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	D4.1 - High Level Design and Specification of RAWFIE Architecture		
Confidentiality	PU	Deliverable type[1]	R
Deliverable File	D4.1	Date	31.05.2015
Approval Status[2]	2nd Reviewer	Version	1.0
Contact Person	Marcel Heckel	Organization	Fraunhofer
Phone	+49 351 / 4640-645	E-Mail	marcel.heckel@ivi.fraunhofer.de

[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
Marcel Heckel	Fraunhofer	marcel.heckel@ivi.fraunhofer.de
Blerina Lika	UOA	b.lika@di.uoa.gr
Kostas Kolomvatsos	UOA	kostasks@di.uoa.gr
Kakia Panagidi	UOA	kakiap@di.uoa.gr
Stathes Hadjiefthymiades	UOA	shadj@di.uoa.gr
Giovanni Tusa	IES	g.tusa@i4es.it
Kiriakos Georgouleas	HAI	GEORGOULEAS.Kiriakos@haicorp.com
Nikolaos Priggouris	HAI	PRIGGOURIS.Nikolaos@haicorp.com
Jason Ramapuram	HES-SO	jason@ramapuram.net
Lionel Blondé	HES-SO	lionel.blonde@hesge.ch
Cveta Dimitrova	Epsilon	cveta.dimitrova@epsilon-bulgaria.com
Ricardo Martins	MST	rasm@oceanscan-mst.com
Alexandre Sousa	MST	alex@oceanscan-mst.com
Elias Kosmatopoulos	CERTH	kosmatop@iti.gr
Philippe Dallemagne	CSEM	pda@csem.ch

REVIEWERS TABLE

Name	Company	E-Mail
Sarantis Paskalis	UOA	paskalis@di.uoa.gr
Philippe Dallemagne	CSEM	pda@csem.ch
Kiriakos Georgouleas	HAI	GEORGOULEAS.Kiriakos@haicorp.com
Nikolaos Priggouris	HAI	PRIGGOURIS.Nikolaos@haicorp.com

DISTRIBUTION

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

CHANGE HISTORY

Version	Date	Reason for Change	Pages/Sections Affected
0.1	2015-03-10	Initial version	all
0.2	2015-04-17	Restructured and components tables added	all
0.3	2015-04-20	"Relevant FIRE projects" added	Section 2.1
0.4	2015-04-21	Component and Use cased responsibilities edited	Sections 4 and 5
0.5	2015-04-23	Components refined	Section 4
0.6	2015-04-23	Addition to state of the art	Section 2
0.7	2015-04-04	Addition to state of the art	Section 2
0.8	2015-04-05	Addition to state of the art	Section 2
0.9	2015-04-06	Addition to state of the art, architectural overview and components	Sections 2, 3, 4
0.10	2015-04-07	Addition to state of the art	Section 2
0.12	2015-04-08	Review of contributions	all
0.13	2015-04-11	Restructuring of document. Moved section 2 at the end.	all
0.14	2015-04-18..25	Several contributions and restructuring	all
0.15	2015-04-26	Finalize version for 1st review	all
0.16	2015-04-28	Review by HAI	all
0.17	2015-04-28	Review by CSEM	all
0.18	2015-04-29..31	Addressing Review comments	all
1.0	2015-04-31	Final Version	

[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 describes the first version of the RAWFIE high lever architecture. An overview of all components and there interaction is given. Also a state of the art summary of technologies that may be used to implement the architecture is given.

Keywords:
architecture, components, interactions, technology overview

Part II: Table of Contents-

List of Figures

List of Tables

Part III: Executive Summary

This deliverable describes the planed high level architecture of RAWFIE. First a general overview of the architecture is given. Following each planned component is described and relation to other components are noted. After this, several use cases of the RAWFIE system are investigated and sequence diagrams visualize how these use cases will be handled using the planned architecture. At the end of the document, an excessive state of the art summary is given that list technologies that may be used to implement the planned architecture.

Part IV: Main Section

1 Introduction

1.1 Scope of D4.1

The DoW contains already a comprehensive overview of the planned high-level architecture. This deliverable will go more into the technical details:

- Clearly outline all the planned components
- Define the capabilities of the component interfaces
- Show possible interactions and dependencies between different components
- Describe the runtime environment of the components (cloud, local server, etc.)

In addition, this deliverable includes an analysis of existing technological solutions in areas related to RAWFIE.

1.2 Relation to other deliverables

A detailed requirement analysis was given in D3.1. Based on these requirements and the planned functionalities from the DoW, this architecture document was created. The stakeholders used in the diagrams are described in detail within D3.1.

D4.2 is expected to provide detailed components descriptions. Therefore, the intention of this deliverable is just to describe the components and their interfaces on a high level.

1.3 Abbreviations

Abbreviation	Meaning
3D	three-dimensional space
API	Application programming interface
AT	Aerial Testbed
AUV	Autonomous Underwater Vehicle
CA	Certification Authority
DoW	Description of Work
EDL	Experiment Description Language
EER	Experiments and EDL Repository
EVS	Experiment Validation Service
GUI	graphical user interface
IDE	integrated development environment
KPI	Key Performance Indicator
MM	Monitoring Manager
MT	Maritime Testbed
NAT	Network Address Translation
OMF	Orbit Management Framework
OML	ORBIT Measurement Library

RC	Resource Controller
REST	Representational state transfer
SAML	Security Assertion Markup Language
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SSO	Single-Sign-On
TM	Testbed Manager
UAV	Unmanned Arial Vehicle
UGV	Unmanned Ground Vehicle
USV	Unmanned Surface Vehicle
UxV	Unmanned aerial/ground/surface Vehicle
VT	Vehicular Testbed
WSDL	Web Services Description Language
XMPP	Extensible Messaging and Presence Protocol

Table 1: Common abbreviations

1.4 Disambiguation

- Module:
  - Modules deal with code packaging and the dependencies among code.
  - A set of code packages within one software product that provides a special functionality
- Component:
  - A reusable entity that provides a set of functionalities (or data) semantically related. A component may encapsulate one or more modules or packages and should provide a well defined API for interaction
- Subsystem
  - A collection of components providing a subset of the system functionalities.
- System
  - A collection of subsystems and/or individual components representing the system as a whole.
- Tool
  - A GUI implementation to do a special thing, e.g. the “Resource Explorer tool” to search for a resource.
  - Can be a module or component

2 Architectural Overview

This chapter gives an overview over the architecture and its components.

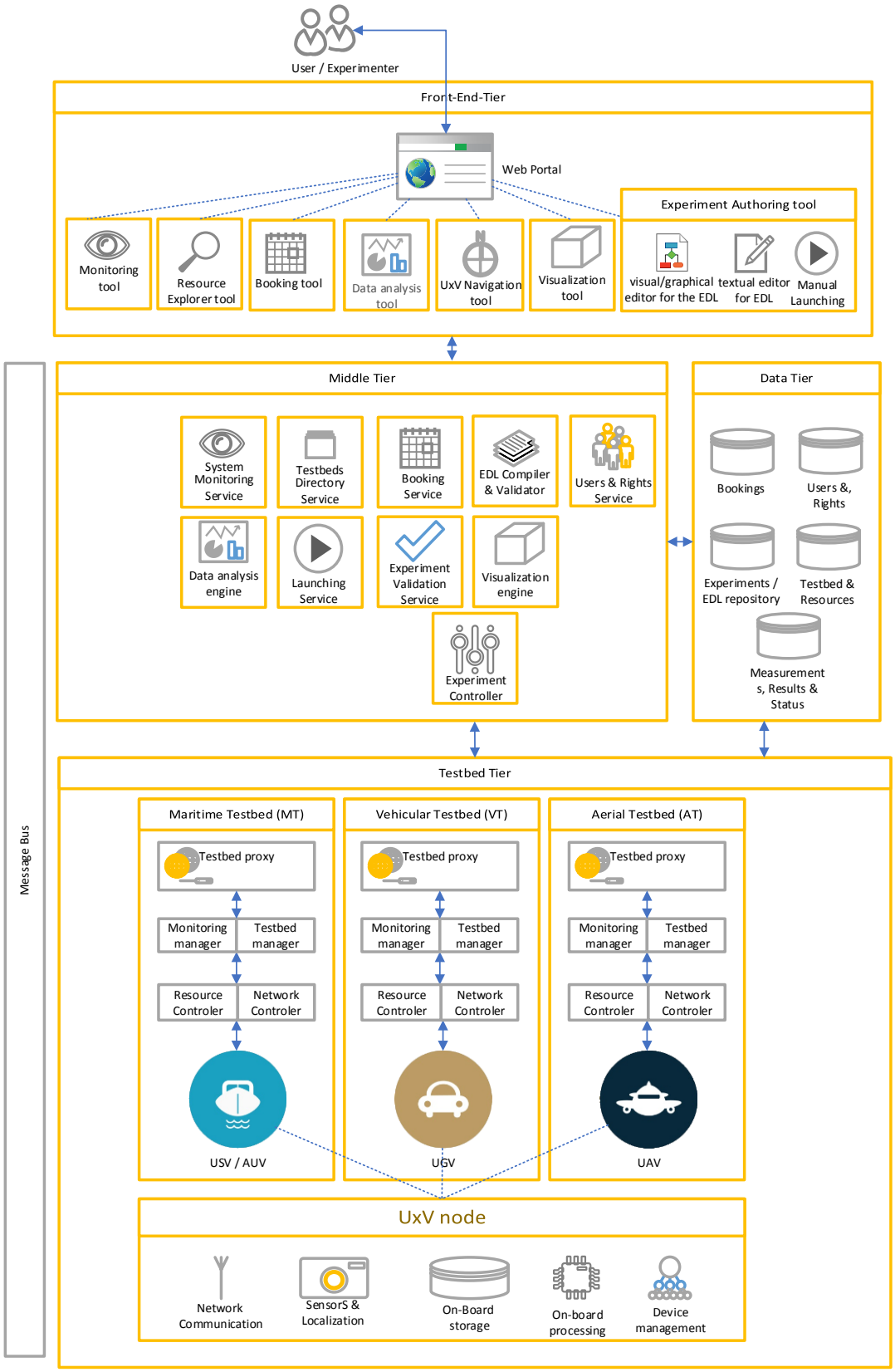


Figure 1 - Architecture diagram

The RAWFIE architecture consist of four tiers (see Figure 1):

- ***Front-end tier***: Providing a web based GUI that enables the user to interact with the RAWFIE system.
- ***Middle tier***: A collection of services and components that provide different management and processing functionalities. Middle tier entities should support deployment in cloud environment
- ***Data tier***: A collection of repositories that store the different data types generated and collected by RAWFIE
- ***Testbed tier***: The software and hardware components that are needed to run the testbeds and UxVs

Also RAWFIE will follow the Service Oriented Architecture [79] paradigm: All components should provide clearly defined interfaces, so that they can be easily accessed by other component or they may be easily replaced by other/better component with the same interface. The services can be described in languages such as Web Services Description Language (WSDL) [80]. Interacting with them is made possible by the use of remote service control protocols such as Simple Object Access Protocol (SOAP) [81] or the Representational State Transfer (and REST) resource invocation style, which are based on the popular HyperText Transfer Protocol (HTTP) These application protocols are relying on any communication system that supports HTTP, such as the Internet protocol stack (aka. IP or TCP/IP).

Additionally, a message-based middleware (via a Message Bus) will be used where suitable. This can provide a coherent communication model with distribution, replication, reliability, availability, redundancy, backup, consistency, and services across distributed heterogeneous systems. This Message Bus communication system will interconnect all components and all tiers. It can be used for asynchronous notifications and asynchronous method calls / response handling. As such, it may be used for transmitting measurements that will be routed from producers (e.g. UxVs) to the consumers pertaining to the Middle tier / Data tier (e.g. experiment monitoring, visualisation or data repositories).

3 Components overview

This chapter describes at a high level the components and the interactions between the components. Deliverable D4.2 will give a more detailed description of the components and interfaces. Chapter 5 provides representative use cases and corresponding sequence diagrams depicting interactions between the various components.

Component table

In the next sections, components will be described by using tables, according to the following template.

Component	Name of the component or subsystem
Responsible partner	The main responsible partner. Other may also be involved (may be added in parenthesis), but this partner has to coordinate the activities for this component.
Parent Component	None
Description	A short description of the component
Provided functionalities	List of functionalities and interfaces provides by this component
Relation to other components	How this component will interact with other components
Related user case sections	Use cases in which the component is involved. See chapter 4

Table 2: Template for components' description

3.1 Front end tier

Component	Web Portal
Responsible partner	Fraunhofer
Parent Component	None
Description	The central user interface that provides access to most of the RAWFIE tools/services and available documentation.
Provided functionalities	- Login and access control - Single sign on for each web tool - Linkage of all web tools
Relation to other components	- Provides a single point of access to the various RAWFIE Tools through a web GUI.
Related user case sections	- 4.1 User login, authentication and authorisation

Table 3: Web Portal

Component	Resource Explorer Tool
Responsible partner	Fraunhofer
Parent Component	Web Portal
Description	The experimenter can discover and select available testbeds as well as resources inside a testbed with this tool
Provided functionalities	- Visualize Data from the "Testbed, Resources" directory - Provide ability to search and select available resources inside a testbed
Relation to other components	- IN ← Testbeds Directory Service - OUT → Booking tool (send selected resources)
Related user case sections	- 4.3.1 Search for a resource

Table 4: Resource Explorer Too

Component	Booking Tool
Responsible partner	Fraunhofer
Parent Component	Web Portal
Description	The experimenter can discover and select available testbeds as well as resources inside a testbed with this tool
Provided functionalities	- Visualize the available dates and timeslots for each testbed resources (calendar view) - Select the preferred date, timeslot and/or space fragment in a testbed - Reserve the UxV resources for a specified time interval
Relation to other components	- IN/OUT ↔ Booking Service (existing bookings/new bookings)
Related user case sections	- 4.3.2 Book a resource

Table 5: Booking Tool

Component	Experiment Authoring Tool
Responsible partner	UOA
Parent Component	Web Portal
Description	This component is actually a collection of tools for defining experiments and authoring EDL scripts through RAWFIE web portal. It will provide features to handle resource requirements/configuration, location/topology information, task description e.t.c.
Provided functionalities	The supported functionalities are: - Experiment Definition Language (EDL) - Textual EDL editor (with syntax highlighting) - Visual EDL editor (describes script with graphical elements) - Textual and visual editors synchronization - Saving EDL scripts - Versioning of EDL scripts - Experiment validation - Manual Experiment launching
Relation to other components	The authoring tool will be connected with the respective components of the middle and data tiers. The use of EDL textual and visual editors will trigger EDL compiler and experiment validation backend services to perform syntactic and semantic analysis of the EDL scripts. The authoring tool will be connected with the launching service for scheduling the experiment executions. Moreover, this tool will interact with the EDL repository of the data tier in order to retrieve and/or store EDL scripts. - IN ← EDL Compiler and Validation

	- IN ←Experiment Validation Service - IN ← Launching Service - IN ← Experiment and EDL Repository - OUT → Textual and visual editor tools
Related user case sections	- 4.2 EDL editing

Table 6: Experiment Authoring Too

Component	**Experiment Monitoring Tool**
Responsible partner	Fraunhofer
Parent Component	Web Portal
Description	Shows the status of experiments and of the resources used by experiments.
Provided functionalities	- Show status of experiments (filtered by user rights) - Show status of resources (filtered by experiments & user rights)
Relation to other components	- IN ← Launching service (state of experiments) - IN ← Launching service (state of resources)
Related user case sections	- 4.4 Experiment launching and execution

Table 7: Experiment Monitoring Tool

Component	**System Monitoring Tool**
Responsible partner	Fraunhofer
Parent Component	Web Portal
Description	Shows the status and the readiness of the various RAWFIE services (mainly the ones residing in the middle tier)
Provided functionalities	- Show status of RAWFIE system infrastructure - Highlight potential problems
Relation to other components	- IN ← System Monitoring Service (state of middle tier infrastructure)
Related user case sections	- 4.8 monitoring

Table 8: System Monitoring Tool

Component	**UxV Navigation Tool**
Responsible partner	CERTH
Parent Component	Web Portal
Description	This component will provide to the user the ability to remotely navigate a squad of UxVs. Through a user friendly interface, the experimenter will specify the required details of the experiment, providing information regarding the number of the vehicles, the number of the

	units etc. Navigating an UxV is not an easy task and requires initial instructions and an extensive training to become proficient. 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. 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. In the sequel, these instructions are transmitted to the "Resource Controller" and sequentially, are translated, evaluated and delivered to the robots. When all the vehicles reach their desired position, the UxV Navigation Tool is ready to accept a new set of instructions. It is worth noting that in collaboration with the Monitoring tool, the component will inform the experimenters about the current position of the units, their sensor's measurements etc.
Provided functionalities	Experiments will have the ability to select the next desired location for each unit using one of the following interfaces: - A map of the area will illustrate the current position of each robot. Simply, by clicking on the map, the users define the next desired location. - Users will also have the option to manually navigate the robots by providing the coordinates of the next chosen position
Relation to other components	- OUT → Resource Control (transmitting the user's instructions) - OUT → System Monitoring Tool
Related user case sections	- 4.10 UxV remote control

Table 9: UxV Navigation Tool

Component	**Visualization Tool**
Responsible partner	EPSILON
Parent Component	Web Portal
Description	2D or/and 3D visualisation of the resources in an experiment
Provided functionalities	• (Real time) Geospatial data visualisation (WMS and WFS services) in 2D or/and 3D; • Show/track all moving UxV resources; • visually connect components and display relevant parameters through WPS service
Relation to other components	- IN ← Use the UxV resource/Data service - IN ← Visualization engine

Related user case sections	- 4.7 View visualization of running experiment

Table 10: Visualization Tool

Component	**Data Analysis Tool**
Responsible partner	HES-SO
Parent Component	Web Portal
Description	Starts data analysis learning tasks and displays their results.
Provided functionalities	- Starts data analysis processes - Visualizes data from the "Measurements, Results, Status" repository - Browses the results from past analysis - Provide commands to the Data Analysis Engine - Specifies data analytical/learning tasks to be executed on specific streaming datasets
Relation to other components	- IN ← Measurements, Results and Status repository - IN/OUT ↔ Data Analysis Engine (results/commands)
Related user case sections	- 4.6 Data analysis

Table 11: Data Analysis Tool

3.2 Middle Tier

Component	**EDL Compiler & Validator**
Responsible partner	UoA
Parent Component	None
Description	The EDL validator will be responsible for performing syntactic and semantic analysis on the provided EDL scripts. The validation will be performed on top of the proposed EDL model that will be based on a specific grammar. The EDL will give the opportunity to developers to define commands related to the experiments covering issues like *spatio-temporal instructions* to the UxVs, *communication*, *control*, sensing or *nodes and data management*. The validator will access the provided script and identify any semantic errors that could jeopardize the execution of the experiment. Specific constraints should be fulfilled when the experiment workflow is defined. These constraints will be continuously checked by the proposed authoring tool and in case some of them are validated to be false, the errors will be presented to the experimenters through various means (e.g., warnings). Finally, when no errors are present, the component will have the opportunity to generate specific files e.g., part of the final code to be uploaded in the UxVs, input to the validator, input to the testbed proxy).

Provided functionalities	Validated EDL scripts created either with the textual or the visual editor are based on the EDL grammar and a set of pre-defined rules (i.e., syntactically, regarding spatial and/or spatiotemporal availability of selected resources, control). The following list presents the functionalities offered by the validator: - 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 performs code generation in the appropriate format in order to be uploaded into the RAWFIE nodes.
Relation to other components	The validator will be connected with the provided editors as well as with components available in the data and the middle tier. The authoring tool will provide input to the validator in the form of an experiment workflow. The validator will retrieve the necessary data (e.g., EDL model, constraints, templates) stored in the data tier and will generate specific code blocks ready to be uploaded in the available nodes. The output of the validator will be adopted by a number of components like the Experiment Validation Service (EVS) or the Launching Service and the Experiment Controller. Moreover, the EDL validator will have access to the services provided in the data tier in order to store or retrieve parts or a whole experiment. Finally, specific parts of an experiment will be transferred to the testbed tier and, thus, the EDL validator will be combined with services available in the lower tier of the RAWFIE architecture. The following reports on the connection of the EDL Compiler & Validator with the remaining components of the RAWFIE architecture: - IN ← Experiment Authoring Tool - OUT → Experiments and EDL Repository - OUT → Experiment Validation Service - OUT → Experiment Controller - OUT → Testbed Proxy
Related user case sections	- 4.2 EDL editing

Table 12: EDL Compiler & Validator

Component	**Experiment Validation Service**
Responsible partner	UOA
Parent Component	None
Description	The Experiment Validation Service (EVS) will be responsible to validate every experiment as far as execution issues concern. This

	means that the EVS will validate if each experiment can efficiently be executed in the selected testbed. The aim is to have the RAWFIE following a pro-active approach through which the framework will be confident that an experiment will be executed without any problems. A number of constraints will be defined by experts that should be met during the experiment execution. Constraints will be related to the spatio-temporal aspect of the experiments. For instance, the EVS should check if during the execution of an experiment collisions are avoided and UxVs will efficiently fulfil their mission. For this, the routes of each UxV should be defined and possible collisions will be identified. This will stand either in terms of a single experiment or in terms of multiple experiments. Hence, RAWFIE will be capable of supporting the execution of multiple experiments running in parallel if, of course, there is availability of UxVs. 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.
Provided functionalities	The EVS aims to secure the qualitative and efficient execution of each experiment. Validated EDL scripts will be the input to the EVS and the result will be a set of possible errors that the experimenter should satisfy before the actual execution of the experiment. The following list presents the functionalities offered by the EVS: • It provides semantic validation of each experiment workflow for the specific testbed. • It checks the fulfilment of a set of constraints defined by experts for the specific testbed. • It is capable of retaining security issues e.g., collision avoidance, and the 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. • It performs cross experiment validation in order to help in maximizing the performance of RAWFIE framework.
Relation to other components	The EVS will be combined with the EDL validator receiving the experiment workflow as input. The EVS will result in a set of errors or will confirm the efficient execution of an experiment, information that will be adopted by other middle tier services (e.g., launching service, experiment control). Moreover, the EVS will have access to the services provided in the data tier in order to retrieve parts or a whole experiment. Finally, specific parts of an experiment will be transferred to the testbed tier and, thus, the EVS will be combined with services available in the lower tier of the RAWFIE architecture. The following reports on the connection of the EDL Compiler & Validator with the remaining components of the RAWFIE architecture: - IN ← EDL Compiler & Validator - IN ← Testbeds Directory Service

	- IN ← Testbeds and resources Repository - IN/OUT ↔ Experiments and EDL Repository - OUT → Experiment Authoring Tool
Related user case sections	- 4.2 EDL editing

Table 13: Experiment Validation Service

Component	**Users & Rights Service**
Responsible partner	Fraunhofer
Parent Component	None
Description	Manages all the users, roles and rights in the system.
Provided functionalities	- Check the authentication of uses - Authorization service (check if a user is allowed to do an specific action)
Relation to other components	- All components that need to check users authentication and authorizations
Related user case sections	- 4.1 User login, authentication and authorisation

Table 14: Users & Rights Service

Component	**Booking Service**
Responsible partner	Fraunhofer
Parent Component	None
Description	Manages bookings of resources
Provided functionalities	- Coordinate use of testbed resources among experimenters - Notification mechanisms (reminder for experiments) - Ensure fairness in resource bookings
Relation to other components	- IN/OUT ↔ Booking Service (new bookings/existing bookings) - IN/OUT ↔ Bookings Repository (existing bookings/new bookings)
Related user case sections	- 4.3.2 Book a resource

Table 15: Booking Service

Component	**Launching Service**
Responsible partner	UOA
Parent Component	None
Description	Schedules and launches executions of the experiments together with the assigned booked resources The Launching Service (LS) will be responsible for scheduling the execution of experiments. It will support two aspects of launching:

	(a) ***Short-term launching***: The LS through a specific interface will give the opportunity to experimenters to execute in real time pre-defined and pre-approved experiments stored in the RAWFIE system. It should be noted that this functionality will be available if the corresponding testbed is already configured (i.e., UxVs are in place and the necessary code is uploaded to nodes). The LS will take as input the experiment(s) and will execute them in sequential or in parallel according to the experimenters directions. (b) ***Long-term launching***: The LS will identify which experiment should be executed according to the available bookings. It should be noted, that the LS will execute only authorized and approved experiments based on spatio-temporal constraints.
Provided functionalities	The LS will provide the following functionalities: - It schedules experiment executions based on experimenters bookings. - It initiates the execution of an experiment or set of experiments real time or according to the scheduling - It schedules the sequential or the parallel execution of experiments. - It supports the real time execution of pre-defined and pre-approved experiments.
Relation to other components	The LS will have interaction with a number of components in the middle, data and testbed tiers. It will receive/retrieve instructions from experimenters through real time interaction or through bookings. Accordingly, it will send instructions to the testbed tier in order to secure the execution of an experiment. - IN ← Experiment Authoring Tool - IN ← Bookings Repository - OUT → Testbed Proxy - OUT → Experiment Controller
Related user case sections	- 4.4 Experiment launching and execution

Table 16: Launching Service

Component	**Experiment Controller**
Responsible partner	CERTH
Parent Component	None
Description	The Experiment Controller (EC) is a service placed in the Middle tier and is responsible to monitor the smooth execution of each experiment. The task of the EC is not on the control of the UxVs directly as this will be done through the Testbed Proxy. The main task is the monitoring of the experiment execution while acting as 'broker' between the experimenter and the resources in (near) real time. The EC will provides capabilities to support 'complex' experiments

	possibly involving multiple testbeds as well as to support the manual override of specific instructions to the resources while the experiment is running. The EC will identify if the experiment runs smoothly and will inform the upper layer in order to present the necessary information to the experimenter. In addition, the EC will control the data (raw or processed) sent back by the nodes. Hence, the EC, among others, will have access in the Data tier in order to be capable of retrieving the necessary data. The use of the EC in the middle tier gives RAWFIE the opportunity to include more intelligence in the functionalities provided related to the execution of the experiments and the level description to waypoints (.eg., implmeent patterns of vehicle movement like expanding ring).For instance, the system could have a view on the correct execution of the experiment workflow, to combine multiple UxV / Testbed types in the same experiment or to be able to monitor the execution of more complex scenarios.
Provided functionalities	• The EC monitors the course of actions during the experiments execution and informs the appropriate services in the Front-end layer. • It gains access to the Data tier in order to be capable of retrieving data that are going to be presented in the upper layer. • It forwards instructions from the experimenter to the resources for overriding the already defined experiment workflows.
Relation to other components	- IN ← Launching Service - IN/OUT ↔ Testbed Proxy - IN/OUT ↔ Measurements, Results & Status - OUT → System Monitoring Tool - OUT → Experiment Monitoring Tool
Related user case sections	- 4.4 Experiment launching and execution

Table 17: Experiment Controller

Component	**Data Analysis Engine**
Responsible partner	HES-SO
Parent Component	None
Description	Enables the execution of data processing jobs by sending requests to the processing engine (either stream processing engine, batch or micro-batch). Contains two major components: - Compute Engine: the implementation that distributes computations (eg: BLAS operations, etc). - Frontend: the portion that relays data to the compute engine. It also is responsible for requesting all available data (requirement: message bus schema registry).
Provided	- Requests available schemas from the Schema registry. This is a

functionalities	sub-component of the message bus. It is the portion that handles version invariance. - Requests the execution of a stream/batch processing job - Browses the results from the different analyses on the key-value cache system, which contains the results of the analysis. - Stores the results of the analysis on a NoSQL database store.
Relation to other components	- IN/OUT ↔ Data Analysis Tool - IN/OUT ← Measurements, Results and Status Repository (measurements/results)
Related user case sections	- 4.6 Data analysis

Table 18: Data Analysis Engine

Component	**System Monitoring Service**
Responsible partner	Fraunhofer (CERTH)
Parent Component	None
Description	Checks readiness of main components and ensure that all critical software modules will perform at optimum levels. Predefined actions are triggered whenever the corresponding conditions are met, or whenever thresholds are reached, or whenever an event or set of events are encountered, or the output of the previously mentioned operations is stored for reference or forwarded to another process
Provided functionalities	- Check the performance, utilizing Key Performance Indicators (KPI) - Run predefined action when triggers are reached - Send notifications (e.g. via email) - Capture alarms caused by malfuction or underperformance of the equipment.
Relation to other components	- IN ← Status and performance values from all middle tier components - OUT → System Monitoring Tool - OUT → some component (preforming a predefined action) - OUT → some messaging system (user notifications e.g. via email)
Related user case sections	- 4.8 System monitoring

Table 19: System Monitoring Service

Component	**Testbeds Directory Service**
Responsible partner	IES
Parent Component	None
Description	Represents a registry service of the middleware tier where all the

	integrated testbeds and resources accessible from the federated facilities are listed, belonging to the RAWFIE federation. This service will be the actual software interface for the Testbed Directory, that will include information relevant to the testbeds and possibly their resources (location, facilities) as well information on the capabilities of a particular resource and its requirements for executing experiments e.g. in terms of interconnectivity or dependencies
Provided functionalities	Provides an Application Programming Interface (API), implemented using standard architectural styles and protocols such as REST or SOAP Web Services. This API allows other components to get access to the information contained in the corresponding repository (Testbeds & Resources). Provides the pointers to the different testbeds belonging to the RAWFIE federation. In particular, using the provided software API it will be possible to: - Populate the Testbeds & Resources Repository - Look at the available testbeds list, and at their status (free, booked, in use, and so on) - Look at the available resources within a given testbed, and at their status (free, booked, in use, not operational, and so on) - Look at the description and characteristics of the testbeds (name, location, available resources) - Look at the description and characteristics of each resource in a given testbed - Look at the testbeds and resources capabilities in terms of available technologies and tests
Relation to other components	- IN/OUT ↔ Resource Explorer Tool The component API is invoked by the Resource Explorer tool to list the available testbeds and, for each testbed, the available resources. This service will be accessed whenever an experimenter wants to retrieve information related to available testbeds and resources using the respective front-end tool. - IN/OUT ↔ Testbeds & Resources Repository The provided API, which will be in charge of executing the queries to the Testbeds & Resources Repository, could also allow advanced searching capabilities based on specific filters.
Related user case sections	- 4.3.1 Search for a resource

Table 20: Testbeds Directory Service

Component	**Message Bus**
Responsible partner	IES
Parent Component	None
Description	Message Oriented Middleware used across all tiers to enable

	asynchronous, event-based messaging between heterogeneous components. Implements the Publish/Subscribe paradigm. Different message brokers implementations and protocols for data formatting and messaging will be investigated, these include: - ActiveMQ - RabbitMQ - Apache Kafka - MQTT One or more of the above solutions are expected to be part of the RAWFIE architecture, according to the requirements that will identified for each specific communication scenarios, ranging from the need to publish UxVs measurements to the software components running on the upper layers, to the communication between the upper layers software components themselves. Some information about the abovementioned message brokers implementation and protocols are provided in Section 5.2.9.
Provided functionalities	- Send asynchronous notifications on specific events (e.g. booking notifications) - Handle Publisher/Subscriber (or Publisher/Consumer) relationships between components - Possibility to buffer messages persistently, to ensure delivery of messages even in case of network or system fault - Ability to handle messages sent at various different revisions: This prevents consumers subscribed to previous revisions from having their components break. This allows for producer side addition/modification of new/existing fields (correspondingly) while not breaking consumer processes. This is added as a general concept in the architecture as 'Schema Registry'
Relation to other components	Different components are involved in the communication through the Message Bus. Possible communication scenarios include the ones described in the following IN ← - the Resource Manager in the Testbed Tier publishes information on measurements through the message broker in the Monitoring & Testbed Manager - the Resource Manager tracks information on the position of the UxVs while the test is running, and publish them for the 3D Visualisation Engine / Tool (for final visualisation of the resources position on the Web Portal) OUT → - the Resource Manager in the Testbed Tier gets measurements or other information about the status of a specific resource (UxV) Communication between different UxVs while an experiment is running - the Monitoring Service / Tool consume information on measurements published by the Resource Manager

	- the 3D Visualisation Engine / Tool consume information about the positions of resources (resources tracks), for final visualisation on the Web Portal
Related user case sections	Use case diagrams related to the Message Bus will be detailed in next WP4 deliverables

Table 21: Message Bus

3.3 Data tier

Component	**Testbeds & Resources Repository**
Responsible partner	IES
Parent Component	None
Description	The Testbed and Resources Repository contains relevant information about available testbeds (federated through the RAWFIE platform) and their resources , such as: - Testbed name and testbed URL (if a dedicated access portal is also available for a specific testbed) - Description and overview of each testbed facilities, and corresponding resources (e.g. available UxVs) - Overview of the reservations linked to each specific testbed (through the relationship with the Booking Directory) - Description and overview of specific resources (e.g. type, technologies, tests that can be executed, and so on) for each given testbed - Information on the capabilities of a particular resource and its requirements for executing experiments e.g. in terms of interconnectivity or dependencies
Provided functionalities	For each testbed at least the following information shall be available: - Its name - Its location - A short description (possibly mentioning guidelines applying to the testbed usage) - Type of resource(s) available - Total number of resources available - Total number of resources in use - List of resources with an indication as “available” or “booked” - EDL control capabilities supported - Connectivity status
Relation to other components	- IN/OUT ↔ Testbeds Directory Service
Related user case sections	- 4.3.1 Search for a resource

Table 22: Testbeds & Resources Repository

Component	Experiments & EDL Repository
Responsible partner	UoA
Parent Component	None
Description	The Experiments and EDL Repository (EER) provides the necessary functionalities for having the experiments and EDL related data stored in to the data tier. The EDL scripts, templates and pre-defined constraints will be stored in the appropriate format in order to be efficiently retrieved by the rest component of the RAWFIE framework. It should be noted that the appropriate metadata will be adopted for each experiment. The access to the EER will be done through interoperable interfaces ensuring the compatibility and interoperability with other components of the architecture.
Provided functionalities	The EEE functionalities are as follows: - It provides functionalities for searching, retrieving, storing and updating of EDL scripts. - It supports versioning of the available experiments.
Relation to other components	The EER will be mainly combined with the components related to the middle tier responsible for handling the experiment workflows.
Related user case sections	- 4.2 EDL editing

Table 23: Experiments & EDL Repository

Component	Bookings Repository
Responsible partner	Fraunhofer
Parent Component	None
Description	Stores bookings and reservations of resources
Provided functionalities	- Store time and resources that are reserved by an experimenter
Relation to other components	- IN/OUT ↔ Booking Service
Related user case sections	- 4.3.2 Book a resource

Table 24: Bookings Repository

Component	Measurements, Results & Status Repository
Responsible partner	HES-SO
Parent Component	None
Description	Data Stored includes: - State of experiment executions - Raw measurements collected during an experiment - Results of data analyses of measurements

Provided functionalities	- Load & store
Relation to other components	- IN ← UxV (via Experiment Controller) (store measurements) - IN/OUT ↔ Data Analysis Engine (results/measurements)
Related user case sections	- 4.6 Data analysis

Table 25: Measurements, Results & Status Repository

Component	**Users & Rights Repository**
Responsible partner	Fraunhofer
Parent Component	None
Description	Management of authorizations and access rights. Will probably use a LDAP server.
Provided functionalities	- LDAP interface
Relation to other components	- IN/OUT ↔ Users & Rights Service
Related user case sections	- 4.1 User login, authentication and authorisation

Table 26: Users & Rights Repository

3.4 Testbed tier

Component	**Testbed Proxy**
Responsible partner	UOA
Parent Component	None
Description	Handles the communication between the testbed facility and the rest tiers of RAWFIE architecture. It lies on the server side of each RAWFIE compliant testbed facility. Several instances of Testbed proxy can run at the same time in a RAWFIE testbed facility.
Provided functionalities	- Ensures communication with Middle Tier - Ensures communication with Data Tier

Relation to other components	- IN/OUT ↔
	  ○ Monitoring Service
	  ○ Resource Directory Service
	  ○ Experiment Controller
	- OUT →
	  ○ IP addresses of TP of the integrated testbed are registered to Testbed Directory.
	  ○ Testbed and Resources Repository
	  ○ Measurements Results and Status

Related user case sections4	- 4.3 Resource booking and reservation - 4.4 Experiment launching and execution - 4.5 Measurements recording - 4.9 Testbed monitoring

Table 27: Testbed Proxy

Component	**Testbed Manager**
Responsible partner	UOA
Parent Component	None
Description	Contains accumulated information from the experiments and the devices in the testbed. It is responsible to address initial testbed registration and periodic updates of testbed on-going experiments. This information is accessed by the relevant middle tier service.
Provided functionalities	- Registers the testbed to the middle tier - Contains the registration log for the experiments in the tested - Periodically checks the status of the experiments - Forwards the status of the experiments to Data Repository and to Monitoring Tool - Stores 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. For example if there is a network problem during the execution of the experiments, TM stores the information that would be forwarded to the Data tier.
Relation to other components	- IN/OUT ↔ Testbed Proxy - IN/OUT ↔ Experiment Controller
Related user case sections	- 4.3 Resource booking and reservation - 4.4 Experiment launching and execution - 4.5 Measurements recording - 4.9 Testbed monitoring

Table 28: Testbed Manager

Component	**Monitoring Manager**
Responsible partner	UOA
Parent Component	None
Description	Monitors the status of the testbed and the devices belonging to it, at functional level, i.e the ‘health of the devices’ and current activity.
Provided functionalities	- 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. - Stores the status of the testbed characteristics and the devices in a data log.
Relation to other components	IN/OUT ↔ Testbed Proxy IN/OUT ↔ Data Tier
Related user case sections	- 4.3 Resource booking and reservation - 4.9 Testbed monitoring

Table 29: Monitoring Manager

Component	**Network Controller**
Responsible partner	Uoa
Parent Component	None
Description	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.
Provided functionalities	- Interfaces a local authorization module for allowing direct booking and executing RAWFIE compliant experiments - Provisioning of the network connections/technologies required during an experiment - Checks the communication when devices are moving between obstacles
Relation to other components	IN/OUT ↔ Testbed Proxy IN/OUT ↔ Experiment Controller
Related user case sections	- 4.4 Experiment launching and execution - 4.9 Testbed monitoring - 4.10 UxV remote control

Table 30: Network Controller

Component	**Resource Controller**
Responsible partner	CERTH
Parent Component	None
Description	The core component of the navigation system is the “Resource Controller” which ensures the safe and accurate guidance of the UxVs. RC commands each device to switch onboard sensors on and off At the same time, Resource Controller informs the “Monitoring Tool” and

	Data Tier for the gathered measurements of the sensors of each device. "Launching Tool" interacts with the "Experiment Controller" so as to transfer user's preferences and instructions regarding the experiment. The "Experiment Controller" initially, triggers the "Experiments and EDL Repository" component and receives the user's directions, translated in a form of a set of waypoints. These waypoints provide basic information about the preferable locations for each UxV. The set of the waypoints for each robot defines the path that the experimenters have shaped. For the navigation of a robot from its current position to the location described by the next waypoint, the system requires a turn. The main objective of the "Resource Controller" component is to optimize the navigation process which takes place during a turn. Resource Controller component navigates simultaneously all the units of the squad. It is worth noting that the time needed for each robot to reach its desired location is not the same for all units. Thus, the turn concludes when all the robots reach their next location. Additionally, it is worth mentioning that in case of emergence, the "Resource Controller" collaborates with the "Testbed Proxy" so as to navigate the units back to a safe position, as soon as possible.
Provided functionalities	- The calculation the near-optimal path that the vehicles should follow in order to reach the desired location. - The translation of the operator's/experiment instructions into a reference scheme, compatible with the "Testbed Proxy". - The assurance that the system is performing as intended and that the equipment is safe. - Publish sensor values to the Data Tier/ Monitoring Tool
Relation to other components	- IN ← Launching Service - IN/OUT ↔Testbed Proxy - IN/OUT ↔Monitoring Tool - IN/OUT ↔Data Tier - IN/OUT ↔Experiment Controller
Related user case sections	- 4.4 Experiment launching and execution (UOA and CERTH) - 4.11 UxV remote control (CERTH)

Table 31: Resource Controller

Component	**Navigation Service**
Responsible partner	CERTH
Parent Component	Resource Controller
Description	The main objective of the "Navigation Service" component is to optimize the navigation process which takes place during a turn. - The optimization algorithm will be based on the Cognitive-based Adaptive Optimization (CAO) approach. CAO transforms the navigation problem into an optimization one, which in every

	time step the goal is to optimize the location of the UxVs so to meet the objectives of the mission with respect to a set of constraints.
Provided functionalities	- Validate the next candidate position for each vehicle
Relation to other components	• IN/OUT -> Resource Controller
Related user case sections	- 4.4 Experiment launching and execution (UOA and CERTH) - 4.11 UxV remote control (CERTH)

Table 32: Navigation Service

Component	**UxV node**
Responsible partner	CSEM (MST, Robotnik)
Parent Component	None
Description	A single UxV node. The UxV is a complete mobile system that interacts with the other Testbed entities. It can be remotely controlled or able to act and move autonomously
Provided functionalities	According to [Brugali 07] the UxV can be decomposed into four groups (hardware interfaces, information processing, vehicle control, decision making). They include the following unordered and non-exhaustive list of functions and services: - Physical interfaces to vehicle actuators and sensors Network connection - Data acquisition - Data storage - Data pre-processing (aggregation, fusion, etc.) - Data management, representation, transfer, etc. - Local time reference and time stamping service - Location reference and geo-tagging service (location retrieval, coordinate management) - Navigation and autonomous control (involves an internal representation of its environment, map, location awareness, path planning, obstacle avoidance, waypoint management, hazard management), decision-making service. - Remote control interface, Human Robot Interaction [Goodrich 07] - Status of the UxV (attitude-inertial measurement unit, energy, speed, sanity, mode, etc.) - Identification, transponding, friend or foe Payload status - Etc. The specific component that allows the UxV for interacting with the Testbed and its constituents is making use of several of the above function and services. It will feed the Testbed experiment database with collected data, recorded events, flight information, etc.) and it will be

	fed with the instructions and commands corresponding to the mission it is assigned to in the context of the experiment. It may offer a relay platform for other Testbed components to transfer data to the ground control and experiment control.
Relation to other components	- IN/OUT ↔ Resource Controller
Related user case sections	- 4.4 Experiment launching and execution (UOA and CERTH) - 4.9 Testbed monitoring - 4.11 UxV remote control (CERTH)

Table 33: UxV node

Component	**UxV - Network communication**
Responsible partner	CSEM
Parent Component	UxV node
Description	Provides communication services to the UxV These services form the basis for the other services to interact with the UxV (basically all features listed in the UxV node.
Provided functionalities	- Identification service - Data transfer service - Status notification - Capabilities and directory services - Reconfiguration
Relation to other components	- IN/OUT ↔ Network controller - IN/OUT ↔ Experiment controller - IN/OUT ↔ System monitoring
Related user case sections	- 4.4 Experiment launching and execution (UOA and CERTH) - 4.9 Testbed monitoring - 4.11 UxV remote control (CERTH)

Table 34: UxV - Network communication

Component	**UxV – Sensors & Localization**
Responsible partner	CSEM (MST, Robotnik)
Parent Component	UxV node, fixed testbed node
Description	Provides interfaces todifferent s installed on the UxVsensor
Provided functionalities	Sensors are providing the application with measurement points, typically tuples made of a location a timestamp, a source sensor and one or several samplings. Localisation is a specific type of measurement using positioning systems or a combination of measurements to estimate a location. - Estimated position of the UxV and collected data - Sensors (fixed and mountable) - Raw data acquisition

Relation to other components	- OUT → On board processing - OUT → On board storage (buffering) - OUT → Measurements, Results &Status Repository - IN/OUT ↔ Experiment controller
Related user case sections	- 4.4 Experiment launching and execution (UOA and CERTH) - 4.9 Testbed monitoring - 4.11 UxV remote control (CERTH)

Table 35: UxV – Sensors & Localization

Component	**UxV – On board storage**
Responsible partner	CSEM (MST, Robotnik)
Parent Component	UxV node
Description	Provides storage of data inside the UxV
Provided functionalities	- The UxV embeds some storage to store data. Typically, the data corresponds to measurements that cannot be sent over the communication link to the testbed manager - Status UxV information produced during an experiment will be internally stored for later UxV maintenance
Relation to other components	- IN ← UxV – Sensors & Localization - IN/OUT ↔ Network controller - OUT ↔ Resource controller - IN/OUT ↔ Measurements, Results & Status Reposotiry
Related user case sections	- 4.4 Experiment launching and execution (UOA and CERTH) - 4.9 Testbed monitoring - 4.11 UxV remote control (CERTH)

Table 36: UxV – On board storage

Component	**UxV – On board processing**
Responsible partner	CSEM (MST, Robotnik)
Parent Component	UxV node
Description	Provides processing of data inside the UxV
Provided functionalities	- 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, etc.
Relation to other components	- IN ← UxV – Sensors & Localization - IN/OUT ↔ Network controller - IN/OUT ↔ Experiment controller - IN/OUT ↔ Measurements, results & status
Related user case	- 4.4 Experiment launching and execution (UOA and CERTH)

sections	- 4.9 Testbed monitoring - 4.11 UxV remote control (CERTH)

Table 37: UxV – On board processing

Component	**UxV – Device management**
Responsible partner	CSEM
Parent Component	UxV node
Description	Provides network management for the UxV
Provided functionalities	- Network connection to the base state - Ad-hoc networks - Device sensor controlling
Relation to other components	- IN/OUT ↔ Network controller - OUT ↔ Resource controller
Related user case sections	- 4.4 Experiment launching and execution (UOA and CERTH) - 4.9 Testbed monitoring - 4.11 UxV remote control (CERTH)

Table 38: UxV – Device management

3.5 Requirement mapping

The following table shows an overview which requirements (from D3.1) can be mapped to a component. Please note, that a couple of requirements were not mapped to the architecture in this first phase. This refers to requirements PT-B-007 and TB-G-009 which are expected to be handled in the second version of the architecture.

Components	Functional Requirements	Non Functional Requirements
Global		PT-NF-008, PT-NF-007, PT-NF-006, PT-NF-004, PT-NF-009, PT-NF-012
Front end tier		PT-NF-001
Web Portal	PT-GEN-001, PT-GEN-002, PT-P-002	
Resource Explorer Tool	PT-P-001, PT-A-016	
Booking Tool	PT-A-016, PT-B-001, PT-B-002, PT-B-003, PT-B-005, PT-L-002	PT-NF-002
Experiment Authoring Tool	PT-A-001, PT-A-002, PT-A-004, PT-A-005, PT-A-006, PT-A-007, PT-A-008, PT-A-009, PT-A-010, PT-A-011, PT-A-012, PT-A-013, PT-A-015, PT-L-010	
Experiment Monitoring Tool	PT-L-001, PT-L-003, PT-L-004	PT-NF-005
System Monitoring Tool	PT-GEN-004	
UxV Navigation Tool	PT-L-008, PT-L-009	
Visualization Tool	PT-A-013, PT-L-006	
Data Analysis Tool	PT-E-003	
Middle tier		PT-NF-001, PT-NF-010, PT-NF-011
EDL Compiler & Validator	PT-A-003, PT-A-014	
Experiment Validation Service	PT-A-009, PT-L-001	
Users & Rights Service	PT-GEN-002	PT-NF-002
Booking Service	PT-B-003, PT-B-005, PT-B-006, PT-B-004	PT-NF-002
Launching Service	PT-L-002, PT-L-003, PT-L-007, PT-E-001	
Experiment Controller	PT-A-005, PT-A-008, PT-L-008, PT-A-010, PT-L-	PT-NF-005

	009	
Visualization Engine	PT-L-005	
Data Analysis Engine	PT-E-002, PT-E-005	
System Monitoring Service	PT-GEN-004	
Testbed Directory Service	PT-P-004	
Message Bus		
Data tier	PT-E-004	PT-NF-001, PT-NF-003, PT-NF-010
Testbeds & Resources Repository	PT-P-003	
Experiments and EDL Repository	PT-P-005, PT-A-015, PT-E-001	
Bookings Repository	PT-A-015	
Measurements, Results and Status Repository	PT-A-007, PT-E-002	
Users & Rights Repository	PT-GEN-002	
Testbed tier		PT-NF-001
Testbed Proxy		
Testbed Manager	PT-P-003	
Monitoring Manager		
Network Controller		
Resource Controller		PT-NF-005
Navigation Service		
UxV node	PT-A-010, PT-E-002	PT-NF-005
UxV - Network communication		
UxV – Sensors & Localization		
UxV – On board storage		
UxV – On board processing		
UxV – Device management		

Table 39: Allocation of Platform Requirements to Architecture Components

Components	Functional Requirements	Non Functional Requirements
Testbed tier		TB-NF-G-002, TB-NF-G-005
Testbed Proxy	TB-G-002	TB-NF-G-001, TB-NF-G-003, TB-NF-G-004
Testbed Manager	TB-G-003, TB-G-004, TB-G-005, TB-G-006, TB-G-007, TB-I-001, TB-I-004, TB-R-005, TB-R-006, TB-D-001, TB-D-002	
Monitoring Manager	TB-G-001	
Network Controller	TB-I-002, TB-I-003	TB-NF-G-004, TB-NF-G-003, TB-NF-G-006
Resource Controller	TB-G-003, TB-G-007	
Navigation Service	TB-G-008	
UxV node	TB-G-004, TB-R-001, TB-R-002, TB-R-003, TB-R-005, TB-R-006, TB-R-007, TB-R-008, TB-R-009, TB-R-010, TB-R-012, TB-R-013	TB-NF-R-001, TB-NF-R-003
UxV - Network communication	TB-G-003, TB-I-002, TB-I-003, TB-R-013	TB-NF-G-006
UxV – Sensors & Localization	TB-G-005	TB-NF-R-002
UxV – On board storage	TB-R-004	
UxV – On board processing		
UxV – Device management	TB-R-011	

Table 7: Allocation of Testbed Requirements to Architecture Components

4 Potential use cases and sequence diagrams

In the following section, some common use cases (or user stories) are described and the collaboration between the different components is examined. The use case consists of a description/analysis followed by a sequence diagram that visualizes the interactions between the components.

4.1 User login, authentication and authorisation

RAWFIE will support two user login mechanisms:

- Password-based:
  - This is the common way to authentication a user by requesting username and password. SSO will be available.
- Certificate-based
  - Using X.509 client certificates to user is automatically authenticated during the SSL handshake. (see section)

The communication between the components themselves will be secured using SSL on the transport layer and X.509 client certificates, to control access to the services. For this purpose the root certificate of the RAWFIE CA needs to be installed in every component and each component will get its own signed client certificate (plus private key) to authenticate itself.

The "Users & Rights Service" will be contacted by all other components to check if a user has the appropriate rights/roles to use them as well as to check whether a user is allowed to access or edit a resource.[4]

4.1.1 Password-based user login

- An user opens via its browser an application of the RAWFIE web page and requests a restricted resource (URL)
- The application checks if the user is locally logged-in (e.g. via cookie for this application)
- If not
  - Redirect to SSO page
  - The SSO page checks if the user is globally logged in
  - If not
    - The user is asked for credentials (username and password)
    - The SSO page sends the credentials to the User & Rights Service

[4] The RAWFIE internal access control between the components will be skipped in the most sequence diagrams, as it is more or less a background process.

        - The User & Rights Service checks the credentials and returns whether they are OK
    - The SSO page redirects to the original web page (with some login token as parameter)
    - The user requests the original web page again (with some login token as parameter)
    - The application checks the login token and creates a user session (e.g. transmitted via an cookie)
- Proceed with “Check user authorisation”

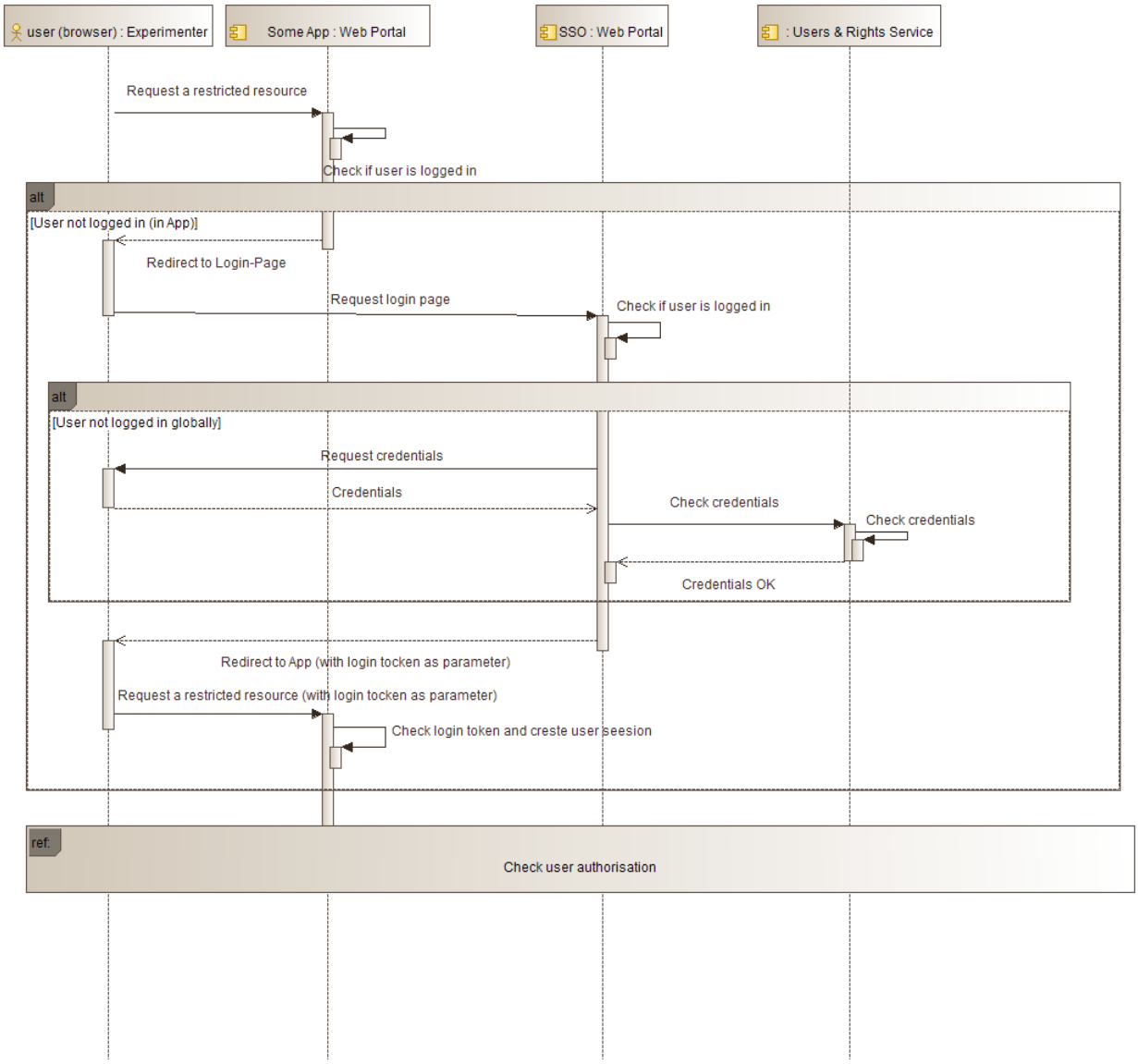


Figure 2 - Sequence Diagram – Password based user login

4.1.2 X.509 Certificate-based user login

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

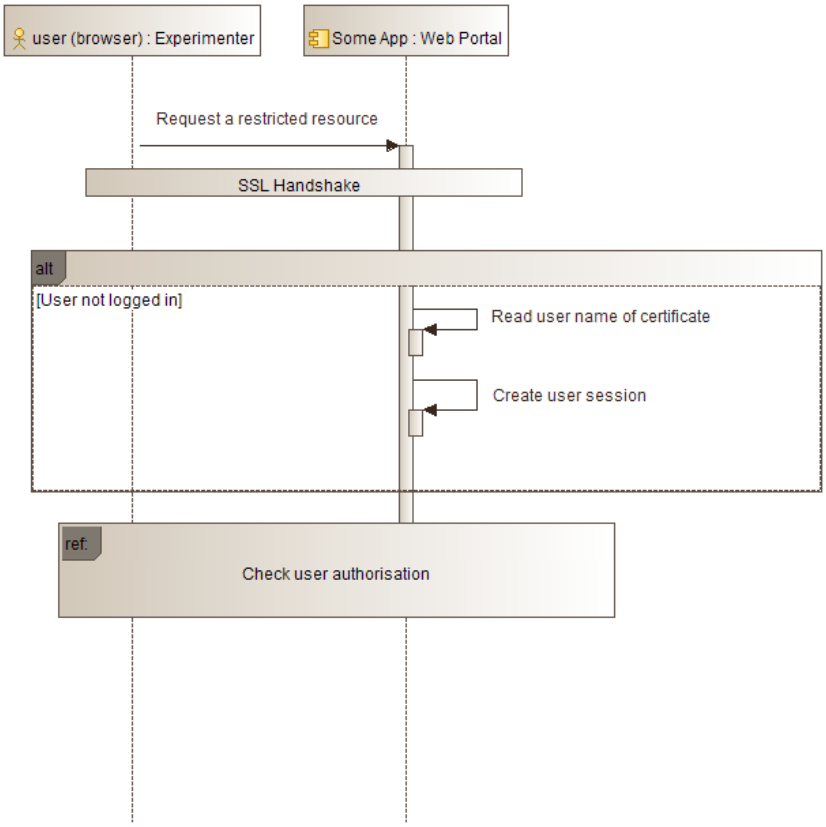


Figure 3 - Sequence Diagram - Certificate-based user login

4.1.3 Check user authorisation

- After the user has logged in and has requested a restricted resource, the web application checks if user is allowed to see the resource
- Component request the Users & Rights Service if there given user has the specific role/right to see/edit this resource
- The Users & Rights Service does…
  - Check if the user exits
  - Get groups of the user

- Check if the role members contains the user or one of the groups of the user
- If ok: grant access to the user
- If wrong: show access denied to the user.

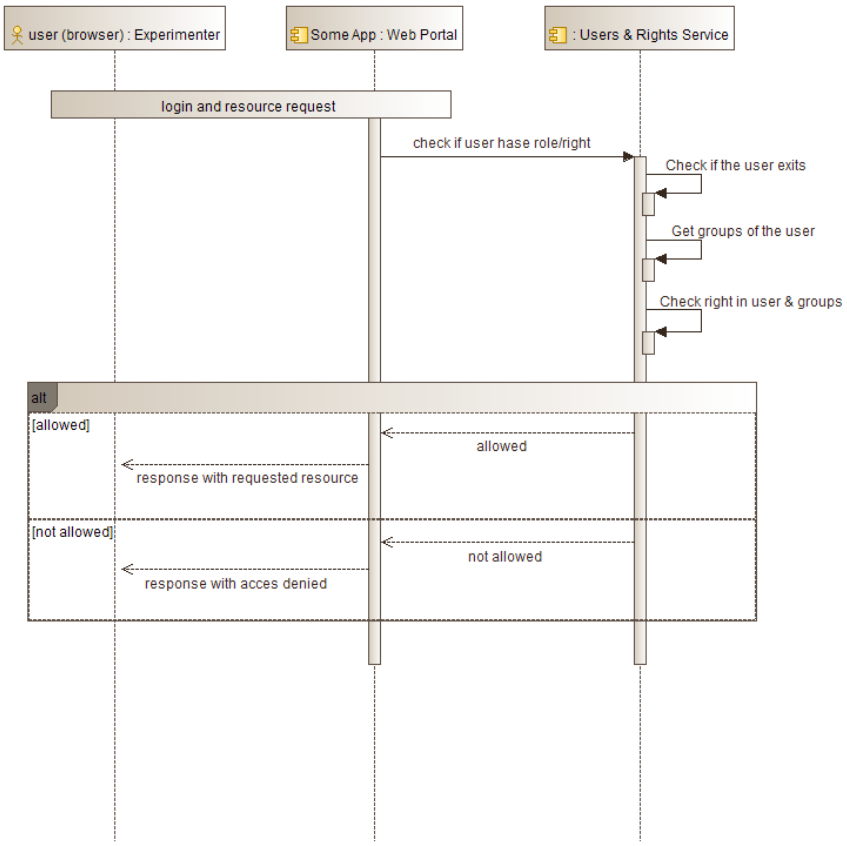


Figure 4 - Sequence Diagram - Check user authorisation

4.1.4 Trusted and secure communication between the components

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

- component A calls service of another component B
- Transport layer: SSL handshake with client and server certificates (on error close connection)
- If there is a need to verify the authorisation
  - checks the certificate of the component A and reads the component name out of the certificate
  - Component B calls Users & Rights Service to check if component A or the user that has initiated the whole process has the needed roles/rights
  - Transport layer: SSL handshake with client and server certificates (on error close connection)

    - The Users & Rights Service
        - checks the certificate of the component B and reads the component name out of the certificate and..
        - checks if component B is allowed to read permissions
        - checks if component A or the user has the needed roles/rights
        - Returns the result (allowed/not allowed)
- If allowed
    - component B executes the service method
    - returns the result to component A
- If not allowed
    - component B returns “access denied” to component A

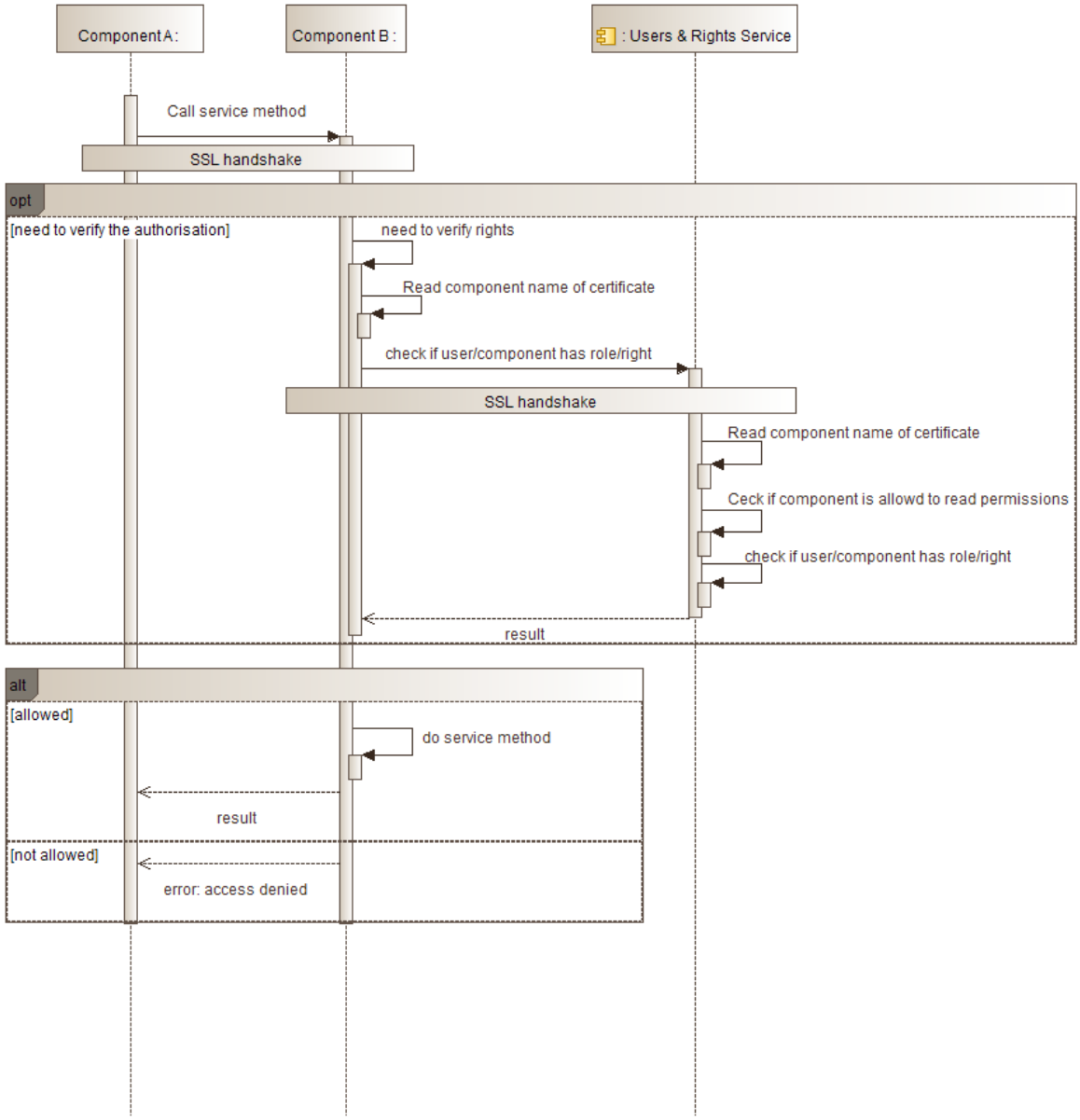


Figure 5 - Sequence Diagram - communication between components

4.2 EDL editing

The EDL editing use case can be de-composed in two main functionalities Create and Validate the EDL scripts. The following paragraphs elaborate on the actions to be performed. Basic operations are also depicted in the sequence diagram. More specifically, the validation of the EDL is performed by a two-phase validation mechanism

- Create an EDL script
    - Open textual and visual editors

- Use functionalities simultaneously from both editors
  - Synchronization is supported
- Create a new EDL script to define an experiment or
- Edit a saved EDL script
- Save changes

- Validate EDL script
  - Experimenter validates the EDL script by using the EDL Complier and Validator component
  - EDL Compiler and Validator retrieves EDL model from the Experiment and EDL Repository in order to apply respectively the validation operations.
  - The EDL validator returns syntactic and semantic errors to the experimenter
  - Experimenter corrects the errors
  - Experimenter starts the Experiment Validation Service (EVS) to validate the defined experiment in terms of execution efficiency and spatiotemporal issues
  - EVS validates the experiment according to specific rules and constraints
  - Semantic errors are displayed to the Experimenter
  - Experimenter corrects the errors
  - The executed EDL script is stored to the Experiment EDL Repository

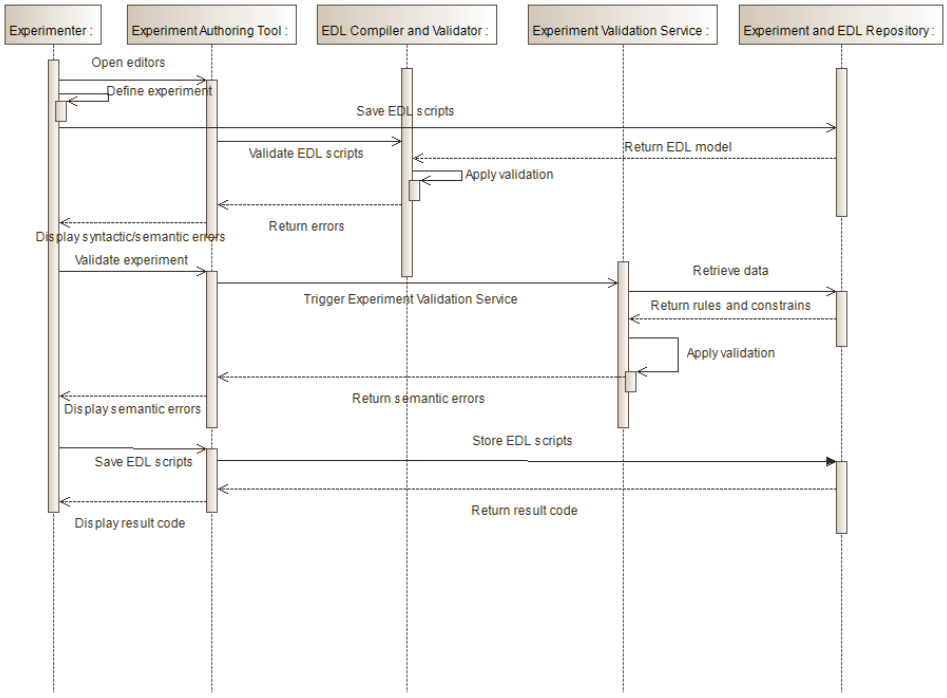


Figure 6 - Sequence Diagram - EDL editing

4.3 Resource booking and reservation

To find the appropriate resources (UxVs, testbeds) of his experimenter, an experimenter can first search for a resource and following he starts the booking of its.

4.3.1 Search for a resource

To reserve a testbed or just some UxVs the experimenter/user needs to search for appropriate resources (testbed, UxVs) via the RAWFIE web portal.

- The user opens the Resource Explorer Tool
- The Resource Explorer Tool loads the available testbeds' info from the Testbeds & Resources Repository
- Resource Explorer Tool returns the list of testbeds to the user
- The user selects a testbed and opens the UxV view of the selected testbed
- The Resource Explorer Tool loads the available UxVs of the selected testbed from the Testbeds & Resources Repository
- Resource Explorer Tool returns the list of UxVs to the user
- The user selects some UxVs and wants to start the booking
- Resource Explorer Tool sends a redirect to the booking tool containing the selected resource IDs.
- The browser of the user follows the redirect and opens the booking tool (with the selected resource IDs)

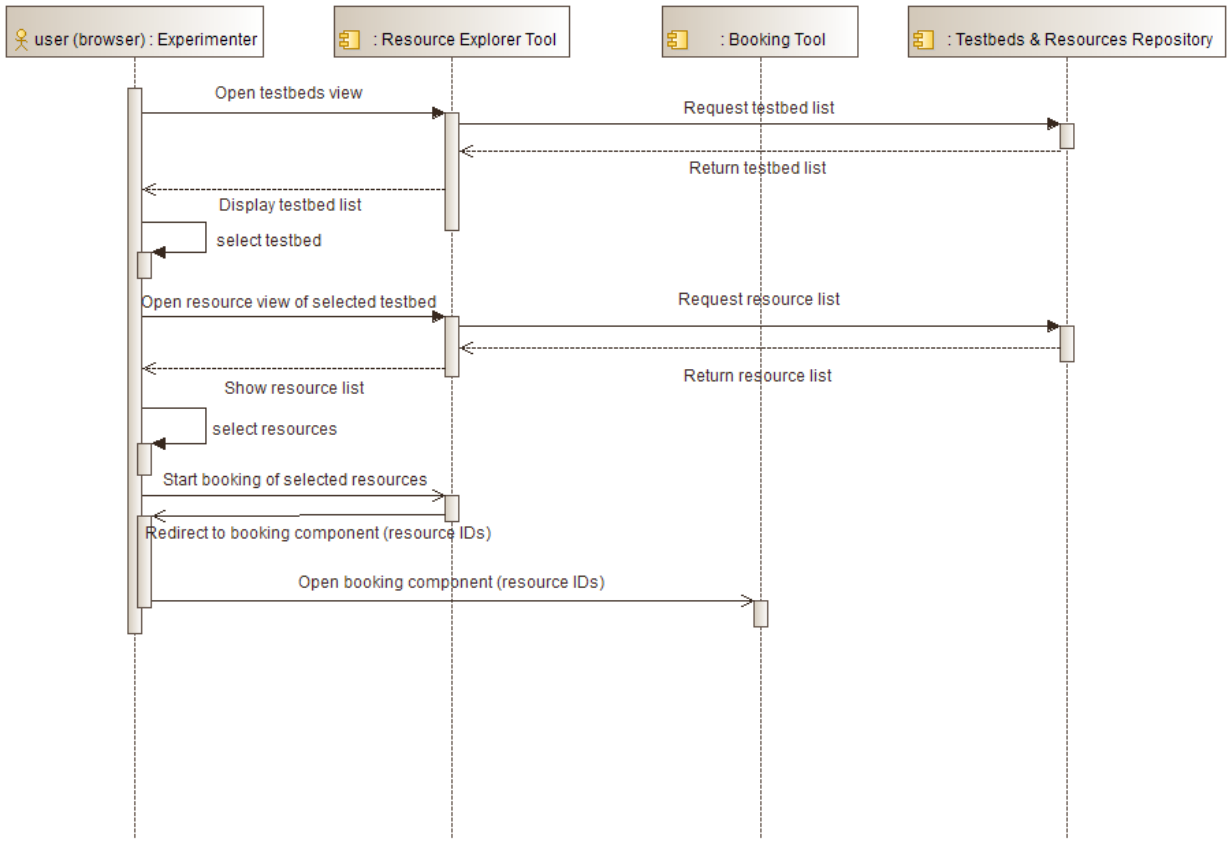


Figure 7 - Sequence Diagram - Search for resource, select one and start booking

4.3.2 Book a resource

- Booking Tool starts with resources of interest
- Show calendar view with current bookings of the resources of interest
- User starts the booking of the resources (selects “New booking“ from the UI)
- Show booking form
- User enters data (name, date, time, comments) and submits the form
- Booking data sent to Booking Service
- Booking Service loads all the bookings of the resource in the given time frame
- Check if there are any booking conflicts
- On OK: save booking and return ok
    - Save the bookings
    - Show success to user
- On conflict: return error
    - Show conflict overview to user

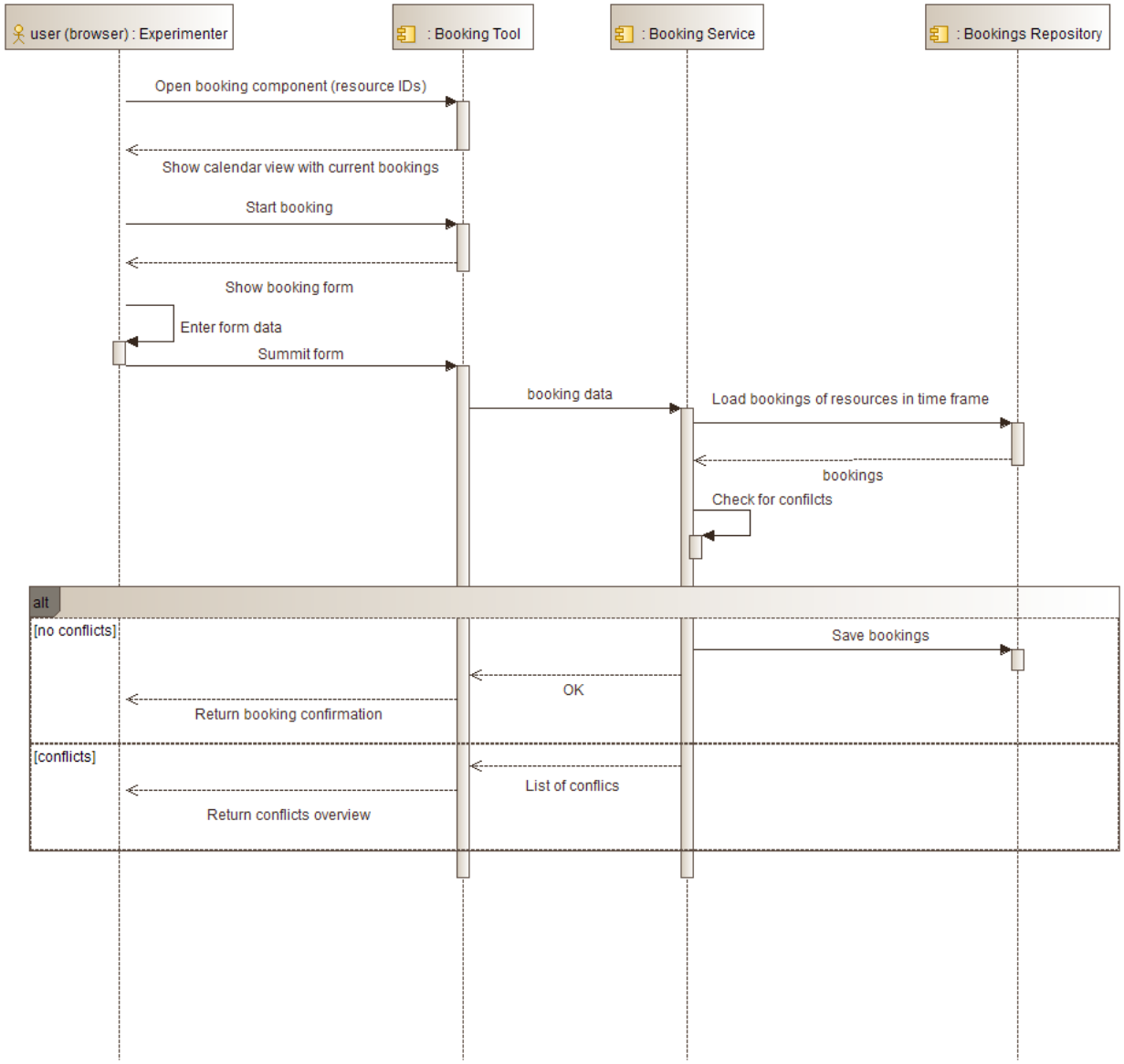


Figure 8 - Sequence Diagram - Book a resource

4.4 Experiment launching and execution

RAWFIE will support two aspects of experiment launching: a) short-term launching and b) long-term launching. In short-term launching (diagram 9) the experimenter through a specific element of the Experiment Authoring Tool will execute pre-defined and pre-approved experiments stored in the Experiment and EDL Repository. In case of the long-term launching (diagram 10) the experiment is executed automatically by the launching service according to the experiment schedule and booking.

4.4.1 Short-term launching

- Experimenter launches the experiment

- Experimenter opens the monitoring tool from the web portal in order to monitor the experiment
- Launching service requests from Experiment and EDL Repository (EER) the instructions for the experiment.
- These instructions trigger Experiment Controller which
    a. Registers the experiment to Testbed Manager
    b. Triggers Network manager for the provisioning of the network connections during the experiment between the nodes
    c. Forwards the instructions for experiment to Resource Controller
- Resource Controller transforms the experimenter instructions into a "global form" of waypoints
- Resource Controller evaluates the path generated in the previous step to avoid constrains and obstacles during the resource mission
- Resource Controller delivers the waypoints to the UxV nodes
- UxV node performs the respective actions and dispatches the relevant information back to the Resource Controller
- Resource Controller after the internal processing returns the data to the Experimenter Controller that transmit the collected data and monitoring results to the Experiment Monitoring Tool
- Information and monitoring data is displayed to the Experimenter through the Experiment Monitoring Tool of RAWFIE web portal component

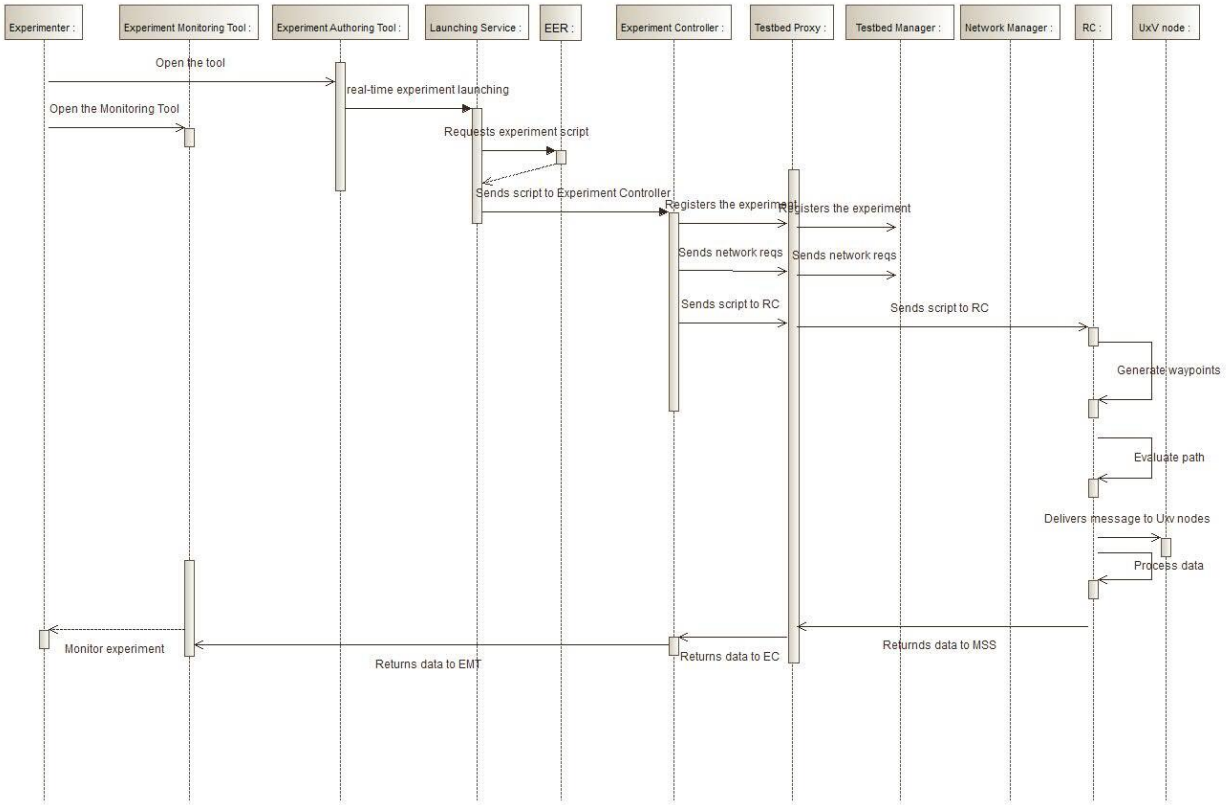


Figure 9 - Sequence Diagram - Real time launching

4.4.2 Long term launching

- Launching service requests from Experiment and EDL Repository (EER) the instructions for the experiment.
- These instructions trigger Experiment Controller which
    a. Registers the experiment to Testbed Manager
    b. Triggers Network manager for the provisioning of the network connections during the experiment between the nodes
    c. Forwards the instructions for experiment to Resource Controller
- Resource Controller transforms the experimenter instructions into a "global form" of waypoints
- Resource Controller evaluates the path generated in the previous step to avoid constrains and obstacles during the resource mission
- Resource Controller delivers the waypoints to the UxV nodes
- UxV node performs the respective actions and dispatches the relevant information back to the Resource Controller
- Resource Controller after the internal processing returns the data to the Experimenter Controller that transmit the collected data and monitoring results to the Experiment Monitoring Tool

- Information and monitoring data is displayed to the Experimenter through the Experiment Monitoring Tool of RAWFIE web portal component

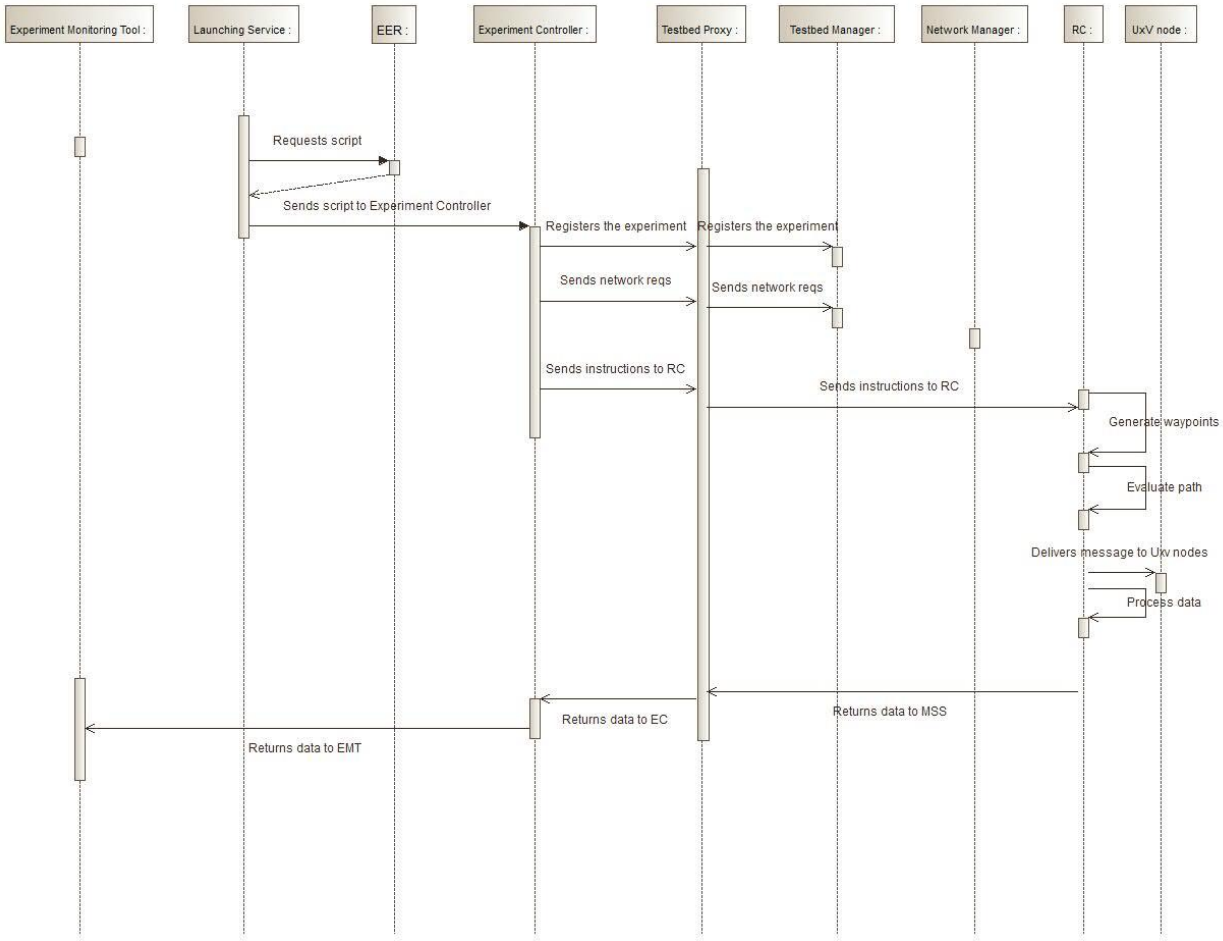


Figure 10 - Sequence Diagram - Long term launching

4.5 Measurements recording

Measurements DB: The measurements database is strongly correlated with the implementation of the message bus system. The reason for this is that data needs to be tee'd (aka forked) from the message bus to a scalable volume storage system such as HDFS. Listed below is a suggested data flow diagram that supports the design we proposed in section 4.6. This model is a standard industry model and will best support the requirements for data analytics which will be worked through in a later section. Storage of measurements is suggested to be done in a relational database and the results of the analytics experiments is suggested to be in a NoSQL database. HDFS provides fault tolerant storage that scales across machines (not like simple RAID based arrays). The specific technology suggestions will be provided in a later section.

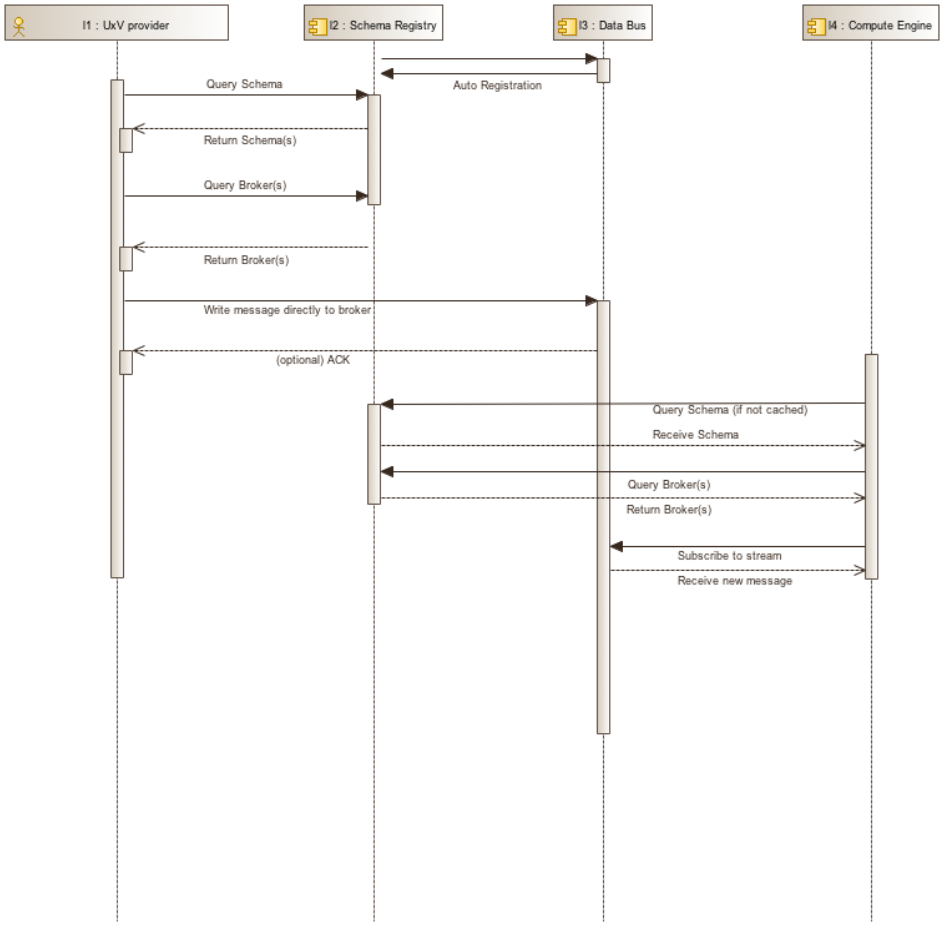


Figure 11 - Sequence Diagram – Measurements recording

4.6 Data analysis

The data analysis engine will seat on the top of the data stream management and computational infrastructure. Through it we will offer the ability to select which streaming data to work with and run a set of available machine learning and data mining operations on it. The interaction of the data analysis engine with the data stream management and computational infrastructure is described in the following sequence diagram.

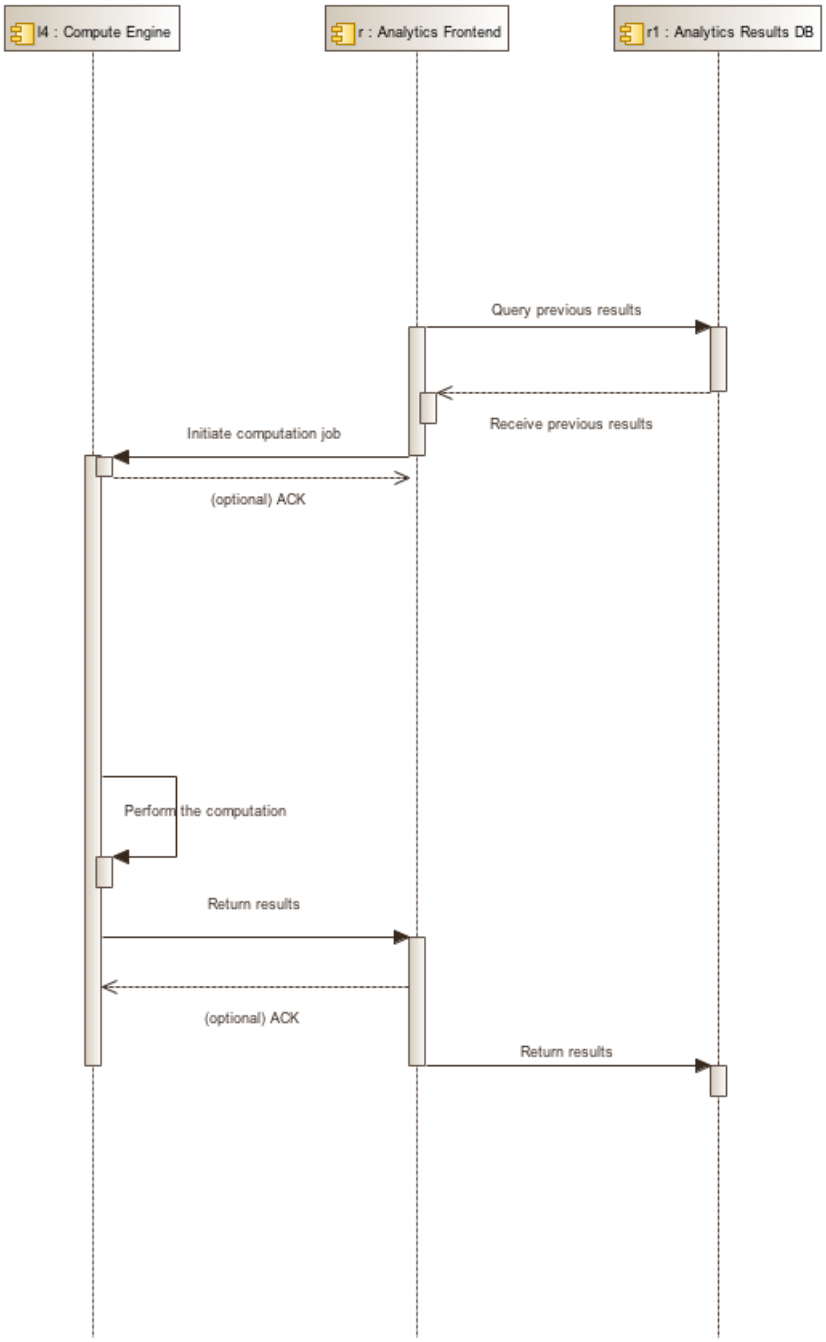


Figure 12 - Sequence Diagram – Data analysis engine

4.7 View visualization of running experiment

The Visualization Tool will be built around a middleware engine that is capable of processing data using parallel processing. The results will be rendered in the client application through the web portal. The steps are sequential and will include:

- Assemble or link to the data to be accessed by the Visualization Engine. This includes video data, sensor data, GPS data, etc in a form that can be used by the visualization engine.
- Transform the data through filters (or the output of other filters) into a new usable output. These filters use algorithms that can be freely chained together. The object outputs can then be rendered separately by the Mappers and Actors.
- Mappers transform the data into graphics primitives. For example, they can be used to specify a look-up table for colouring specific data. They are an abstract way to specify what to display.
- Actors represent an object (geometry plus display properties) within the scene. Things like color, opacity, shading, or orientation.

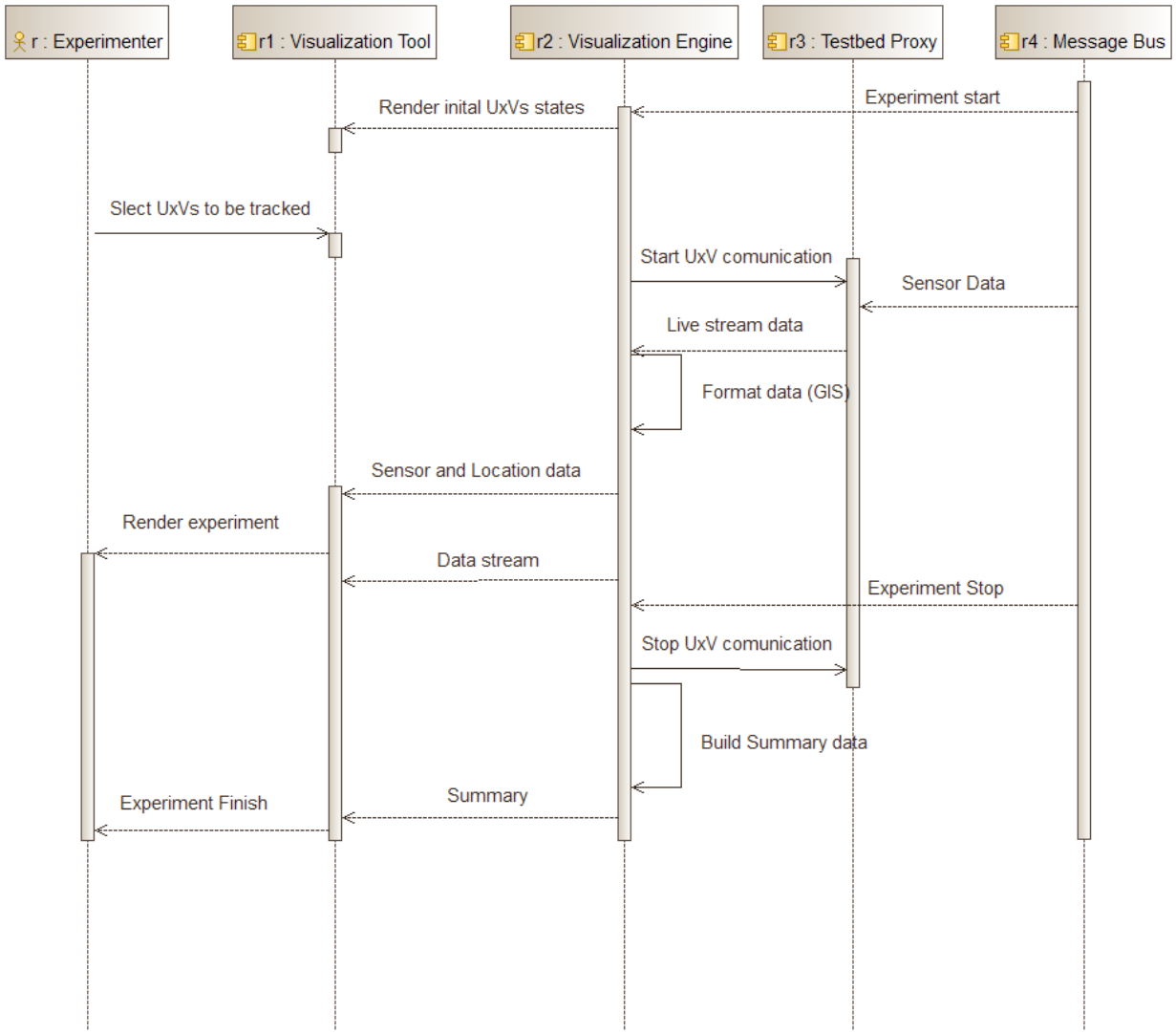


Figure 13 - Sequence Diagram – Running experiment visualisation

4.8 System monitoring

The system monitoring corresponds to a continuous processing of events or parameter values, (good or bad), which triggers notification, actions or other stream processing, such as filtering, log, traces, storage, etc.

4.8.1 General monitoring activities

- Events and parameter values are received by the system monitoring service (KPI)
  - via Message Bus
  - via Service Call
- They are analysed by a pre-defined set of rule, filters, combinations, correlators, etc.
- Predefined actions are triggered
- Actions can be
  - other computations,
  - user notifications (e.g. via email),
  - message posted to the Message Bus
  - service calls on other components
- Storing of the logs and traces
- The logs and traces can be analysed after the execution (post-mortem analysis)

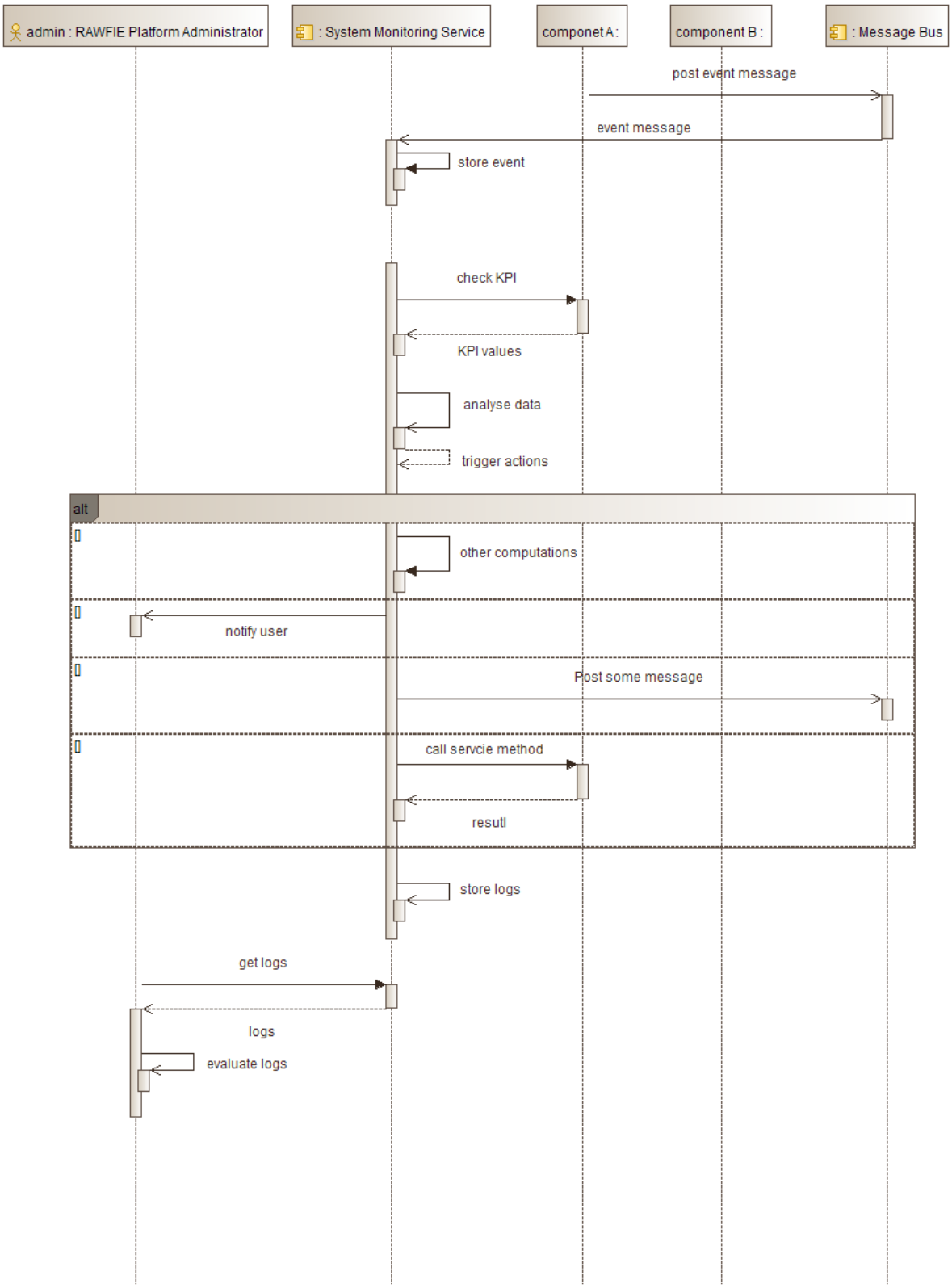


Figure 14 - Sequence Diagram – System monitoring service – General activities

Note that this process may be hierarchically applied at any level of the system, e.g. on a UxV (Device management), in the Testbed (Monitoring manager) and at the RAWFIE Federation level (Monitoring tool). An example is given in the Testbed monitoring (section 4.9).

4.8.2 Error notifications

- Messages of special events are permanently received by the Message Bus
- The System Monitoring Service runs on a periodic basis (triggered by an internal timer)
- Load event collected from the Message Bus
- It requests from all middle tier components the status values (KPI)
- After collection of all the status values, a summary is created
- In case of an error or serious problem, an error notification is issued via email to the a RAWFIE Platform Administrator (admin)
- The admin reads the email and opens the System Monitoring Tool to get an up-to-date overview of the system state
- The System Monitoring Tool request the current summary from the System Monitoring Services and returns this report back to the admin
- The admin evaluates the summary and performs the appropriate steps the resolve the issues.

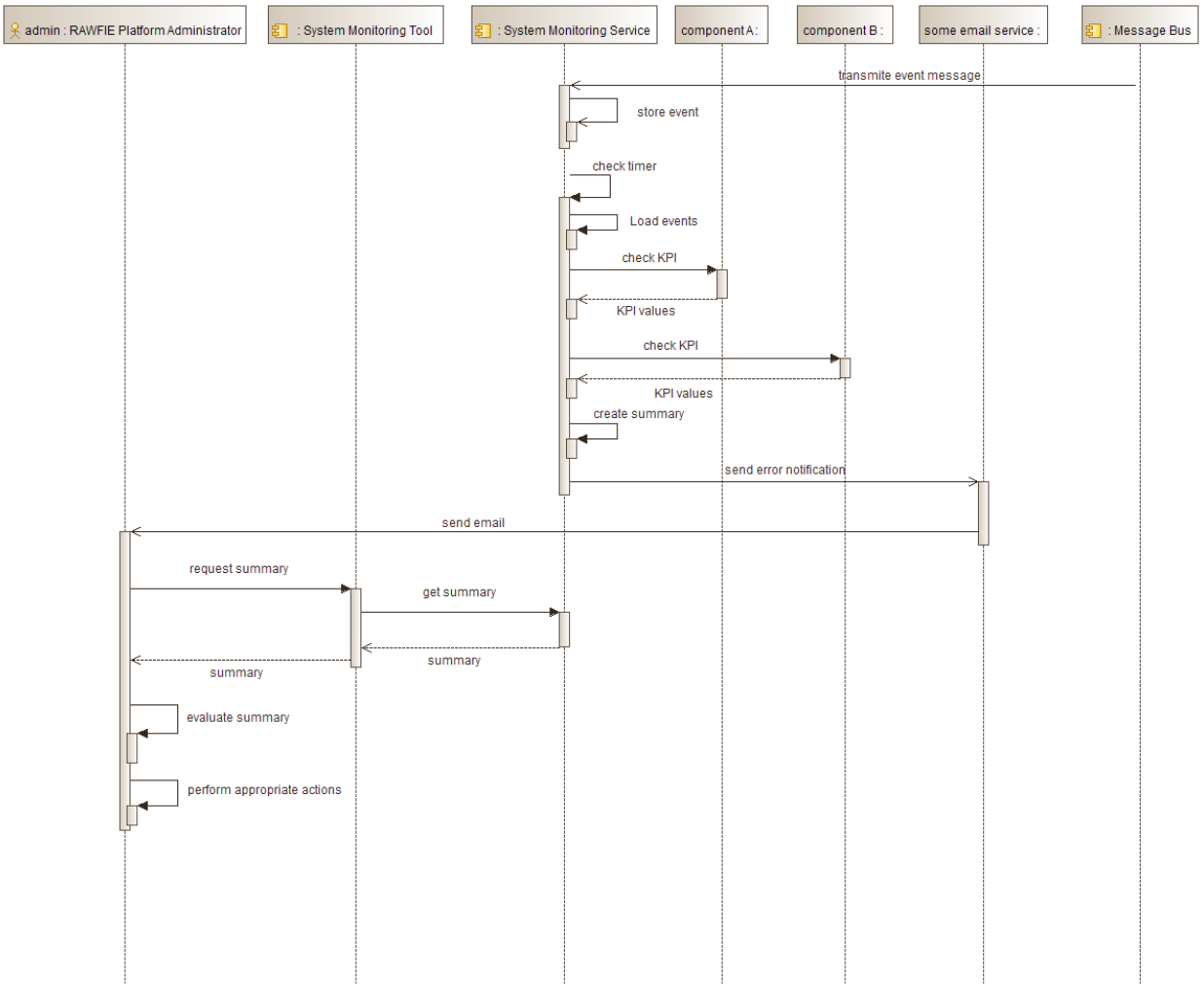


Figure 15 - Sequence Diagram – System monitoring service – Error notification

4.9 Testbed monitoring

RAWFIE provides overview of each integrated testbed with the status of the relevant devices. System Monitoring Service has a request for the information of a specific testbed. This request is forwarded by Testbed Proxy to two related components: Testbed Manager and Monitoring Manager. Testbed Manager contains the information about the status of experiments that are registered in this testbed. This is send back to System Monitoring Service through Testbed Proxy. Monitoring Manager is responsible for the micro-management of the resources. Information as battery lifetime, CPU load, free RAM, biterror rate is gathered periodically for both booked and unbooked UxVs. In order to receive the most recent instance from booked resources a request is also send to Resource Controller. Resource Controller controls the UxVs per experiment taking into consideration UxV's system parameters like remaining battery lifetime. This information is send back to Monitoring Manager and all the accumulated information of the resources is forwarded as a response to System Monitoring Service via Testbed Proxy.

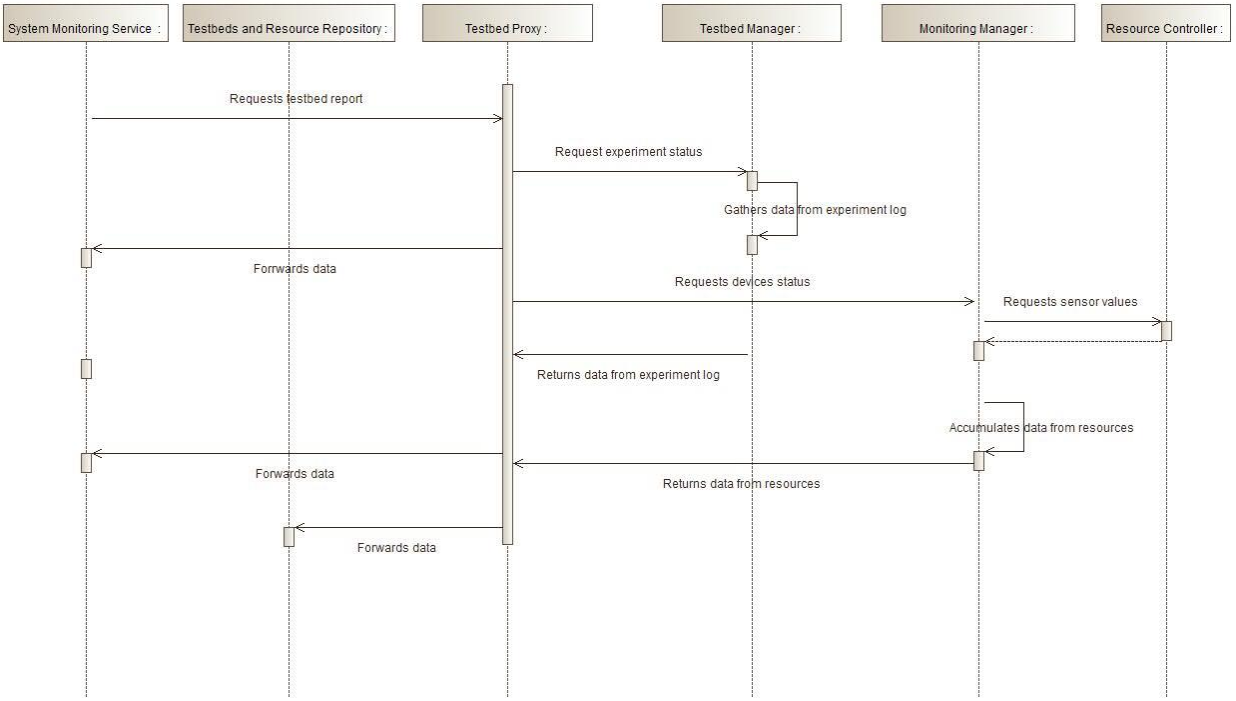


Figure 16 - Sequence Diagram – Testbed monitoring

4.10 UxV remote control

RAWFIE will also provide to the experimenters the ability to remotely control the vehicles. The following diagram illustrates a real time experiment performed by an operator using the provided remote control.

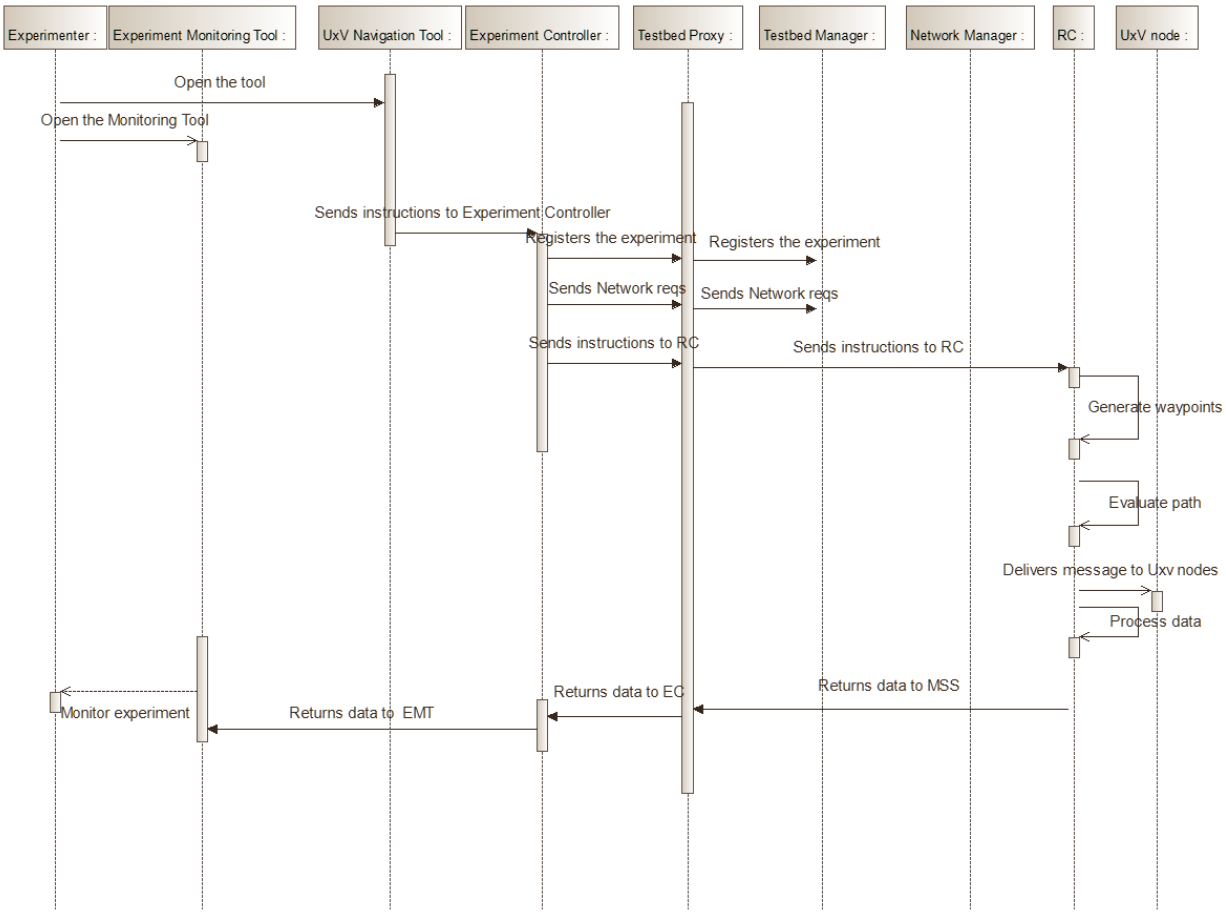


Figure 17 - Sequence Diagram – UxV remote control

Initially the experimenter specifies the required details of the experiment through the UxV Navigation Tool. Such details contain information regarding the number of the vehicles, the purpose of the experiment etc. This tool is part of the Web Portal component.

Remotely controlled guidance mechanism relies on the philosophy of the turn-based navigation approach. At each turn, experimenter defines the next desired location for all the UxVs through the graphical user interface, provided by the UxV Navigation Tool. These directions are transferred through the Experiment Controller to the Resource Controller.

Experiment Controller registers the experiment to Testbed Manager and triggers Network manager for the provisioning of the network connections during the experiment between the nodes

In the sequel, the Resource Controller evaluates these desirable positions and calculates the near-optimal path that the vehicles should follow in order to reach this location. The path planning algorithm takes into account the current location of the vehicles, the model of the robot, navigational obstacles, the system dynamics etc. It is worth noting that the path planning algorithm is provided by the Navigation Service component.

Next, the Resource Controller translates this path into a sequence of waypoints and similarly with the case of 4.4 (Experiment launching and execution) transmits a compact file with the desired coordination and the orientation of the vehicle to the Testbed Proxy. At each time-step, the Resource Controller transfers only one waypoint to the Testbed Proxy.

Then, the Testbed Proxy transmits these instructions to the vehicles which in turn, perform their tasks. When all the UxVs reach the desired location they inform, through the Testbed Proxy, the Resource Controller regarding their current location, their orientation and their battery level. The Resource Controller, taking into account the actual location of the vehicles (sometimes the actual location of a UxV may not be the most likely location due to possible localization issues), recalculates the near-optimum path and transmits to the Testbed Proxy the next set of waypoints (again, one waypoint for each unit). The turn concludes when all the units reach their final location.

At this point of time, the Resource Controller interacts with the Portal so as to inform the experimenter through the Monitoring Tool that the vehicles are ready to accept new instructions. The Resource Controller returns the data to the Experiment Controller, and the Experiment Controller interacts with the Monitoring Tool.

It is worth noting that:

- 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 robots so as to avoid possible collision during the navigation.
- Apparently, the time needed for each robot to reach its desired location is not the same for all units. As a result, some of the UxVs will have to wait (after they have reached their desired location), so as for the rest of UxVs to reach their respective next desired position.
- The Resource Controller ensures that the system is performing as intended and additionally, guarantees the safety of the equipment. If one of the following conditions occurs, automatically, the component activates 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

  In such a situation, the Resource Controller navigates the units back to a safe position, as soon as possible. Additionally, the experimenters receive appropriate warning messages through the Monitoring Tool

5 State of the art

There are different possibilities to reach out goals described in the high level architecture above. This chapter examines some of the FIRE projects belonging to the Future Internet Research & Experimentation open research environment as well are relevant technological solutions that may be adopted and expanded in RAWFIE.

5.1 Relevant FIRE projects

There were several research projects in the FIRE domain in the last years. Even though RAWFIE targets on something special and new (the usage of UxV in the testbeds), there are probably many ideas and architectural elements that could be reused or adapted. These projects can be identified also in the FIRE pentagon (Figure 18 - FIRE pentagon).

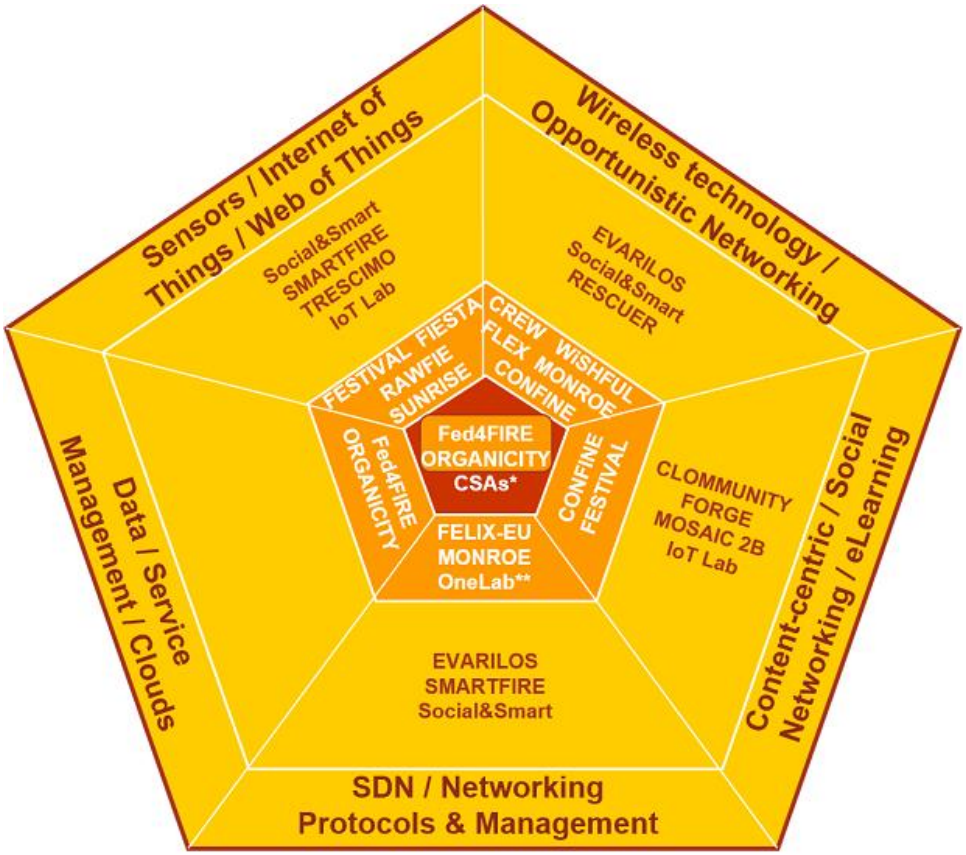


Figure 18 - FIRE pentagon

5.1.1 Fed4FIRE

5.1.1.1 General description and goals

The Federation for Future Internet Research and Experimentation (Fed4FIRE) [1] is an Integrating Project (IP) under the European Union's Seventh Framework Programme (FP7) in the context of the Future Internet Research and Experimentation (FIRE) programme. The

Fed4FIRE architecture is mainly divided in three categories: a) Federator, b) Testbed side and c) Experimenter side

Federator

From the perspective of the Federator, several components will be provided in one or more central locations for resource discovery, resource requirement, resource reservation and provisioning. The components that are offered by the federator are:

- Portal: a central starting and registration place for new experimenters
- Member and slice authority: experimenters who register at the portal are registered at this authority.
- Aggregate Manager (AM) directory: which is readable by computers to have an overview of all testbeds available in the federation
- Documentation center: which gives an overview of available tools, testbeds and tips to the experimenter
- Authority directory: for authentication/ authorization between experimenters and testbeds. Fed4FIRE adopts a trust model where testbeds and authorities establish trust relationships among each other.
- Service directory: directory service for federation and application services

Testbed

- Testbed resources: can be virtual or physical nodes that will be accessible through SSH
- Testbed management component (called Aggregate Manager): is responsible for the discovery, reservation, and provisioning of the testbed's resources. The testbed has the freedom to adopt any desired software framework to implement this functionality, as long as it can expose functions through the Aggregate Manager API. The basis of the Fed4Ffire Aggregate Manager API adopted by the GENI Aggregate Manager API [36].
- A testbed may have an authority in the federation. This makes the testbed independent of the availability to the Federator to allow its own experiments to participate in the federation.

Experimenter

- Some of the tools made available to the experimenter are hosted tools. The experimenter will make use of these tools through a browser
- Several experimenter tools for resource discovery, reservation, and provisioning already exist and run locally on the experimenter's computer. So these are stand-alone tools instead of hosted tools. Examples are Omni, SFI, NERI and jFed. The experimenter has the freedom to choose the tools of his or her preference that will be supported by Fed4FIRE as long as it is compatible with the adopted AM API.

5.1.1.2 Architectural and technological solutions

The most important software components and standards adopted by the Fed4FIRE are presented in the following subsections.

Testbed Management

Slice Federation Architecture (SFA): key interfaces and standards of the SFA framework are the following:

- GENI Aggregate Manager (AM) API version [36]:
  - ✓ This contains a description of the API for discovery, resource requirements and provisioning
  - ✓ It defines the GENI certificates for authentication. The GENI AM API uses XML – RPC over SSL with client authentication using X.509v3 certificates
  - ✓ It defines GENI credentials for authorization
  - ✓ It defines GENI URN identifiers for identifying and naming users, slices, slivers, aggregates and others

- Ontology based resource specifications GENI RSpec [52] that comes in three parts:

  - ✓ Advertisement RSpec: for resource discovery(getting a list of all resources)
  - ✓ Request RSpec: for requesting specific resources
  - ✓ Manifest RSpec: for describing the resource in an experiment

Experiment Control

OMF: Framework for test bed management, measurement and control

- From the experimenter's point of view, OMF [37] provides a set of tools to describe and instrument an experiment, execute it and collect its results.
- From the testbed operator's point of view, OMF provides a set of services to efficiently manage and operate the testbed resources (e.g. resetting nodes, retrieving their status information, installing new OS image).

The following figure presents a general overview of OMF from the user's point of view. The user/experimenter describes her experiment in a high-level domain-specific language, and passes it on to OMF. The framework will in turn deploy and configure the experiment on the test bed(s) according to the user's description. Then it will initiate and control the execution of this experiment. Finally, during the experiment execution, the framework will measure and collect data according to the user's description. These measurements are sent to a repository available to the user and can also be used to dynamically steer the experiment control.

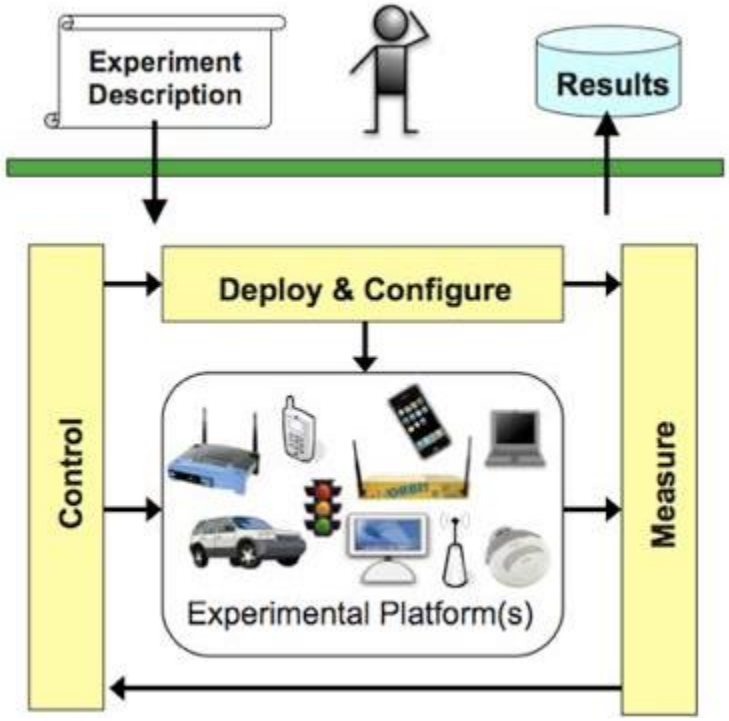


Figure 19 - OMF framework

It is important to note that OMF tools and software are not tied to a specific testbed technology. Indeed, OMF has been deployed and maintained on multiple testbeds with many different types of technologies, at NICTA, Winlab, Intel, SUNY and a US Government Lab. It has been used regularly by many researchers and students in the previous years with some testbeds operating 24/7 providing access to experiments infrastructures to a large number of experimenters around the world.

Experiment Measurements

OML [38] is an instrumentation tool that allows application writers to define customizable measurement points (MP) inside new or pre-existing applications. Experimenters running the applications can then direct the measurement streams (MS) from these MPs to remote collection points, for storage in measurement databases.

OML consists of two main components:

- ***OML client library****:* the OML client library provides a C API for applications to collect measurements that they produce. The library includes a dynamically configurable filtering mechanism that can perform some processing on each measurement stream before it is forwarded to the *OML Server*. The C library, as well as the native implementations for Python (OML4Py) and Ruby (OML4R) are maintained here.
- ***OML Server:*** the OML server component is responsible for collecting and storing measurements inside a database. Currently, SQLite3 and PostgreSQL are supported as database back ends.
- The optional OML Proxy Server: when doing experiment involving disconnections from control/measurement network (such as with mobile devices), the proxy server can be used temporarily buffer measurement data until a connection is available to transfer it to the collection server.

TopHat [41] is a tool that collects data from various sources of data and aggregates them in order to expose enriched measurement data to the user. These data can typically be used for monitoring or more generally to have a better understanding of the network. TopHat provides measurement data and is built above the Manifold framework [41] (as is MySlice, which provides testbed-oriented data). Manifold allows the user to query various sources of data through a single API (in the case of TopHat, measurements through the TopHat API) while relieving the user from needing to know which platforms must be queried. Each source of data announces what kind of data it provides according to a common ontology. Manifold dispatches the user queries to each relevant platform, collects their replies, combines them, and sends the result to the user. For example TDMI provides traceroute measurements, while Maxmind can map an IP with localized city names. Since each platform uses its own API (database, webservice), each query issued by TopHat Manifold is translated into the platform's API through a dedicated gateway. In the same way, the reply of the platform is translated by the gateway to be expressed in the TopHat/Manifold format.

Facility and Infrastructure Monitoring

Zabbix [39] is an open-source solution for facility and infrastructure monitoring. It supports performance monitoring natively, in addition to facility monitoring and alerting. It also supports an extensive list of operating systems and platforms, including virtual machines. Three types of resource components are available for monitoring: native Zabbix agents, SNMP monitoring and agentless script-based queries; all data is then aggregated within a central collection server which relies on SQL databases (MySQL or PostgreSQL) for storage.

Nagios [40] is another open-source base for infrastructure monitoring solutions. Unlike Zabbix, it does not support performance monitoring natively. It provides status reports for hosts, applications and networks. Nagios can also be extended through the use of user scripts run by the Nagios Remote Plugin Executor (NRPE). It has built-in support for raising alerts on problematic situations. Data is stored in an ad hoc backend, but some plugins allow export to SQL databases.

Data can also be processed and exchanged between instances using the Nagios Remote Data Processor (NRDP).

5.1.1.3 Conclusions and relevance for RAWFIE

RAWFIE project could leverage some elements developed in the context of the Fed4FIRE and relevant projects. One mechanism that could be adopted by the project is the SFA standard (specifically GENI AM API v3) together with the ontology based resource specifications (Rspecs) for the testbed experiment lifecycle management. By using this approach, the RAWFIE testbeds facilities must be SFA compliant to map the concepts from their resource management system and to generate software to implement the mappings. With the SFA standard RAWFIE will also adopt a specific well-defined trust and security framework that can be achieved by the use of the X.509 certificate. Experimenters can obtain this certificate by the RAWFIE identity providers. The certificates can be presented to any test facility, which makes a decision as to whether it trusts the identity provider that signed the certificate. A test facility can do this by reference to directory of root certificates from RAWFIE ID providers. The trust decisions are made at the test facility level.

In addition, RAWFIE could use the OML standard to collect data from any resource (e.g., input from temperature sensors). In this case, all RAWFIE testbeds will have to support OML. This requires the installation of the OML client library on all testbed nodes, and the deployment of at least one OML collection server reachable by all.

5.1.2 SUNRISE

5.1.2.1 General description and goals

The SUNRISE FP7 project (2013-2017) [67] is a project targeting Internet of Underwater Things with main objectives to develop:

- Five federated underwater communication networks, based on pilot infrastructure already designed, built and deployed by consortium partners, in diverse environments (Mediterranean, Ocean, Black Sea, Lakes, Canals), web-accessible and interfaced with existing FIRE facilities to experiment with Future Internet technologies;
- A software-defined open-architecture modem and protocol stack that will empower open collaborative developments;
- Standard platforms for simulation, emulation and replay testing to estimate underwater communication networks at a fraction of time, cost, complexity of current at-sea experiments, validated by tests conducted on the SUNRISE networks over a variety of applications and environments;

A user-friendly interface for diverse users to interact with SUNRISE systems in order to conduct trials and benefit from databases of underwater Internet of Things (IoT) performance data gathered over long periods from the SUNRISE infrastructure.

SUNRISE aims to exceed and enhance the features supported by existing partner's test-beds and installations by exploring, implementing, and developing a novel paradigm of software defined modem and software defined communication stack, which allows experimentation with novel physical layer techniques, novel cross-layer optimized and adaptive underwater communication protocols, and novel schemes for underwater devices cooperation. SUNRISE facilities will also provide large scale testing infrastructures (with respect to the expected size of underwater networks and to the size of current deployments), and a level of heterogeneity in terms of assets involved, marine environments and applications, that which is not available as of today. The SUNRISE developed tool chain will also reduce time to market and time to experiment considerably and will provide a key tool to support optimization of solutions, code generation, programmability and re-programmability of underwater deployments.

SUNRISE directly addresses FIRE objectives by combining technology with novel paradigms in new, open experimental facilities, integrating physical systems with software development into the Internet of Underwater Things. It is the first project that develops this concept, based on joint research performed in the partners in the last few years. SUNRISE will also provide a way to select Internet of Underwater Things standards based on objective measures of performance, strengthening its facilities as more sites are added in the future as a result of the two envisioned open calls.

5.1.2.2 Architectural and technological solutions

In the SUNRISE architecture, depicted in Figure 20, a number of test-beds are federated so that unified commands and data are sent/received by users (both common and expert users), by means of applications and services. Data and events coming from the Federated Test-bed are harmonized and integrated in storing systems and made accessible to applications in a standardized format. On the other end, suitable commands can be delivered to the Federated Test-bed in a unified format. While traversing the stack, the unified commands are transparently translated into specific commands that can be executed on the target test-bed.

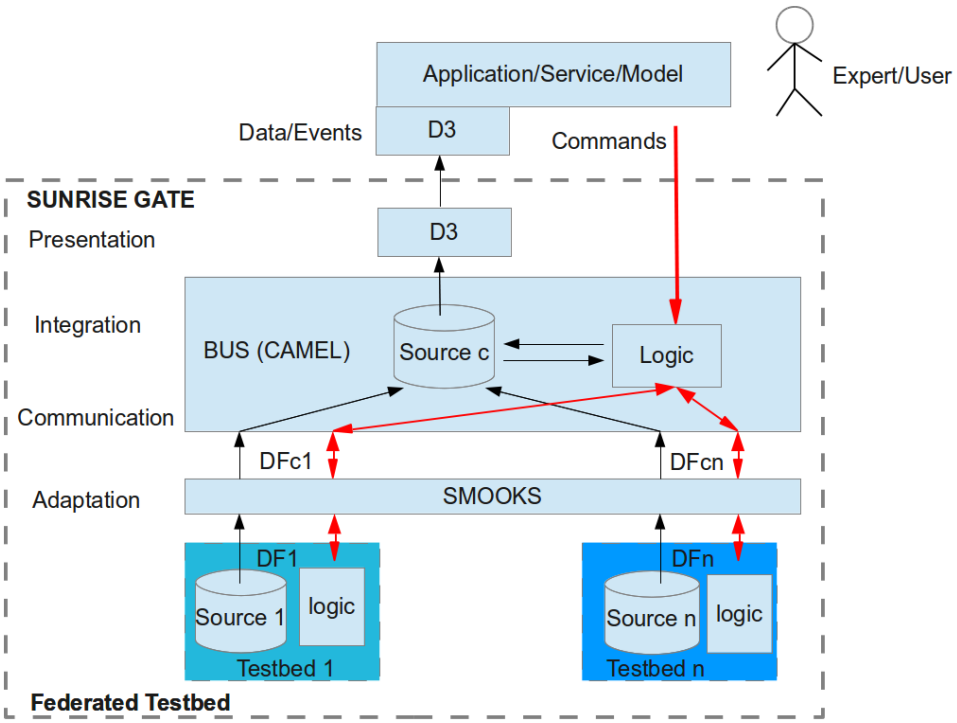


Figure 20: Logic view of the SUNRISE architecture

The core of the architecture is the SUNRISE Gate. It interfaces to different experimental infrastructures and monitoring systems (i.e. test-beds) in order to create added value applications and services development. It seamlessly integrates heterogeneous flows of information available in different remote systems and over the Internet. It enables presentation through customizable GUIs and allows model-driven data processing and decision support at the application level. The main functionalities that are provided by the SUNRISE Gate are the following:

- Resource discovery: it allows the SUNRISE gate to become aware of the functionalities offered by the underlying test-beds,
- Reservation: users can reserve the resources on the test-beds to perform a specific experiment,
- Provisioning: resources are granted to the user that booked them,
- Experiment control: it allows to start/stop and pause the experiment,
- Monitoring: it visualizes the main parameters of ongoing experiments,
- Permanent storage: experimental results are stored in order to be accessed for further analysis,
- Resource release: once an experiment has been completed the resources are made available for further experiments.

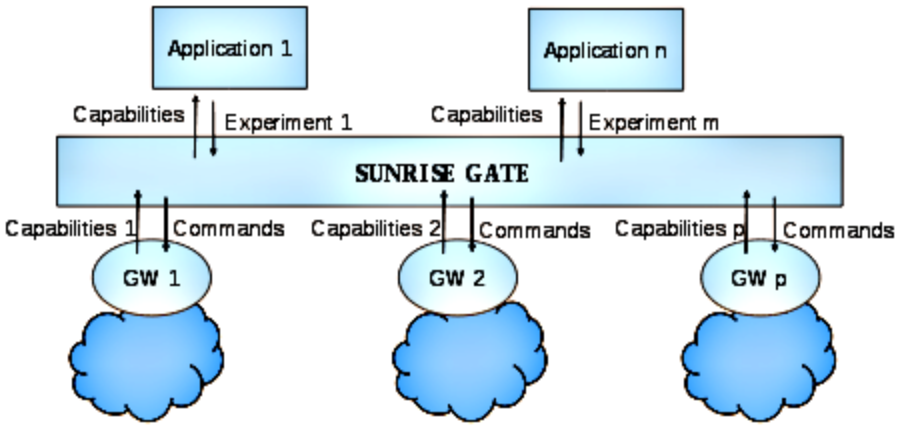


Figure 21: SUNRISE Gate in its essense

The SUNRISE Gate allows gateways of the test-beds to register themselves and to upload information about the nature and structure of the test-beds, i.e. resource discovery. Registered users can view this information and can control some nodes in the test-beds and issue experiments on them.

The SUNRISE Gate collects capabilities of test-beds (from 1 to p in Figure 21) and exposes them to the applications. Several applications (from 1 to n in Figure 21) can be developed independently. Applications can issue experiments (from 1 to m in Figure 21) on one or more test-beds, by means of the SUNRISE Gate that knows how to launch commands on test-beds. The SUNRISE federating architecture is intended to be integrated into the FIRE Facility (Future Internet Research and Experimentation).

5.1.2.3 Conclusions and relevance for RAWFIE

The software infrastructure developed for the SUNRISE project has a special emphasis on underwater acoustic communications. In particular, the middleware used to interconnect test-bed assets and test-beds with the SUNRISE Gate was initially designed to test and simulate/emulate underwater acoustic communications. This middleware, called SUNSET (Sapienza University Networking framework for underwater Simulation Emulation and real-life Testing) is partially open-source but key components are not distributed freely and require software licenses and/or agreements with the copyright holders (University of Rome "La Sapienza"). Given that in the RAWFIE project limited work will be done using acoustic communications and the number of test-beds is significantly lower and less diverse, the architecture of SUNRISE project might not be entirely suitable. However, inspiration can be drawn from a broad interpretation of the SUNRISE architecture.

5.1.3 RELYonIT

5.1.3.1 General description and goals

The aim of RELYonIT [3] is to provide a dependability framework for IoT infrastructures by addressing challenging interactions between WSANs and their environment. This comes from

the fact that in most situations IoT deployments cannot become a reality if dependable operation expressed by parameters such as data delivery reliability and low latency is not taken into account. The major hurdle to providing a dependable IoT is that the operation of WSAN in real world is deeply affected by their surrounding environment. Environmental properties such as electromagnetic interference, ambient temperature and humidity have significant impact on achievable network performance and thus on the operational quality of service. These conditions are not only hard to predict for a given deployment site but they also may largely vary from one deployment site to another.

5.1.3.2 Architectural and technological solutions

Towards the realization of the previous goal the first step is the extraction of participating entities in the dependability requirements which in the case of RELYonIT are expressed by:

- System lifetime (T)
- Data yield (R)
- Latency (L)

Then a dependability requirement is formulated by maximizing or minimizing one performance objective subject to constraints on the remaining metrics as in equation below:

$$\begin{aligned} &\text{Maximize/Minimize} && M_1(c) \\ &\text{subject to} && M_2(c) \geq, \leq C_1 \ probability \ 1 - P_{c1} \\ & && M_3(c) \geq, \leq C_2 \ probability \ 1 - P_{c2} \end{aligned}$$

Where each M_i is one among {T, R, L} and $\{C_1, C_2\}$ are constraints to be eventually satisfied with probability $1 - P_c$, where P_c is the user-provided maximum tolerance for violating the constraint.

The next step is the embedding of the above described optimization problem in the tool-chain where source code development for WSN nodes is performed. The following assumptions are also applied:

- Dependability requirements are applied to a specific protocol (sensor nodes typically support different classes of protocols)
- Dependability requirements may change over the application execution
- Dependability requirements apply system-wide and it is not possible for two nodes to have different dependability requirements for the same protocol at the same time.

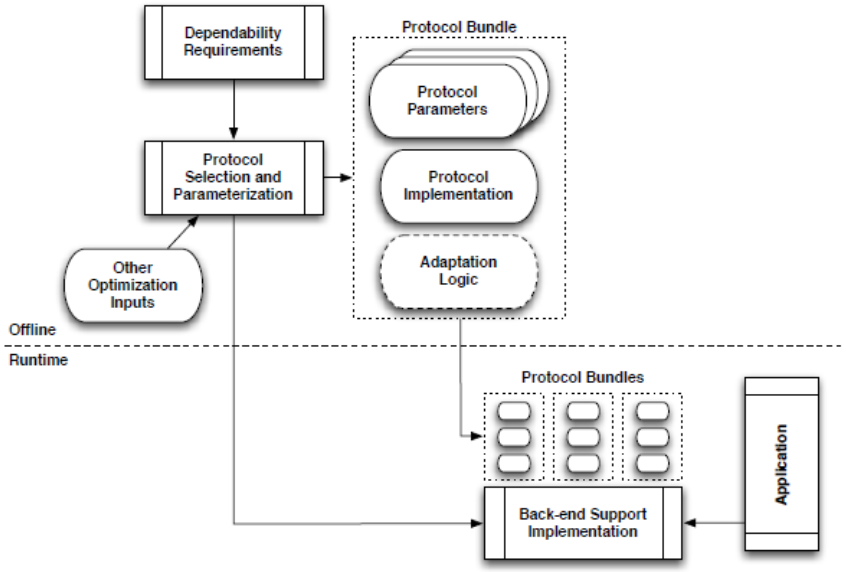


Figure 22: - Design diagram of RELYonIT tool-chain

Figure 22 presents a design sketch of RELYonIT tool-chain that connects dependability requirements with a bundle of protocols that will exist as run-time code in each one of the sensor nodes participating in the network. A protocol selection and parameterization tool, takes a machine-readable specification of dependability requirements as input together with other optimization information useful to determine the most appropriate protocol and its parameters; the output of the protocol selection and parameterization tool is one or more protocol bundles. A single bundle includes the chosen protocol implementation together with a predefined set of operating parameters. The protocol bundles are deployed together with the executable application code and a software layer that implements back-end support functionality to manage protocol bundles. This contains a language-specific encoding of the dependability requirements and is also in charge of the monitoring functionality needed to trigger protocol adaptations based on changing environmental conditions.

Dependability requirements are specified using a XML schema where beyond the metric of interest (lifetime, data yield, latency), the protocol class describing the three dominant protocol classes of WSANs (dissemination, collection and peer-to-peer) is also included. The dependability specification was embedded with programming languages in two ways with the first integration target being a Java-like high level macro programming language developed in the 7th FP project makeSense [14] (MPL) and the second one a C-like based integration for the Contiki OS, a popular operating system for WSNs [15].

Examining the possible reuse of RELYonIT architectural elements in the RAWFIE architecture the following conclusions can be drawn:

- The specification language of RELYonIT is restricted to the formulation of an optimization problem and expressivity is not broad enough for describing complex experiments.
- Due to the different natures of nodes participating in the two projects (UxVs for RAWFIE and sensor nodes for RELYonIT) some optimization parameters (data loss, latency) are relevant in both cases while others like lifetime are not important in UxVs, which are considered much more powerful devices in energy terms and selection of communication protocols with low energy consumption is not considered a factor of primary importance.
- Different protocol bundles (mainly focusing on data link and network layers in OSI model terms) could be considered an element of experimentation for autonomous devices like UxVs and any -preexisting efforts that could be applied in operating systems and network interfaces of UxVs could be of great help for the realization of experiments.
- The level of the embedding of dependability requirements specification in a tool-chain producing source code is also a parameter under investigation. Solutions based on a high-level macro programming language seems more attractive (provided that implementations of native code for abstraction classes are given) as this gives the opportunity to test the requirements in a wide range of source codes running possibly in different testbeds.

5.1.3.3 Conclusions and relevance for RAWFIE

Architectural concepts that could also be applied to RAWFIE are limited as the main goal of RELYonIT is to optimize protocol performance of IoT devices in real environments and it does not include a middleware for handling different IoT testbeds. However network experimentation based on protocols optimization can be foreseen as a possible candidate for future RAWFIE extensions with the downloading of different optimization engines in each one of the UxV nodes. The optimization engine should be able to directly interact with the data link and network layer and even with the physical layer (in the case of SDR/Cognitive Radio) of the UxV's protocol stack with the aim to achieve better network performance in real-world environments.

5.1.4 IoT Lab

5.1.4.1 General description and goals

"The IoT Lab is a research project exploring the potential of crowdsourcing to extend IoT testbed infrastructure for multidisciplinary experiments with more end-user interactions" [4]. More information can be found at the IoT Lap webpage [5].

5.1.4.2 Architectural and technological solutions

To handle the heterogeneous testbeds in a common way, the testbeds are "virtualized" via a common API: There are four individual IoT Lab testbeds. Each one with a different architecture set of provided functionalities, etc. Each testbed is responsible for exposing its resources and

services over a commonly understood API. Each testbed is responsible for dealing with its specific characteristics, such as network connectivity or its special resources.

This virtualization results in:

- Testbeds could be handled in a common way
- Testbeds provide a Fed4FIRE-compliant point of access to the outside world
    - OMF client interface [25]
    - Communication based on XMPP
    - Resource Specification (RSpec) version 3[52] used from GENI
- There is common addressing scheme for all resources in IoT Lab
- Resource Directory can be maintained at the IoT Lab cloud (instead in the TB themselves)

The ORBIT Measurement Library (OML) [26] is used in addition to OMF(see also 5.1.1).

The networking in IoTLab is fully based on IPv6. This guaranties interoperability between multiple physical interfaces (Ethernet, Wi-Fi, Bluetooth, IEEE802.15.4, etc) and enables end-to-end connectivity without the need for a NAT. In addition, IPsec is used to enable secure/encrypted communication of potential non-secure networks.

On top of IPv6 two transport/application layers protocols are currently under validation:

- Constrained Application Protocol (CoAP) [42] (via UDP): This is HTTP like communication. It is suitable for state transfer.
- Message Queuing Telemetry Transport (MQTT)[57]: This is a messaging protocol used on top of TCP/IP. It is based on a publish-subscribe communication model, which is suitable to for live data (measurements)

The current version of RSpecs was greatly extended, to also include the description of IoT resources by incorporating into them the IPSO Application Framework [44]. This was necessary because the RSpecs mainly focus on describing testbed facilities of computer networks typically consisting of few servers and some network level devices such as routers and switches.

5.1.4.3 Conclusions and relevance for RAWFIE

The direct use of OML is not reasonable in the context of RAWFIE, as OML was initially designed for regular networks. OML does not provide with any mechanism for addressing network availability/connection problems, which will regularly occur with moving UxVs. So OML may only be used as interface between the Testbed proxy and the Middle Tier. But inside the testbed, other technologies should be used to transmit measurements (e.g. messages buses with persistence).

The approach of virtualizing the testbed will also fit well to RAWFIE, as the different testbeds for the UxV will be very heterogeneous. So, integrating a testbed into RAWFIE requires a common interface (e.g. FED4FIRE compliant) exposed by the Testbed Proxy, which should hide internal management, networking and other issues.

RAWFIE should also follow the consequent use of IPv6 to address resources.

5.1.5 WISEBED

5.1.5.1 General description and goals

The WISEBED project [6] was a 3-years EU project (2008 - 2011) funded by the European Commission under the FP7 framework.

The aim of the project, focused on the Wireless Sensor Networks domain, was to build a distributed infrastructure of heterogeneous and interconnected testbeds. The final target was to move from local, small-scale testbeds with limited number of devices, to a larger-scale, federated testbed infrastructure, resulting in a pan-European Wireless Sensor Network environment. An interdisciplinary approach was adopted with the possibility to integrate and test hardware solutions, software, algorithms and data, by applying the more recent research results on algorithms and protocols for Wireless Sensor Networks measurements and communication.

5.1.5.2 Architectural and technological solutions

The WISEBED general architecture is composed by the following elements:

- The Testbed Management System
- The Testbed Interconnection System
- The per-node Software Development Kit
- The Simulation System

A set of Application Programming Interfaces (API) are defined to ensure the different functionalities and the integration in a distributed environment:

- the WSN API provides access to the local testbed of each partner site
- the Interconnection API permits inter-testbed federation of resources
- the Monitoring API presents live testbed data in a standard format
- the OSA/V (OS Abstraction/Virtualisation) API operates on each WSN node to provide heterogeneous and virtualised hardware support.

Together, these define the set of services that the major software elements need to offer in order to properly interact and communicate.

The WISEBED project developed a specific Testbed Management Architecture called TARWIS (Testbed Management Architecture for Wireless Sensor Networks). It provides the interface through which users upload experiment software to testbeds and download results. It interacts

with a testbed using the developed WSN API indicated above. TARWIS provides the functionalities such as reprogramming and reconfiguration of sensor nodes; provisions to debug and remotely reset sensor nodes in case of node failures; and a solution for collecting and storing experimental data. Using the WSN APIs to interact with a testbed enables TARWIS to perform all of these standard functions independently from the node type and node operating system that is being employed in any specific testbed. TARWIS has been integrated with the standard WISEBED authentication/authorisation and reservation systems, providing SSO (Single Sign-On) access for all testbed resources federated in WISEBED.

At a more detailed level from the software perspective, the following components are comprised in each single WISEBED testbed:

1. A graphical user interface. The WiseGUI is an Open Source, Web-based Frontend, written in HTML5 and JavaScript, for accessing WSN testbeds using suitable API. It allows users to: look at available testbeds and testbed resources (sensor nodes); authenticate to specific testbeds; make reservations for tests; control sensor nodes and visualise live data streams while tests are running
2. The Testbed Runtime (TR), which provides the middleware environment with the set of services needed to run a WSN testbed infrastructure. Specifically, it implements testbeds and resources management and control API, such as:
    a. RS (Reservation System) API, used to perform reservations
    b. SNAA (Sensor Network Authentication and Authorization) API, used to control accesses
    c. iWSN (Wireless Sensor Network) API, used for controlling sensor nodes (resources)
3. The sensor nodes (resources) themselves

The Testbed Runtime implements both SOAP and REST Web Services API. SOAP API was actually deprecated and replaced by the new REST API used by the WiseGUI through its JavaScript library (wisebed.js, a client library to interact with the Testbed Runtime REST API) to allow client side access and control of the testbeds. WiseGUI also implements a direct, near real-time WebSocket-based communication with the sensor nodes through the serial port.

The WISEBED project has also implemented and integrated additional software to support different needs, and specifically:

- Wiselib, a template-based library, written in C++, which contains various algorithm classes (e.g. for implementing localization and routing algorithms). It aims at helping the realization of algorithms and protocols for Wireless Sensor Networks, and can be compiled and used in different platforms and sensing devices
- A set of command line utilities to control experiments, in the form of a library called wisebed.js-scripts

- WiseML, an XML format for representation of experiment traces (like topology description and time representation of the experiments output), which is an extension of the GraphML format
- The existing Shawn Wireless Sensor Network Simulation tool
    - It is integrated in the WISEBED architecture
    - Supports WISEBED API
    - SupportsWiseML files to read and write simulation tests results and measurements
    - Supports WiseLib to simulate algorithms before bringing them in the real testbed

A WISEBED Virtual Machine (WISEBED VM) is also available, with a pre-configured development environment for WSN applications and WISEBED testbed experimentation. It can run also in a desktop environment allowing to test the full WISEBED toolchain deploying to large scale testbeds. It contains:

- Compilers for various WSN hardware platforms
- Several WSN operating systems
- Wiselib library
- Shawn Simulator
- Experimentation Scripts
- Testbed Runtime (TR) WISEBED backend implementation

5.1.5.3 Conclusions and relevance for RAWFIE

WISEBED has a different scope if compared to RAWFIE. It is only focused on the Wireless Sensor Networks domain, and on the interconnection of completely separated and already existing testbed platforms, each of them providing its own Web accessing Portal, software infrastructure and resources. These testbeds are interconnected, in WISEBED, through the so called “Interconnection API”. Conversely, RAWFIE aims at developing a common platform for the management of a federation of testbeds, by providing a single point of access and a common software infrastructure (RAWFIE Frontend, Middle Tier and Data Tier components), for management, execution and analysis of experiments, which are conducted using specific resources, available at the different testbed facilities.

Nevertheless, some of the technical solutions implemented by the WISELIB project, which are anyway freely available, could be further investigated for use in RAWFIE. These are, mainly:

- Testbed Runtime, and especially the connected testbeds and resources management and control API:
    - RS (Reservation System) API
    - SNAA (Sensor Network Authentication and Authorization) API
    - iWSN (Wireless Sensor Network) API
- wiselib library, which could be implemented into the sensors carried by the UxV devices

- wisebed.js client library, for building custom Web clients which, using the abovementioned API, could directly communicate with testbeds resources
- WiseML format, for the representation of experiments outputs
- The provided WISEBED VM itself

5.2 Relevant technologies

5.2.1 Experiment Description Language

5.2.1.1 OMF Experiment Description Language (OEDL)

The OEDL is a language that runs under the OMF [25], a framework for control and management of networking testbeds. To define a scenario through the OMF framework the experimenter uses the Experimenter Controller (EC) for running the experiment script and steer the resources. The EC can be installed on a user-facing machine inside the testbed, or alternatively on the user's own computer.

The OEDL language was used by several FIRE projects among them are Fed4FIRE [7] and GENI [8] projects.

Language characteristics

An OEDL experiment description is composed of 2 main parts:

a. A first part where we declare the resources that we will use in the experiment, such as applications or computing nodes, and some configurations that we would like to apply to these resources.
b. A second part where we define the events that we would like to react to during the experiment's execution, and the associated tasks we would like to execute when these events occur.

A very simplistic OEDL experiment would look like this:

```
# Comment: This is the 1st section, where we declare resources and configurations
#
defProperty('my_resource', "leecher-player-1-gec18-tutorial-5", "ID of a resource")
defGroup('Actor', property.my_resource)

# This is the 2nd part, where we define events and associated tasks
#
onEvent(:ALL_NODES_UP) do |event|
  info "This is my first OEDL experiment"
  group('Actor').exec('/bin/hostname')
  after 5.seconds do
    Experiment.done
  end
end
```

Figure 23: Experiment described in OEDL language

The main features of the OEDL can be distinguished into different categories:

Application Definition

In this category all application resources taking part in the experiment are defined. Each application defines specific properties like input parameters, measurement points, and application specific binaries. Some specific constructs derived from this category are *defApplication* command is used to define a new application. Such defined application can be used in any group of nodes as required. The *defProperty* command defines an experiment property or an application property. Experiment property is a variable definition that can be used anywhere inside the OEDL code. The *defMeasurement* command is used to define a single Measurement Point (MP) inside an application definition. Also, the *defMetric* command defines different output formats for the given MP.

Group Definition

In the group definition, a group of nodes are defined and combined with the applications defined earlier. In this category, a number of OEDL specific constructs are used. The first one is *defGroup* which is used to define group of nodes. *addApplication* is the second command used and it adds application into the group of nodes from the application pool. Since it is possible to add a number of identical applications within a single group, it is good practice to give unique IDs to each added application.

Timeline Definition section

The timeline definition defines the starting and stopping moments of each defined applications within each group of nodes. Inside OEDL language, events play a huge role. An *Event* is a physical occurrence of a certain condition and an event handler performs a specific task when the event is triggered.

Mapping OEDL to XML experiment description

Recall from experiment definition tool section that an experimenter passes three steps to finish configuring its experiment and produce XML, OEDL. It is hidden to the experimenter however that an OEDL file is generated from the XML file. During the making process of configuration package, an XML template is first produced out of which the OEDL file is constructed. The mapping of OEDL to XML is straightforward. It follows a one to one mapping except rearrangement of text. Federation Scenario Description Language (FSDL)

The Federation Scenario Description Language (FSDL) is a domain specific language for specifying experimentation scenarios. FSDL provides an abstract syntax of a resource broker meta-model. The FSDL's concrete syntax is supported by a text editor that implements instances of this meta-model. Syntax highlighting, context assistance, validation errors and warnings are some of the features. FSDL is available for the Eclipse workbench and the specification

framework is provided as Eclipse plugins. Together they form the Federation Scenario Toolkit (FSToolkit) [9].

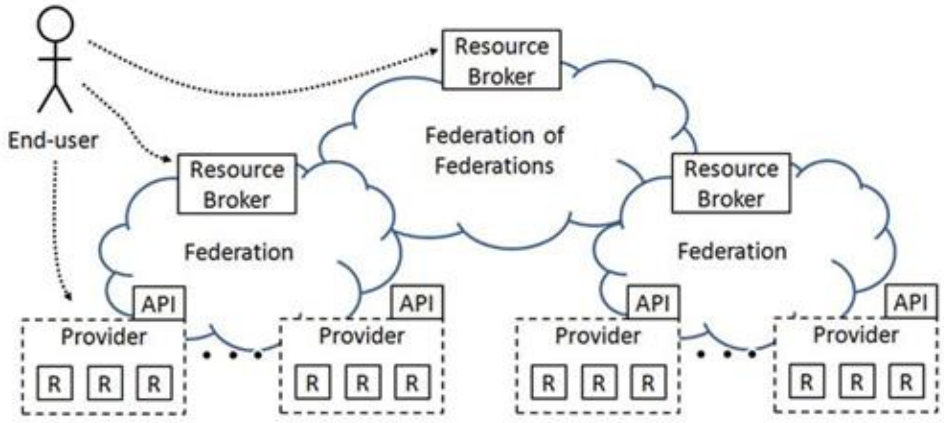


Figure 24: – Federation of resource providers

Figure 24 displays resource providers forming a federation where a resource broker is capable of exposing resources R to end-users in a uniform manner to create richer infrastructures. Resource providers must have an API that exposes resources and enables brokers to browse and manage them. The figure displays also that it's possible to create a federation of federations for even richer environments. The end-user can create scenarios involving resources in three different ways: by directly going to a resource provider, by going to a resource broker of a federation or finally to a large resource broker of the upper federation.

The FSToolkit tool is used in the Teagle framework for defining experiments in a textual way. It enables inter-testbed federation scenarios by accessing different testbeds via different authentication methods and API schemes. Thus, it is made possible that a Federation is created for this experiment on behalf of a trusted user. Usually the user should use different tools for each testbed and configure resources for each testbed. Then by applying proper configurations to each testbed (i.e. public IPs on machines, installing and configuring applications) can create the experimentation scenario. FSToolkit makes it possible to automate the whole process and enable the formation of an inter-testbed federation scenario. In OpenLab [10], FSToolkit supports the SFA API and thus it is able to provision resources in SFA enabled testbeds and federations.

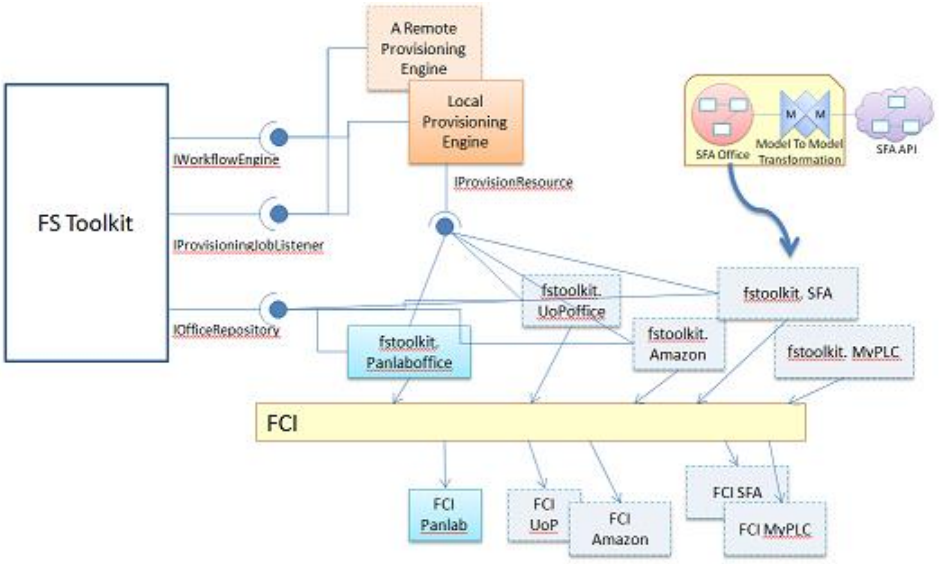


Figure 25: Plugins enable access to testbeds

The tool can handle multiple accounts even if there are many facilities offering the same API. For example the user can configure multiple SFA accounts on many SFA enabled sites, public or private. Identity is used by the tool for accessing the facility on behalf of the user. Queries are made against the facility in order to find out offered resources to the user. Identity information is also used for provisioning the resources to each individual facility. If the facility offers enough detailed information regarding for example, resource constraints, reservation information or other policies, these are mapped to the editor in order to ease the description of the experimentation scenario.

In FSDL the experimenter can create an experimentation scenario that contains requested services or resources with their configurations. In the simplest usage an FSDL definition starts with the keyword *RequestedFederationScenario?* followed by a name. A set of import broker statements that contain definitions of the brokers (the resource brokers, services and resources) may follow. Next, one can define either a resource agnostic scenario request or specific resources of providers.

5.2.1.2 IPAC Application Description Language (ADL)

The IPAC Application Description Language (ADL) is the most important component of the IPAC Application Creation Environment (ACE). It provides all the necessary elements and structures that are required for the definition of an application in the context of the IPAC project [11]. Its syntax is simple and combines some common characteristics of well-known programming languages. It follows a Model-Driven Architecture (MDA) approach. The IPAC ADL consists of the basis of the ACE editors providing the necessary elements for the creation of an application workflow. Though its syntax is simple, it covers all the aspects for applications development. Some of its characteristics are common with other known programming languages such as iteration commands, etc. The ADL is based on the XText framework (Eclipse XText

Framework) [12] that is used for the definition of the language model and the automatic generation of the appropriate tools. The ADL grammar contains a number of BNF-like rules, defined for every element of the language. Rules depict the functionality of each element and they are a combination of static and dynamic parts.

5.2.1.3 Service Control Language (SCL)

The SCL language both integrates the main features of a programming language and allows access to the various capabilities of the platform that was developed under the Polos project [13]. As far as the SCL part of a service is concerned, a service consists of one or more entry points, which are mapped 1-to-1 to the business methods of the service EJB that is subsequently generated and deployed on the PoLoS platform. Each entry accepts input parameters, returns a result, and consists of a sequence of commands that allow taking the following actions:

- Variable definition and assignment
- Writing and retrieving variables to/from three specific repositories
- Operations on variables
- Indirect referencing of variables
- Invocation of PoLoS platform components
- Scheduling the execution of another service
- Invocation of other entries, services, Java methods
- Loops
- Conditional execution
- Error handling similar to the Java try/catch mechanism.

Development of the Service Creation Environment (SCE) was focussed on the design of an appropriate service specification language, capable to support the description of the functionality pertaining to each Location-based service (LBS). This language is based on similar languages that exist today. The language is flexible and easy to use in order to allow for the specification and easy deployment of any type of LBS without too much effort and cost from the service provider. Each service will be defined through scripts written in this new language. Such scripts run within the PoLoS kernel. To render the use of the language as efficient as possible, the project pursues the design of an integrated development environment (IDE) that will greatly facilitate the development and deployment of new services.

5.2.1.4 Dependability Specification Language (DSL)

The dependability specification language is an integrated language that enables the user to define dependability requirements for different operation states of the program. Together with the protocol and environment model, this dependability specification forms the basis for the protocol selection and configuration. Two strategies to integrate the dependability specification with different WSN programming languages are proposed. The first integration target is a Java-like, high-level macro-programming language developed in the 7th framework program project

makeSense [14]. The second integration combines the specification with standard C code for the Contiki platform [15]. Both solutions build on a common XML based specification language. This DSL language was developed under the RELYonIt project [3].

The proposed DSL language focuses on three key applications properties.

a. Lifetime
b. Data yield
c. Latency

The notion of dependability includes four major aspects:

- Availability
- Reliability
- Safety
- Security

Requirements of the DSL

The dependability specification language must meet some key requirements such as:

- it must empower developers with ways to express dependability requirements on the underlying network protocols
- it must provide inputs for protocols selection and parameterization , thus being amenable to automated processing by dedicated optimization tools
- it must allow embedding within existing programming languages to allow the specification of different requirements based on an application's execution state

5.2.2 Authentication mechanism

Authentication is the process of ascertaining that somebody really is who he claims to be – short: it verifies "who you are".

Beside the simple username/password authentication, the following sections investigate special authentication mechanism more in detail: Federated Identity, Single-Sign-On and X.509 client certificate authentication.

5.2.2.1 Federated Identity

Federated identity is the means of storing electronic identity and attributes across multiple distinct identity management systems. This means that a service or application does not need the users credentials in order to authenticate users but relies on another trusted service or application that is already storing the users' electronic identity.

These are three different Federated Identity standards that were originally built to address very different needs:

- SAML [16] (developed in 2002 by the OASIS) is an XML-based open standard for exchanging authentication and authorization between parties. Its main purpose was to facilitate Single Sign-On (SSO) for enterprise users.
- OpenID [17] is an open standard released in 2006 with the same purpose as SAML (SSO) but for consumer apps and services
- OAuth [18] also became available in 2006 as an open standard to allow apps to share information via APIs with the right level of authorization

It is important to note that SAML and OpenID were both authentication protocols, but OAuth was initially an authorization protocol. SAML further developed to version 2.0 in 2005. The standards OpenID and OAuth also evolve to OAuth 2.0 (2012) and OpenID Connect [19] (2014), which is based on OAuth 2.0.

SAML is a quite complex standard. The SAML standard specifies three components: assertions, protocol, and binding. There are three assertions: authentication, attribute, and authorization. Authentication assertion validates the identity of the user, attribute assertion contains specific information about the user, and authorization assertion identifies what the user is authorized to do. Protocol defines how SAML asks for and receives assertions. SAML works with multiple protocols including Hypertext Transfer Protocol (HTTP), Simple Mail Transfer Protocol (SMTP), File Transfer Protocol (FTP) and also supports SOAP, BizTalk, and Electronic Business XML (ebXML). A SAML binding is a mapping of a SAML protocol message onto standard messaging formats and/or communications protocols. For example, the SAML over SOAP or SAML over HTTP. Depending on the protocol and binding chosen, the communication flows between the parties can vary greatly.

OpenID Connect is focuses on client developer simplicity: It used a REST API together with JSON formatted messages. It provides specific authentication and authorization flows for web applications, desktop applications and mobile phones.

If just authentication is needed, also X.509 client certificate (see section 5.2.2.3) could be used, by installing the root certificates of trusted CA on the service provider.

5.2.2.2 Single-Sign-On (SSO)

SSO is the process of access control of multiple related, but independent software systems. Different approaches are discussed in the following.

5.2.2.2.1 Kerberos

A widely used protocol is Kerberos [20]. In Kerberos three parties are involved: the client, the server that the client wants to use, and the Kerberos Server. Kerberos uses tickets for authentication. To use the Kerberos service, a client must first log in to the Kerberos server. He asks the Kerberos server for a Ticket Granting Ticket (TGT). With the TGT, the client is able to request additional tickets for services without having to enter a password again.

5.2.2.2.2 Client certificate

Certificates installed on the client machine may also be used to setup a SSO environment. See section 5.2.2.3 for more details.

5.2.2.2.3 Cookie based Web Browser SSO

Using redirects and cookies on a central login domain, it is also possible to implement a SSO for web applications. The main problem is that cookies are only valid for a specific domain. To have a Cross-Domain Single Sign-On the following steps could be performed (this is just an example, different implementation may differ):

- The user attempts to access a resource (e.g. */hello.html*) at the application domain (e.g. *example.com)*
- The agent realizes that the resource access restrictions and that there is no active session yet (unable to find the user's session cookie), so it redirects the user to central user session page on the user management domain (e.g. *users.com/check)* and providing the requested resource URL as parameter
- When the user accesses the central user session page, the agent is able to detect whether the user has an active session (hence a previously created session cookie should be visible there)
- When there is no session, the user is redirected to sign in page (e.g. *users.com/signin*),
    - The user submits his credentials.
    - If the credentials were correct, we redirect the user back to the session page (e.g. *users.com/check*)
- Now there is a valid session (cookie) for the user management domain (e.g. *users.com*)
- The session page (e.g. *users.com/check)* now redirect this user to the source web application (e.g. *example.com/sso*), while also transmitting user ID, session ID and some other encrypted data to verify the login on the source web application. (Transmission of all the data could be done by displaying a self-submitting HTML form to the user or by using redirects together with URL parameters)
- The agent on web application checks the validity of the posted data (show error if not)
- A session cookie for the application's domain (e.g. *example.com)* is created for the user.
- The user is again redirected to the initial requested resource (e.g. *example.com/hello.html).*
- The web page sees now the session cookie on the application domain (e.g. *example.com),* hence the user is logged in this web application.
- The agent checks authorization (is the user allowed to access this web page), different methods may be used for this depending on the standard.
    - If allowed: The user sees the requested content
    - If not allowed: The user sees an HTTP 403 Forbidden page.
- For subsequent requests the web page will also see the session cookie on the application domain (e.g. *example.com)*

SAML Web Browser SSO Profile and OpenID cConnet (see section 5.2.2.1) use similar methods to enable SSO inside a web browser.

5.2.2.3 X.509 client certificate authentication

The X.509v3 [21] standard forms the basis for digital certificates. These identify a person, a service or a server. The most important application is the authentication of servers in SSL / TLS (Secure Sockets Layer / Transport Layer Security) – SSLv3. But SSLv3 also supports authentication of clients. It is a mutual authentication within which the connection is established, both the server - as always with SSL - and the client will be authenticated.

To use it, all relevant applications (like the browser) need to support SSLv3. Additionally the digital certificate need to be present on the client computer: it can be stored locally or on an external storage (usually a smartcard). Browsers (like Internet Explorer or Firefox) provide tools for managing the certificates.

The client certificate itself is created by a certification authority (CA), which signs the certificate with an own (root) certificate (Chain of trust).

The applications themselves react automatically on certificate requests, the user just need to initially unlock the certificate store. For the Internet Explorer this is done with the Authentication on Windows, smartcards are usually unlocked using a PIN number.

The authentication via client certificates could also be used as SSO solution: As the authentication runs in background, it is transparent for the user how often the certificate is presented to different servers. In addition, the developer do not need to worry about lifetime for browser cookies or SSO over several web domains.

On the server side, the options to enable authentication via client certificates need to be enabled. Additionally the certificate of the signing CA needs to be registered as root certificate in the key store of the server.

X.509 certificate authentication also used in several other FIRE projects E.g. in Fed4Fire [1] uses this to enable user from other trusted organisations to login into their platform. This works because they have imported the CA certificates of the trusted organisations.

5.2.3 Data analysis

Data streaming analysis is a quite young field as a result there is a quite limited set of available open source systems that could be used in such setting and all of them have considerable limitations, namely in terms of the quite reduced set data streaming analytics algorithms they deliver.

- Data Stream Processing Engine (DSPE) + Samoa: SAMOA (Scalable Advanced Massive Online Analysis) is a platform for mining big data streams. It is a distributed streaming machine learning (ML) framework that contains a programming abstraction for distributed streaming ML algorithms. Apache SAMOA enables development of new ML algorithms without directly dealing with the complexity of underlying distributed stream

processing engines (DSPEs, such as Apache Storm, Apache S4, and Apache Samza). Apache SAMOA users can develop distributed streaming ML algorithms once and execute them on multiple DSPEs.

- Spark Streaming + Spark's Machine Learning Library (MLlib): Spark Streaming is an extension of the core Spark API that enables high-throughput, fault-tolerant stream processing of live data streams. Data can be ingested from many sources like Kafka and be processed using complex algorithms expressed with high-level functions like map, reduce, join and window. Finally, processed data can be pushed out to filesystems, databases, and live dashboards. Spark provides built-in machine learning algorithms from its Machine Learning library (MLlib), and graph processing algorithms through its graph and graph-parallel processing API (GraphX) on data streams.

MLlib provides a distributed machine learning (ML) library to address the growing need for scalable ML. MLlib is developed in Spark [69], a cluster computing system designed for iterative computation. Moreover, it is a component of a larger system called MLbase [70] that aims to provide user-friendly distributed ML functionality both for ML researchers and domain experts. MLlib currently consists of scalable implementations of algorithms for classification, regression, collaborative filtering and clustering.

Comparison between the solutions Spark Streaming + MLlib and Storm + Samoa:

Comparative feature-by-feature table:

	Spark Streaming + MLlib	**Storm + Samoa**
Status	Under the Apache Software Foundation	Undergoing Incubation at the Apache Software Foundation
Processing Engine	Spark Engine (batch) - Spark Streaming =microbatch	Storm Engine (stream)
Distributed	Yes	Yes
Open Source	Yes	Yes
Machine Learning library	MLlib	Samoa
Stream source	Kafka, Flume, HDFS (storm entity associated to sources : spouts)	Kafka, HDFS
Stream primitive	Tuple	Dstream
Computation	Bolts	Transformation, window operation
Stateful operations	No	Yes
Output/persistence	Bolts	foreachRDD
Hadoop distribution	Hortonworks, MapR	Hortonworks, Cloudera, MapR
Language option	Java Scala Python	Java Clojure Scala Python

		Ruby *enable the use of virtually any programing language given the right binding
Performance (engine)	400,000 records/s/node	10,000 records/s/node
Activity	Under active development	Incubation state

Table 40: Comparison of features providedfeature by Spark Streaming/MLlib and Storm/Samoafeature

Topology of both solutions:

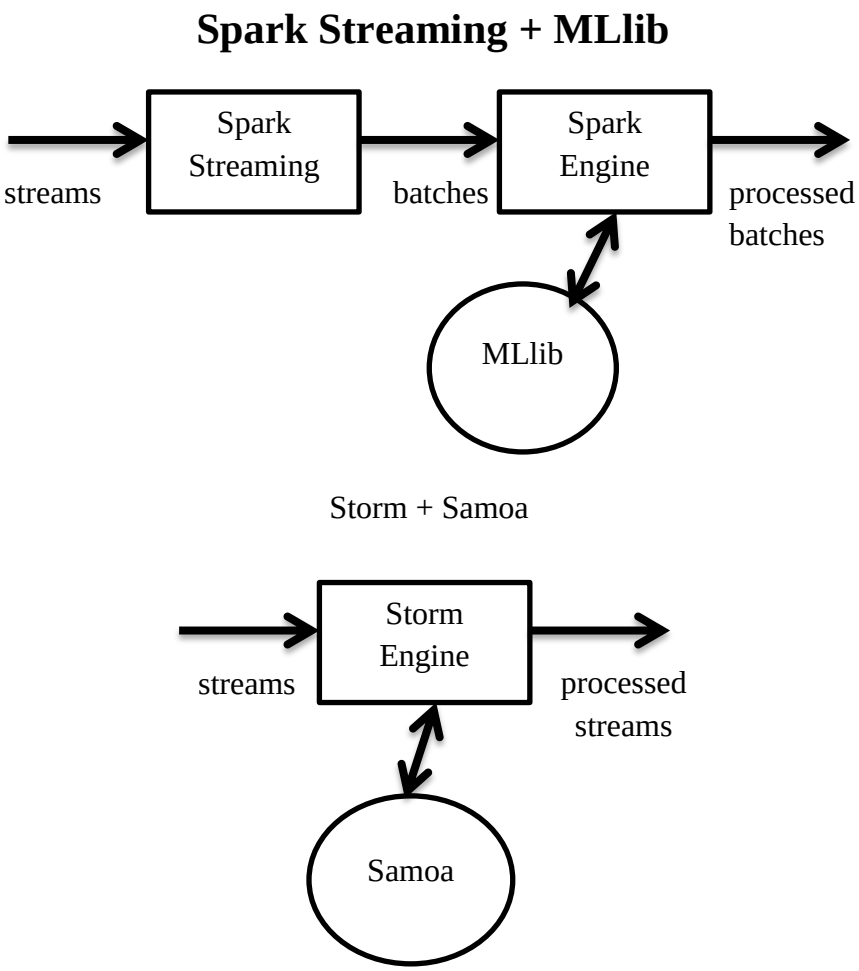


Figure 26 - Topologies of Spark Streaming + MLlib and Storm + Samoa

Machine Learning algorithms provided:

When it comes down to the machine learning analytical tool that has to be developed, both solutions can be used in RAWFIE but they do not directly provide what is needed. MLlib offers a large and active set of machine learning algorithms but these are not adapted to the streaming nature of the data that we will be dealing. Samoa on the other hand provides learning and mining

algorithms for data streams, however the number of available algorithms is quite limited. The main part of the data analytical project effort will thus go to the design and development of new learning and mining algorithms for data streams in order to complement the limited pool of available algorithms.

Currently available algorithms are:

MLlib	**Samoa**
- Basic statistics : summary statistics, correlations, stratified sampling, hypothesis testing, random data generation - Classification and regression : linear models (SVMs, logistic regression, linear regression), naive Bayes, decision trees, ensembles of trees (Random Forests and Gradient-Boosted Trees), isotonic regression - Collaborative filtering: alternating least squares (ALS) - Clustering : k-means, Gaussian mixture, power iteration clustering (PIC), latent Dirichlet allocation (LDA), streaming k-means - Dimensionality reduction : singular value decomposition (SVD), principal component analysis (PCA) - Feature extraction and transformation - Frequent pattern mining: FP-growth -Optimization (developer): stochastic gradient descent, limited-memory BFGS (L-BFGS)	- Prequential Evaluation Task - Vertical Hoeffding Tree Classifier - Adaptive Model Rules Regressor - Bagging and Boosting - Distributed Stream Clustering - Distributed Stream Frequent Itemset Mining - Every ML algorithm **MOA** offers : - Classification: Bayesian classifiers, Decision trees classifiers, Meta classifiers, Function classifiers, Drift classifiers, Multi-label classifiers, Active learning classifiers - Regression: FIMTDD, AMRules - Clustering: StreamKM++, CluStream, ClusTree, D-Stream, CobWeb. - Outlier detection : STORM, Abstract-C, COD, MCOD, AnyOut -Recommender systems: BRISMF Predictor -Frequent pattern mining: Itemsets, Graphs -Change detection algorithms

Table 41: Supported Algorithms in MLlib and Samoa:

5.2.4 Navigation mechanism for UxVs

In the context of the project, the experimenters will have the ability to navigate the UxVs either through the launching tool (by executing a scenario), or directly via the supplied remote control. The instructions of the operators will be translated/converted into waypoints and in the sequel, they will be transmitted to the vehicles in the form of a compact message. These messages consist of a set of coordinates for each UxV, describing their next desired location together with information regarding their orientation.

Waypoint navigation is now considered as a standard practice and it is extensively used in the modern literature mainly because of the following reasons:

1. It allows the utilization of heavyweight - but extremely accurate - navigation algorithms on UxVs with reduced computing power. These navigation methodologies are executed in ground control units and their results are transmitted in the form of compact messages to the UxVs.
2. It allows the interaction of the operator with the vehicles at each time step. Even in cases where the UxVs are executing a predefined mission, the operator has the ability to step in and modify the scenario according to the needs that may arise.

On the other side of the spectrum, waypoint navigation requires continuous data exchange between vehicles and the ground control unit. But, the utilization of compact messages, the size of which is limited to few bytes per message, makes this procedure feasible and efficient.

Once the message is received by the UxVs, an on-board microcontroller takes over the manipulation of the motion control parts of the vehicle, so that it reaches the desired location. This controller is also known as "motion controller" and undertakes for example, in the case of aerial robots, the rotational speed of the propeller, the pitch angle etc. There has been an intensive effort, especially recently, towards developing efficient motion control methodologies for aerial and ground robots, and their results are extremely encouraging. On the other hand, the problem of accurate and efficient motion control designs for underwater vehicles still remains an open issue [47][48].

A recent example in which the waypoint navigation approach was successfully employed is the EU-FP7 sFly project [49]. Among the objectives of the project was also the deployment of a team of flying robots so as to perform surveillance coverage missions over unknown terrain of complex and non-convex morphology. According to the objective of the mission, the robots attempt to maximize the part of the terrain that is visible while keeping the distance between each point in the terrain and the closest team member as small as possible. A ground station receives from every robot, its location and information regarding its field of view and, at every time-step, transmits at every vehicle the new desired position (after an appropriate validation of the destination). An on-board micro-motion controller translates the received messages and guides the UAV to the next position.

In another example, within the EU-FP7 NOPTILUS project [50], the waypoint navigation approach enables a fully functional methodology for cooperative, fully-autonomous navigation of teams of AUVs when deployed in static or dynamic underwater map construction or dynamic underwater phenomena tracking missions. In this case, a web-service produces waypoints with the next optimum position for each AUV. Next, these waypoints are transferred to the vehicles through acoustic modems and the AUVs are reaching the desired location. The highly nonlinear nature of the AUV dynamics, the presence of strong currents and turbulences render the problem of motion control of underwater vehicles a very challenging task. For this reason, a new adaptive-based motion control methodology for AUVs [51] was presented and evaluated in real world experiments. In the sequel, the vehicles are transmitting back to the web-service

measurements from their equipped sensors together with their exact locations, so as to fix possible localization issues.

5.2.5 Device communication for UxVs

This paragraph presents the challenges faced by the RAWFIE UxV designers and manufacturers in the choice of communication system to be used. It presents and discusses some of the responses communication. The present chapter does not address the on-board communication means, which are typically using wired communication means. They may eventually be implemented using wireless solutions, whenever the wiring becomes overly complex or the weight of the cables is compromising the useful UxV payload or autonomy. In such case, a care must be brought to the coexistence of these implementations and the surrounding wireless context.

The communication with UxV supports many functions, the most obvious being UxV control, sensor sample collection, including video of the environment, commands, UxV status, etc.

Historically, the unmanned vehicles were first remotely controlled by directly acting on the control devices through physically acting on them by connecting stiff wires between them and the controller (typical of circular flight), or driving the actuators through analogue electrical signal sent through metallic wires. Nowadays, UxV commonly typically rely on wireless communication means, be it infrared or electromagnetic. Small UxVs operating indoors are often using infrared transceivers for their low cost and appropriate range. Many other UxVs (indoor and outdoor) are using electromagnetic wireless systems, which are shifting from analogue (typically in the 27 MHz band) to digital (2.4 GHz unlicensed band is very popular), in particular to provide a better coexistence of remotely controlled UxV in the same area.

Due to the national regulations, the UxV are often limited to Line of Sight (LoS) range, which is corresponds in general to the best cases of using direct wireless control. These solutions are based on short range wireless of technologies such as IEEE 802.11 or Bluetooth in all of their flavours. Bluetooth is based on connections that may require up to a few seconds for establishment before traffic may take place. IEEE 802.11 is quite performing in the ad-hoc mode but at the expense of high consumption. Other technologies, developed under the generic name of “wireless sensor networks”, are interesting and promising solutions because they exhibits low power consumption, relatively high bit rates (up to 1Mbit/s) and are now available in single chip or small packages solutions.

Some systems are extending the range of the wireless link by using high gain directive antennas, but low latency and availability requirements in the interaction discard most of these solutions in professional systems. Some professional systems are using the data communication services offered by telecom operators (based on GSM of any generation) or satellite service providers, typically for uploading sampled data and device status or for receiving commands, software

updates, etc. Still low-latency (or high reactivity) is difficult to provide to highly mobile, high speed or long-range vehicles.

Over-the-horizon (OTH) communication raises a number of issues that are partly solved by using satellite relay, for example UHF Satcom services, providing a data rate of 16 kbit/s. However, due to satellite response delays, the real-time control can not be implemented over such link, even though high-level commands, such as waypoints, can be transmitted to the UxV. [23]

More expensive or critical systems can rely on a specific communication infrastructure, fixed or mobile, or on hybrid configurations involving several types of UxVs, such as UAVs and USVs in which the UAV acts as a relay for the USVs.[63]

Specific requirements are put by the UxV operations. This is the case in particular when the UxV are operated in complex, changing or unknown environments or in combination with other UxV (collaborative teams of UxVs, swarms) and other sensors disseminated across the testbed. On a qualitative standpoint, these requirements may impose ad-hoc communication principles, such as connection-less interactions, delay-tolerant networking, multi-hop relay-based transmission, combined with the support of real-time and highly reconfigurable communication topologies. It is clear that no such technologies may comply to all of these requirements. The approach is therefore to select a set of technologies, each of which would be appropriate to meet some of the requirements. Furthermore, requirements can be inconsistent or contradictory when considered globally. Once consistent sets of requirements are grouped together, then technologies may be selected to cover each group.

There are three types of UxV control, from the most to least demanding to in terms of quality of service for the communication and from the least to the most demanding in terms of on-board resources: Man-in-the-loop or manual (the man is controlling the UxV in real-time, typical of radio-controlled UxV) vs. Man-on-the-loop or semi-autonomous (mission is programmed in advance and monitored, updates of the plan are transmitted to the UxV, as well as human intervention, e.g. real-time compensation of deviations) vs. Autonomous (the UxV is acting accordingly to the plan and adapts its behavior depending on an purely local control loop, potentially aided by on-board heuristics). Usually, the autonomous systems can also be controlled in semi-autonomous and manual modes. UxV may switch to any of the modes it is supporting.

Commercial UxVs are usually communicating with remote controller via RF links in the ISM bands (2.4 GHz or 5.6 GHz). More and more systems are using WIFI. Some UxV uses approaches such as sandboxing or separation of concerns, to increase the robustness: for example, they separate communication channels for the control function of the UxV (2.4 GHz) and the sensory information transmission (5.8 GHz) [23].

Beyond proprietary representation of the sampled data, status or commands, the industry has issued a number of recommendations and standards for allowing a relative vendor independent

market to emerge. The recent standards are based on XML, by specialising it to domains or technologies. Standardisation is especially relevant for the data representation, for being re-used and transferable to other bodies. Furthermore, the semantics of the carried information has also been addressed under several initiatives towards interoperable and interchangeable systems, processes and data exploitation. However, standards are fixed and every evolution is an issue for the compatibility across platforms of different generations. That is the rationale behind XML, which can embed the definition of the data representation.

Synchronisation is a function that is necessary to any entity participating to the testbed operation. In general, GPS can be used as a global clock. Its precision is about 50 ns. However, its cost and power consumption do not allow for its integration in all devices (e.g. simple low-cost battery-operated sensors). Other techniques can be used, in particular those exploiting the available communication properties.

5.2.6.1 Unmanned Aerial Vehicles

Size and weight are critical to UAVs, especially when long range and long life lifetime is sought after. The power vs. weight ratio is of primary importance for reaching performance objectives (speed, range, manoeuvrability, payload, etc.). Since the UAVs control relies on “sense and avoid”, the primary concern for UAVs is to reduce the latency in the interactions between the controller and the vehicles. This is very difficult to meet this constraint in a wireless environment. In addition, since the wireless medium is shared among all UxVs, the access rules must guarantee a certain level of fairness and or priorities across the network to allow for emergency situations to be managed appropriately. [62]

Today, the most popular communication link for UAVs is based on IEEE 802.11 (WIFI), in all its flavours, which provides a range of about 100 to 300 m in the best conditions. Directional antennas may increase slightly this range at the risk of losing the connection when the vehicle moves out of the transmission cone, which is a serious event for a remotely controlled aerial vehicle. WIMAX and AirMAX were candidates for increasing the range of such wireless links in addition to offering large datarates, but these technologies did not gain popularity and were seldom deployed. 3G-LTE is also an option. Other issues include the absence of mobility support in the traditional Local Area Network technologies such as WIFI or Bluetooth. WIMAX has a support for slow mobility (10 km/h), which does not meet the requirement of swift UAVs, not even considering the induced Doppler effect on the wireless signal. [62]

Light UAV often relied in the past on dedicated and proprietary wireless communication systems, typically using infrared or radio-frequency physical layers. Nowadays, a lot of models are using standard RF wireless technologies, among which Bluetooth and WIFI are the most popular. They are also often controlled using dedicated controller or directly from tablets or smartphones. The Wimax and Ubiquiti AirMax [63] and technologies have also been used in aerial vehicles, but these technologies have not gained momentum.

5.2.6.2 Other aerial communication technologies

Drones become major components of military applications and deployments. Although military applications are out of the RAWFIE scope and that their specification may be far off the RAWFIE target, the underlying technologies are mentioned for several reasons: the required quality of service is higher and the requirements are more demanding; the operational environments are often harsh; the missions assigned to UxV are often critical; they put the bar as high as the most demanding application today and may be regarded as the ultimate target to be achieved in the next generation solutions. Until now, military devices were based on components specifically developed for the mission or mission families, but the military budget reductions led to the search for solutions produced in greater volumes, on which the development cost can be spread thin. More and more consumer grade and industrial grade designs and products are used in military devices and systems, which, in turn, may be reused directly or indirectly in civil applications (e.g. GPS). It is therefore not excluded that the following technologies or components will be found in commercial UxVs in the future.

The Predator UAV: "Externally accessible communications capabilities include HF, UHF, VHF radios, tactical voice and direct electronic connectivity to the UAV communications system, such as the AN/TSQ-190 TROJAN SPIRIT II, USAF Theater Deployable Communications (TDC), or Very Small Aperture Terminal (VSAT) for dissemination of all types of collected products." It includes Ku band Satcom, GBS at 6Mbit/s and LoS communication at 4.5 Mbit/s in C band. It is planned to integrate VHF/UHF radio relays to provide AVOs communications capability with air traffic control. [22]

The Global Hawk and DarkStar includes mission control (MCE) and ground communications (GCE) elements. The LoS communication system is based on the Common Data Link (CDL) technology, which supports supports 274 Mbit/s (downlink) and a 200 kbit/s (uplink). The communication for tactical voice and connectivity to TROJAN SPIRIT II (or LMST/ICAP) and theater communications networks between the Global Hawk and the GCE is supported by HF, UHF, VHF radios (SHF: 1.544 Mbit/s for DarkStar and 50 Mbit/s for the Global Hawk). "A point-to-point data link will enable UAV ground segments to send downlinked video/frame sensor/other unexploited products directly to a collocated or geographically separate exploitation system. Desired data rates range from 45 Mbit/s (T-3) to 1.5 Mbit/s (T-1)." [22]

The HAE UAVs will take advantage of future advancements in data link technology. The ABIT program provides a wide band (274 Mbit/s) air-to-air relay of imagery using a UAV as a collector and another as the relay vehicle. The CDL LOS remains the same between the relay aircraft and the ground segment. The concept will extend the range for LOS data collection by 400-500 NM. [22]

Note that many of the military solutions partly rely on satellite data transmission (in addition to positioning, clock synchronisation and observation data). This approach has great advantages

such as the global reach on large areas, possibly global when using a satellite constellation. It has drawbacks as well, such as introducing delays in the communication (especially when using geostationary satellites) and hieratic quality and coverage in anything else than a flat and sunny area. For example, subterranean areas, tunnels, building indoors, etc. are not convered by satellite. If needed, signals from and to satellites may be forwarded using relaying repeaters between covered and uncovered areas. Flat and lightweight antennas are developed to reduce their payload footprint so that they can used on a broader range of UAVs, including small ones that have difficulties to beam and catch the satellite feed because they stabilisation difficulties.

5.2.6.3 Unmanned Ground Vehicles

Unmanned Ground Vehicles (UGV) are less constrained than UAV because the trade-off between energy and weight is much looser. The fact that the UGV may often stay immobile for long periods of time without catastrophic consequences allows for more relaxed requirements on the communication means. Other constraints are nevertheless observed due to specific requirements such as for all-terrain ability, speed, traffic, regulations, etc. Still the required qualities of service include the support for sensitive, time-critical (low latency) and exchange of mission-critical sensory information. In the past, a very high rate of failures was experienced on the communication between operators and UGV: “Wireless communications is a known problem in field environments” [65].

In spite of all its limitations, IEEE 802.11b is still used in much experimentation and commercial products are now using the latest evolutions of the same standard. Interesting approaches combine two different UxV types to achieve a given mission, during which the UAV provides communication relay capabilities to the UGV in the case the wireless infrastructure is out of reach of the UGV [64]. Specific radio relay infrastructures have also been proposed to compensate the loss of signal between the UGV and the ground control station in BLOS operations [66].

5.2.6.4 Unmanned Underground Vehicles

Building, mines or cave exploration probes, are examples of Unmanned Underground Vehicles. They are usually not very mobile and most remain tethered. They are typically connected using wires. When mobility is required, they can use wireless communication systems such as Wifi if the distance is short (a few meters at most), medium frequency (in the range of 500 kHz) or specific sound or vibration-based low data-rate communication systems (Through-the-earth communication operating at frequencies below 10 kHz).

A technique called leaky feeding has been developed for the mining and tunnel industry. It is based on a wired infrastructure made of coaxial cables that act as antennas; this allows for any devices for wirelessly interact with the devices that are connected to the cable antenna. This can be used in building as well. Note that the medium frequency based systems may use a similar

approach, since they would benefit from the wave guiding properties of any present metallic environment.

5.2.6.5 Unmanned (water) Surface Vehicles

Unmanned (water) Surface Vehicles can use similar technologies to those used by ground or aerial vehicles. Currently, WIFI has been successfully experimented for medium range communication between the shore and USV over a few nautical miles [Bibuli Marco, Massimo Caccia, Lionel Lapierre, Bruzzone Gabriele. Control of Un-manned Surface Vehicles: Experiments in Vehicle Following. IEEE Robotics and Automation Magazine, Institute of Electrical and Electronics Engineers (IEEE), 2012, pp.92-102.], using directional antennas on the shore and omnidirectional antennas on the vehicle. The water environment imposes the antenna of the wireless system to be put at a minimum height to minimise the skin effect of the surface. In many cases, the use of directional antennas is preferred due to the scarce number of participants to the wireless networks typically found at sea. However, such antennas must be combined with other features such as stabilisation or multiple antenna inputs to compensate the effect of waves or wind. Omnidirectional antennas are also used when numerous participants are involved in the application or high-speed mobility is required [MOBESENS].

5.2.6.6 Unmanned Underwater Vehicles

Unmanned Underwater Vehicles traditionally use acoustic wave or optics based communication systems. Wave-based communication is operating in the frequency range from 10 Hz to 1MHz. Their range spans from a few hundreds of meters (vertical channel, from the surface to the seabed, with “high” datarate) to a few kilometres (horizontal channels, low datarate). They are used in telemetry, point to point communication, control of remotely operated vehicles and data exchange with autonomous vehicles. Usually, moored stations are relaying (as a gateway) the signal from and to underwater vehicles to the surface (e.g. to boats, planes, satellites or shore stations). Underwater vehicles can also be organised in ad-hoc and mesh networks, in which the surface relay acts as a sink or source of information. Optics based communication uses similar principles as in the fibre optics communication, with high-power LEDs (Superflux or LumiLED) around 500 nm wavelength and frequency modulation. It is used in line-of-sight configurations. Note that the LED-based communication setup can also be used as a sensor.

The performance of both approaches depends on the depth of operation (euphotic, disphotic or aphotic strates), which determines the attenuation and scattering level.

5.2.6 Cloud specifics

There are three distinguishing characteristics defining “Cloud computing” (see Figure 27):

1. involves ***virtualised*** resources (workloads are allocated among a multitude of interconnected computers acting as a single device);

2. is dynamically ***scalable*** (the system can be readily enlarged);

3. acts as a ***service*** (the software and data components are shared over the Internet).

The result is a "hosted elsewhere" environment for data and services, and advantageous environment for many data heavy and computationally demanding applications..

Figure 27 - Pros and Cons of Cloud computing

In RAWFIE, we propose to use a SaaS Cloud solution to host both the data repositories (part of the Data Tier) and all other software services running on the Frontend, Middle and Data Tiers. The Testbed Tier software and infrastructures (UxV resources and Base Station/tTesbed Proxy) will stay, of course, at the owner's premises, and will interact with the other Tiers via network interfaces through Internet.

Figure 28Figure 28 below shows the difference between the Cloud paradigms IaaS, PaaS and SaaS. As it is clear from this picture, the final business idea for RAWFIE would be to provide a platform hosted by a Cloud service provider (such as e.g. Amazon), with the running software completely managed by the RAWFIE consortium (SaaS mode) and therefore transparent for the end users. End users (Experimenters) will just access the platform from the online Web Portal for registration purposes, and for booking, running experiments and analysing results.

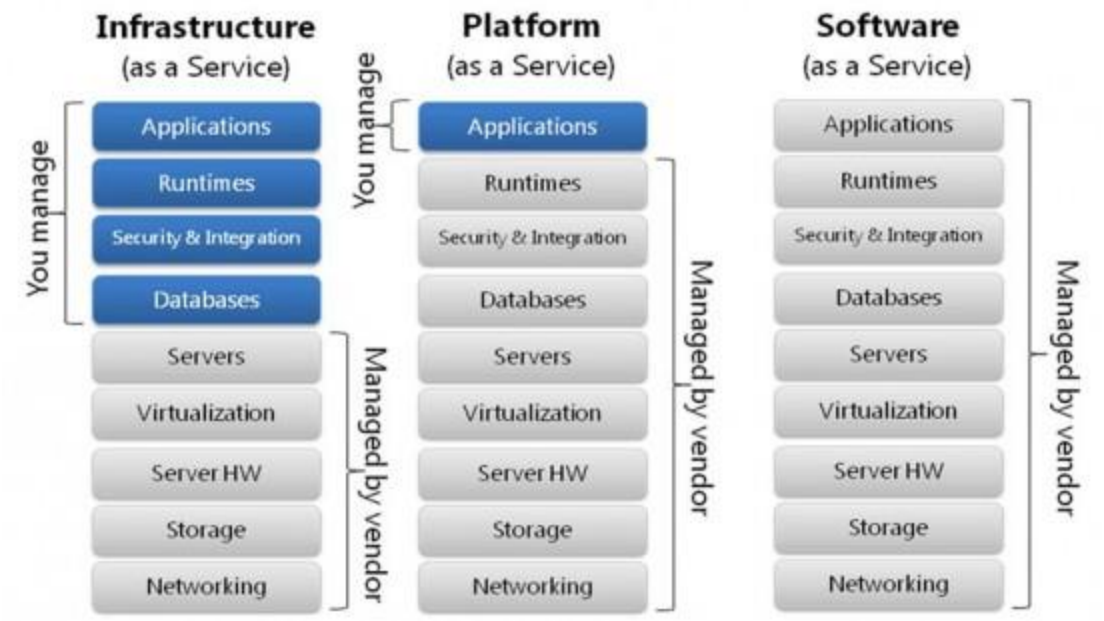


Figure 28: - Difference between IaaS, PaaS and SaaS. [24]

The Cloud is an architectural model that employs many of the same components used in datacenters around the world today, but in a more flexible, responsive, and efficient way. The primary difference is in how these components are tied together by using what is called a "dynamic control plane", which helps enlighten and inform the architecture about the rapidly changing requirements of today's applications and data (e.g. in terms of storage capacities or traffic volumes to handle). The "dynamic control plane" intercepts the traffic as it traverses the Cloud, interprets the data and instructs the Cloud architecture on how to efficiently connect the user to the appropriate application instance. In order to be ready for enterprise deployment, it must also be scalable, adaptable, extensible, manageable, and secure with real-time performance.

Among the commercially available solutions for hosting relational databases (for the data repositories) and services in the Cloud there is Amazon RDS (Amazon Relational Database Service) [71] and Amazon EC2 (Amazon Elastic Compute Cloud [72]). These Amazon services provide, in relation to the provision of a dynamic control plane, the possibility to:

- choose the needed server instances capacity (e.g. small, micro, and so on)
- get the needed server instance and configure it in such a way it will automatically scale up or down to a new server instance (vertical scaling) with different capacity in case more or less computational resources are required (e.g. CPU, RAM, Storage)
- get the needed server instances and configure them in such a way they can run in parallel, using a load balancer in charge of controlling and balancing the traffic among all available servers (horizontal scaling, i.e. parallel execution of software in multiple server instances)

When a Cloud architectural model is used, the software has to be designed in such a way to get real advantages from the possibilities described above. This is especially true when horizontal scalability is the target: specific precautions have to be adopted in order to implement software services that can run in a distributed way and in parallel in different servers. In practical terms, this means that the software components have to be designed by avoiding all elements, which link each transaction or communication flow to a single server instance. The typical example is a horizontally scaling scenario where requests from users can be dispatched by the load balancer to different servers in round-robin mode: a web application running on the Web Portal (Frontend Tier) for example, has to implement the ability to maintain users' sessions among different server instances.

For what concerns scalability of databases, while horizontal scalability can be achieved by using NoSQL database solutions such as CouchDB, MongoDB and Cassandra, it is hard to obtain scalability of write intensive applications using traditional relational databases solutions like MySQL or PostgreSQL. Amazon RDS provides a solution to cope with the needs of applications with read-heavy database workloads. The solution is called Read Replicas, and consists in creating many replicas of a single database instance. Once Read Replicas are created database updates on the source database instance will be replicated using an asynchronous replication. Multiple Read Replicas can be created for a given source database instance, and the application's read traffic distributed amongst them. For what concerns NoSQL databases, Amazon provides DynamoDB, a solution expected to provide high degree of scalability.

5.2.7 Data Pipeline Architecture:

To make our analytics platform viable we envision that the data will flow in the following manner listed below. This is a pretty standard pipeline that can be scaled at both the BUS level (kafka [60]) as well as the compute infrastructure level (Storm/Spark). We envision our analytics platform sitting to the side of storm/spark & passing jar jobs to it.

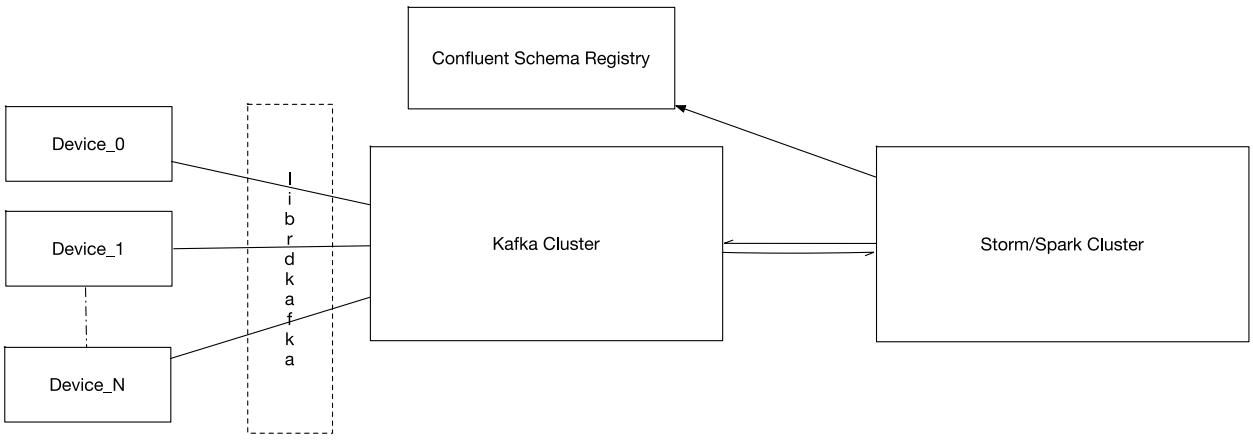


Figure 29 - Data pipeline architecture

- The diagram listed above is a standardized pipeline used in industry to pipe data into a compute cluster (such as storm or spark).
- Here we see that the storm/spark cluster can directly talk to the confluent REST interface to query the required schema. This ensures proper construction of RDD / batch data structure types.
- Data between the storm/spark cluster & kafka is bi-directional: i.e. a stream is read from & 'enhanced' with Machine Learning features such as anomaly detection, etc. This 'enhanced' stream is written to a different stream.
- We suggest confluent because of the power of it's schema evolution features. This allows for data structure to change over time (i.e. adding of fields/etc). More information can be found in [73].

Problem of connectivity: Having a kafka broker on each of the devices will be a massive amount of overhead. Furthermore there will always be re-balancing efforts taking place. Instead of that it is better to use a cyclic buffer within librdkafka to retain a manageable portion of messages. librdkafka also has the ability to batch compress messages (taking advantage of run-length-encoding). Finally, there are callback functions that can be utilized to verify message delivery.

The case for structured data: structured data on write will be immensely helpful as data will not need to be de-mangled at a later time. This also provides the ability to modularize 'revisions' of different data schemas. Devices can directly talk to the schema registry on initialization to get information about:

1. Stream's schema (and a specific revision if need be)
2. Kafka brokers.

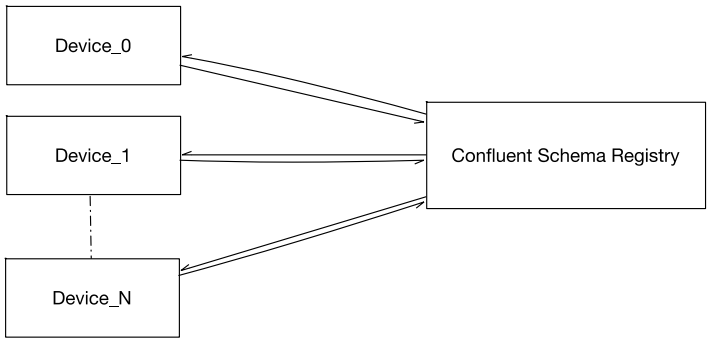


Figure 30 - Communication between devices and schema registry

5.2.8 Data storage

The data going into storm/spark can be tee'd off into HDFS storage for later consumption by databases such as Impala (relational high performance in-memory). Data storage will generally

need to initially be dumped into HDFS & then placed into a DB. Note: Hadoop is NOT required to have a database as the HDFS layer is independent of this. Please refer to the confluent diagram for more information regarding how the data can be tee'd off into HDFS for use in a database tool.

- **Relational**: relational databases are particularly useful if one needs to do SQL type operations such as joins or unions, etc. They generally have quite a bit of overhead as they need to index the data. Popular choices include Postgres & Impala. The use of Impala is highly recommended as it byte-compresses its data and has all of its operations written in C for effective performance. A performance comparison of Impala with other databases is presented in Figure 31:

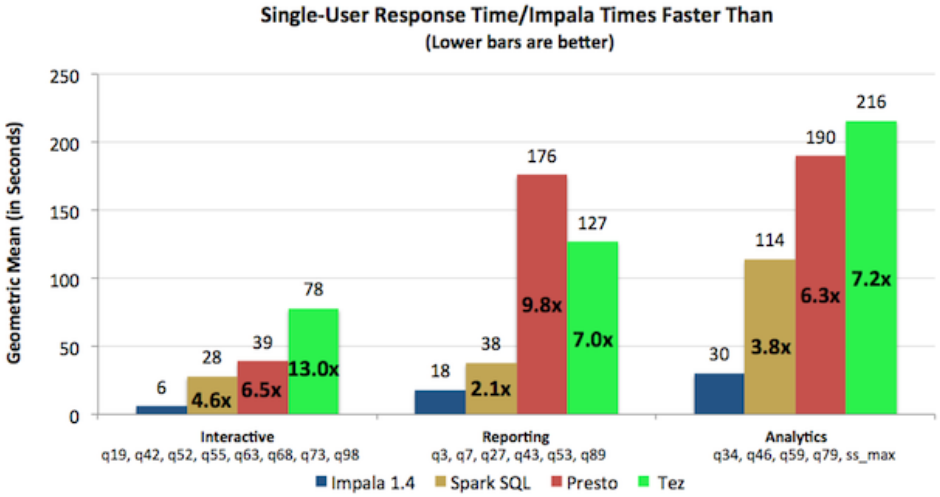


Figure 31 - Performance comparison of different databases

- **NoSQL**: these type of databases are generally used to store key:value pairs where the value can be objects/documents, etc. They provide raw storage & access via iteration or hash key. They do not provide the ability to do joins on columns or any other advanced features. Redis, a lightweight key:value database is a possible solution to be utilised in RAWFIE to store the results from the experiments.

	RDB (i.e. MySQL)	**Document Store (i.e. MongoDB)**	**Column Family Store (i.e. HBase)**
DB Schema	Relational Model, Hard for graph model	Complete schema-less	Semi schema-less
Performance	Too many join for graph model	High read performance; Potential write performance bottleneck	High write performance Fast key based read & Slow range query

Scalability	Difficult to scale-out (manual sharding)	Auto-sharding on pre-defined shard key	Horizontally scalable by tablet
Query	SQL	Limited query language (no join)	Key-value access; Pig & Hive based on MapReduce
Consistency	ACID Transactional	Eventual Consistency	No multi-row transaction
Concurrency Control	Locking or MVCC	node-level locking & atomic operation	row-based atomic
Security	AuthZ & AuthN	Basic security	Basic security
Notification Mechanism	Trigger	No build-in notification	No build-in notification

Table 42: Comparison between Relation and NoSQL databases [84]

5.2.9 Message Bus technologies and related communication protocols

This section describes the available software solutions that suit the requirements of RAWFIE in terms of asynchronous communication between the components, using a Publish/Subscribe or Publisher/Consumer communication model, as described in Section 3.2 - Message Bus component. As illustrated in Figure 32, this communication model is typically realised by means of a Message Broker, which connects different applications that can simultaneously act as Publishers and/or Subscribers (resp. Publishers and/or Consumers).

In this section, we provide an overview of the relevant solutions that will be considered and further analysed for possible adoption in RAWFIE.

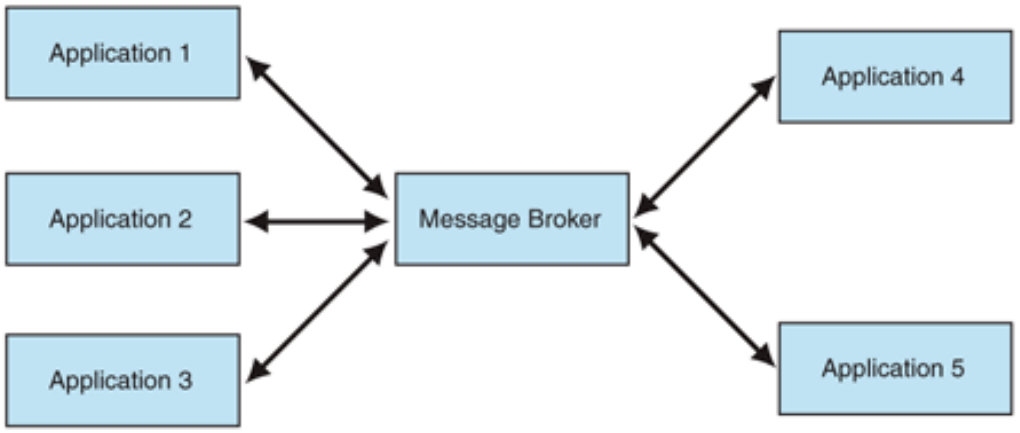


Figure 32 - Publish/Subscribe communication pattern through the use of a Message Broker

ActiveMQ

The Apache ActiveMQ [54] Message Broker provides an open source implementation of the Java Message Service (JMS) specifications. In acts as a reliable hub in any message oriented

enterprise application, and integrates perfectly with Java EE containers, ESBs, and other JMS providers.

It is designed for high performance clustering, client-server and peer-to-peer based communication. It uses a specific protocol, called Open Wire, to allow access to Active MQ brokers using different programming languages and protocols. For enabling cross-language/ platform communication of different clients with ActiveMQ, the STOMP protocol [55] is also supported. STOMP is a simple text orientated messaging protocol, which provides an interoperable wire format which enables messaging interoperability among many languages, platforms and brokers.
In addition, ActiveMQ provides support for different messaging protocols, transport options and interfaces, such as:

- AMQP protocol [56] - a platform-agnostic protocol, suitable for real-time data streams communication, and business transactions between applications, across distributed cloud computing environments. AMQP is an OASIS standard, thus avoids the need to use proprietary technologies and would be an interesting solution for interoperability and ease of integration and extension of the RAWFIE platform
- REST software API - ActiveMQ implements a RESTful API to messaging, allowing any web capable device, or web application, to publish or consume messages using a regular HTTP POST or GET
- TCP transport - through the TCP transport Apache ActiveMQ also provides clients with the possibility to connect to a remote ActiveMQ server by using a simple TCP socket interface
- MQTT protocol [57] – an OASIS standard, and a protocol specifically designed to allow connections and communication in an IoT environment. A more detailed description is included in a following subsection.

RabbitMQ

RabbitMQ [59] is another message broker implementation which supports several messaging protocols, directly and through the use of plugins. Several RabbitMQ servers on a local network can be clustered together, forming a single logical broker. Like Apache ActiveMQ, supported protocols include STOMP, AMQP and MQTT. Further, it provides:

- HTTP API to send and receive messages from a web browser (management plugin)
- STOMP messaging to the browser (Web-Stomp plugin)
- JSON-RPC lightweight remote procedure call protocol (synchronous protocol) to the browser (channel plugin)

Apache Kafka

Apache Kafka [60] is designed so that brokers can handle terabytes of messages with minimal or no-performances impacts. Clients (Publisher, Consumers) for Apache Kafka exists in JAVA and most other programming languages including a high performance C library. Kafka is often used for operational monitoring data: this involves aggregating statistics from distributed applications to produce centralized feeds of operational data.

Apache Kafka implements a tag-wise processing of data where data is consumed from topics of raw data and then aggregated, enriched, or otherwise transformed into new Kafka topics for further consumption. Apache Kafka multi-brokers architecture is presented in Figure 33.

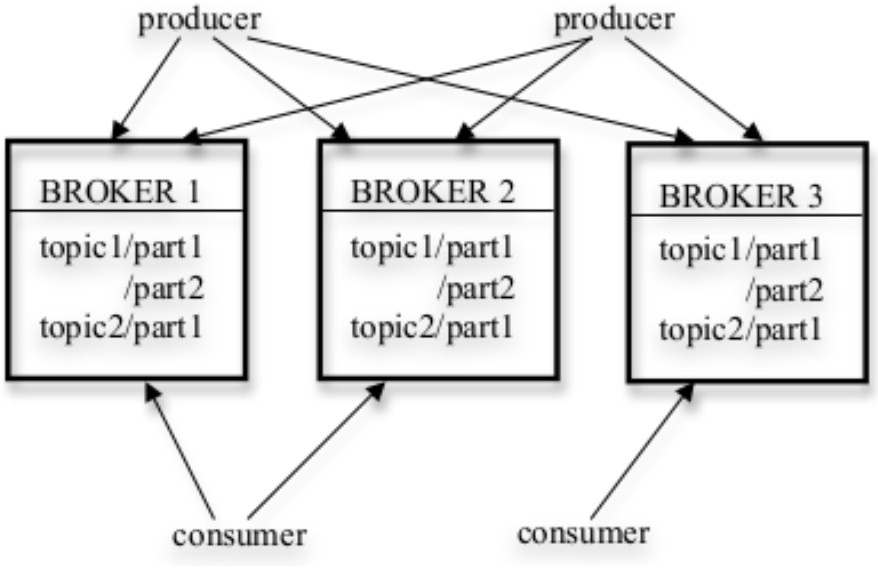


Figure 33 - Apache Kafka multi-broker architecture

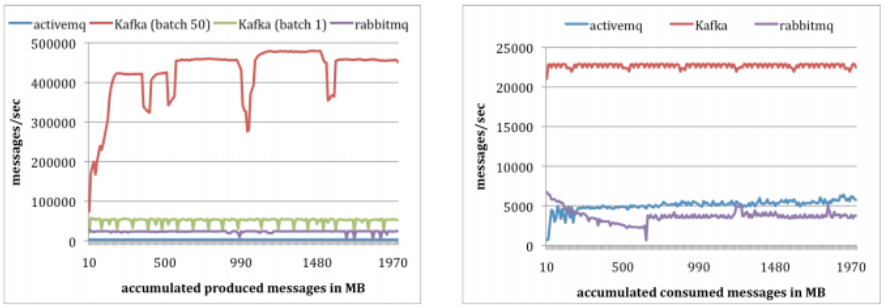


Figure 4. Producer Performance

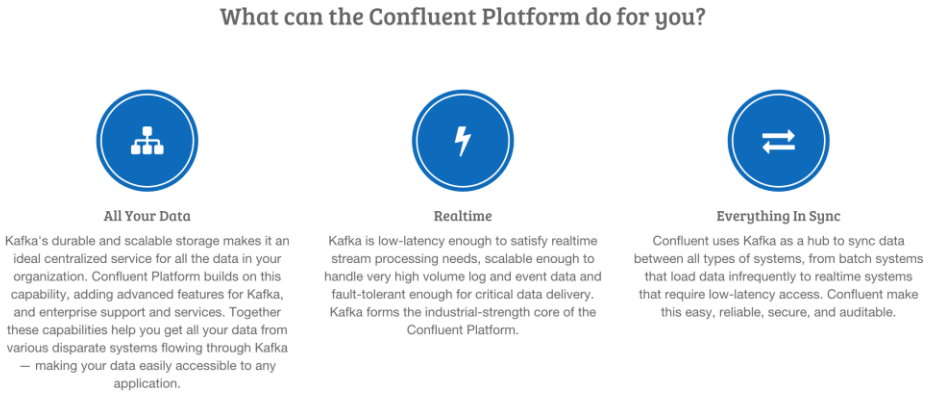
Figure 5. Consumer Performance

Figure 34 - Comparison of throughput for different message brokers [82]

Confluent

Confluent is an architecture built on top of Apache Kafka. It is developed by the same individuals who developed the original Kafka. Kafka is an industry de facto standard and has been demonstrated in wide use by companies such as LinkedIn, Yahoo, Spotify, Tumblr, Coursera, etc . This should be with the preferable choice for RAWFIE as there is a massive open

source community companies) that upstream changes all the time and extensive changes support



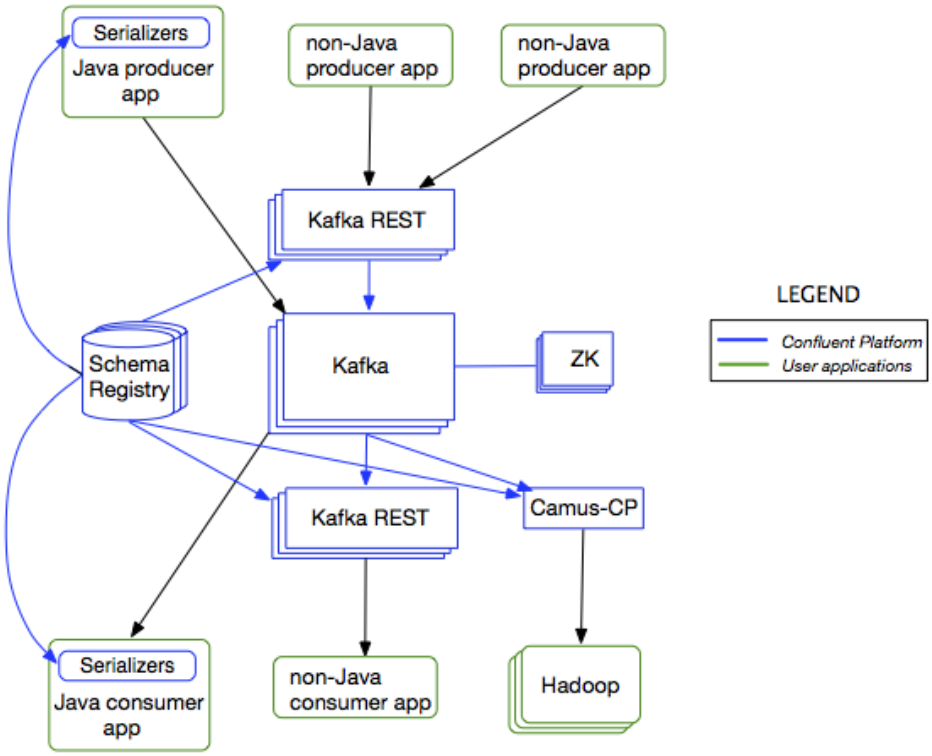


Figure 35 - Confluent architecture

- **Schema Registry**: houses all the schema definitions. A schema definition is a simple JSON blob that gives structure to raw data which is then compressed using AVRO. More information can be found in [73].
- **Kafka REST: co**nfluent provides a simple REST endpoint to push and pull data to specific stream revisions (which are tagged with a specific schema). They will however also provide the functionality to directly write into Kafka without an intermediary REST interface. More information can be found in [73]

- **ZK: Zookeeper**, another standard de facto synchronization mechanism. More information can be found in [74].
- **Camus:** Camus is a simple MapReduce job developed by LinkedIn to load data from Kafka into HDFS. This will be necessary to provide data to a relational (or nosql) database down the pipeline. Confluent provides this feature for free. More information can be found in [75].
- **Hadoop**: distributed file system. More information can be found in [76].

The benefits of using the confluent platform are evident above. It uses Kafka internally and offers the ability to tee off [forked] to HDFS for use in a database storage system using their Camus solution. It also provides a schema registry that enforces schema on write. There is no need for any custom binary messages, etc. The data is simply JSON blobbed, on the fly batch compressed using snappy or gzip and sent to Kafka.

MQTT Protocol and Message Brokers

MQTT (Message Queuing Telemetry Transport) [57] is a Client Server Publish/Subscribe messaging transport protocol. It is an OASIS standard [58], and can be used to provide near to real-time experiences and massive messaging capacity. It is supported by some of the Message Bus solutions described before, namely Apache ActiveMQ and RabbitMQ. In addition, MQTT Message Brokers implementation exist which only "talk" MQTT, which could be more suitable if only the use of MQTT is the focus.

MQTT is light weight, open, simple, and designed so as to be easy to implement. These characteristics make it ideal for use in many situations, including resource-constrained devices and environments, such as for communication in Machine to Machine (M2M) and Internet of Things (IoT) contexts, where a small code footprint is required and/or network bandwidth is limited. The protocol runs over TCP/IP, or over other network protocols that provide ordered, lossless, bi-directional connections.

Its features include:

- Use of the Publish/Subscribe message pattern which provides one-to-many message distribution and decoupling of applications
- A messaging transport that is agnostic to the content of the payload
- Three qualities of service for message delivery:
    - "At most once", where messages are delivered according to the best efforts of the operating environment. Message loss can occur. This level could be used, for example, with ambient sensor data where it does not matter if an individual reading is lost as the next one will be published soon after.
    - "At least once", where messages are assured to arrive but duplicates can occur.
    - "Exactly once", where message are assured to arrive exactly once. This level

could be used, for example, with billing systems where duplicate or lost messages could lead to incorrect charges being applied.

- A small transport overhead and protocol exchanges minimised to reduce network traffic.
- A mechanism to notify interested parties when an abnormal disconnection occurs.

The MQTT specification defines fourteen different types of Control Packets, one of which (the PUBLISH packet) is used to convey Application Messages. Messages are encoded and carried in binary format, using specific encoding schemes for each type of packet. Compression of the payload data has to be applied at application level, i.e. the appropriate payload flag fields to handle the compression details has to be defined in the application. Application-specific flags cannot be specified in the fixed or variable headers.

MQTT-SN (MQTT for sensor networks)

MQTT-SN [61] is a variant of MQTT designed and implemented specifically for Sensor Networks contexts. It is designed to be as close as possible to MQTT, but is adapted to the peculiarities of a wireless communication environment such as low bandwidth, high link failures, short message length. It is also optimized for the implementation on low-cost, battery-operated devices with limited processing and storage resources.

Compared to the original protocol, MQTT-SN defines a new offline keep-alive procedure for the support of sleeping clients. With this procedure, battery-operated devices can go to a sleeping state during which all messages destined to them are buffered at the MQTT Broker / Gateway and delivered later to them when they wake up (Figure 36). The MQTT broker / gateway needs to be aware of the sleeping state of these clients and will buffer messages destined to them, for later delivery when they wake up.

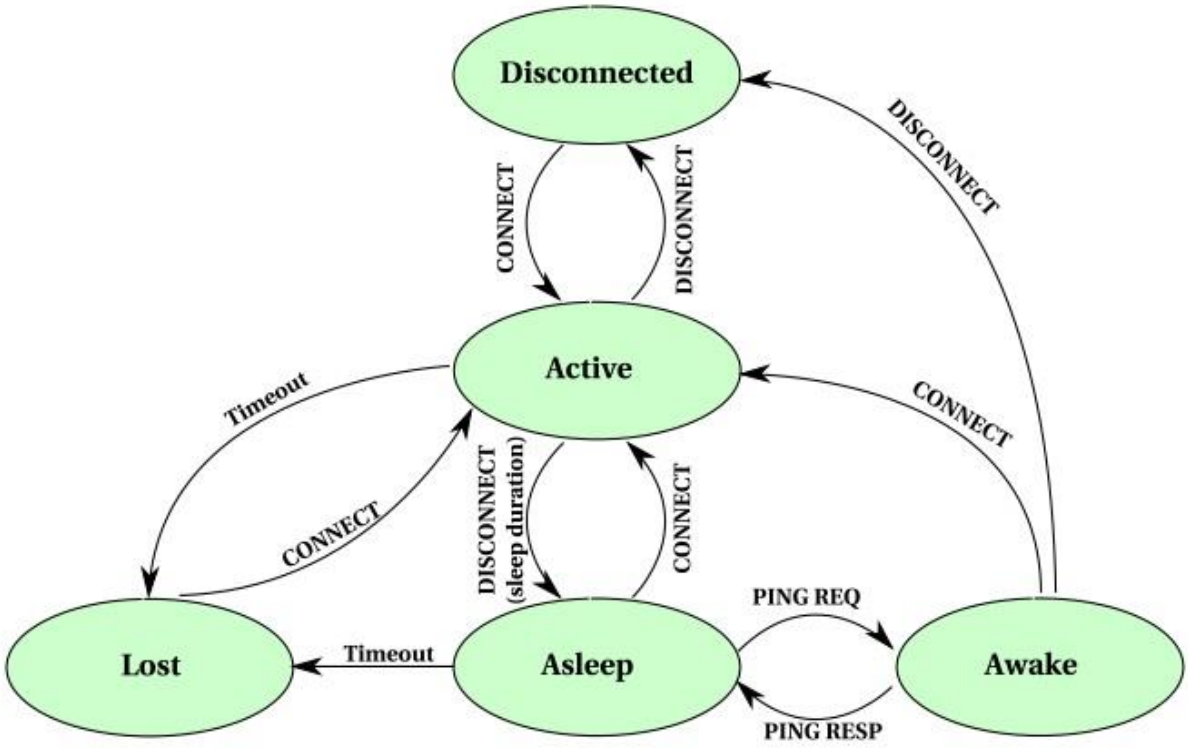


Figure 36 - MQTT-SN clients state transition diagram [61]

5.2.10 Resource discovery

In this section, we describe the relevant standards and protocols for resource and testbed discovery. The standards presented in this section cover both attempts to format a common language for the description of resources as well as protocols used to find these resources in constrained devices or over the internet.

GENI resource-request specification (RSpec)

Interoperability among different *Aggregate Managers* (AMs) is covered by the *Global Environment for Network Innovation* (GENI) through a common language for describing resources, resource requests, and reservations. GENI uses standardized *Request Specification* (RSpec) documents which are XML documents following agreed schemas to represent resources [52]. The schemas support aggregate or resource specific extensions. Ongoing work covers agreeing upon ontologies for other resource types.

The GENI infrastructures have been built for exploring future Internet at-scale. It supports at-scale experimentation on shared and heterogeneous GENI resources among multiple users, permits users deep programmability throughout the network, and offers collaborative and exploratory environments for innovative research and education.

In GENI, there are three different types of specifications for request, each used to describe resources when communicating with an AM. The communication with the AM is based on the common GENI AM API that requires AMs to communicate using RSpec data types.

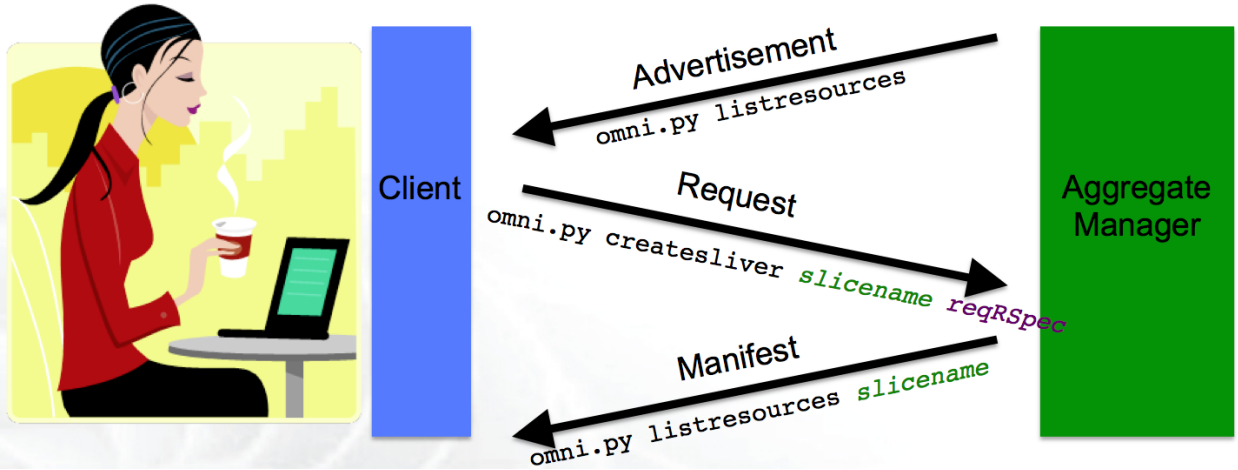


Figure 37 - Advertisement of available resources and request for resources reservation in GENI

- *Advertisement RSpec*, which describes the resources that the AM has. It is sent by the AM to the user.
- *Request RSpec*, which describes the resources that a user has reserved. It is sent by the user to the AM.
- *Manifest RSpec, which describes the resources that a user has reserved. It is returned by the AM to the user.*

Constrained Application Protocol (CoAP) Resource Discovery

The Constrained Application Protocol (CoAP) [42] is a specialized web transfer protocol for use in constrained nodes (often low-end microcontrollers with small amounts of memory) and constrained networks (low-power, lossy). The protocol is designed for machine-to-machine (M2M) applications such as smart energy and building automation.

The discovery of resources offered by a CoAP endpoint is extremely important in M2M applications where there are no humans in the loop and static interfaces result in fragility. The main function of the discovery mechanism in CoAP is to provide Universal Resource Identifiers (URI's, called links) for the resources hosted by the server, complemented by attributes about those resources and possible further link relations. Based on the HTTP Link Header field [45] CoAP specifies the Constrained RESTful Environments (CoRE) Link Format [46] which is carried as a payload in a CoAP response and is assigned an Internet media type. A well-known relative URI "/.well-known/core" is defined as a default entry point for requesting the list of

links about resources hosted by a server and thus performing CoRE Resource Discovery. An illustrative example of calling this URI follows.

CoAP Client:

```
REQ: GET /.well-known/core
```

CoAP Server:

```
RES: 2.05 Content

</sensors/temp>;rt="temperature-c";if="sensor",</sensor/light>;rt="light-
lux";if="sensor"
```

In the above example a request for the supported resources by a CoAP server resulted in a response which includes the links of the two different sensors supported by this endpoint. CoAP utilizes port number 5683 over UDP for resource discovery.

Service Location Protocol (SLP)

The Secure Location Protocol (SLP) is one of the more used service discovery protocols. It consists of three basic entities [53]:

- Service Agent (SA)
- Directory Agent (DA)
- User Agent (UA)

It is a scalable, lightweight, simple, decentralized protocol and also independent by HW, SW and the language programs. A SLP service has some "properties" that describe it. The first is the Service URL: it is a string with a specific form, and it specifies the general category of the service that it describes. Each service, beyond a Service URL, has a list of Attribute-Values couples. Each attribute is a property of the service, and it is indicated by a name. This property, typically, has one or more values: so a couple can be "Supported_Resolutions = 640X480,800X600,1024X768". This attribute indicates that that service (that can be a monitor or a projector) has an attribute, named "Supported_Resolutions" (that is auto-explicative) that can assume 3 possibly values: 640X480, 800X600 and 1024X768. In addition, it has capabilities of performing "complex" queries on attributes' values where Boolean operators (AND, OR, NOT), comparators (<, >, =) and functions of string matching could be adopted.

Figure 38 - Service Location protocol components

The SA is the service agent: it has to register its services on the DA. The DA is the "core" of the architecture, because it registers all the services that are offered by a network. Finally, the UA is the client that interrogates the DA to find a specific service of a SA.

Two type of messages exchange are considered in SLP: one is for a registration of a new service and the other for service requests.

New Service Registration

1. the SA sends a service registration request (*SrvReg*) to the DA when it want share an own service;
2. the DA registers the service and replay to the SA with a service acknowledgement message *(SrvAck)*;

Service Request

1. the UA sends a service request (*SrvRqst*) asking the type and the parameters of the desired service;

the DA checks for the UA request and answer with a service replay message (*SrvRply*) including the address of the services required.

5.3 UxV technologies

In this section the two UxV manufacturer partners of the RAWFIE project describe how their UxVs may be connected to the RAWFIE platform.

5.3.1 ROS platform control architecture

Robotnik's mobile platforms are based on ROS (Robot Operating System).

ROS is an open-source, meta-operating system that provides inter-process message passing services (IPC) in a network. It is usually running over a Linux Ubuntu 12.04 SO installed in the robot's computer.

ROS is also an integrated framework for robots that provides:

- Hardware abstraction layer
- Low level device control
- Robot common functionality (simulation, vision, kinematics, navigation, etc.)
- IPC
- Package and stack management

The ROS runtime "graph" is a peer-to-peer network of processes (potentially distributed across machines) that are loosely coupled using the ROS communication infrastructure.

It also implements several different styles of communication, including synchronous RPC-style communication over services, asynchronous streaming of data over topics, and storage of data on a Parameter Server.

These topics are the main tool used by the system to command the movement of the robot and stream sensor data to the user.

As an example of how Robotnik's robots are working a more detailed description of the summit_xl_robot control stack follows.

The summit_xl_robot real robot control stack is composed of the following packages:

- summit_xl_complete
    - This package launches the complete robot. It is called from the .bashrc at system startup. The summit_xl_complete.launch file can be configured to start the available devices, usually: hokuyo, axis_ptz, sphere_camera, summit_xl_controller, robotnik_gyro, summit_xl_pad, etc. Integrates several test launch files to test kinect sensor, hokuyo laser sensor or imu.
- summit_xl_controller
    - Low level robot control of the 4 servos. This package contains the robot control functions to operate the skid-steering structure (velocity control of the axes and position control of the robot) and to get accurate odometry estimations from the robot sensors. This node subscribes to cmd_vel messages.
- summit_xl_navigation
    - This package uses the ros navigation stack (move_base node) to allow sending goals to the robot in Cartesian coordinates. It allows also gmapping configuration.
- summit_xl_pad

    - Allows to use a joystick to operate the summit_xl_controller, sending the messages received through the joystick input, correctly adapted to the topic. The speed level is also commanded. The node allows to load different types of joysticks (PS3, Logitech, Thrustmaster). New models can be easily added by creating new .yaml files. If modbus_io node is available, the digital outputs (ligths, axes, etc.) can also be controlled with the pad. If a sphere_camera is available, the pan-tilt can also be commanded with the pad.
- robotnik_msgs
    - Simple package that contains standard services and messages commonly used in mobile robots.
- summit_xl_web
    - ROS Web Tools based package to access the robot via web browser. It allows to move the PTZ camera and the robot and is intended as a template to be configured by the user.
- gps_map_tf
    - This package is intended to publish the transform from map->odom in order to use gps coordinates to localize the robot and in order to use the move_base stack in Cartesian coordinates (it does the equivalent tf publication). It allows to set the coordinates origin and to align the robot heading with the gps coordinate system heading. This package can be used together with summit_xl_waypoints to set a sequence of GPS cartesian coordinates to be sent to the robot.

To address the issue of the communication between ROS environment and RAWFIE UGV node, the system will have to make use of some tools that ROS provides in order to stream data and read/write the topics needed to command the robots.

There is a particular ROS Stack called rosbridge_suite [68] that provides a JSON API to ROS functionality for non-ROS programs. There are a variety of front ends that interface with rosbridge, including a WebSocket server for web browsers to interact with.

The conclusion is that we can implement a Rosbridge server in the robot system, while UGV node should make use of Rosbridge API.

This Rosbridge library is a Python library responsible for taking JSON strings and converting them to ROS messages, and vice versa. Rosbridge library is meant to be used as a library for transport layer packages. For example, the rosbridge_server package creates a WebSocket connection and uses the rosbridge library to handle the JSON to ROS conversion.

Any Python package or program can use rosbridge library for direct JSON to ROS communication. For example, a TCP server, a serial bridge, etc.

A step further would be to share a list of these needed topics and exact functionalities that are being controlled by RAWFIE Testbed Tier Components.

FurtherMore information about the Summit_XL Simulation stack is available can be found at [77] and about the for Kobuki stack (turtlebot mobile platform) is available at [78].

5.3.2 USV platform

The autonomous vehicles developed by MST for the RAWFIE project use the LSTS (underwater Systems and Technology Laboratory) toolchain, developed in cooperation with the University of Porto. This toolchain comprises four components which are described in the following paragraphs.

GLUED (GNU/Linux Uniform Environment Distribution)

GLUED is a minimal Linux distribution targeted at embedded systems and based on cross-compiled binaries. The target's runtime environment and compilers are generated on a host machine with different computer architecture. The main computer of MST vehicles will use GLUED for reliability and performance reasons. The most important features of GLUED are the following:

- Small footprint (around 10 MiB);
- Fast boot time (2 to 5 seconds depending on target machine and peripherals);
- Reproducible root filesystem;
- Fast and controlled full system upgrades (10 to 30 seconds);
- Support for several x86, ARM, and MIPS targets.

IMC (Inter-module Communication) Protocol

IMC is a message-oriented protocol for autonomous vehicles and sensor networks. The IMC protocol defines and documents a set of messages, encoded in a single XML file that is then translated to source code of several programming languages. The IMC protocol comprises different logical message groups for networked vehicle and sensor operations. It defines an infrastructure that is modular and provides different layers of control and sensing. The set of messages defined in the IMC protocol is sufficient to monitor and control AUVs, ASVs, ROVs, UAVs, and static sensors. The message flow corresponds to the several control and sensing layers within IMC:

- ***Mission control*** messages define the specification of a mission and its life-cycle, for the interface between a CCU (Command and Control Unit) such as a Neptus console and a mission supervisor module. A mission is a sequence or graph of maneuvers.
- ***Vehicle control*** messages are used to interface the vehicle from an external source, typically a CCU or a mission supervisor module, for example to issue maneuver commands or other external requests, and to monitor the vehicle's state.
- ***Maneuver*** messages are used to define maneuvers, associated commands and execution state. The simplest maneuver types are related to waypoint tracking and loitering patterns.

- ***Guidance*** messages are related to the guidance used for autonomous maneuvering. Usually a guidance step generates new reference measures for the vehicle's heading, depth, and velocity, in the form of a Desired Guidance message.
- ***Navigation*** messages define the interface for reporting a vehicle's navigation state. The Estimated State message defines a vehicle's navigational state by the SNAME convention.
- ***Sensing*** messages are used to report sensor readings by the respective hardware controllers.
- ***Actuation*** messages specify the interface with hardware actuator controllers, such as fins and thrusters.

DUNE

DUNE provides an operating-system and architecture independent C++ programming environment for writing efficient and modular real-time reactive tasks. Besides running on-board in several different types of autonomous vehicles (e.g., AUVs, UAVs, and ASVs), DUNE is also the controlling software of communication gateways.

DUNE uses the publish/subscribe pattern to provide loose coupling between modules, which in DUNE are called tasks. DUNE tasks publish and subscribe messages without knowing any details about the other tasks. Effectively, message passing is the only mechanism available to exchange information between tasks. For example, a task that interacts with a sensor produces a message of a given type with a sensor reading (e.g., acceleration), that can later be consumed by another task which integrates that information and produces a state estimate (e.g., estimated position), another task can consume that estimate and produce commands to actuators (e.g., increase thrust). Figure 39 illustrates this concept.

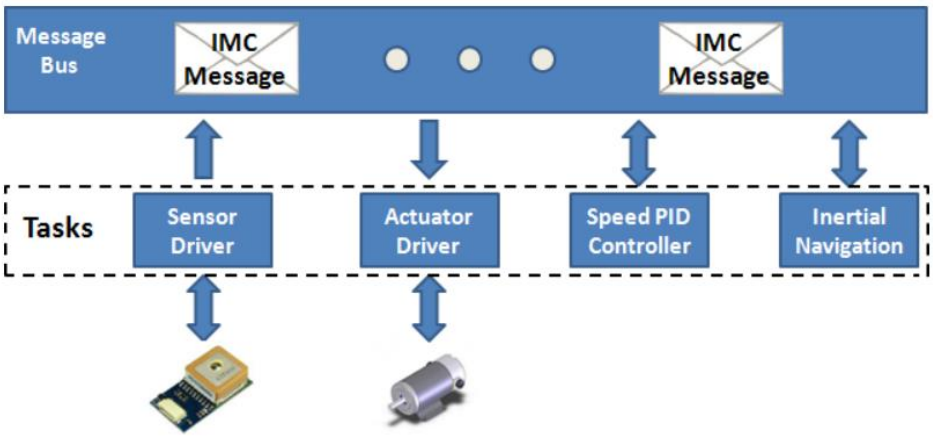


Figure 39 - Dune Tasks for communication between modules

This modularity and loose coupling between tasks facilitates incorporating new sensors and functionalities without developing new code. In most cases a simple configuration change is enough to support new functionality. This fact eases not only the life of the everyday developer, but also of new, or temporary developers that will only be working with some specific module of the software, being shielded from the complexity of the remaining tasks of the framework.

Communication between tasks is made exclusively using IMC messages. A task in DUNE may be used by one or more vehicles, for example, the same navigation task is used in all MST's vehicles. That is possible due to the ability to define a set of parameters in a configuration file, without the need to recompile code. DUNE has one configuration file for each vehicle it supports. This same file is used in real-vehicles and in simulated/emulated instances. DUNE uses the concept of profiles to selectively enable or disable tasks from running. For example when DUNE is started with the pure simulation profile, all tasks that interact with real sensors and actuators are replaced by analogous simulation tasks. Hardware-in-the-loop simulation can be achieved using the same method.

Neptus

Neptus is a distributed command, control, analysis, and intelligence framework for operations with autonomous vehicles and human operators. It supports all phases of the life-cycle of autonomous vehicle's missions:

- World representation
- Planning
- Execution
- Monitoring
- Post-mission review and analysis

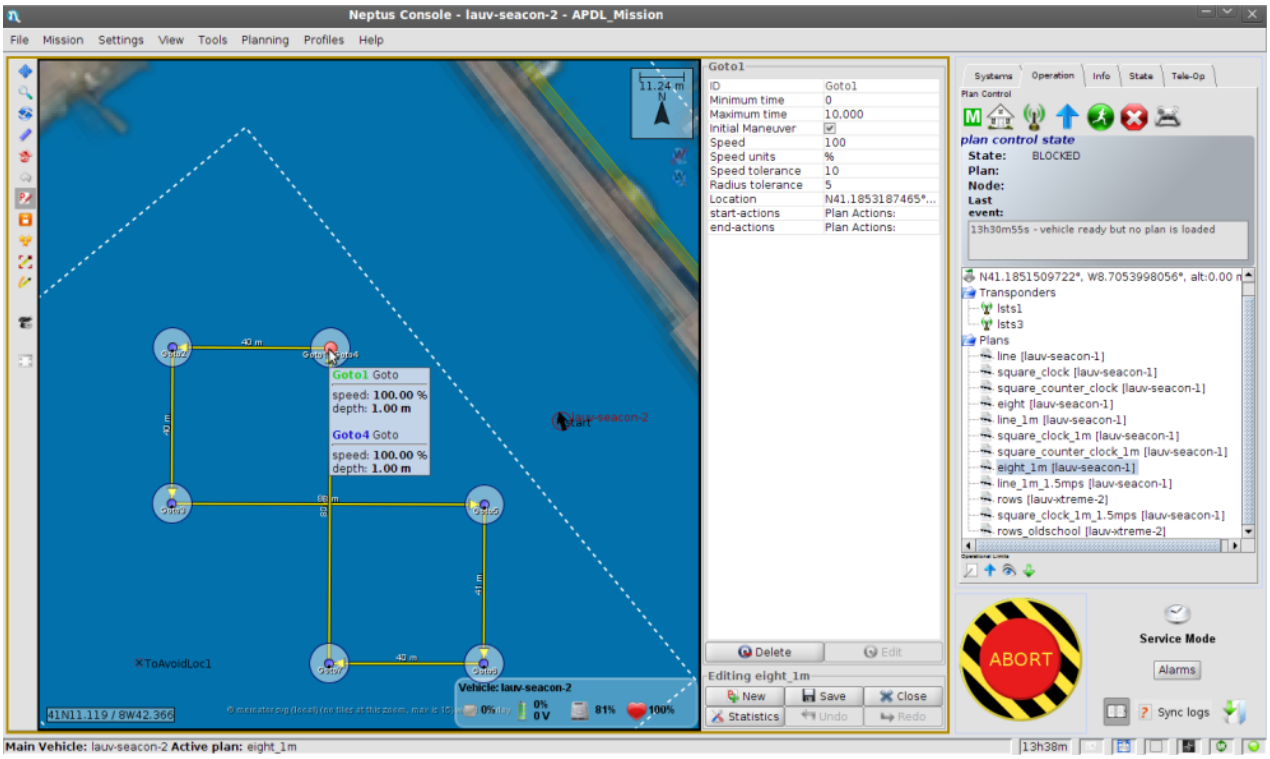


Figure 40: Neptus mission planner interface

Neptus supports controlling and monitoring multiple vehicles in a single computer screen, and allows the operator to design graphically any kind of mission supported by IMC enabled autonomous vehicles. An illustration of Neptus' main planning interface is shown in Figure 40. One important tool provided by Neptus is the Mission Review and Analysis (MRA) program. This tool allows users and developers to process, analyze, and transform data files collected by vehicles during missions, using a sophisticated graphical user interface. With this program the user can replay missions, visualize collected data in predefined plots, tables, and 3D maps, and create dynamic plots of sensor measurements.

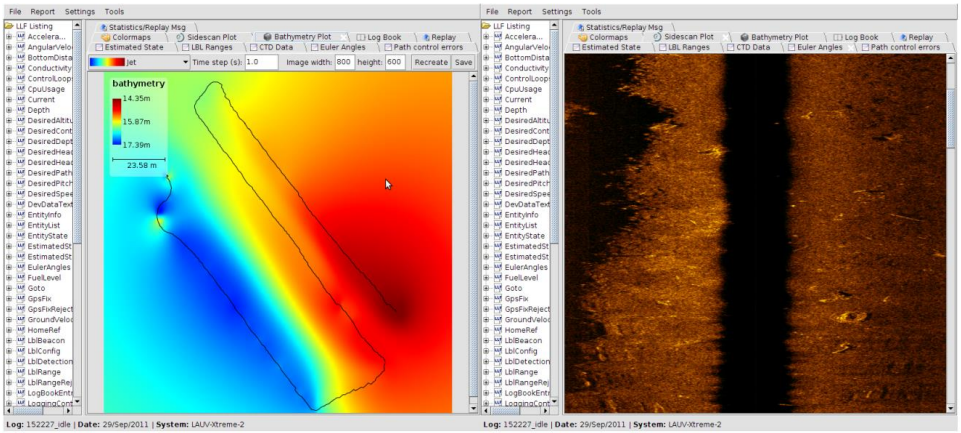


Figure 41 – MRA visualizations

In the RAWFIE project MST will draw from its experience in integrating the LSTS toolchain with third-party protocols and will develop a new software module to translate IMC messages to/from the RAWFIE protocols and a DUNE task that implements the required flow control logic.

References

[1] http://www.fed4fire.eu/
[2] http://groups.geni.net/geni/wiki/GeniDesign
[3] http://relyonit.eu/
[4] IoT Lab-Project deliverable „D3.1 Open Interfaces"
[5] http://www.iotlab.eu/
[6] http://www.wisebed.eu/
[7] http://www.fed4fire.eu/omf6/
[8] http://groups.geni.net/geni/wiki/GEC18Agenda/LabWikiAndOEDL/Introduction
[9] http://nam.ece.upatras.gr/fstoolkit/trac/wiki/TheOfficeModel
[10] http://www.ict-openlab.eu/technologies/control-plane.html
[11] http://ipac.di.uoa.gr/
[12] http://eclipse.org/Xtext/
[13] http://polos.di.uoa.gr/
[14] http://www.project-makesense.eu/
[15] http://www.contiki-os.org/
[16] https://wiki.oasis-open.org/security/FrontPage
[17] http://openid.net/
[18] http://oauth.net/
[19] http://openid.net/connect/
[20] http://web.mit.edu/kerberos/
[21] X.509 version 3: http://tools.ietf.org/html/rfc5280http://tools.ietf.org/html/rfc5280
[22] AIR COMBAT COMMAND CONCEPT OF OPERATIONS FOR ENDURANCE UNMANNED AERIAL VEHICLES 3 Dec 1996 - Version 2, SECTION 6 - COMMUNICATION INTEGRATION AND INTEROPERABILITY http://fas.org/irp/doddir/usaf/conops_uav/part06.htm
[23] UAV Communications & Data Links Sample, UAV Executive Certificate Course, UnmannedUniversity.
[24] http://www.smartfile.com/blog/the-differences-between-iaas-saas-and-paas/
[25] https://omf.mytestbed.net/projects/omf6/wiki/OEDLOMF6
[26] https://mytestbed.net/projects/oml
[27] Wiselib - https://github.com/ibr-alg/wiselib/wiki
[28] Shawn WSN - simulator https://github.com/itm/shawn/wiki
[29] Testbed Runtime (TR) - https://github.com/itm/testbed-runtime/wiki
[30] WISEBED SOAP API - http://wisebed.eu/#docs_soap
[31] WISEBED REST API - http://wisebed.eu/#docs_rest
[32] WISEBED JavaScript Client library - https://github.com/wisebed/wisebed.js
[33] WISEBED JavaScript experimentation scripts and CLI - https://github.com/wisebed/wisebed.js-scripts
[34] GraphML format - http://graphml.graphdrawing.org/

[35] WISEBED Virtual Machine - http://wisebed.eu/#appdev_vm
[36] GENI Aggregate Manager http://groups.geni.net/geni/wiki/GAPI_AM_API_V3http://groups.geni.net/geni/wiki/GAPI_AM_API_V3
[37] OMF Framework: http://mytestbed.net/projects/omf/
[38] OML Framework: https://oml.mytestbed.net/projects/oml/wiki/
[39] Zabbix: http://www.zabbix.com/
[40] Nagios: http://www.nagios.org/
[41] TopHat: http://trac.top-hat.info/wiki/manifold
[42] https://tools.ietf.org/html/rfc7252
[43] http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html
[44] http://www.ipso-alliance.org/wp-content/media/draft-ipso-app-framework-04.pdf
[45] https://tools.ietf.org/html/rfc5988
[46] https://tools.ietf.org/html/rfc6690
[47] Nguyen, P.-H., Jung, Y.-C., Control of autonomous underwater vehicles using adaptive neural network: Decoupled control of heading, depth, and velocity, ATC 2009 - Proceedings of the 2009 International Conference on Advanced Technologies for Communications , art. no. 5349385, pp. 133-136, 2009
[48] Peng, P.-F., Liu, Z., An independent ups and downs control method of underwater submersible vehicle based on adaptive fuzzy control, International Conference on Intelligent Human-Machine Systems and Cybernetics, IHMSC 2009 1, art. no. 5336166, pp. 292-295, 2009.
[49] www.sfly.org
[50] www.noptilus-fp7.eu
[51] A. Ch. Kapoutsis, G. Salavasidis, S. A. Chatzichristofis, J. Braga, J. Pinto, J. B. Sousa, Elias B. Kosmatopoulos, “THE NOPTILUS PROJECT OVERVIEW: A FULLY-AUTONOMOUS NAVIGATION SYSTEM OF TEAMS OF AUVS FOR STATIC/DYNAMIC UNDERWATER MAP CONSTRUCTION”, «IFAC Workshop on Navigation, Guidance and Control of Underwater Vehicles (NGCUV’2015)», April 28-30 2015, Girona – Catalonia (Spain), 2015.
[52] http://groups.geni.net/geni/wiki/GENIExperimenter/RSpecs
[53] http://en.wikipedia.org/wiki/Service_Location_Protocol\
[54] Apache ActiveMQ Message Broker - http://activemq.apache.org/
[55] STOMP (Simple Text Oriented Message Protocol) - http://stomp.github.io/
[56] AMQP (Advanced Message Queuing Protocol) - https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=amqp
[57] MQTT (Message Queuing Telemetry Transport) - http://mqtt.org/
[58] MQTT standard by OASIS - https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=mqtt
[59] RabbitMQ - https://www.rabbitmq.com/

[60] Apache Kafka - http://kafka.apache.org/
[61] MQTT-SN - http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf
[62] Raj Jain, Fred L. Templin, Yin, Kwong-Sang, Wireless Datalink for Unmanned Aircraft Systems: Requirements, Challenges and Design Ideas, 2011, DOI:10.2514/6.2011-1426
[63] Johansen, Tor Arne; Zolich, A.; Hansen, T.; Sørensen, Asgeir J. Unmanned Aerial Vehicle as Communication Relay for Autonomous Underwater Vehicle – Field Tests, IEEE Globecom Workshop – Wireless Networking and Control for Unmanned Autonomous Vehicles, Austin, TX, 2014, NTNU
[64] Tony Stentz, Alonzo Kelly, Herman Herman, Peter Rander, Omead Amidi, and Robert Mandelbaum, Integrated Air/Ground Vehicle System for Semi-Autonomous Off-Road Navigation, AUVSI Symposium, July 10, 2002
[65] Jennifer Carlson and Robin R. Murphy, How UGVs Physically Fail in the Field, IEEE TRANSACTIONS ON ROBOTICS, VOL. 21, NO. 3, pp 423-437, JUNE 2005
[66] Narek Pezeshkian, Hoa G. Nguyen, and Aaron Burmeister, UNMANNED GROUND VEHICLE NON-LINE-OF-SIGHT OPERATIONS USING RELAYING RADIOS, Proceedings of the 12th IASTED International Conference Robotics & Applications, August 14-16, 2006 Honolulu, Hawaii
[67] http://fp7-sunrise.eu/
[68] http://wiki.ros.org/rosbridge_suite
[69] http://spark.apache.org/
[70] www.mlbase.org
[71] http://aws.amazon.com/rds/
[72] http://aws.amazon.com/ec2/
[73] http://confluent.io/docs/current/avro.html
[74] https://zookeeper.apache.org/doc/trunk/zookeeperOver.html
[75] http://confluent.io/docs/current/camus/docs/intro.html#key-features
[76] http://hadoop.apache.org/
[77] https://github.com/RobotnikAutomation/summit_xl_sim
[78] https://github.com/yujinrobot/kobuki
[79] Reference Model for Service Oriented Architecture 1.0,Committee Specification 1, 2 August 2006 – URL http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm
[80] WSDL version 2.0 - http://www.w3.org/TR/wsdl20/
[81] SOAP version 1.2 - http://www.w3.org/TR/soap/
[82] Jay Kreps, Kafka:Neha Narkhede, Jun Rao: A Distributed Messaging System for Log Processing, LinkedIn & Microsoft Research
[83] http://confuent.io
[84] eBay Cloud CMS based NoSQL Presentation from China SoftCon 2011