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	D4.2 (a) - Design and Specification of RAWFIE Components		
Confidentiality	PU	Deliverable type[1]	R
Deliverable File	D4.2	Date	31.07.2015
Approval Status[2]	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
Ezequiel Primo	IES	e.primo@iessolutions.eu
Marcel Heckel	Fraunhofer	marcel.heckel@ivi.fraunhofer.de
Kostas Kolomvatsos	UoA	kostasks@di.uoa.gr
Vasil Kumanov	Epsilon	vasil.kumanov@epsilon-bulgaria.com
Blerina Lika	UoA	b.lika@di.uoa.gr
Philippe Dallemagne	CSEM	philippe.dallemagne@csem.ch
Damien Piguet	CSEM	damien.piguet@csem.ch
Kakia Panagidi	UoA	kakiap@di.uoa.gr
Savvas Chatzichristofis	CERTH	schatzic@iti.gr
Lionel Blondé	HES-SO	lionel.blonde@hesge.ch
Jason Ramapuram	HES-SO	jason.Ramapuram@etu.unige.ch
Miquel Cantero	Robotnik	mcantero@robotnik.es
Ricardo Martins	MST	rasm@oceanscan-mst.com

REVIEWERS TABLE

Name	Company	E-Mail
Nikolaos Priggouris	HAI	priggouris.nikolaos@haicorp.com
Kiriakos Georgouleas	HAI	Georgouleas.Kiriakos@haicorp.com
Sotiris Glykofrydis	HMOD	sglikofridis@dideap.mil.gr

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	2015-06-05	TOC / Initial version	all
0.2	2015-06-15	Discussion about content structure	all
0.3	2015-06-29	Conventions definition and status of contributions	all
0.4	2015-07-16	Finalisation of first round of contributions	all
0.5	2015-07-17	Review started	all
0.6	2015-07-24	Reviewed version, ready for refinements	all
1.0	2015-07-31	Final version	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:

This deliverable is based on the results of the following tasks: T4.2, T4.3 and T4.4.

The report presents the design documentation of RAWFIE subsystem and belonging components. It contains the architectural definition and specification of the components as well as the different interactions and responsibilities among them, highlighting their interfaces and communication flow.

Keywords:
design, architecture, module, component, subcomponent, responsibilities, interactions, relationships, interfaces, diagrams, functionalities, operations, attributes, methods, scenarios, data flows, message broker, repository, web service.

Part II: Table of Contents-

List of Figures

List of Tables

Part III: Executive Summary

The present document is the first in a series of three documents about RAWFIE platform components design and specification, which will be delivered at the beginning of each RAWFIE development iteration cycle.

D4.2 consists of a design specification of all components at the different application tiers. The main purpose is to observe in depth the data flow and interaction between the components specified in the previous deliverable D4.1. UML is used as design and modelling approach.

Two different use case scenarios were selected from D3.1, with the aim to demonstrate the relationships between components, and through overall sequence diagrams, diverse experimental cycles are described.

Further to the tiers based classification of components as proposed in deliverable D4.1, the current document provides a new classification, specifically focused on the operational goals of the different components. This component subdivision represents a high level categorisation, therefore the goal is to analyse the role of each component from a behavioural and functional perspective. In addition, for each class of components, a high level components diagram is provided, highlighting the communication with any of the others platform components, and with the respective data repositories.

Furthermore, for each individual component, the most relevant attributes and operations are exposed by means of class and sequence UML diagrams.

Part IV: Main Section

1 Introduction

1.1 Scope of D4.2

This deliverable specifies the design of each one of the components involved in the RAWFIE platform architecture.

Practically it answers:

- Overall understanding of the global architecture
- Components interactions and responsibilities
- Functionalities and operations

1.2 Relation to other deliverables

A detailed requirement analysis was carried out in D3.1. Based on these requirements specification and the planned functionalities extracted from the DoW, this architectural document was created. The stakeholders related to the diagrams are described in detail within D3.1.

D4.2 is actually expected to provide a deeper architecture analysis and further detailed components descriptions. Therefore, the main intention of this deliverable is merely to design conceptual models for the components and illustrate their interfaces, operations and responsibilities.

1.3 Abbreviations

Abbreviation	Meaning
API	Application Programming Interface
AUV	Autonomous underwater vehicle
CA	Certificate Authority
CAO	Cognitive Adaptive Optimization
CSR	Certificate Signing Request
DoW	Description of Work
EC	Experiment Controller
ECV	EDL Compiler & Validator
EDL	Experiment Description Language
EER	Experiment and EDL Repository
EVS	Experiment Validation Service
GIS	Geographic Information System
GPS	Global Positioning System
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment

JSON	JavaScript Object Notation
KPI	Key Performance Indicator
LDAP	Lightweight Directory Access Protocol
LS	Launching Service
MVC	Model View Controller
NC	Network Controller
RC	Resource Controller
REST	Representational State Transfer
ROS	Robot Operating System
SOA	Service Oriented Architecture
TM	Testbed Manager
UML	Unified Modelling Language
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
USV	Unmanned Surface Vehicle
UUV	Unmanned Underwater Vehicle
UxV	Unmanned Vehicle
VE	Visualization Engine
VT	Visualization Tool
WCS	Web Coverage Service
WFS	Web Feature Service
WMS	Web Map Service
WPS	Web Processing Service

Table 1: Common abbreviations

2 Global Architecture

This particular section describes the overall design of the global RAWFIE architecture from a high level standpoint. Before describing the individual and conceptual design of each one of the components, it is important to discover, analyse and establish the interactions between main components, in order to achieve an efficient working process of all the use cases outlined so far.

2.1 Inputs from other deliverables

In order to provide a more detailed specification and refinement of the RAWFIE platform components, several inputs from previous deliverables are taken into account. These are:

- The conceptual architectural components defined in the document D4.1 “High Level Design and Specification of RAWFIE Architecture” together with corresponding use cases and sequence diagrams that are expected to be refined and extended in more details in the present deliverable.

- Platform and Testbed functional requirements extracted from the document D3.1 “Specification and Analysis of RAWFIE Components Requirements” as a result of the identification, and requirements elicitation process.

- Certain non-functional requirements considered important for the purpose of designing the respective interfaces and architectural artefacts during system implementation and prototype development.

- Proposed implementation scenarios that, based on their completeness and relevance, will form the basis for the definition-elaboration of certain interactions. In this way the communication among the diverse components of the platform can be represented and the overall system functionality can be clearly depicted.

2.2 High level architecture and interactions

Having D4.1 as a reference, we present a new version of the global architecture. To achieve this, the interactions between the conceptual components defined so far are highlighted, while a further breakdown of all conceptual components in subcomponents remarking their operations, responsibilities and interactions will take place along the following sections.

Below an interaction diagram describing all feasible communications among the multiple platform components is presented. The diagram provides also a classification of components according to the suggested RAWFIE architecture.

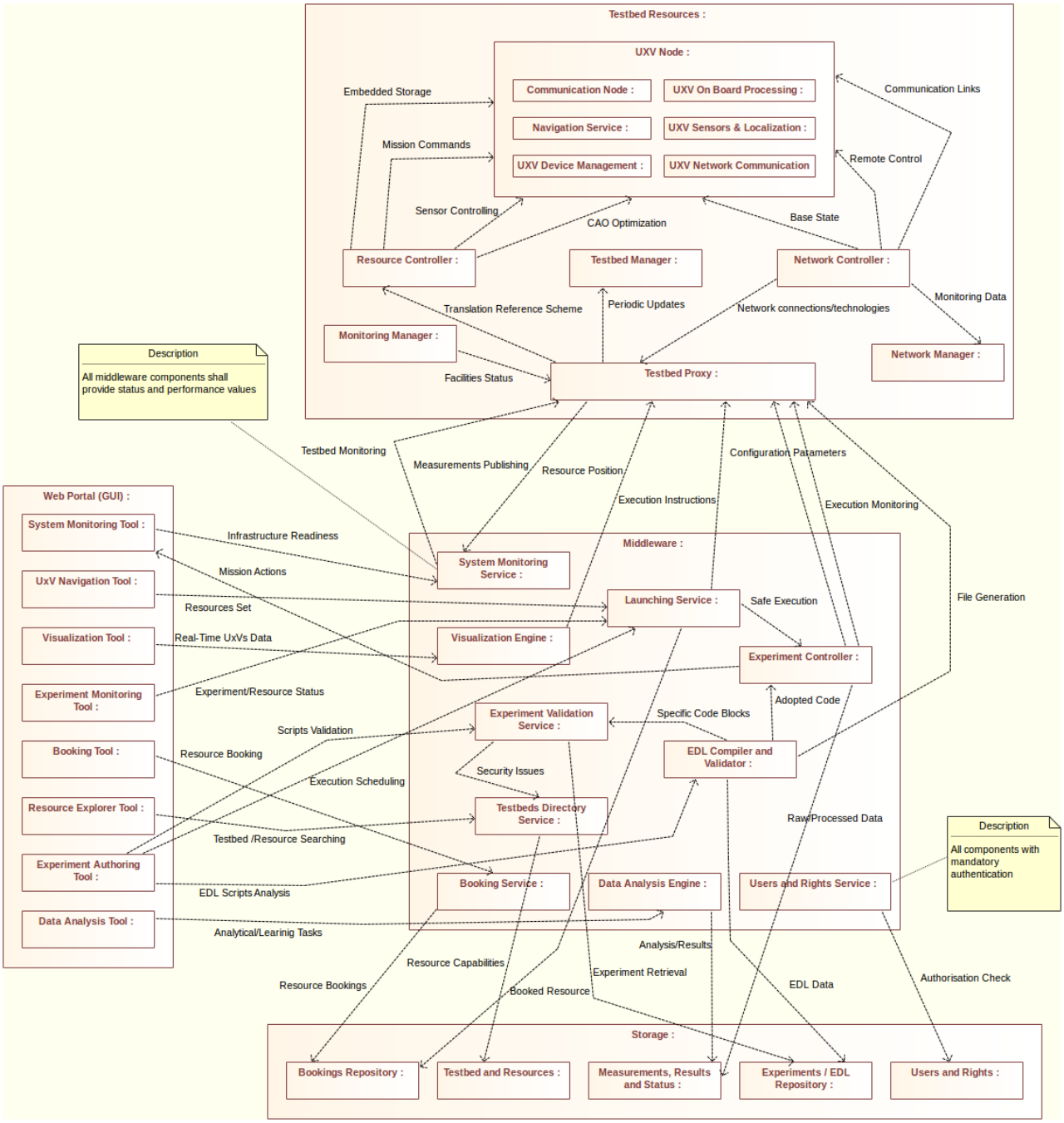

Figure 1: Global Architecture

2.2.1 Description and representation of main conceptual components

For the purpose of describing in detail the various components in a very well-organized way, the following logical component grouping is suggested:

- Web Portal (GUI) components
  Likewise in the deliverable D4.1, the web portal category includes all those components related to user interaction with the system. Hence through a web browser for example, the user will be able to access and utilize all the tools provided by the platform. In this way, the service provisioning according to the cloud computing approach principle is satisfied, and at the same time the ubiquity advantage is leveraged.

- Communication components
  Includes all components that represent a logic service and communicate with the front end tier as well as with the various repositories. Core components implementing essential experiment functionalities are explained in this section.

- Testbed control, monitoring and analysis components
  These are basically middleware components, but only those strictly related and linked to the resource mission control and status of the different devices across multiple facilities in the federation.

- Testbeds facility and resource components
  All the components that take actions on any testbed facility and resource management are included in this category. Particular components units making up a single node resource are also described here.

2.3 Sequence Diagrams from Input to Output

With the objective to depict practically all the components involved in the system, two (2) main scenarios have been identified. These scenarios, originated from the use cases presented in D3.1, aim to thoroughly describe the sequence of actions to be performed by RAWFIE federation components and clearly depict the input and output information flows between them. For example, starting with a concrete experiment action or resource mission, and finishing with the consequent data transmission to the UxV resource node. The general idea is to design global interaction sequence diagrams that involve practically all the components of the platform.

2.3.1 Sequence diagram A - Monitoring water canals

2.3.1.1 Components involved

- Resource Explorer Tool
- Testbeds Directory Service
- Testbeds & Resources Repository
- Experiment Authoring Tool
- EDL Compiler & Validator
- Experiment Validation Service

- Experiments & EDL Repository
- Launching Service
- Experiment Controller
- Message Bus
- Testbed Proxy
- Testbed Manager
- Resource Controller
- UxV Node
- Experiment Monitoring Tool
- Visualisation Tool
- Visualisation Engine
- Data Analysis Tool
- Data Analysis Engine
- Measurements, Results & Status Repository

2.3.1.2 Data flow description

This first use case scenario presents the process executed by an experimenter for creating and launching a specific experiment. After the data generation and retrieval, the experimenter can get access to this particular information in order to perform a consequent analysis and examination. The sequence of activities that take part in the scenario are described below:

i. Initially, the experimenter searches for available resources, depending on desired device characteristics and capabilities, in terms of the experiment to conduct.
ii. The experimenter proceeds with the experiment authoring, using the previously defined EDL. Then, registers the script in the repository for future utilization.
iii. After the experiment has been chosen and the script has been saved in the database, the experimenter continues with resource booking. The platform offers the possibility to carry out a long term launching as an alternative to the short one.
iv. The experimenter performs a short term launching of the experiment right away, sending the set of instructions that must be translated or transformed to understandable commands for the resource node.
v. The experimenter obtains the result data generated by the node that executed the mission defined according to the experiment, thus proceeds to examine the experimental data.
vi. Finally, the experimenter decides to visualise some geospatial information with regard to a particular resource employed during the experiment. The platform tracks the resource and the experimenter receives the precise coordinates in real-time.

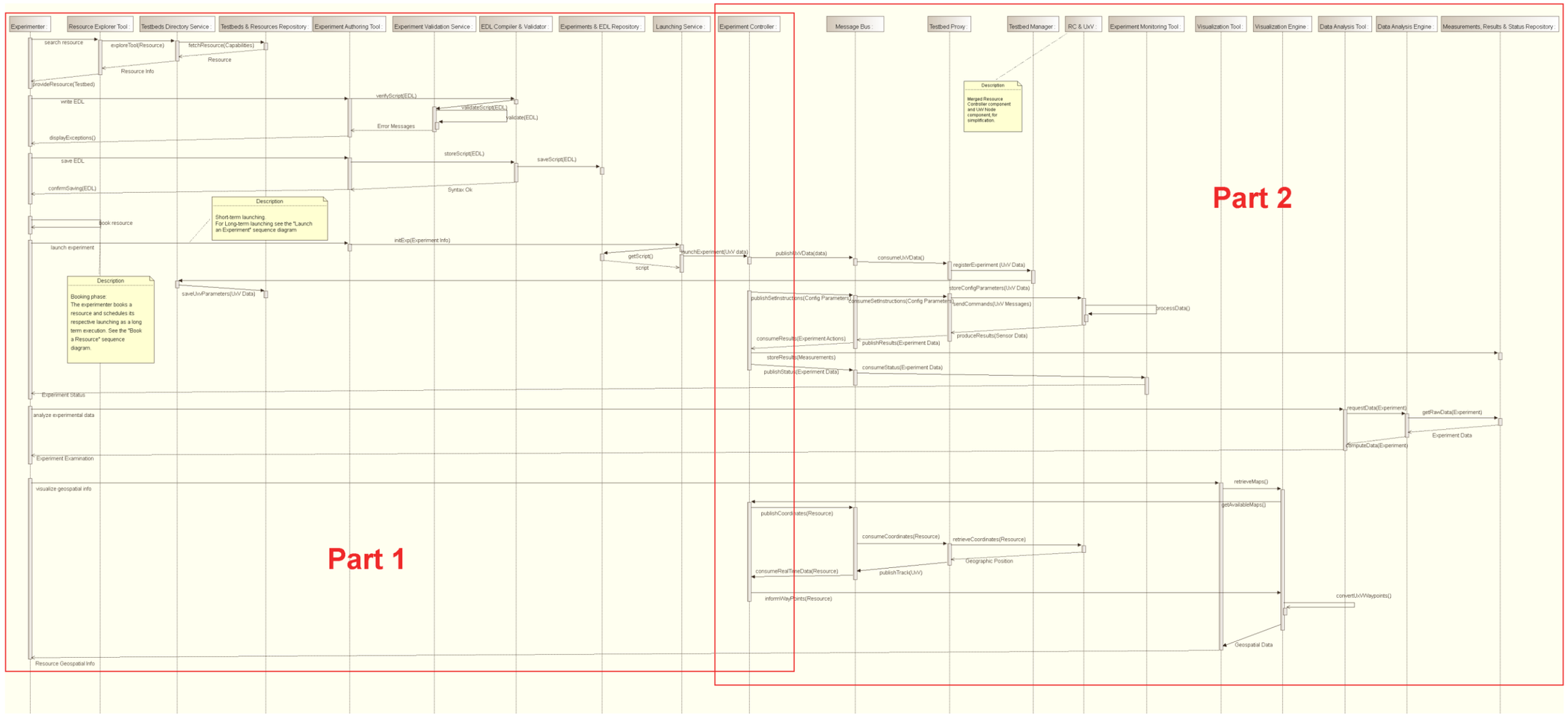

Figure 2: Overall sequence diagram A – Monitoring water canals

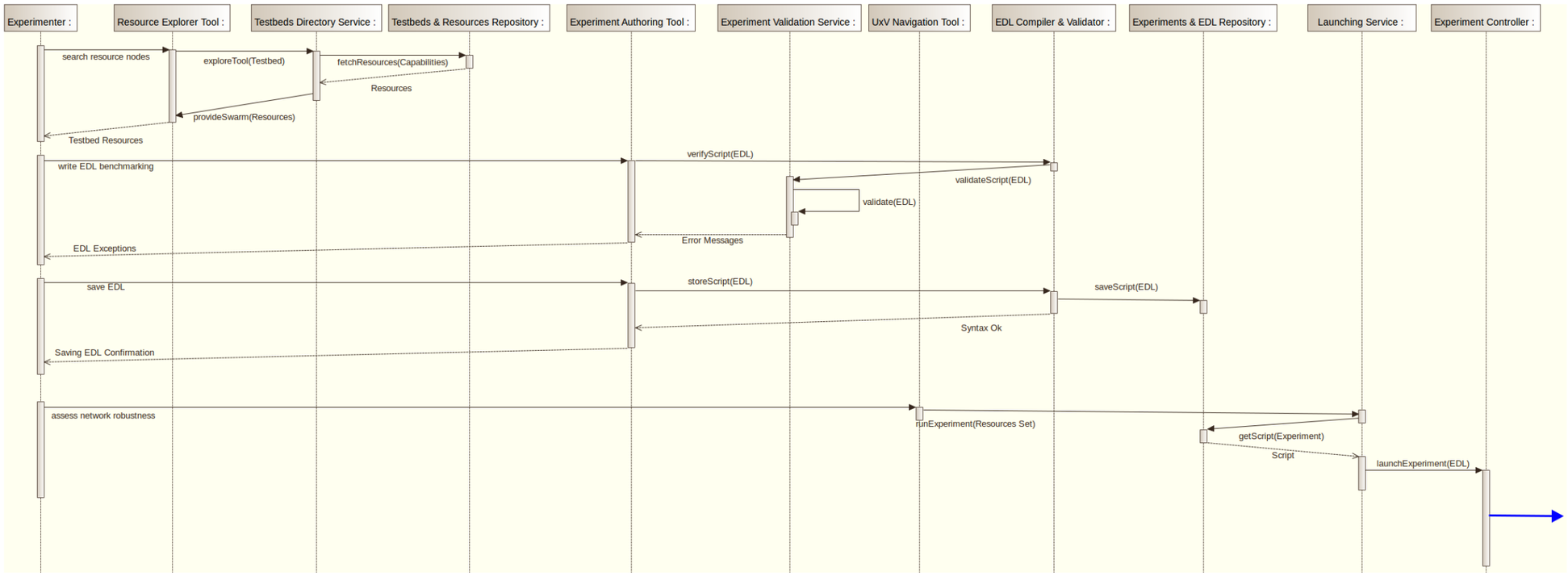

Figure 3: Sequence diagram A – Part 1

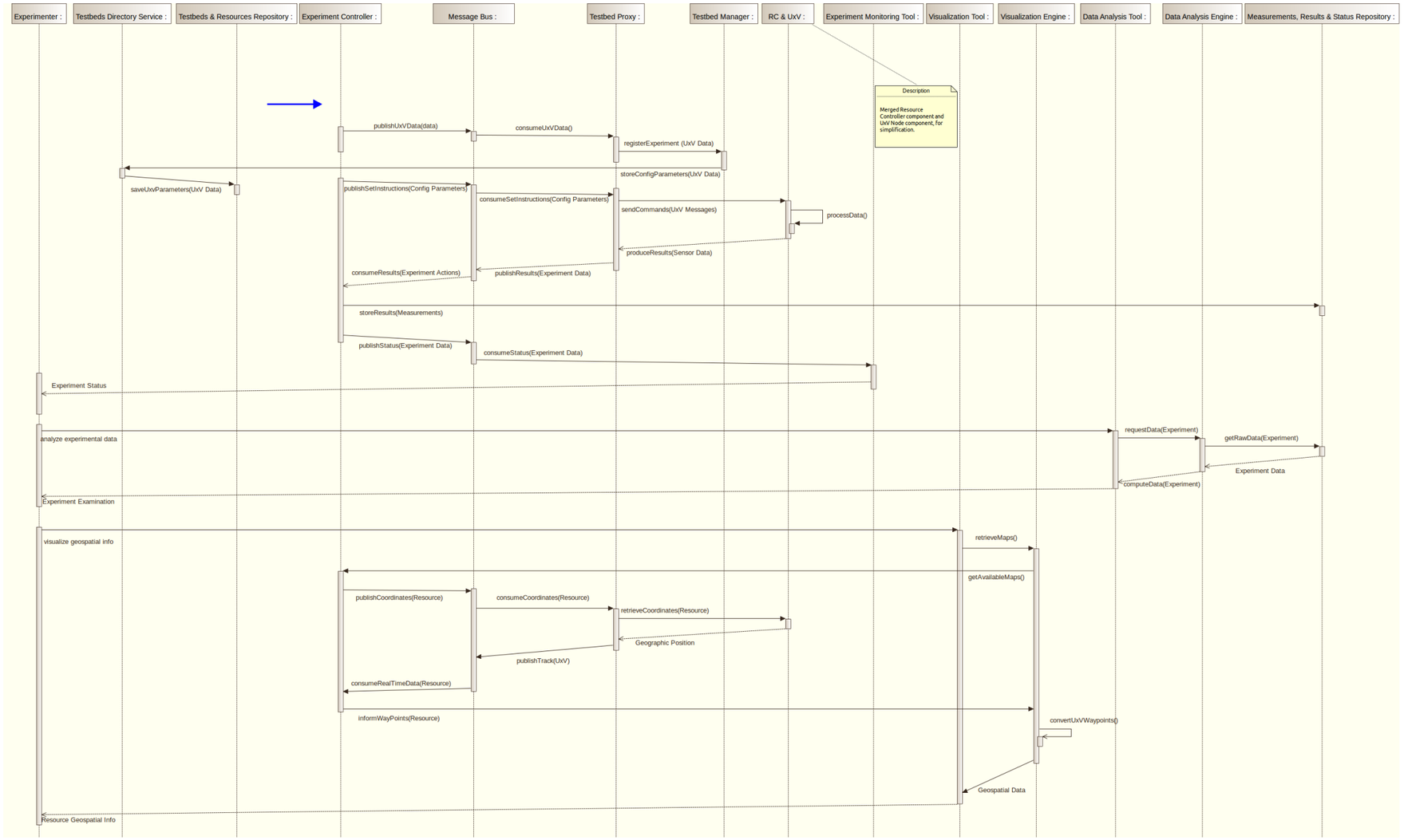

Figure 4: Sequence diagram A – Part 2

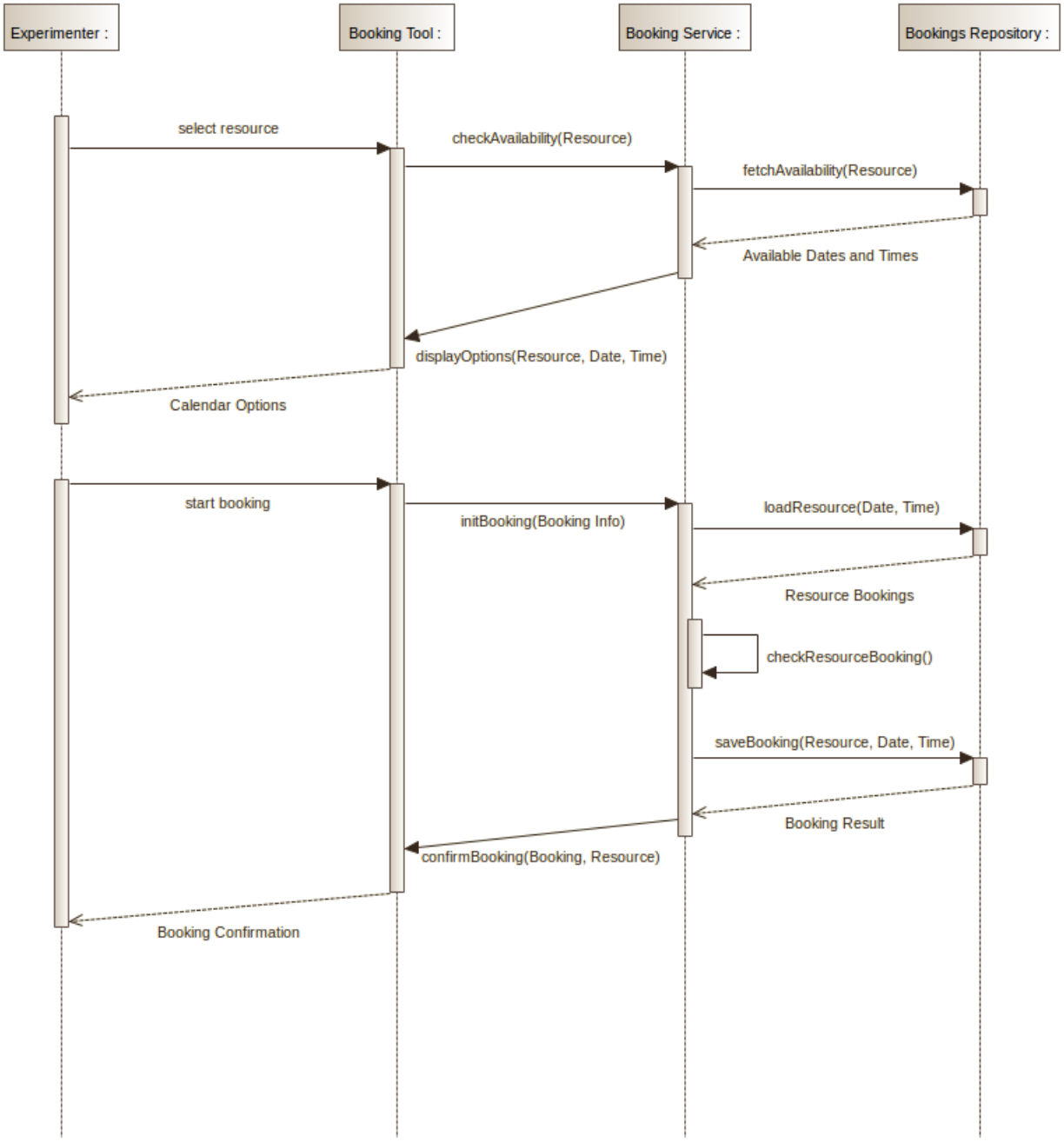

Figure 5: Monitoring water canals – Book a resource

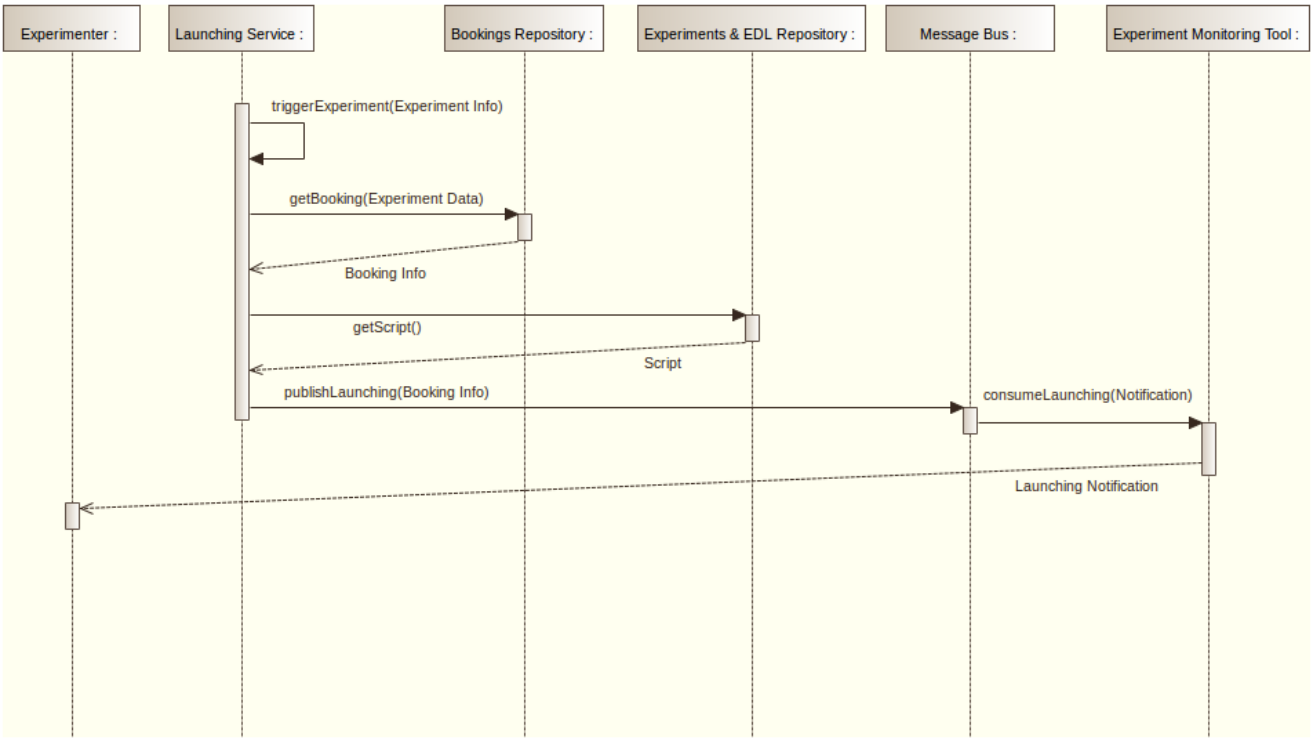

Figure 6: Monitoring water canals – Launch a long term scheduled experiment

2.3.2 Sequence diagram B - Analysing network robustness and performance

2.3.2.1 Components involved

- Resource Explorer Tool
- Testbeds Directory Service
- Testbeds & Resources Repository
- Experiment Authoring Tool
- UxV Navigation Tool
- EDL Compiler & Validator
- Experiment Validation Service
- Experiments & EDL Repository
- Launching Service
- Experiment Controller
- Message Bus
- Testbed Proxy
- Resource Controller
- UxV Node
- UxV Network Communication
- UxV On Board Storage
- UxV On Board Processing

- Experiment Monitoring Tool
- Data Analysis Tool
- Data Analysis Engine
- Measurements, Results & Status Repository

2.3.2.2 Data flow description

The second use case scenario was extracted from the exploration and assessment of network technologies robustness with the goal of performing a benchmark through certain parameters and communication factors for the data distribution over the network coverage area. The following sequences of steps are described:

i. The experimenter first of all selects the group of resources in order to execute the experiment. Once the different capabilities and availability of the resources are fetched, the experimenter selects each one of them for adding it to operational swarm for the mission.
ii. After the resources selection, the experimenter is ready to elaborate the benchmarking experiment and save the corresponding script in the repository.
iii. Through the UxV Navigation Tool the experimenter runs the desirable experiment to the resources set. The launching instructions are parsed to command level scripts and every single resource node receives the command to proceed with the test.
iv. The mission is executed and afterwards the experimenter retrieves all the relevant data such as obstacles impact results and georeferenced data.
v. In order to analyse the network performance, the experimenter makes use of the Data Analysis Tool and corresponding Analysis Engine, by fetching the collected data from measurements repository

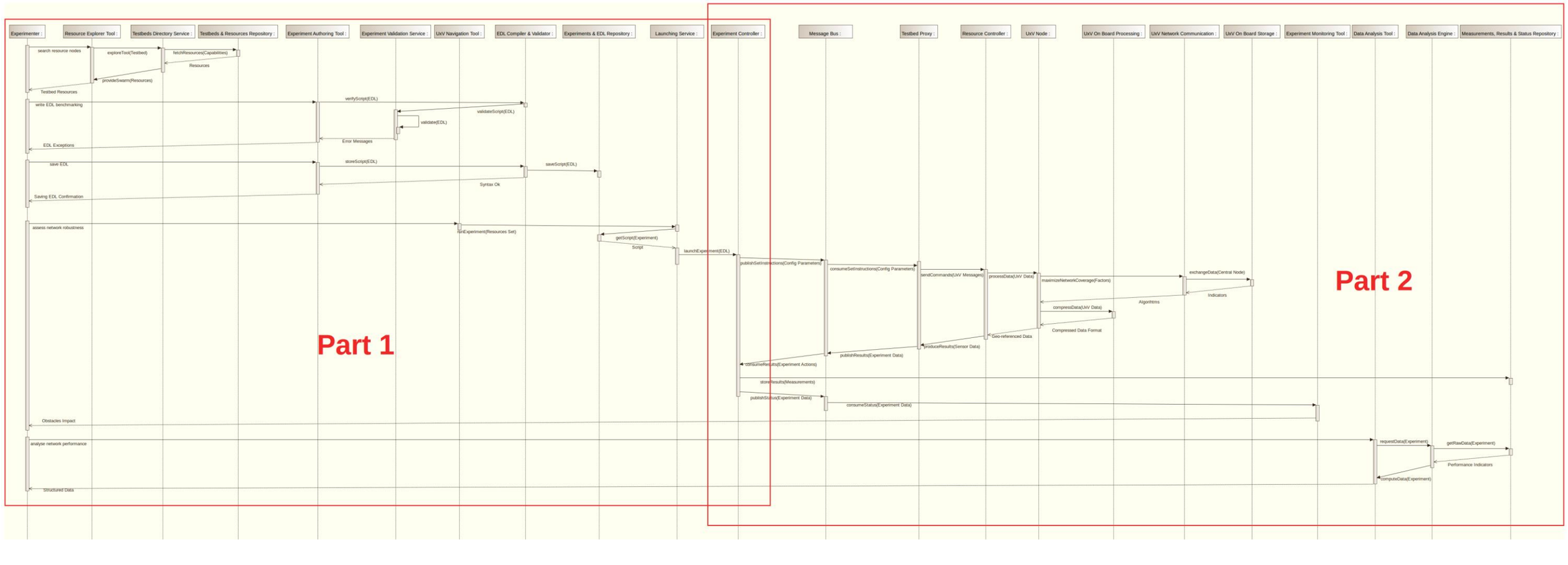

Figure 7: Overall sequence diagram B – Analysing network robustness and performance

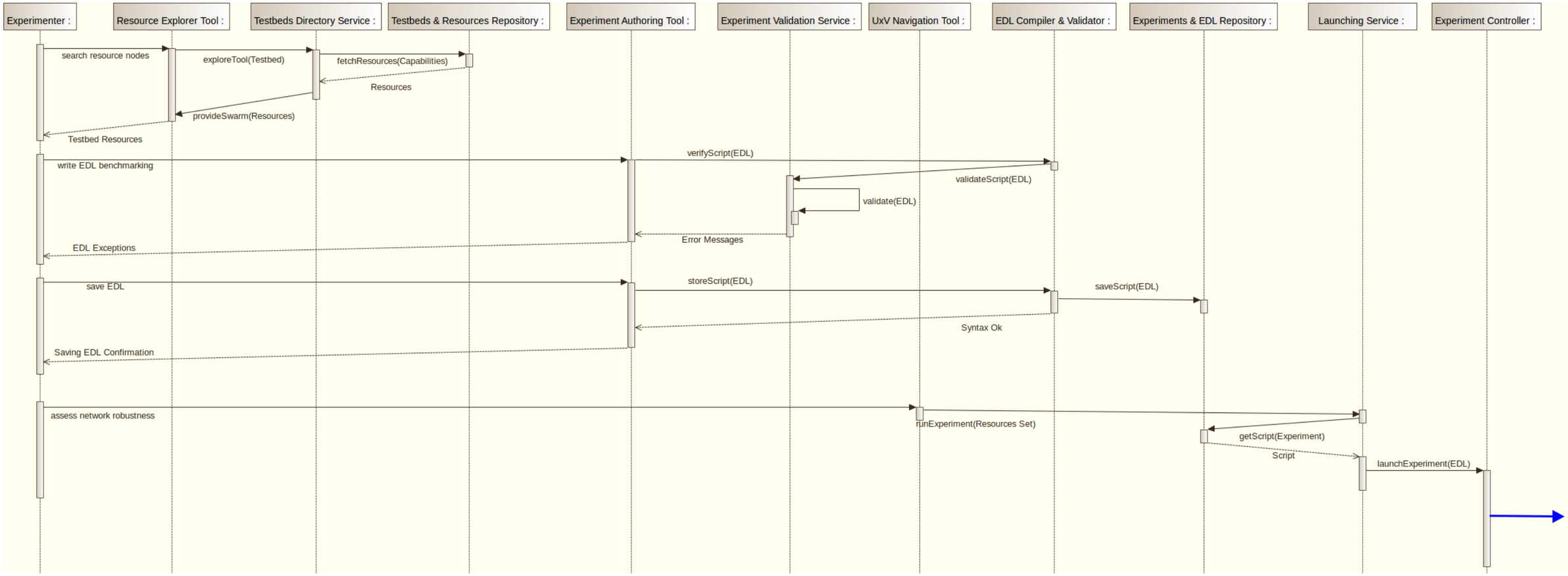

Figure 8: Sequence diagram B – Part 1

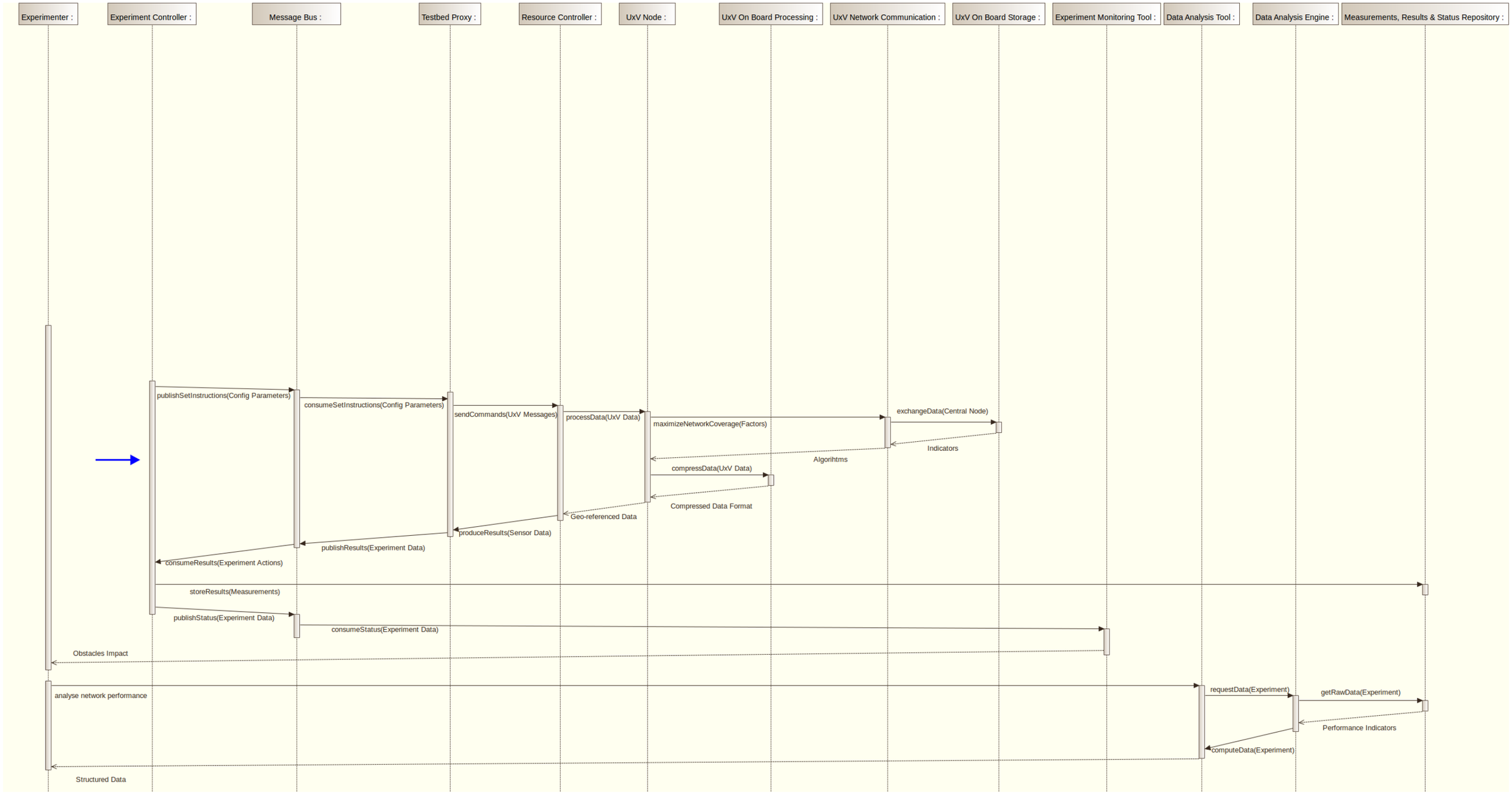

Figure 9: Sequence diagram B – Part 2

3 Definition of conceptual components

3.1 Web portal (GUI) component

3.1.1 Description

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 high level representation of all web portal components and their interactions with middle tier and data tier components is presented in Figure 10.

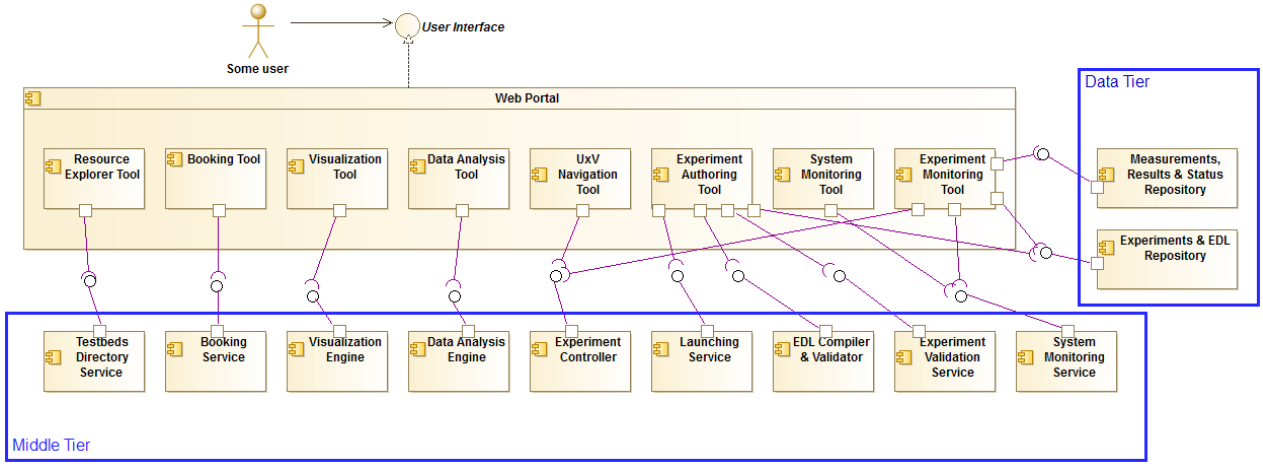

Figure 10: Web Portal – High level Components Diagram

3.1.2 Booking Tool

Making use of this tool the experimenter can discover and select available testbeds as well as resources into a testbed facility.

Responsibilities

The main responsibilities of the Booking Tool are:

- Provides the visualisation as a calendar view mode to watch available dates and prospective time slots of resources and facilities.
- Books the resource for a specific period of time and space fragment in a testbed

Operations and attributes

The Booking Tool provides different web pages to display a calendar view with all the reservations for a specific testbed and its UxVs resources. All the details of a booking could be viewed in an extra page. The tool also provides the ability to add new or edit existing bookings.

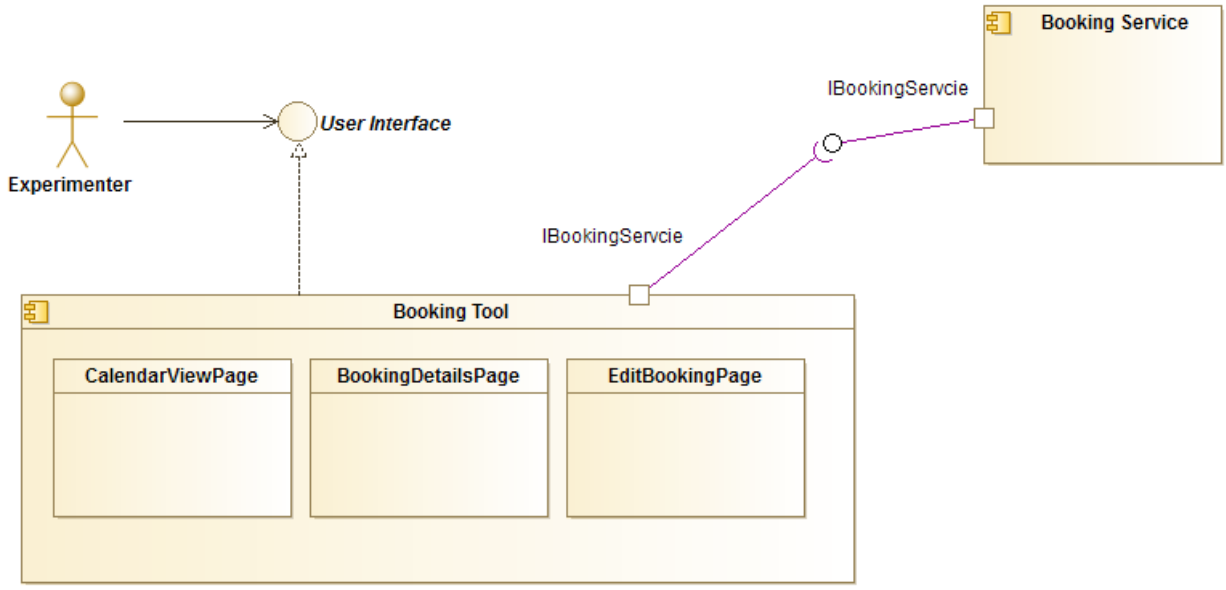

Figure 11: Booking Tool - High level class diagram

The Booking Tool only interacts with the Booking Service. Please see section about the Booking Service (3.2.7) to get a more detailed description.

Interactions and relationships with other components

Provided Interfaces

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

Required Interfaces

- Booking Service Interface (IBookingService):
  Read the bookings for visualisation, add and edit bookings

3.1.3 System Monitoring Tool

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

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 SystemMonitoringPage 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.

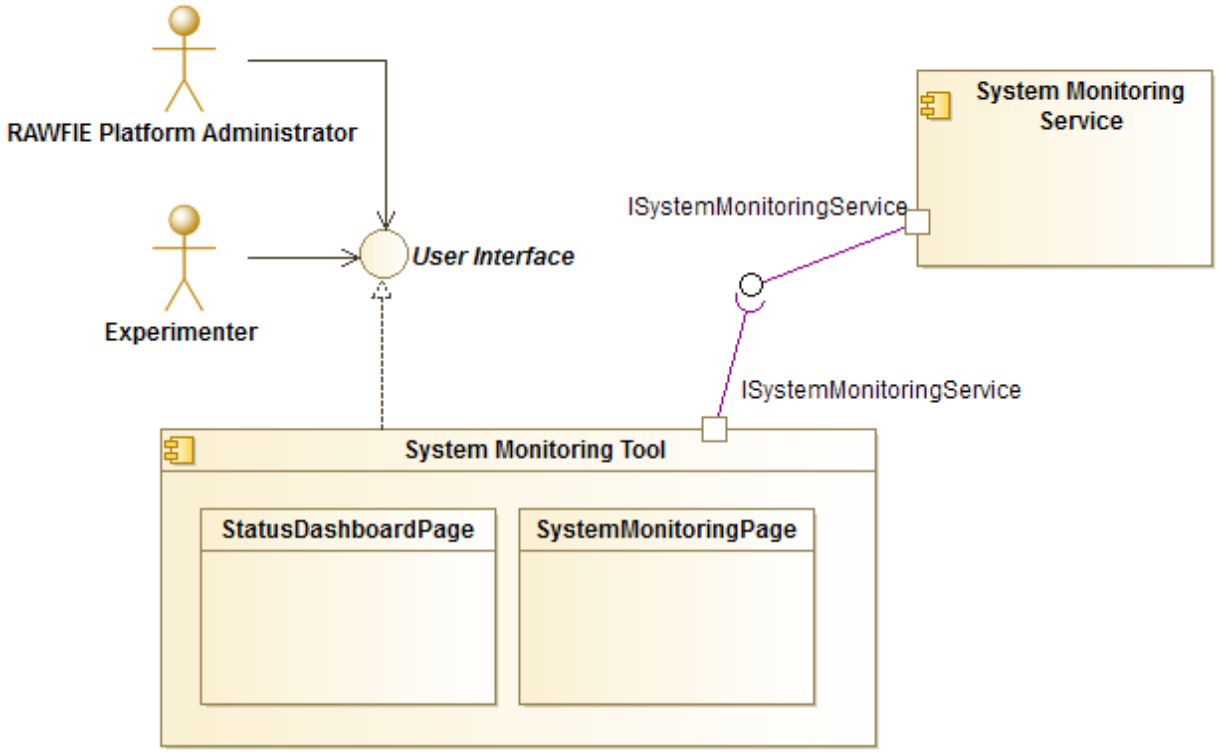

Figure 12: System Monitoring Tool - High level class diagram

The System Monitoring Tool only interacts with the System Monitoring Service. Please see section about the System Monitoring Service (3.3.4) 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

Required Interfaces

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

3.1.4 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.

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 detail of Testbeds and UxV could be viewed on the details web pages. Adding and editing of Testbeds and UxV could be done via the edit web pages.

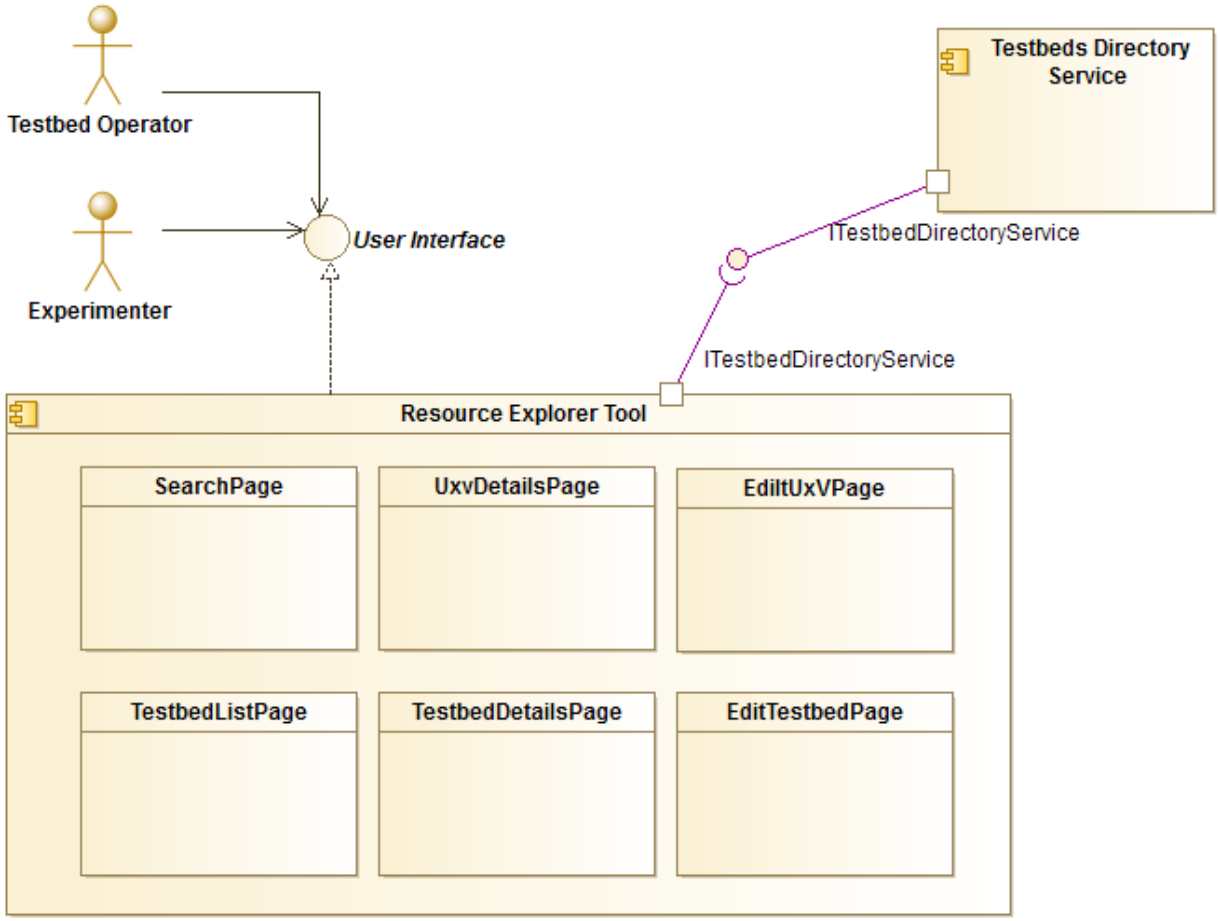

Figure 13: Resource Explorer Tool - High level class diagram

The Resource Explorer Tool only interacts with the Testbeds Directory Service. Please see section about the Testbeds Directory Service (3.2.3) to get a more detailed description.

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 data about testbeds and UxVs.

Required Interfaces

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

3.1.5 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.

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 editors synchronization.
- 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

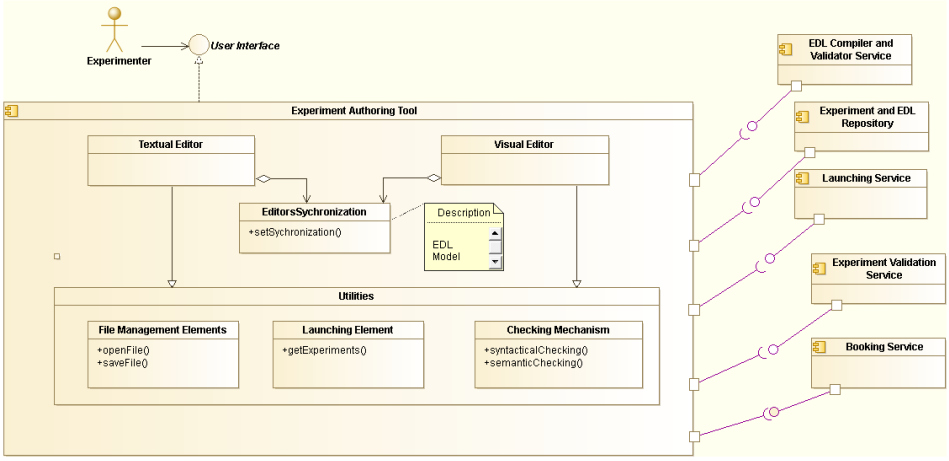

Figure 14: Experiment Authoring Tool - High level class diagram

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

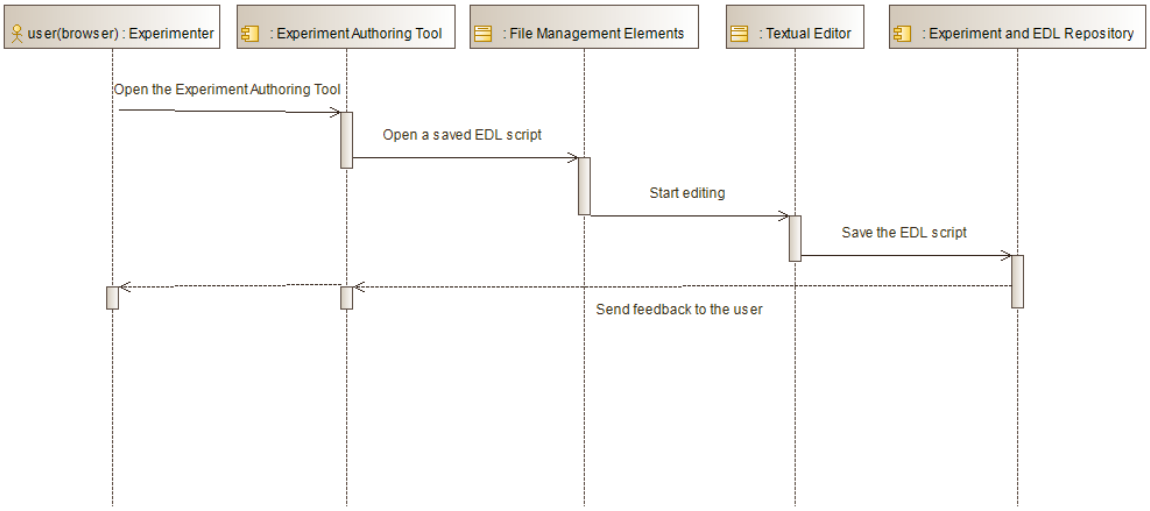

Figure 15: Experiment Authoring Tool – Open and edit an EDL script

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

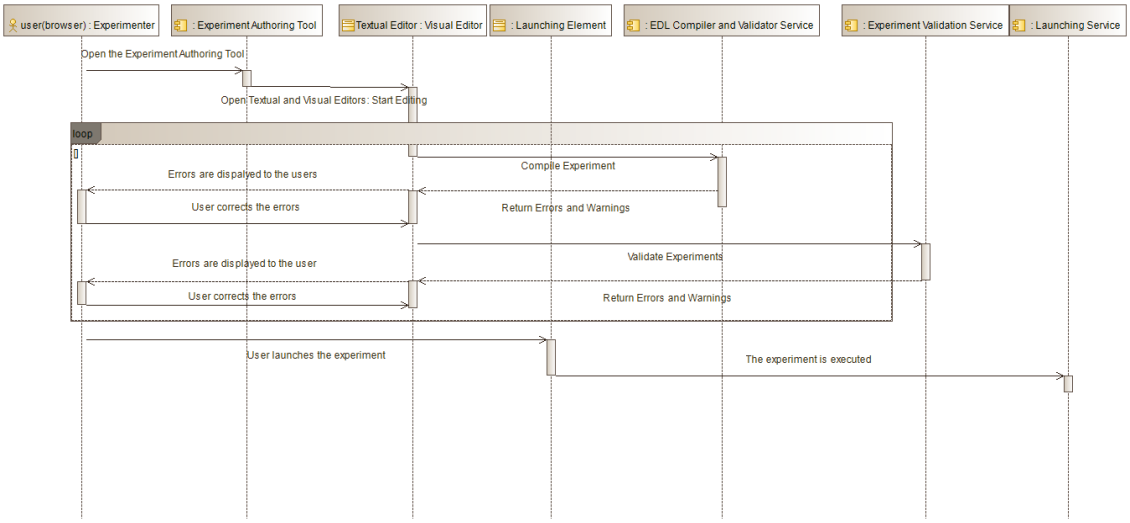

Figure 16: Experiment Authoring Tool - Create and validate an EDL experiment

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

3.1.6 Experiment Monitoring Tool

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

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 collected 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.

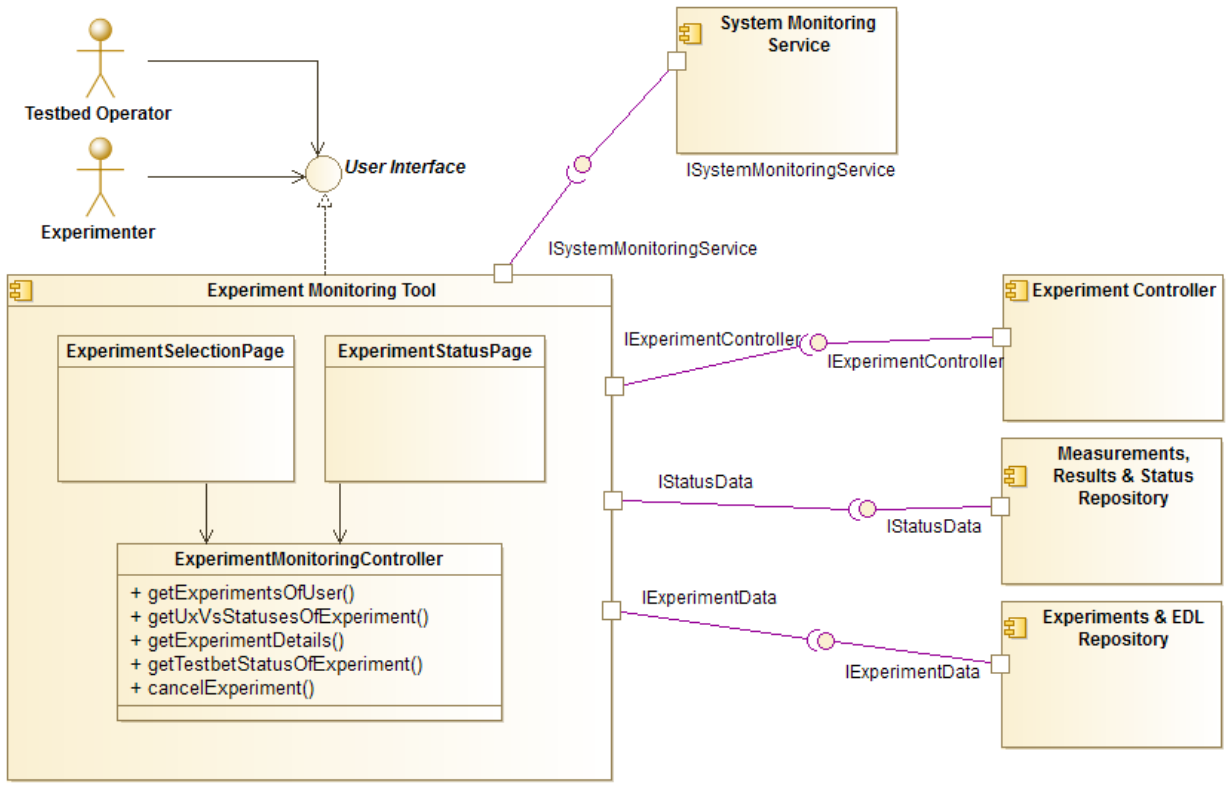

Figure 17: Experiment Monitoring Tool - High level class diagram

Select experiment

1. The user opens the experiment selection page at the Experiment Monitoring Tool
2. The ExperimentMonitoringController is asked for the list of experiments of the current user

3. ExperimentMonitoringController loads the experiment list from the Experiments & EDL Repository
4. The list is shown to the user and he selects an experiment to get more details

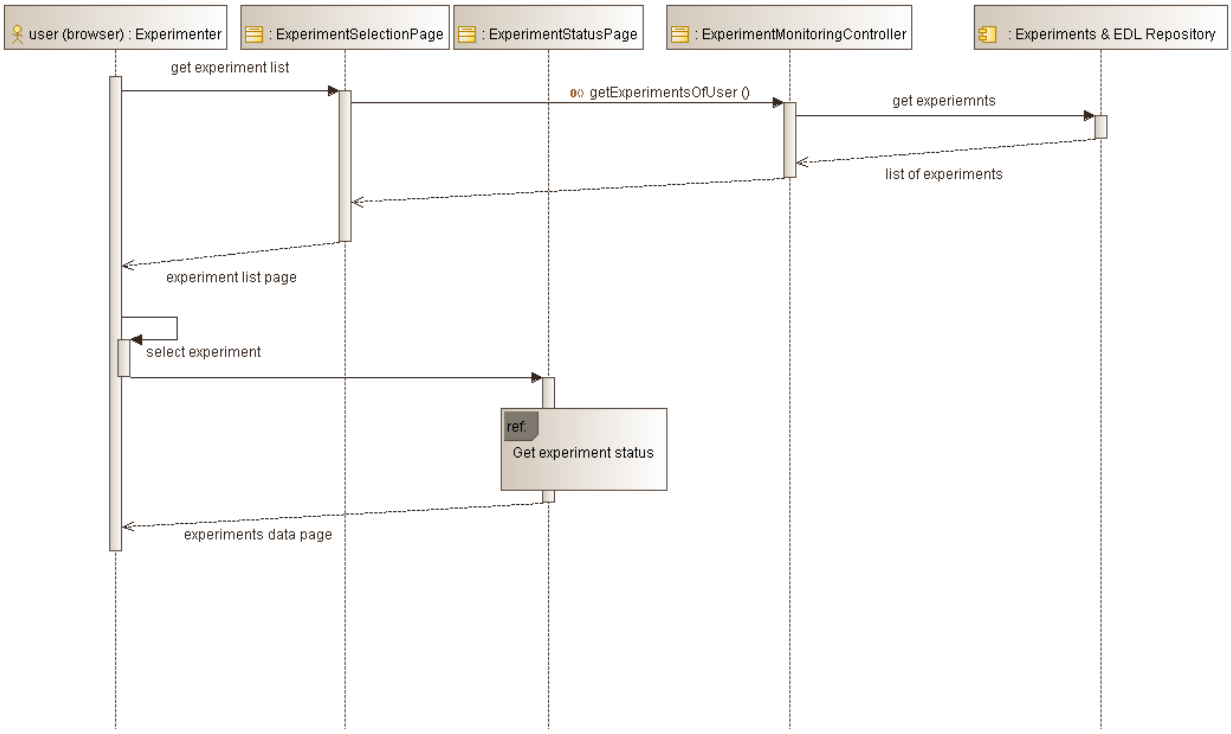

Figure 18: Experiment Monitoring Tool – Select experiment

Get experiment status

1. The user requests the status of the experiment
2. The experiment details are queried
    a. Static details of the experiment are loaded from the Experiments & EDL Repository
    b. Status of the experiment are loaded from the Measurements, Results & Status Repository
3. The UxV statues are queried from the System Monitoring Service
4. The Testbed statues are queried from the System Monitoring Service
5. The collected data is shown to the user

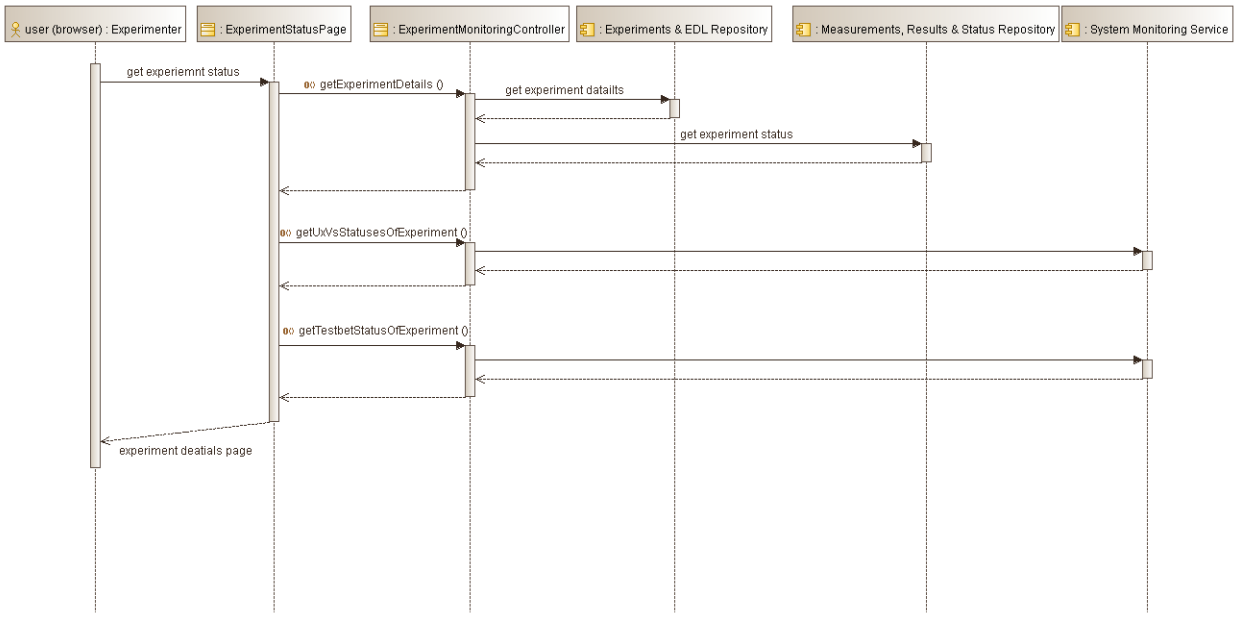

Figure 19: Experiment Monitoring Tool – View experiment details

Cancel experiment

1. On the experiment status page, the user clicks on the "Cancel" button
2. The request is forwarded to the experiment controller that executes the necessary steps to cancel the experiment

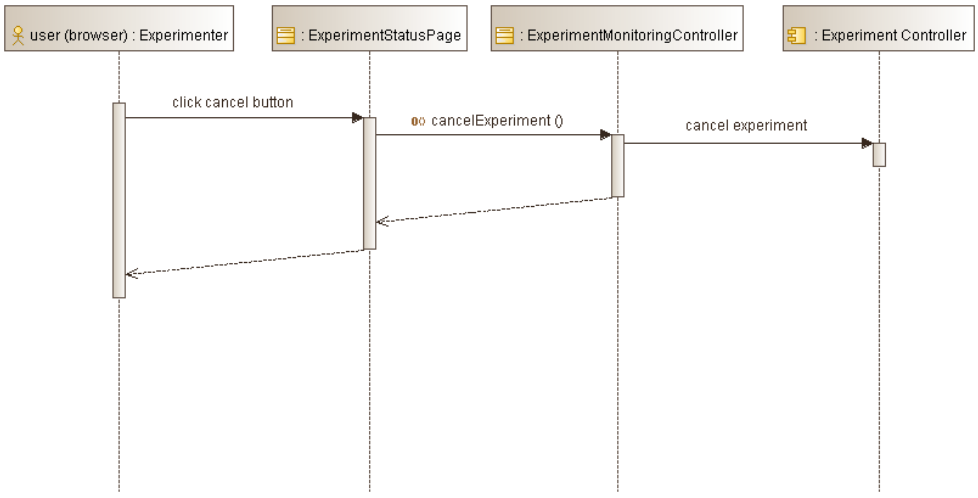

Figure 20: Experiment Monitoring Tool – Cancel experiment

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 Interface (ISystemMonitoringService):
  get status information about involved testbeds and UxVs.
- Experiments & EDL Repository Interface (IExperimentData):
  Query the experiments of a user
- Measurements, Results & Status Repository Interface (IStatusData):
  Query information about the experiment status.
- Experiment Controller Interface (IExperimentController):
  To cancel an experiment the Experiment Controller is called to execute the necessary steps.

3.1.7 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 so as to perform basic navigation missions such as waypoint navigation, map construction, area surveillance and path planning.

Responsibilities

The main responsibilities of the UxV Navigation Tool are:

- guide the vehicles using a turn based navigation mechanism
- collect data from their equipped sensors.

Operations and attributes

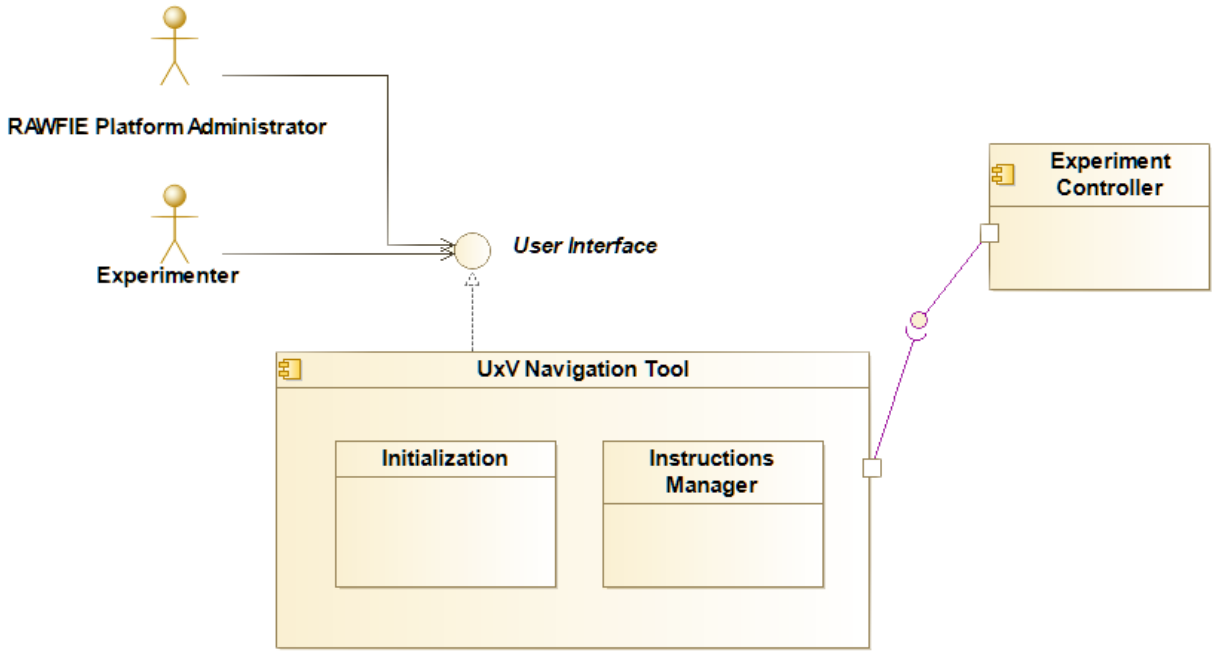

Figure 21: UxV navigation tool – High level class diagram

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.

- Through the provided interfaces, users, specify the next desired location for each unit.
- The instructions manager class translates these instructions into a JSON file and transmits this file to the Experiment Controller.

- When all the vehicles reach their desired position, the UxV Navigation Tool is ready to accept a new set of instructions.

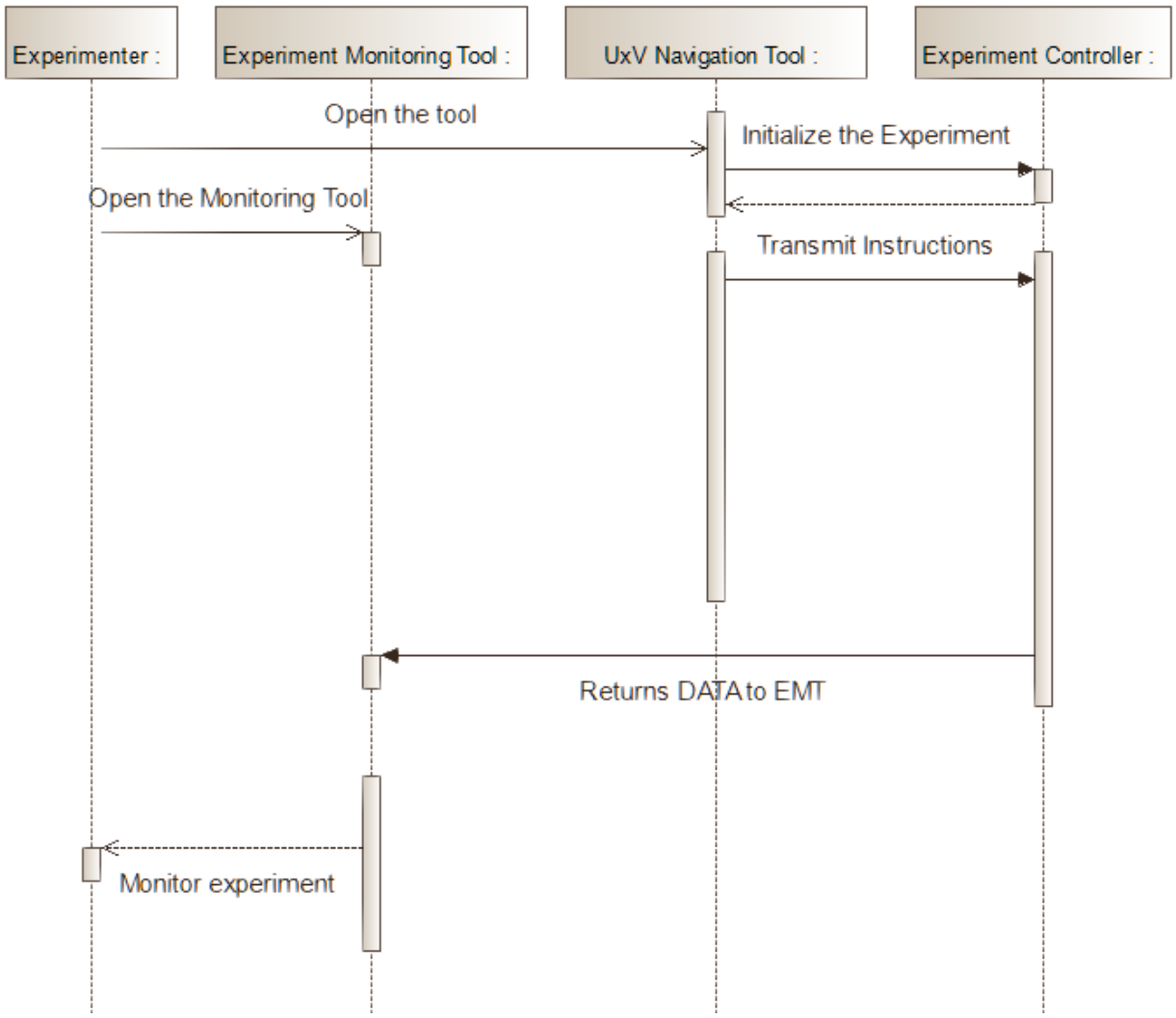

Figure 22: UxV navigation tool – High level sequence diagram

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 2 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 cancelation of an experiment. Experiment Controller is responsible for transferring messages between these two components.

3.1.8 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.

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.

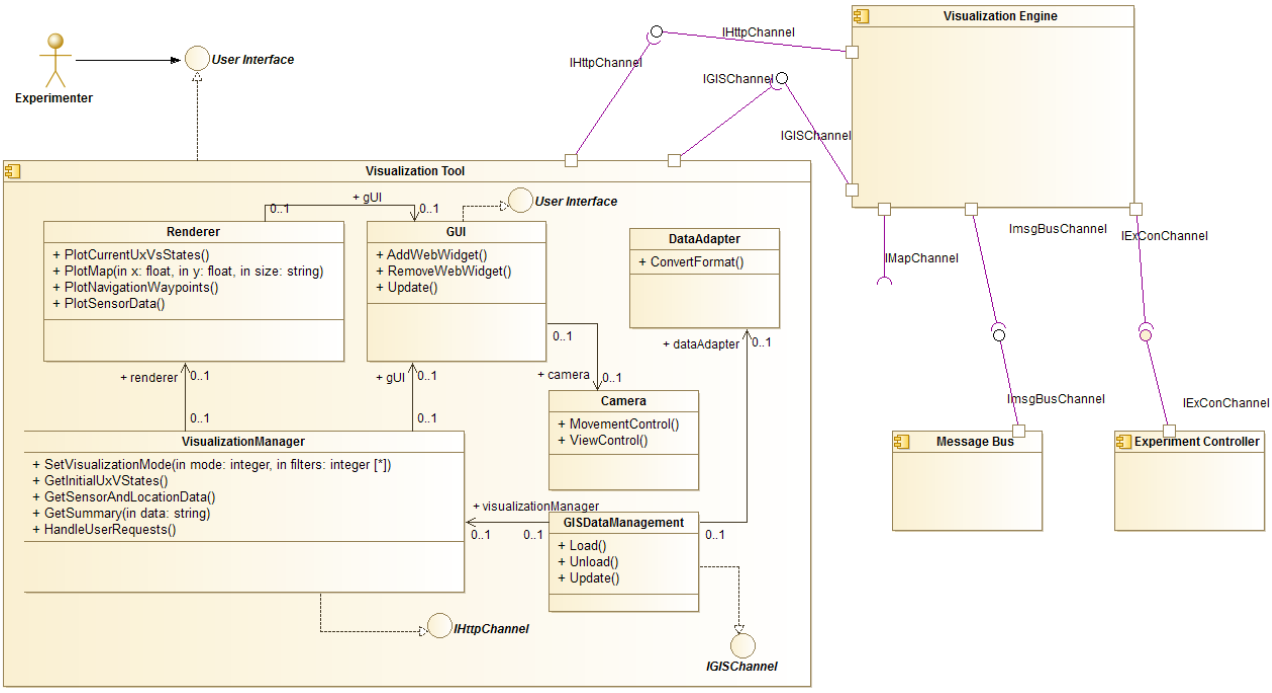

Figure 23: Visualisation Tool - High level class diagram

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 http channel, but the response will be received over the GIS 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 others. This data will be received from the VE over the http channel
- 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

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

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

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.

The user can then match the view of the VT to hers/his requirements.

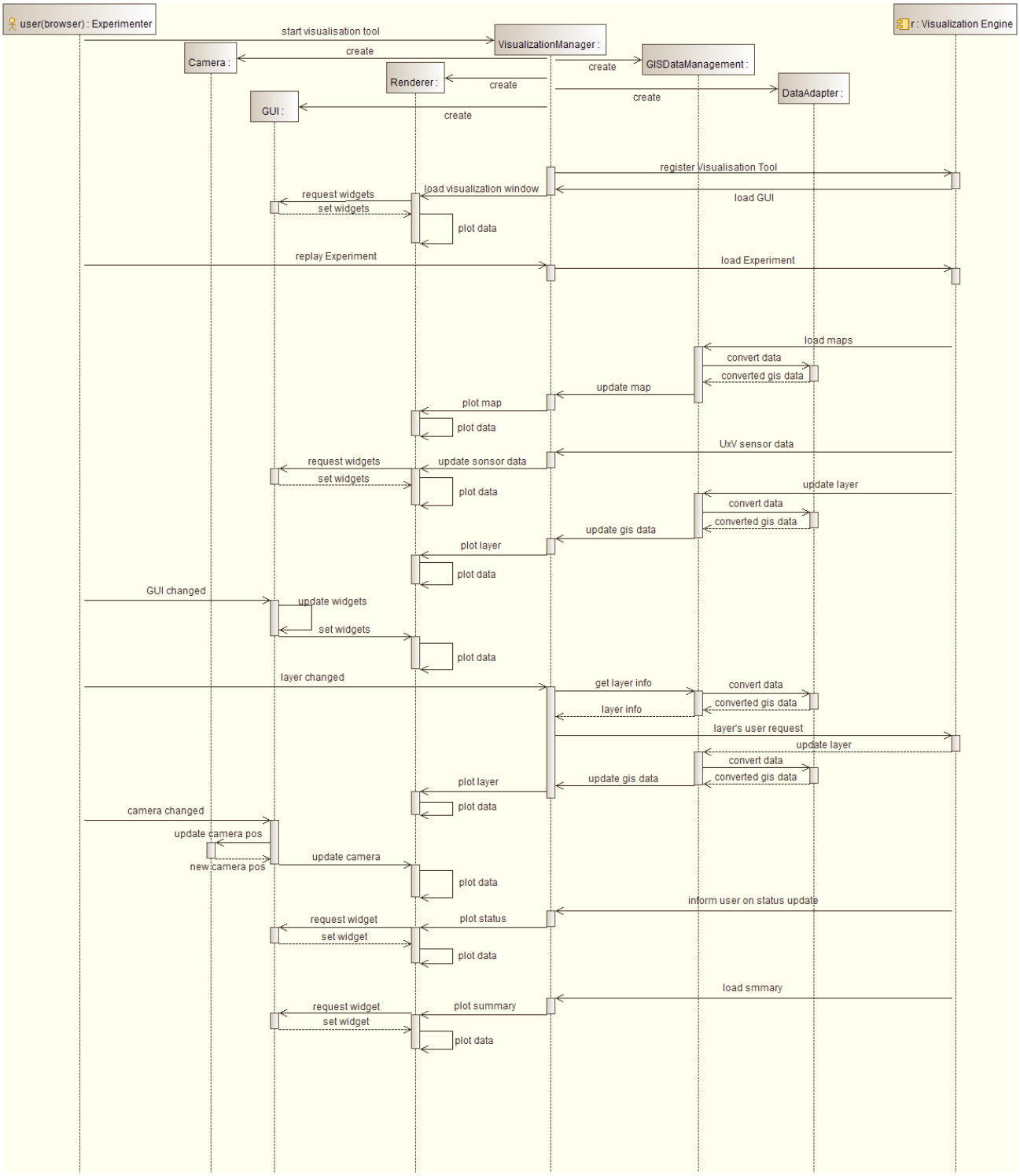

Figure 24: Visualisation Tool - High level sequence diagram

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 http interface and the geo-information data is sent over the GIS interface.
- The http 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.

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.

3.1.9 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.

Responsibilities

The main responsibilities of the Data Analysis Tool are:

- Browse data and results for further analysis
- Select data and results for further analysis
- Enable the design of data analysis jobs by the Data Analysis Engine

Operations and attributes

The high level class diagram of Data Analysis Tool is presented in Figure 25

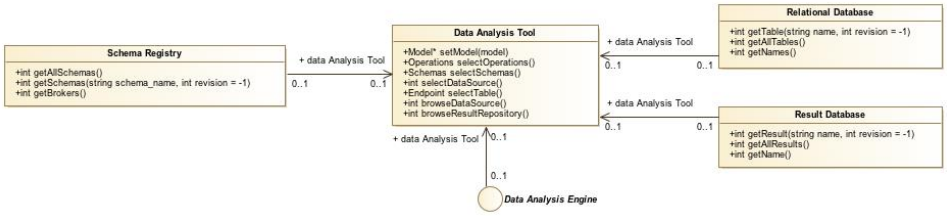

Figure 25: Data Analysis Tool – High level class diagram

- A sequence diagram representing the data flow and the interactions between subcomponent of the Data Analysis Tool is presented in Figure 26.

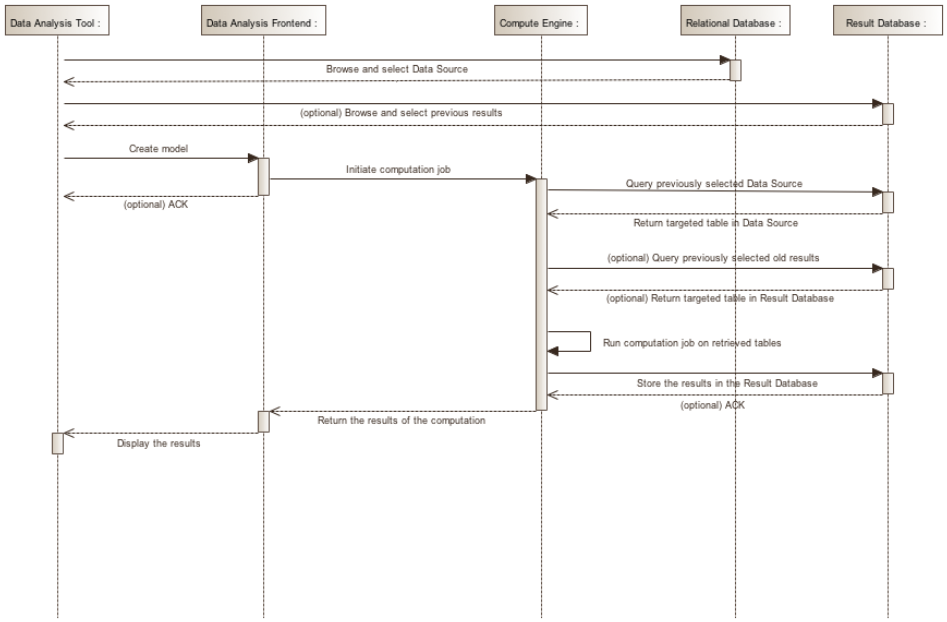

Figure 26: Data Analysis Tool – High level sequence diagram

3.2 Communication components

3.2.1 Description

Communication components represent the core of the RAWFIE platform, since they execute and control all the operations and information flows realized by the implemented methods across the underlying software architecture. Hence, an optimal design of the communication between the different components assigned to this classification must be provided, considering the implementation and deployment of the previously analysed cutting-edge technologies presented in the past deliverables. Figure 27 provides an overview component diagram listing the various involved entities.

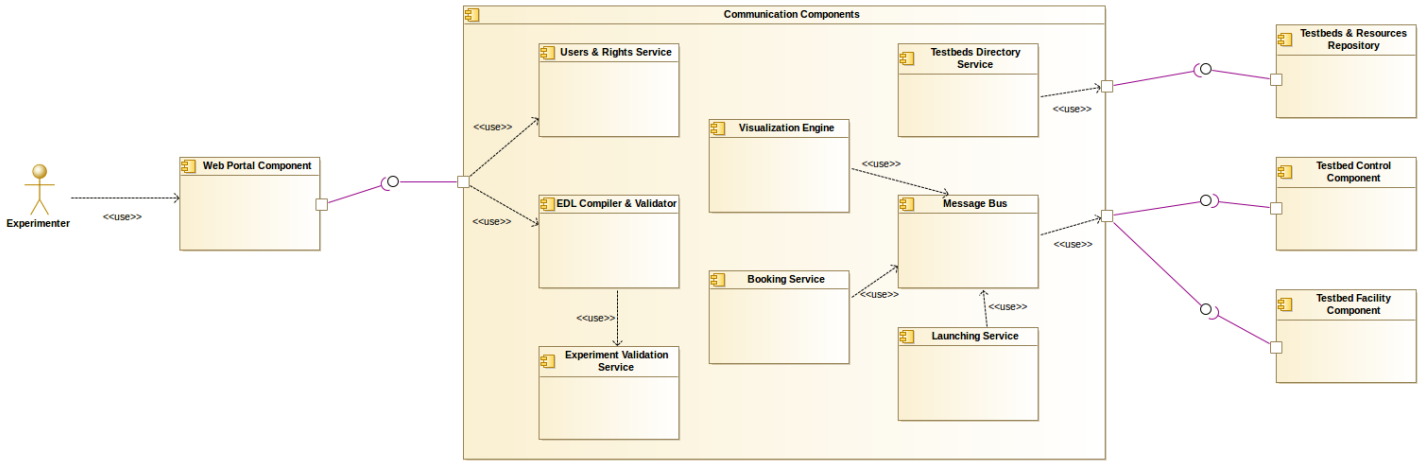

Figure 27: Communication components – High level Components Diagram

3.2.2 Message Bus

A message oriented middleware is shared across all application tiers in order to enable asynchronous and event-based messaging communication between heterogeneous components. It implements the Publish/Subscribe paradigm.

Responsibilities

The main responsibilities of the Message Bus are:

- Deal with asynchronous notifications caused by specific platform events.
- Handle Publisher/Consumer requests among different components.
- Handle messages versioning through its inner schema registry module.

Operations and attributes

The Message Bus represents one of the core components of the RAWFIE platform, since allows the asynchronous communication between several components at different application levels. In this way, critical platform activities and operations can be successfully and efficiently executed. From the cloud computing deployment model perspective, it is necessary to describe carefully and analyse such functionalities, especially those related to interface communications and information flow between different components.

In order to depict the working context of the *Message Bus* middleware component, the subsequent abstract class diagram is provided, seeking to shed light to the interaction aspects between several components and highlighting the respective interfaces.

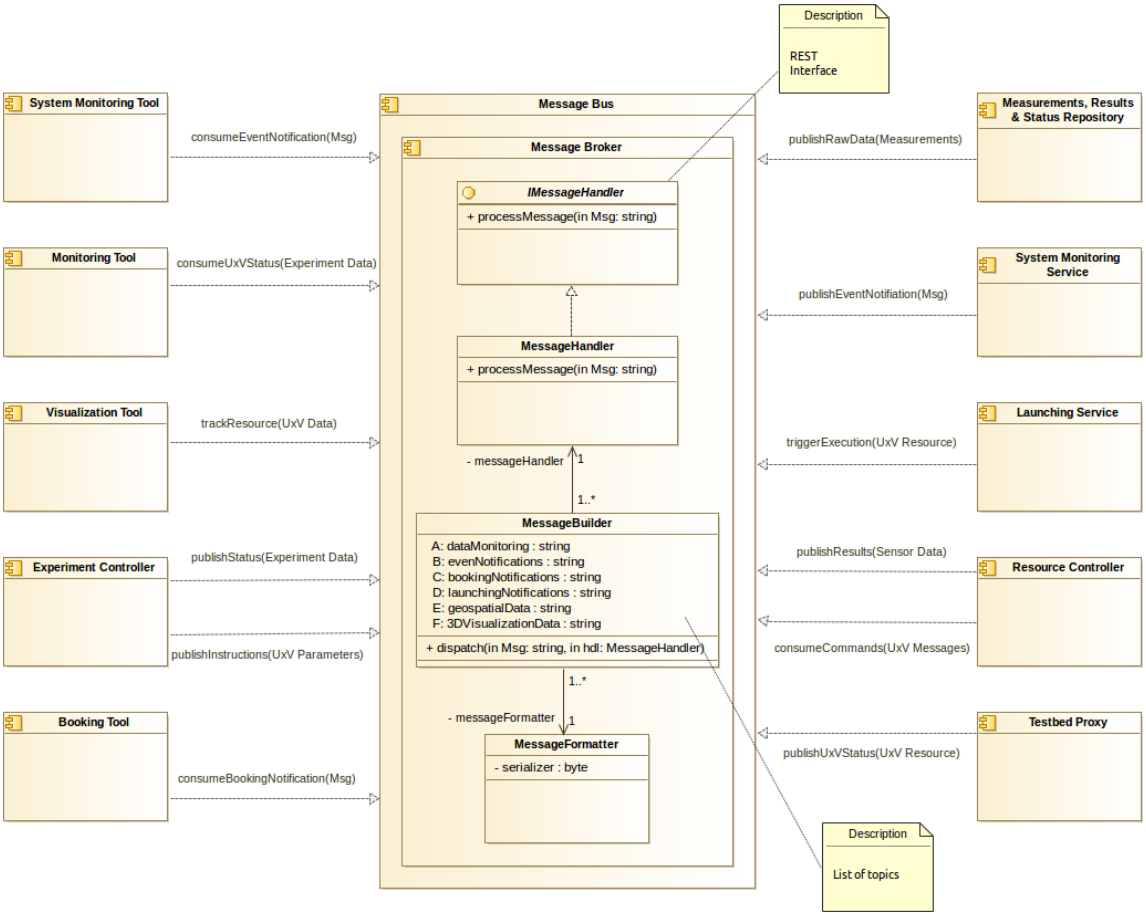

Figure 28: Message Bus - High level class diagram

In the section below the most relevant use cases are shown, where the element interactions through the message bus are considered of prime importance, with the objective of managing an experiment's lifecycle and/or analysing the collected sensor data. These functionalities are provided by means of the following multiple sequence diagrams:

Platform level event/alert notification

1. The Message Bus gets periodically from all middle tier components the status values (KPI) regarding their performance and safety.
2. The MessageHandler processes the values received and generates the corresponding message by creating and serializing the KPI data. Afterwards, it dispatches the message recently created to the subscribers for its consumption.
3. The System Monitoring Service consumes the KPIs values from the Message Bus, and evaluates the different status and thresholds.
4. The System Monitoring Service detects an issue, due to bad functioning of any component or low performance underneath certain threshold.
5. The System Monitoring Service immediately generates the user notification message with regard to the warning produced.
6. The System Monitoring Service forwards the message to the broker for the proper distribution.
7. The Message Bus acts as broker and redistributes the incoming message to the System Monitoring Tool.
8. The platform administrator receives asynchronously by email the notification generated previously from the System Monitoring Tool, therefore he can implement the proper measures to solve or mitigate the issue.

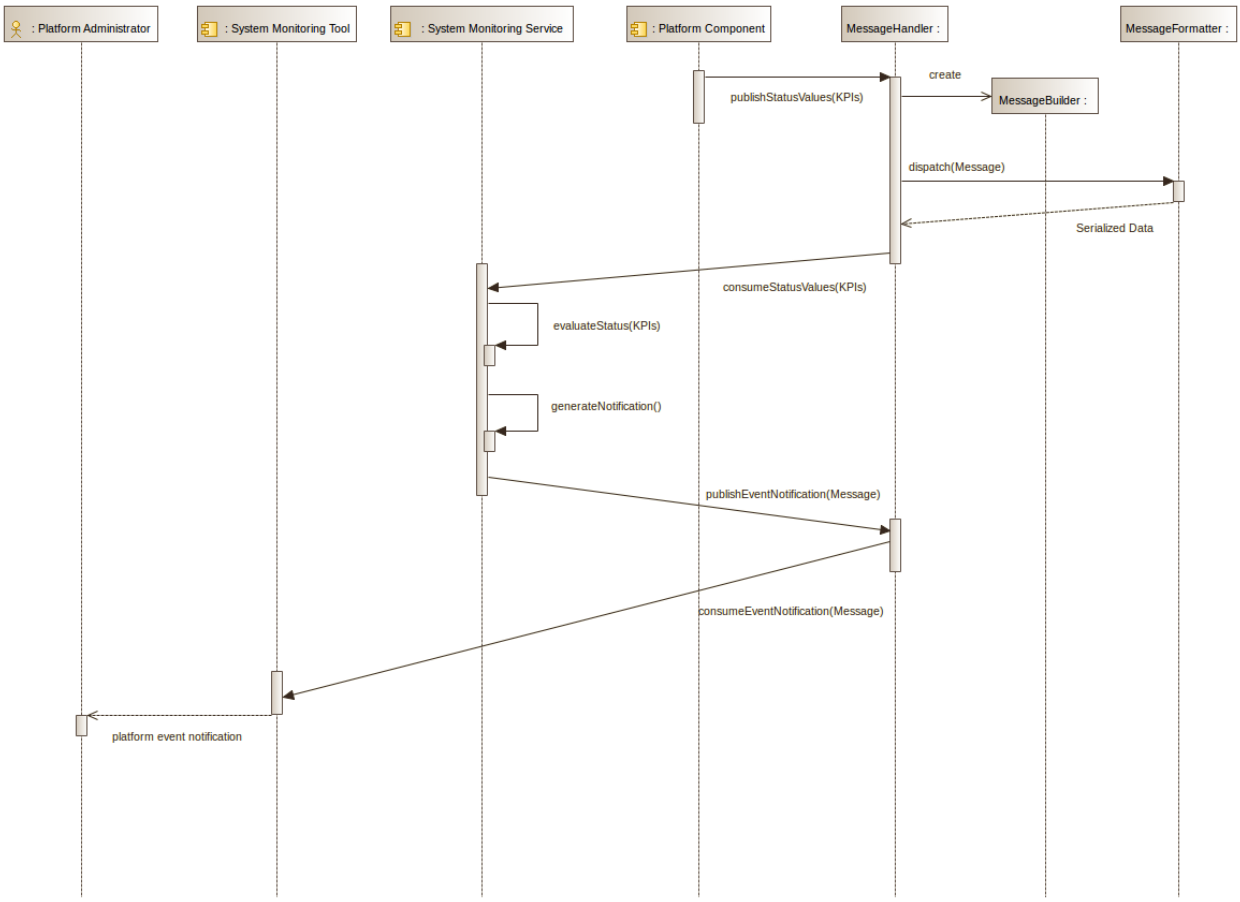

Figure 29: Message Bus - Platform Level event/alert notifications

Launching execution notification

1. The Booking service generates the event according to the experiment schedule and booking.
2. The Launching Service triggers the execution of the experiment to the booked resource, by requesting to the EER the set of instructions.
3. The Launching Service forwards the instructions to the Experiment Controller, which transforms and serializes the data.
4. The Experiment Controller publishes the experiment data to the Message Bus.
5. The MessageHandler gets the request and generates the corresponding command message by creating and serializing the instruction data gotten from the Experiment Controller. Then, it dispatches the message generated to the subscribers for the mission execution.
6. The Testbed Proxy consumes the experiment instructions from the Message Bus and forwards the data to the Testbed Manager for the experiment registration.
7. The Testbed Proxy also sends forward the experiment data to the corresponding Resource Controller, which delivers the execution to the desired UxV node.

8. The UxV node performs the respective actions and dispatches the sensor data back to the Resource Controller.
9. The data collected together with the geospatial location of the resource are published in the Message Bus.
10. The Experiment Controller consumes the results and positioning from the Message Bus, and finally notifies the experimenter about the scheduled launching.

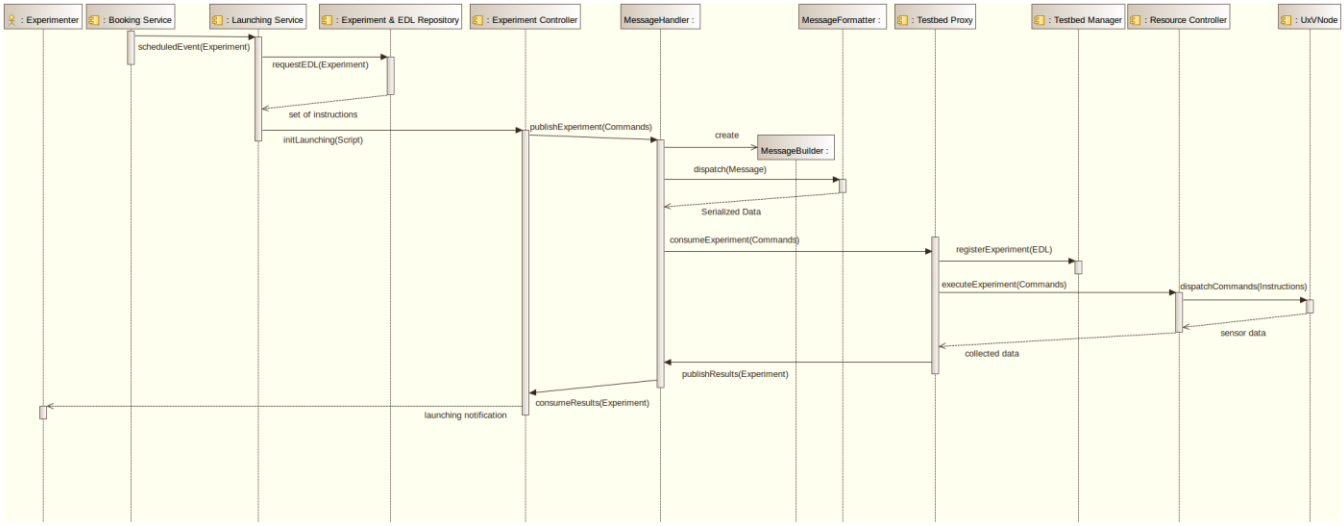

Figure 30: Message Bus - Launching execution notification

Running experiment visualization

1. The experimenter selects the resource to be tracked and evaluated making use of the Visualisation Tool.
2. The Visualisation Tool receives the resource information and forwards the data to the Visualisation Engine.
3. The Visualisation Engine launches the experiment request by sending the communication to the Message Bus.
4. The Message Bus forwards the instructions to the Testbed Proxy, which establishes the link with the requested resource. In order to perform this, it creates first the request for the resource node, serialises the message with the last tracking information and finally the MessageHandler dispatches the message to the subscribers.
5. The Testbed Proxy fetches the current sensor data of the resource and publishes the information in the Message Bus.
6. The Visualisation Engine consumes the updated experiment data from the Message Bus and brings backwards the resource information to the web Visualisation Tool.

7. The Visualisation Tool displays the current resource location and sensor data to the experimenter which requested it.

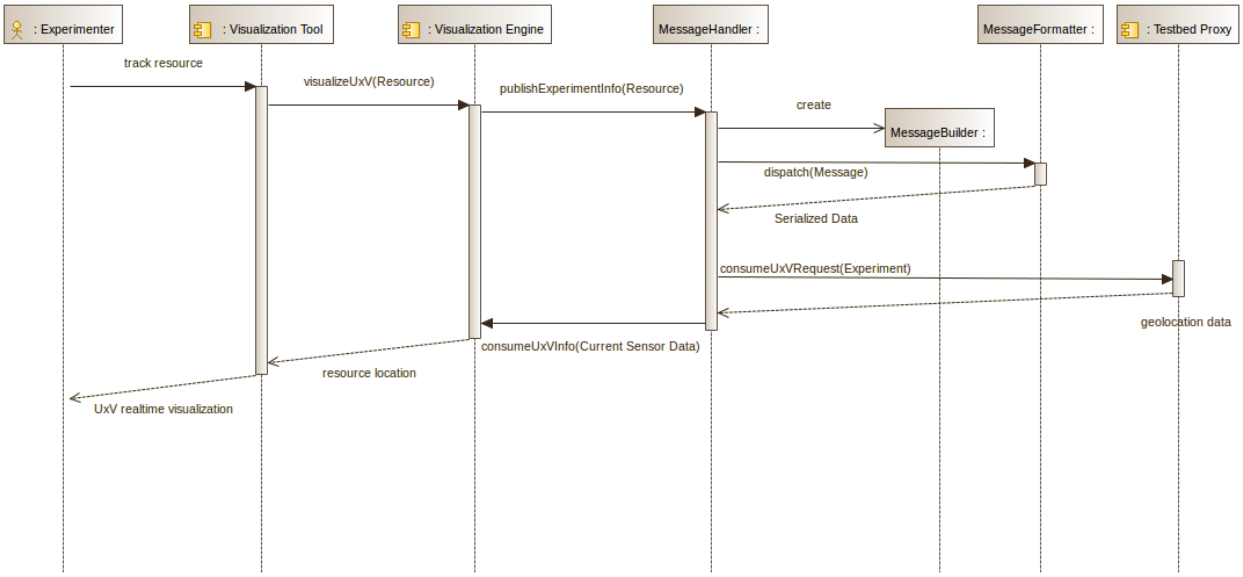

Figure 31: Message Bus - Running experiment visualization

Interactions and relationships with other components

The Message Bus component may interact with a multitude of different components, which are involved in communication actions across distinct application tiers. Message bus provides a loosely coupling mechanism for interaction and therefore the notion of a provided or requested interface does not strictly apply. Representative communication scenarios that illustrate use of the component include:

Provided Interfaces

- The Testbed Proxy Publishes information about measurements using the message broker for the communication with the Experiment Monitoring Tool and Testbed Manager components. It also tracks the information concerning the exact position of the UxVs while the test is running.
  Afterwards, it publishes the data for the Visualisation Tool through the Visualisation Engine in order to enable visualization of resources' location to the Web Portal.
- The Testbed Proxy gets measurements or any other relevant information about the status of a specific resource (UxV). Additionally, this latter fetches communication data between different UxVs, while an experiment is running.

- The Experiment Monitoring Tool obtains data of measurements and results published by the Experiment Controller component.
- The Visualisation Tool consumes information about the resources location (device tracks), for the final visualisation through the web portal interface.

3.2.3 Testbeds Directory Service

The Testbeds Directory Service is basically a registry service of the middleware tier, where all the integrated testbeds and resources accessible from the federated RAWFIE facilities are managed and listed.

Responsibilities

The main responsibilities of the Testbeds Directory Service are:

- Provide access to the information, contained in the corresponding Testbeds & Resources Repository, to other components.
- Provide the pointers to the different testbeds belonging to the RAWFIE federation.
- Manage the entries of the Testbeds & Resources Repository.
- Provide the available testbeds lists 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.
- Provide the testbeds and resources capabilities in terms of available technologies and corresponding tests.
- Execute queries to the Testbeds & Resources Repository, based on advanced searching capabilities through specific filters.
- Permits updates of the Testbeds & Resources Repository data based on resource updates received from each testbed

Operations and attributes

In order to analyse in depth the provided functionalities, interfaces and specification of the given component, class and sequence diagrams are provided.

With the aim to represent the structure of the Testbed Directory Service component, an abstract class diagram is provided, showing the associations, operations and responsibilities of the classes. The Directory Manager is in charge of processing all the incoming requests from the Web Portal by making use of the Resource Explorer Tool.

This is the responsible class to carry out the information retrieval as well as for the update or persistence requests of testbeds and/or resources, specifying an implementation of a web service interface with the available methods for the data store management. Basically, it creates instances of required testbeds and/or resources for manipulate them later on and produce the necessary input in order to build the queries.

Afterwards, the Directory Manager calls the Repository Handler which executes the proper queries against the Testbeds & Resources Repository, according to the request specified. As last step, the Testbeds Directory Service displays the requested information with the success criteria result by sending the data back through the web interface, managed by the Resource Explorer Tool.

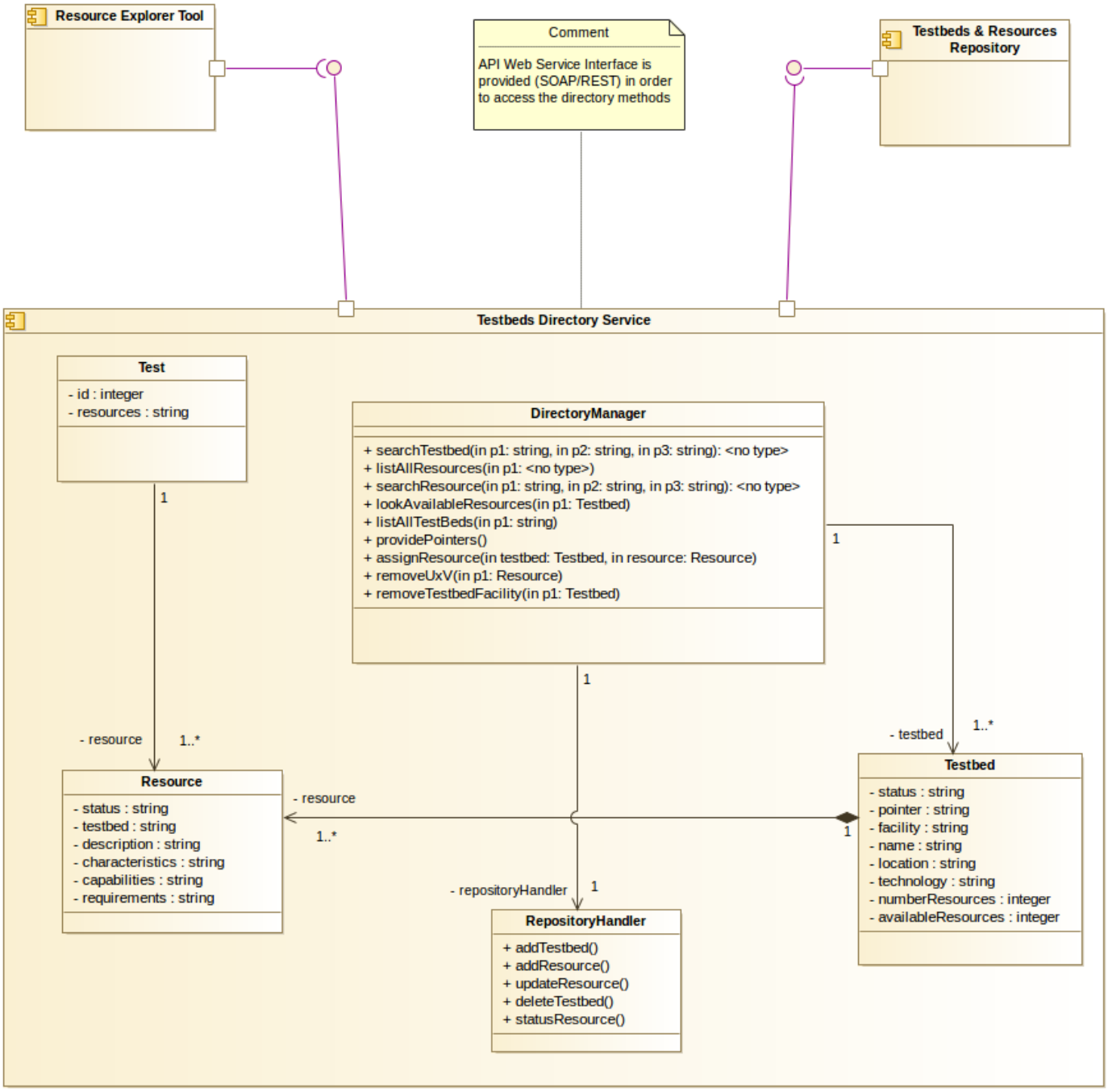

Figure 33: Testbeds Directory Service - High level class diagram

The relevant operational functionalities are explained and represented below with the help of appropriate sequence diagrams.

Search for an available resource

1. The Experimenter issues a search request by specifying the parameters relative to the specific resource information.
2. The Directory Manager process the search query by interacting with Testbeds and Resources instance objects. This means, creating and managing the necessary class stubs for retrieving the data from the database. The actual queries are forwarded to RepositoryHandler which acts as a delegate to the Testbeds and Resources Repository
3. The Repository Handler fetches the data from the Testbeds & Resources Repository and provides the results to the Directory Manager.
4. Finally the Directory Manager formats and serializes the data. Then, forwards the output to the Resource Explorer Tool, so that the Experimenter can visualise the available resources and their characteristics.

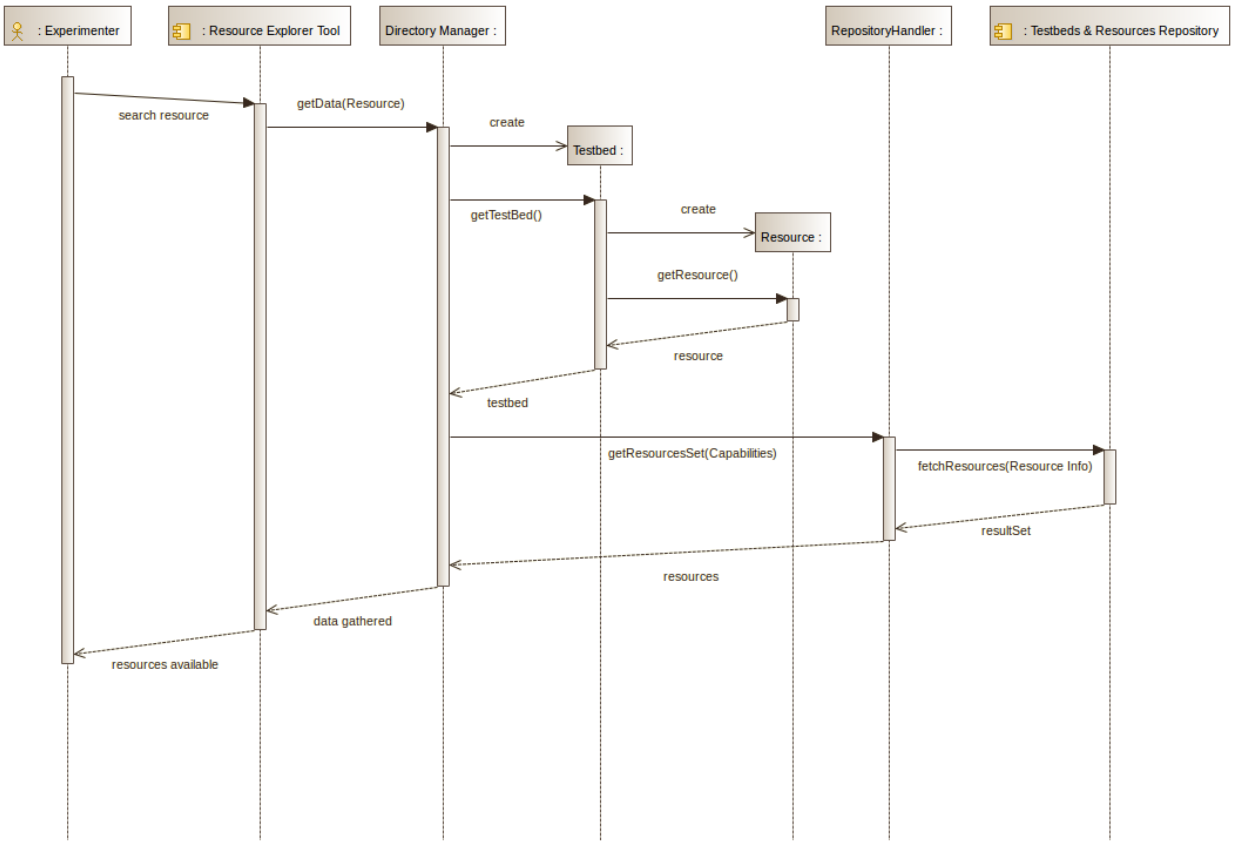

Figure 34: Testbeds Directory Service – Search for an available resource

Register a new Testbed in the platform

1. The Platform Administrator starts with the process of registering a new Testbed into the RAWFIE federation, prior its formal approval and compliance with specific regulations.

2. The Directory Manager receives the request concerning the new Testbed to be added, and proceeds with the creation of a new Testbed.
3. The RepositoryHandler obtains the information pertaining to the Testbed from the Directory Manager, and prepares the respective query for inserting a new entry.
4. The Repository Handler executes the query to the database and holds for the acknowledgement from the Testbeds and Resources Repository.
5. The Repository Handler triggers the return message backwards until the Administrator receives the feedback regarding the Testbed registration.

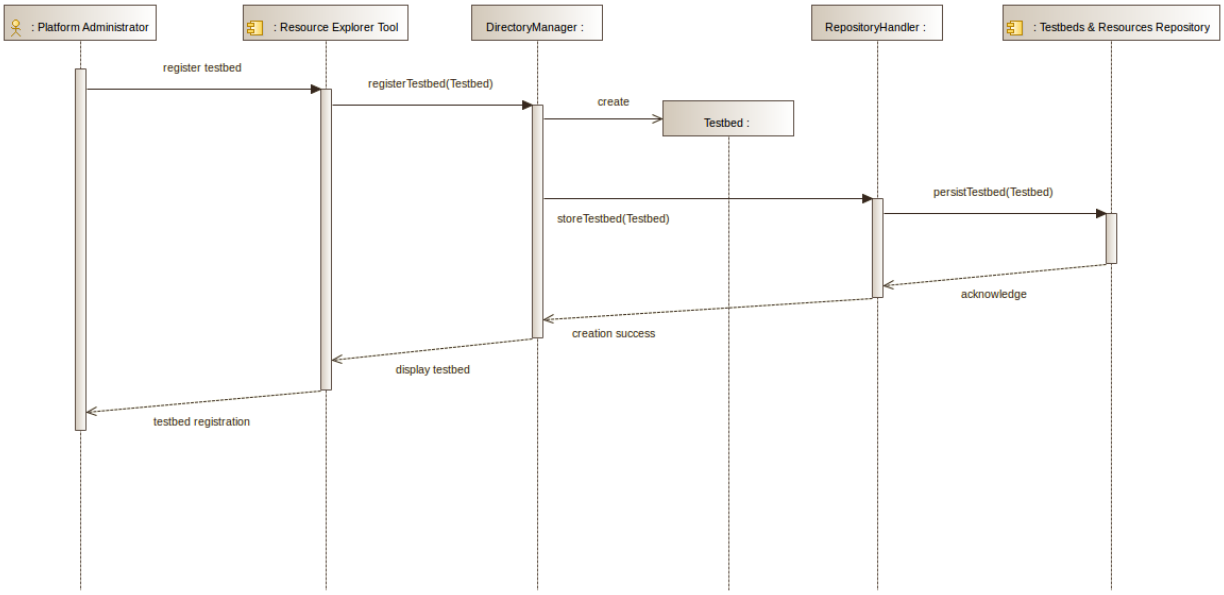

Figure 35: Testbeds Directory Service – Register a new testbed in the platform

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. The operator receives all the detailed information about the UxV node, obtained it respectively from the device manufacturer. Then through the Testbed Manager performs the registration request to the Testbeds Directory Service.
3. The Directory Manager receives the request and creates a new instance of a Testbed and Resource class objects.
4. The Repository Handler gets the objects with the specific attributes and executes the appropriate insert query. After this, it awaits for an acknowledgement from the Testbeds and Resources Repository.

5. The Repository Handler triggers the return message backwards till the Testbed Operator receives the feedback according to the Resource registration into the established Testbed Facility.

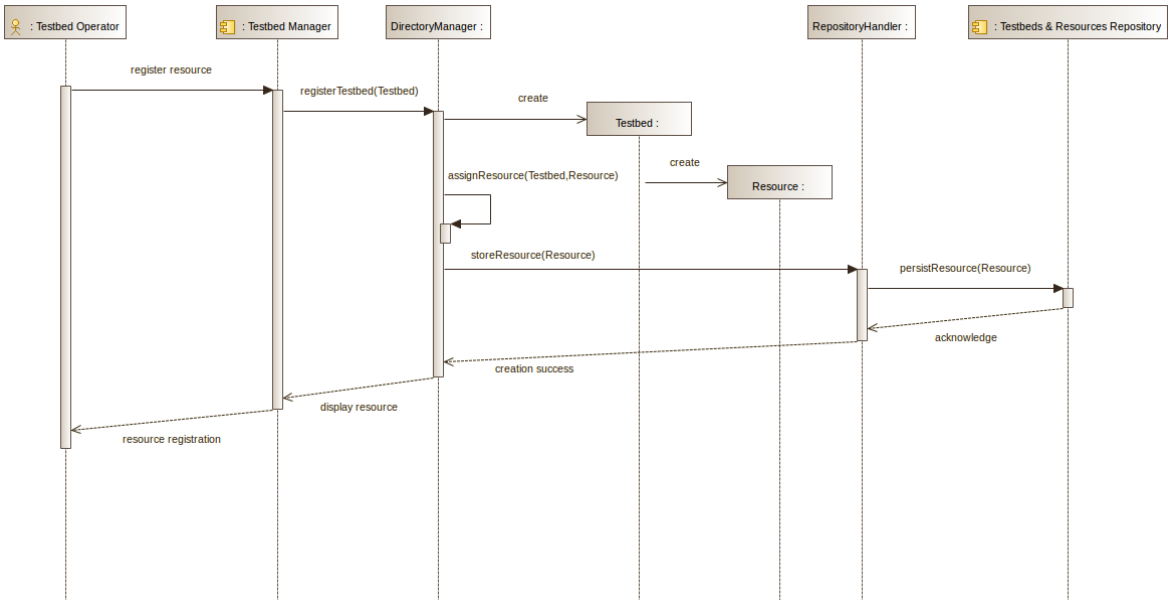

Figure 36: Testbeds Directory Service - Add a new UxV device into a Testbed facility

Interactions and relationships with other components

The Testbed Directory Service interacts with the Resource Explorer Tool and the Testbeds & Resources Repository for the data exchange representing the different testbeds and resources involved.

Provided Interfaces

- The Testbed Directory Service component API is invoked by the Resource Explorer tool in order to list the available testbeds and, for each testbed, the respective resources. This event occurs whenever an experimenter wants to retrieve certain information related to available testbeds and resources, using the respective front-end tool.

The Testbed Directory Service is also utilized y the Testbed Manager fot adding or updating UxV Resource data.**Required Interfaces**

- The Testbed Directory Service API will be in charge of executing the queries to the Testbeds and Resources Repository in order to administrate the database directory.

3.2.4 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.

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 32 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) undertake the responsibility of identifying errors and warnings. Cross link validation will be responsible to cover complex aspects of the experiments workflow. Finally, a generator will be responsible producing the final code that could be adopted by the remaining architecture.

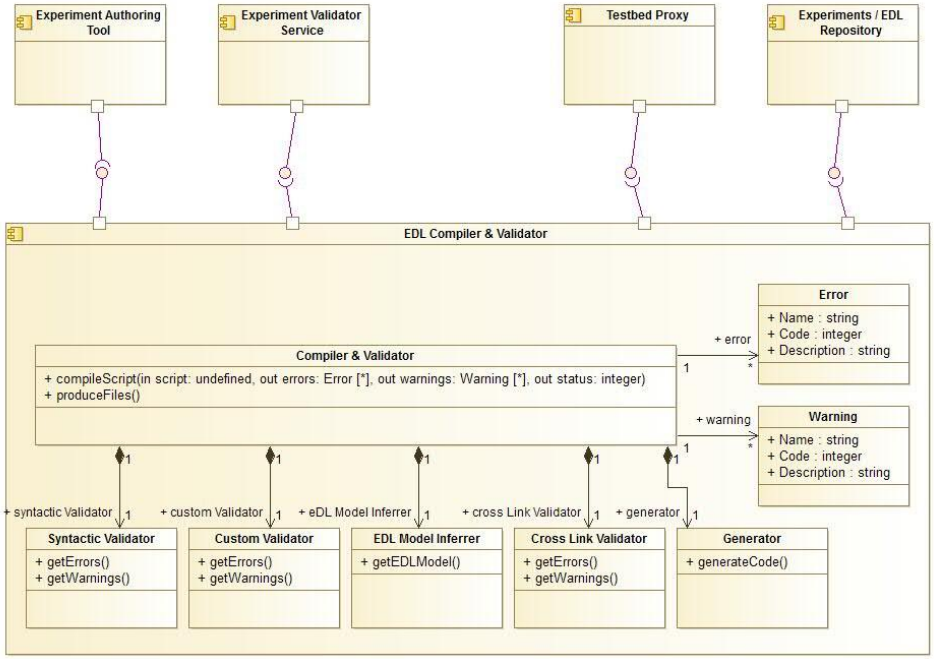

Figure 32: Experiment Compiler – High level class diagram

Interactions and relationships with other components

The interaction of the ECV with other components of the RAWFIE architecture is given in Figure 33. The Experiment Authoring Tool gives the experimenters the opportunity to trigger the ECV. Afterwards, the ECV communicates with the repository to retrieve all the necessary information related to the EDL model and the examined experiment. The final step, after successive iterations in correcting the EDL script, is the generation of the appropriate files and code.

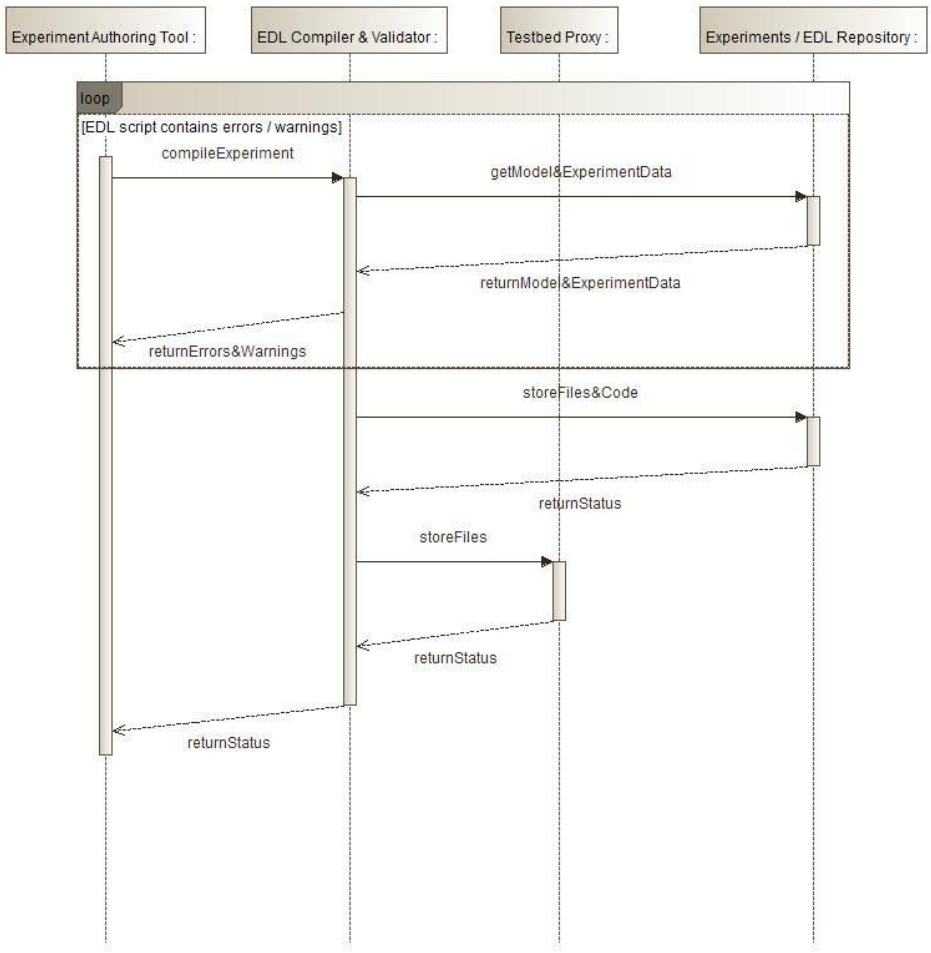

Figure 33: Experiment Compiler - Sequence diagram.

3.2.5 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 efficiently be executed in the selected testbed. Cross experiments validation will be performed accompanied by qualitative characteristics of an experiment. For instance, the EVS, based on each experiment workflow, will retain security and qualitative issues. Communication between nodes will be secured as well as collision avoidance and qualitative control activities.

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.

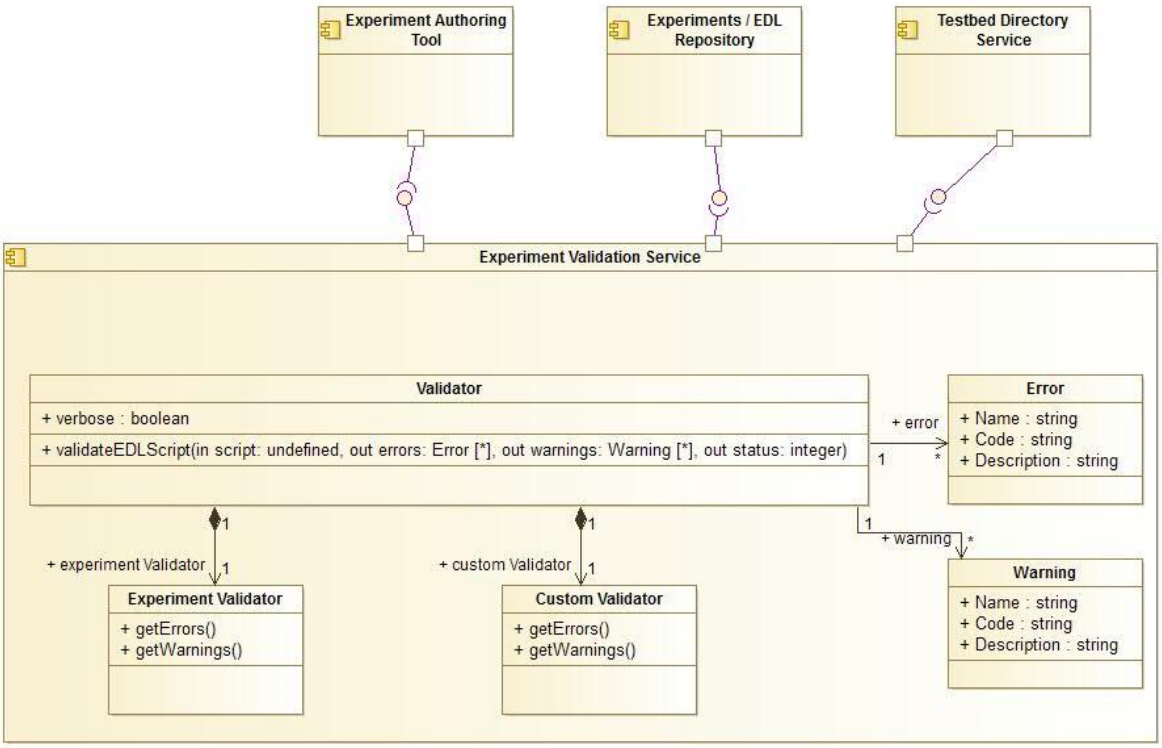

Figure 34: Experiment Validator – High level class diagram

Interactions and relationships with other components

In Figure 35, we see the interaction of the EVS with other components of the RAWFIE architecture. Mainly, the EVS is triggered by the Experiment Authoring Tool and, accordingly, it retrieves the necessary data related to the testbed and the resources from

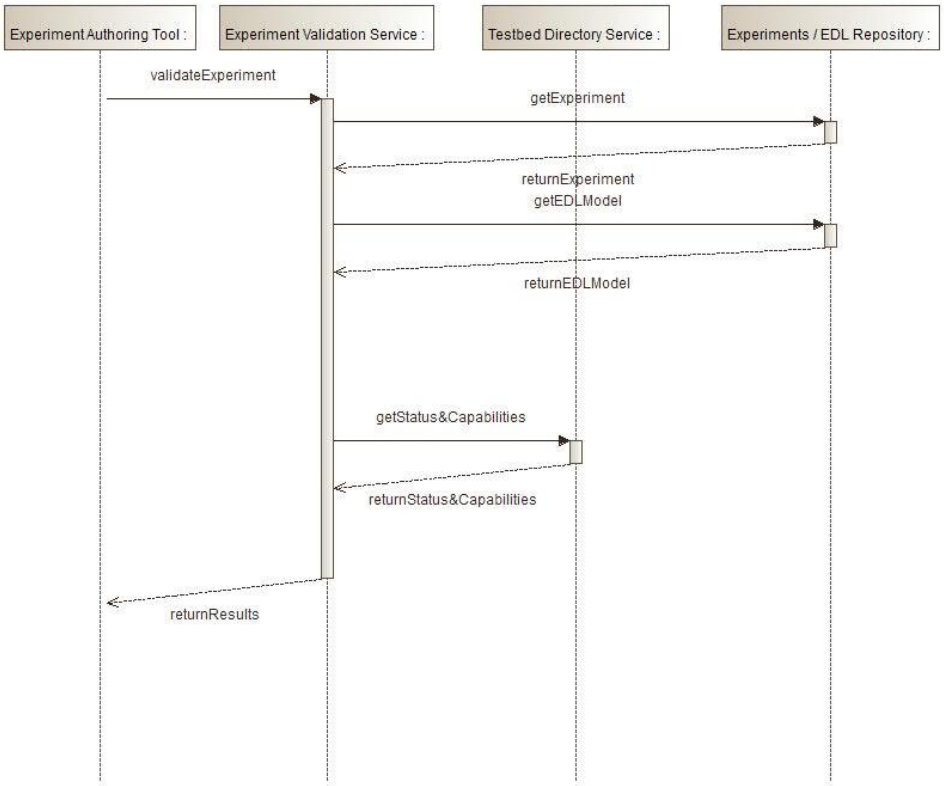

Figure 35: Experiment Validator - sequence diagram.

3.2.6 Users & Rights Service

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

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 will be a LDAP server. The LDAP server will store 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 use, 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.

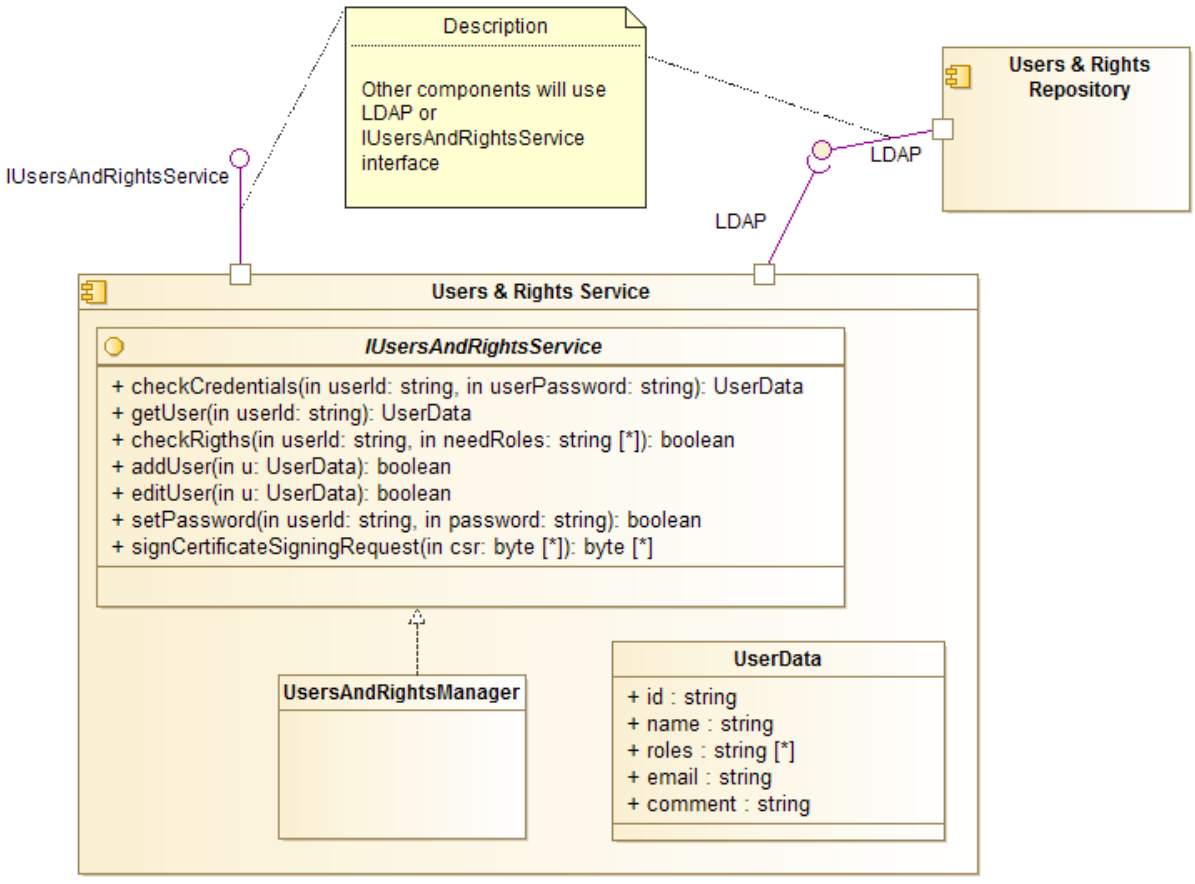

Figure 36: Users & Rights Service - High level class diagram

D4.1 contains already several sequence diagrams that show how the authentication and authorization process will be done. They are still valid for the current development, so no further sequence diagrams will be provided for this component.

Interactions and relationships with other components

Provided interfaces

- Users & Rights Services interface: 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

3.2.7 Booking Service

The Booking Service manages bookings of resources (testbeds and UxVs)

Responsibilities

The main responsibilities of the Booking Service are:

- Store and load bookings from the Bookings Repository
- Check for conflicts in bookings before accepting them
- Coordinate use of testbed resources among experimenters
-

Operations and attributes

The Booking Service interface (IBookingService), which is used by the Booking Tool, provides all necessary methods to manage the bookings including add, edit and delete operations. 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 booking 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. 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.

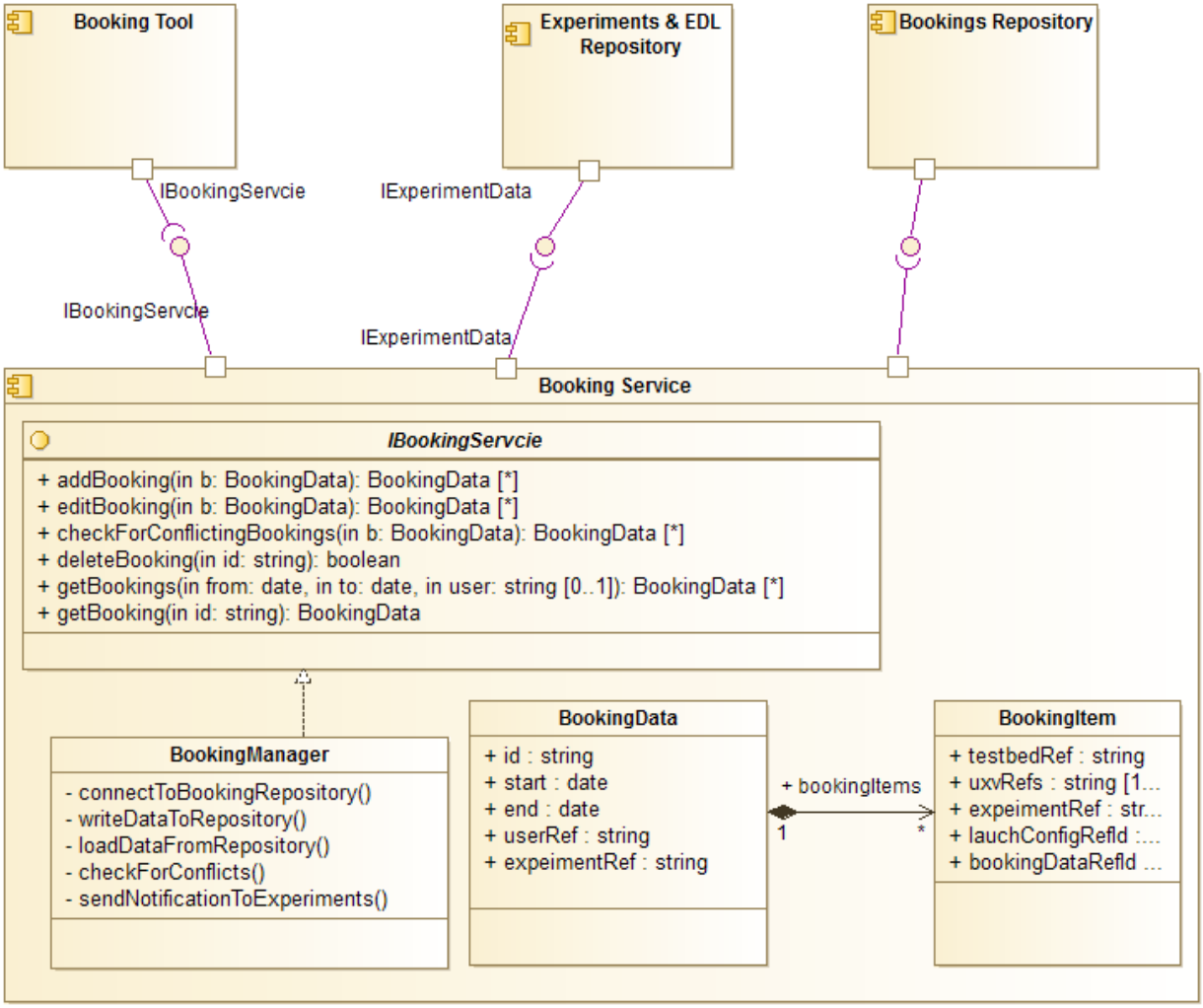

Figure 37: Booking Service - High level class diagram

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 (Booking Service)
3. The BookingManager loads the data from the Bookings Repository and returns the processed data to the Booking Tool
4. The Booking Tool creates the appropriate calendar view and shows it to the user.

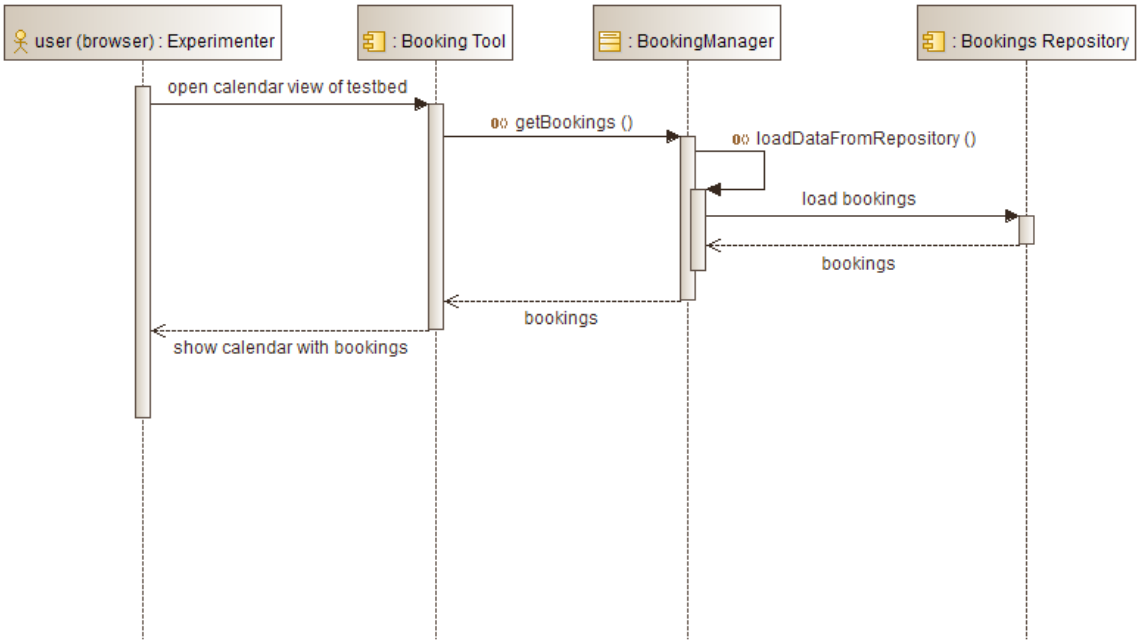

Figure 38: Booking Service – View bookings of a testbed

Store a booking

1. The experimenter submits the form with the booking details to the Booking Tool
2. The Booking Tool calls the addBooking method of the BookingManager (Booking Service)
3. The BookingManager reads all bookings of the given resources in the given timespan from the Bookings Repository
4. If there are any conflicting booking, they are returned to the Booking Tool, which shows them to the user
5. If there are no conflicting booking, the BookingManager stores the new booking in the Bookings Repository and return a success message to the Booking Tool, which shows it to the user

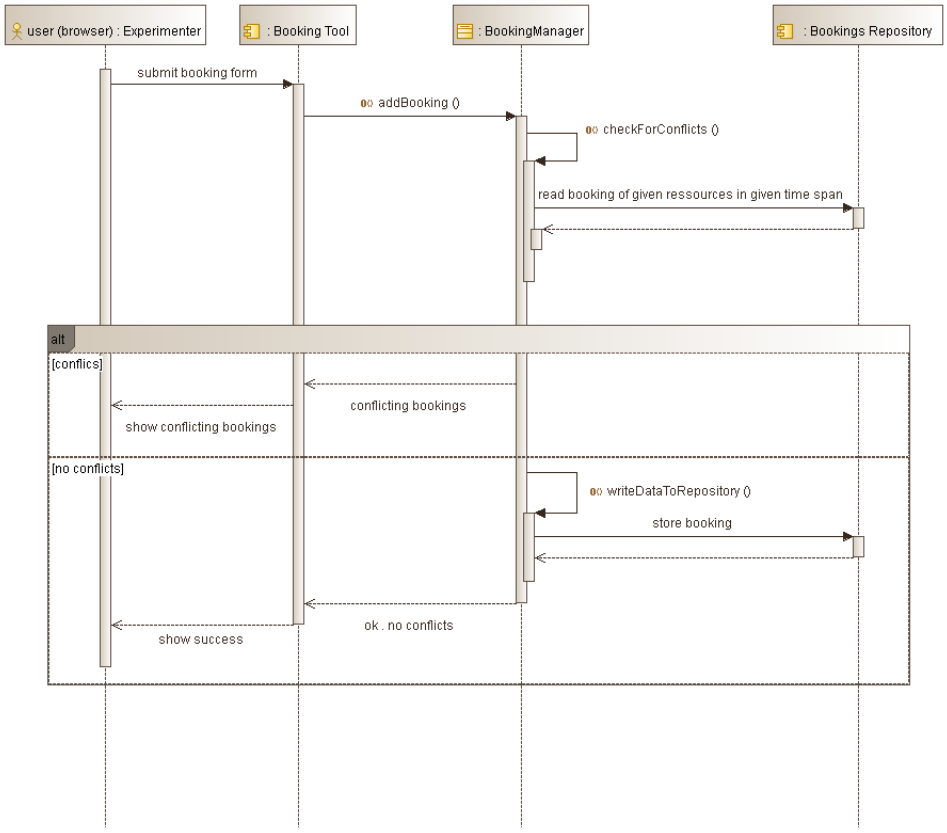

Figure 39: Booking Service – Store a booking

Interactions and relationships with other components

Provided interfaces:

- Booking Service Interface (IBS): This interface is manly used by the Booking Tool that provides a GUI to manage the bookings.

Required interfaces:

- Bookings Repository: The booking service loads and store its raw data in the Bookings Repository (direct database connection)
- Experiment & EDL Repository: Maintain references to experiments, e.g. when a booking is deleted.

3.2.8 Launching Service

The LS schedules and launches executions of the experiments together with the assigned booked resources. The LS is responsible for scheduling the execution of experiments. It supports short term and long term launching.

Responsibilities

The main responsibilities of the LS are:

- Manage experiment executions based on bookings from experimenters.
- Initiate the execution of an experiment or set of experiments in real-time, or according to an experiment schedule.
- Schedule the sequential or parallel execution of experiments.
- Supports real-time execution of predefined and pre-approved experiments.

Operations and attributes

The LS performs short- and long-term launching. The LS contains two methods one for each launching type. The first method receives the experiment ID that the experimenter wants to execute. The second method initiates the long-term launching through the communication with other components of the system like the bookings repository. In Figure 40, we see the class diagram of the LS while Figure 41 presents the sequence diagram for short- and long-term launching.

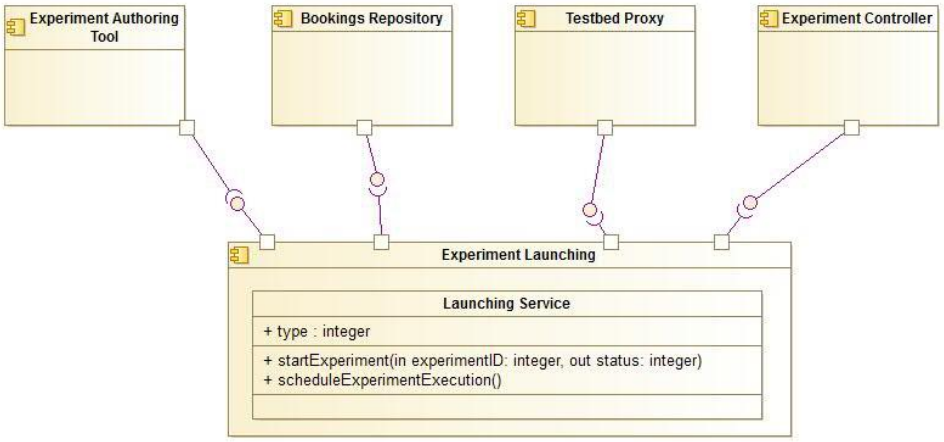

Figure 40: Launching service – High level class diagram.

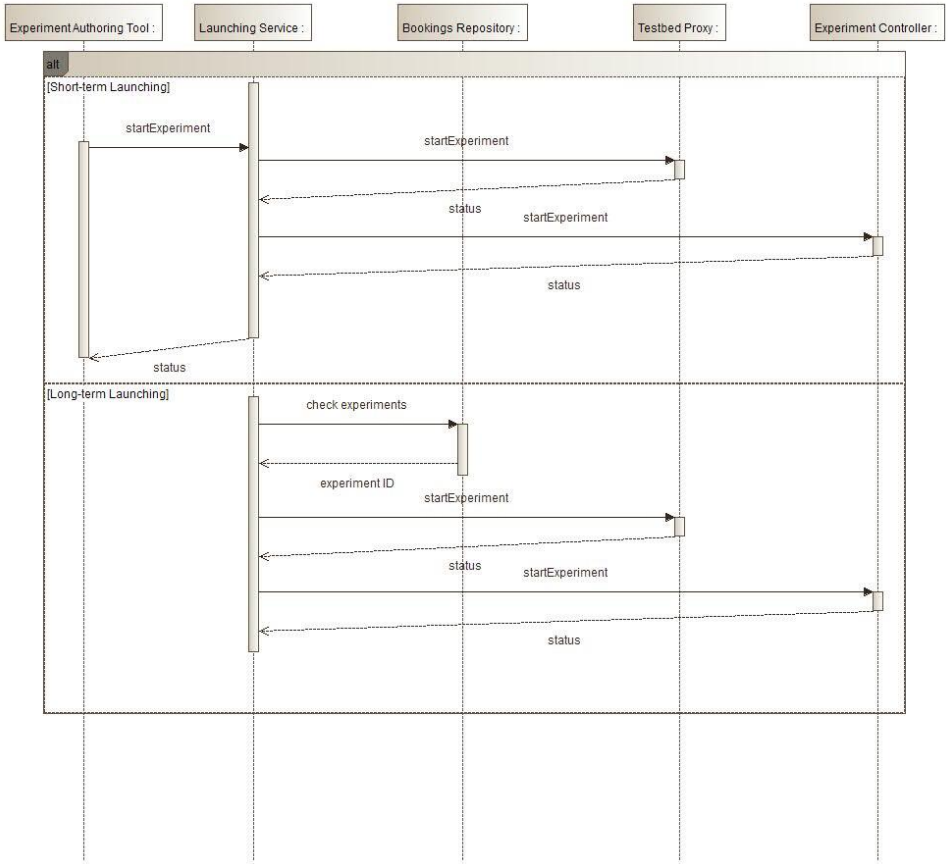

Figure 41: Experiment launching - Sequence diagram.

Interactions and relationships with other components

The launching service is initiated by the experiment authoring tool where the experimenters could select the experiment they want to execute. This process corresponds to the short-term launching. In the long term launching mode the LS operates as a scheduler, periodically checking, with the help of the bookings repository, the experiments that should be executed and initiating them based on their start time. In both cases, the LS informs the Testbed Proxy and the Experiment Controller about the initiation of the experiment's execution.

3.2.9 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.

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.

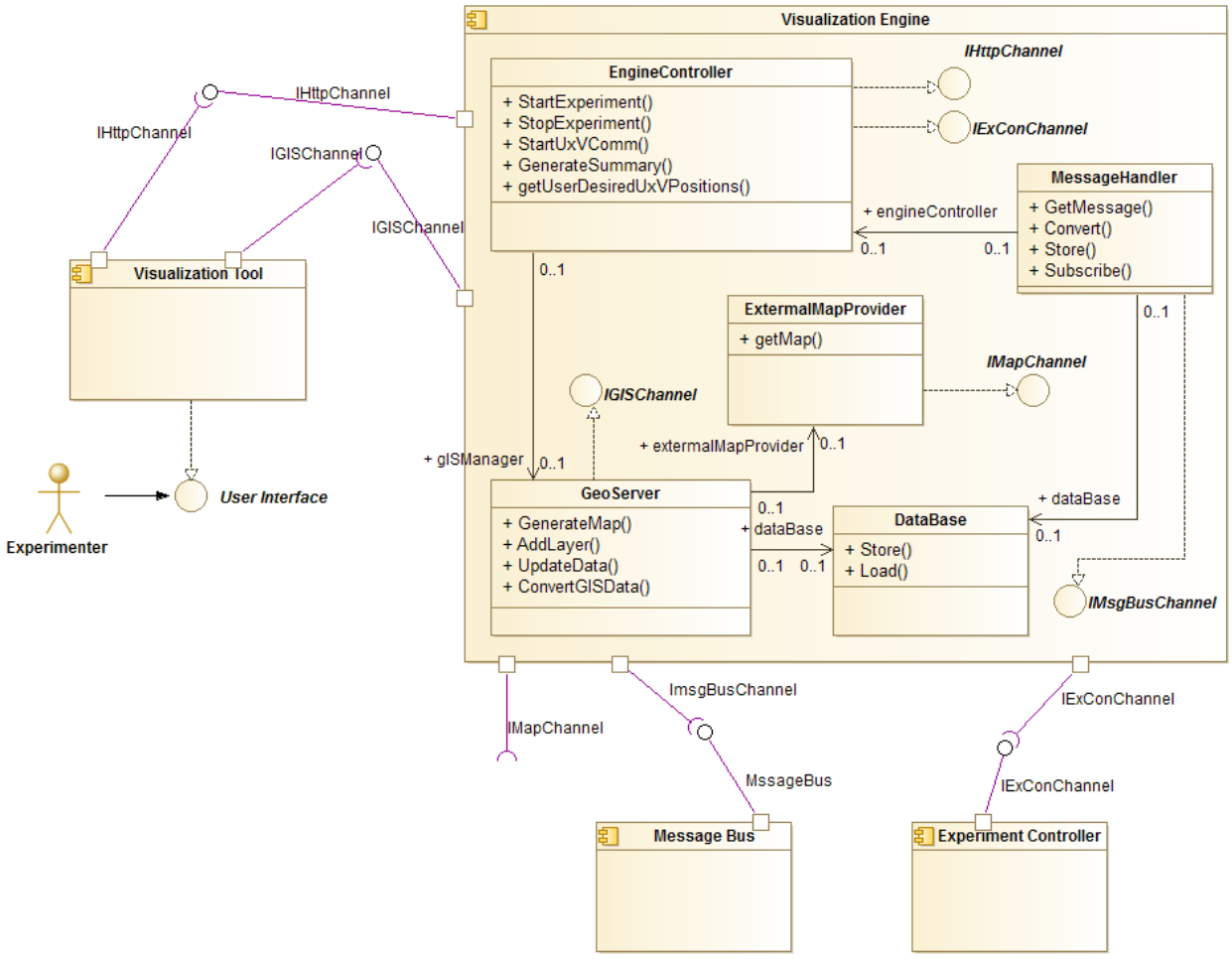

Figure 42: Visualisation Engine - High level class diagram

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 http channel
- 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 http channel

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

Start an experiment:

1.
1. The Experiment Controller sends a message to the Visualization Engine Controller to inform the VE that the experiment is started
2. If the VT is not started, then the VE does not do anything and waits for the experimenter to start the VT.
3. If the VT is started then the Engine Controller sends two requests:
    a. To the GISServer and the Database for triggering map and layer generation. The GISServer fetches the map from the external provider and the additional layers from the database and sends them over the gis channel to the VT
    b. To the Message Handler, which subscribes to the Message Bus for UxV related messages
4. The data is adjusted by the Data Adapter for the map renderers and is passed to the Renderer for plotting

The browser windows is properly initialized and set for the VT to show the experiment's run.

UxV update:

1. Upon receiving updates from the Message Bus, the Message Handler converts the message and the sensor data are sent to the VT while the UxV position is sent to the GISServer
2. The GISServer prepares the appropriate layer together with the Database and sends the GIS information to the VT
3. The Visualisation Manager receives the sensor data, decides which widgets need to be updates/shown/hidden with the GUI and the Renderer plots them
4. The updated layers are converted by the Data Adapter if necessary and are plotted by the renderer by updating the existing layers or by adding an additional layer

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 Engine Controller receives the update of the experiment and forwards this to the VT over the http channel
2. The Renderer requests from the GUI the appropriate widget upon reception of the status update
3. The widget with the proper status information is plotted accordingly

The experimenter is informed on the update of the experiment.

The experimenter updates one of the layers:

1. The experimenter shows/hides/changes one of the layers. This request is sent to the VE for generating a new layer
2. The new layer is generated by the GISServer and sent back to the VT
3. The new layer is plotted by the Renderer.

The user can then show/hide different layers like covered area by different UxVs, past positions and future waypoints, thermal map of the area and others.

Stop an experiment:

1. The request to stop the experiment is received from the Experiment Controller. The GISServer generates a layer with the summary of the experiment like distance, covered area, start and stop positions of the UxVs and others.
2. The summary is loaded onto a widget by the GUI and plotted by the Renderer. The layer is plotted on top of the current view.

In that way the experimenter will be able to see the complete run of the experiment, the different geo information for the UxV and in a widget statistics will be presented.

Replay an experiment:

1. The experimenter chooses the experiment that should be replayed. This request is sent to the VE, which retrieves information about the experiment from the Experiment Controller. This Data is stored in the Database for generating layers for the experiment.
2. The Engine Controller sends a request to the GISServer and the Database for triggering map and layer generation. The GISServer fetches the map from the external provider and the additional layers from the database and sends them over the gis channel to the VT
3. The data is adjusted by the Data Adapter for the map renderers and is passed to the Renderer for plotting

By replaying any of the experiments, the user can gather or check data for an experiment at a convenient time after the experiment finished.

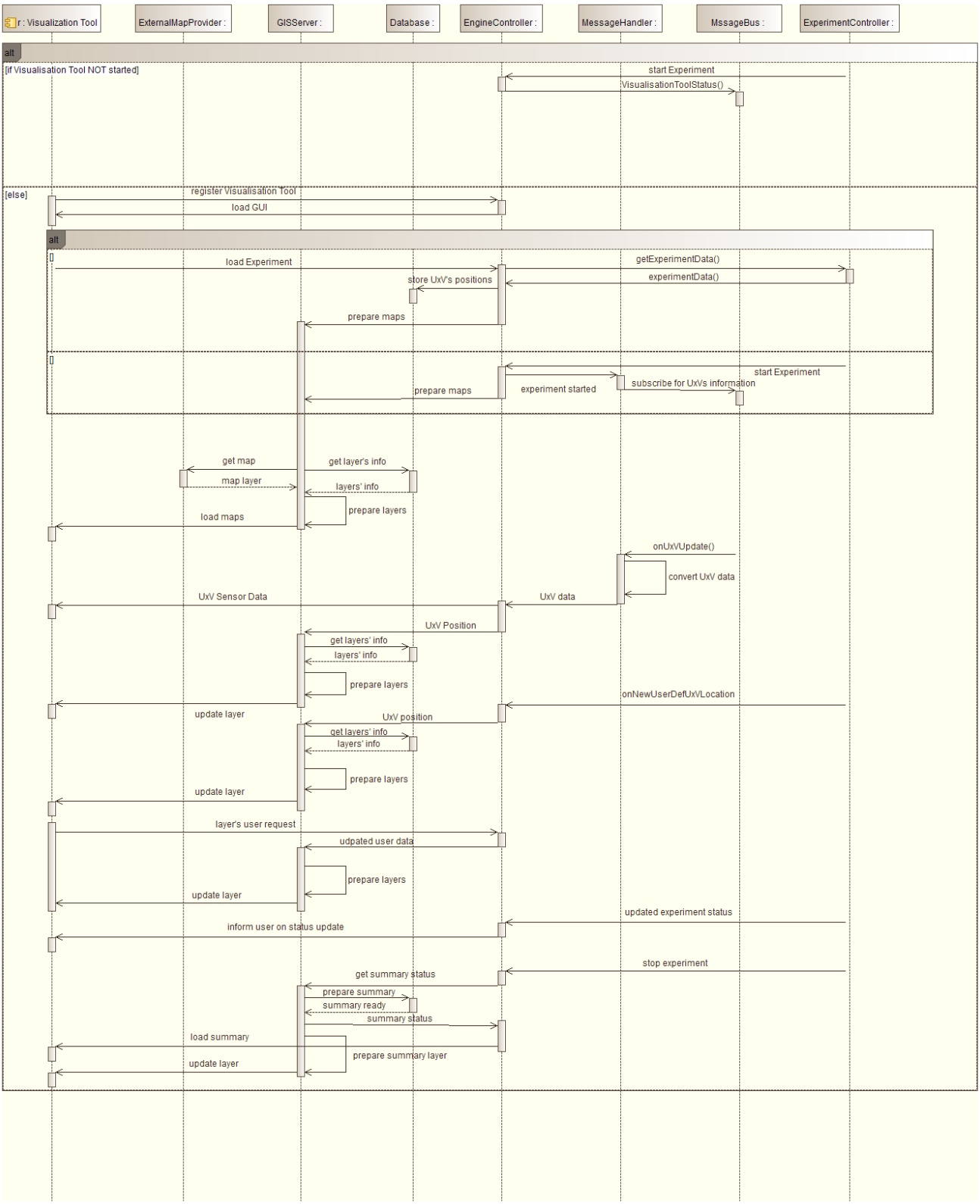

Figure 43: Visualisation Engine - Sequence diagram

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 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 http interface and the geo information data is sent over the GIS interface.
- The http 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.

3.3 3. Testbed control, monitoring and analysis components

3.3.1 Description

This subsection describes the Testbed Control, monitoring and analysis components. The Data Analysis Engine enables the execution of data processing jobs by sending requests to the processing engine. 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 System Monitoring Service shows the status and the readiness of the various RAWFIE services.

The main components of the navigation system is the Experiment Controller and the Resource Controller which ensures the safe and accurate guidance of the UxVs based on the user's preferences.

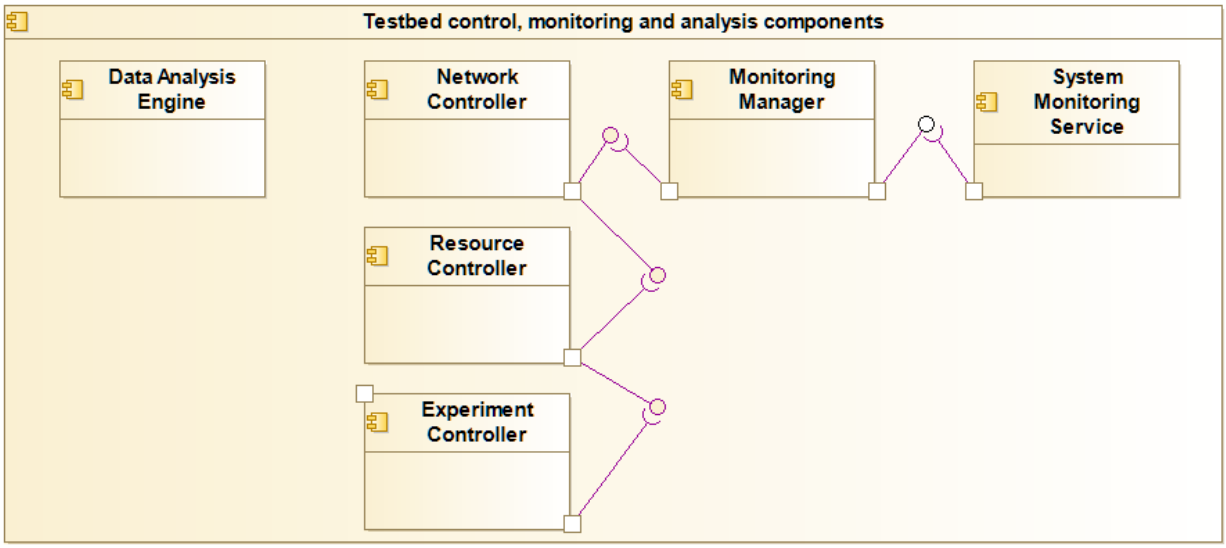

Figure 44: Testbed control, analysis and monitoring– High level Components Diagram

3.3.2 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.

Responsibilities

The main responsibilities of the Experiment Controller are:

- Transfers the instructions from the launching service to the Testbed Proxy and subsequently 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 Testbed Proxy and subsequently to the Resource Controller
- Transfer the data from the Testbed Proxy and from the Resource Controller to the Experiment Monitoring Tool

Operations and attributes

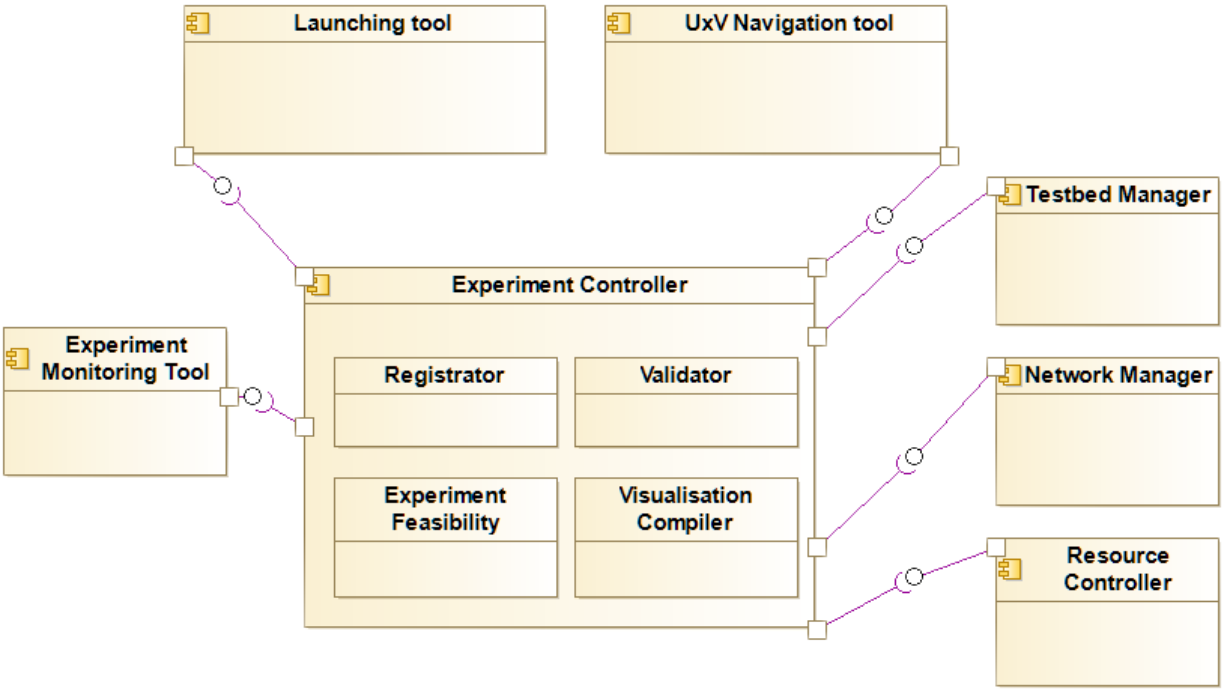

Figure 45: Experiment Controller - High level class diagram

Registers an Experiment

1. Register Class interacts with the Testbed Manager as to register the testbed to the middle tier. Additionally, the same class triggers the Network Manager for the provision of the network connections during the experiment between the nodes

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 Testbed Proxy and to the Resource Controller.

1. Validator Class validates the format of the provided JSON file
2. Transfers the file to the Testbed Proxy

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

1. Validator Class validates the format of the provided JSON file
2. Transfers the file to the Testbed Proxy

Transmits Back to the Experiment Monitoring Tool information regarding the experiment

1. Resource Controller and Testbed proxy returns to the Experiment Controller the actual position of the vehicles together with their sensor's measurements.
2. Visualisation Compiler translates these data into a compatible with the Experiment monitoring tool format.
3. Transmit these data to the Experiment Monitoring Tool

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 to the Testbed Proxy.
3. Informs the Experiment Monitoring tool.

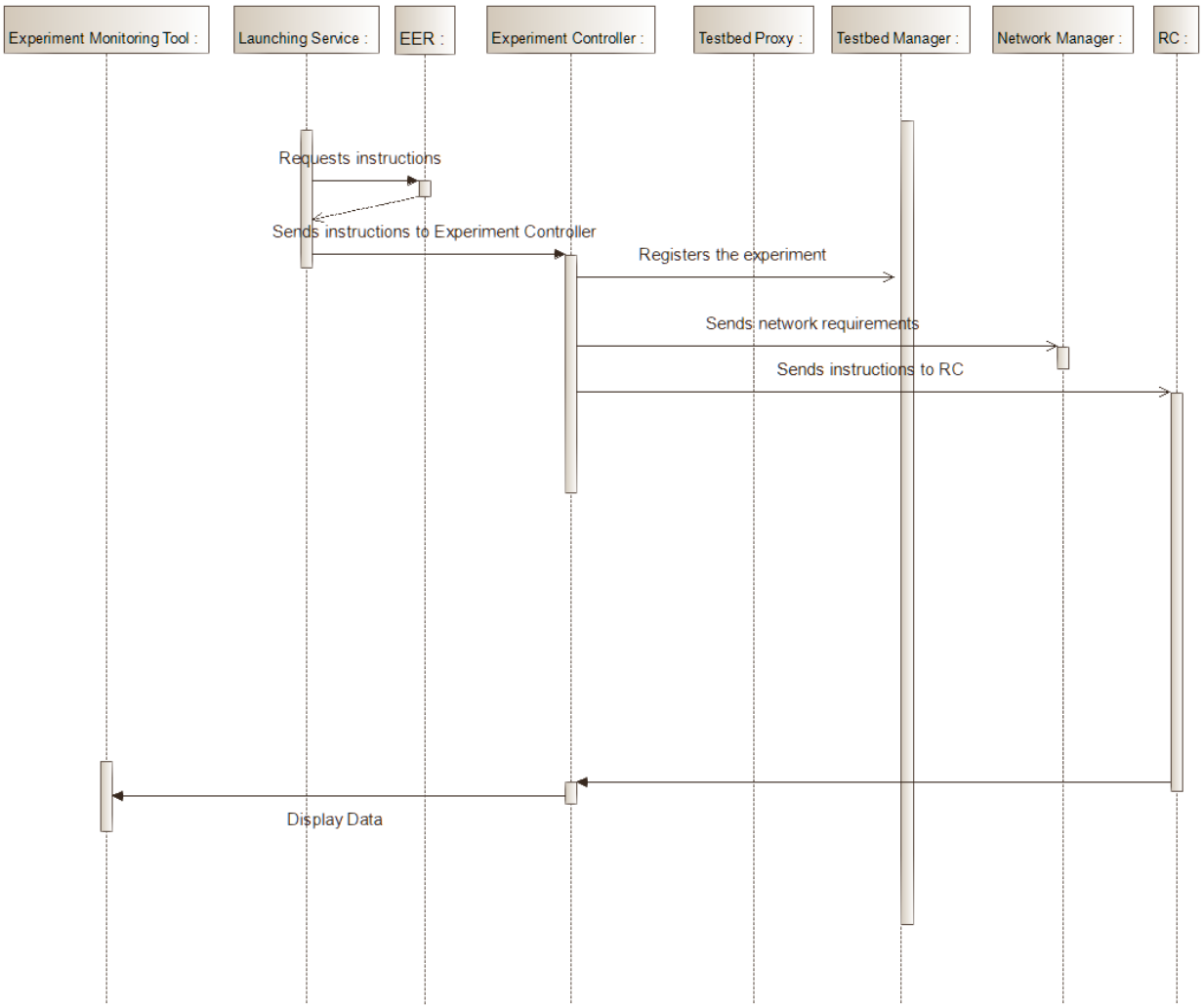

Figure 46: Experiment Controller - Sequence diagram

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 .

3.3.3 Data Analysis Engine

The Data Analysis Engine will be responsible for the creation of analytics jobs that are prescribed by the user from the GUI web interface. It will also handle model creation, pipeline management (of operations) as well as ensure correct information is relayed to the Measurements, Results & Status database.

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

Analysis of the provided functionalities, including the design:

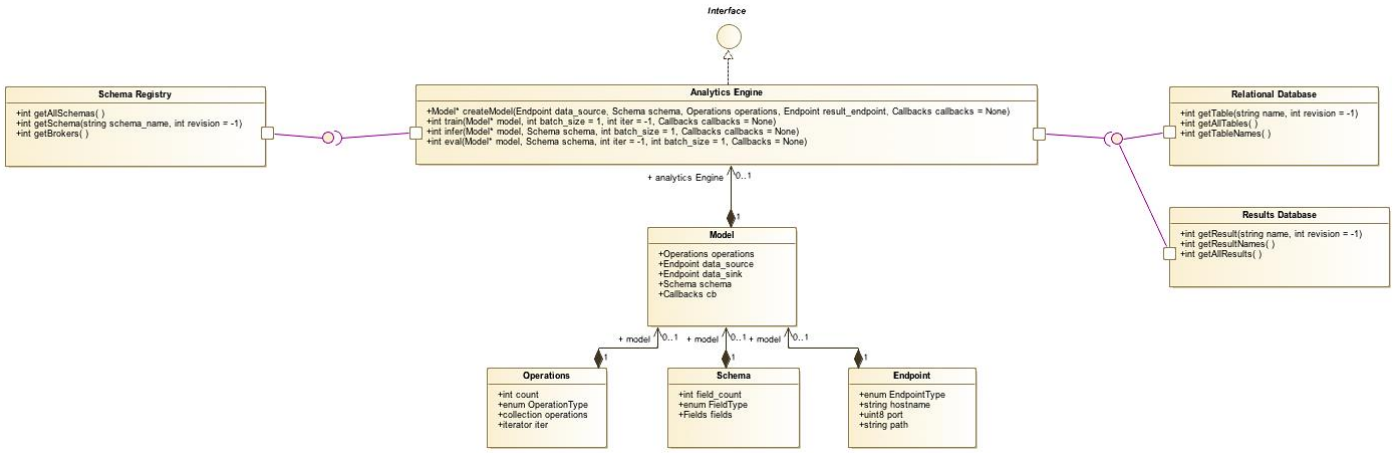

Figure 47: Data Analysis Engine - High level class diagram

.

- Sequence diagram provided in section 3.1.9

Interactions and relationships with other components

The Data Analytics Engine will communicate with :

1. Schema registry [Read Only]
2. Results database [Read Only]
3. Relational database [Read Only]
4. Web Interface (GUI) [Read/Write]
5. Compute Engine [Read/Write]

3.3.4 System Monitoring Service

The System Monitoring Server 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.

The System Monitoring Service will be realized by configuring and extending an existing monitoring solution like Nagios [2] or a fork of it like Icinga [3]. Nagios is open source and a de-facto standard software for system monitoring. So the described classes and function are more or less placeholders for the functionalities of the used underlying monitoring solution.

To get detail health status information from all RAWFIE components, if may be necessary to write special plugins for the System Monitoring Service that collect this information. Plugins can be:

- local: they run on the same server like the System Monitoring Service and collect status information by evaluation responses of special requests that are sent to the monitored components
- remote: they are collocated to the monitored system or even directly integrated in it. Since remote plugins have direct access to the resources of the monitored component, they can collect more detailed status data.

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
- Load health status data from the Monitoring Manager (via and special plugin)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

System monitoring software like Nagios [2] 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. These standard functionalities are summarized as "pingServer()" in Figure 48.

"executePlugins()" summarizes the execution of all configured plugins that collect special status information from the RAWFIE components.

"sendAlarm()" is executed when servers or services are not responding or if the defined thresholds of performance indicators are exceeded. A notification (e.g. an email) is sent to the RAWFIE System Administrator.

The interface "ISystemMonitoring" is used by the System Monitoring Tool (3.1.3) to display the data on a web page. Many monitoring solution support the Livestatus API [4], which then will also be part of the interface.

Monitoring plugins (local and remote) are called by the SystemMonitoringServer with a command line string that defines e.g. thresholds of performance indicators. The plugins check the metrics and return a human readable message (to be displayed in the System Monitoring Tool) and status code (integer) that is evaluated by the SystemMonitoringServer.

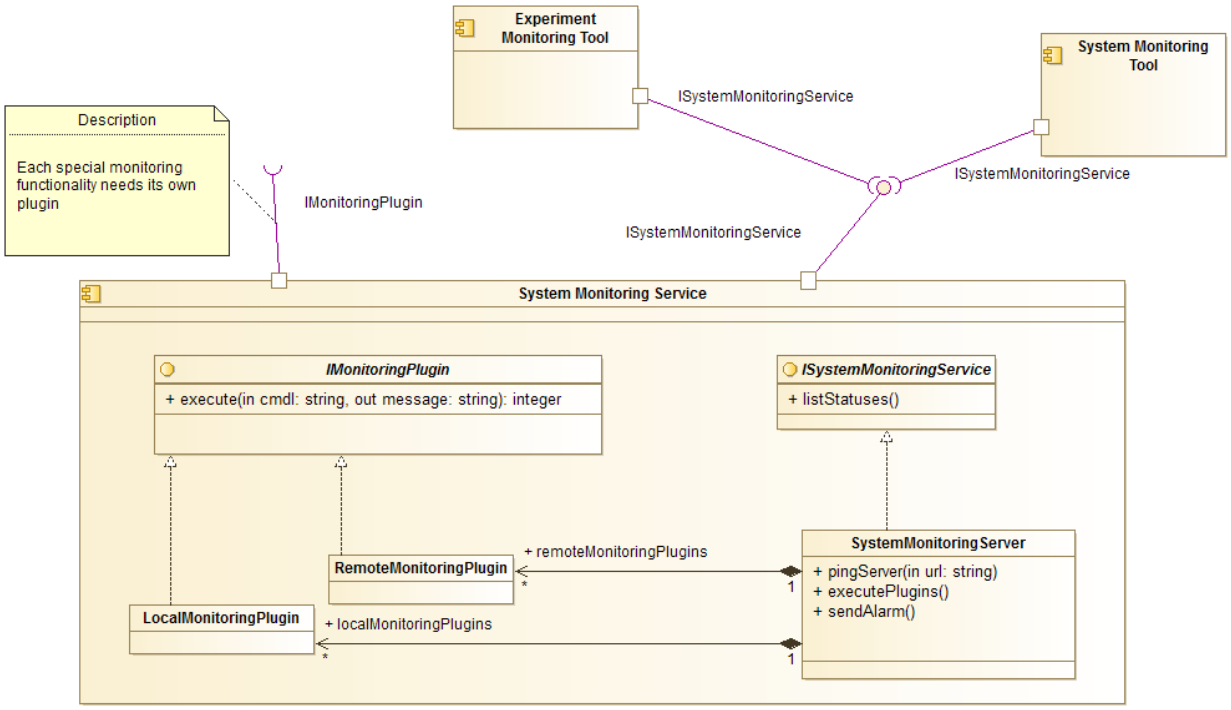

Figure 48: System Monitoring Service - High level class diagram

The interactions between the System Monitoring Service and the monitored components are displayed in the sequence diagram in Figure 49. If some errors were detected, an alert is sent.

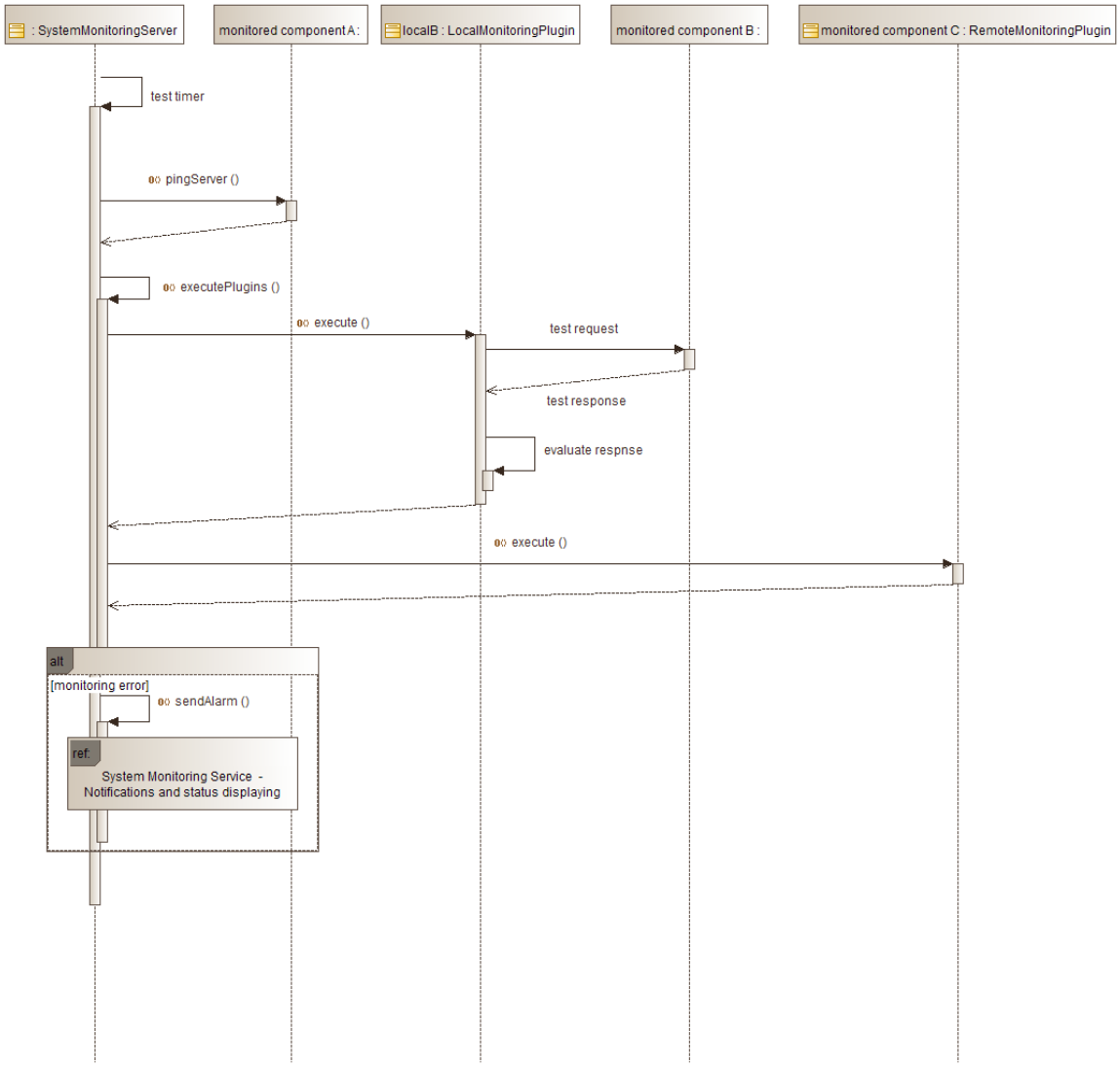

Figure 49: System Monitoring Service – Monitoring sequence diagram

Figure 50 shows the actions when an alert is raised and the users of the system want to know the system health state: When "sendAlert()" is called, the responsible RAWFIE Platform Administrators are notified (e.g. via email). The administrator then opens the SystemMonitoringPage to get detailed information about the problem. Also an Experimenter has noticed that something is not working and opens the StatusDashboardPage to see which components cause the problem.

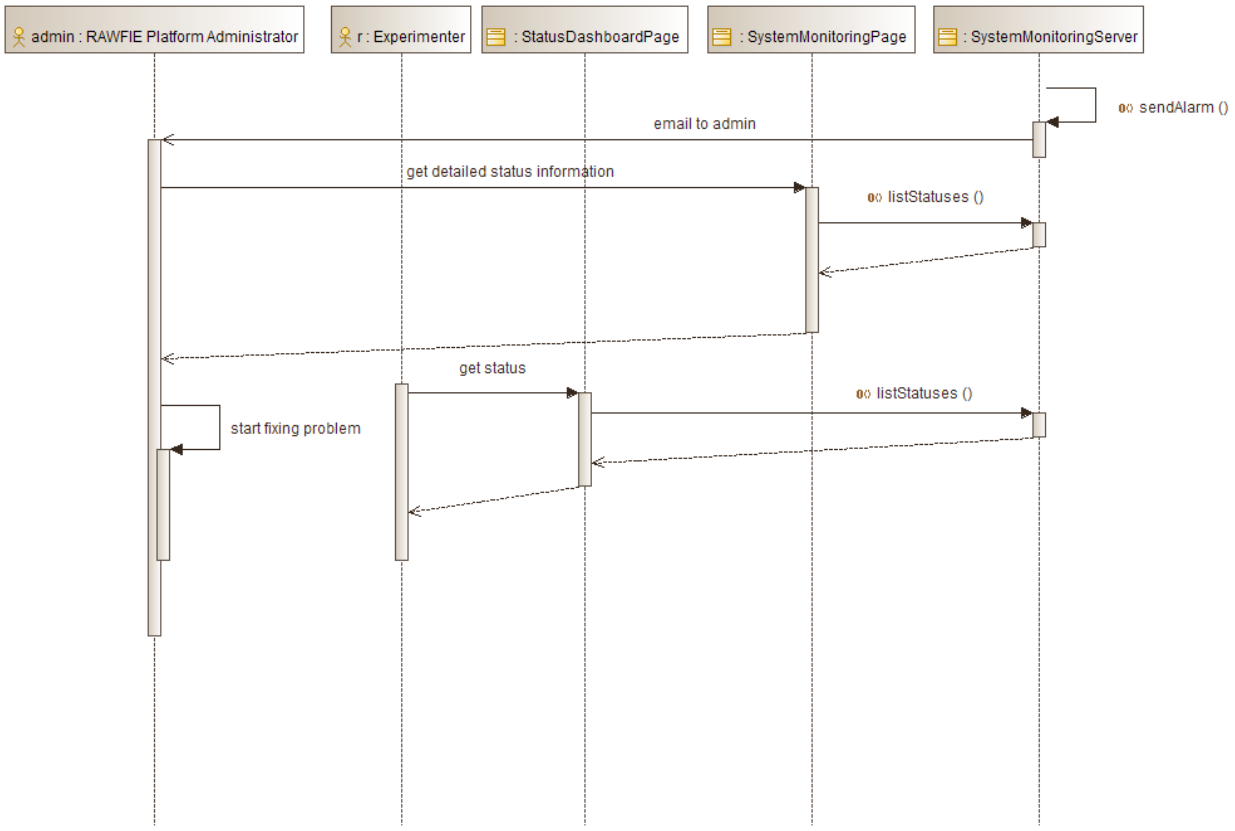

Figure 50: System Monitoring Service – Notification and status displaying sequence diagram

Interactions and relationships with other components

Provided interfaces

- System Monitoring Service:
  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:

- IMonitoringPlugin
  All important components of the RAWFIE system are monitored via standard procedures or via special plugins.

3.3.5 Monitoring Manager

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

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 Figure 51. 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.

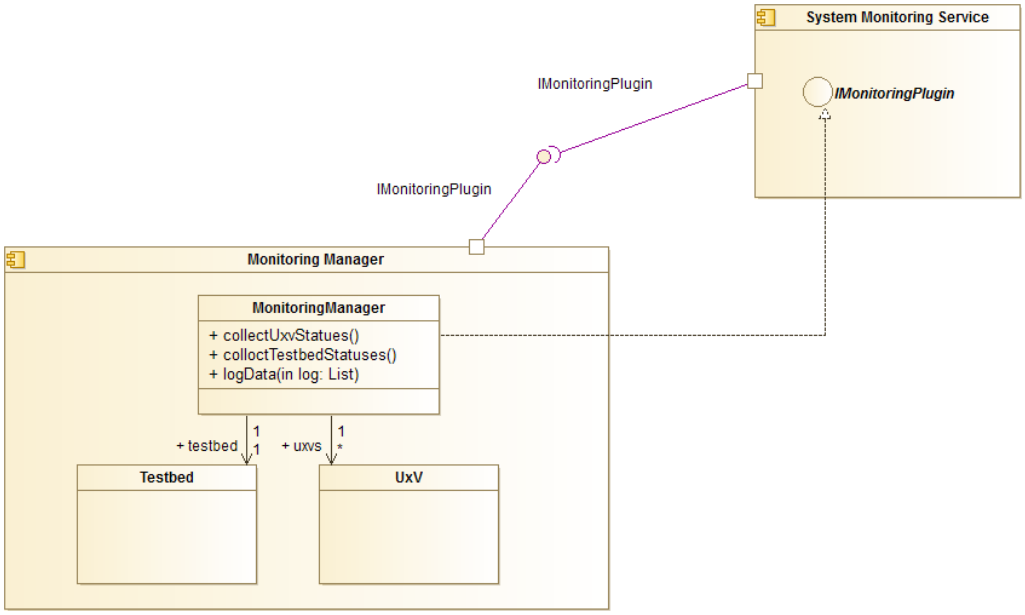

Figure 51: Monitoring Manager – High level class diagram

The interactions of the Monitoring Manager are presented in the sequence diagram of Figure 52 Periodically it connections to the UxVs and the testbed to collect 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.

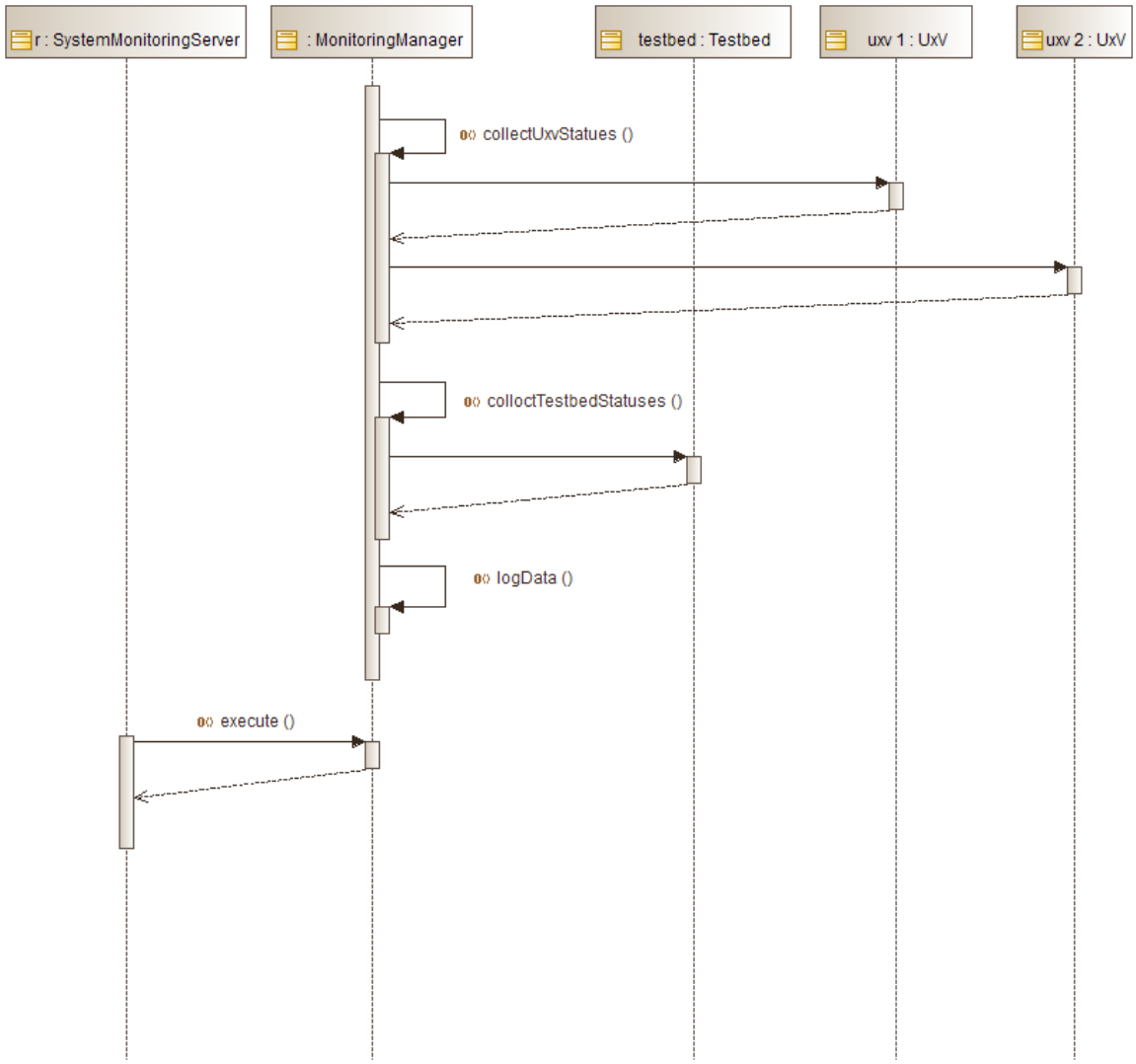

Figure 52: Monitoring Manager – Monitoring sequence diagram

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.

3.3.6 Network Controller

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.

Responsibilities

The main responsibilities of network controller are:

- Provision network communication with Resource Controller
- Enable switching between network technologies
- Check the communication when devices are moving between obstacles

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() as shown at Figure 53. For each experiment Network Controller checks periodically with CheckStatusConnection() the state of the network as depicted in Figure 54. 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. This is depicted in Figure 55.

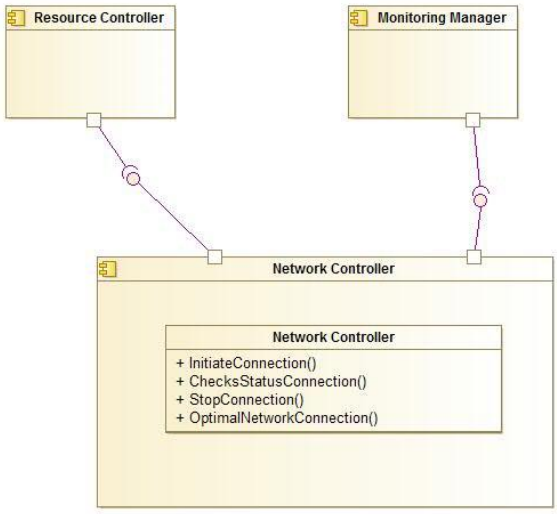

Figure 53 - Network Controller Class Diagram

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

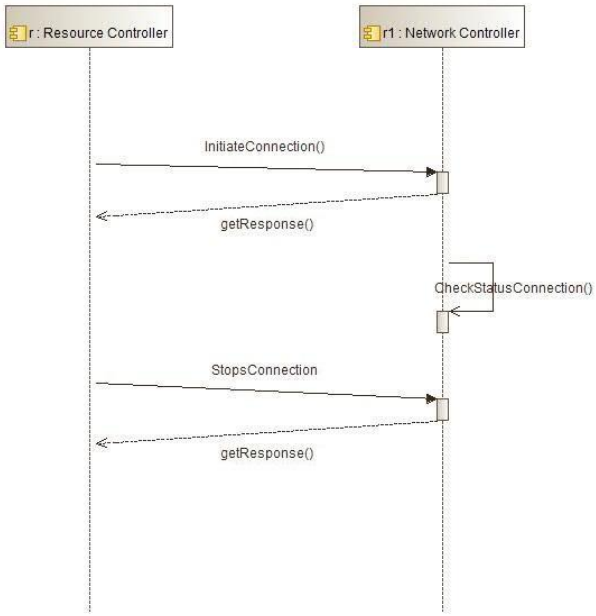

Figure 54 – Starts New Experiment Connection Provisioning

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

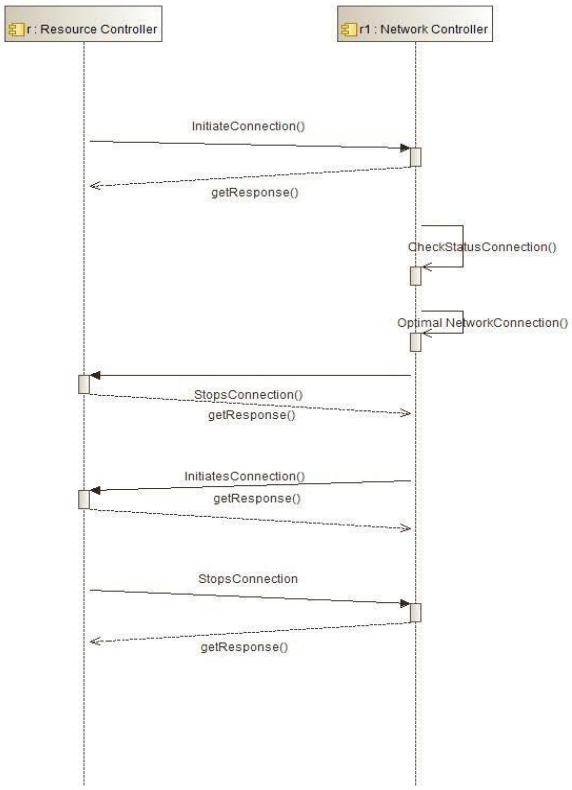

Figure 55 - Change Connection during an experiment

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.

3.3.7 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.

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

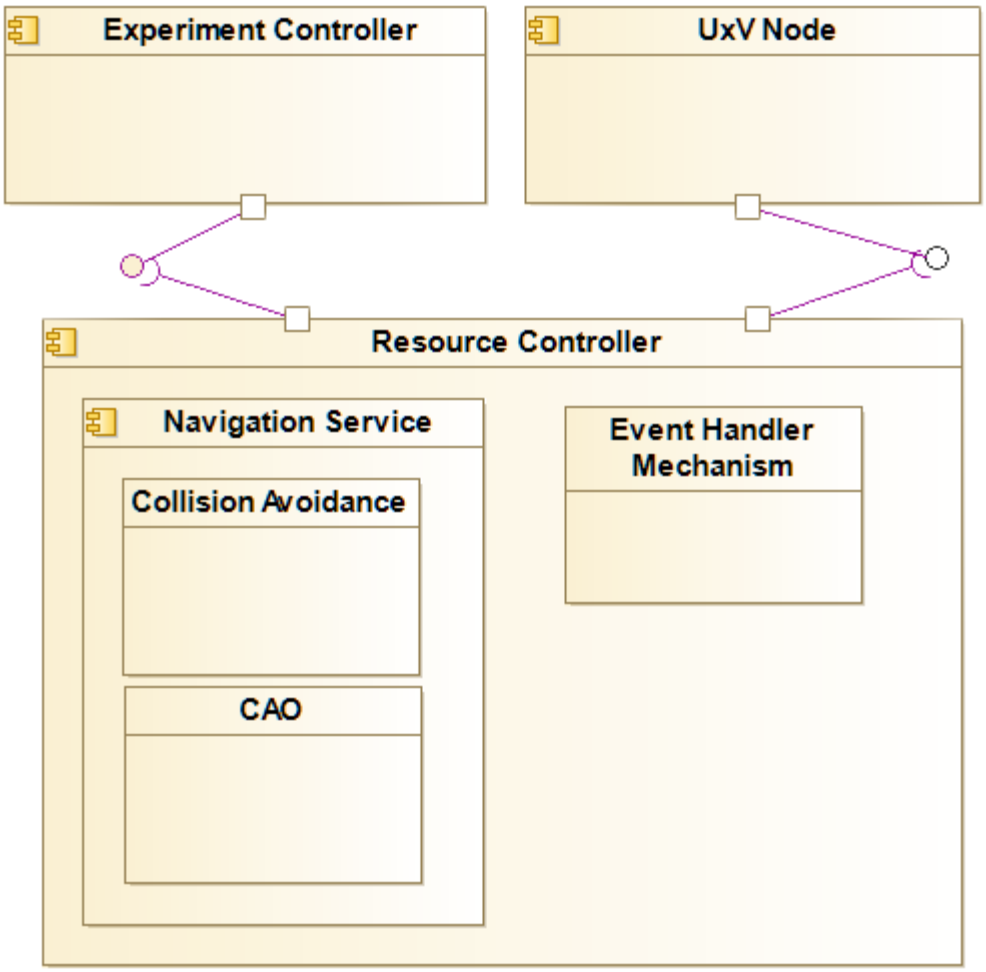

Figure 56: Resource Controller – High level class diagram

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 Testbed Proxy.
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

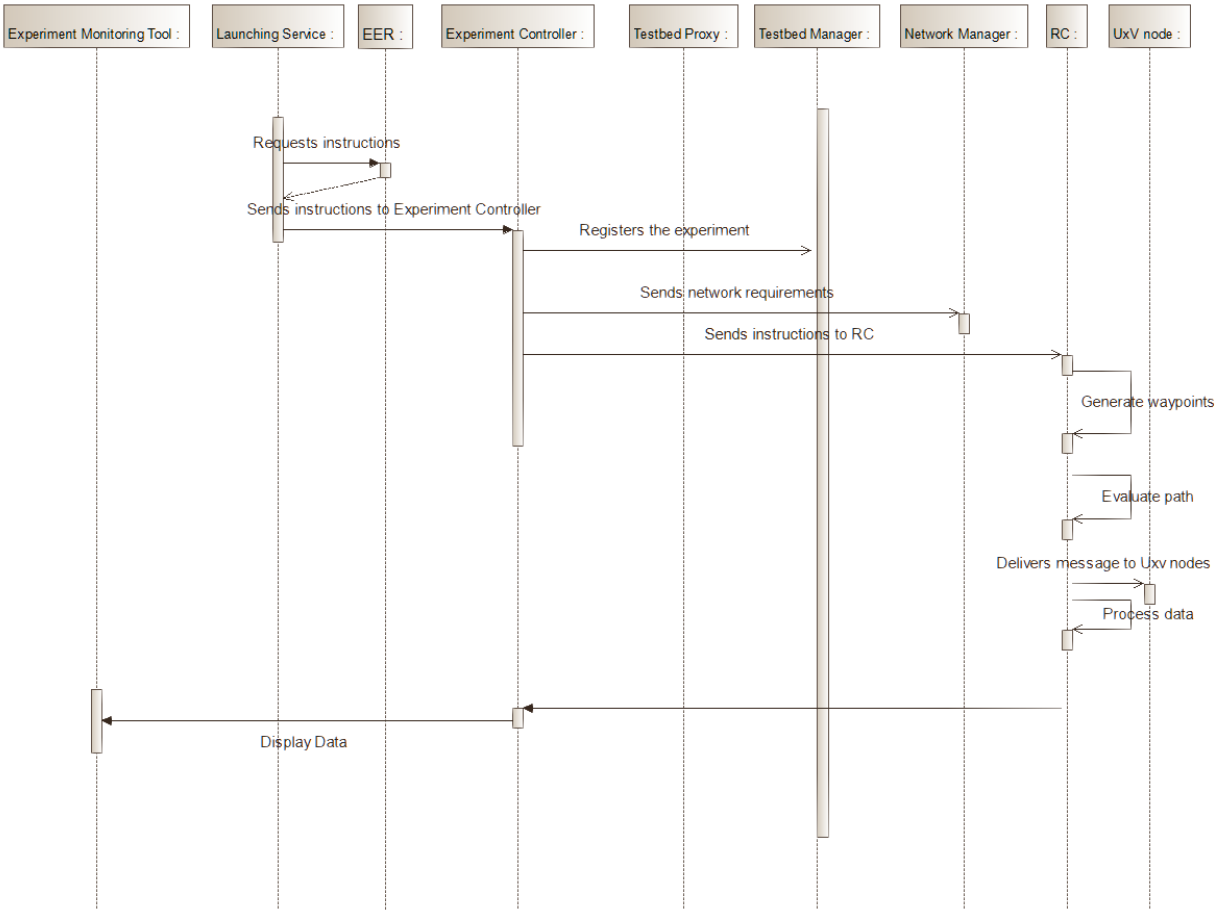

Figure 57: Resource Controller – High level sequence diagram

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

3.4 Testbed facility and resource components

3.4.1 Description

The components described into this category are needed to run the experiments over the correlated UxV resources. These components are also responsible for the monitoring and administration functionalities in each testbed. A high level components diagram with regard to

this subsystem classification is depicted in order to offer a better comprehension of the structure and performance belonging to this portion of the platform.

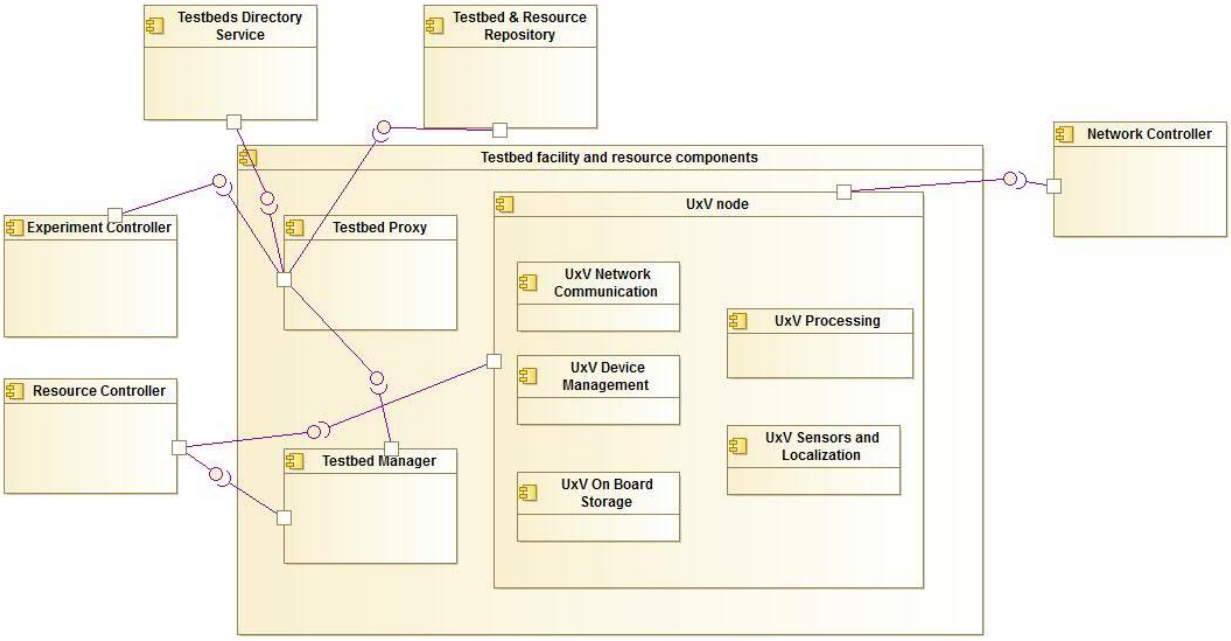

Figure 58: Testbed resources– High level Components Diagram

3.4.2 Testbed Proxy

Testbed Proxy handles the communication between the testbed facility and the middle and data tier.

Responsibilities

The main responsibilities of Testbed proxy are:

- Forward the messages to the relevant components transparently
- Ensure the communication between the middle and testbed tier

Operations and attributes

Testbed Proxy represents a gateway between the middle and the testbed tier. It forwards as shown in Figure 59 the messages from the components that belong to middle tier to the relevant components of testbed tier. Examples of sequence diagrams of testbed proxy have been previously described in Figure 33, Figure 35, Figure 36 and Figure 41

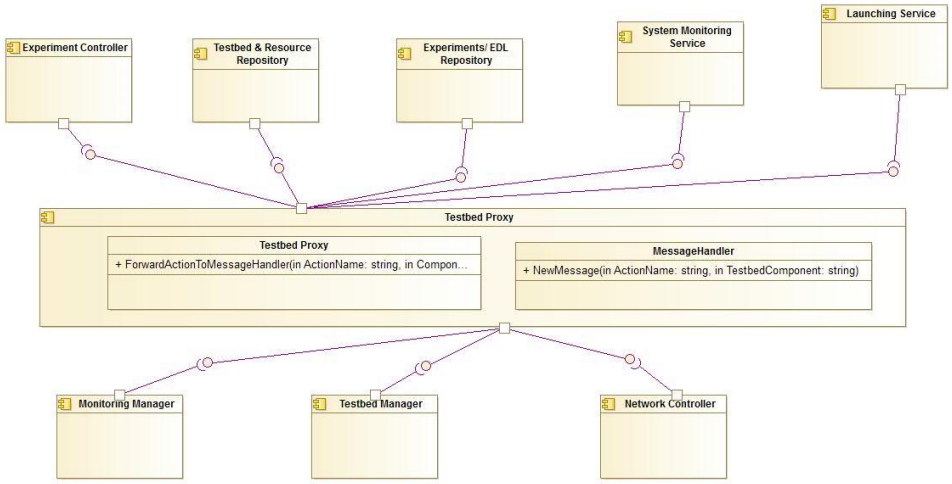

Figure 59: Test Proxy – High level class Diagram

Interactions and relationships with other components

Testbed Proxy interacts as a gateway between middleware and testbed components. Some of the possible scenarios are described below:

- Incoming Messages
    1. Experiment Controller: sends the experiment script to the testbed manager.
    2. Launching service: Sends message to the Resource controller.
- Out coming Messages
    1. Testbeds and Resource Repository: Monitoring metrics are sent via Testbed proxy to this component by Monitoring manager.
    2. Experiment/ EDL Repository: The status of the experiments is periodically being reported to this component by Testbed Manager
    3. System Monitoring Service: The System Monitoring Server checks if system components and services are running. This also includes data about the status of the Testbeds and UxVs from the Monitoring Manager of each Testbed.

3.4.3 Testbed Manager

The Testbed Manager contains accumulated information regarding the experiments and devices of each one of the federation testbeds.

Responsibilities

The main responsibilities of the Testbed Manager are:

- Register an experiment
- Periodically check the status of the experiments and maintains the registration log for the experiments in the tested
- Periodically inform for the status of the experiments Data Repository
- 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.

Operations and attributes

The core operations of Testbed Manager are presented in Figure 60. Testbed Manager contains three sub-classes, i.e. Experiment, UxV and Emergency class. Experiment class is related with the operation related to the experiments.

- AddnewExperiment() registers a new experiment locally and sends this information to the Data Tier
- CollectExperimentStatus() periodically checks for the status of the experiments
- ForwardExperimentScrpttoRC() sends the experiment script to RC in order to start the experiment

UxV class is responsible for adding a new UxV at the testbed –AddNewUxV()- and for its configuration –ConfigureNewUxV()- as shown in Figure 63.

Emergency class contains two operation in case of network problem to start or to stop and to store locally the gathered information from each experiment that would be forwarded to the Data tier.

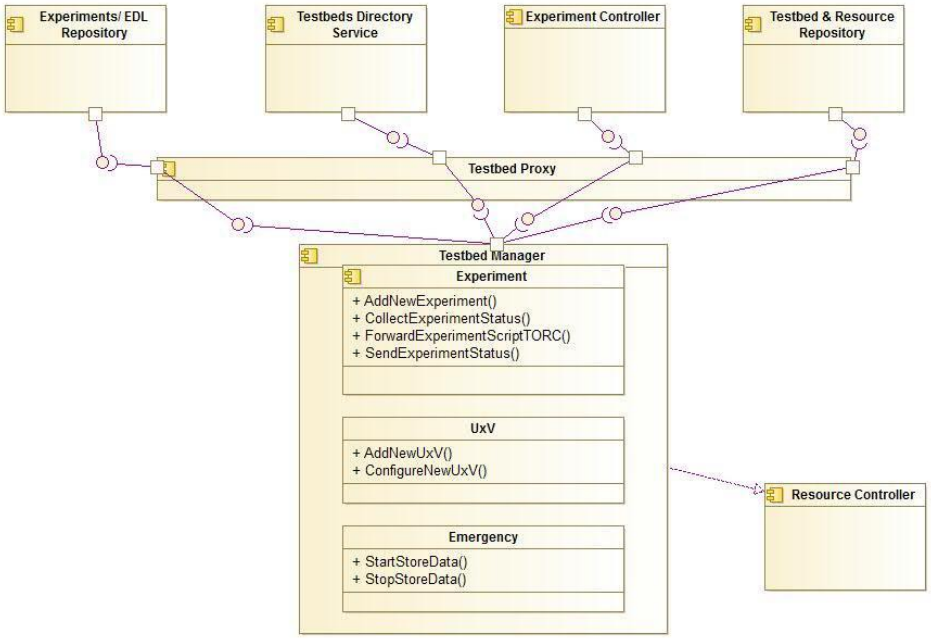

Figure 60: Testbed Manager- High Level class diagram

At Figure 61Figure 52 the needed interaction of Testbed manager are depicted in order to add a new experiment. Experiment Controller sends experiment script with the request to Testbed Manager to start a new experiment. In addition Testbed Manager periodically checks for the status of the experiments as shown in Figure 62.

Add new Experiment

1. EC sends a request to TM to add a new experiment in the testbed
2. TM stores this information locally for experiment
3. TM sends the experiment id to Experiment EDL Repository
4. TM forwards the experiment script to RC in order to start the experiment

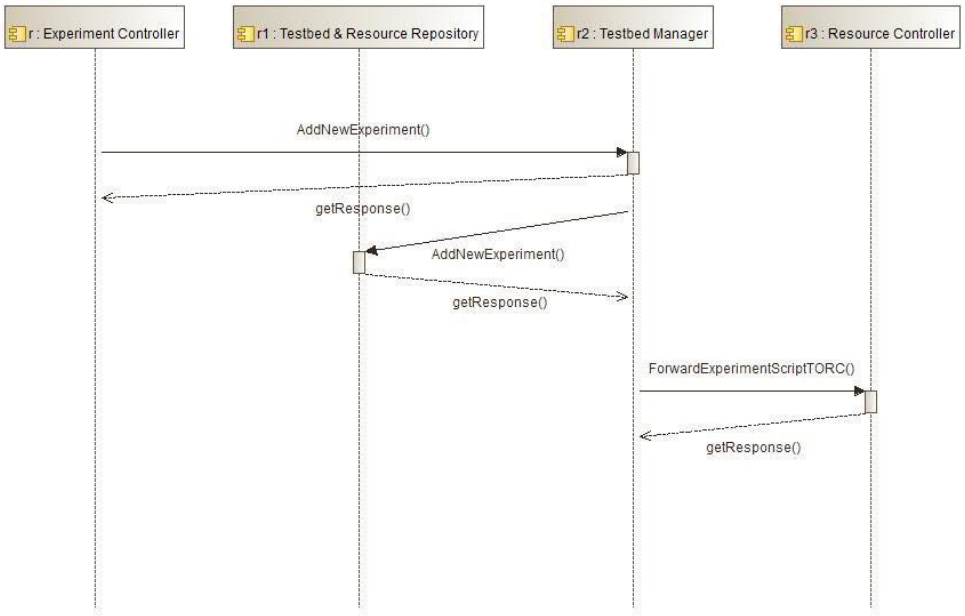

Figure 61: Add New Experiment

Check Experiment Status

1. Testbeds Directory Service request the experiment status of TM
2. TM forwards the request to RC
3. RC answers with the status of the experiment
4. TM stores locally the information and forwards the same information to Testbeds Directory Service

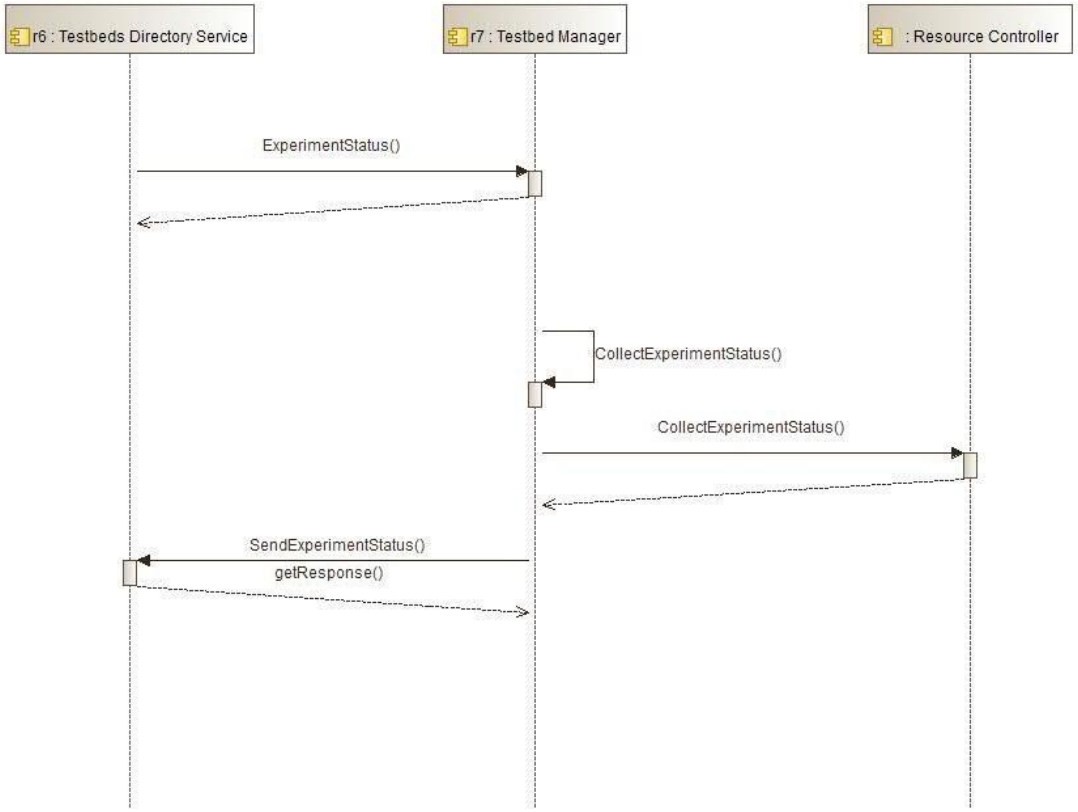

Figure 62: Check experiment status

Add new UxV

1. Testbeds Directory Service sends a request to TM to add a new UxV in the testbed
2. TM adds new UxV and configures it
3. TM informs Testbed & Resource Repository for new UxV
4. TM informs Testbeds Directory Service that the operation ended without error

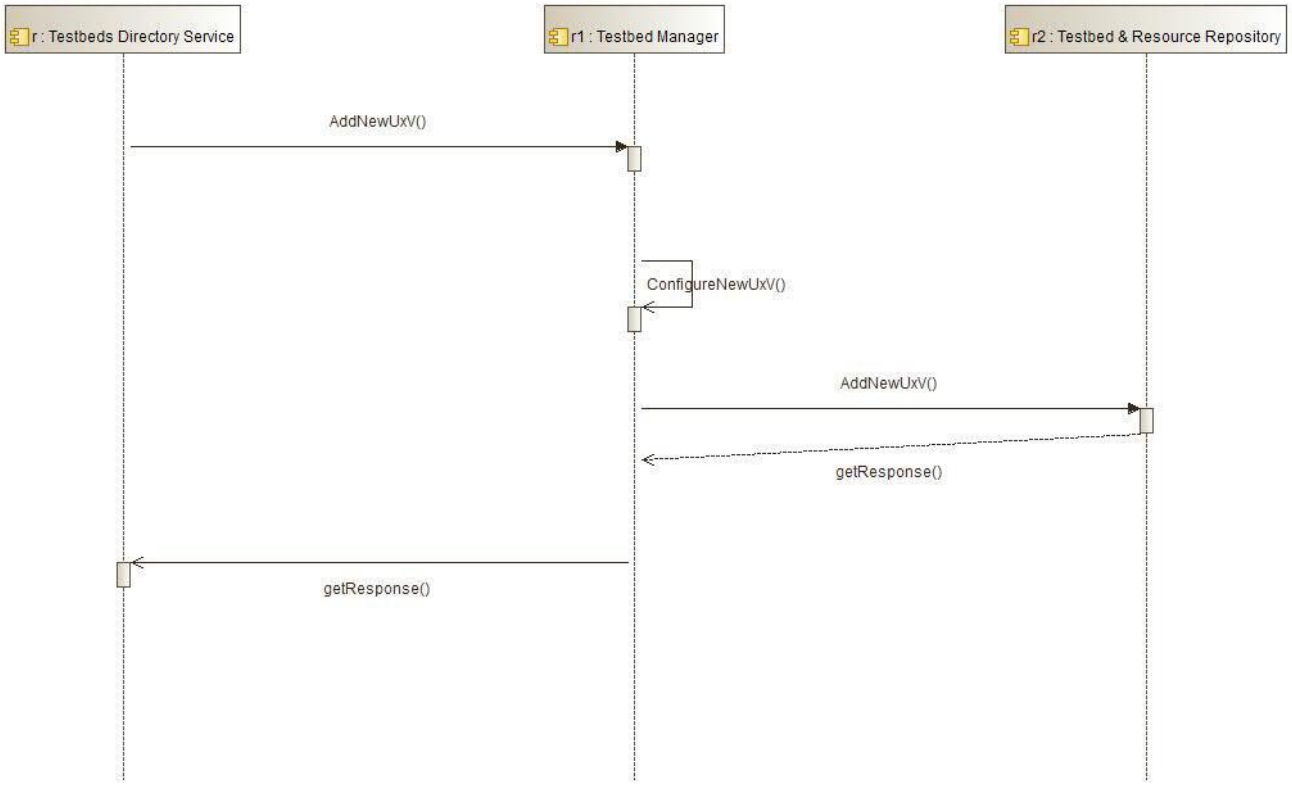

Figure 63: Add new UXV

Interactions and relationships with other components

Testbed Manager communicates with Data Tier: Testbed and Resource Repository and Experiments/EDL repository. It also communicates with experiment controller and resource controller. The experiment script is sent by Experiment Controller to Testbed Manager via testbed proxy that works as a gateway between middle and testbed tier. Each experiment script is forwarded to Resource Controller in order the experiment to be started.

3.4.4 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.

The UxV Node component provides an abstraction layer to the unmanned vehicle systems (such as ROS and other proprietary operating systems) 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.

A high-level preliminary system design was given in project deliverable D4.1, in which the UxV components were established. In that first description, some components responsibilities overlap. Therefore in the present document, the components responsibilities are reworked for more coherence and to avoid redundant responsibilities. For clarity, the updated responsibility distribution is summarised in **Error! Reference source not found.** below.

UxV Component	Main responsibilities
UxV Node	Provide an interface to the robot control mechanisms (waypoints, speed, remote control) and publish the robot localisation information and odometry.
UxV 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.
UxV 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.
UxV 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.
UxV 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.
UxV 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 UxV components tasks and responsibilities

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 such as ***CAM_node, GPS_node and IMU_node***. However, 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. Hence, the components, classes and interfaces described here do not use ROS terminology even though they can be implemented as existing or custom ROS

nodes. ROS nodes or components are given here and in the following sections for illustration purpose and to give hints for a future implementation.

For instance, the ROS controller implements the main UxV control features which form the UxV Node component in RAWFIE.

Responsibilities

The main responsibilities of the UxV Node component are:

- 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.

Operations and attributes

The interface and internal classes of the UxV Node component are given in the class diagram of Figure 65.

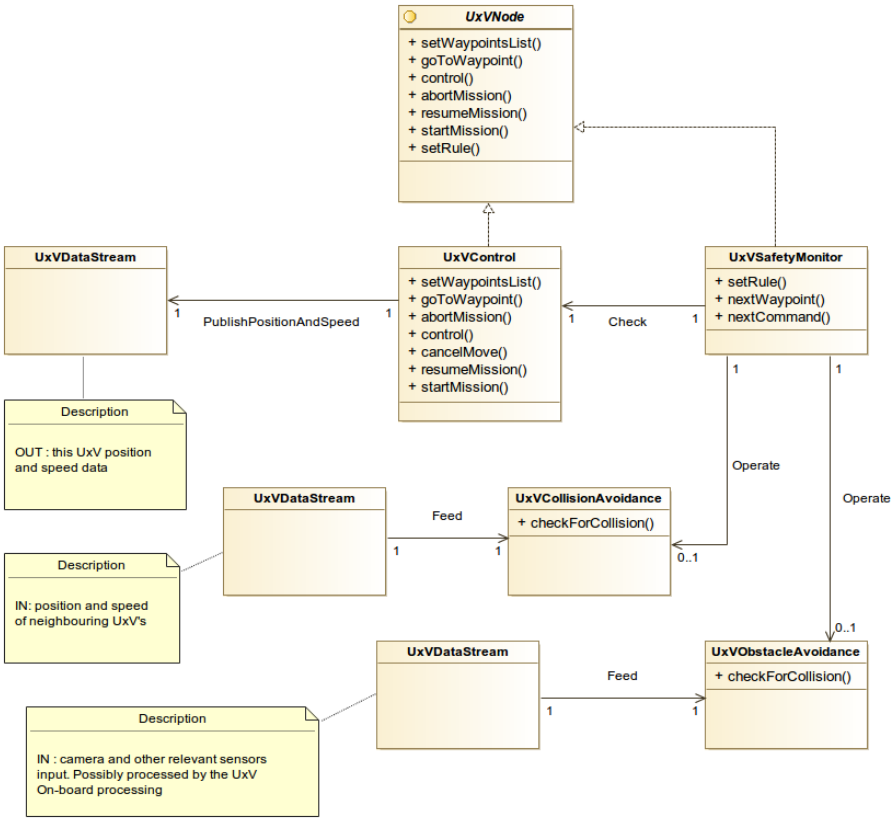

Figure 65: UxV Node component class diagram

The component interface has the following methods:

- setWaypointsList() – provide a list of timestamped waypoints that the robot will have to follow during it mission (implemented by autonomous robots).
- goToWaypoint() – interrupt the on-going mission to move to the specified waypoint and stay there.
- abortMission() – cancel the mission and go back to base.
- control() – low level command for real-time remote control.
- resumeMission() – resume following the waypoints list after an interruption
- startMission() – start the mission (right now or at a specified time)
- setRule() – provide a rule related to the UxV movements such as no-go zone, maximal or minimal depth or altitude, minimal or maximal speed and so on.

Moreover, the UxV Node publishes and subscribed to the following data streams:

- Publish position and speed vector information

- Optional collision and obstacle avoidance mechanism subscribe to the position and speed of neighbouring UxV and to some (possibly processed) sensor outputs, e.g. vision sensor.

The UxV Node controller is a UxVController instance. The controller submits each movement command (either direct command or next waypoint) to a SafetyMonitor which checks it against the rules, the energy level and possibly against information about other UxV's and the neighbouring environment. If the move would break a rule, use too much energy or induce a collision risk, it is cancelled. The SafetyMonitor may cancel the whole mission depending on the UxV condition.

Interactions and relationships with other components

UxV components communicate between each-other using straight method or procedure calls. As there interface is also available to other RAWFIE components, the operations are invoked by messages sent to the UxV after being relayed by the Testbed proxy. The UxV Network Communication component dispatches the messages to the intended component.

The UxV Node receives steering instructions via the Testbed proxy. Direct communication with neighbouring UxV's may be used in relation to a collision avoidance mechanism.

Internally, the UxV Node interacts with the Network and Communication component to receive commands, neighbourhood information and network interface status. It can subscribe to sensor readings published by the Sensors and localisation component and to processed data streams published by the UxV On-board processing. Finally, the UxV Node interacts with the UxV Management component to control on-board resources and publishes status information that the Management uses to construct its dashboard view of the robot.

For instance, the UxV Node controller may communicate with the user by means of commands prompted in the robot system or use interfaces like the pad controller or JSON commands through HTML using Rosbridge or Rostful packages (which could be a practical solution to communicate with the resource controller of the Testbed proxy).

3.4.5 UxV Network Communication

The UxV Network Communication component provides communication services to and from the UxV with the other RAWFIE components. These services form the basis for the other services to interact with the UxV.

In addition, it is planned to offer the UxV the ability to discover its surrounding environment. This applies to fixed entities, such as the RAWFIE testbed infrastructure, the ground based

sensors, etc. as well as vehicles (UxV). This allows for tighter interaction between UxVs, for the dynamic reconfiguration of the UxV communication system, which is then able to circumvent some propagation limitations to support the autonomous and self-coordinated operation of several UxV.

The UxV Network Communication component handles two kinds of data:

- RAWFIE Messages as defined by the MessageBus abstraction (see section 3.2.2) representing commands or remote procedure calls to access the UxV components functions.
- Data streams as defined by the UxV Sensor and localisation component. Typically, such data stream contain sensor data.

Messages and data streams sent by RAWFIE components to a UxV component are relayed by the Testbed proxy.

Responsibilities

The main responsibilities of the UxV Network Communication component are:

- Detect, set-up, control and use the network interfaces installed on the UxV
- Receive and de-capsulate incoming messages from the Testbed, deliver them to the relevant on-board component.
- Encapsulate and send messages originating from on-board components to the RAWFIE platform via the Testbed.
- Detect and observe the neighbouring UxV's.

Operations and attributes

The services provided by the UxV Network Communication are:

- Identification service: let the UxV to be uniquely identified within the Rawfie ecosystem. Answer identification requests coming from other Rawfie entities (Testbed, UxV's sensors, …) and broadcast the identifier when exploring the neighbourhood.
- Data transfer service: this is the core service of the component. Support the exchange of messages and data streams with the Testbed that controls a given UxV.
- Status notification: this service broadcasts, or, depending on the user preference, responds to requests with important status information such as position, battery charge and so on.
- Capabilities and directory services: this service allows the neighbourhood, in particular neighbouring Testbeds to discover the capabilities of a UxV, such as performance (speed, altitude, depth, processing power, energy) and on-board peripherals (sensors, actuators).

- Reconfiguration: the Network Communication component provides a service for the reconfiguration and selection of the network interface, giving the possibility to escape difficult transmission conditions such as a channel suffering from heavy interferences.
- Network interface monitoring: observe the active on-board network interfaces and inform the UxV Management component on their performance (current data rate, interferences, packet success rate).
- Neighbourhood monitoring: by means of existing or tailor-made protocols, observe neighbouring robots and relay their position and speed vector information.

The following class diagram details the UxV Communication component interface and internals:

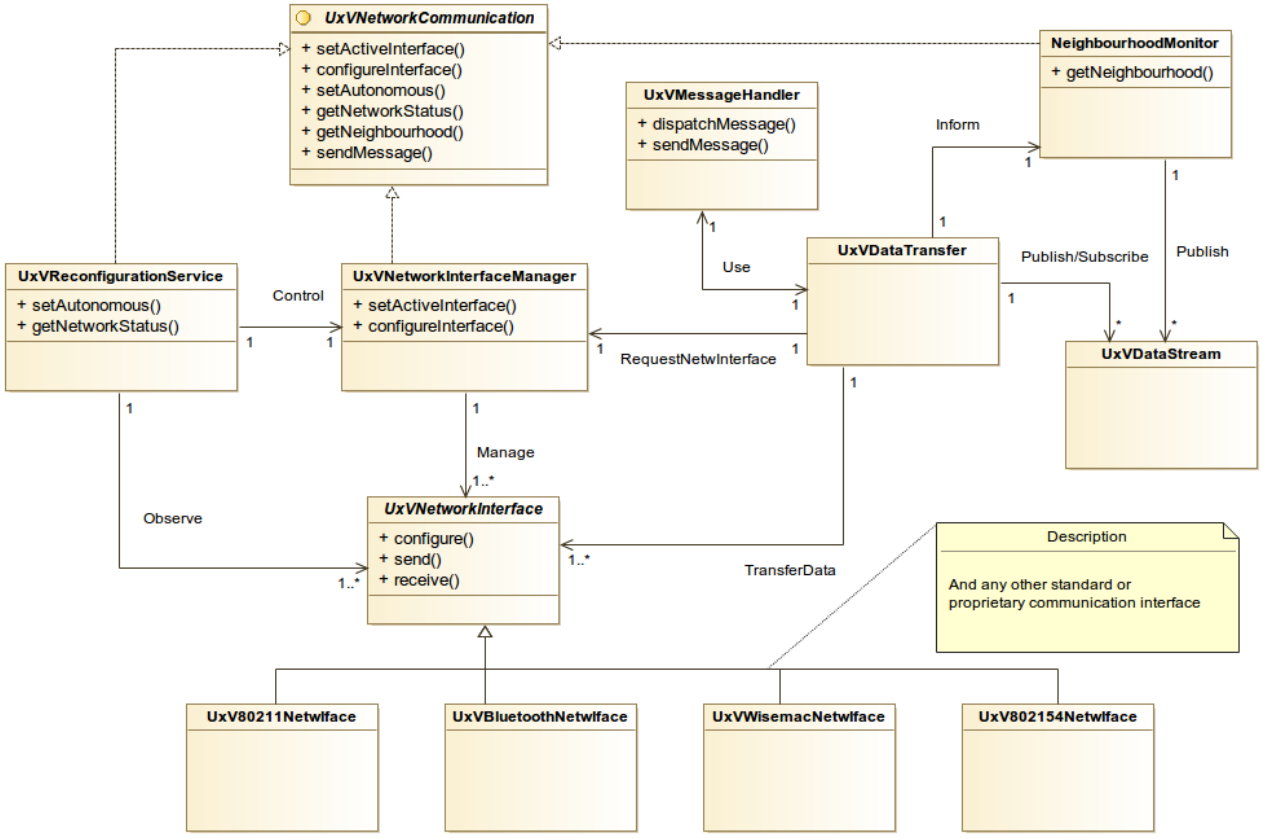

Figure 64: UxV Network communication component

The Network Communication component exposes the following interface to other RAWFIE components:

- setActiveInterface() – enforce which network interface the UxV shall use.
- configureInterface() – configure a UxV network interface (includes turn it ON or OFF).

- setAutonomous() – enable or disable the network reconfiguration autonomous mode. If enabled, the component may change the interface configuration (e.g. channel) or select another interface autonomously.
- getNetworkStatus() – returns a representation of the status and performance of the active network interface.
- getNeighbourhood() – returns a representation of the other UxV's detected in the vicinities using the information gather from the network.
- sendMessage() – submit a message for transmission.

The component also relays the Data Streams published by the other components of the same UxV to the RAWFIE platform via the Testbed and publishes Data Streams received from the network for the use of the internal UxV components. In a ROS implementation, this tasks is implicit as it is fulfilled by ROS.

Received messages are given to a MessageHandler which dispatches them to the recipient component and provides an interface for sending responses or events.

Interactions and relationships with other components

With the outside world, the UXV Network Communication Component will interact with its counterpart installed on peer UxV devices and with the Testbed owning the UxV (Network Controller, Resource Controller). Internally, this component will interact with all the other Rawfie components installed on the UxV, mostly to relay messages and data streams as well as providing information on the network status.

3.4.6 UxV Sensors and Localization

Sensors are providing the application with measurement points, typically tuples made of a location, a timestamp, a source sensor (e.g. an unique identifier) and one or several samplings.

Localisation is a specific type of measurement using positioning systems or a combination of measurements to estimate a location.

The main difficulty for this component is that the kind of sensors and the way they are operated, the type and size of data they provide varies considerably. This makes the definition of a standard interface for sensor operation difficult. The design choice chosen for this component is to provide detailed classes to describe common and widely used sensors and their interface (cameras, GPS, temperature) and to define more generic classes that can be derived to described particular sensors that are not commonly used.

To enable the interaction with any kind of sensor, the UxV Sensor component will comprise an extensible Sensor Control Interface. This interface shall be defined as a protocol specifying commands for commonly used sensors and allow the easy addition of new commands for particular sensors. Inheritance will be used to maximise reuse.

Each UxV shall be able to list the available sensors as well as describe the Sensor Control Interface commands, either standard or particular, that are available. This is list shall be accessbiel from the UxV Network communication component directory service.

The UxV Sensor and localisation component design should be established with ROS in mind to promote reuse since the ROS meta-OS is implemented on many existing UxV's. The ROS system has indeed different processes running for the operation of sensors such as the publish/subscribe mechanism for sensor readins. Widely used ROS sensor nodes are: ***CAM_node, GPS_node and IMU_node***. However, the present design shall not depend on ROS as some UxV's may not be equipped with it. Hence, the components, classes and interfaces described here do not use ROS terminology even though they can be implemented as existing or custom ROS nodes. ROS nodes or components given here are for illustration purpose only.

Responsibilities

The main responsibilities for the UxV Sensor and Localisation component are:

- Read data from physical sensors (sensor driver)
- Convert raw data to standard messages
- Publish the sensor values
- List available sensors and describe their interface
- Support a Sensor Control Interface that is standard in the RAWFIE ecosystem.

It is assumed that each UxV is equipped with localization and timing capability to be able to tag sensor readings with localization and timestamp.

Operations and attributes

This component provides a sensor directory, a command interface and several layers of sensor abstractions.

The following interface is defined:

- findSensor() – attempt to find a sensor on the UxV which fills the given criteria.
- startAcquisition() – start acquisition on the selected sensor at a given rate.
- stopAcquisition() – stop the on-going acquisition on the given sensor.
- sampleOnce() – sample the given sensor once.

- getDescription() – get a standardised description of the sensor and the output format (e.g. in XML).

The Sensor controller checks for the sensor availability, power consumption and the UxV energy status before starting an acquisition.

Each sensor output is provided through the publication of a Data Stream (see the UxV On-board storage component).

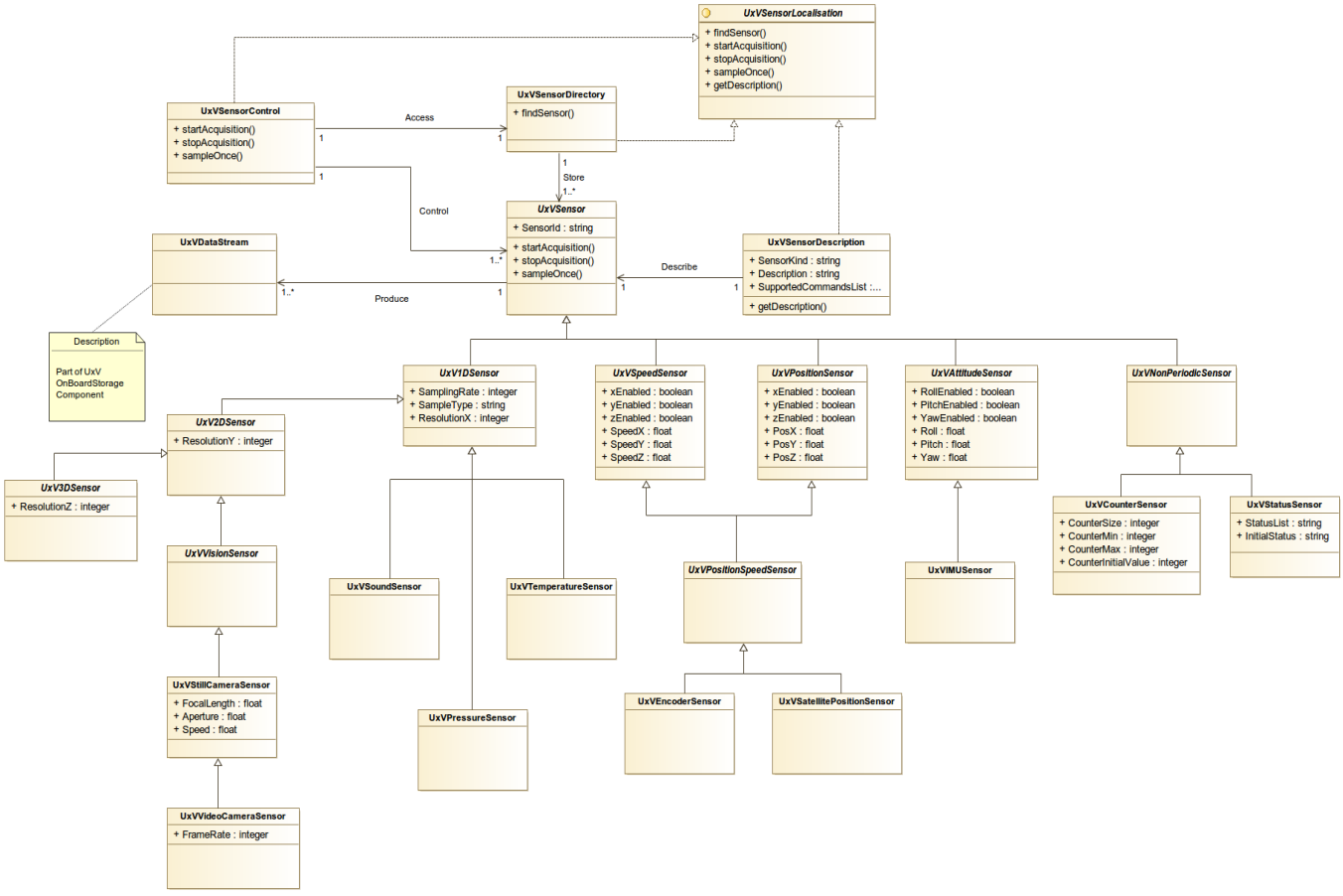

Figure 65: UxV Sensor and Localisation Component

Here also, ROS nodes implement part of the Sensor and Localisation component functionality:

- CAM_node:
  - setPanTiltZoom: sets the position of the camera pan and tilt motors, and the zoom value
    - + setPanTiltZoom(robotnik_msgs/ptz)
  - publishVideoStream: reads from the camera and publish the stream
    - - publishVideoStream(): sensor_msgs/Image

- IMU_node:
  - publishIMUValues: reads from the IMU sensor and publish the current values.
    - - publishIMUValues(): sensor_msgs/Imu
- GPS_node:
  - publishPosition: reads from GPS sensor and publish the position
    - - publishPositon(): sensor_msgs/NavSatFix

Interactions and relationships with other components

The UxV Sensor and Localisation component interacts with the UxV Network Communication directory service for sensor availability and capability listing as well as with the Data Transmission service of the same component which relays messages of the Sensor Control Interface protocol. It interacts indirectly with the same service which subscribes to the sensors data flows.

If the whole testbed is implemented in ROS, the sensors nodes are interacting with the robot user (Resource Controller) throughout standard ROS servers such as Rosbridge or Rostful and most of the Network Communication component are provided by ROS.

The On-Board storage and On-board processing can also subscribe to the data published by the sensors and interrogate the sensor list for sensor capability and data format.

Interactions with Rawfie components outside the UxV such as the Experiment Controller are done via the Message bus and the Network communication component.

3.4.7 UxV On Board Storage

The UxV embeds some storage to keep data for a variable duration. Typically, sensor measurements are stored locally when they cannot be immediately sent over the communication link to the other entities or components. Another example is the status of the UxV during an experiment will be internally stored for later UxV maintenance or post-mortem analysis.

This service can be offered through ROS which has a tool that allows the user to record and play back all the data being generated at some point. RAWFIE partner Robotnik We havehas implemented a couple of services capable of using this tool to record, stop recording and play back the data.

Responsibilities

The main responsibilities of the On Board storage are being able to record in permanent storage any data flow produced by the UxV and to play it back for any entity that requires it, possibly via the Network Communication component. The On Board Storage component shall also store the Data flow metadata such as size, type, format and so on.

Storage capability is provided by the UxV hardware and operating system. Hence this component provides a data stream abstraction and an interface to the storage system installed on the UxV to other Rawfie components. The data stream abstraction is useful beyond storage and can be used for internal data communication and also for network transmission. The data stream abstraction follows the publish/subscribe principle.

In ROS, the On-board Storage component would use ROS components to perform the following actions:

- Record the specified data in a .bag file and store it in a system folder
- Stop the recording
- Play back the data stored in the .bag file (publish again the topics)

Operations and attributes

The attributes and operations of the On-board storage component are described in the class diagram below.

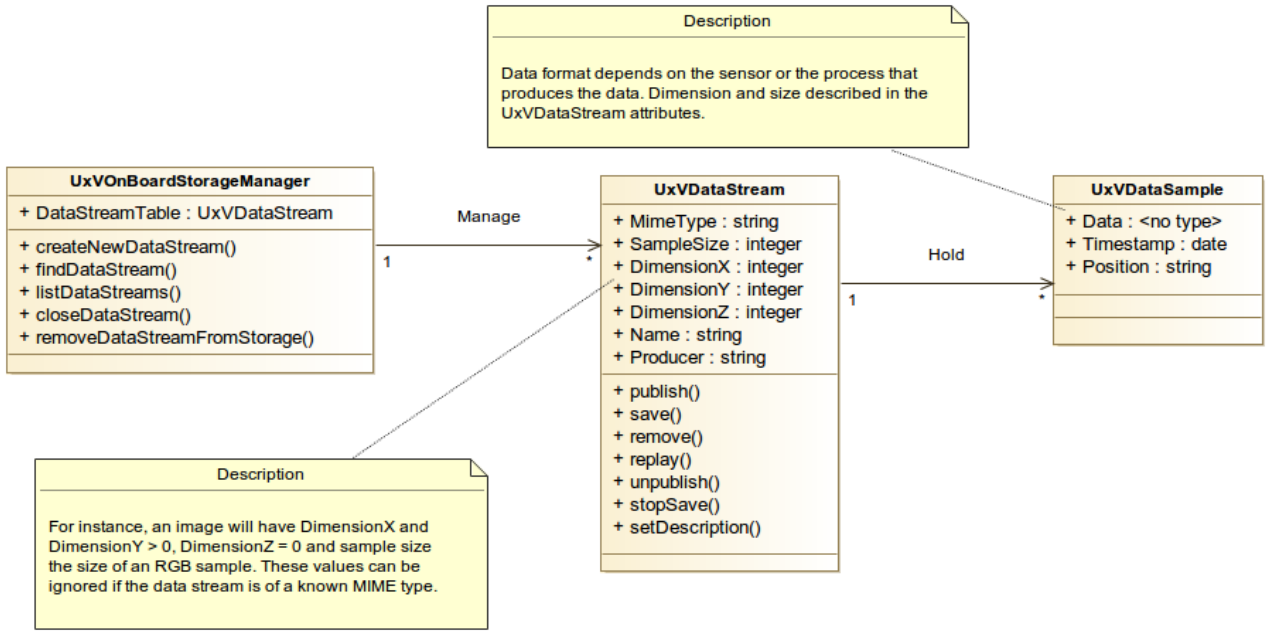

Figure 66: UxV On-board storage class diagram

The following interface is exposed:

- createNewDataStream() – create and open a new data stream instance.
- findDataStream() – attempt to find an on-going or saved data stream that corresponds to the search criteria.
- listDataStreams() – list all opened and saved data streams.
- closeDataStream() – close an on-going data stream.
- replayDataStream() – publish a saved data stream.
- configureDataStream() – set the data stream properties: description, whether it must be saved or not, publication properties such as publication topic and whether it must be published on the network or only internally.

Depending on the implementation, some of the above functions my be provided by the robot operating platform. For example, if the component is implemented in ROS, the save() and replay() operations translate to the following ROS methods:

- save(): It activates or deactivates the recording.
  - *RosbagRecord(Bool)*
- replay(): It activates or deactivates the playback of a certain bag file (passed as a string).
  - RosbagPlayBack*(Bool,String)*

Interactions and relationships with other components

The On-board storage interacts locally with the UxV Sensor and Localisation component and any other data producer on the UxV. Any data must be published using a UxVDataStream managed by the On-board storage component even if permanent storage is not used.

Access to the On-board storage UxVDataStream instances is provided to external Rawfie components through the UxVOnBoardStorageManager which provides a find method.

3.4.8 UxV Processing

The UxV is able to process the sampled data produced by its sensors or other information it has received through the communication links to either increase the information level or compress the data elements into more concise or aggregated forms, such as compressed format, spectrographic analysis, averages, FFT, aggregation, etc.

The UxV processing component aims at providing the UxV with the typical means for processing the sampled data (e.g. temperature, pressure, etc.), including the data coming from the UxV management, message bus and operational modules (engine, ancillary systems, localisation system, specific actuators, etc.) and making the results available to any other relevant modules.

Data streams produced by the on-board processing component may be used by the robot itself, for instance localisation data derived into speed vectors.

Responsibilities

The main responsibilities of the UxV processing component are:

- Process data streams produced or received by the robot to compress, extract and/or present information in a manner that is useful for the experiment.
- Accept and recognize data from all sources accessible by the embedded computing unit.
- Compute outputs according to pre-defined programs or rule.
- Make the results of the computation available to the other modules or entities, e.g. through a message bus abstraction. The results shall be available for whichever entity needs it: a process of the same UxV, another UxV or a component of the Rawfie middleware.

The On-board processing component has no other responsibility than producing an output stream from an input stream and a processing algorithm. It is not involved in the robot operations such as navigation. Even though some internal robot operation process may use data processed by the On-board processing component, the operation process itself is not part of this component. For instance, the On-board processing may execute an algorithm that processes data received from neighbouring robots on their localisation and publish them for a collision avoidance mechanism. In that case, the collision avoidance mechanism uses the data produced by the On-board processing, but it is not part of it. A collision avoidance mechanism would be located in the UxV Node component.

Operations and attributes

The On-board processing component comprises standard or tailored-made data processing algorithms, connects them to one or several input data streams to produce and publish an output data stream representing the result of the input processed by the algorithm.

This component has a central management entity which maintains a list of the available processing algorithms, their input conditions and a description of the output format. The central management entity is responsible to create the algorithm instance and connect the input streams on request from an internal or external component. The On-board processing manager is able to search into the algorithms list and to integrate new algorithms received over the air by the UxV. It includes some safety feature such as checking for the available resources (battery, computing power, memory) before accepting to start a processing task.

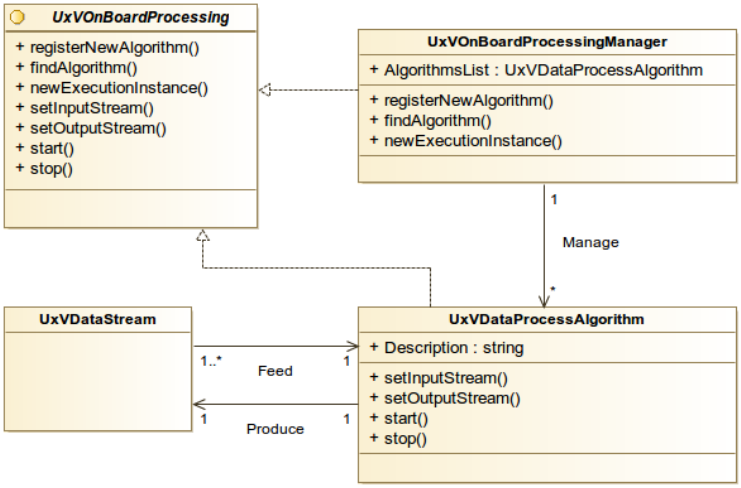

Figure 67: UxV On-board processing component class diagram

The interface of the UxV On-board processing component provides the following methods:

- registerNewAlgorithm() – register a new algorithm with the On-board processing component. The algorithm code and metadata including resource usage shall be passed with the call. Some UxV's may not support his feature or it may require off-line operation, possibly by a human.
- findAlgorithm() – find a data processing algorithm that corresponds to the search criteria provided with the call.
- newExecutionInstance() – create a new execution instance of a given algorithm.
- setInputStream() – connect a data stream to one input port of a given algorithm execution instance.
- setOutputStream() – connect a data stream to the output port of a given algorithm execution instance.
- start() – start an algorithm execution.
- stop() – stop an algorithm execution.

Interactions and relationships with other components

This component interacts with other entities through subscription and publication of UxVDataStreams. It also provides an interface for the components that which to add a new processing algorithm and/or start some processing task.

3.4.9 UxV Device Management

The UxV Device Management offers a centralised dashboard view of the UxV and allows to browse through its resources. It also offers an interface to set some vehicle configuration parameters and enable or disable resources. It also implements the Identification service.

Responsibilities

The main responsibilities of the UxV device management component are:

- Identification service: let the UxV to be uniquely identified within the Rawfie ecosystem. Answer identification requests coming from other Rawfie entities (Testbed, UxV's sensors, …) and broadcast the identifier when exploring the neighbourhood.
- Resource status and capabilities directory service: dashboard view of all the UxV resources, their nominal performance and their status. The status can be accessed once or regularly published. A search function is also provided. A resource can be enabled or disabled by the Management component. If enabled, another component may start or use the resource. If disabled, a resource cannot be used at all.
- Configuration service: provides a list and search engine for all the UxV global configuration parameters as well as write and read access.

Operations and attributes

The internal structure and interface of the UxV Management component is given in the class diagram in Figure 70 below.

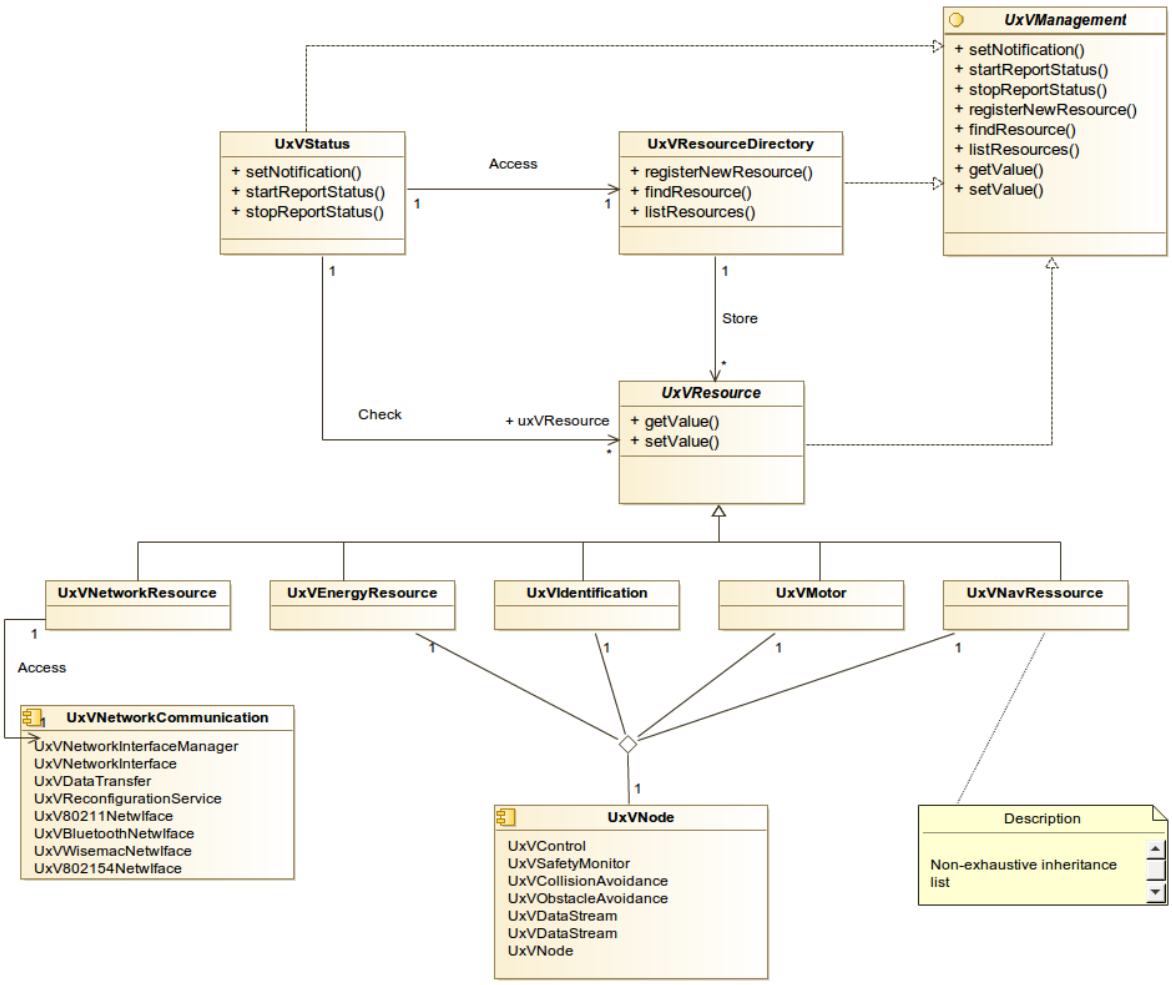

Figure 70: UxV Management class diagram

The following interface is provided:

- setNotification() – receive a notification if the status of a certain resource reaches a given value.
- startReportStatus() – add the given resource in the status report published by the UxV Management.
- stopReportStatus() – remove the given resource in the status report published by the UxV Management.
- registerNewResource() – register a new resource for the UxV Management to observe and provide access to.
- findResource() – search for a registered resource that correspond to the search criteria provided.
- listResources() – list all registered and available resources of the UxV.

- getValue() – get the value or status of the accessed resource.
- setValue() – set the value of the accessed resource (includes its enabler).

Interactions and relationships with other components

The UxV Management component interacts with the UxV operating platform and with other components such as the Network and communication to gather a global view of the UxV status. The status is published for use by other components, either inside or outside the UxV. The UxV Management receives UxV configuration commands from the Testbed and publishes identification information.

4 Summary and Outlook

The design proposed into this document provides an architectural overview and individual component specific definition, explaining how the multiple components over the RAWFIE platform interact and communicate between them.

As a result, this document will lead to future design activities and important improvements during the forthcoming iterations with the aim to provide a precise and correct implementation inception, and even fulfill all the user functional requirements previously specified as well as those critical non functional requirements.

Concerning the next document version, this one will be focused on detailed UML artifacts, mainly class diagrams. Also a deeper immersion into the technologies adopted in order to carry on with the implementation and consecutively verification tests for integrating and deploying the resource environments design.

5 References

[1] http://www.fed4fire.eu/
[2] http://www.nagios.org/
[3] http://www.icinga.org/
[4] http://mathias-kettner.com/checkmk_livestatus.html