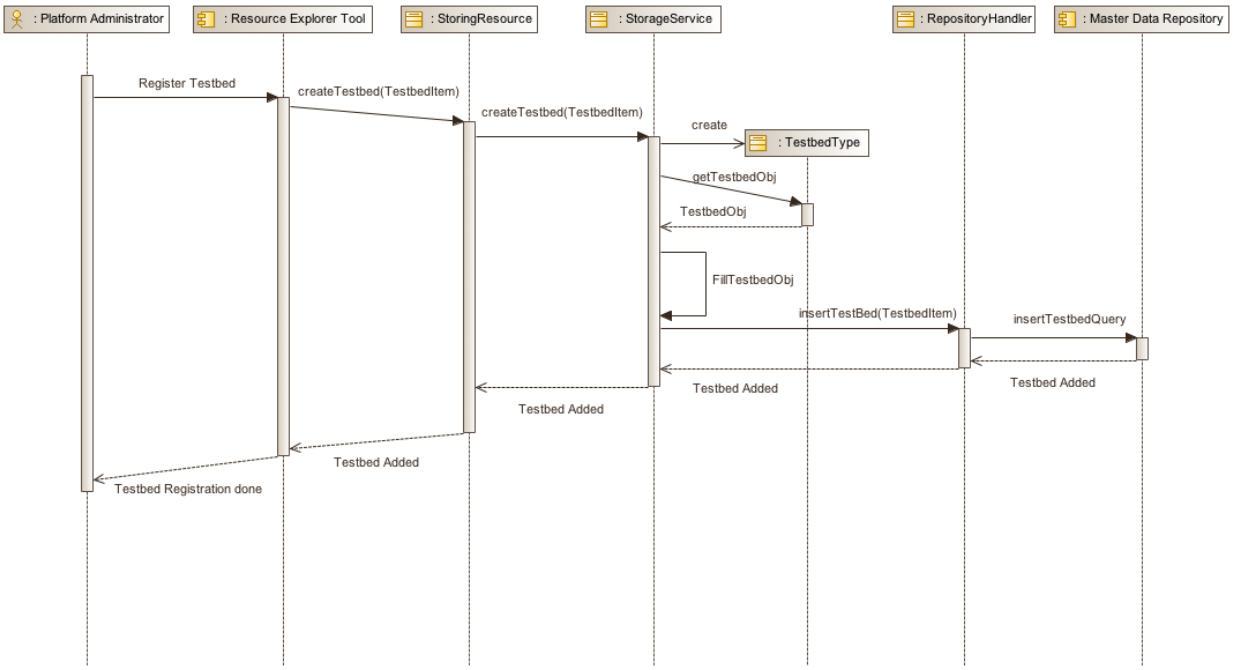

Figure 23: Testbed Directory Service – Register a new testbed in the platform

Add a new UxV device into a Testbed facility

1. The Testbed Operator starts with the process of the new resource registration into the given Testbed
2. In this case, the *createResource* REST interface, implemented by the *StoringResource* class is called, by providing in input the Testbed Id and the Resource information structure
3. The REST interface method uses the *assignResourcesTestbed* method of the *StorageService* class, which in turn inserts the new information in the Master Data Repository, through the *RepositoryHandler* class, providing a JPA (Java Persistence API) interface to the database

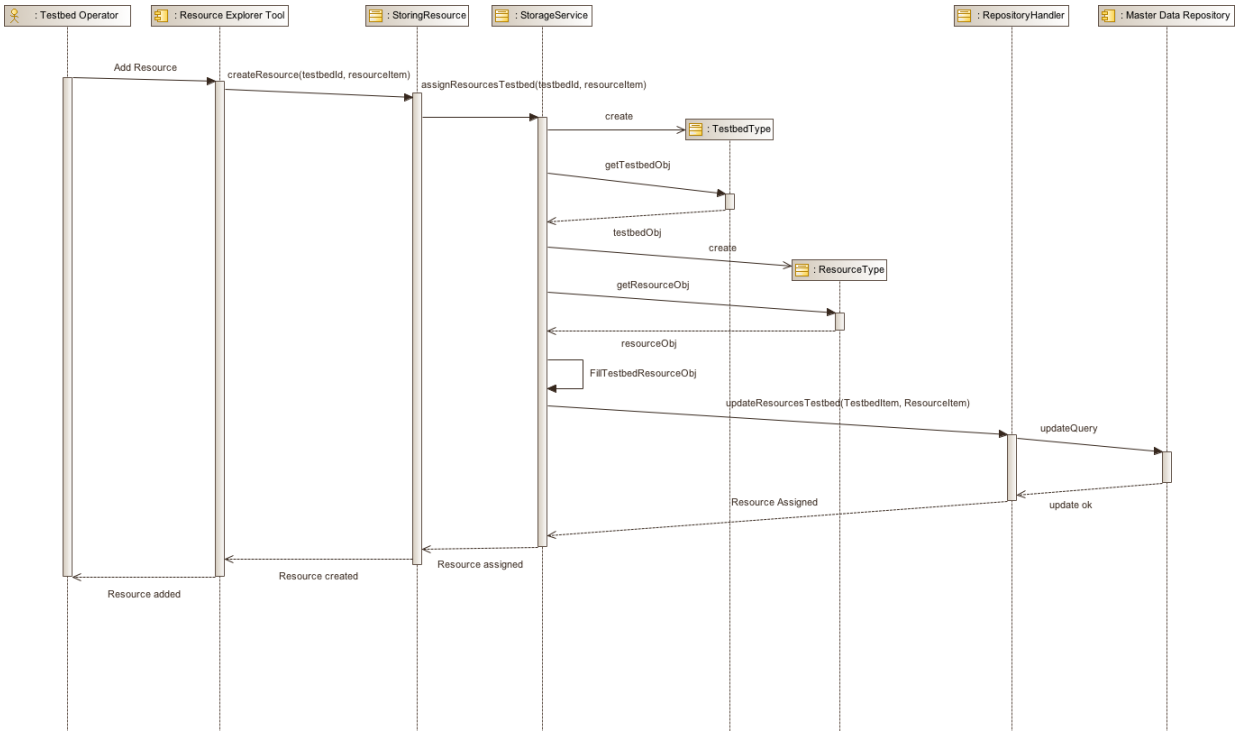

Figure 24: Testbed Directory Service - Add a new UxV device into a Testbed facility

Interactions and relationships with other components

The Testbed Directory Service interacts mainly with the Resource Explorer Tool and the Master Data Repository. It also interacts with the Booking Tool and the Experiment Validator Service.

Provided Interfaces

- The Testbed Directory Service component API is invoked by the Resource Explorer tool in order to perform the CRUD (Create, Read, Update, Delete) operations on testbeds and resources belonging to the RAWFIE platform.
- It is also utilized by the Booking Tool and by the Experiment Validator Service, to get access to information on testbeds and resources (UxVs).

Required Interfaces

- The Testbed Directory Service API will be in charge of executing the queries to the Master Data Repository for performing CRUD operations with testbeds and related UxV resources.

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.

Component requirements as identified in D3.2

ID (Priority)	Description	Requirement Mapping
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 (High)	Experiments defined via EDL shall be validated after their authoring	The Compiler & Validator provides this 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.

Responsibilities

The main responsibilities of the ECV are:

- It provides syntactic and semantic validation of each experiment workflow.
- It applies a set of constraints that should be met in order to have a valid experiment.
- It is capable of applying semantic checking for nodes communication, spatio-temporal management, sensing and data management.
- It may perform code generation in the appropriate format in order to be uploaded into the RAWFIE nodes.

Operations and attributes

Figure 25**Error! Reference source not found.** provides a class diagram that depicts the internal architecture of the ECV. 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) undertakes 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.

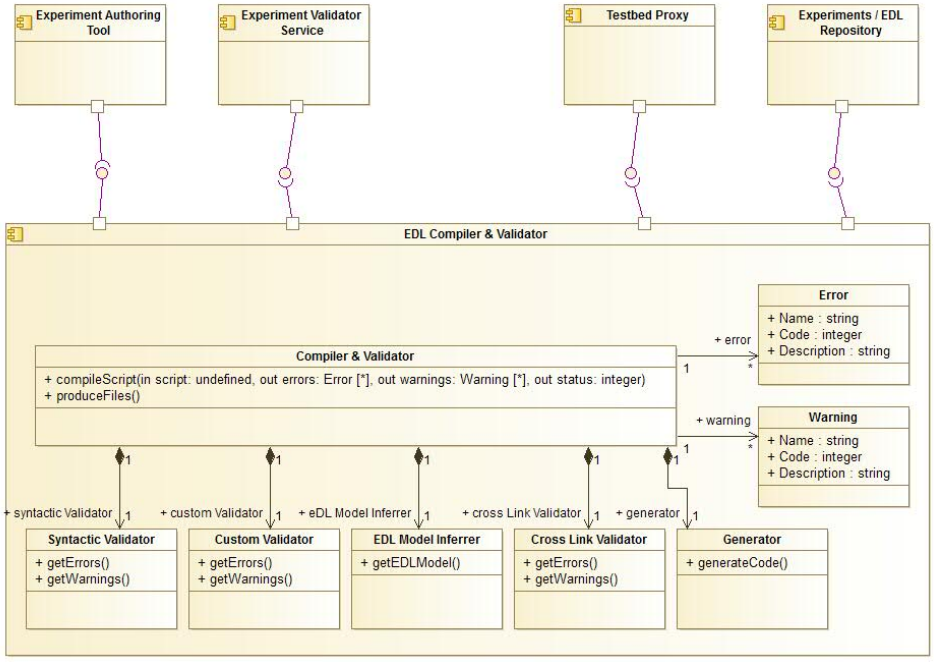

Figure 25: Experiment Compiler – Class diagram

The internal interactions of the ECV classes / modules are given in Figure 26**Error! Reference source not found.**. The Experiment Authoring Tool gives the experimenters the opportunity to trigger the ECV. Afterwards, the ECV continuously adopts the functionalities provided by the available validators to check the correctness of each experiment. The final step, after successive iterations in correcting the EDL script, is the generation of the appropriate files and code.

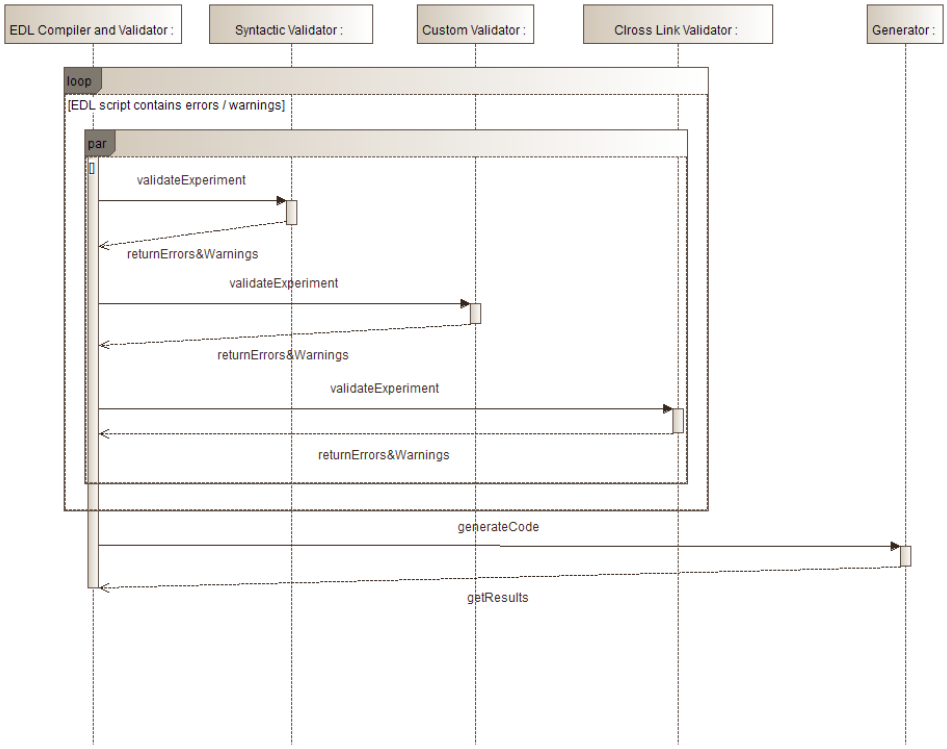

Figure 26: Experiment Compiler - Sequence diagram

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.

Component requirements as identified in D3.2

ID (Priority)	Description	Requirement Mapping
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.

Responsibilities

The main responsibilities of the EVS are:

- Provide semantic validation for each experiment at a specific testbed.

- Check the fulfilment of a set of constraints defined by experts for the specific testbed.
- Handle security & safety issues e.g., collision avoidance, and other non functional (qualitative) aspects of each experiment. Efficient communications and control of the UxVs team will be performed in order to increase the performance of the system.
- Perform cross experiment validation in order to help in maximizing the performance of the RAWFIE framework.

Operations and attributes

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.

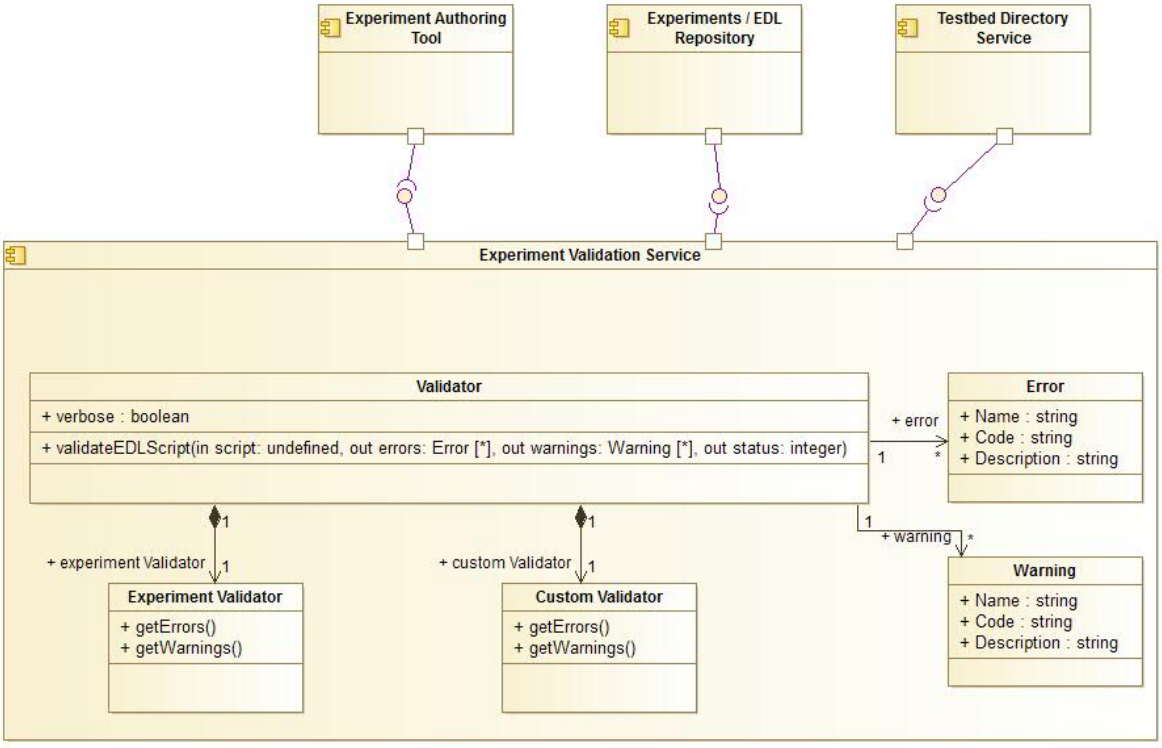

Figure 27: Experiment Validator – Class diagram

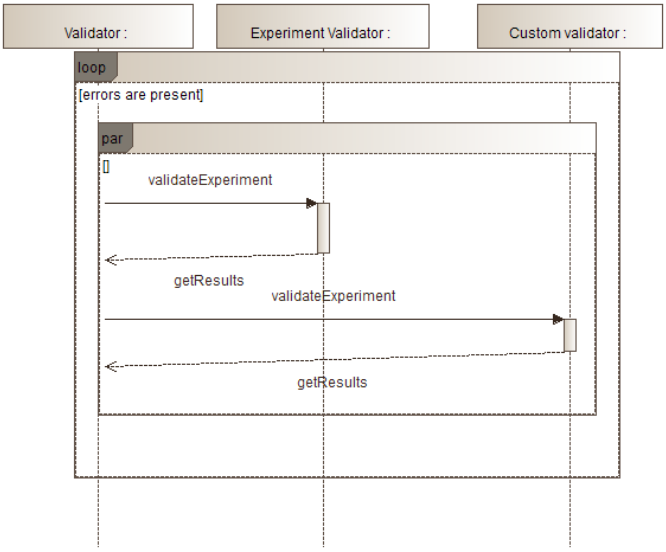

Figure 28: Experiment Validator - Sequence diagram

4.2.5 Users & Rights Service

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

Component requirements as identified in D3.2

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.

Responsibilities

The main responsibilities of the Users & Rights Service are:

- To check and authenticate users and interacting components
- To provide authorization services (check if a user/component is allowed to do a specific action)
-

Operations and attributes

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). 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 is 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 the following 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. Also, the Users & Rights Service will act as certification authority (CA): it will sign certificate signing request (CSR) of new users.

An additional ProxyService is provided for application 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.

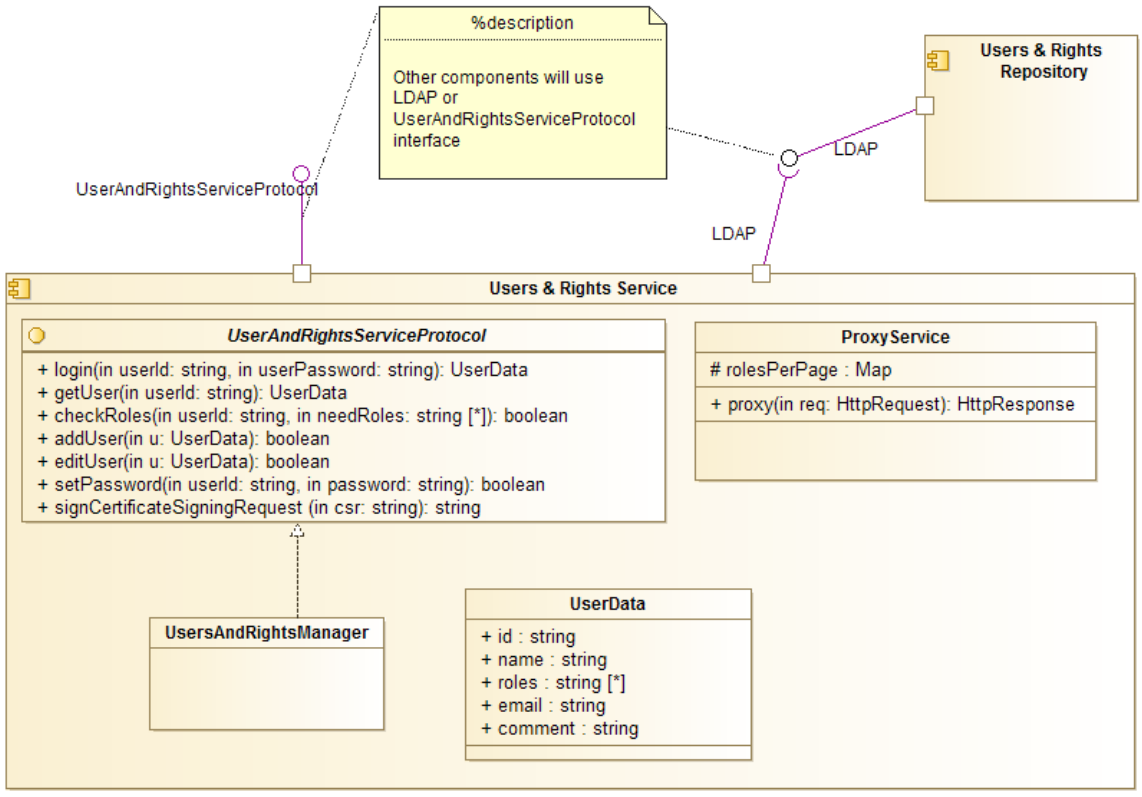

Figure 29: Users & Rights Service - Class diagram

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 an cookie)
4. Proceed with “Check user authorisation”

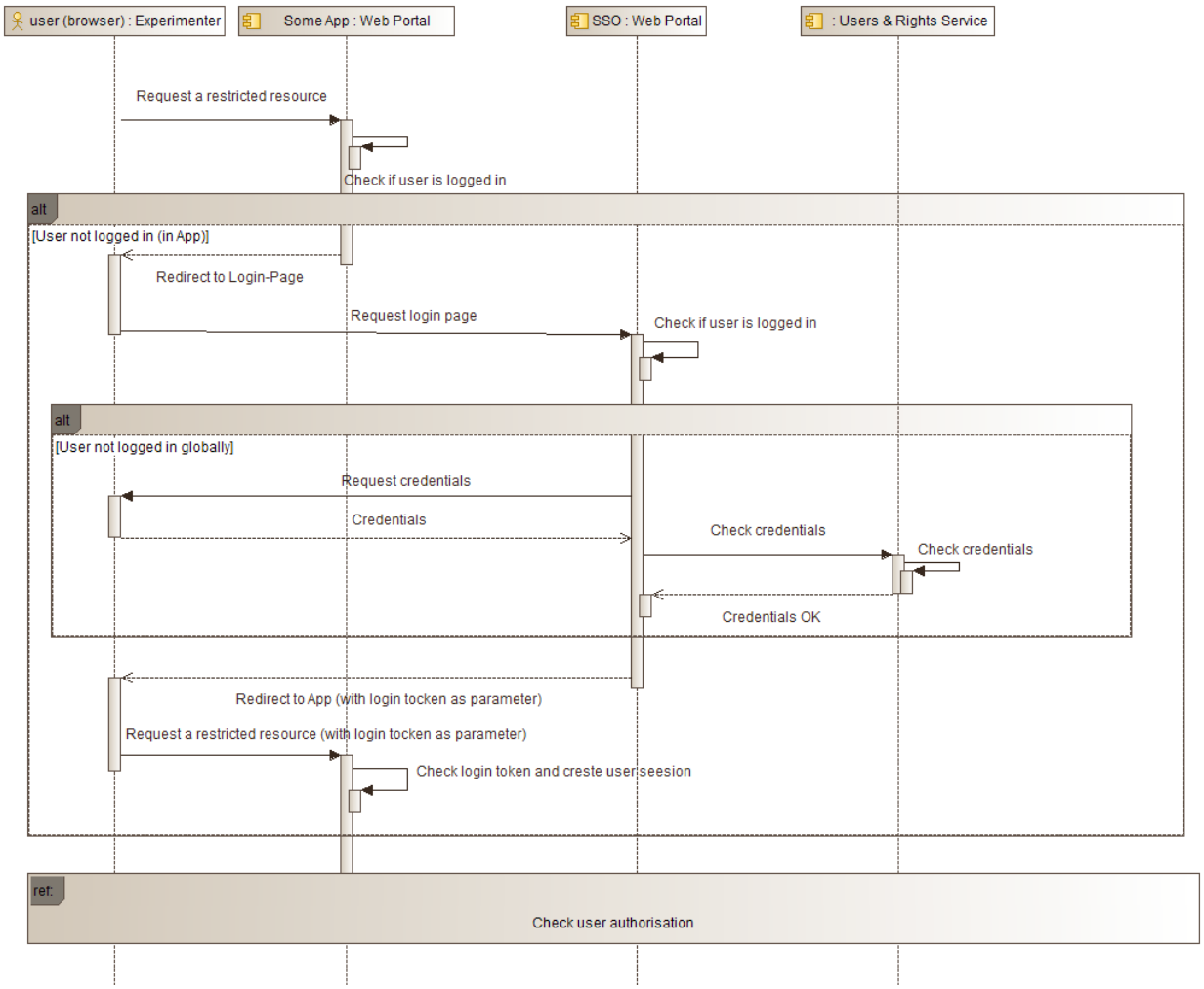

Figure 30: Users & Rights Service – Password-based user login

X.509 Certificate-based user login

1. A user opens via its browser an application of the RAWFIE web page and requests a restricted resource (URL)
2. Client certificate validated during SSL handshake (transport layer)
    a. If correct: proceed processing in application layer
    b. If wrong: cancel SSL connection (end).
3. Check if not logged-in (application layer)
    a. Read user name of the X.509 certificate and create a user session
4. Proceed with "Check user authorisation"

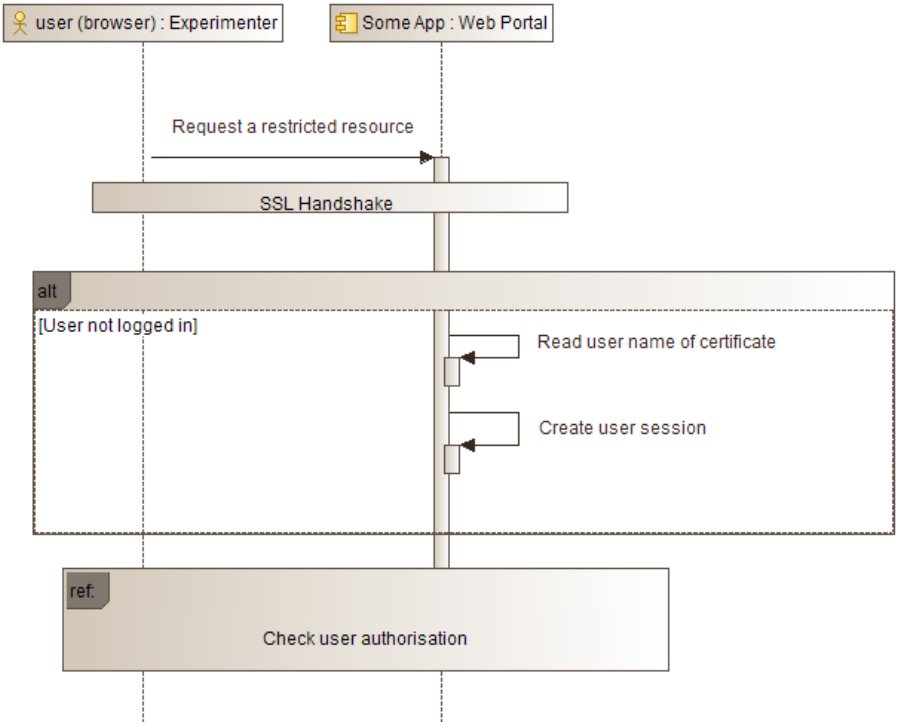

Figure 31: Users & Rights Service – X.509 Certificate-based user login

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 contains 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.

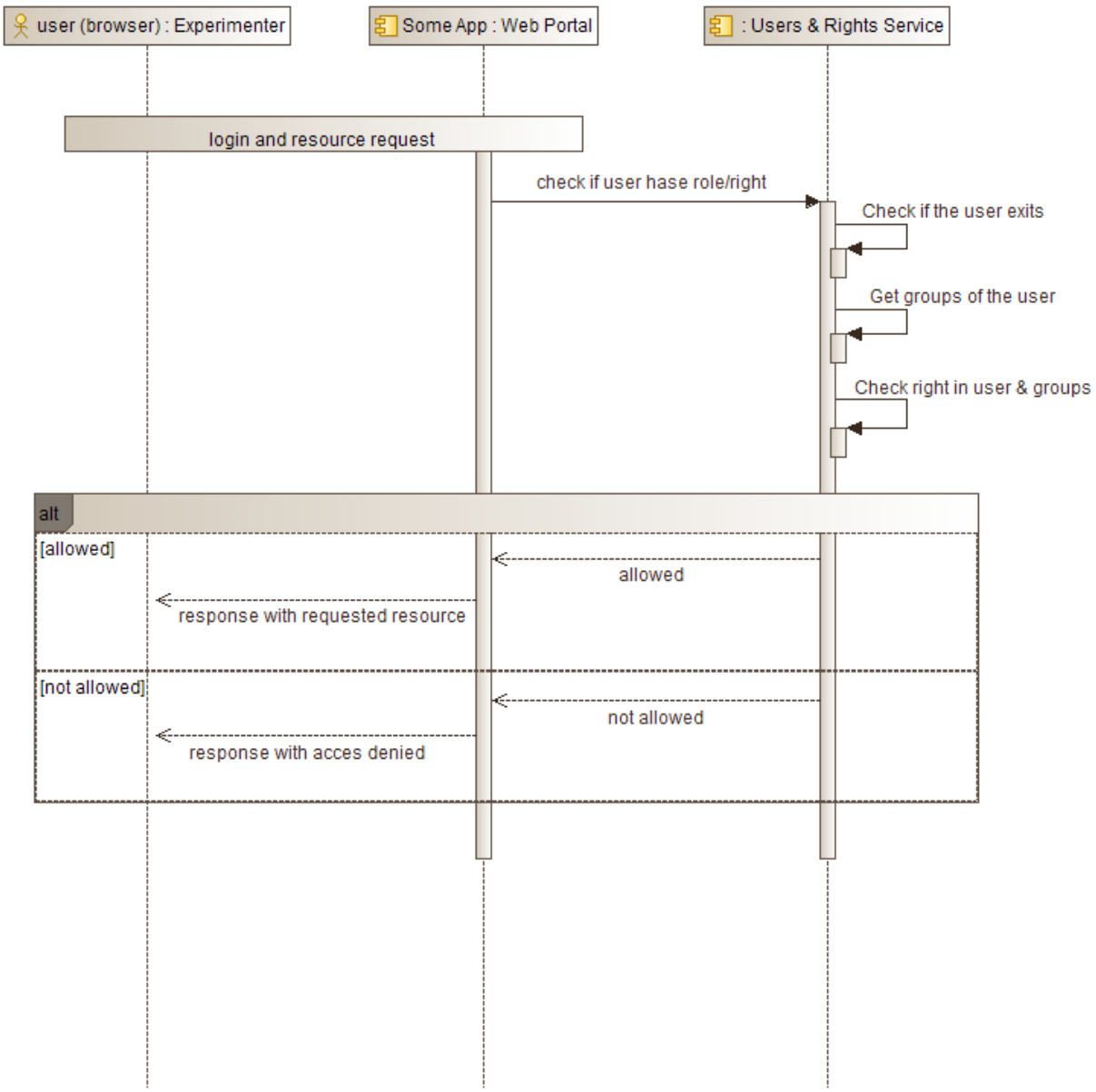

Figure 32: Users & Rights Service – Check user authorisation

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

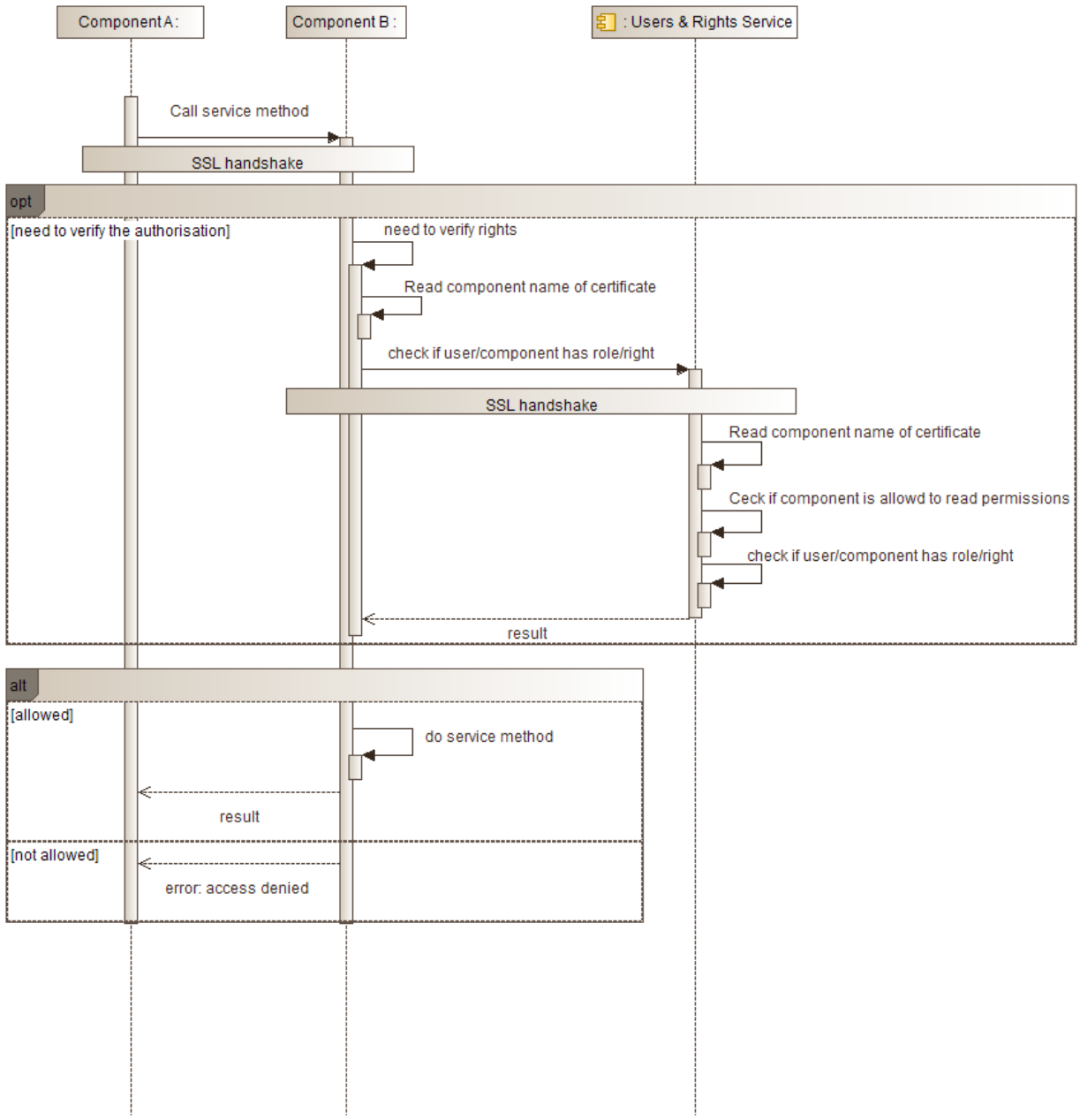

Figure 33: Users & Rights Service – Check user authorisation

Interactions and relationships with other components

Provided interfaces

- UserAndRightsServiceProtocol:
  Any other RAWFIE component (especially from the Front Tier) may access this interface to get user related information.

Required interfaces

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

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.

Component requirements as identified in D3.2

ID (Priority)	Description	Requirement Mapping
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 **Error! Reference source not found.**)
PT-BOO-S-002 (HIGH)	User level booking shall be triggered by the Booking Tool via a REST API.	Booking Tool can call the *IBookingService* 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-009 (HIGH)	Booking functionality shall support reservation of resources involving multiple testbeds	NOT IMPLEMENTED Due to the nature of the resources Booking Functionality currently supports reservations on single testbeds
PT-BOO-S-010	Booking functionality shall be able	*IBookingService methods* will be exposed as

(HIGH)	to correctly handle simultaneous Reservations requests by end users	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-T-012 (HIGH) (*Moved from Booking Tool*)	Booking functionality should provide means to ensure fairness in resource booking as well as protect for malevolent actions that a user may perform.	B*ookingRequestChecker* will apply a set of checks on the proposed reservation ensuring 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

Responsibilities

The main responsibilities of the Booking Service are:

- To validate all reservations requests (add, edit, delete, show) coming from the Booking tool based on a set of predefined constraints/checks
- To coordinate reservations of testbed resources among experimenters ensuring fairness
- To provide notifications for the status of pending reservations via email or message bus
- To interact with the Master Data Repository for persisting/updating information regarding a specific booking

Operations and attributes

The Booking Service provides all necessary methods to manage the bookings including add, edit and delete operations (*IBookingService* interface). Add and edit methods should return conflicting bookings in case of error or an empty list if the operation was successful. It is also possible to check explicitly for conflicts with other bookings and to query all bookings in a given timespan (optionally also filtered by user-id). A booking data entry stores, beside a unique ID and a timespan (start and end date), the reference to the experiment and to experimenter/user that performs the reservation. Also a list of booking items (references to the testbed and to the used UxVs) is stored with the booking data. The *BookingManager* implements the Booking Service interface and implements the necessary business logic to retrieve, store and generally manage data in the Bookings Repository. Two additional modules B*ookingRequestChecker* and *BookingStatusUpdater*, are also envisioned responsible for checking the validity of booking requests and for sending status update messages respectively.

B*ookingRequestChecker* module will be responsible for applying a series of checks, besides the check for conflicts) that will validate whether reservation process should continue or not. This may include but not be limited to:

- Ensuring some fairness in reservations (i.e. not allowing a user to book all testbed resources)
- Checking user validity and authorization

Booking service provides also an approveBooking(id) method through which a pending booking can either be accepted or rejected. Only platform users registered as testbed administrators (or a similar role) should be allowed to call this method.

BookingService will also provide functionality for sending email notifications to the platform registered users that are involved in a booking action. This may involve registered experimenters creating or editing their booking as well as testbed administrators or testbed managers responsible for deciding whether a proposed booking request can be fulfilled or not.

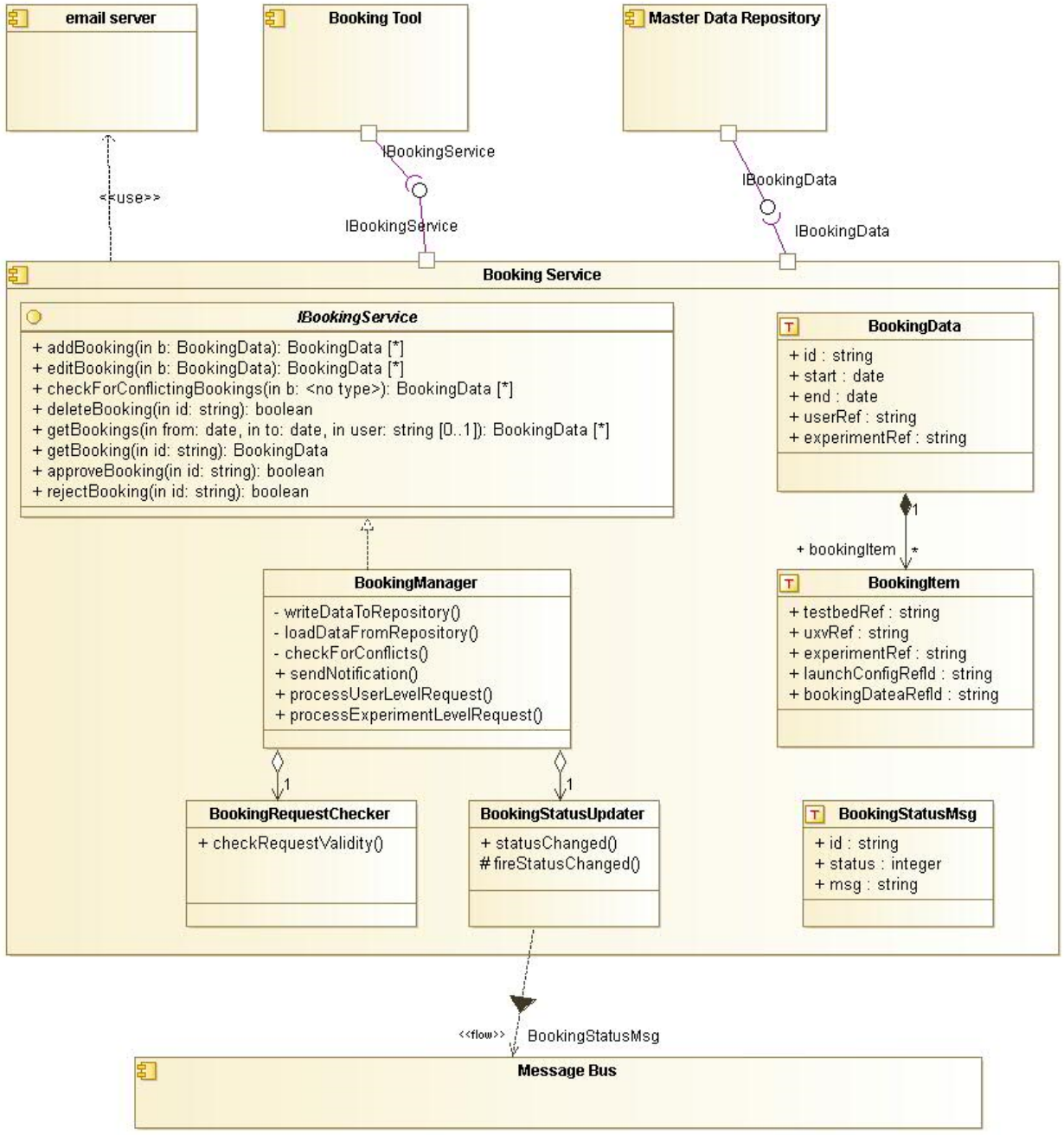

Figure 34: Booking Service - Class diagram

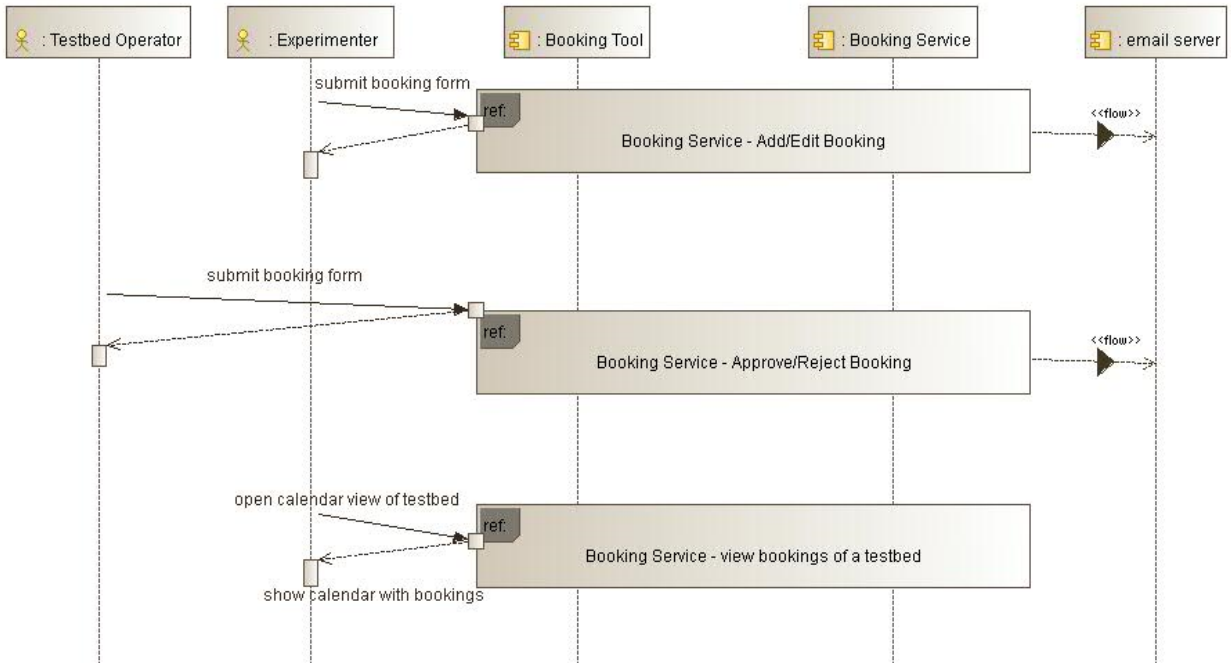

Figure 35: Booking Service - Overview

View bookings of a testbed

1. Experimenter opens the calendar view of a specific testbed in the Booking Tool
2. The Booking Tool requests the bookings of the testbed in the shown timespan from the *BookingManager* (*IBookingService interface*)
3. The *BookingManager* loads the data from the Master Data Repository and returns the results to the Booking Tool
4. The results are displayed in a calendar view

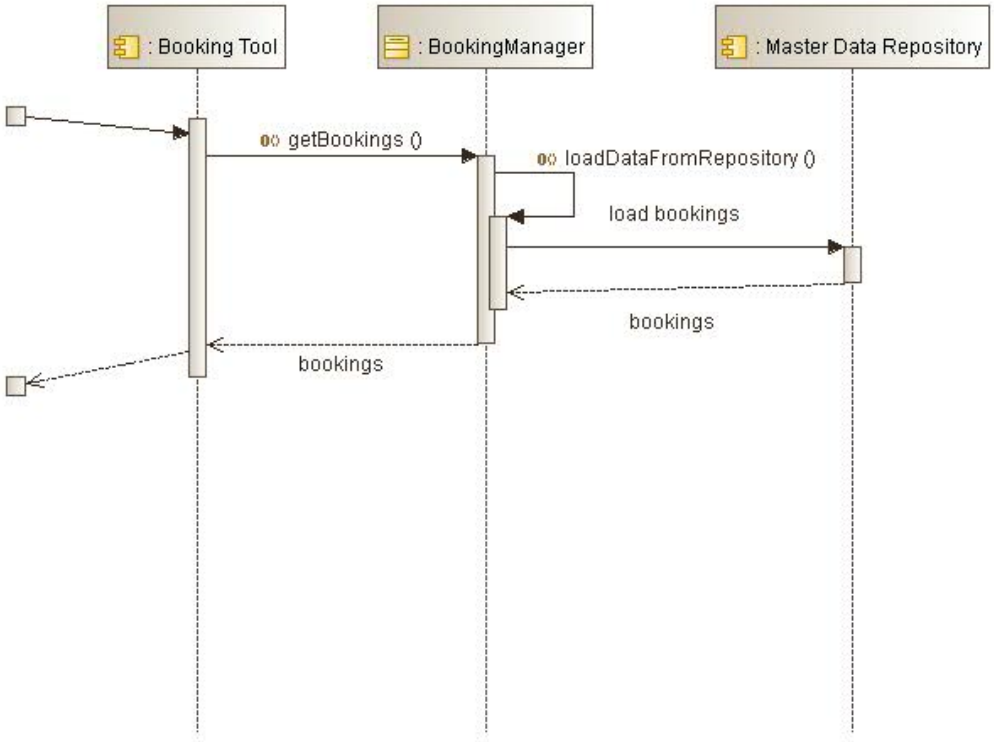

Figure 36: Booking Service – View bookings of a testbed

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. 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),
    b. 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
    c. return a success message to the *Booking Tool*, which shows it to the user

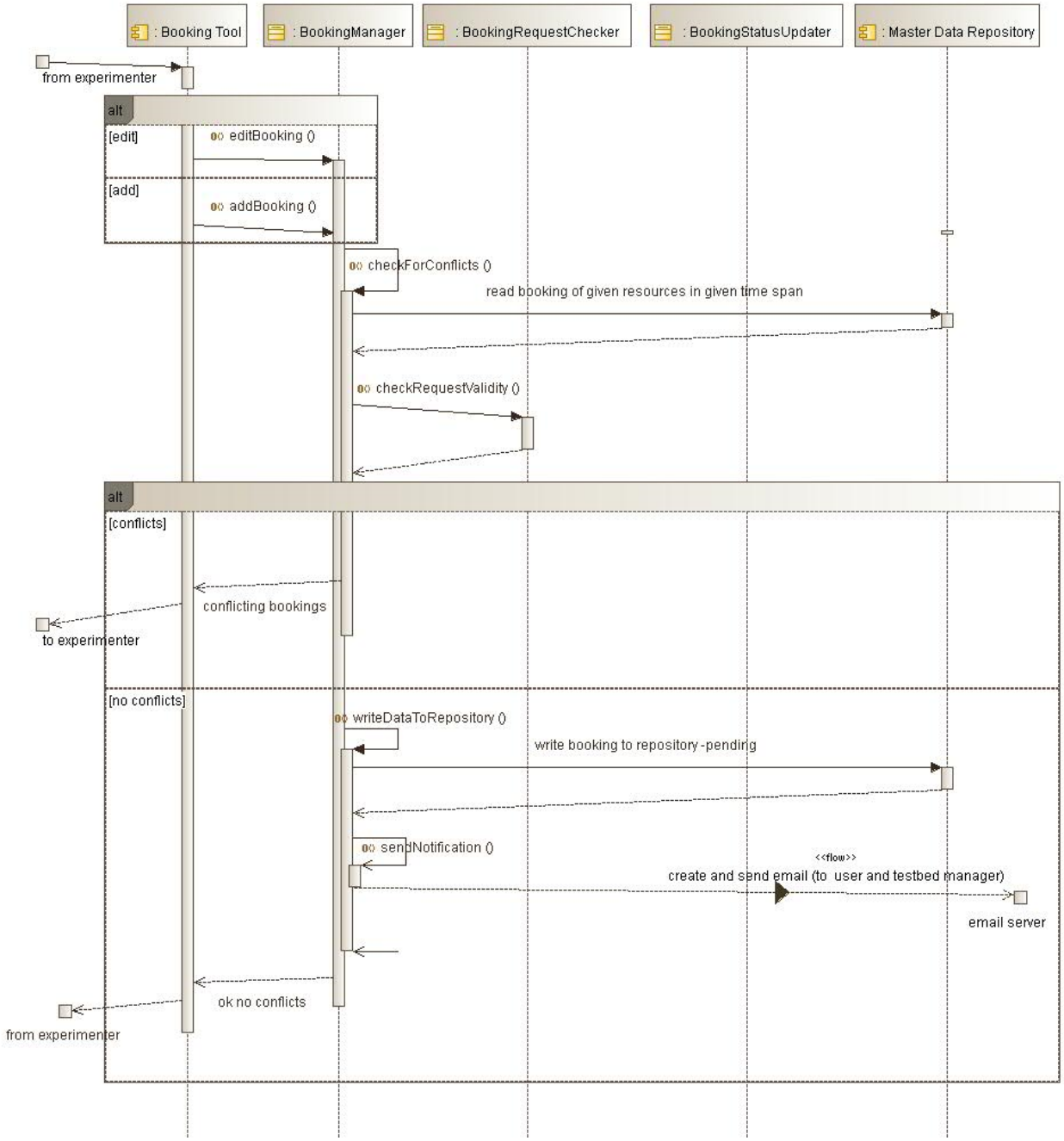

Figure 37: Booking Service – Add/Edit a booking

Approve/Reject a pending booking (testbed operator)

1. A Testbed operator (or similar role) loads the calendar view where all bookings (pending and confirmed) are displayed
2. Testbed operator selects an appropriate action for a pending booking request (*approveBooking(…)* or *rejectBooking(…)*)
3. The *BookingManager* interacts with the *Master Data Repository* in order to update the booking request status (*CONFIRMED* or *REJECTED*) in the repository
4. The *BookingStatusUpdater* module fires a status change involving:

- Sending of corresponding email to registered experimenters that initiated the booking
- Sending of a *BookingStatusMsg* to the message bus for any other component that might be interested of being informed.

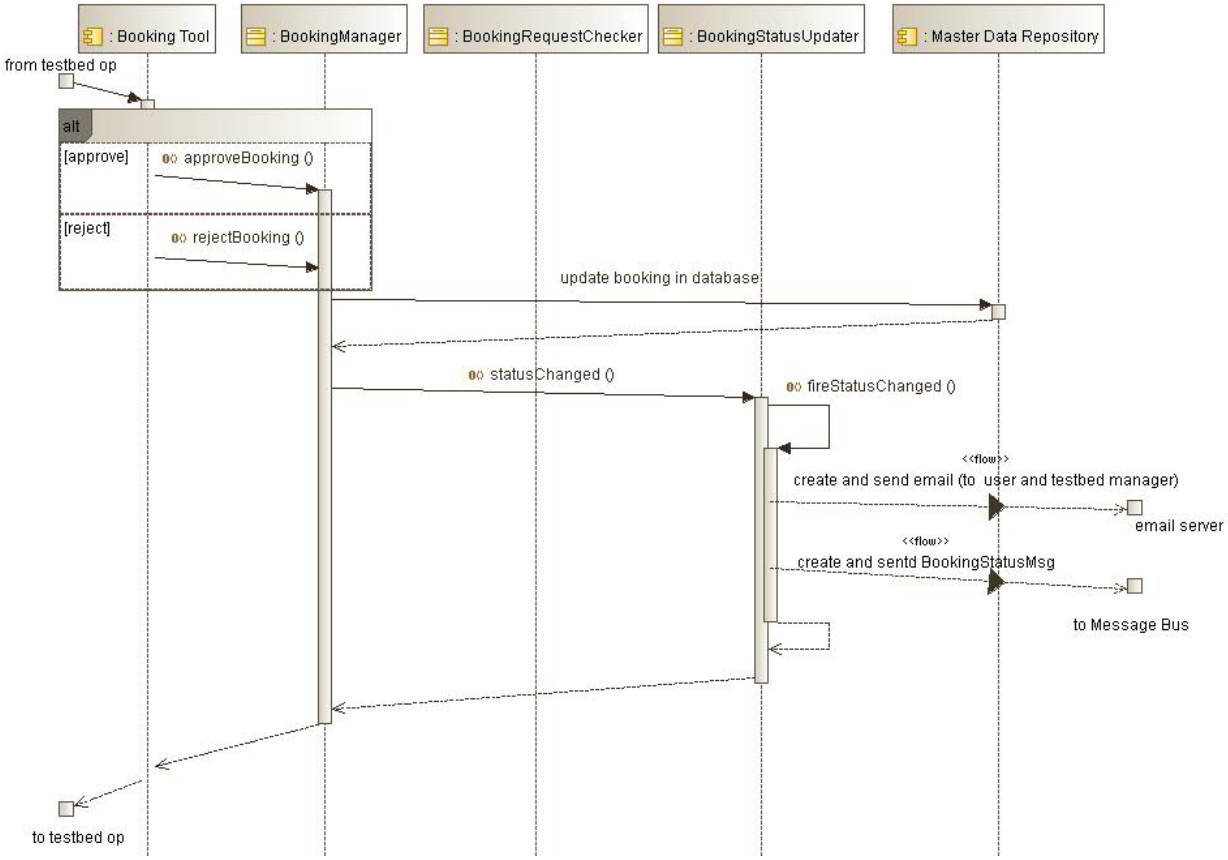

Figure 38: Booking Service – Approve/Reject booking

Interactions and relationships with other components

Booking Service implements the *IBookingService* interface which used mainly by the Booking Tool that provides a Web UI to manage the bookings (edit/add/approval etc.). However other components (i.e *Launching Service*, *Experiment Authoring Tool*) are also possible to call certain methods of the *IBookingService* in order to obtain information on specific user level reservations that may be needed for performing their activities.

IBookingService methods will be exposed both via RPC or REST API.

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.

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 authorised and

approved experiments based on spatio-temporal constraints validated just prior to the actual launching.

Component requirements as identified in D3.2

ID (Priority)	Description	Requirement Mapping
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 starting of an experiment	*ExperimentStartRequest* JSON message is sent by createAndSendMessage() upon successful processing of manual or scheduled launching (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	By design, Launching Service interacts only with middle tier components, the message bus and the master repository.

	other components should respect the RAWFIE platform boundary	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-011 (MEDIUM)	RAWFIE platform shall provide means to ensure fairness in experiments execution	**NOT APPLICABLE** The LS is responsible only for initiating the experiment execution process and does not intervene to the actual experiment execution
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-BOO-S-011 (MEDIUM) (moved from booking service)	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

Responsibilities

The main responsibilities of the Launching Service are:

- To initiate *StartExperiment* requests either manually or on a scheduled basis and sent it to the message bus
- To initiate *CancelExperiment* requests for a running or scheduled experiment and sent it to the message bus
- To generate the experiment execution Identifier (executionId) that uniquely identifies a launched experiment within the RAWFIE system
- To update the Master Repository with information related to the initial launch configuration for an experiment.

Operations and attributes

Launching Service implements the *ILaunchingServiceProtocol* and supports two (2) types of launching modes:

(a) ***Short-term launching (manual launching)***: This gives the opportunity to experimenters to initiate pre-defined and pre-approved experiments directly after authoring them in the Authoring tool. The *manualStart(...)* method is used for this purpose.

(b) ***Long-term launching (scheduled launching)***: by calling the *schedule(...)* method after authoring and assigning resources to an experiment the experimenter is able to schedule

an experiment to be executed at a feature time. This is achieved by using the Launching service internal scheduler.

Experiment launching in both cases is subject to a number of internal pre-validations including consistency of the experiment with the associated experiment level reservations, experimenter's authorization, and/or absence of already running instances of the experiment under launch.

Launching service is also responsible for handling cancellation of experiments. This may include experiments already running or experiments already scheduled. Cancellation is achieved via calling the *cancel(…)* method providing preferably the experiment *executionId*[5]

Figure 39, provides a class diagram depicting the internal structure of the Launching Service including the non-primitive data types used, as well as the expected interactions with other RAWFIE services and the Master Data Repository. Following pictures presents sequence diagrams elaborating the internal behaviour of the system.

[5] executionId uniquely identifies a running or scheduled experiment within the RAWFIE platform. It should not be confused with the experimentId, which adheres to an authored experiment. executionId represents a running instance of an authored experiment

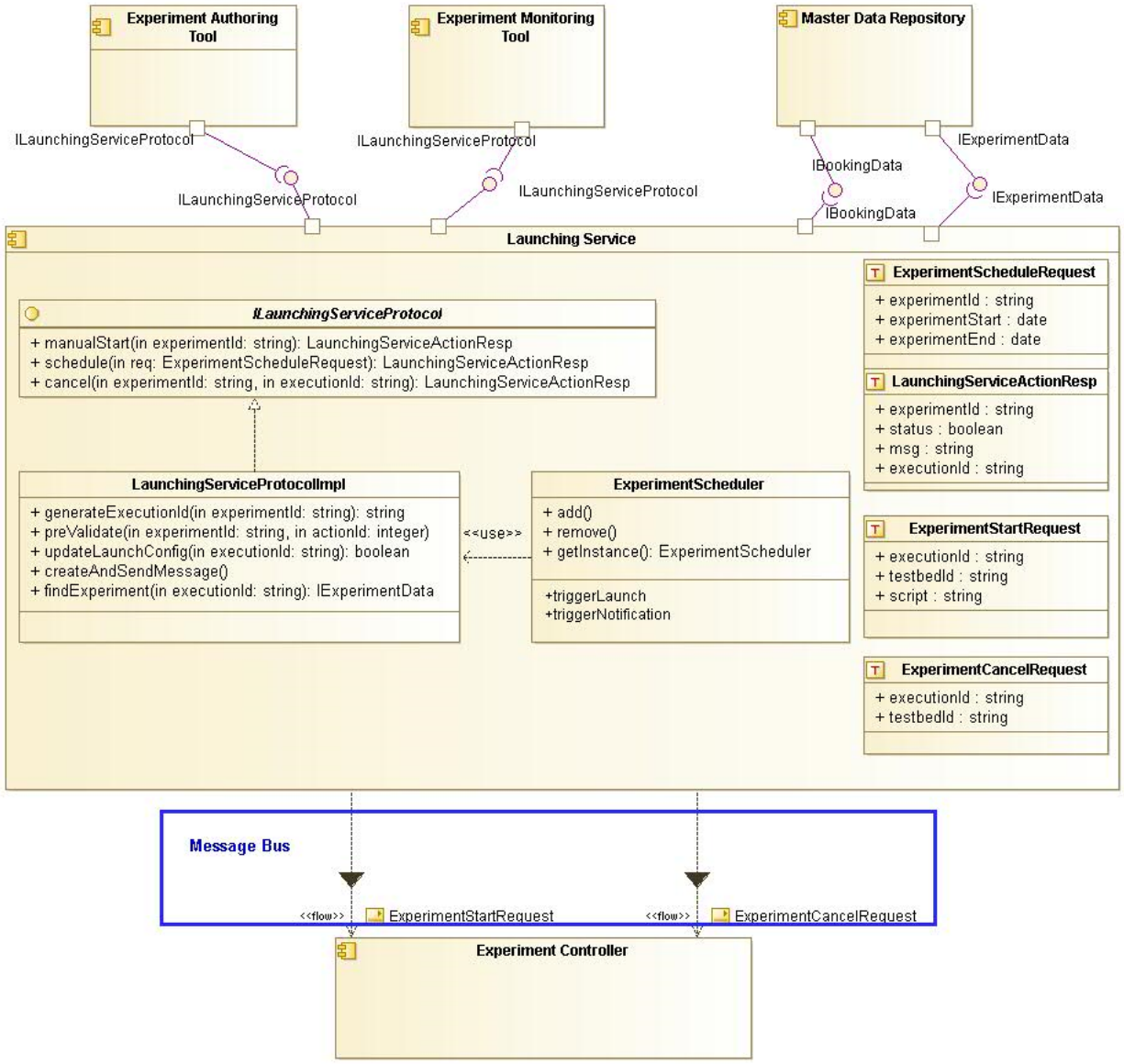

Figure 39: Launching service – Class diagram

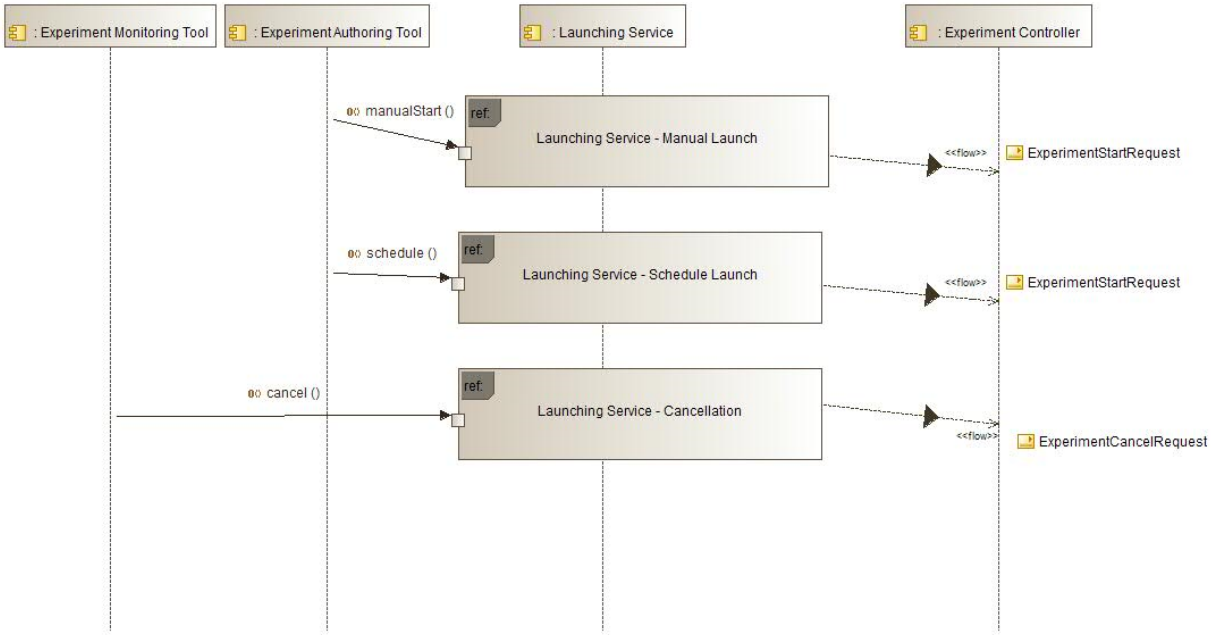

Figure 40: Experiment launching Service Overview - Sequence diagram.

Figure 41: Experiment Launching Service - Manual Launch Sequence diagram.

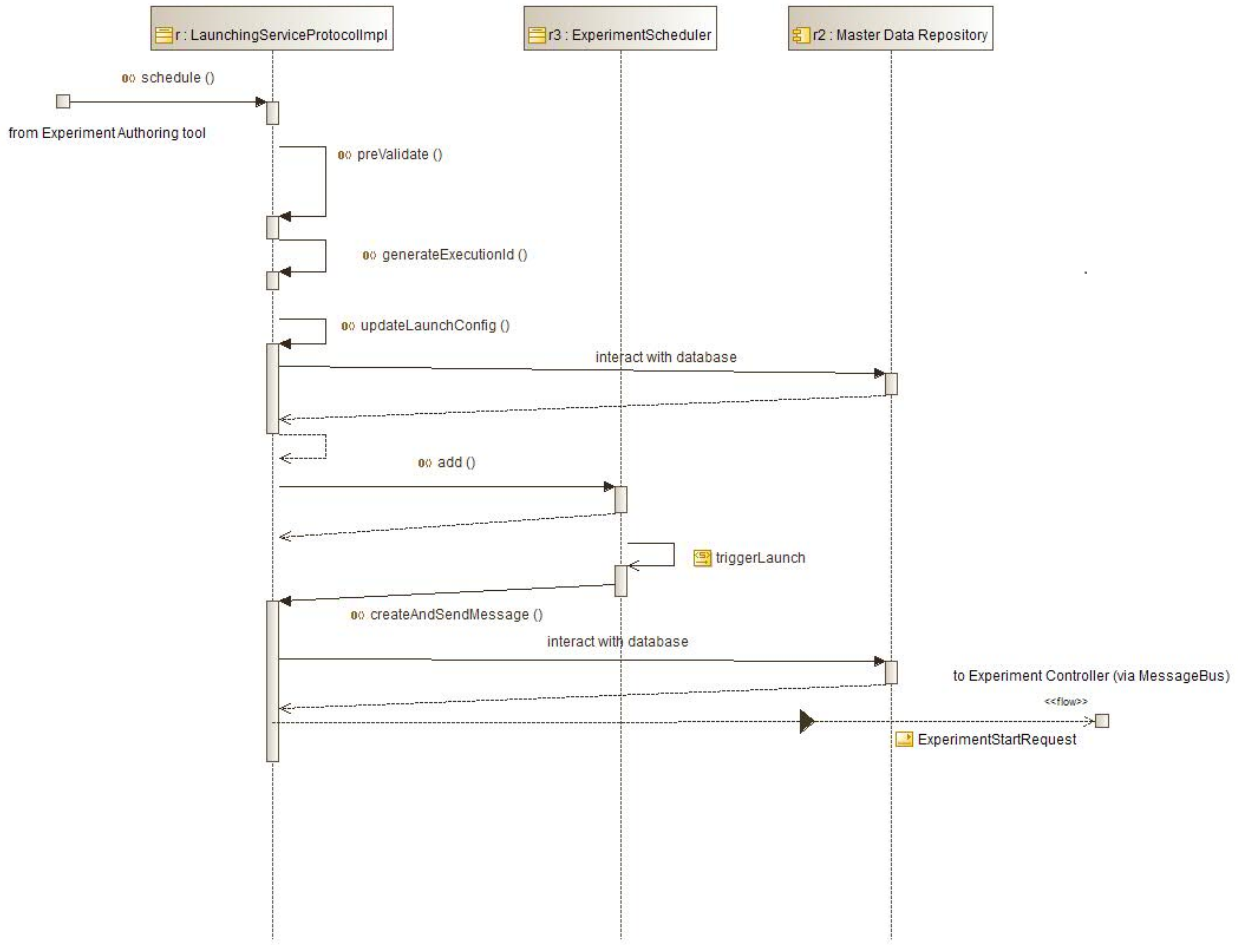

Figure 42: Experiment Launching Service - Scheduled Launch Sequence diagram.

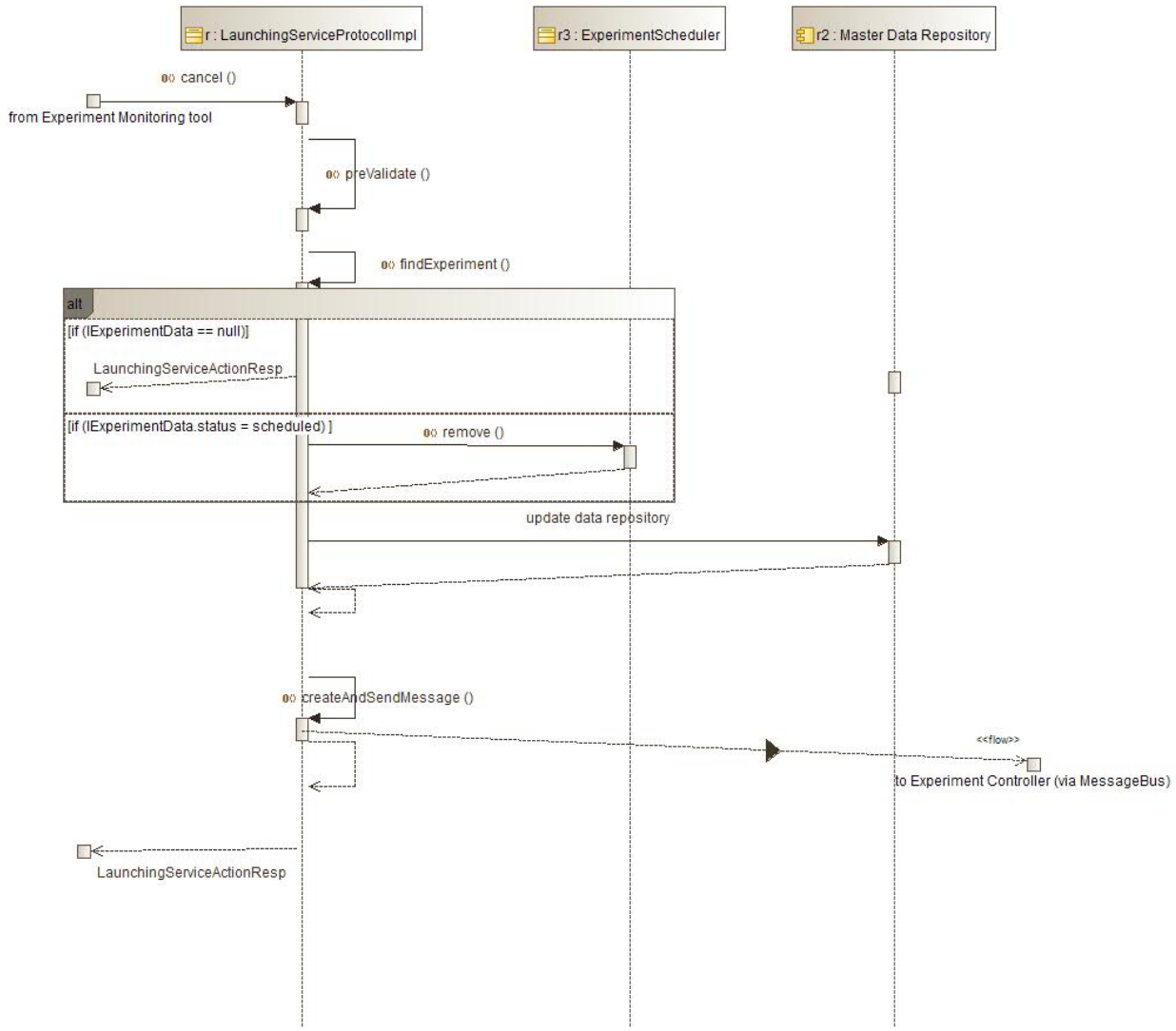

Figure 43: Experiment Launching Service - Cancellation Sequence diagram

Interactions and relationships with other components

As depicted in the above UML diagrams the Launching Service listens for requests either by the *Experiment Authoring Tool* (for starting or scheduling an experiment) or from the *Experiment Monitoring Tool* (for experiment cancellation). Launching Service interacts with the Master Data Repository for retrieving all the necessary information for processing its requests and creates appropriate JSON messages targeting the *Experiment Controller*. The latter communication is asynchronous and implemented via the *Message Bus*.

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.

Component requirements as identified in D3.2

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

Responsibilities

The main responsibilities of the visualisation engine are:

- Convert waypoints and GPS data to geo information layers and pass them to the VT
- Retrieve maps from external providers and pass them to the VT
- Cache data for faster download of maps

Operations and attributes

In this part of the description a high level class diagram of the visualisation engine (VE) is presented that will be in the middle tier, and will work closely with visualisation tool (VT) that will reside in the front end tier.

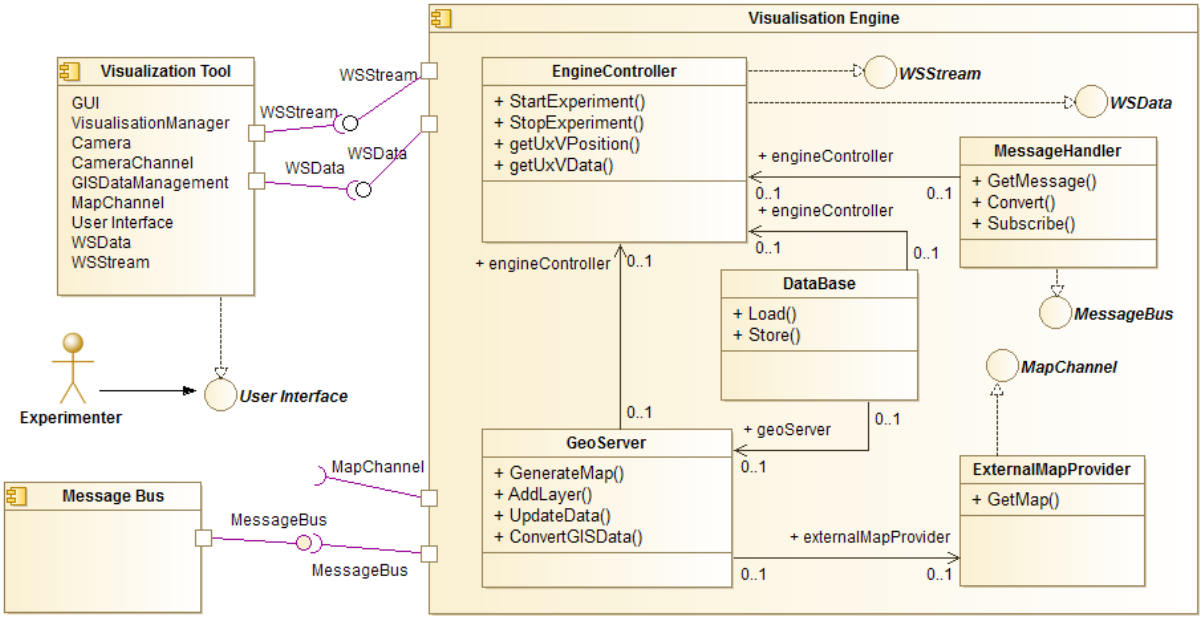

Figure 44: Visualisation Engine - Class diagram

The VE will take 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

The sequence diagrams below provide information about the data flow and the interaction between the different components.

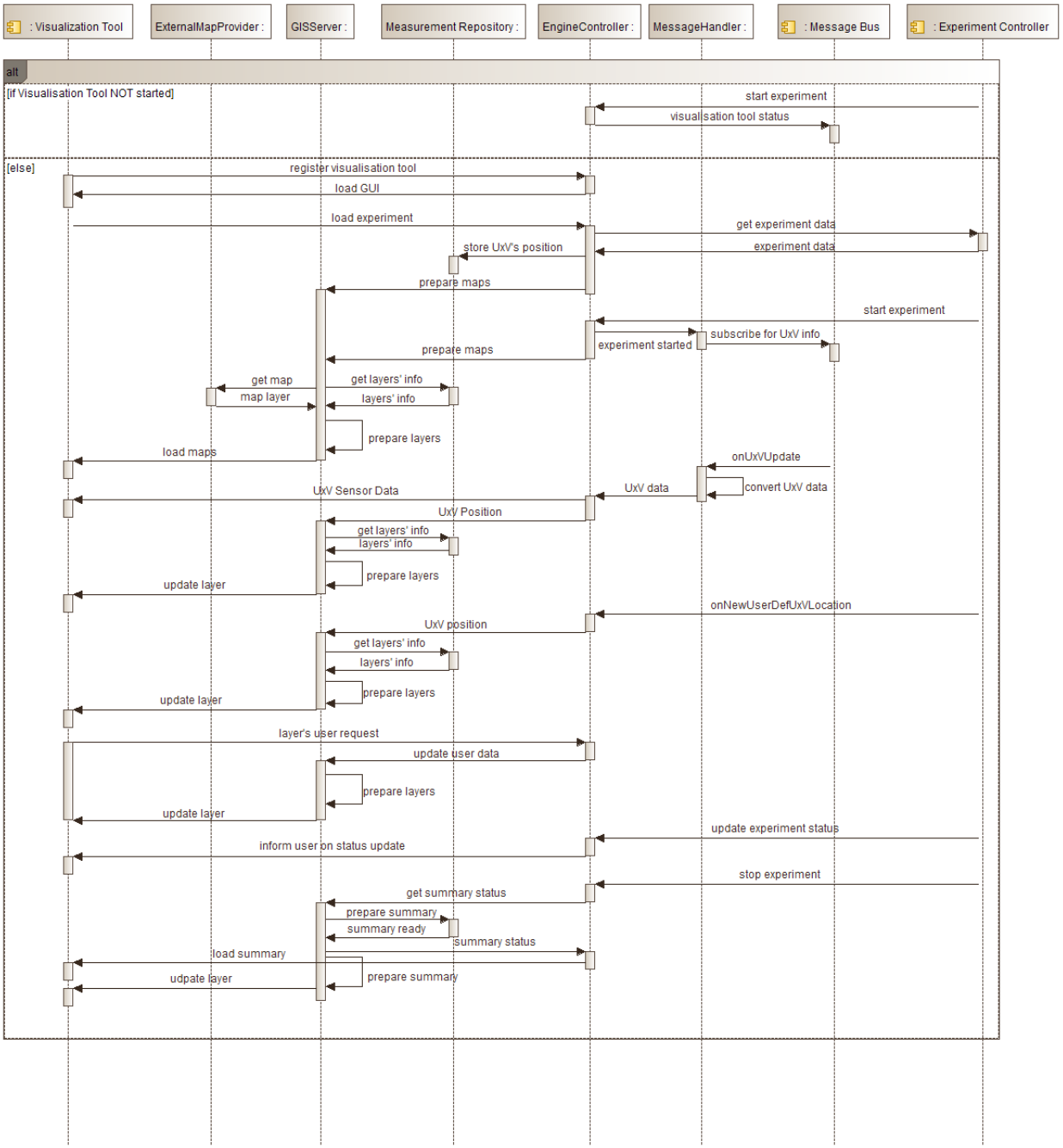

Figure 45: Visualisation Engine - Overview

Start an experiment:

1. The Experimenter clicks on a button to start the visualisation of the experiment.
2. The VT opens two websocket connections. One is for static data for the experiment, the other is for streaming of data in realtime.
3. The VT requests via WebSocketData the parameters for the experiment like number of UxVs, their type, Sensor number, their type, units etc.
4. The VE retrieves this data from the Master Data Repository and sends it back to the VT

5. The VE subscribes to the UxV topics on the MessageBus. Whenever a new data from the UxVs from the experiment comes, then this data is updated, adjusted for the VT and sent to it.
6. The VT udpates the visualisation with the new received data.
7. The browser window is properly initialized and set for the VT to show the experiment's run.

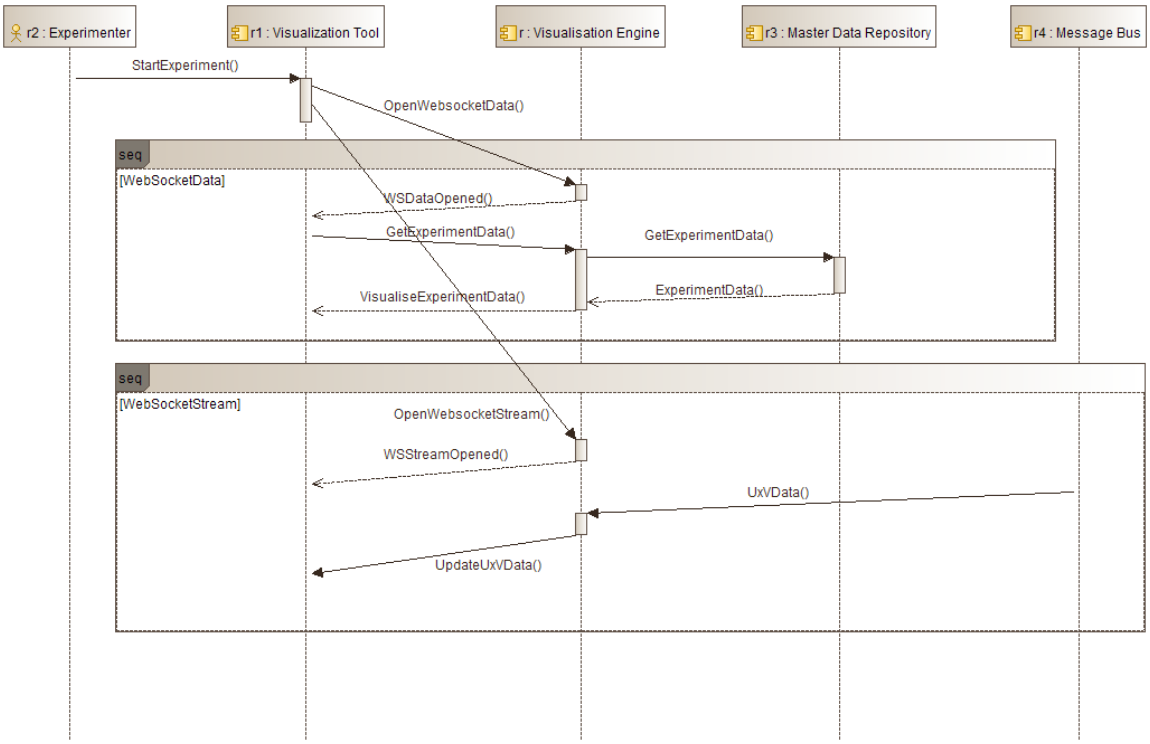

Figure 46: Visualisation Engine – Start an experiment visualisation

UxV update:

1. Whenever there is a change in the experiment data, the VE receives the data from the topics that it is subscribed to
2. The VE needs to update the received data in order to put it in the proper coordinate system, in the expected units and others.
3. Then the VE sends this data to the VT. The VT updates the screen with the new position of the UxVs for example or with the new sensor data

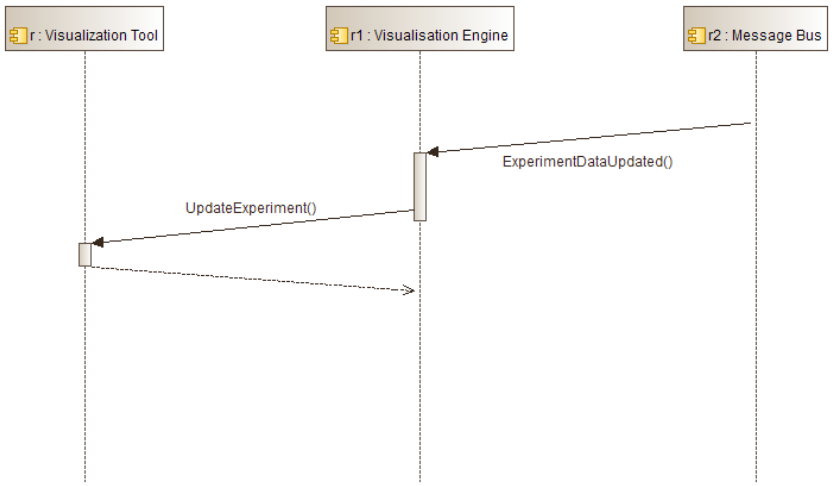

Figure 47: Visualisation Engine - Position Update

The experimenter can follow the positions of the UxV during the execution of the experiment.

The Experiment Controller updates the status of the experiment:

1. The Experiment Controller updates the status of the experiment. This could mean that there was an interference during the execution of the experiment, or there was an unexpected interruption, or another issue. This data is sent to a predefined topic on the message bus
2. The VE receives this data, because it is subscribed to this topic.
3. The VE forwards this information via WebSocketData to the VT
4. VT informs the Experimenter about the updated status.

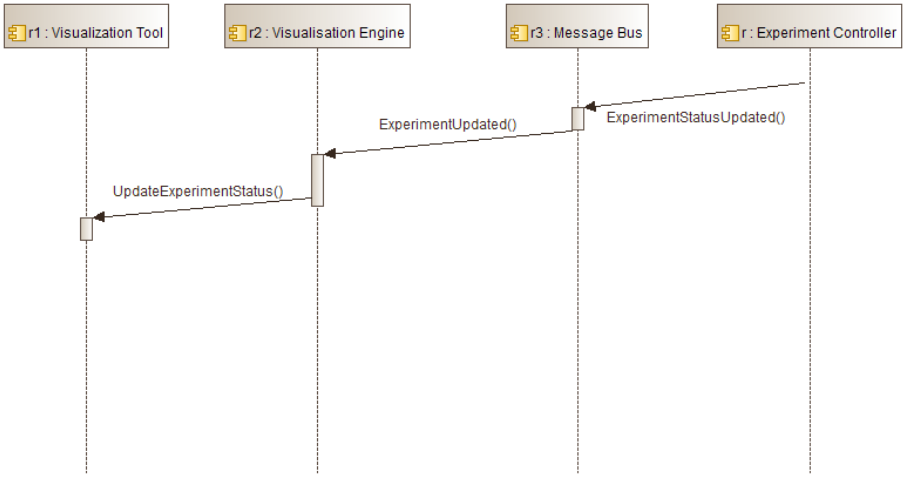

Figure 48: Visualisation Engine - Update Status of an Experiment

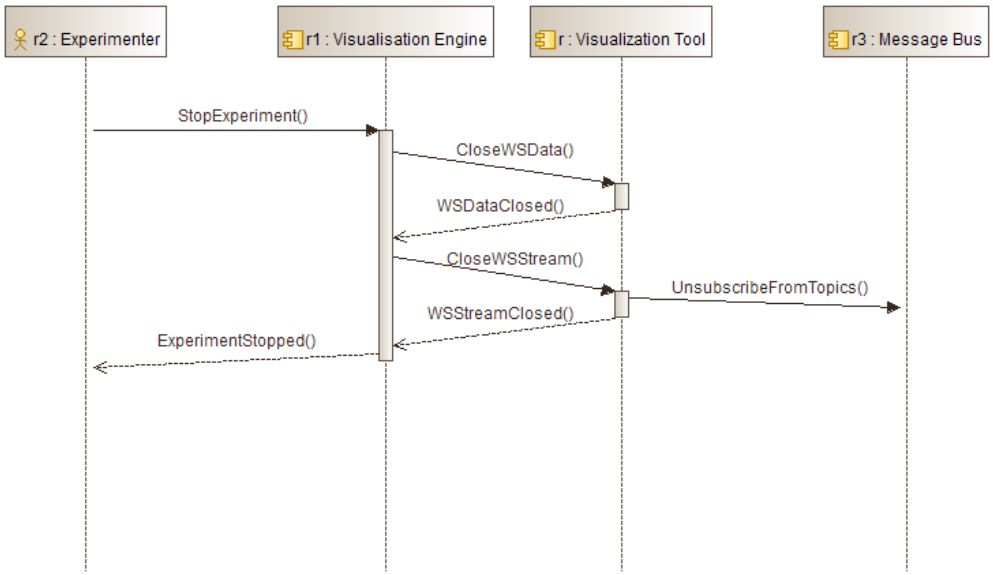

Figure 49: Visualisation Engine – Stop an experiment visualisation

Replay an experiment:

1. The Experimenter start the replay of the experiment.
2. The VT opens WebsocketStream for getting the data for the experiment.
3. The VE starts fetching data from the Master Data Repository and, after updating it for the proper visualisation in the VT, sends it to the VT.
4. This continues until the experiment is over or until the Experimenter decides to stop the visualisation

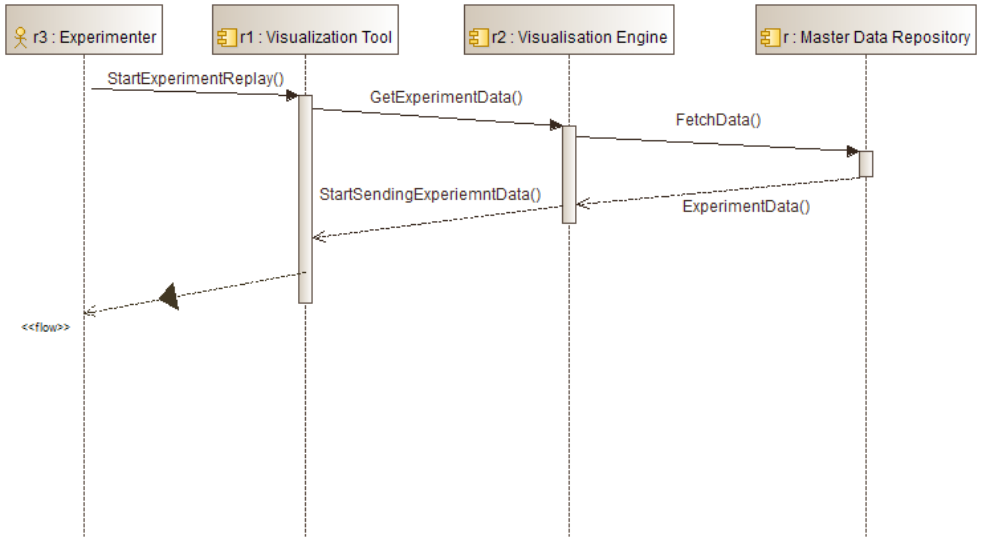

Figure 50: Visualisation Engine - Replay an experiment

Interactions and relationships with other components

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.

4.2.9 Data Analysis Engine

Component requirements as identified in D3.2

ID (Prioriority)	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.

Responsibilities

The main responsibilities of the Data Analysis Engine are:

- Schedule analytics job
- Provide compute engine with correct endpoints
  - data source
  - data sink
- Create appropriate jar file for compute engine
- Provide train, evaluation and infer methods to work with built model

Operations and attributes

- Class diagram involving the Data Analysis Engine:

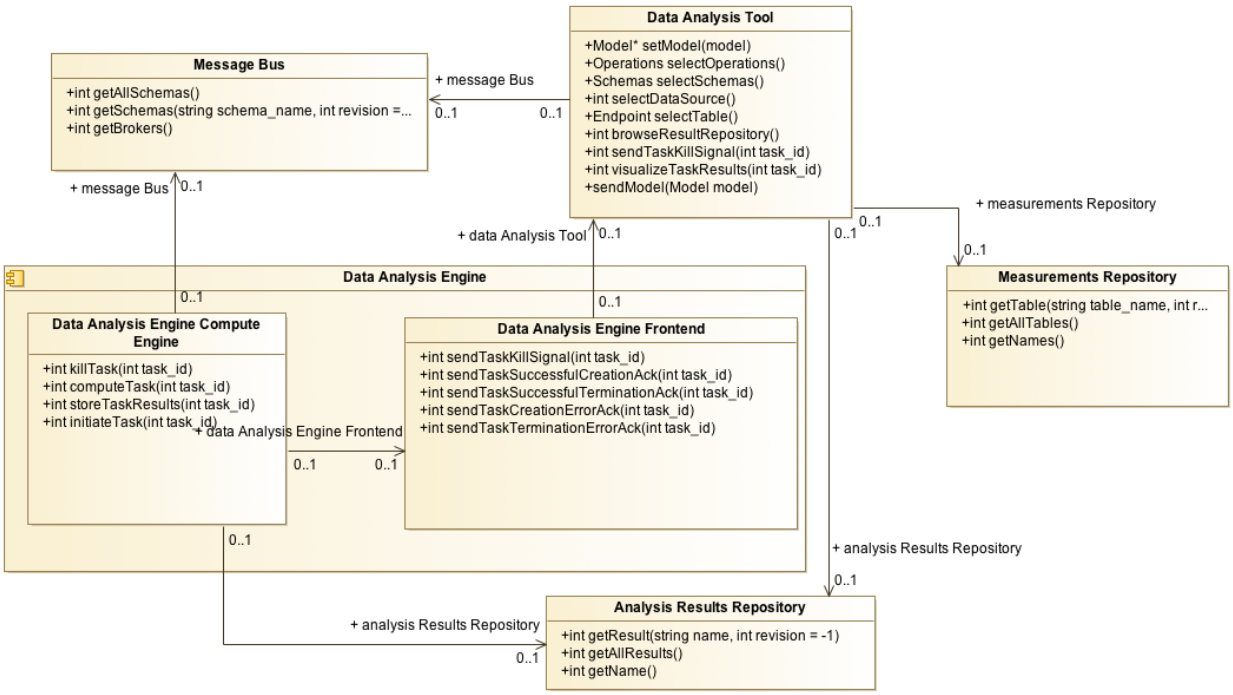

Figure 51: Data Analysis Engine - Class diagram

Sequence diagrams involving the Data Analysis Engine in the case of a streaming analytics task:

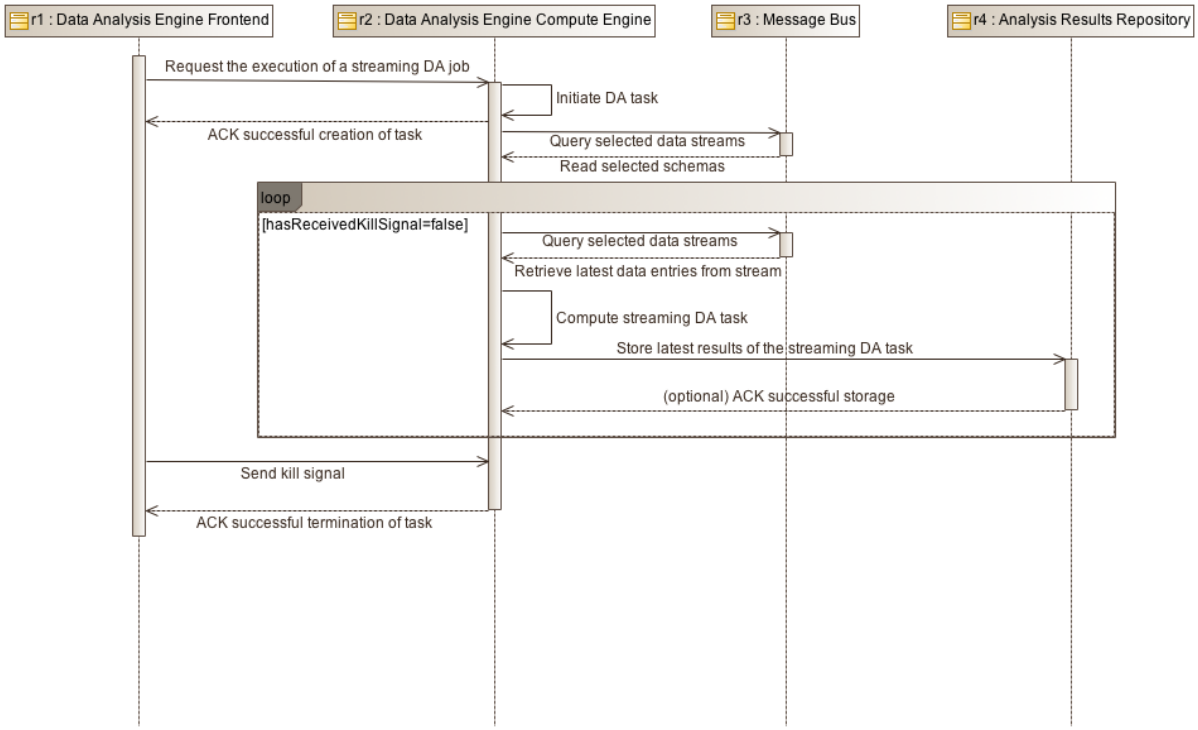

Figure 52: Data Analysis Engine – streaming analytic task

Interactions and relationships with other components
The Data Analytics Engine will communicate with:

1. Message Bus [Read Only]
2. Analysis Results Repository [Read/Write]
3. Measurement Repository [Read Only]
4. Data Analysis Tool [Read Only]

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.

Component requirements as identified in D3.2

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.

Responsibilities
The main responsibilities of the System Monitoring Service are:

- Collect health status information from all relevant RAWFIE components
  - "Pinging" servers and services
  - Run special plugins to get detail status information
- Receive health status data from the Monitoring Manager (via message bus).
- Store and aggregate status information to be displayed via the System Monitoring Tool.

- Send alerts to the RAWFIE System Administrator when servers or services are not responding or if the defined thresholds of performance indicators are exceeded.

Operations and attributes

The System Monitoring Service will be realized by configuring and extending the existing monitoring solution Icinga [7] (of fork of Nagios [9]). 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 [10] extension will be 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.

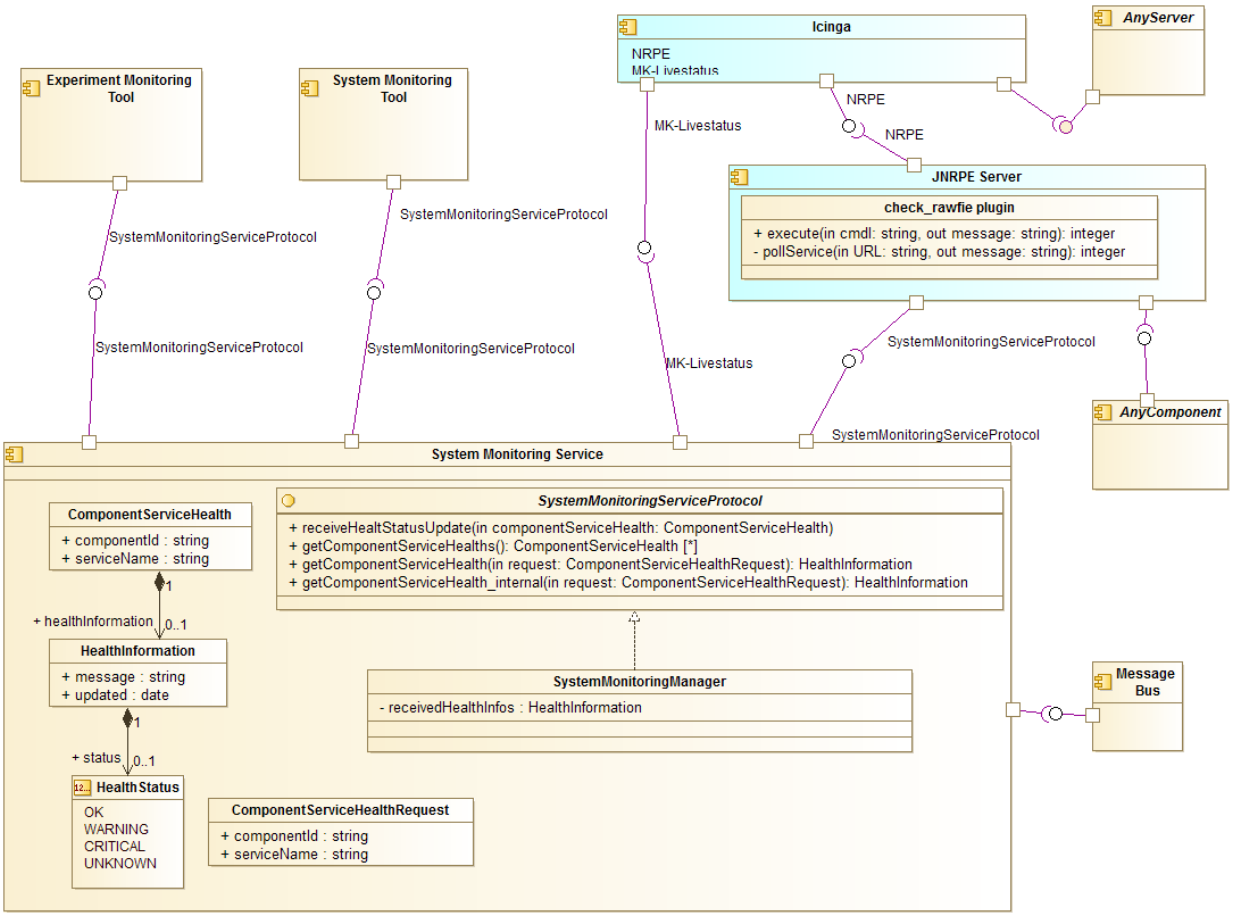

Figure 53: System Monitoring Service - Class diagram

Checking procedure

1. Internal timer of Icinga starts the monitoring procedure
2. For all configured servers / services do…
    a. Check type of server / service
        i. If type is a standard Icinga checker, execute it (e.g. ping, CPU load, RAM usage, HTTP alive)
        ii. If type is "Poll RAWFIE component status"
            1. JNRPE Server is called (check_rawfie plugin)
            2. JNRPE Server polls the health status of the RAWFIE component
            3. JNRPE Server returns the result
        iii. It type is "Health status received via message bus"
            1. JNRPE Server is called (check_rawfie plugin)
            2. JNRPE Server requests the latest health status received via message bus from the System Monitoring Service (getComponentServiceHealth_internal())
            3. JNRPE Server returns the result
    b. In case of an monitoring error
        i. Send notification (e.g. email) to all configured receivers (e.g. the RAWFIE system administrator)

3. Store the results

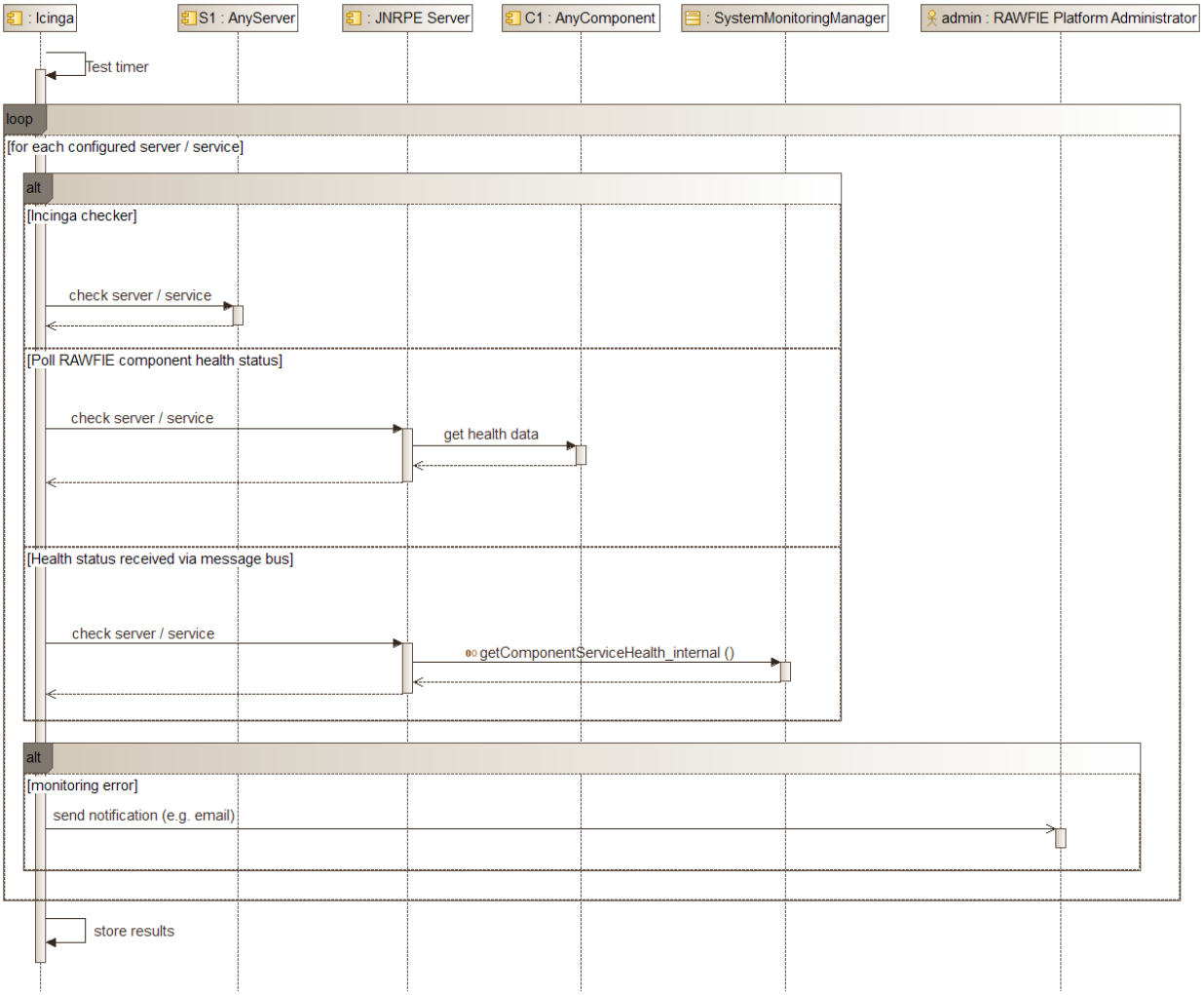

Figure 54: System Monitoring Service – Checking procedure

Received health status via message bus

Some components only send their health status autonomously to the message bus. The System Monitoring Service is listening for this messages and stores the latest health status per component internally. The JNRPE Server (check_rawfie plugin) can than request the status, to transmit is to the Icinga server.

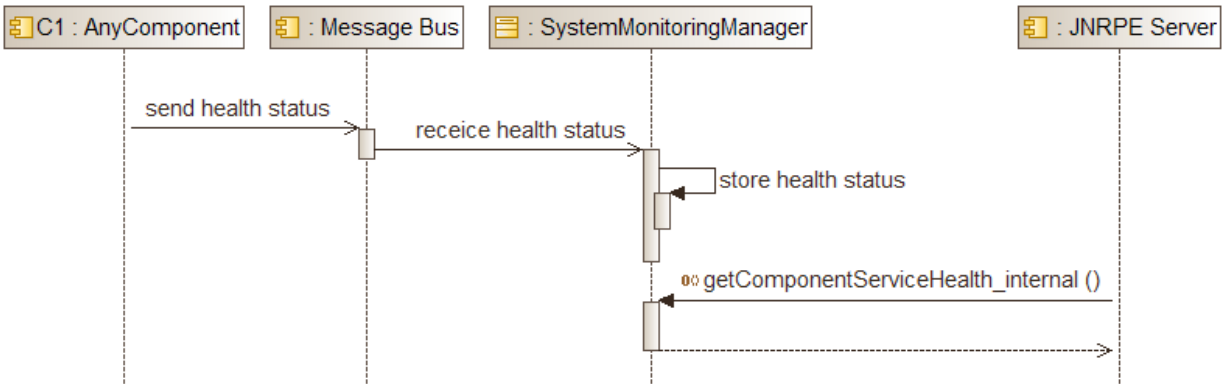

Figure 55: System Monitoring Service – Received health status via message bus

View health statuses

The are two way to get information about the current status of the system: The easy status dashboard in the Web Portal and the detailed Icinga Web application.

1. Via the status dashboard
    a. Admin request the status dashboard from the System Monitoring Tool
    b. The System Monitoring Tool calls getComponentServiceHealths() from the SystemMonitoringManager
    c. The SystemMonitoringManager request from Icinga the health statues via the MK-Livestatus interface.
    d. SystemMonitoringManager returns the health statues
    e. System Monitoring Tool renders the status page and returns it to the user
2. Via the Icinga Web
    a. Admin logs in into Icinga Web
    b. Admin request a detailed status page (e.g. history or chart)
    c. Icinga Web loads data from Icinga (via internal APIs)
    d. Icinga Web renders the page and returns it to the user

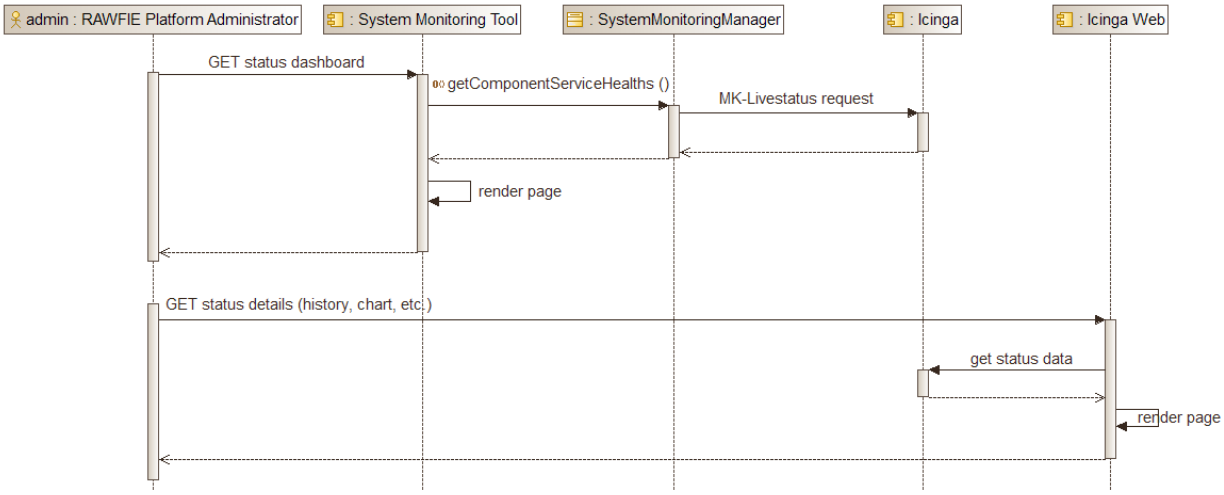

Figure 56: System Monitoring Service – View health statuses

Interactions and relationships with other components

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:

- Some health status interface
  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.

4.2.11 Accounting Service (FRAU)

Keeps track of resources usage by individual users to charge them later.

Component requirements as identified in D3.2

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	A CostModel will be defined that will various parameters to compute the cost of an resource usage
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. Notification 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.	The data type ResourceUsage will be used for this
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.	Will be covered by the CostModel
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.)	Will be covered by the CostModel
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	Will be covered by the CostModel

PT-ACC-S-007 (MEDIUM)	The accounting service should be able to handle the addition of new services that may be incorporated in the RAWFIE platform during time.	Will be covered by the CostModel

Responsibilities

The Accounting Service should cover all aspects that are needed to charge user/experimenter based on their resource usage.

- Determine the used resources for an experiment
- Compute the cost for the resource usage, based on different cost models that apply to the experimenter
- Create bills and sent them to the experiments
- Register payments and manage the balance of an experimenter
- Inform the user about his resource usage and his balance.

Operations and attributes

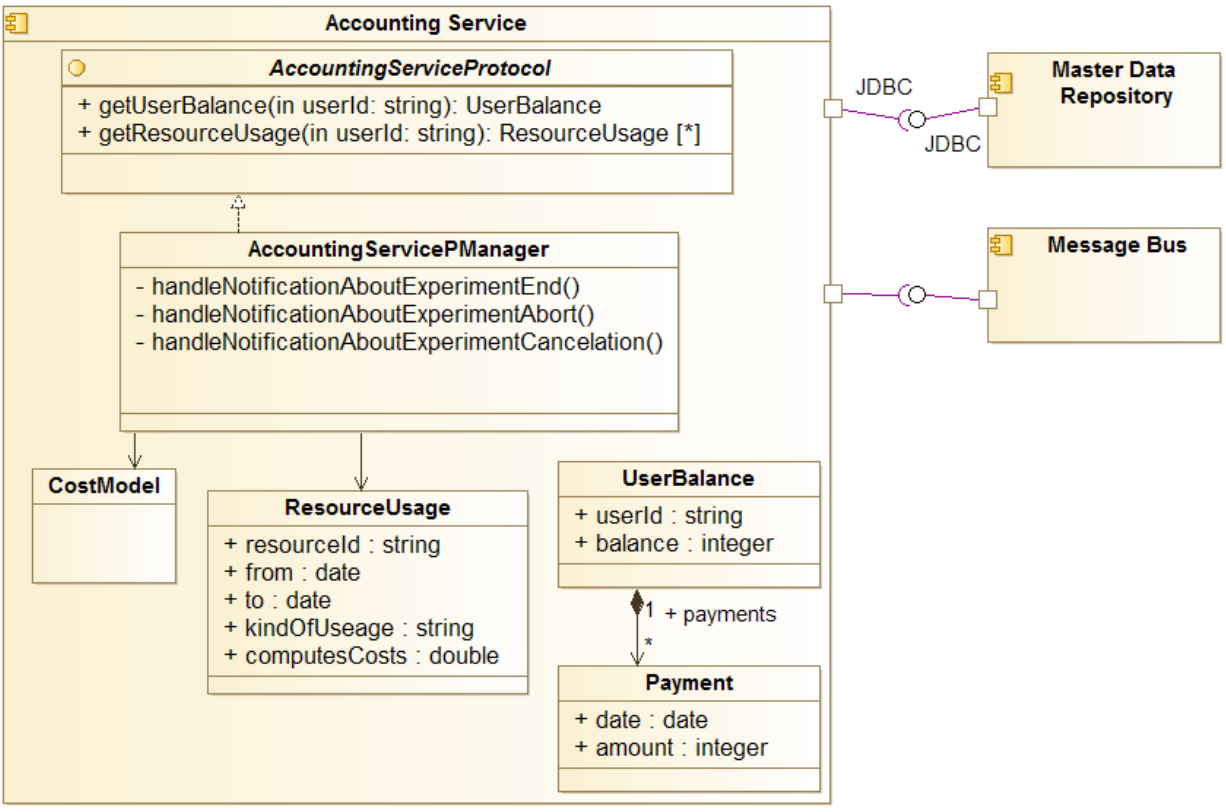

Figure 57: Accounting Service – Class diagram

Error! Reference source not found.

Picture above shows an abstract overview of the possible elements of the Accounting Service. Accounting and billing is common task where several solutions exits, e.g. *Kill Bill*[6], *Opencell*[7], *jBilling*[8], *OpenSourceBilling* [9], *InvoicePlane*[10], *MEVEO*[11] or *BillRun*[12]. These solutions will be evaluated and tested in the upcoming iteration, how far they fit in the RAWFIE ecosystem. If no fitting or adaptable solution is found, an own application for RAWFIE will be developed.

Interactions and relationships with other components

Required Interfaces

- Master Data Repository (JDBC)
  - Load data about booked resources
- Message Bus
  - Get notification about finished, cancelled and aborted experiments.

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.

Component requirements as identified in D3.2

ID (Priority)	Description	Requirement Mapping with component's functionalities
PT-EXP-C-001 (HIGH)	Cancellation of running experiments should be possible	Validator will do this at the second integration of development
PT-EXP-C-002 (MEDIUM)	Experiment Controller shall allow experimenters to remotely navigate UxVs.	Validator, Registrator and ExperimentFeasibility will do this at the second integration of development
PT-EXP-C-003 (HIGH)	The Experiment Controller shall support the execution of experiments that involve multiple testbeds	Registrator and ExperimentFeasibility will do this at the second integration of development
PT-EXP-C-004 (HIGH)	The Experiment Controller shall be able to support multiple experiments running the same time in parallel	Validator, Registrator and ExperimentFeasibility will do this in future
PT-EXP-C-	The Experiment Controller	Validator will do this at the second integration

[6] http://killbill.io/
[7] http://opencellsoftware.com/
[8] http://www.jbilling.com/
[9] http://opensourcebilling.org/
[10] https://invoiceplane.com/
[11] http://manaty.net/products/meveo-open-source-billing-system
[12] https://billrun.com/

005 (HIGH)	shall be able to analyse the whole experiment script and dispatch the appropriate parts to each responsible testbed facility	of development
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	Agent and Validator will do this at the second integration of development
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	Validator will do this at the second integration of development
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	Agent will do this at the second integration of development
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	Agent and Validator will do this at the second integration of development

Responsibilities

The main responsibilities of the Experiment Controller are:

- Control the distibuted status of the experiment
- Transfers the instructions from the launching service to the Resource Controller
- Execute the necessary actions to stop/cancel a running experiment on request
- Transfer the instructions from the UxV Navigation Tool to the Resource Controller
- Transfer the data from the Resource Controller to the Experiment Monitoring Tool

Operations and attributes

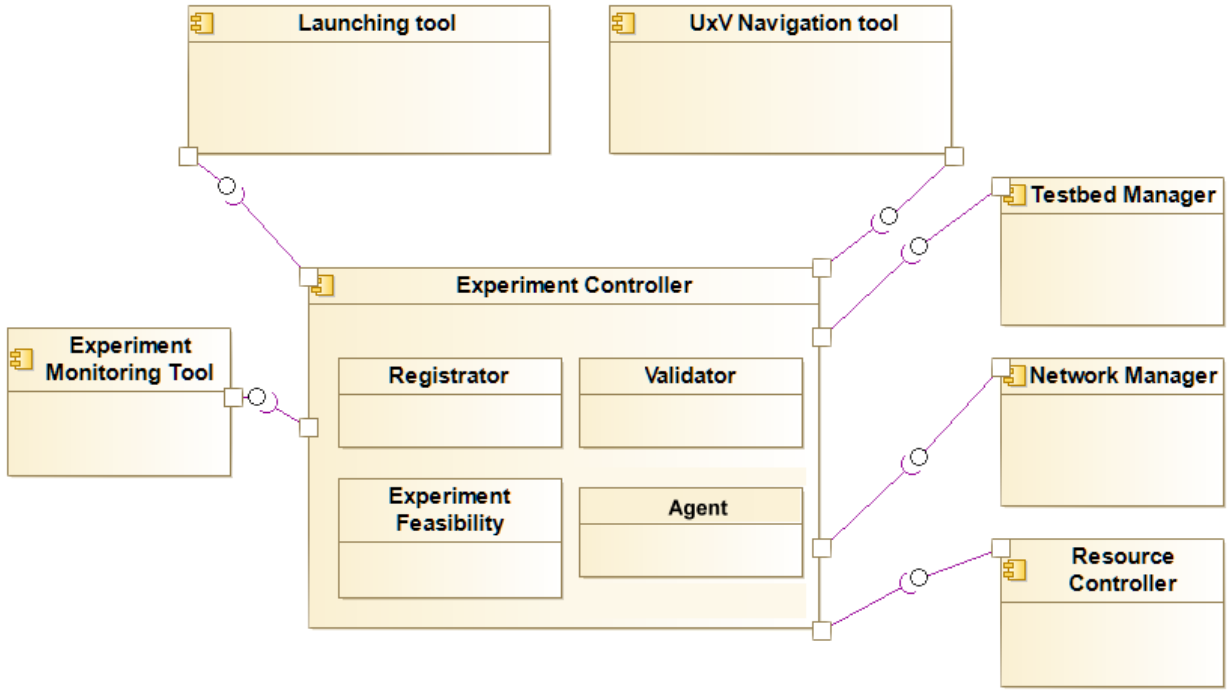

Figure 58: Experiment Controller - Class diagram

Register an Experiment

1. Registrator Class interacts with the Testbed Manager as to register the testbed to the middle tier.

Checks the Availability of the Resources

The Experiment controller is responsible for the evaluation of the feasibility of the experiment.

1. User initializes the experiment using the Launching Tool or the UxV Navigation Tool. The Experiment Feasibility class evaluates the user's preferences and inform the Experiment Monitoring tool about the ability of the system to perform such an experiment.

Transfers the user's instructions from the Launching Tool to the Resource Controller.

1. Validator Class validates the format of the provided JSON file
2. Transfers the file to the Resource Controller.

Transfers the user's instructions from the UxV Navigation Tool to the Resource Controller.

1. Validator Class validates the format of the provided JSON file
2. Transfers the file to the Resource Controller.

Stop/cancel a running experiment on request

1. The user clicks on the "Cancel" button (from the Experiment Monitoring Tool).
2. Experiment Controller transmits an appropriate message so as to cancel the mission.

Inform the Experiment Monitoring Tool

The Experiment Controller shall be able to continuously feed the front-end tier (Experiment Monitoring Tool) through the Agent, giving the experimenter a clear view of the experiment workflow as a whole

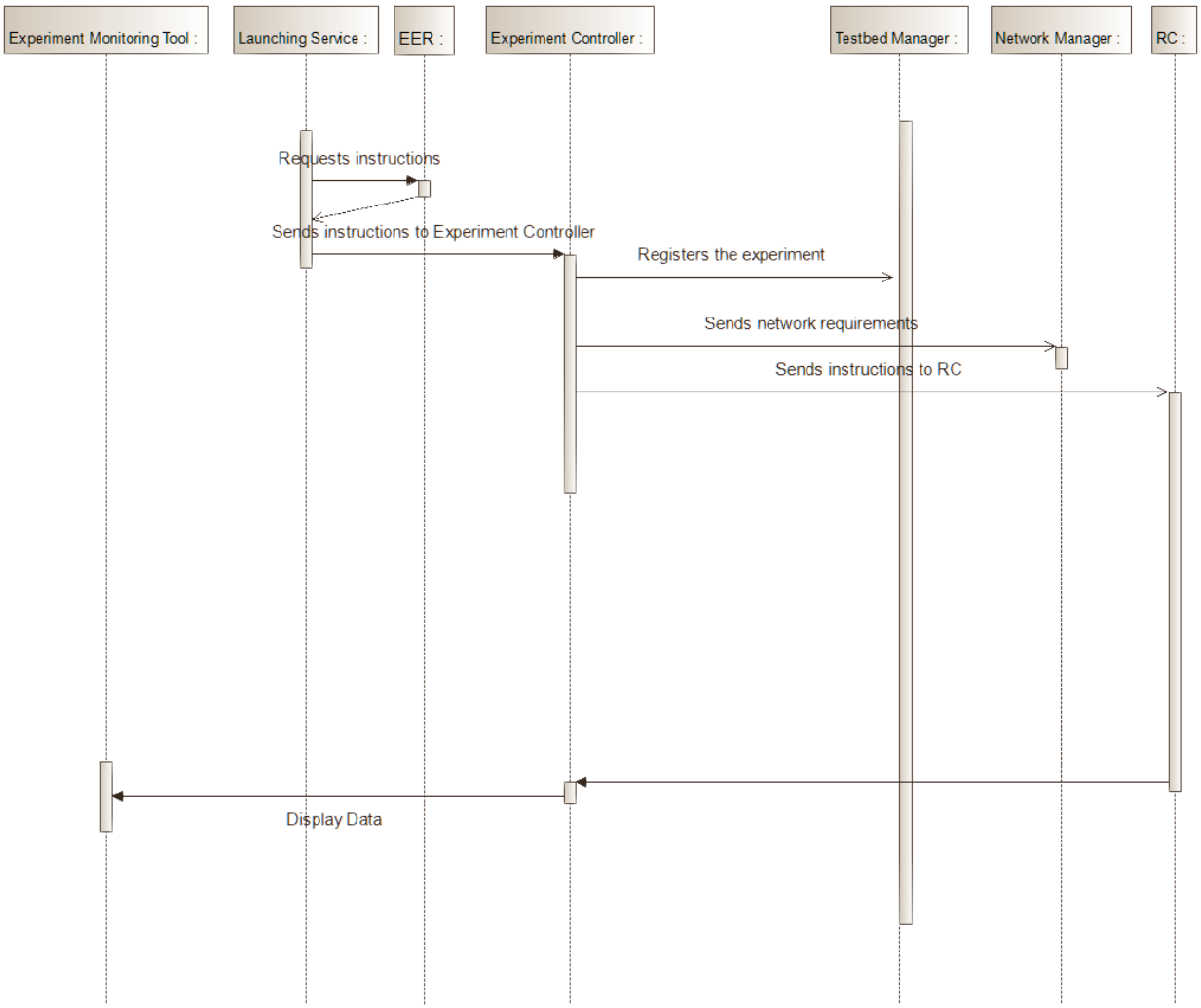

Figure 59: Experiment Controller – Sequence diagram

Interactions and relationships with other components

Required Interfaces

- Launching Service: This interface is mandatory so as to initialize the experiment and to transfer the user's instructions to the Experiment Controller.
- UxV Navigation Tool: This interface is mandatory so as to initialize the experiment and to transfer the user's instructions in case of UxV Remote Control to the Experiment Controller.
- Testbed Manager: Experiment Controller utilizes this component as to register the testbed to the middle tier.
- Network Manager: Experiment Controller triggers Network manager for the provisioning of the network connections during the experiment between the nodes.
- Experiment Monitoring Tool: This component displays the current status of the experiment. Additionally, Experiment Monitoring tool allows the experimenter to stop/cancel a running experiment
- Resource Controller: Experiment Controller forwards the instructions for experiment to the Resource Controller.

4.3 Testbed Tier (Testbeds and Resources control components)

4.3.1 Description

This subsection describes the Testbed Control, monitoring and analysis components. The Network Controller manages the network connections and the switching between different technologies in the testbed. The Monitoring Manager Component is responsible for the micro-management of the resources. 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. 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.
The main components of the navigation system are the Experiment Controller and the Resource Controller which ensures the safe and accurate guidance of the UxVs based on the user's preferences. More details about the Testbed Tier components are given in the following subsections.

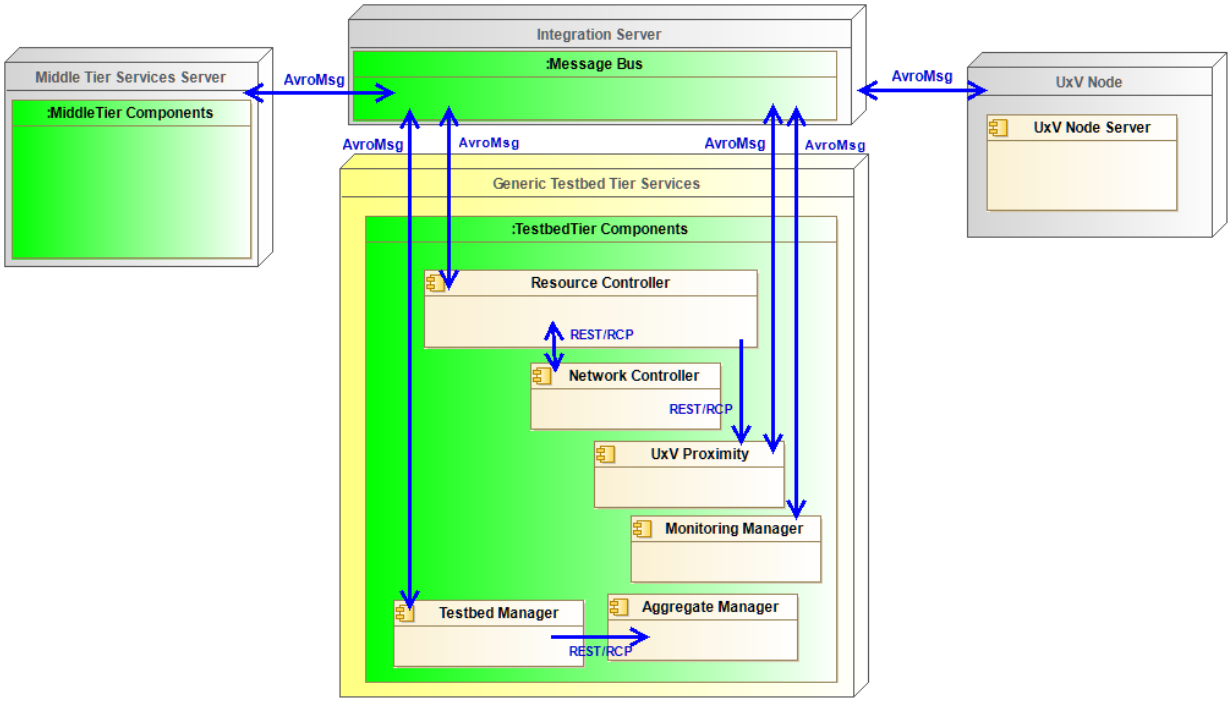

Figure 60: Testbed control, analysis and monitoring– Deployment / Components Diagram

4.3.2 Monitoring Manager

Component requirements as identified in D3.2

ID (Priority)	Description	Requirement Mapping
TB-MOM-001	The Monitoring Manager component should be able to	

(HIGH)	provide information about the capabilities of each resource node.	
TB-MOM-002 (HIGH)	The Monitoring Manager component should collect and report current status of testbed facilities	
TB-MOM-003 (HIGH)	The Monitoring Manager component should store periodically all testbed information	
TB-MOM-004 (HIGH)	Testbed monitoring manager should be able to transmit the current status to the System Monitoring Service.	

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.

Responsibilities

The main responsibilities of the Monitoring Manager are:

- Periodically check the current status of the available resources in the facility like battery lifetime, CPU load, free RAM, bit error rate, etc.
- Periodically check the status of the testbed facilities like weather conditions, network connections available, etc.
- Store the status of the testbed characteristics and the devices in a data log.
- Transmit current status information to the System Monitoring Service (as special plugin)

Operations and attributes

The operations of the Monitoring Manager are shown in the class diagram below. The following methods are implemented inside Monitoring Manager:

collectUxvStatuses(): tries to connect to all UxVs in the testbed to read properties like battery lifetime, CPU load, free RAM, bit error rate, etc.

collectTestbedStatuses(): reads the available testbed properties like weather conditions or network connections

logData(): logs the data in a log file.

Also the *IMonitoringPlugin* interface is implemented, so the System Monitoring Service can read status data of the testbed.

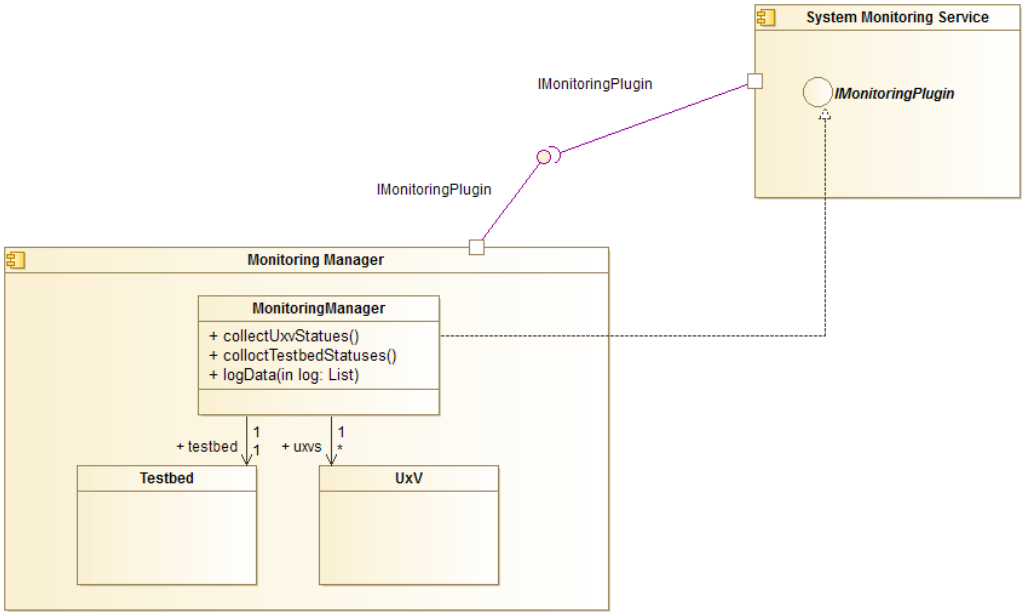

Figure 61: Monitoring Manager – High level class diagram

The interactions of the Monitoring Manager are presented in the following sequence diagram. Periodically it collect UxVs and testbed status information. This information is logged inside the component. Via the *IMonitoringPlugin* interface the System Monitoring Service can load – relevant data for the current status of testbeds and their resources.

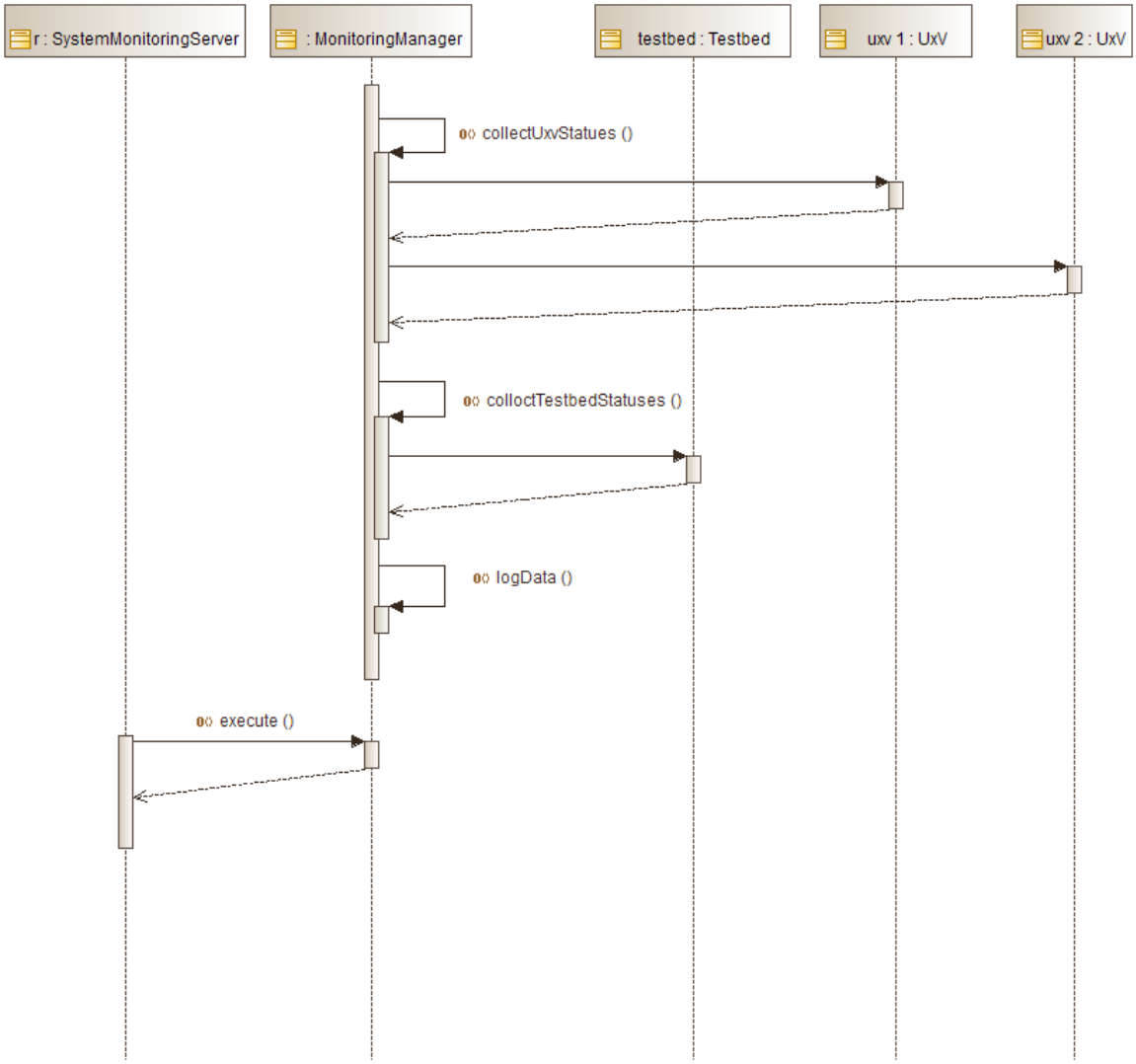

Figure 62: Monitoring Manager – Monitoring sequence diagram

Interactions and relationships with other components

Provided interfaces

- IMonitoringPlugin
  The collected status information is read by the System Monitoring Service.

Required interfaces:

- Access to testbed and UxVs
  The Monitoring Manager communicates with all UxVs in the testbed and the testbed itself to read their status information.

4.3.3 Network Controller

Component requirements as identified in D3.2

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
TB-NEC-001 (MEDIUM)	The RAWFIE communication resources shall be managed to offer seamless connectivity in the normal operations of the system.	
TB-NEC-002 (MEDIUM)	Provision of network communication resource	
TB-NEC-003 (MEDIUM)	Alternative communication system	
TB-NEC-004 (MEDIUM)	Management of the communication system	
TB-NEC-005 (MEDIUM)	Time constraint verification and notification	

Responsibilities

The main responsibilities of network controller are:

- Provision of the network connections/technologies required during an experiment
- Enable switching between network technologies
- Check the communication when devices are moving between obstacles
- Verification that the time constraints specified on the exchanged data for the different types of UxVs are met.
- Sends notifications produced in message bus to Resource controller and System Monitoring Service when the time constraints are not met via a specific network
-

Operations and attributes

The main operations of network controller are to start and to stop network connections between UxVs and the testbed for each experiment. When an experiment starts, Resource Controller via an implemented interface informs the network controller to start the function InitiateConnection(). When an experiment stops then Resouce Controller informs the Network Controller to stop network connection between UxVs via function StopConnection(). For each experiment Network Controller checks periodically with CheckStatusConnection() the state of the network. In case that UxVs should change from one network technology to another, Network Controller runs a decision support tool OptimalNetworkConnection() in order to decide which technology should be initiated..

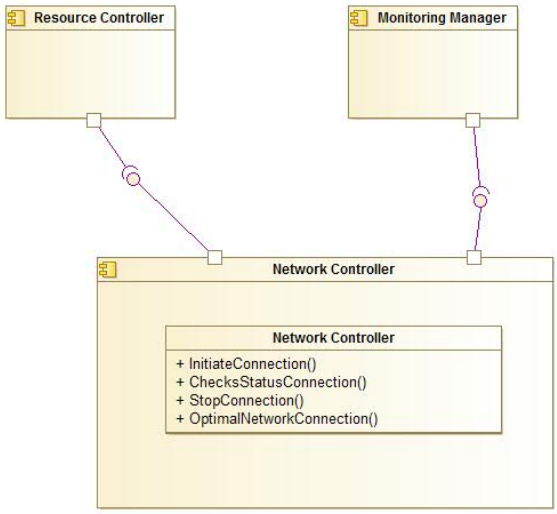

Figure 63 - Network Controller Class Diagram

Starting an experiment

1. RC sends to NC request to initiate a network connection
2. NC initiates the specific network control and answers to RC
3. NC periodically reports the state of the network
4. RC sends a request to stop the network because the experiment will stop

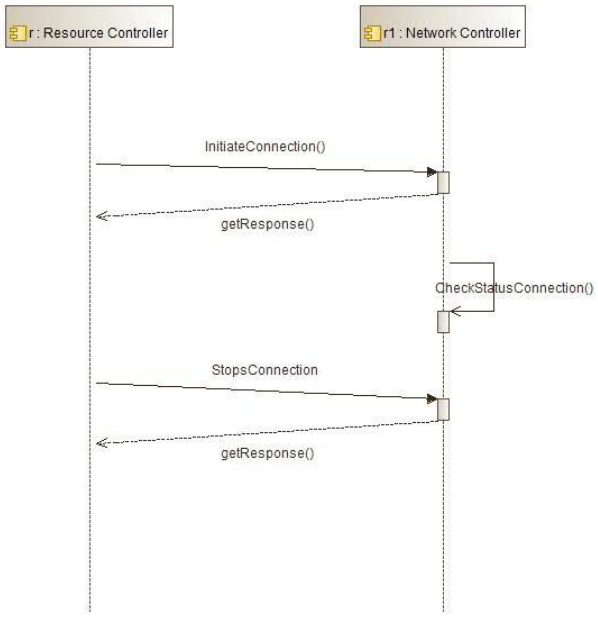

Figure 64 – Starts New Experiment Connection Provisioning

Dynamically change of network technology

1. RC sends to NC request to initiate a network connection
2. NC initiates the specific network control and answers to RC
3. NC periodically reports the state of the network
4. NC is triggered for low performance of network

5. NC periodically checks network connection OptimalNetworkConnection() in order to decide if the quality of the connection is good and if not which connection should be established between the UxVs
6. NC informs RC that stops the network connection
7. RC sends to NC request to initiate a network connection
8. NC initiates the specific network control and answers to RC
9. RC sends a request to stop the network because the experiment will stop

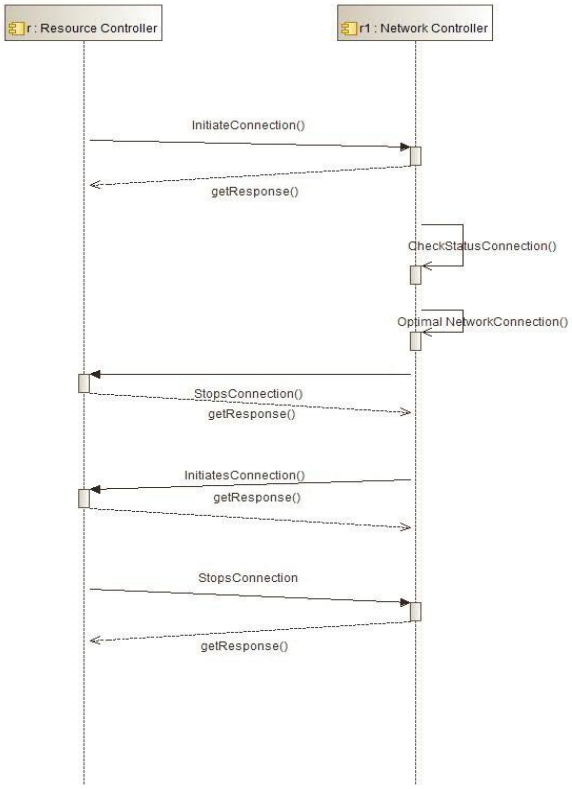

Figure 65 - Change Connection during an experiment

Interactions and relationships with other components

The Network Controller interacts with the Resource Controller in order to acquire information from the UxVs.

Monitoring Manager can also gather statistics for the network technologies and status of the experiments.

4.3.4 Resource Controller (plus Navigation Service sub-component)

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.

Component requirements as identified in D3.2

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 is responsible for this feature
TB-REC-002 (MEDIUM)	RAWFIE platform should be able to activate the "Emergency Scenario"	Event_Handler, will do this in future
TB-REC-003 (HIGH)	The Resource Controller shall receive location messages from the vehicles at regular intervals	Navigation_Service is responsible for this feature
TB-REC-004 (HIGH)	The Resource Controller shall transmit the next location for the current experiment to the vehicles	Navigation_Service is responsible for this feature
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	Navigation_Service is responsible for this feature
TB-REC-006 (HIGH)	For the experiment accomplishment the Resource Controller shall operate in close coordination with the Experiment Controller	Navigation_Service and Event_Handler will be responsible for this feature

Responsibilities

The main responsibilities of the Resource Controller are:

- Translate and transmit the experimenter's instructions to the vehicles
- Calculate the optimum trajectory for each vehicle
- Resource controller evaluates user's preferences and calculates the near-optimal path that the vehicles should follow in order to reach the desired location.
- The Resource Controller is able to detect and identify possible safety violations. If the given instructions violate the safety constraints, for example, the experimenter guides 2 units at the same position, the Resource Controller ignores these directions and returns appropriate warning messages to the user.
- The path planning algorithm takes into account the location of all the UxVs

Operations and attributes

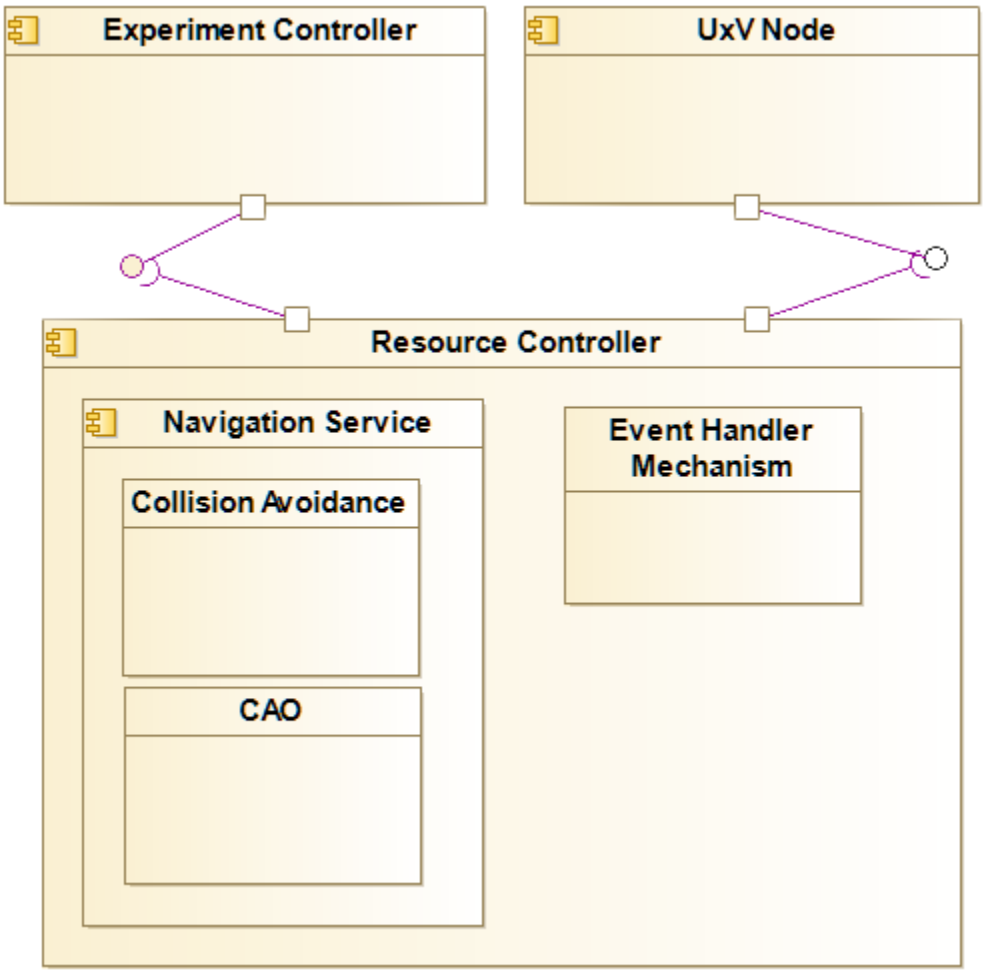

Figure 66: Resource Controller – Class diagram

Navigation of the UxVs

1. For each waypoint, the Resource Controller evaluates the desirable position and calculates the near-optimal path that the vehicles should follow in order to reach this location. CAO (Cognitive Adaptive Optimization) class is responsible for the calculation of the trajectory. CAO class is part of the Navigation Service sub-component
2. The path planning algorithm takes into account the current location of the vehicles, the model of the UxVs, navigational obstacles, the system dynamics etc. Collision Avoidance class is responsible for this task.
3. The Resource Controller translates this path into a sequence of waypoints and transmits a compact file with the desired coordination and the orientation of the vehicle to the UXV node.
4. At each time-step, the Resource Controller transfers only one waypoint to the UxV devices through the corresponding communication interface (using the message bus)
5. When all the UxVs reach the desired location they inform the Resource Controller regarding their current location, their orientation and their battery level.
6. The Resource Controller, taking into account the actual location of the vehicles recalculates the near-optimum path and transmits to the UxVs the next set of waypoints
7. The turn concludes when all the units reach their final location.

8. At each timestep, the Resource Controller interacts with the Experiment Controller, so as to inform the Monitoring Tool about the status of the experiment.

Protection of the Equipment:

1. If one of the following conditions occurs, automatically, the component activates, through the Event Handler Mechanism class an emergency scenario.
    - The component does not receive any feedback from the units for several time steps
    - The component receives feedback from the units which report severe localization issues
    - The component identifies crucial low battery levels
2. In such a situation, the Resource Controller navigates the units back to a safe position, as soon as possible.
3. The experimenters receive appropriate warning messages through the Experiment Monitoring Tool

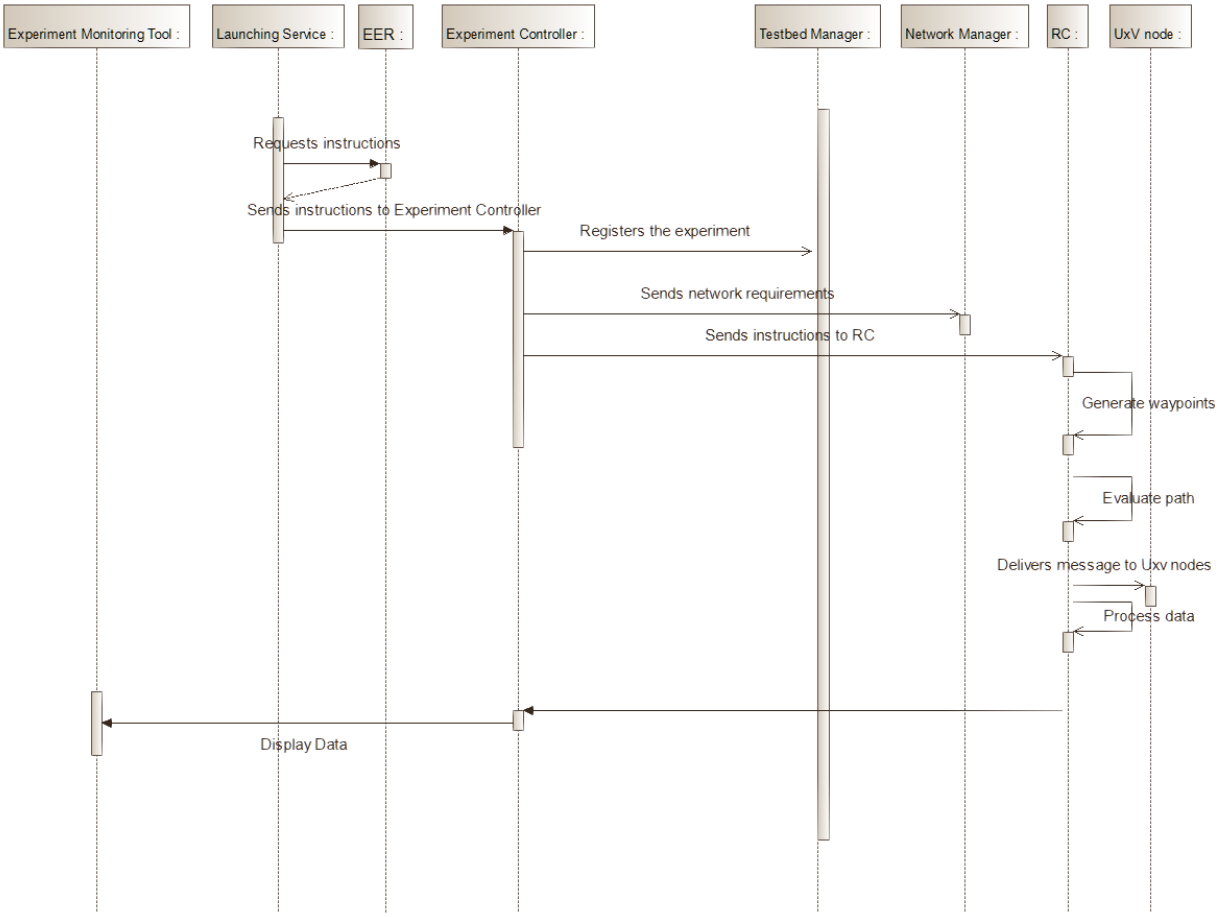

Figure 67: Resource Controller – Sequence diagram

Interactions and relationships with other components
Required interfaces:

- Access to testbed and UxVs
  The Resource Controller communicates with all UxVs in the testbed and the testbed itself to navigate the vehicles
- Access the Experiment Controller: So as to read the user's instructions
- Access the Monitoring tool: So as to inform the experimenters about the current status of the experiment

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.

Component requirements as identified in D3.2

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 Proximity component exposes the perceived

	the vehicles at regular intervals	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-008 (MEDIUM)	Neighbouring UxV monitoring	Neighbours detection, distance or proximity 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

Responsibilities

The proximity component is a local communication mean offering neighbour discovery and publish-subscribe services. Its responsibilities are:

- Translate subscriptions received from other nodes through the proximity component radio interface and forward them to the other UxV components.
- Subscribe to and receive topics published by the other UxV components.

- Access to some UxV component properties such as identifier, status, etc…
- Translate subscriptions coming from other UxV component into proximity component subscriptions.
- Forward data received from the proximity component radio interface to the other UxV components subscribing to it.

The discovery of neighbours is provided by the subscription to some identification service.

Operations and attributes

The UxV Proximity component provides an interface available to all inner UxV components for the subscription to topics published by other robots over their secondary communication interface (the proximity component radio). The proximity component is shared into two parts: the Proximity Head implements the publish-subscribe service on the secondary communication interface controller. The Proximity Delegate is the part of the component that runs on the UxV main computer alongside the other UxV components.

In particular, the Proximity Head executes the Publish-Subscribe protocol, forwards uplink subscriptions (coming from other UxVs) to the Proximity Delegate and transmits publications form its own UxV submitted by the Proximity Delegate. 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.

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”.

The class diagram in the following picture shows the decomposition of the Proximity component into head and delegate. The deployment diagram in the subsequent picture, shows how the proximity component decomposition fits into hardware.

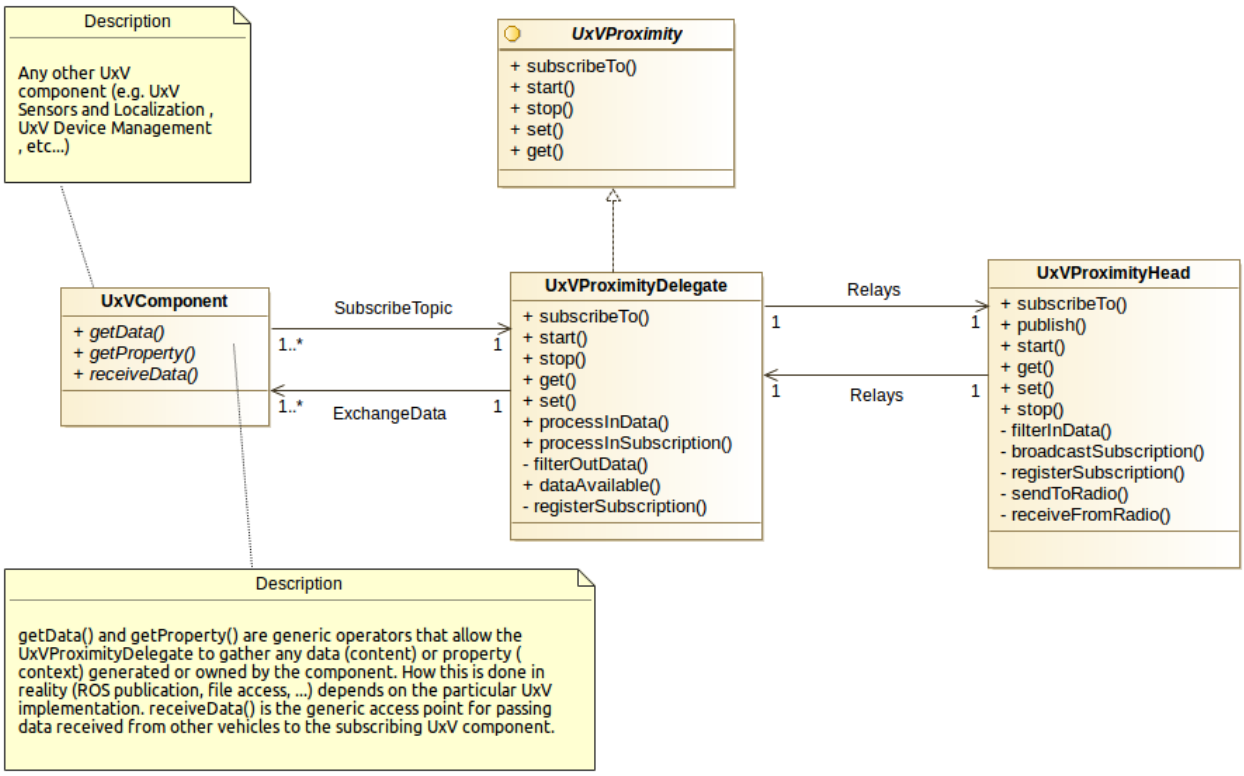

Figure 68: UxV Proximity component class diagram

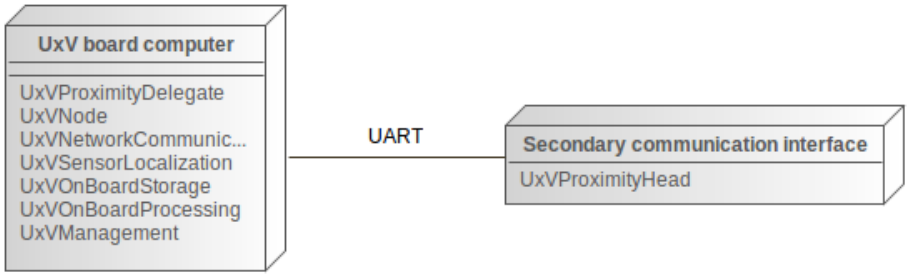

Figure 69: UxV Proximity Deployment diagram

The proximity component operation available the other UxV component are given below. The data and parameter types are generic because what they are in practice depends on the V implementation.

Content and context based subscribe service:

The subscribe service is an extension from the corresponding primitive in CCBR [11] in the sense that the lease time parameter (duration of a subscription) is extended to a full "delivery specification" containing lease time and real time constraints as well as maximal traffic parameters. Real time constraints pertain to performance metrics such as delivery period, latency and jitter. The maximal traffic parameters describe the maximal throughput allowed for a given subscription.

The additional data parameter of the CCBR subscribe primitive is removed here for two reasons. The first reason is that the delivery specification contains enough information for the publishing node to infer which system or sensor it should activate to be able to produce the data. The second reason is that in the world of Rawfie, a command may be issued at any time, not only when a subscription is started like in CCBR. Thus it is better to pass commands to the publish service primitive.

```
subscribeTo(contentFilter df, contextFilter pf, deliverySpec ds, receiver rc)
```

`contentFilter df`	description of the data topic which is subscribed to (topic name and operators on values).
`contextFilter pf`	description of the requested publisher context as a filter on an arbitrary number of system properties.
`deliverySpec ds`	instructions for the publisher on the size, rate and timing minimas and maximas that are requested for a valid publication. If a candidate publisher cannot respect the delivery specification, it does not publish the requested topic.
`receiver rc`	reference to the handler to which incoming data shall be delivered (e.g. receive call-back)
Return value	Success or error code.

Publish service

The publish primitive takes a topic name and a reference to the data to be published. The publish-subscribe layer is responsible for filtering.

```
publish(topic t, data d)
```

`topic t`	Topic name.
`data d`	Reference to the outgoing data.
Return value	Success or error code.

Note that the above primitive can be used to send a command to another UxV (or a set of UxVs), that had previously subscribed to it (i.e. expressed its interest to receive commands). As a matter of fact, topics can be defined to establish communication "pipes" across the UxV swarm for many purposes, including control.

Start

Enable the proximity component, start the services.

```
start(deviceManagement m)
```

`deviceManagement m`	Reference to the UxV Device Management component for access to the UxV properties.
Return value	Success or error code.

Stop
Disable the proximity component, stop all its services.

```
stop()
```

Return value	None.

Set
Modify a property of the proximity component.

```
set(objectID oid, value v)
```

`objectID oid`	Identifier of the property to be modified.
`value v`	New value of the property.
Return value	Success or error code.

Get
Get the current value of a proximity component property.

```
get(objectID oid)
```

`objectID oid`	Identifier of the property to be read.
Return value	Current property value.

Data available
Pass data produced by a UxV component to the Proximity Delegate. Depending on the implementation, the operation may be implicit (return of the *getData()* operation on the UxV component).

```
dataAvailable(data d, topic name t)
```

`data d`	Data sample or array
`topic t`	Data topic identifier
Return value	Current property value.

Internal operators

Some of the UxV Proximity component internal operators are shown as public in their respective classes because they are either Proximity Head operators used by the Proximity Delegate or Proximity Delegate operators used by the Proximity Head:

Proximity Delegate:

- *processInData():* used by the Proximity Head to pass incoming data that corresponds to an on-going subscription to the delegate. The delegate forwards the data to the

subscribing component. How this is done practically depends on the particular UxV implementation.

- *processInSubscription():* when the Proximity Head receives a subscription from a distant UxV, it calls the Delegate method processInSubscription() so that the relevant data can be gathered from the producing UxV component.
- *filterOutData():* this internal operation of the Delegate filters the data produced by the other UxV component according to the on-going subscription so that it is ready to be published by through the Proximity Head.
- *registerSubscription():* record the subscription submitted by a UxV Component calling *subscribeTo()*.

Proximity Head:

- *publish():* this is the send function of the Proximity Component. It is called by the Delegate to trigger the publication of filtered data.
- *filterInData():* this Proximity Head operation filters the incoming traffic according to the current subscriptions of the UxV. The data is then passed on to the Proximity Delegate through processInData().
- *broadcastSubscription():* broadcast a subscription on the secondary communication system. This operation is not necessary if subscriptions are only registered locally and publications are spontaneous regardless of on-going subscriptions.
- *registerSubscription():* record a subscription submitted by the Proximity Delegate through *subscribeTo()* to allow the filtering of incoming data.
- *sendToRadio():* submit outgoing data to the secondary communication system for transmission.
- *receiveFromRadio():* data reception call-back from the secondary communication system.

Interactions and relationships with other components

The Proximity Component shall be able to interact with any other component present on the UxV:

- A UxV component shall be able to register its interest with the Proximity Component to receive particular data (e.g. identification, position, speed) from another UxV. This is done through the *subscribeTo()* operation. UxV components shall have a *receiveData()* operation so that the Proximity Component can pass on incoming data that corresponds to a registered subscription.
- The properties, statuses and data flow produced by a UxV component that are of interest for direct publication to neighbouring UxVs shall be made available to the proximity

component. These operations are named *getData()* and *getProperty()* in the class diagram of the component

The communication between the UxV components and the Proximity component goes through the Proximity Delegate. How it is done in practice depends on the UxV implementation and operating system (e.g. ROS topic, direct calls, file access, …).
The UxV shall provide a UART interface (hardware and software) to the proximity component radio (i.e. the local communication system).

Sequence diagram: neighbours discovery and position

The following sequence diagram depict the operations of the proximity component by means of an example:
The UxV Node component of a UxV wants to know the position of other UxVs in its vicinities in real time. Therefore, it subscribes to the topic "position" with the filter "any UxV" and delivery specification "every 250 ms" at its Proximity Delegate. The subscription is propagated and eventually the neighbouring UxV publish the required data, which is then passed on to the UxV Node subscriber.
The sequence diagram of in the following picture shows the UxV Node subscribing to the position topic. The subscriber specifies the content (i.e. the topic name and filter), the context (the desired properties or status of the publisher) and the delivery specification (here the period). The Proximity Delegate registers the subscription and passes it on to Proximity Head which also registers it. Then the subscription is regularly broadcasted by the Proximity Head on the secondary communication system for the attention of potential publishers in the neighbourhood. If the number and throughput of the available topics are sufficiently limited, an alternative implementation could use local subscription only and the spontaneous broadcast of all available topics by publishers. In the opposite case, the actual dissemination of the subscriptions is necessary to avoid overloading the communication channel with unnecessary publications.

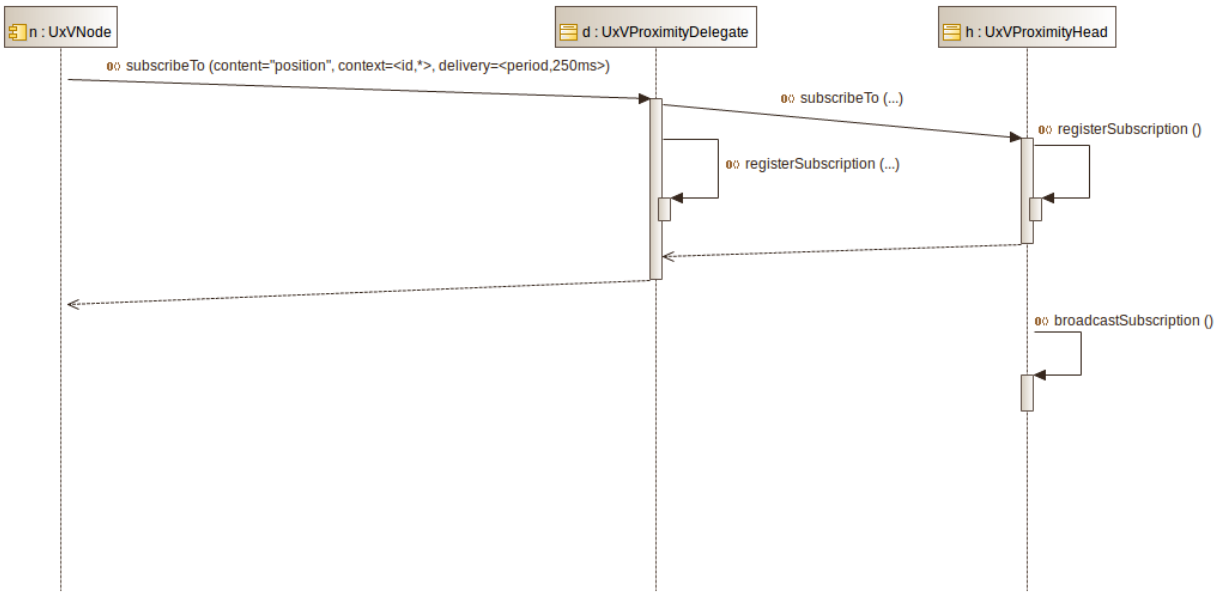

Figure 70: Sequence diagram - topic subscription at the Proximity component

The following sequence diagram shows how the received subscription is managed at a publisher and how the produced data is published. Following the example, the “position” topic is published by the UxV Sensor and Localization component.
Subscriptions received by the Proximity Head are passed on to the Delegate which gathers the data from the relevant UxV component. The Delegate is notified [*dataAvailable()*] each time new data is available. The data is filtered before being passed down to the Proximity Head which broadcasts it on the communication channel. Filtering at the delegate is used to ensure that data sent on the channel does not exceed the delivery specification of the registered subscriptions. For instance, if the subscribers wants a position every 250 ms and the data is produced by the UxV Sensor and Localisation every 5 ms, the *filterOutData()* operation of the delegate sends only one out of each 50 samples to the Proximity Head.
Filtering may also include compressing or encoding the data to limit bandwidth usage. Depending on the implementation on a particular UxV, the filtering may be implicit, e.g. when the *getData()* operation of the producer component allows selecting data according to the subscription delivery specification.

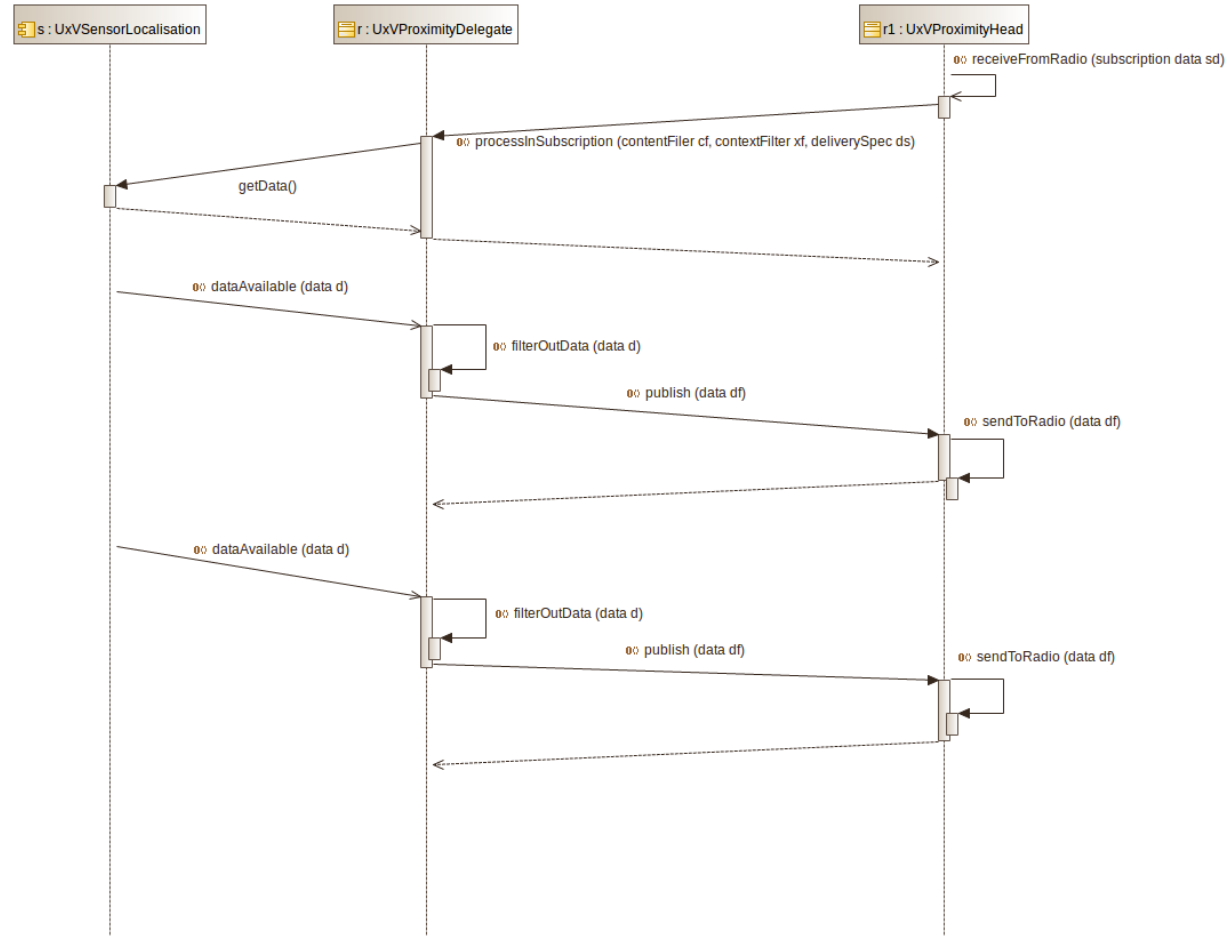

Figure 71: Proximity component subscription reception and data publication sequence diagram

Finally, the last sequence diagram depicts how data is received, filtered and dispatched at the subscriber. The *receive()* operation of the Proximity Head denotes the data reception call-back from the radio communication stack. The data is then filtered according to the registered subscription, passed on to the Delegate which dispatches it to the subscribing component.

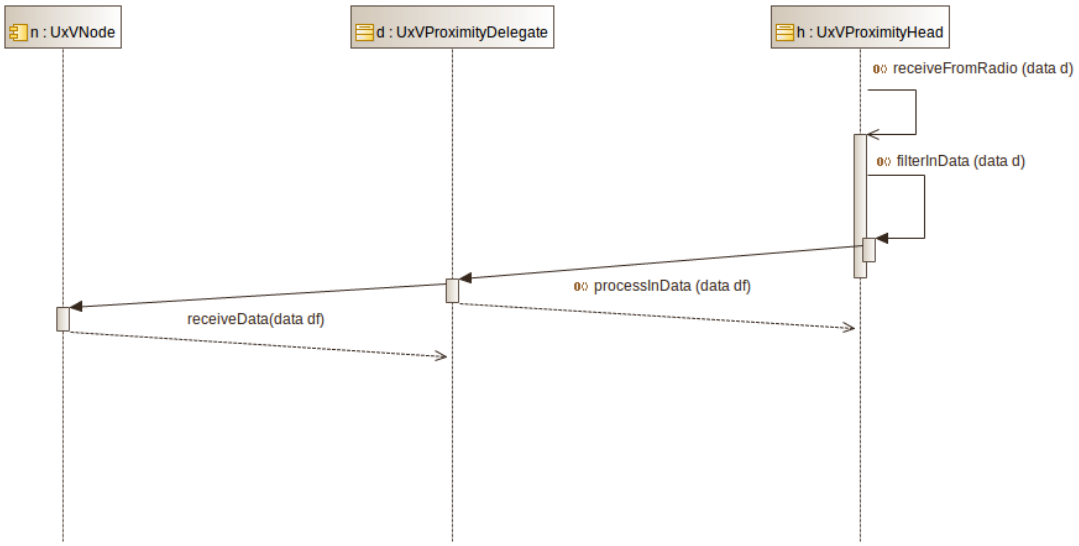

Figure 72: data reception at the proximity component sequence diagram

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.

Component requirements as identified in D3.2

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	This functionality is provided with the help LogExperiment() operation of the Experiment class
TB-MAN-005 (HIGH)	Testbed Manager shall be periodically informed about the status of all running experiments in the testbed	CheckExperimentStatus() (Experiment class) receives periodically the status of all running experiments from Resource Controller
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 transmission failure	StartStoreData() and StopStoreData() implement local storage in case of temporary communication failure
TB-MAN-009 (LOW)	Testbed Manager may provide statistical data/information about testbed operation	Testbed Statistics class provides this functionality

Responsibilities

The main responsibilities of the Testbed Manager are:

- Provide a user interface through which a user can add new testbed resources and view the capabilities of each resource node
- Periodically check the status of all other services running at testbed
- Keep a registration log for all the experiments in the relevant testbed
- Store configuration parameters for the UxVs in the relevant Testbed
- Buffer data in case of network connection loss to the Middle Tier. The TM stores the last instance of each experiment as a fall back mechanism in case that testbed loses the

connection with the middle tier. If there is a network problem during the execution of the experiments, TM stores the information that would be forwarded to the Data tier.

- Provide the required interfaces to interact with SFA Aggregation Manager

Operations and attributes

The structure of Testbed Manager is depicted in the class diagram of Figure 73, where the associations and operations of its core sub-classes are presented. A permanent storage in the form of a local database is implemented at testbed level with all classes enabled to access it. 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.

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. 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 of all UxV resources participating in the experiment and logs locally the start and stop as well as all emergency events of the experiment.

The Testbed Services class is responsible to collect the status of all the other components that run at testbed, inform the platform (System Monitoring Service) about the overall status and take the appropriate actions in cases of non-responding components. Emergency class contains all the functionality needed to activate local storage for the experiments execution in the case of network problem and connection loss with the middle-tier. Finally the Statistics class can present information about the utilization of the testbed or specific resources in user defined temporal intervals.

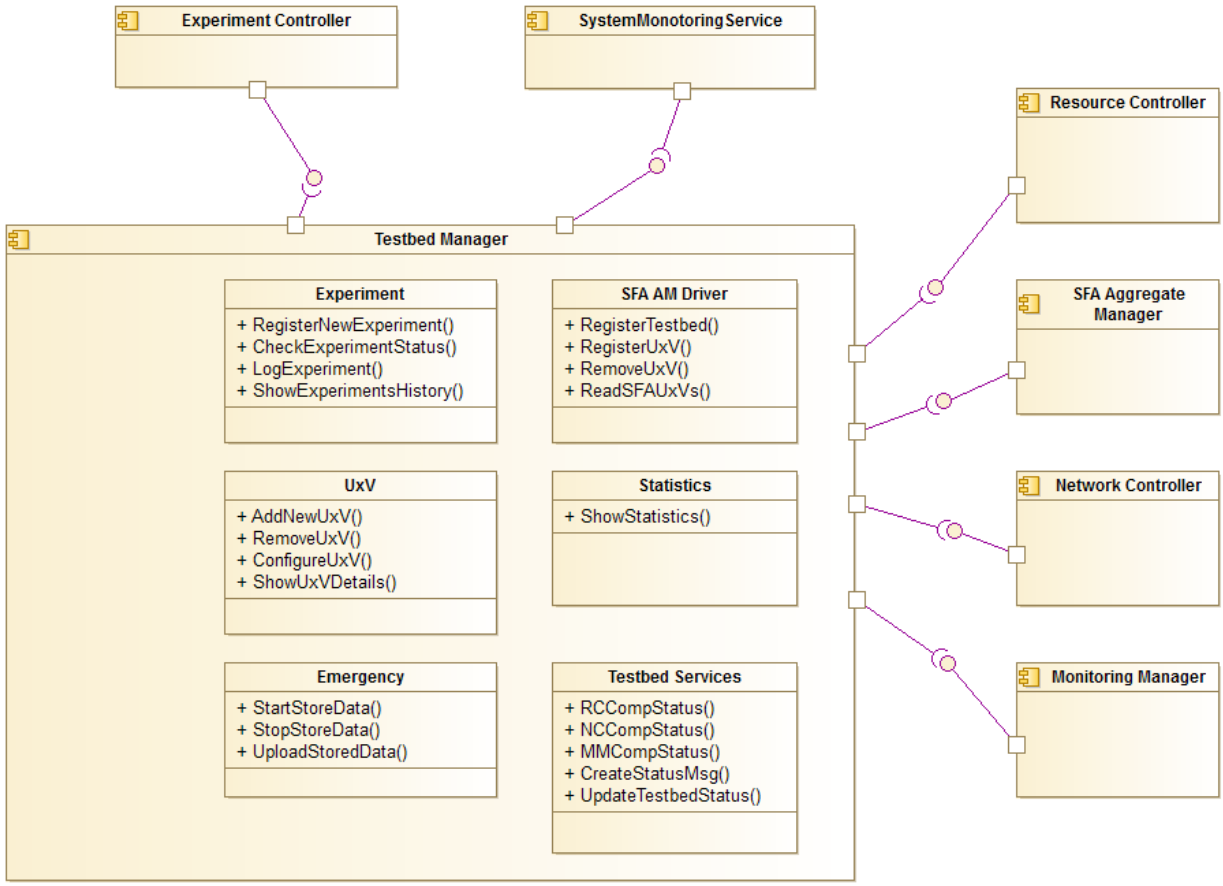

Figure 73: Testbed Manager – Class Diagram

Testbed level experiment handling

The sequence diagram of **Error! Reference source not found.** Figure 74 presents sequence of actions related to experiment handling at testbed level:

1. Testbed Manager receives StartNewExperiment message from platform Message Bus
2. RegisterNewExperiment() operation is called to write all required information about the new experiment in the local database.
3. Testbed Manager receives periodically a message about the current status of the experiment from Resource Controller
4. LogExperiment() operation is called to store in the local database a trace of experiment execution steps
5. A user can ask information about what experiments have been executed in the testbed within specific temporal intervals and get the results through ShowExperimentsHistory()

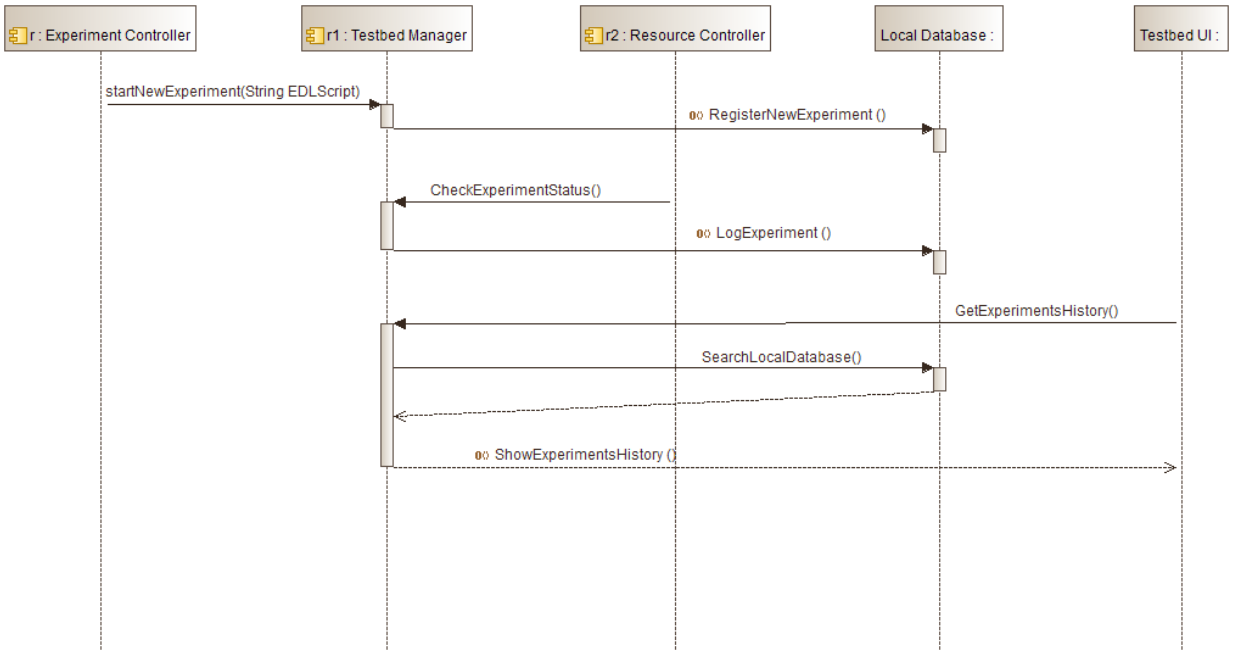

Figure 74:Testbed Manager experiment handling sequence diagram

Monitor Testbed components

The sequence diagram of **Error! Reference source not found.** Figure 75 presents the sequence of actions contained in the TestbedServices class of Testbed Manager:

1. Other components running at testbed level namely Resource Controller, Network Controller and Monitoring Manager send a message about their status at regular intervals using the local message bus.
2. Testbed Manager periodically creates an overall assessment about testbed components based on received messages and creates the status message using CreateStatusMsg()
3. Testbed Manager informs System Monitoring Service about the status of all testbed components using the platform Message Bus (UpdateTestbedStatus).

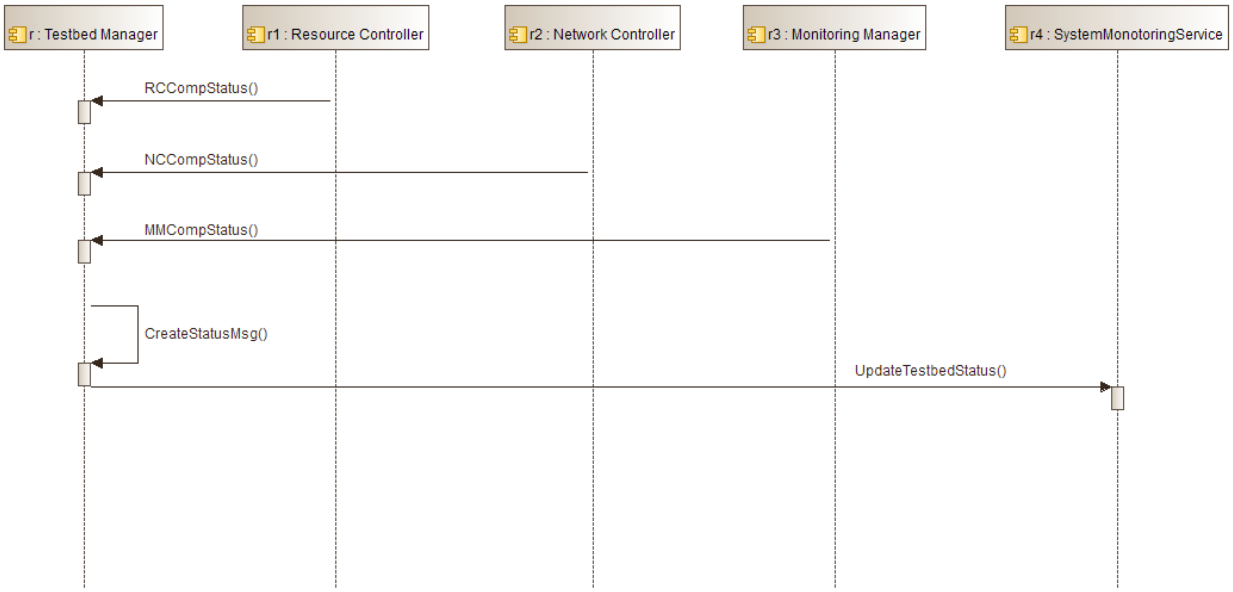

Figure 75: Testbed components monitoring sequence diagram

4.3.7 Aggregate Manager

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.

Component requirements as identified in D3.2

ID (Priority)	Description	Requirement Mapping
PT-GEN-R-001 (HIGH)	RAWFIE Platform should adopt Sliced Federated Architecture (SFA)	It would be implemented in the second development iteration cycle
PT-DIR-S-003 (HIGH)	The Testbed Directory Service shall provide access to information about available resources (UxVs) belonging to the testbeds registered in RAWFIE	It would be implemented in the second development iteration cycle

Responsibilities

The main responsibilities of the Aggregate Manager component are:

- Implements Aggregate Manager (AM) API in order to
  - Get a list of the available resources
  - To allocate resources to a slice
  - To retrieve a manifest Rspec describing the resources in a specific slice
  - To provision the state of the allocation of the resources
  - To de-allocate resources
  - To retrieve the status of the allocated resources by experimenters

- Publish all the aforementioned services of AM API to Web

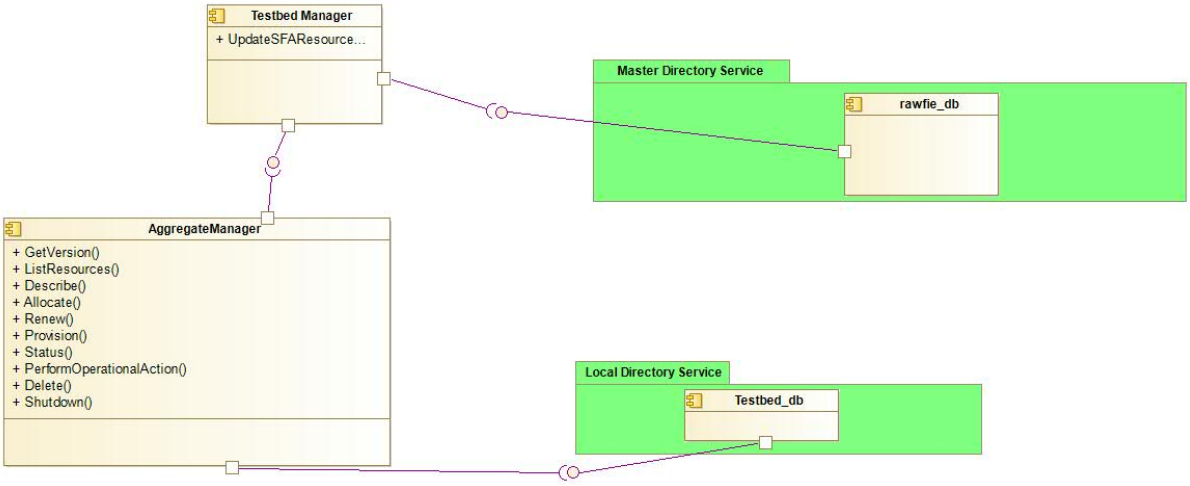

Figure 76: Aggregate Manager – Class Diagram

Get List of the Resources

1. Experimenter requests from SFA_Service the list of resources in the specific testbed
2. This request is forwarded to the AM of the specific testbed
3. AM checks the local database (Testbed_db) for the requested resources
4. AM returns the list of the resources if there is success. Otherwise an error message is returned to the SFA_service and consequently to the Experimenter.

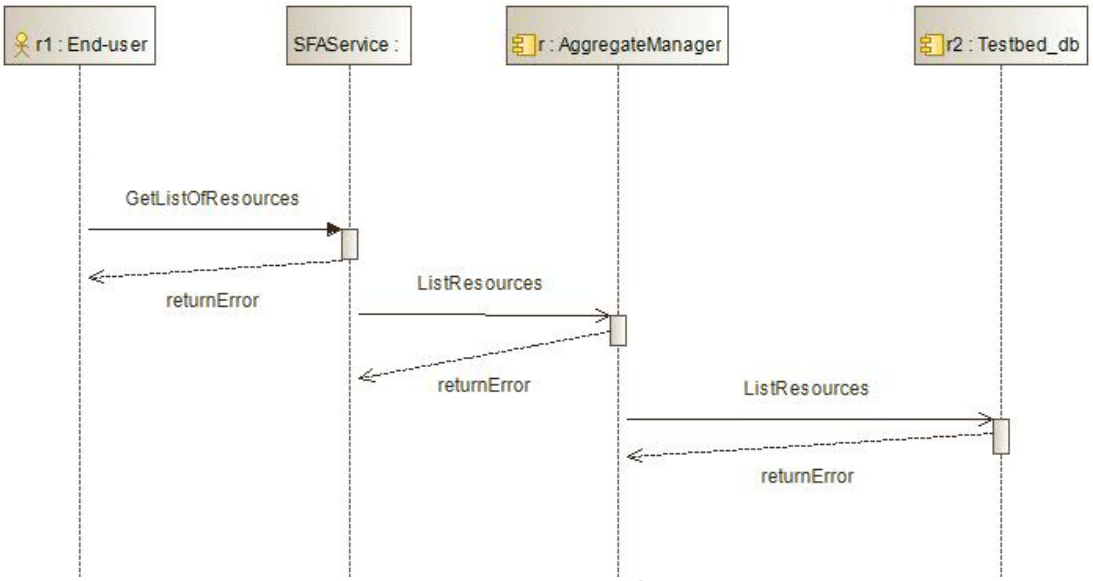

Figure 77: Aggregate Manager- get the list of available resources sequence diagram

Allocate Resources

1. Experimenter requests from SFA_Service to allocate specific resources.
2. This request is forwarded to the AM of the specific testbed

3. AM checks the local database (Testbed_db) for the requested resources
4. AM forwards the request to Testbed Manager for
    a. Check the Master Data Respository for the availability of the resources
    b. To allocate the resources in the Master Data Repository
5. In case of the successful allocation of the resources in the Master Data Repository, Testbed Manage returns the success message; otherwise an error message.
6. AM updates the local database based upon the answer of Testbed Manager

AM returns the answer to the SFA_service and consequently to the Experimenter

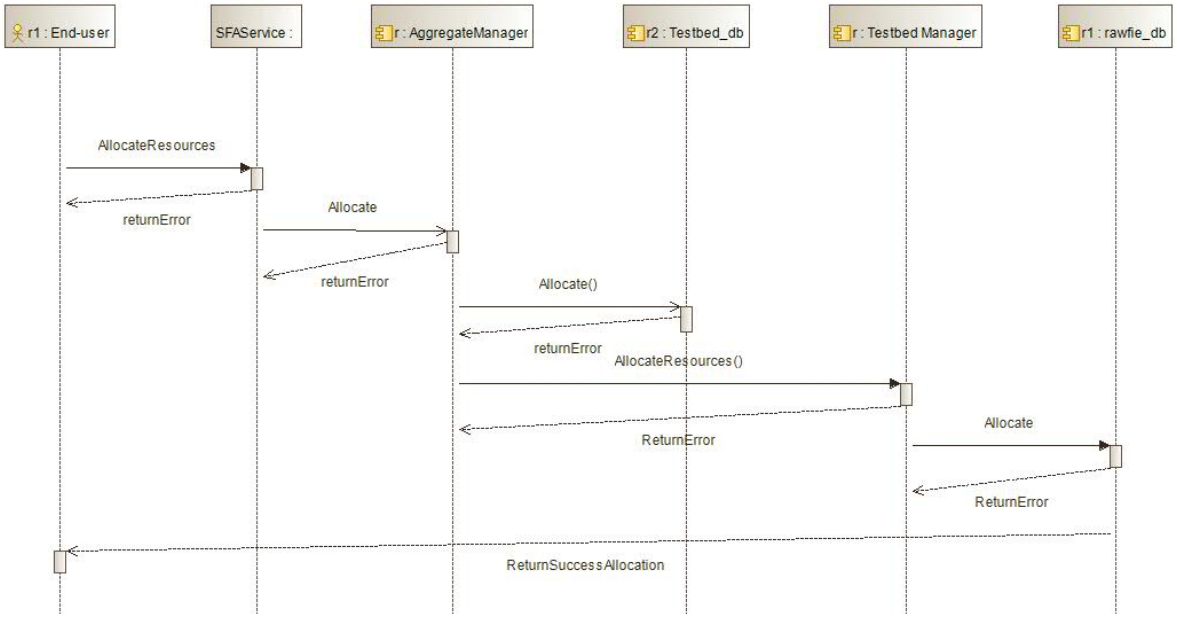

Figure 78 – Aggregate Manager allocate resources sequence diagram

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 Section 4.3.9). Examples of the implementation on two different platforms are given at the end of the section.
The basic requirements a generic UxVs Nove should satify, in order to make it possibile to use ot within the RAWFIE platform (to "plug-in" within an existing Testbed, for example), have been identified by the consortium partners and are listed in **Error! Reference source not found.**.
Supported sensors that UxV may embeds are defined in section 4.3.9.

TB-UVG-001	Compliance of UxV to RAWFIE specification and interfaces
UXV-NOD-001	Each UxV shall have a unique Identification code.
UXV-NOD-002	Each UxV node should ensure a minimum autonomy of 15-30 minutes
UXV-NOD-003	Each UxV node should ensure payload
UXV-NET-001	Capability of taking the control of the UxVs from distance
UXV-NET-002	UxVs should be able to Synchronizs their Time-References between them.
UXV-NET-003	The UxV should provide Access Point functionality
UXV-NET-004	Each UxV node shall be equipped with primary and secondary communication means.
UXV-NET-005	UxV network interface management
UXV-NET-006	UxV communication interoperability with RAWFIE (inbound)
UXV-NET-007	UxV communication interoperability with RAWFIE (outbound)
UXV-NET-008	Neighbouring UxV monitoring
UXV-NET-009	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
UXV-SEN-001	Each UxV node should tag location and timing capability to each sensor readings
UXV-SEN-002	Each UxV node shall be able to list the available sensors
UXV-SEN-003	UxV location and sensor data should be made available to the experimenter
UXV-SEN-004	Location sensors should be supported in each UxV unit and can be used remotely during testbed demonstrations
UXV-SEN-005	UxVs should sent a notification to the Resource Controller when they reach the desired location
UXV-STO-001	UxVs shall be able to **store data** on board
UXV-STO-002	UxV's shall provide a **management tool** of the available storage
UXV-STO-003	UxV's shall provide an **authorized access** to the data management tool.
UXV-STO-004	UxV's shall provide a **data log**
UXV-STO-005	UxV's may provide an **automated syncing** of servers
UXV-PRC-001	Each UxV shall be able to operate autonomously
UXV-PRC-002	The UxV should provide collision avoidance mechanism
UXV-PRC-003	Capability of task planning of the UxVs nodes during run-time
UXV-PRC-004	UxVs should be able to cooperate during the execution of an experiment
UXV-PRC-005	Each UxV node shall keep position while waiting for new instructions
UXV-MGT-001	UxVs shall offer on demand resources (Network, Sensor, Processing, and Controller).
UXV-MGT-002	UxV shall be capable to revert to a safe mode
UXV-MGT-003	UxV shall be capable to restart each component independently
UXV-MGT-004	UxV shall be capable to monitor the health of the system
UXV-MGT-005	UxV shall be capable to enable/disable each component
UXV-MGT-006	UxV shall be capable to offer safe maintenance access for manufacturers

Table 1: List of requirements for an UxV node to be used in RAWFIE

The UxV Node component provides an abstraction layer (e.g. software adaptors) to the unmanned vehicle to make it appearing as a RAWFIE compliant component. It provides interfaces to the robot operation resources such as setting the robot waypoints and speed or real-time remote control, but also every other resource of the robot accessible by the system.
In the present document and from now on, the common characteristics each UxVs should ensure are identified:

- The UxV Node is referred as the main (hardware / software) interface between the RAWFIE platform and the UxV's. According to the successfully tested implementation, this interface is programmed by robot manufacturers in order to provide a Message Bus adapter.
- Kafka Message Bus and the Confluent schema-registry are the tools that have been chosen for the communication with the RAWFIE platform components. A specific set of common messages has been defined to cover the general functionalities of UxV's within RAWFIE system.
- This set of messages is gathering not only the data expected from the robot but also the commands that the robot is expecting in order to operate.
- A common data format allows the system to access this data remaining agnostic of the robot operation. It is permitting also the use of a common database.
- The former UxV Components distribution still makes sense as a generic view of the expected functionalities of the UxV's.
- The following subset of components (**Error! Reference source not found.**) is no longer relevant for the whole RAWFIE platform but somehow interesting for the manufacturers and their singular approach for programming the UxV Node Interface.

	Main responsibilities
Node	Provide an interface to the robot control mechanisms (waypoints, speed, remote control) and publish the robot localisation information and odometry.
Network and Communication	Operate the vehicle network interfaces to provide reliable message exchange and data transfer services with external entities (testbed, other UxV's). Dispatch messages (including commands) to the intended UxV component. Handle subscriptions to data streams and transmit them.
Sensor and Localisation	Give a high-level interface to access the sensors installed on board the UxV. Publish the sensor readings and keep a description of all activated sensors.
On-board storage	Define a data stream abstraction that includes metadata. Allow data streams to be saved in permanent storage and/or be published for transmission.
On-board processing	Connect data streams to on-board processing algorithms and publish the resulting output after checking for sufficient computing and energy resources. Allow the installation of new data processing algorithm and keep a registry.

Management	Provide a centralised dashboard view and control of the UxV operations and resources. Keep a searchable registry of the UxV functions and resources.

Table 2: Summary of UxVs functions

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. The reference frame of the UxV is defined as follows and must be used in the messages described in the next sections.

- **Header**

All messages of the UxV Message API contain the same header, used to encode basic information about the dispatching entity.

Field	Units	Description
sourceSystem	-	Canonical name of the originating system
sourceModule	-	Canonical name of the module within a given system that originated the message
time	ms	Time elapsed since the Unix epoch

- **System Information**

Each UxV shall periodically dispatch general system information in the form of the System Information Message. This message allows other software modules to dynamically identify the UxV.

Field	Units	Description
Vendor	-	UxV vendor/manufacturer
Model	-	UxV model
Type	-	UxV type
Name	-	UxV canonical name
Owner	-	UxV owner's name

- **Sensor Information**

This message encodes basic information about a given sensor. This message allows other software modules to dynamically identify the sensors installed on a UxV.

Field	Units	Description
Vendor	-	Sensor's vendor/manufacturer

Name	-	Unique sensor name/model
Serial	-	Sensor's serial number
Types	-	List of quantities that the sensor is able to measure

- **CPU Usage**

Amount of CPU resources currently in use.

Field	Units	Description
Value	%	Amount of CPU resources used by the UxV on-board software and associated services

- **Storage Usage**

Measurement of storage usage

Field	Units	Description
Available	MiB	The amount of available storage
Used	%	Percentage of used storage

- **Fuel Usage**

Amount of available fuel

Field	Units	Description
Value	%	Amount of available fuel

- **Location**

The Location message encodes the position of the UxV in the World. It was designed to support all kinds of UxVs even when they are not capable of localizing themselves in the World. This message allows the UxV to encode its position in absolute (Latitude, Longitude, and Height) or relative (North/East/Down) coordinates. This message shall be published to the message bus and shall be consumed by any entity that needs to know the location of the UxV.

Field	Units	Description
Latitude	rad	Latitude in the WGS 84 reference coordinate system
Longitude	rad	Longitude in the WGS 84 reference coordinate system
Height	m	Height above the WGS 84 ellipsoid
North	m	The North offset of the North/East/Down field with respect to Latitude/Longitude/Height
East	m	The East offset of the North/East/Down field with respect to Latitude/Longitude/Height
Down	m	The Down offset of the North/East/Down field with respect to Latitude/Longitude/Height
Depth	m	Optional field denoting the vertical distance of the UxV to the water surface

Altitude	m	Optional field denoting the vertical distance of the UxV to the ground

- **Attitude**

Angles describing the attitude of a rigid body (i.e., Euler angles).

Field	Units	Description
Phi	rad	Rotation around the body's longitudinal axis
Theta	rad	Rotation around the body's lateral or transverse axis
Psi	rad	Rotation around the body's vertical axis. A value of 0 means the body is oriented towards true north

- **Linear Velocity**

Vector quantifying the direction and magnitude of the measured linear velocity that a system is exposed to.

Field	Units	Description
X	m/s	X Component
Y	m/s	Y Component
Z	m/s	Z Component

- **Angular Velocity**

Vector quantifying the direction and magnitude of the measured angular velocity that a system is exposed to.

Field	Units	Description
X	rad/s	X Component
Y	rad/s	Y Component
Z	rad/s	Z Component

- **Linear Acceleration**

Vector quantifying the direction and magnitude of the measured linear acceleration that a system is exposed to.

Field	Units	Description
X	m/s/s	X Component
Y	m/s/s	Y Component
Z	m/s/s	Z Component

- **Current**

Measurement of electrical current.

Field	Units	Description
Value	A	Measured electrical current

- **Voltage**

Measurement of electrical voltage.

Field	Units	Description
Value	V	Measured electrical voltage

- **Sensor Reading Scalar**

This message is used to encode scalar measurements of sensors.

Field	Units	Description
Value	-	Measured value
Unit	-	Value's unit

- **Abort**

This command instructs the UxV to stop any executing actions and enter standby mode

- **Goto**

This command instructs a system to move to a given location at a given speed.

Field	Units	Description
Location	-	Desired location (see Location message)
Speed	-	Optional field defining the desired speed
Timeout	s	The amount of time to perform this command

- **KeepStation**

This command instructs a system to keep station at a given location.

Field	Units	Description
Location	-	Desired location (see Location message)
Speed	-	Optional field defining the desired speed
Timeout	s	The amount of time to perform this command
Radius	m	Tolerance radius around the desired location

4.3.8.2 An approach to existing implementations: ROS-based UGV

The ROS meta-OS is implemented on many existing UxV's. Its runtime "graph" is a peer-to-peer network of processes (potentially distributed across machines) and loosely coupled using the ROS communication infrastructure itself. It has indeed different processes running for the UxV control and the operation of sensors such as the publish/subscribe mechanism for sensor readings and robot localisation information. Although many features of the UxV components are available in ROS, the present design shall not depend on ROS as some UxV's may not be equipped with it, and, more importantly this design shall integrate the UxV's into the RAWFIE ecosystem in a way that is agnostic to their underlying operating platform.

As we have explained above, the responsibility of the UxV Node is to provide an interface to make possible the operation of the robot throughout a common message bus, including:

- Process and execute robot steering commands (either waypoints or real-time remote control commands).
- Control the speed of the robot and enforce any safety rule given: no-go areas, minimal or maximal altitude or depth, collision avoidance.
- Estimate and publish the robot odometry and any other localisation and speed information
- Monitor the vehicle critical resources such as the battery. Take safety measures (e.g. return to base) if energy is too low to complete the mission.
- Publish identification information.

- Publish all the data gathered by the robot sensors

The actual development of this interface is based on Kafka, schema-registry and AVRO libraries compatible with the robot software. In this case, several API's have been studied and tested.
In the end, the python compatible libraries have been adopted for convenience.

The list of resources running with the interface is the following:

- AVRO 1.7.7
- KAFKA 2.10-0.8.2.0
- KAFKA-PYTHON API
- PYTHON-CONFLUENT-SCHEMA REGISTRY

The architecture of this ROS-KAFKA-AVRO interface remains very simple.
The automatic execution of python-coded nodes is subscribing the peer-to-peer data within ROS framework, and then reformatting it into the AVRO common format. This schemed data is sent through the KAFKA message bus with a simple kafka-publisher.
This way of operating stands for every sensor data and also localization, odometry etc.
On the other hand, a kafka-subscriber is set up to receive the agreed commands. This data is then converted within the interface to a ros-understandable information, either a service or a control topic.

4.3.8.3 An approach to existing implementations: USV implementation

OceanScan-MST software infrastructure is based on the "LSTS Toolchain" (http://www.lsts.pt/toolchain) developed by the University of Porto. This toolchain comprises three software components: DUNE: on-board software with modules for control, navigation, simulation, networking, sensing, and actuation; Neptus: distributed command and control infrastructure supporting the different phases of a typical mission life cycle: interactive planning, simulation, execution, and post-mission data analysis; IMC: message set used by DUNE and Neptus to interchange commands and data. All software modules are open-source and the source

code is published in the Internet under the European Union Public Licence V.1.1. With these open-source software tools the user can reuse the existing code to access all data, and variables provided by the vehicle's hardware at the maximum sampling rate. These can be raw sensor readings, data processed by controllers or messages exchanged between components or even other systems (other UxVs or any other system that uses the IMC message set). New software routines can be easily created and interfaced with the original framework, or even replace original routines.
For the purpose of the RAWFIE project OceanScan-MST developed OIRT (OceanScan-MST IMC/RAWFIE Translator) a standalone software module written in Java to bidirectionally translate IMC messages and related logic to messages in the Apache Avro data serialization format used in the RAWFIE project. OIRT module interacts with IMC capable UxVs and the Kafka Message Bus and is responsible for:

- Discovering UxVs on the local network via broadcast and multicast announcements
- Establishing and managing a stable network connection to each discovered UxV
- Receiving and decoding IMC messages from UxVs
- Gathering the required data to produce RAWFIE UxV Protocol messages from information conveyed by IMC messages.
- Publishing RAWFIE UxV Protocol messages to the Kafka Message Bus
- Subscribing to RAWFIE UxV Protocol messages on the Kafka Message Bus
- Decoding RAWFIE UxV Protocol messages.
- Gathering the required data to produce IMC messages from information conveyed by RAWFIE UxV Protocol messages
- Encoding and sending IMC messages to UxVs
- Guaranteeing that commands received via the RAWFIE Message Bus are valid and can be safely executed

The OIRT software module acts as a proxy between the RAWFIE infrastructure and the UxVs or corresponding simulated instance. This approach was chosen for three reasons:

- The on-board software of the UxVs does not need to be modified in any way to support integration in the RAWFIE infrastructure
- The basic computational system of the OceanScan-MST UxVs is optimized for low power consumption and therefore cannot run a Java Virtual Machine satisfactorily. It is easier to develop this kind of software in a general purpose desktop computer than on an embedded system.
- If the system is well designed it can later run on board each UxV if a auxiliary compute is installed (as is envisioned).

4.3.9 AVRO formatted messages and Kafka Schema Registry

Next we will introduce some of the main tools making part of the custom solution to create a common framework for every robot to adhere to RAWFIE agnostically of their system through the Adaptor of the UxV Node.

4.3.9.1 AVRO

According to its own documentation, the apache avro tool is a data serialization system with some useful capabilities.

https://avro.apache.org/docs/current/
Apache Avro™ is a data serialization system.
Avro provides:

- Rich data structures.
- A compact, fast, binary data format.
- A container file, to store persistent data.
- Remote procedure call (RPC).
- Simple integration with dynamic languages. Code generation is not required to read or write data files nor to use or implement RPC protocols. Code generation as an optional optimization, only worth implementing for statically typed languages.

Schemas
Avro relies on schemas. When Avro data is read, the schema used when writing it is always present. This permits each datum to be written with no per-value overheads, making serialization both fast and small. This also facilitates use with dynamic, scripting languages, since data, together with its schema, is fully self-describing.
When Avro data is stored in a file, its schema is stored with it, so that files may be processed later by any program. If the program reading the data expects a different schema this can be easily resolved, since both schemas are present.
When Avro is used in RPC, the client and server exchange schemas in the connection handshake. (This can be optimized so that, for most calls, no schemas are actually transmitted.) Since both client and server both have the other's full schema, correspondence between same named fields, missing fields, extra fields, etc. can all be easily resolved.
Avro schemas are defined with JSON . This facilitates implementation in languages that already have JSON libraries.

4.3.9.2 Kafka Schema Registry

Schema Registry provides a serving layer for metadata.

- It provides interface for storing and retrieving Avro schemas.
- It stores a versioned history of all schemas, provides multiple compatibility settings and allows evolution of schemas according to the configured compatibility setting.
- It provides serializers that plug into Kafka clients that handle schema storage and retrieval for Kafka messages that are sent in the Avro format.

In the end this Schema Registry is heavily based on the Java API of Confluent Schema Registry. See http://docs.confluent.io/2.0.0/schema-registry/docs/index.html for more info.
Kafka Clients
The main tool of the adaptors will be relying on a number of Kafka Clients.
According to Apache Org. we can find several clients that support different programming languages such as Python, C++, Ruby etc. A complete list and useful documentation can be found at https://cwiki.apache.org/confluence/display/KAFKA/Clients#Clients-Python
On the other hand, the common approach will be to use the Java main code base.
One way or the other, the adaptor activity described in a high level, will consist of publishing and consuming specific avro-formatted messages. A helpful reminder of these concepts:

- Kafka maintains feeds of messages in categories called topics.
- We'll call processes that publish messages to a Kafka topic producers.
- We'll call processes that subscribe to topics and process the feed of published messages consumers.
- Kafka is run as a cluster comprised of one or more servers each of which is called a broker.

4.3.10 Common Sensors for UxVs

In the past sections we have described the adaptor between RAWFIE system and the robots.
Once the abstraction layer required by the project has been obtained, we will also point out some of the most common specifications in the UxV's frame.
A great number of sensors can be deployed within the robot with several different purposes. In this section, we describe some of the most common, which may be present in every robot adhering to the RAWFIE platform. This information also provides the needed framework to understand what kind of common messages are to be exchanged between the UxV's and the RAWFIE components.

4.3.10.1 UGV common sensors

- 2D range finders
  - This range finders such as laser scanners provide accurate high resolution data necessary to achieve good results from SLAM algorithms. Most 2D range finders feature a wide field of view and a wide sensing range useful for mapping and obstacle avoidance.
- 3D Sensors
  - Once of the most important recent developments in robotics sensors is the production of low cost 3D range finders and RGB-Depth cameras. These sensors open up a whole new dimension for robotics due to the huge number of their application.
- Pose Estimation Sensors
  - A pose estimation sensor may provide information about the absolute or relative position and orientation of a robot. These sensors include gyroscopes, magnetometers, accelerometers, satellite navigation systems, which are often packaged together as a single hardware device with an internal state estimator providing a fused output.
- Cameras
  - Cameras provide image data to the robot that can be used for object identification, tracking and manipulation tasks. A number of different cameras with different specifications is available in the market, including monocular, stereo cameras and those that include a pan-tilt-zoom mechanism (PTZ cameras). We should also mention here the IP cameras (internet protocol cameras) that offer a ease of connectivity.

4.3.10.2 USV sensors

- Acoustic Modem
- Side-Scan Sonar
- Single & Multi-beam Echo-sounder
- Forward Looking Sonar
- Digital Camera
- CTD sensor
- Water Quality Sensors
- Turbidity Sensor
- Scattering Sensor
- Fluorescence Sensor
- Doppler Velocity Log (DVL)
    - For underwater vehicles, the bottom tracking feature can be used as an important component in the navigation systems. In this case the velocity of the vehicle is combined with an initial position fix, compass or gyro heading, and data from the acceleration sensor. The sensor suite is combined (typically by use of a Kalman filter) to estimate the position of the vehicle.

- Sound Velocity Sensor (SVS)
- High Precision INS
- Long BaseLine (LBL)
    - Long baseline systems determine the position of a vehicle or diver by acoustically measuring the distance from a vehicle or diver interrogator to three or more seafloor deployed baseline transponders. These range measurements, which are often supplemented by depth data from pressure sensors on the devices, are then used to triangulate the position of the vehicle or diver.

5 Global Sequence diagrams showing main RAWFIE processes

5.1 Registration of Testbed Resources

The sequence diagram in Figure 79 shows a sample information flow with the components involved in registration of a new resource in a specific, already registered, Testbed. As this is a user process, the Testbed Administrator actor is involved:

1. The Testbed Administrator opens the Login page of the RAWFIE Web Portal (centralised point for accessing the RAWFIE system features), and inserts the credentials
2. The user authentication process starts, involving the Users & Rights Service which, in turn, make use of the Users & Rights Repository to check user's credentials
3. Once the Testbed Administrator gets access to the platform, he/she may ask for the list of available Testbeds, for selecting the Testbed the new Resource (UxV) has to be assigned to. The Resource Explorer Tool is involved in this process, which is actually part of the RAWFIE platform Web GUI
4. The Resource Explorer tool then, uses the Users & Rights Service and Repository, to check if the user role has the capabilities to access such information
5. If yes, the actual requests for the Testbeds list, is forwarded to the Testbed Directory Service, which will return back the list of available Testbeds, by querying the Master Data Repository
6. The Testbed Administrator requests the assignment of the new resource to a specific Testbed
7. Again, the role of the user is checked against the Users & Rights Repository, to ensure the user has the capabilities to perform this operation
8. If yes, the actual resource assignment request is forwarded by the Resource Explorer Tool to the Testbed Directory Service, and the new resource information is stored in the Master Data Repository

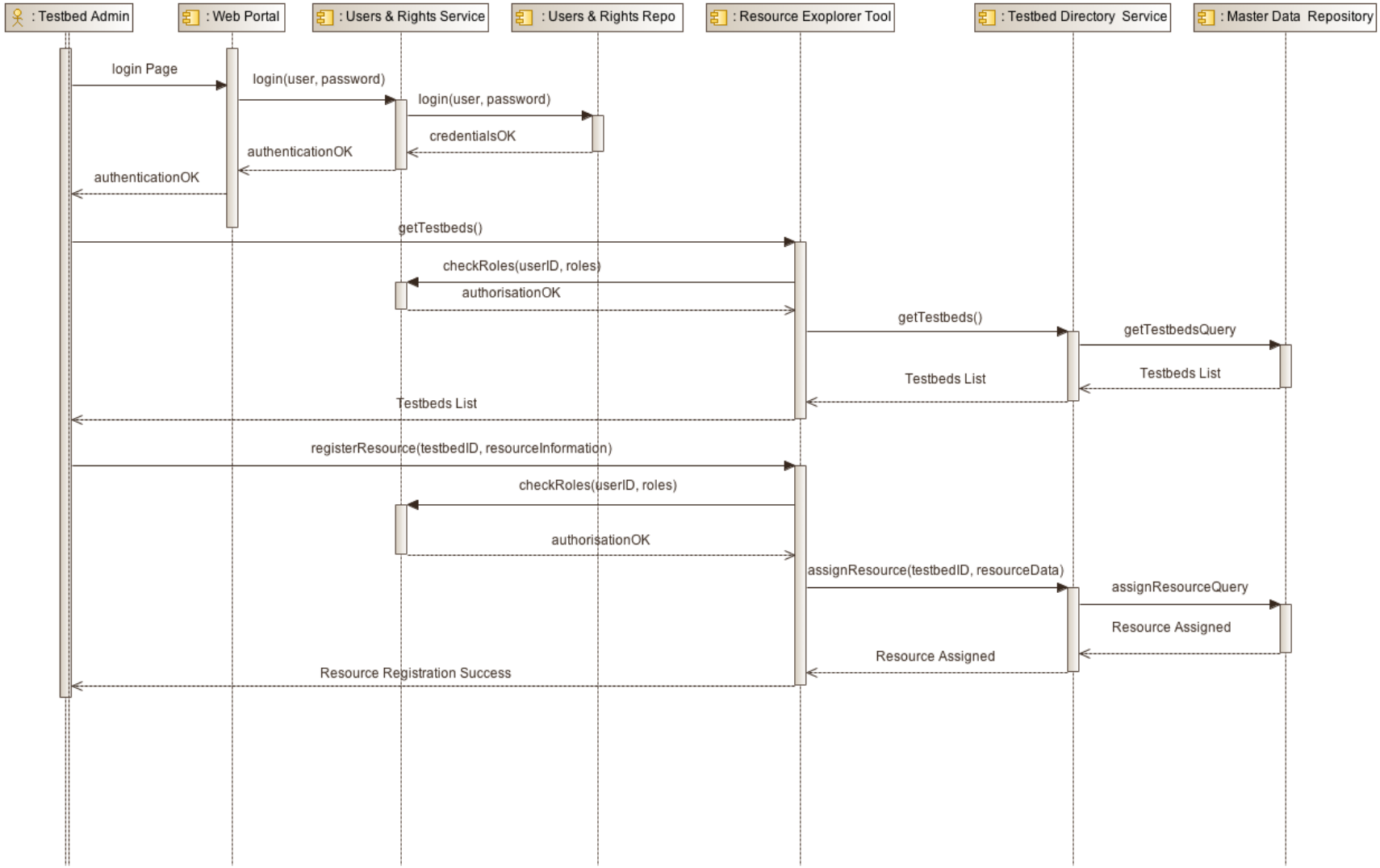

Figure 79: Sequence Diagram for "Registration of Testbed Resources" process

5.2 Booking Testbed Resources (HAI)

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. After performing the necessary authorisation checks Booking Service performs its internal actions and if successful, the booking request is persisted in the database with status PENDING
6. Appropriate email notifications are sent to both the experimenter initiating the booking as well as to the testbed operator responsible for approving it
7. Following the reception of booking notification an authorised Testbed Operator loads the ApproveBooking page showing all pending booking requests
8. The Testbed Operator calls Booking Service approveBooking() method which after checking for the proper authorisation performs all the internal logic required for confirming the experimenter's request
9. If no conflict or any other problem occurs, the booking request is CONFIRMED
10. Following the confirmation a BookingStatusMsg is sent to the Message Bus informing any interesting consumer component, that the booking request has changed status
11. Both Experimenter and Testbed Operator are informed by appropriate email notifications

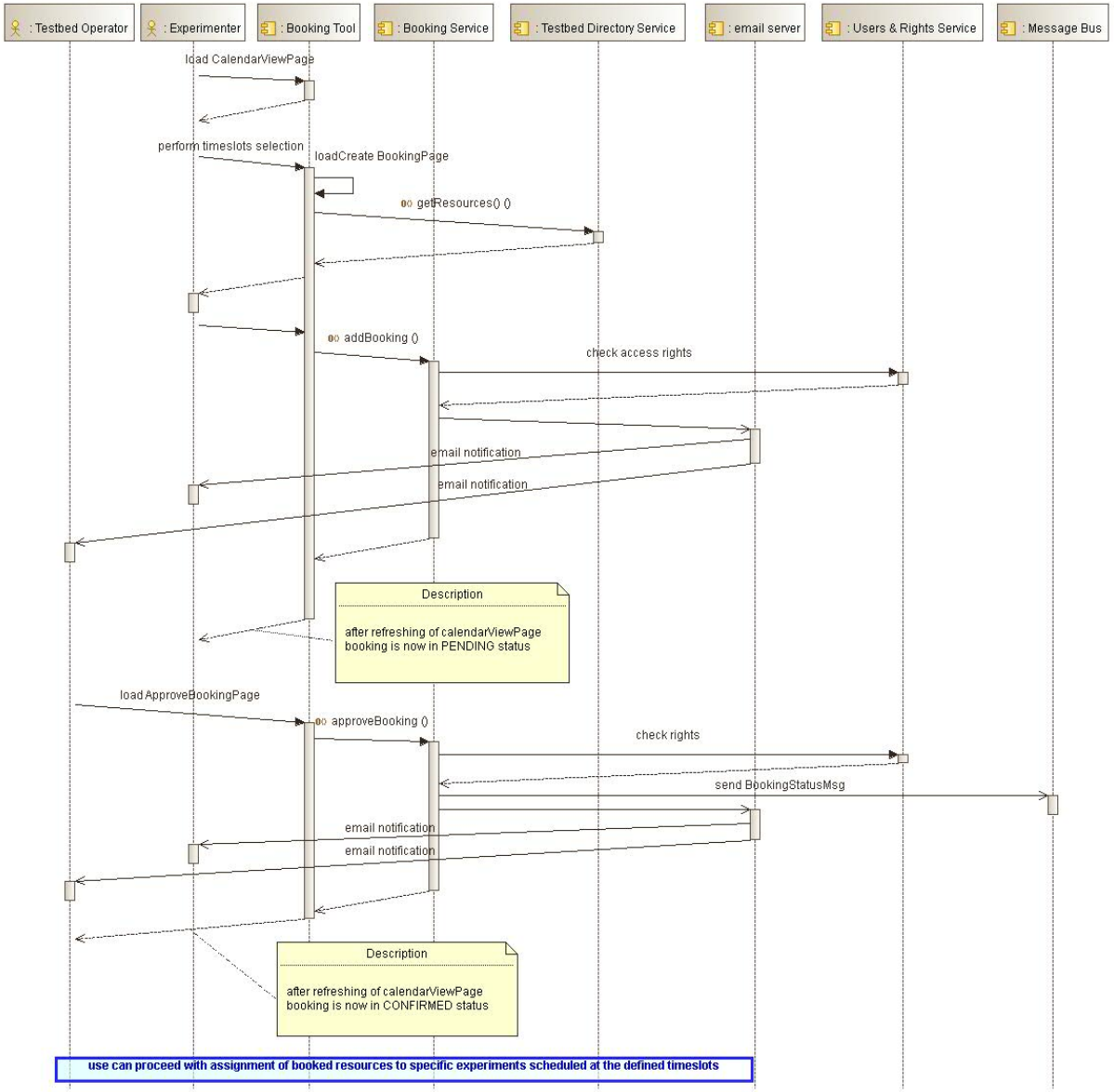

Figure 80: Sequence Diagram for "Booking Resource" process

5.3 System Monitoring

3. A Monitoring Manager of a Testbed publishes health status information about the Testbed and UxVs to the related Message Bus topic
4. SystemMonitoringManager reads the status messages from the Message Bus and stores the latest values per component internally
5. Internal timer of Icinga starts the monitoring procedure
6. A standard Icinga checker is executed to check a server (e.g. ping, CPU load, RAM usage, HTTP alive)
7. Icinga calls JNRPE Server with the check_rawfie plugin
8. JNRPE Server requests the latest health status received via message bus from the System Monitoring Service (getComponentServiceHealth_internal())
9. JNRPE Server returns the result
10. Icinage stores the results
11. Testbed Admininistrator requests the status dashboard from the System Monitoring Tool
12. The System Monitoring Tool calls getComponentServiceHealths() from the SystemMonitoringManager
13. The SystemMonitoringManager requests from Icinga the health statuses via the MK-Livestatus interface.
14. SystemMonitoringManager returns the health statuses
15. System Monitoring Tool renders the status page and returns it to the user

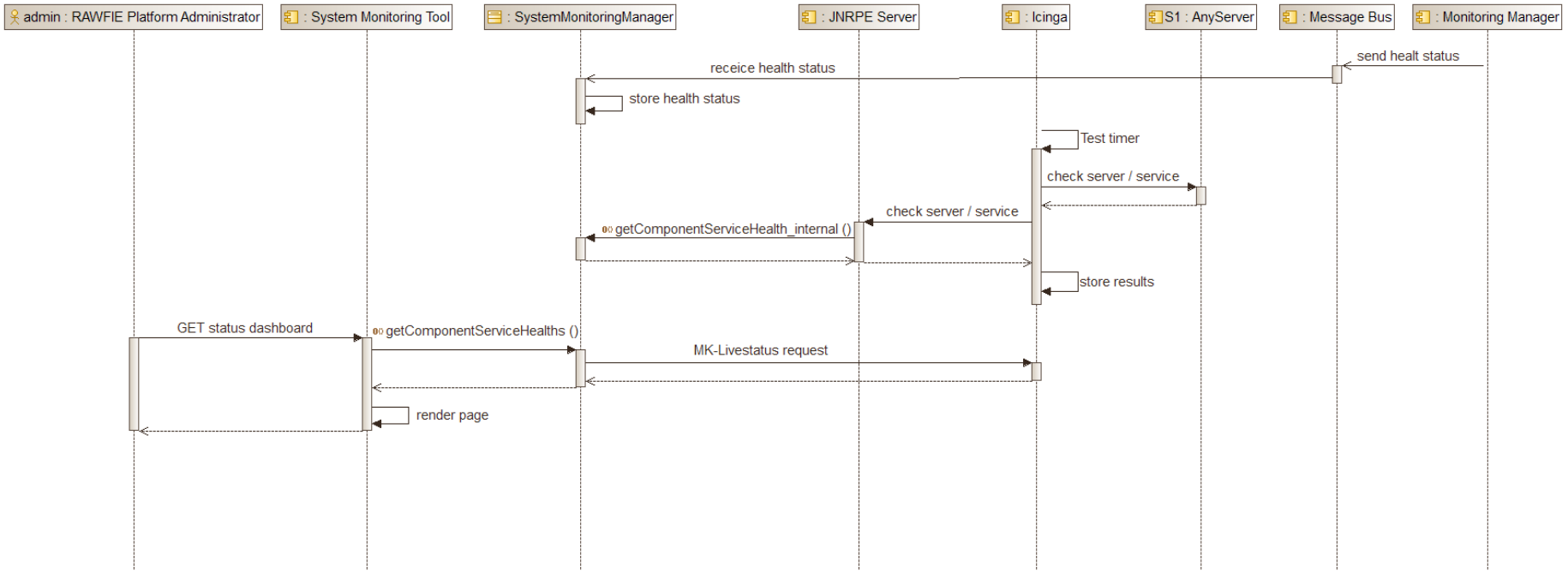

Figure 81: Sequence Diagram for “System Monitoring” process

5.4 Experiment Execution and Monitoring

The sequence diagram in Figure 82 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 has the right to observe the experiment and do modifications
2. The Experimenter starts the Experiment Monitoring Tool in order to view the available experiments, then selects an already running experiment from the list
3. The Experimenter enters a new position for a specific UxV. This command is sent through different modules to the Resource Controller, which converts it to a waypoint, evaluates the path and sends it to the UxV
4. The UxV starts executing the command
5. The Experimenter opens the visualisation page
6. 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
7. 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
8. 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
9. 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

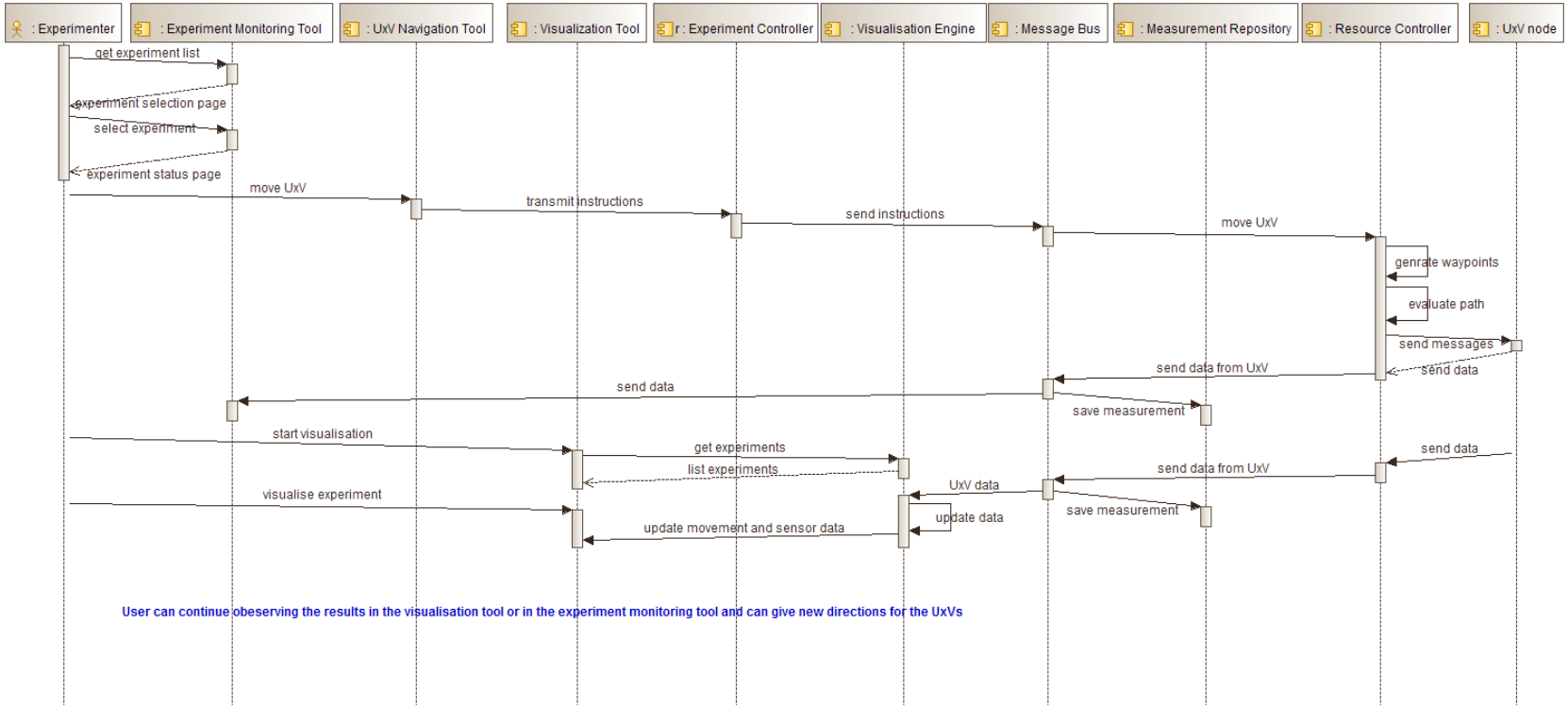

Figure 82: Sequence Diagram for "Experiment Execution and Monitoring" process

5.5 Experiment Measurements Recording

The sequence diagram in Figure 83 shows a sample information 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:

9. 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
10. The instructions are then consumed, from the same Message Bus topics, by the Resource Controller on the Testbed side
11. 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
12. 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), and ensure UxVs safety
13. Together with the position updates, the UxVs also publishes all other expected sensors' measurements
14. Sensors' measurements are continuously consumed, besides the Resource Controller, by the Measurements Backend Service component, which is in charge of ensuring the persistence of the same data within the Measurements Repository, a NoSQL data storage
15. At runtime, other RAWFIE components such as the Visualisation Engine may directly access the Measurements Repository data, to perform operations on the collected historical information

It is important to notice that currently the Measurements Backend Service is still a conceptual component. It needs to be practically defined and actually, the recording of the measurements in the NoSQL data storage may be executed by:

a. The Experiment Controller itself, that always gets access to the acquired data
b. A dedicated Backend Service component, in charge of listening for new available measurements to store them in the NoSQL data storage
c. A connector implemented using the Apache Kafka Connect component, available with the Confluent platform

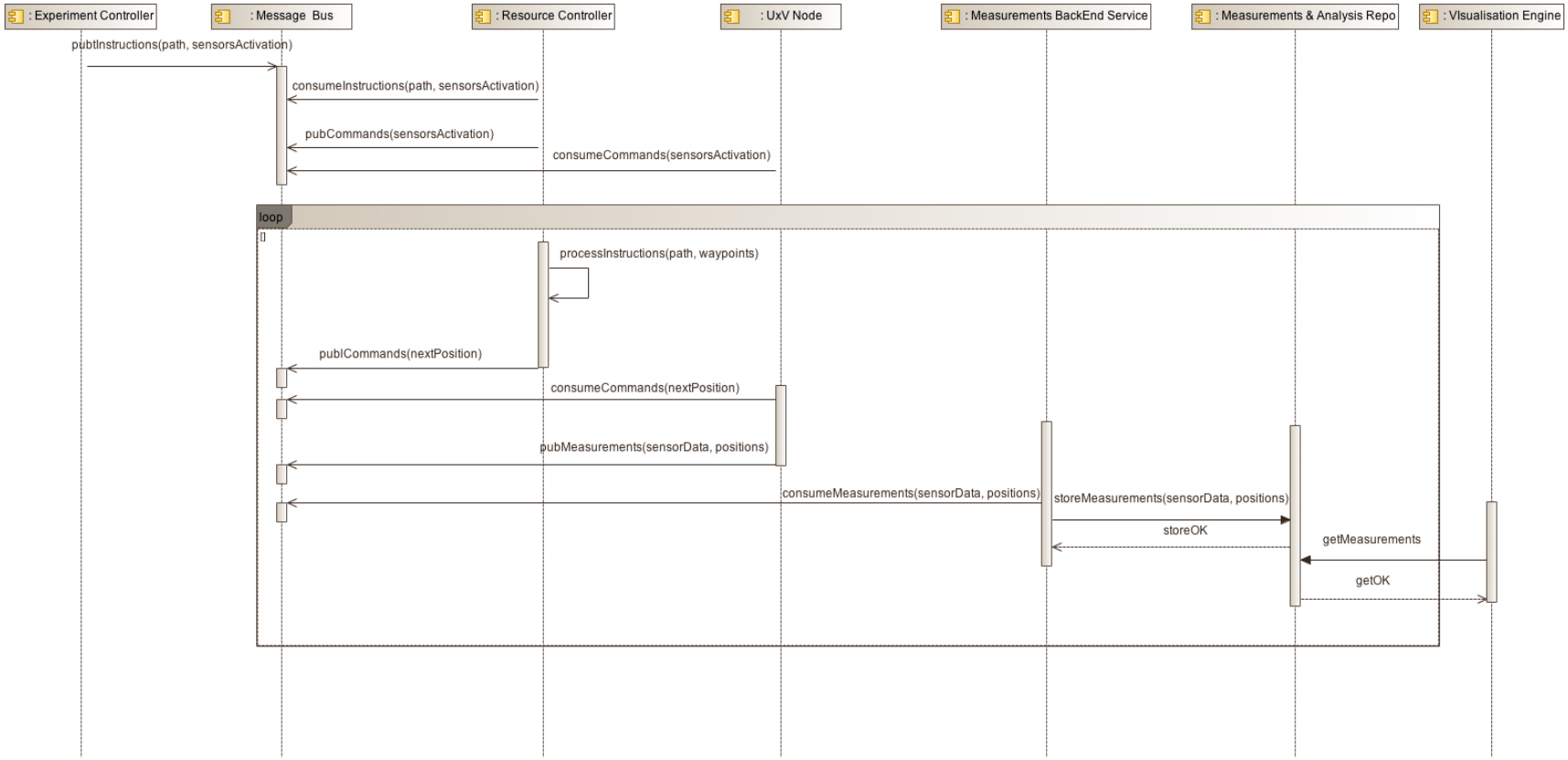

Figure 83: Sequence Diagram for “Experiment Measurements Recording” process

5.6 Authoring and Launching of an Experiment

The sequence diagram in Figure 84 shows a sample flow for authoring and launching an experiment. The adopted steps are as follows:

1. The Experimenter books a testbed where the experiment will be executed.
2. The Experiment gains access to the Authoring tool.
3. The Experiment defines the experiment and gives commands to the tool.
4. The Authoring tool performs a continuous validation process by communicating with the Compiler and Validation Tool.
5. The Compiler and Validation service communicates with the core validation service and returns the results to the Authoring tool.
6. The Authoring returns the retrieved messages to the Experimenter.
7. The Experimenter, after the definition of the experiment, produces the required files to be adopted by the remaining parts of the architecture.
8. The Authoring tool invokes the Compiler and Validation Tool and, accordingly, the required files are stored to the data repository.
9. Finally, the Experimenter selects to launch an experiment by choosing its ID.
10. The Launching tool sends the required message to the Experiment Controller that undertakes the responsibility to control the execution process of the experiment.
11. 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.
12. A continuous communication between UxV nodes and Resource / Experiment Controller is held until the end of the execution

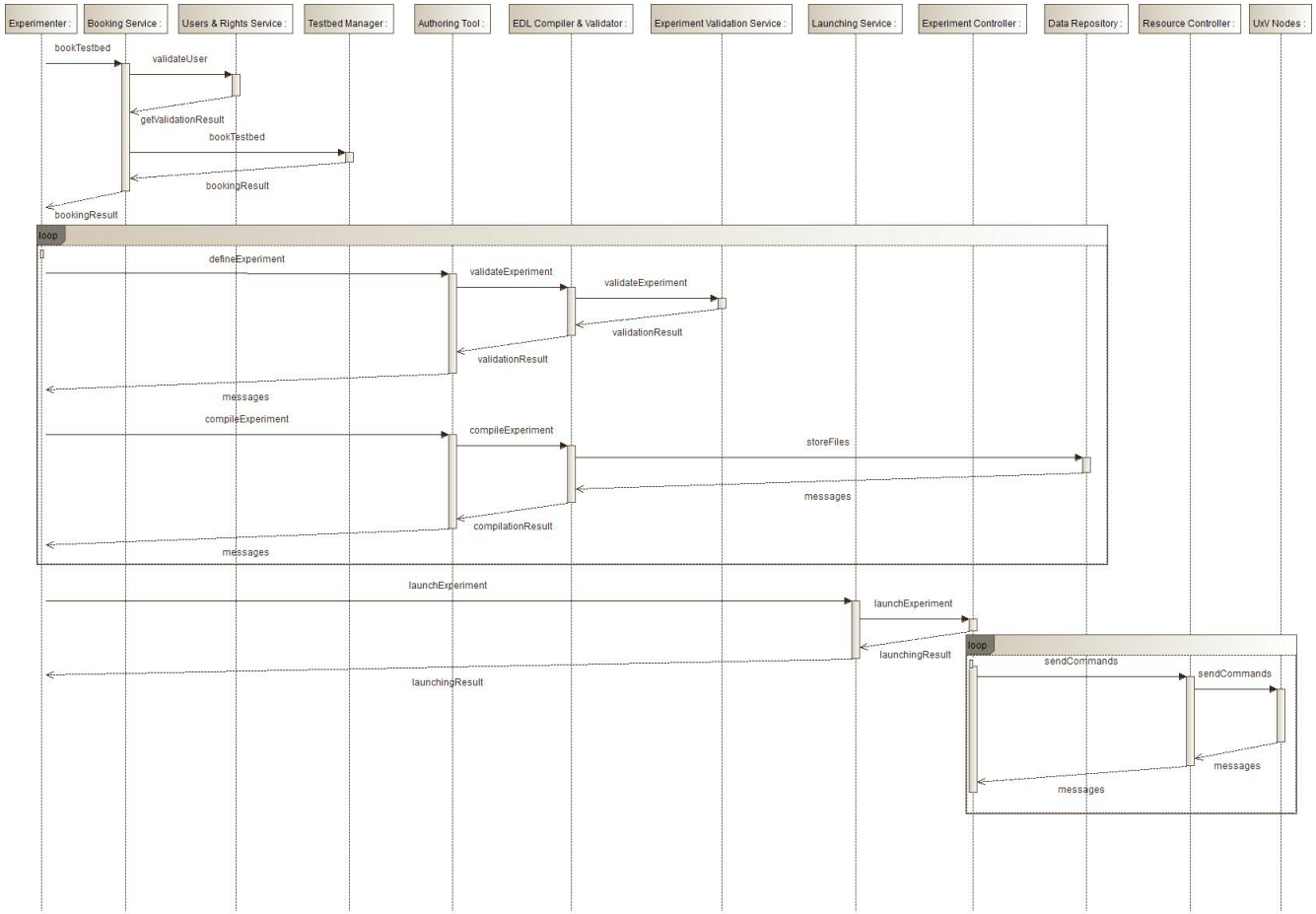

Figure 84: Sequence Diagram for "Authoring and Launching of an Experiment" process

5.7 Data Analysis

The sequence diagrams in the following pictures illustrates 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 a fixed non-time-dependant 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. However, batch tasks are performed on data coming from the Measurement Repository whereas streaming tasks are performed on real time data coming directly from the drones through 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).

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

1. The user defines the data analysis task to run through the data Analysis Tool. He specifies the schemas associated with the data he wants to run computations on via simple Message Bus query (schema registry more precisely)
2. The Tool relays the task creation order to the Data Analysis Engine which initiate the task.
3. The Engine computes sequentially on the most recently added data on the stream that is retrieved from the Message Bus.
4. In the absence of computation error, the computation is interrupted if and only if the user sends a kill signal from the Tool.
5. The results can be visualised through the Tool which integrates the dashboard associated with the Analysis Results Repository. Such a visualisation can be done 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 other the task would have to be interrupted to see the results).

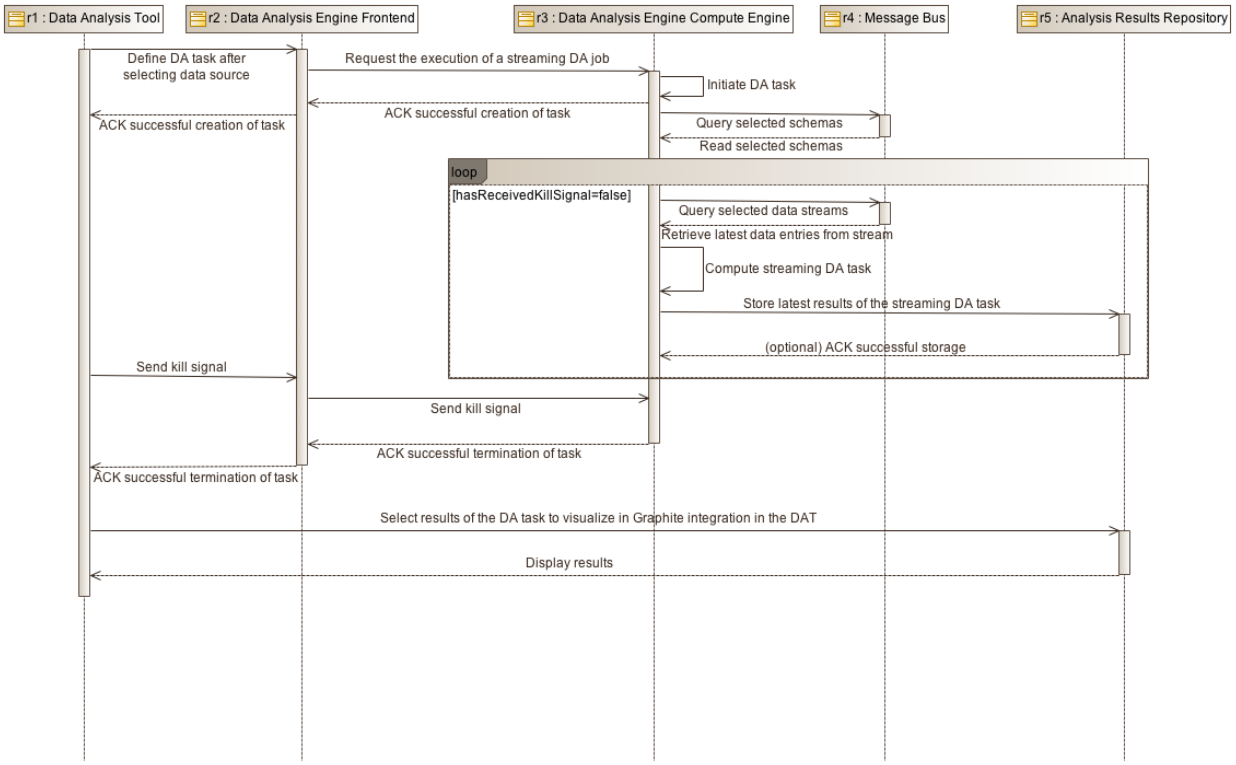

Figure 85: Sequence Diagram for the "Data Analysis in a streaming scenario" process

As for batch task (below), the synopsis differs in the following aspects:

1. Data is from the Measurement Repository, not from the Message Bus.
2. The task ends, it is not tied to an infinite stream of data.

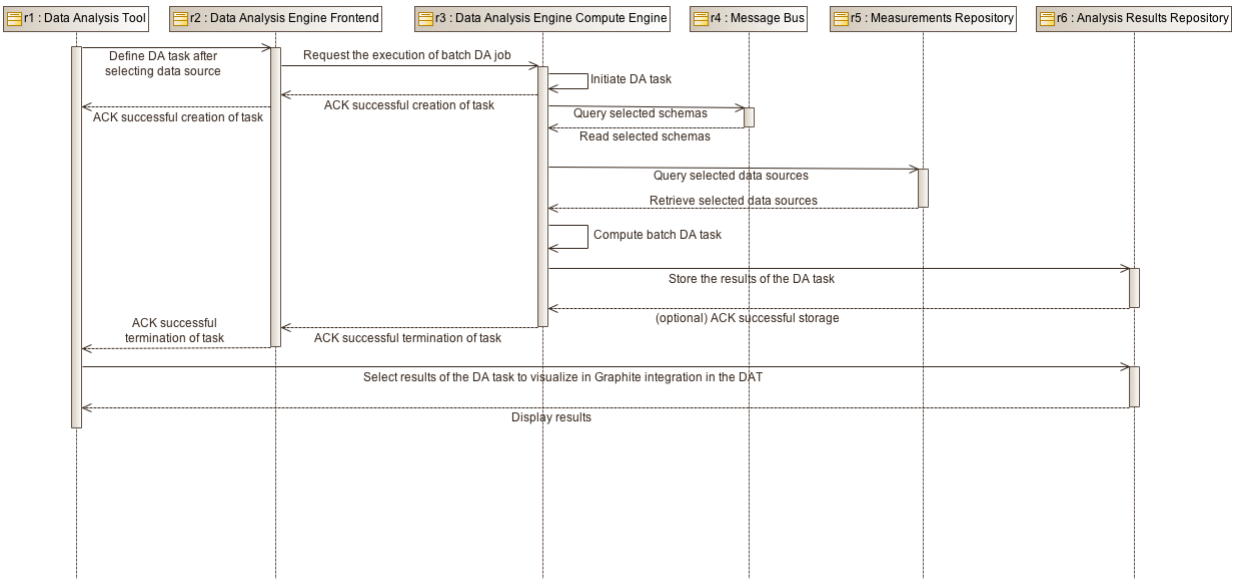

Figure 86: Sequence Diagram for the "Data Analysis in a batch scenario" process

6 Summary and Outlook

The design proposed into this document provides concepts that may be adopted for the physical deployment of the RAWFIE platform from a physical, cloud oriented standpoint (physical infrastructure), in Section 3. It also provides instructions on how the several software components needs be deployed within different servers, together with the base technologies (Java, Tomcat, etc.) used for the software execution environments, in the 2nd version of the prototype (Section 3 and 4).

The mapping of the requirements identified in D3.2, 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 as the starting point for the 2nd implementation cycle, whereas the information flows for some of the most relevant use cases highlighted in Section 5, will be the basis for the new comprehensive tests on components functionalities and interfaces.

7 References

[1] http://geoserver.org/
[2] http://docs.confluent.io/3.0.0/
[3] http://kafka.apache.org/
[4] https://en.wikipedia.org/wiki/NoSQL
[5] tomcat.apache.org
[6] https://forgerock.org/opendj/
[7] http://www.icinga.org/
[8] http://www.jnrpe.it/
[9] http://www.nagios.org/
[10] http://mathias-kettner.com/checkmk_livestatus.html
[11] G. Cugola and M. Migliavacca, "A Context and Content-Based Routing Protocol for Mobile Sensor Networks", Proc. European Conf. Wireless Sensor Networks, pp. 69-85, 2009.

8 Annex

8.1 Abbreviations

Abbreviation	Meaning
3D	three-dimensional space
ACL	Access Control List
AGL	Above Ground Level
AHRS	Attitude and Heading Reference System
AJAX	Asynchronous JavaScript and XML
AM	Aggregate Manager (of SFA)
AP	Access Point
API	Application Programming Interface
API	Application programming interface
AT	Aerial Testbed
AUV	Autonomous underwater vehicle
B-VLOS	Beyond Visual Line Of Sight
CA	Certification Authority
CAA	Civil Aviation Authority
CAO	Cognitive Adaptive Optimization
CBNR	Chemical Biological Nuclear Radiological
CEP	Circular Error Probability
CPU	Central Processing Unit
CSR	Certificate Signing Request
DETEC	Department of the Environment, Transport, Energy and Communication
DGCA	Directorate General of Civil Aviation
DoA	Description of Actions
EASA	European Aviation Safety Agency
EC	Experiment Controller
ECC	Error Correction Code
ECV	EDL Compiler & Validator
EDL	Experiment Description Language

EDL	Experiment Description Language
EER	Experiment and EDL Repository
EU	European Union
E-VLOS	Extended Visual Line Of Sight
EVS	Experiment Validation Service
FIRE	Future Internet Research & Experimentation
FOCA	Federal Office of Civil Aviation
FPS	Frames Per Second
FPV	First Person View
GAA	German Aviation Act
GIS	Geographic Information System
GNSS	Global Navigation Satellite System
GPIO	General Purpose Input/Output
GPS	Global Positioning System
GUI	Graphical user interface
HD	High Definition
HTTP	Hypertext Transfer Protocol
HW	Hardware
IAA	Irish Aviation Authority
IaaS	Infrastructure as a Service
IDE	Integrated Development Environment
IDE	integrated development environment
IFR	Instrument Flight Rules
IP	Internet Protocol
ISO	International Standards Organization
JDBC	Java Database Connectivity
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
KPI	Key Performance Indicator
LBL	Long Baseline
LDAP	Lightweight Directory Access Protocol
LS	Launching Service
MEMS	MicroElectroMechanical System
MM	Monitoring Manager
MSO	Multi Swarm Optimization
MT	Maritime Testbed
MOM	Message Oriented Middleware
MVC	Model View Controller
NAT	Network Address Translation
NC	Network Controller
NF	Non Functional
ODBC	Open Database Connectivity
OEDL	OMF EDL
OMF	cOntrol and Management Framework
OMF	Orbit Management Framework
OML	ORBIT Measurement Library
OS	Operating System
OTA	Over The Air
P2P	Point to Point

PSO	Particle Swarm Optimization
PTZ	Pan Tilt Zoom
RC	Resource Controller
RC	Resource Controller
RE	Requirement Engineering
REST	Representational state transfer
RIA	Research and Innovation Action
ROS	Robot Operating System
ROV	Remotely Operated Vehicle
RPA	Remotely Piloted Aircraft
RPAS	Remotely Piloted Aircraft System
RPS	Remotely Piloted Station
RSpec	SFA Resource Specification
SaaS	Software as a Service
SAML	Security Assertion Markup Language
SFA	Slice-based Federation Architecture
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SQL	Simple Query Language
SSO	Single-Sign-On
SVN	Apache Subversion
TM	Testbed Manager
TMS	Testbed Manager Suite
TP	Testbed Proxy
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
UI	User Interface
UML	Unified Modelling Language
USV	Unmanned Surface Vehicle
UUV	Unmanned Underwater Vehicle
UxV	Unmanned aerial/ground/surface/underwater Vehicle
VE	Visualization Engine
VT	Vehicular Testbed
VT	Visualization Tool
WCS	Web Coverage Service
WFS	Web Feature Service
WMS	Web Map Service
WPS	Web Processing Service
WSDL	Web Services Description Language
XMPP	Extensible Messaging and Presence Protocol

8.2 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.