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	D4.4 - High Level Design and Specification of RAWFIE Architecture (2nd version)		
Confidentiality	PU	Deliverable type[1]	R
Deliverable File	D4.4	Date	08.05.2016
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
Vasil Kumanov	Epsilon Bulgaria	vasil.kumanov@epsilon-bulgaria.com
Kiriakos Georgouleas	HAI	GEORGOULEAS.Kiriakos@haicorp.com
Nikolaos Priggouris	HAI	PRIGGOURIS.Nikolaos@haicorp.com
Lionel Blondé	HES-SO	Lionel.blonde@hesge.ch
Giovanni Tusa	IES Solutions	g.tusa@iessolutions.eu
Kostas Kolomvatsos	UoA	kostasks@di.uoa.gr
Miltiadis Kyriakakos	UoA	miltos@di.uoa.gr
Ricardo Martins	MST	Rasm@oceanscan-mst.com
Philippe Dallemagne	CSEM	Philippe.Dallemagne@csem.ch
Elias Kosmatopoulos	CERTH	kosmatop@iti.gr

REVIEWERS TABLE

Name	Company	E-Mail
Philippe Dallemagne	CSEM	Philippe.Dallemagne@csem.ch
Sarantis Paskalis	UoA	paskalis@di.uoa.gr
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	2015-03-14	Start editing 2nd version of the architecture	all
0.2	2015-03-15	Added "Overview of Changes"	Overview of Changes
0.3	2015-03-31	Handled reviewer comments for D4.1	all
0.4	2015-04-06	Update Architectural Overview	Architectural Overview
0.5	2015-04-08	Excessively updated all texts took from D4.1	all
0.6	2015-04-15	Added contributions from partners	mainly Components Overview
0.7	2015-04-21	Added contributions and handle comments from partners	all
0.8	2015-04-25	Added contributions and handle comments from partners	all
0.9	2015-04-27	Added contributions and handle comments from partners	all
0.9	2015-04-29	Added contributions and handle comments from partners	all
0.10	2015-05-05	Added contributions and handle comments from partners – ready for first review	all
0.11	2015-05-04	First review	all
0.12	2015-05-06	Second and third review	all
1.0	2015-05-08	Final version	all

[3] Deliverable Distribution: PU (Public, can be distributed to everyone), CO (Confidential, for use by consortium members only), RE (Restricted, available to a group specified by the Project Advisory Board).

Abstract:
This deliverable describes the second version of the RAWFIE high-level architecture. An overview of all components and their interaction is given.

Several changes were made on the architecture to reflect the latest developments and also improvements as a result of several internal discussions..

Keywords:
architecture, components, interactions

Part II: Table of Contents

List of Figures

List of Tables

Part III: Executive Summary

This deliverable describes the planned high-level architecture of RAWFIE. It is the second version of this deliverable

Initially, a general overview of the architecture is given, describing the general component integration, the four abstraction tiers (front-end, middle, data, testbeds), the SFA compatibility and the used MOM.

Then, each component is described and its relation to other components are listed.

Finally a requirement mapping for all general requirements of D3.2 is done.

The state of the art analysis of D4.1 is still valid and not repeated in the deliverable. However, some additional information on certain technologies (that complement the state of the art) are describe in an annex.

Part IV: Main Section

1 Introduction

1.1 Scope and overview of D4.4

D4.4 updates the first version of the architecture from D4.1. It reflects all the necessary changes that have been done or need to be done based on the experience of the first implementation period.

The sections “Architectures Overview” and “Component Overview” of D4.1 are replaced with updated information in the present deliverable. The section “State of the art” of D4.1 is still valid and will not be repeated, but is extended with the introduction of additional relevant technologies (see Annex A). Section “Potential use cases and sequence diagrams” is not provided anymore, since updated sequence diagrams, taking into account use cases and workflows deriving from the new version of the requirements (see D3.2), will be included in D4.5.

1.2 Relation to other deliverables

D4.4 is an update of D4.1 so it will share many of the contents with it.

A detailed updated requirement analysis was given in D3.2. Using these updated requirements (which also reflect the experiences of the first implementation period) the architectural definition was updated.

D4.5 will provide updated detailed components descriptions. Therefore, this deliverable aims at describing the components and their interfaces at a high level.

D4.6 will provide information on verification and validation plans and scenarios for the architecture.

D4.7 will then finalize the architecture of RAWFIE (3rd version).

2 Overview of changes

This chapter shortly summarises the most important changes made in comparison to D4.1:

- “State Of The Art” is removed as the State Of The Art of D4.1 is still valid.
    - Newly introduced relevant technologies are described in Annex A in this deliverable.
- Sequence diagrams has been moved and updated only into D4.5.
- Architectural changes are reflected in the following parts:
    - Chapter “Architectural Overview” was completely revised.

- The architecture diagram provides further details and was adapted to reflect structural changes.
- The SFA interface was added
- Testbed now have to implement a *Common Testbed Interface*.
- After long internal discussion the Testbed Proxy was removed as part of the RAWFIE architecture (testbeds may still have some internal proxies to restrict internet access). The securing of communication will be done using the Kafka message bus. It supports encryption and authentication using SSL (client and server certificates) and authorizations can be defined via ACLs [16].
- Components added/changed:
  - A wiki (Wiki Tool) was added, to fulfil the need for documentation and manuals
  - The Accounting Service was added. Its main purpose is, based on usage statistics gathered for each RAWFIE service, to allow the introduction of appropriate charging schemas (especially at the later stages of the project) for the use of the platform, its services and the UxV resources.
  - Several repositories (Testbeds & Resources Repository, Experiments & EDL Repository, Bookings Repository, Status Repository) were fused into the Master Data Repository. These data sets will only be small to medium sized and have relational dependencies.
  - The Measurements repository is now specialised for the big data sets that will very likely be managed.
  - The Results repository uses a database specialized for data analysis.

- Components' descriptions have been updated. Changes are directly noted in the last line of the respective component table
- The requirement mapping was reduced to include only the “general” requirements from D3.2. Component specific requirements will be handled in D4.5.
  - Additionally, for each mentioned requirement it is described, how it is addressed.
- Aberrations where moved to the annex and a glossary was added.

3 Architectural Overview

This chapter gives an overview of the architecture and the various components in each tier. Figure 1 shows an overview of the architecture: it provides updates and enhancements to the one presented in the D4.1, being the result of the continuous activities of the consortium for the refinement of the functional requirements. It also takes into consideration the outcomes of the first prototype implementation (see WP5). The main design principles are described in the following sub-sections.

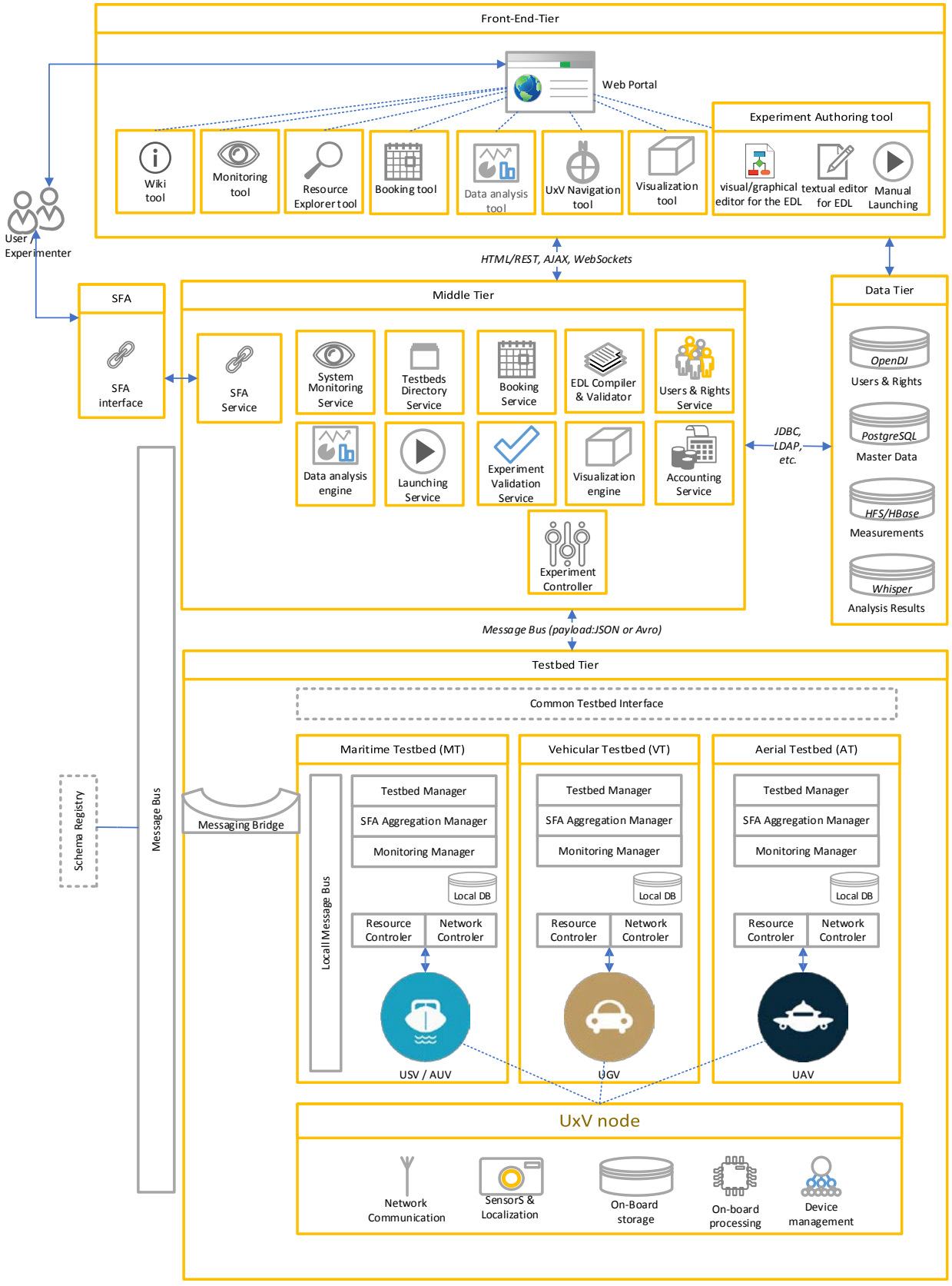

Figure 1 – RAWFIE architecture

3.1 Components integration

RAWFIE follows the Service Oriented Architecture [1] paradigm: all components provide clearly defined interfaces, so that they can be easily accessed by other components, and their business logic can be easily updated, with new functionalities without affecting the interfacing with other components. Interacting with them is made possible by the use of remote service control protocols such as Representational State Transfer (REST) resource invocation style or the or the Avro RPC [3], 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-oriented middleware (via a Message Bus) is used where suitable, using a convenient communication model providing distribution, replication, reliability, availability, redundancy, backup, consistency, and services across distributed heterogeneous systems. The Message Bus communication system interconnects components in the same tier, as well as components located in different tiers (e.g. Middle Tier and Testbed Tier). It can be used for asynchronous notifications and asynchronous method calls / response handling. As such, it is used for transmitting measurements that are routed from producers (e.g. UxVs) to the consumers pertaining to the Middle Tier (e.g. Experiment Monitoring, Visualisation) as well as for information that generally addresses multiple components. See also section 3.8 for more information of the communication through the message bus.

Chapter 4 will give more information about the components highlighted in Figure 1. A more detailed up to date description of interfaces and interactions between the various components will follow in D4.5.

3.2 Real-time constraints and impacts in the architecture

3.2.1 Rationale

Real time constraints for the communication between Middle Tier and Testbed components, and between Testbed components, may depend on the type of experiment as well as on the type of devices involved. Navigation of UxVs may require low latency, in order to ensure proper control and, as a consequence, safety of the devices themselves and their environment, including people.

3.2.2 Approach

Dealing with highly dynamic vehicles, such as aerial vehicles or drones (UAVs), implies providing fast response time to meet their timing requirements. Dealing with less dynamic vehicles such as UGV or maritime vehicles implies providing more relaxed response times.

However, in all cases, there are operational boundaries in terms of time, be they short or long, that must be dealt with by the operational entities and stakeholders (the drone itself, the experimenter, the resource manager and possibly many others). The consortium will identify and

elaborate on time constraints and real time requirements for several types of devices that may be involved in the RAWFIE experiments, with the help of owners/providers of UxVs, first those pertaining to the consortium, then those with new devices that will join the RAWFIE project in the context of the Open Calls. The consortium will also evaluate how these constraints may affect the RAWFIE architecture and the chosen technologies in the second iteration.

It is therefore proposed to allow the RAWFIE system to identify and specify the time constraints, which will be used to instruct the system and its components about their characteristics so that the system can

1) try to meet the constraints,
2) check if the constraints are met,
3) take appropriate measures in case they are not met.

These activities can be done at any level in the system (in all tiers, in components, at system-level, in testbeds, in proxies, in UxV, etc.)

3.2.3 Techniques

In a system like RAWFIE, dealing with latency and other real-time requirements implies checking for the properties of the components of the system and their behaviour, independently or once integrated as a system. The components must be checked for the support for real-time constraints, i.e. the possibility to specify these constraints in the description and the parametrisation of the components and then to check if these constraints are met by the components once they are running in an integrated system. Of course the same applies at the system level and it is often difficult to separate these activities (thus it makes difficult to separate what should be in D4.4 and D4.5).

Constraints identification and specification. Timing requirements can be extracted from several sources, such as regulation and local recommendations, from technical notes issued by UxV manufacturers, conclusions given by previous similar experiments and other application requirements. For example, the UxV manufacturer will require a round trip time of n seconds; the regulations may require to be able to limit the deviation of a trajectory or path to x meters, which translates into a control period given by function f(x, v, w, …), which is a formula, taking into account the speed, weight and other variables; other deployments of UxV in a similar setup may have shown that the delay between issueing a command from the control centre to the UxV shall be at most y seconds, where y can span from 0.1 to hundreds. Etc.

The specification of time constraints should include the corrective action (that corresponds to the fallback scenario) to be performed in case the constraint is not met, e.g. activate emergency mode or return to a safe location, etc.

Constraints specification. All the above constraints must be translated into durations that will be checked by the appropriate entities in the operational system: for example the visualisation

tools needs to be updated with the location of a UxV every hour, if not, then a question mark should be displayed in red at the latest know position. The corresponding pieces of code should be created in the EDL editor (constraint and fallback scenario).

Constraints verification. The verification of the constraints is done by the entities for which these constraints have been specified in the description of the entity, using the EDL. The implementation of the verification is left to the developer of the entity. A possible yet simplistic way is to have a timer dedicated to the constraint, that is started on the initial conditions (usually after a system reset) and reset when specific conditions are met; if these conditions are not met, then the routine associated to the elapsed timer is executed.

3.2.4 Benchmarking and dimensioning

Benchmarking the software on reference platforms will give indications for the dimensioning of the RAWFIE system so that, in similar conditions, it can meet the time constraints for most of the experimentation execution.

Even though benchmarking and the knowledge of performance indicators allows for such dimensioning, it shall not prevent the insertion of specific mechanisms for checking that the constraints are met and defining the fall-back scenarios in the experiments.

3.3 Front-end Tier

A web based GUI is provided that enables the user to interact with the RAWFIE system. Most of the available frontend tools are integrated into a common web app framework, with some third party web applications accessible via web links.

The aim of the frontend tier is to provide centralised access to a single RAWFIE web portal that integrates all the functionalities available for the experimenters.

It communicates with the middle tier services via commonly used web technologies (SOAP, HTTP/REST, AJAX, and Web Sockets). Some server side back-ends of the tools may also directly access the Data Tier via repository specific protocols (i.e. JDBC).

3.4 Middle Tier

The Middle Tier is made of a collection of services that provide several management and processing functionalities. These services implement the core functionality of the RAWFIE platform. Middle Tier entities will support deployment in cloud environments.

The internal communication between the different services uses REST and Avro RPC [3] interfaces for direct and Request/Response based communication, as well as the Message Bus for asynchronous notifications. The communication to the Data Tier uses the application specific protocols, like JDBC, LDAP, as well as Java-HDFS-API or WebHDFS REST API [13] for

accessing data stored in the Hadoop Distributed File System ([14] and A.1). The communication with the Testbed Tier is mainly done via the Message Bus.

Every RAWFIE component described in Chapter 4 uses the above communication interfaces for the exchanges with the other components. The component descriptions mention these exchanges as inputs and outputs.

3.5 Data Tier

The Data Tier consists of several repositories and databases. There is no direct interconnection between the components in this tier (relation will be indirectly via Middle Tier components).

The different repositories are:

- *Master Data Repository*, to contain all the management data sets (experiments, EDL scripts, bookings, testbeds and resources, status information of testbeds and their resources, and so on) of RAWFIE. It will only be small to medium sized and have relational dependencies. This is the main reason for using a relational database [5] for storing this data. PostgreSQL [6] with PostGIS extension was chosen for the implementation, as it is well supported, open source and stable, and to be able to easily handle geo-referenced data
- *Measurements Repository*, that will use a big data storage system for storing the large number of measurements that will be coming from the sensors on board of the UxVs during the experiments. The popular big data solution “Hadoop Distributed File System” ([14] and A.1) is one of the potential solutions for this purpose, however the specific technological choice will be detailed in further WP4 deliverables and in WP5. In addition, a NoSQL solution is expected to be adopted in the 2nd implementation iteration to better manage the data sets. Currently HBase (running on top of HDFS) has been identified for this purpose.
- *Analysis Results Repository*, uses a special database for performing the Data Analytics job over the results of the experiments. The Graphite [17] data analysis framework will used with it database called Whisper [18]
- *Users & Rights Repository*, uses a LDAP [7] repository, as LDAP is a de facto standard for user management. It stores all user related data (name, organisation, address, password) and group memberships (roles based access control). The selected implementation is OpenDJ [8].

Except for the Analysis Results Repository, all used repository systems (PostgreSQL [9], HDFS [14], OpenDJ [10]) support replication, hence they do provide fault tolerance. In case of data loss in the Analysis Results Repository, they can be recomputed using data stored in the Measurements Repository

3.6 SFA interface and service

To provide an interface compatible to other FIRE facilities, an SFA Service that implements the SFA interface will be developed. This will allow the experimenters to access parts of the RAWFIE systems also via SFA clients. SFA service in the middle-tier will act as a parser of the RAWFIE data structure (resource specification) that will be used. This resource specification will be used by the SFA Aggregate Manager to list and describe the RAWFIE testbed local resources (either to advertise all resources or reserved resources) and by the SFA clients to describe the desired resources. SFA implements the Registry API and Aggregate Manager API. The SFA Registry can be part of the SFA service module and should run on top of a database in order to store any related registration information. In every testbed, an Aggregate Manager API should be implemented based on the Geni Aggregate Managers [12] in order to provide to SFA clients all the testbed information. Both APIs are describe in detail in A.2.

3.7 Testbed Tier

This testbed tier encompasses the infrastructure (both in terms of software and hardware elements) that needs to be deployed to the Testbeds facilities in order to support the execution and monitoring of experiments as well as the data exchanged with the middle tier and the UxVs. The UxV nodes are considered as part of the testbed tier. The testbed tier maintains a local database for storing information needed for the testbed and its experiments and does not directly interact with the RAWFIE data tier.

The different kinds of Testbeds (Maritime, Vehicular and Aerial) share a common testbed interface that abstracts their particularities and exposes a unified access to and from the other tiers.

3.7.1 Common Testbed Interface

The Testbeds themselves may be very heterogeneous due to different constraints and characteristics of the selected area/region. Each testbed however shall adhere to a Common Testbed Interface that includes:

- message bus clients, that is, publishers and consumers of the Message Bus (software perspective),
- high bandwidth connection capabilities with sufficient security for the communication with the Middle Tier (networking perspective).

The messages exchanged between Testbed Tier and the Middle Tier includes:

- Messages related to the control of an experiment (start, stop, cancel, etc.),
- Messages related to sending of status and health information for each testbed,
- Messages related to experiment data/measurements collected during an experiment that need to be analysed by the platform data analytics engine,

- Location information of the various devices that can be used for coordination, monitoring and visualization purposes.

More details on the exact Messages/Commands that are supported between the Testbed Tier and the Middle Tier are provided in WP5 deliverables. All these messages refer to the actual application specific interactions imposed by the type of experiments that need to be supported by RAWFIE. It does not address issues related to resource discovery and reservations that are expected to be based on the SFA standard.

It must be also noted here that the testbed components implemented in RAWFIE comprise just a reference implementation that may be adopted by Testbed providers. Testbeds facilities of core RAWFIE project partners will use these components and defined structure as much as possible. However, external testbeds, including the ones that will be integrated through the Open Calls, will decide on their own whether to use already software components implemented by the project, or implement their own.

3.7.2 Constraints for testbed integration

This section summarizes the general needs and constraints that a testbed facility must fulfil in order to be able to connect and operate within the premises of the RAWFIE federation. These do not only adhere to the envisaged testbed architecture but are also related to administrative and logistics aspects. The integration of a new testbed site in RAWFIE implies that the candidate testbed shall:

1. implement the Common Testbed Interface that mandates asynchronous message bus communication in all types of interactions that relate to RAWFIE specific experiments' handling and data gathering (as described in section 0)
2. implement the required SFA Aggregate Manager Interface prescribed for FIRE compatible testbeds for what has to do with resource discovery (as described in section 3.6)

Besides that, each testbed should provide additional infrastructure/resources that include at least:

1. Dedicated computational resources for executing the UxVs control commands and handling sensor data messages from multiple devices with a reasonable rate (testbed/UxV specific),
2. A high quality internet connection, as the testbed needs a connection to the RAWFIE Cloud platform,
3. Appropriate maintenance area (usually protected) for storing UxV devices that are not in the field,
4. Monitoring infrastructure that provides timely information on the exact location of all UxV devices involved in experiments. The monitoring should be independent of the positioning info that UxVs may provide,

5. Power supply that guarantees uninterrupted operation during experiments execution for the testbed's ground components,
6. Availability of personnel during testbed operational hours (needed for safety reasons and for transporting devices from/to the testing field).

A general requirement analysis is given in D3.2

3.7.3 Constraints for UxV integration

This section lists the main architectural constraints regarding the UxV integration. A general requirement analysis is given in D3.2.

The hereafter-described constraints shall be provided directly by the on-board software suite of the UxV or, optionally, by a proxy translating UxV specific protocols and network interfaces to the RAWFIE UxV Protocol

1. UxVs shall be equipped with network communication devices,
2. UxVs shall publish/subscribe to information to the RAWFIE message bus,
3. UxVs shall record and transmit sensor data,
4. UxVs shall be able to store sensor data internally (for later transmission),
5. UxVs shall periodically publish position, orientation, velocity,
6. UxVs shall periodically publish on-board storage usage, fuel usage, CPU usage,
7. UxVs shall be capable of processing sensor data in order to summarize large sensor data-sets,
8. UxVs shall receive and act upon RAWFIE command messages to control the UxV remotely (e.g. from the Front-end Tier).

3.8 Message Bus

The Message Bus is used for two main purposes: for asynchronous communication inside the Middle Tier and for all data exchange between the Middle Tier and the Testbed Tier.

As reported in D5.1, for the implementation of the RAWFIE prototype in the first iteration, Apache Kafka [4] has been chosen for implementing the message bus, by taking into consideration the following aspects / advantages:

- capability to automatically spread data and, consequently, workload across a cluster of machines, thus allowing scalability in a cloud environment,
- capability to automatically replicate data over multiple servers (brokers), thus ensuring fault tolerance,
- built-in persistence mechanisms, which allows the system to easily deal with issues like the temporary overload of the network connection, or temporary disconnections
    - a Kafka broker stores all messages received in a ring buffer for a configurable amount of hours (until disk is full or the max log size is reached). So messages

could also be read hours later. Producers could also be implemented in a way, that they buffer messages locally, until they can be sent to a Kafka broker.

- high throughput, in terms of messages per seconds.
- build in security mechanisms that can be enabled during message exchange

As serialization format of the messages on the bus, Apache Avro [2] war chosen (a preformat binary format).

An important parameter to consider when analysing the different communication patterns in RAWFIE is the latency (see also Chapter 3.2), defined as the amount of time a message takes to reach the receiver/s, after it has been sent by the publisher. The latency of the message bus has a great impact on the timeliness of the RAWFIE operations, since it may induce delays and jitter.

Different aspects of the system architecture may affect the latency in the communication, apart from the chosen software technology itself: these aspects include the communication network, too.

Performance tests for specific RAWFIE use case scenarios are ongoing and will be presented in WP6 deliverables. Since the tests results can also affect the final technological choices for the communication between UxVs and other testbed components in the 2nd implementation iteration, the following configuration and implementation optimisations are currently used or envisaged, for reducing the latency when using a publish/subscribe communication pattern, and specifically Apache Kafka:

- use of different partitions (a partition in Apache Kafka is the equivalent of a message queue for other messaging systems, which can be spread across different servers for scalability) for the different UxVs: this ensures that the messages of the various UxVs do not intermix, provides much sorter message bus queues dedicated to a particular UxV and much faster response times,
- use of producers with small batch size or small linger time: messages are sent (almost) immediately after produced (without waiting for other messages to be sent in a batch),
- small fetch times and small fetch sizes of consumers: messages are read (almost) immediately after being available at the broker (without waiting for other messages to be read in a batch),
- configure different topics for messages with different priorities (e.g. for critical commands, if any, and for simple sensors measurements). This way, the consumers can also be customised so as to consume the messages from the “higher priority” topics first,
- using of asynchronous messages producers, i.e. producers will not have to wait the response from the broker. The synchronous response may be then replaced by a callback to check for communication errors.

As a further optimisation, a local Message Bus (message broker) installation could also be envisaged within each Testbed: the internal communication between e.g. the Resource Controller and the UxVs will be performed in a local, controlled network environment, thus reducing the impact of the network in the latency of the communication. The overall workload in the message bus will be reduced, and the local message bus system can be adjusted to the needs of the Testbed itself. Testbed components may access both message bus systems or only one. For component that only access the local message bus but have to communicate with the Middle Tier (deployed in the Cloud) a *Messaging Bridge*[19] may be used to connect both systems. It will filter and transform messages appropriately.

4 Components Overview

This chapter provides a high level description of the components and the interactions between them. Deliverable D4.5 will give a more detailed description of the components and interfaces.

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. Also the (main) data flow should be described in this way: • Some other component o IN ← data read from the other component o OUT → data sent to the other component o IN/OUT ↔ data read and sent to the other component
Changes	Describes short the difference to D4.1, e.g. • none (no or minor changes) • new (new component) • add functionality … • removed functionality…

Table 1: Template for components' description

4.1 Front end tier

4.1.1 Web Portal

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.
Changes	• none

Table 2: Web Portal

4.1.2 Wiki Tool

Component	Wiki Tool
Responsible partner	Fraunhofer
Parent Component	Web Portal
Description	Provides documentation and tutorials to the users of the platform.
Provided functionalities	Contains: • well-organized and self-contained tutorials and documentation • read access to all user • write access only to special users
Relation to other components	• Other tools may have direct links to the manual in the wiki
Changes	• new

Table 3: Wiki Toll

4.1.3 Resource Explorer Tool

Component	Resource Explorer Tool
Responsible partner	Fraunhofer
Parent Component	Web Portal
Description	The experimenter can discover and select available testbeds as well as resources/UxVs inside a testbed with this tool. Administrators can manage the data.
Provided functionalities	• Visualize Data from the “Testbed, Resources” directory • Provide ability to search and select available resources inside a testbed • Update/edit/delete data (for administrators)
Relation to other	• Testbeds Directory Service (REST/RPC API)

components	o IN/OUT ↔ read and add/update testbed and UxV data • Booking tool (HTTP redirect) o OUT → send selected resources
Changes	• Editing and updating of testbed and UxV data added

Table 4: Resource Explorer Tool

4.1.4 Booking Tool

Component	**Booking Tool**
Responsible partner	HAI
Parent Component	Web Portal
Description	The Booking tool will provide the appropriate Web UI interface for the experimenter to discover available resources and reserve them for a specified period. The Reservation of resources should be compatible with the SFA Architecture and the concept of slices allocations employed by SFA implementations such as myslice. • The Booking Tool will be responsible for handling only user level reservations[4].
Provided functionalities	• Visualize the available dates and timeslots for each testbed resources (i.e. through a calendar like view) • Enables selection of the preferred date, timeslot(s) in a testbed • Enables reservetion of UxV resource(s) for specified time interval (one or more consecutive timeslots) • Enables modification or removal of existing user/experimenter selections • Visualize the status of a reservation (pending/reserved) • Initial, selections, modifications and removals will be validated by the Booking Service.
Relation to other components	• Resource Explorer Tool (HTTP redirect) o IN → selected resources for booking • Booking Service (REST/RPC API) o IN ←load existing bookings o OUT → new/edited/deleted bookings • Testbed Directory Service (Resource Explorer Tool) (REST/RPC API) o IN ← load selected UxV resources for testbeds
Changes	• modification/removal or reservations added • Visualization of user reservation status added

Table 5: Booking Tool

[4] Experiment level reservations are performed indirectly during experiment authoring or scheduling based on initial user level reservations.

4.1.5 Experiment Authoring 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 etc.
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 (i.e., EDL Compiler and Validator, Experiment Validation Service, Experiment and EDL Repository, Launching Service). 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 while the experimenter will have the opportunity to store / retrieve EDL scripts, at any time, through specific buttons and menus. 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. • EDL Compiler and Validation (HTTP, SOAP) o IN ← EDL script, errors, warnings o OUT → EDL script • Experiment Validation Service (HTTP, SOAP) o IN ←EDL script, errors, warnings o OUT → EDL script • Launching Service (REST/RPC API) o OUT → Experiment ID • Experiment and EDL Repository (JDBC) o IN ← EDL script, experiment data o OUT → EDL script, experiment data
Changes	• Visual editor functionalities: The experimenter can insert waypoints for each node.

Table 6: Experiment Authoring Too

4.1.6 Experiment Monitoring Tool

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) • Cancel a running experiment
Relation to other components	• Launching service (REST/RPC API) o OUT → cancel a running experiment • Master Data Repository (JDBC) o IN ← state of experiments • System Monitoring Service (REST/RPC API) o IN ← state of resources
Changes	• Cancelation of running experiments added

Table 7: Experiment Monitoring Tool

4.1.7 System 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 and testbed
Provided functionalities	• Show status of RAWFIE system infrastructure (servers, services, testbeds, UxVs) • Highlight potential problems
Relation to other components	• System Monitoring Service (REST/RPC API) o IN ← state of middle tier infrastructure
Changes	• Testbed & UxV monitoring added

Table 8: System Monitoring Tool

4.1.8 UxV Navigation Tool

Component	UxV Navigation Tool
Responsible partner	CERTH
Parent Component	Web Portal
Description	This component will provide to the user the ability to (near) real-time 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 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 the communication between the UxV Navigation Tool and the Resource Controller, does not need to be real time, as the execution of the instructions will start right after the whole list of instructions/commands are provided. On the other side of the spectrum, the Resource Controller communicates in real time with the UxVs so as to transmit and receive the given instructions.
Provided functionalities	• Experiments will have the ability to select the next desired location for each unit using one of the following interfaces: o 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. o Users will also have the option to manually navigate the robots by providing the coordinates of the next chosen position
Relation to other components	• Resource Control (via Message Bus) o OUT → transmitting the user's instructions
Changes	• None

Table 9: UxV Navigation Tool

4.1.9 Visualisation Tool

Component	**Visualisation Tool**
Responsible partner	EPSILON
Parent Component	Web Portal
Description	Visualisation of an ongoing experiment as well as visualisation of experiments that are already finished
Provided functionalities	• Geospatial data visualisation from available external providers; • Show/track all moving UxV resources; • visually connect UxVs and display relevant parameters for each of them • Store and load user settings for a specific experiment • Replay an experiment that is already finished
Relation to other components	• Visualisation Engine (via websockets) o IN ← Load experiment settings o IN ← Get real time data for UxVs and sensors

	o IN ← Get data for experiment replay o OUT → Store preferred user settings
Changes	• Updated the relation to the visualisation engine and the functionalities and updated the description with more details

Table 10: Visualization Tool

4.1.10 Data Analysis Tool

Component	**Data Analysis Tool**
Responsible partner	HES-SO
Parent Component	Web Portal
Description	The Data Analysis Tool enables the user to browse available data sources for subject to analytical treatment as well as previous analysis tasks' outcomes. Through the tool, data analysis learning tasks can then be initiated and carried out for as long as needed depending on the nature of the analysis to be performed or the type of the data being analysed. It namely enables the user to launch jobs on streaming data which by definition are being performed continuously and never end. Once the results of a given analytical job are obtained, either continuously in a streaming scenario or in a batch manner when the execution is complete and ended, the Data Analysis Tool provides access to the associated results persistently stored in the Analysis Results repository. Finally, results coming from streaming analytical tasks can be displayed in a real time fashion on a dashboard configured from the Data Analysis Tool coupled with the data storage in use for the given analysis.
Provided functionalities	• Visualises past results from the Analysis Results Repository via its dashboard integration • Defines and specifies data analytical/learning tasks to be executed on specific data sources by the Data Analysis Engine • Requests available schemas from the Schema registry. This is a sub-component of the message bus. It is the portion that handles version invariance • Provides commands to the Data Analysis Engine to launch the execution of the various analytical task defined beforehand through the Data Analysis Tool
Relation to other components	• Message Bus o IN ← read schemas • Analysis Results Repository o IN ← read results (via the Graphite UI from the Result Repository) • Data Analysis Engine o IN ← get analytical job status (via Message Bus) o OUT → submit commands (via Message Bus)
Changes	• Updated the description to be more thorough and include the elements mentioned in the functionalities section

	• Refined the provided functionalities to be more explicit and sorted them to fit the order in which they will most likely be used • There is no change in the relations to other components compared to the previous iterations of the document. • Added all the changes triggers by the separation of the Measurements and Results Repository into two separate entities, the Measurement Repository and the Analysis Results Repository.

Table 11: Data Analysis Tool

4.2 Middle Tier

4.2.1 EDL Compiler & Validator

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*.
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 through the provided inter. 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. • Experiment Authoring Tool (HTTP, SOAP) o IN ← EDL script o OUT → EDL script, errors, warnings • Master Data Repository (JDBC) o OUT → EDL script, experiment data • Experiment Validation Service (HTTP, SOAP) o OUT → EDL script • Experiment Controller (REST/RPC API) o OUT → Experiment data
Changes	• none

Table 12: EDL Compiler & Validator

4.2.2 Experiment Validation Service

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. Cross experiments validation will be performed accompanied by qualitative characteristics of an experiment. 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: • EDL Compiler & Validator (HTTP, SOAP) o IN ← EDL script o OUT → EDL script, errors, warnings • Testbeds Directory Service (REST/RPC API) o IN ← Testbeds information • Experiments and EDL Repository (JDBC) o IN ←EDL script, experiment data o OUT →EDL script, experiment data • Experiment Authoring Tool (HTTP, SOAP) o IN ← EDL script o OUT → EDL script, errors, warnings
Changes	• none

Table 13: Experiment Validation Service

4.2.3 Users & Rights 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 • Users & Rights Repository (LDAP interface) o IN/OUT ↔ Read and update user and rights data
Changes	• none

Table 14: Users & Rights Service

4.2.4 Booking Service

Component	**Booking Service**

Responsible partner	HAI
Parent Component	None
Description	The Booking Service manages bookings of resources by registering data to appropriate database tables and possibly providing notification mechanisms to the experiments The Booking Service is responsible for processing and validating all reservations requests at user or/and experiment level initiated by the platform.
Provided functionalities	• Validates all reservations requests (add, edit, delete) based on a set of predefined constraints/checks • Coordinates reservations of testbed resources among experimenters • Provides Notification mechanisms (reminder for experiments) for the status of their reservation • Ensures fairness in resource bookings (part of validation process) • Interacts with the persistence store (Relational DB Tables)
Relation to other components	• Booking Tool (REST/RPC API) o IN ← new/edited/deleted bookings o OUT → Changed status of pending reservation o OUT → existing Booking info • Master Data Repository (JDBC) o IN/OUT ↔ Execution of SQL queries for retrieving or updating reservation related entities in the DB • Launching Service, Experiment Authoring Tool (REST/RPC API) o OUT → existing user level reservation info for an experiment
Changes	• None or minor changes

Table 15: Booking Service

4.2.5 Launching Service

Component	**Launching Service**
Responsible partner	HAI
Parent Component	None
Description	The Launching Service is responsible for handling requests for starting or cancellation of experiments. Regarding launching functionality it will support: (a) ***Short-term launching (manual launching)***: The LS will give the opportunity to experimenters to directly initiate pre-defined and pre-approved experiments stored in the RAWFIE system. This type of launching will involve user interaction following the authoring and validation of an experiment and is expected to be used mainly for debugging purposes[5]. (b) ***Long-term launching (scheduled launching)***: The LS will

[5] this **functionality** will be available if the corresponding testbed is already configured (i.e., UxVs are in place after a user level reservation has taken place and the necessary code is uploaded to nodes).

	maintain an internal scheduler which will be triggered and initiate experiment requests according to the available experiment level reservations. It should be noted, that the LS will execute only authorized and approved experiments based on spatio-temporal constraints that will be validated just before launching. In both cases the LS will require as input an experiment Identifier and will initiate a proper *StartExperiment* request containing a unique execution Identifier or return proper error feedback.
Provided functionalities	• The LS will provide the following functionalities: • Enables direct initiation of an experiment from the UI. • Enables scheduling of experiments at a future time based on experiment level reservations. • Enables cancellation of running or scheduled experiments
Relation to other components	The LS will interact 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. • Experiment Authoring Tool (REST/RPC API) o IN ← experiment to be launched • Experiment Monitoring Tool (REST/RPC API) o IN ← experiment to be cancelled • Experiment Validation Service (REST/RPC API) o IN/OUT ↔ validation feedback for provided experiment • Experiment Controller (Message Bus) o OUT → Start or Cancel Experiment request • Master Data Repository (JDBC) o IN/OUT ↔ Execution of SQL queries for retrieving or updating experiment and reservation related entities in DB
Changes	• Cancellation of experiments added • Interaction with Experiment Validation Service added • Interaction with TestbedProxy removed • Interaction with Experiment controller achieved via the Message Bus

Table 16: Launching Service

4.2.6 Experiment Controlle

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 main task of the experiment controller is the monitoring of the experiment execution while acting as ‘broker’ between the experimenter and the resources.

	The EC will provide 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 (e.g., implement 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 acts as 'broker' between the experimenter and the resources. • The EC monitors the course of actions during the experiments execution and informs the appropriate services in the Front-end layer. • It forwards instructions from the experimenter to the resources.
Relation to other components	• Launching Service (Message Bus) o IN ← experiment to be launched • Resource Controller (Message Bus) o IN/OUT ↔ Experiment Details, Start or Cancel Experiment request • System Monitoring Tool (Message Bus) • OUT → Experiment Status Experiment Monitoring Tool (Message Bus) o OUT → Measurements and Experiment Details (Position of the vehicles)
Changes	• None

Table 17: Experiment Controller

4.2.7 Data Analysis Engine

Component	**Data Analysis Engine**
Responsible partner	HES-SO
Parent Component	None
Description	The Data Analysis Engine enables the execution of data processing jobs by sending requests to a processing engine (either stream processing engine, batch or micro-batch) which will perform the computations specified when the analytical task was defined through the Data Analysis Tool to be transmitted to the processing engine for execution. The Data Analysis Engine contains two major subcomponents: • Compute Engine: the implementation that distributes data analysis computations (e.g. BLAS operations, etc.) over nodes in

	a cluster, built using the Apache Spark framework. • Frontend: the portion that, based on the information given by the user through the Data Analysis Tool, relays the associated data to the compute engine that will perform the desired analytics. The frontend component of the Data Analysis Engine uses the message bus schema registry in order to filter what will be subject to computation.
Provided functionalities	• Requests the execution of a stream/batch processing job • Stores the results of the analysis in the Results Repository (Whisper).
Relation to other components	• Data Analysis Tool (via Message Bus) o IN ← receive analytical task execution command o OUT → send analytical task status • UxV - Sensor & Localisation (via Message Bus) o IN ← read data from UxV direcly through the message bus (streaming) • Measurements Repository o IN ← read measurements (batch) • Analysis Results Repository o OUT → send results
Changes	• Updated the description to involve explicitly the component coupled with the engine, the Data Analysis Tool, and exhibit the nature of the relation. • Added all the changes triggers by the separation of the Measurements and Results Repository into two separate entities, the Measurement Repository and the Analysis Results Repository.

Table 18: Data Analysis Engine

4.2.8 System Monitoring Service

Component	**System Monitoring Service**
Responsible partner	Fraunhofer
Parent Component	None
Description	Checks readiness of main components and ensure that all critical software modules will perform at optimum levels. Predefined notification are triggered whenever the corresponding conditions are met, or whenever thresholds are reached
Provided functionalities	• Check the performance, utilizing Key Performance Indicators (KPI) • Send notifications (e.g. via email) when triggers are reached • Capture alarms caused by malfunction or underperformance of the equipment.
Relation to other components	• all middle tier and testbed components (Message Bus or REST/RPC API) o IN ← status and performance values • System Monitoring Tool (REST/RPC API) o OUT → collected status and performance values

	• some messaging system (e.g. email or SMS) o OUT → send notifications to user
Changes	• monitoring of testbed components added • concept of general actions removed

Table 19: System Monitoring Service

4.2.9 Testbeds Directory Service

Component	**Testbeds Directory Service**
Responsible partner	IES
Parent Component	None
Description	Represents a 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 software interface for most of the information available / that will be stored in the Master Data Repository. This includes information relevant to the testbeds (name, location, description, type of resources supported, status of testbed and related resources, etc.) and associated resources (name, location, description, type, status) as well as information on the capabilities in terms of available technologies and associated tests of a particular resource
Provided functionalities	• Provides a REST API. This API allows other components to get access to the information contained in the corresponding repository (Master Data Repository), by realising traditional CRUD (Create, Read, Update, Delete) operations • Provides the pointers to the different testbeds belonging to the RAWFIE federation • In particular, using the provided software API it will be possible to: o Register / Unregister new testbeds in the RAWFIE platform (Create / Delete) o Register / Unregister resources to specific testbeds of the RAWFIE platform (Create / Delete) o Look at the available testbeds list, description, available resources, and at their status, e.g. free, booked, in use, and so on (Read) o Look at the available resources within a given testbed, their description, characteristics, capabilities and their status, e.g. free, booked, in use, not operational, and so on (Read) o Edit information on testbeds and resources (Update) o Look at the testbeds and resources capabilities in terms of available technologies and tests (Read) o Get information on testbeds and resources according to pre-defined filters, e.g. type of UxVs available (Read)
Relation to other components	• Resource Explorer Tool (REST/RPC API) o IN/OUT ↔ read and add/update testbed and UxV data, via a REST interface • Master Data Repository (JDBC)

	o IN/OUT ↔ Perform SQL queries and updates on Testbeds and Resources (UxVs) entities (via JDBC connection to the database)
Changes	• Enhancements to the final list of functionalities supported, according to the new requirements definition in D3.2

Table 20: Testbeds Directory Service

4.2.10 Accounting Service

Component	**Accounting Service**
Responsible partner	Fraunhofer
Parent Component	None
Description	Keeps track of resources usage by individual users.
Provided functionalities	• Receive events from the other subsystems • Count charge in Credit units per User
Relation to other components	• Master Data Repository (JDBC) o IN ← Perform SQL queries status of experiments o OUT → accounting information for the users
Changes	• new

Table 21: Accounting Service

4.2.11 Visualisation Engine

Component	**Visualisation Engine**
Responsible partner	Epsilon
Parent Component	None
Description	Used for providing the necessary information to the Visualisation tool, to communicate with the other components, to handle geospatial data, to retrieve data for experiments from the database, to load and store user settings and to forward them to the visualisation tool
Provided functionalities	• Load and store user setting to and from the database in order to have the same settings loaded for the same experiment for a specific user • Load data for an already finished experiment from the database, update it and provide it to the visualisation tool • Load data for UxVs, for sensors, for location and others for an experiment that is currently being visualised • Provide data for an existing experiment in appropriate format as layers to the visualisation engine • Retrieve real time data from the message bus, update it as necessary and forward it to the visualisation tool • Open websocket interface to the visualisation tool and handle all its requests
Relation to other components	• Visualisation Tool (via websockets) o OUT → Load experiment settings o OUT → Get real time data for UxVs and sensors o OUT → Get data for experiment replay o IN ← Store preferred user settings

	• UxVs and sensors (via Message Bus) o IN ← Get real time data for • Master Data Repository o IN ← SQL queries for retrieving testbed, UxV and experiment entities from DB
Changes	• New

Table 22: Visualisation Engine

4.2.12 Message Bus

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 were investigated and, as reported in D5.1, for the implementation of the RAWFIE prototype in the first iteration, Apache Kafka has been chosen for implementing the message bus (see also Section 3.8).
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	Being the Message Bus the main integration component in the RAWIFE architecture, different components are involved in the communication through the Message Bus. Details in the specific components sections
Changes	• Decision to use Apache Kafka as implementation • Relation to other components simplified

Table 23: Message Bus

4.3 Data tier

4.3.1 Master Data Repository

Component	**Master Data Repository**
Responsible partner	UOA (supported by all technical partners)
Parent Component	None

Description	Stores all main entities that are needed in the RAWFIE platforms. Is an SQL-database
Provided functionalities	• SQL interface • Stored entities (not comprehensive) o Testbeds (name, location, type, …) o UxVs/resources (name, type, status, …) o Experiments and EDL scripts o Experiment status o Bookings/reservations o User settings
Relation to other components	• IN/OUT ↔ almost all components in the Middle tier.
Changes	• New, combination of Testbeds & Resources Repository, Experiments & EDL Repository, Bookings Repository, Status Repository

Table 24: Master Data Repository

4.3.2 Users & Rights Repository

Component	**Users & Rights Repository**
Responsible partner	Fraunhofer
Parent Component	None
Description	Management of users and their roles. Is a directory services (LDAP).
Provided functionalities	• LDAP interface • Stores users and their roles (a role correspond to a group membership)
Relation to other components	• Users & Rights Service (LDAP) o IN/OUT ↔ Read and update user and rights data
Changes	• none

Table 25: Users & Rights Repository

4.3.3 Measurements Repository

Component	**Measurements Repository**
Responsible partner	HES-SO
Parent Component	None
Description	Stores the raw measurements from the experiments in HDFS (file system). Those measurements can also be accessed in a database fashion with HBase, the NoSQL distributed database that will run on top of HDFS.
Provided functionalities	• Persistent storage the raw measurements from the experiments in HDFS for later analysis.
Relation to other components	• UxV - Sensor & Localisation (via Message Bus) o IN ← receive measurements • Data Analysis Engine (via Message Bus) o OUT [Symbol] send measurements
Changes	• Split into Measurements Repository and Results Repository

	• Status of experiments now stored in the Master Data repository

Table 26: Measurements Repository

4.3.4 Analysis Results Repository

Component	Analysis Results Repository
Responsible partner	HES-SO
Parent Component	None
Description	Whisper, as part of the Graphite project, is a big data time series database that constitutes the results repository. Whisper stores the results of data analyses and those results can be displayed via Graphite's dashboard, called Graphite-Web.
Provided functionalities	• Stores results of data analysis tasks • Displays the results via Graphite's dashboard (which will be integrated in the Data Analysis Tool)
Relation to other components	• Data Analysis Engine (via socket) o IN ← receive results • Data Analysis Tool (via HTTP/HTML) o OUT →integration of the dashboard in the Data Analysis Tool
Changes	• Split into Measurements Repository and Analysis Results Repository

Table 27: Analysis Results Repository

4.4 Testbed tier

4.4.1 Testbed Manager

Component	Testbed Manager
Responsible partner	HAI
Parent Component	None
Description	Contains accumulated information about the UxVs resources and the experiments of each one of the federation testbeds. It is responsible to address initial resources configuration and periodic updates of testbed on-going experiments. It supports an alternative storage path (*Local Data Repository*) for logging and configuration activities which can additionally be used for storing experiments data in case of connection failure to the Middle Tier.
Provided functionalities	• Contains the registration log for the experiments in the tested • Periodically checks the status of the experiments • Forwards the status of the experiments to the Middle Tier • 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	• Experiment Controller o IN ← Experiment launching info and status • System Monitoring Service o OUT → Testbed Manager status and performance • SFA Aggregate Manager o IN/OUT ↔ SFA Aggregate Manager API (XML-RPC) • Resource Controller o IN ← Current experiment status o IN ← UxV health status • Local Data Repository o IN ← UxVs configuration parameters o IN ← Data buffer in connection loss o IN ← Experiments log
Changes	• Local Data Repository added

Table 28: Testbed Manager

4.4.2 Aggregate Manager

Component	**Aggregate Manager**
Responsible partner	UOA
Parent Component	Testbed Manager
Description	The SFA Aggregate Manager API is the interface by which experimenters discover, reserve and control resources at resource providers. See section A.2.1
Provided functionalities	• Implements Aggregate Manager API
Relation to other components	• Testbed Manager o IN ← Testbed information about resources, configurations of resources and resources capabilities • SFA Service o OUT → An API to discover, reserve and control resources.
Changes	• Local Data Repository added

4.4.3 Monitoring Manager

Component	**Monitoring Manager**
Responsible partner	UOA
Parent Component	None
Description	Monitors the status of the testbed and the UxVs belonging to it, at functional level, e.g. the 'health of the devices' and current activity.
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. • Transmit the current status of the Testbed and its resources to the System Monitoring Service.
Relation to other components	• System Monitoring Service (via Message Bus) o OUT → testbed and resources status and performance values
Changes	• added transmission of status information to the System Monitoring Service

Table 29: Monitoring Manager

4.4.4 Network Controller

Component	**Network Controller**
Responsible partner	UOA
Parent Component	None
Description	Manages the network connections and the switching between different technologies in the testbed in order to offer seamless connectivity in the operations of the system. For example if a problem occurs in the communication of the resource with the RC and subsequently with the Experiment Controller 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	• Provisioning of the network connections/technologies required during an experiment • Checks the communication when devices are moving between obstacles • Verification that the time constraints specified on the exchanged data for the different types of UxVs are met. • Sends notifications produced in message bus to Resource controller and System Monitoring Service when the time constraints are not met via a specific network technology.
Relation to other components	• U-xV - Network communication o IN ← Communication statistics and status values (via network traffic analysis) • Resource Controller (via Message Bus) o OUT → notifications when network time constraints are not met • Monitoring Manager o OUT → notifications about network communications status
Changes	• added transmission of status information to the Monitoring Manager

Table 30: Network Controller

4.4.5 Resource Controller

Component	Resource Controller
Responsible partner	CERTH
Parent Component	None
Description	The Resource Controller can be considered as a cloud robot and automation system and ensures the safe and accurate guidance of the UxVs. Moreover, RC commands each device to switch onboard sensors on and off. At the same time, this component informs the Experiment Controller for the gathered measurements of the sensors of each device. “Launching Tool” interacts with the "Experiment Controller" 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 (“Navigation Service” sub-component) is to optimize the navigation process, which takes place during a turn. The optimization algorithm is 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. In other words, this component acts as an agent/daemon and invokes actions on the request of the experimenter. Through the module, each resource can be made discoverable to the rest infrastructure while the access control is also ensured. Moreover, the Resource Controller is responsible for the dispatch of information related to the current status of the node (i.e., energy reserves, currently active modules, location, velocity etc.). Resource Controller will be able to detect and identify possible safety violations. If the given instructions violate the safety constraints, e.g., the experimenter guides two units at the same position, the Resource Controller identifies and ignores these directions returning to the portal appropriate warning messages. Additionally, the Resource Controller ensures that the system is performing as intended. If one of the following conditions occurs, automatically, the component activates an emergency scenario: • 1. The component does not receive any feedback from the units for several time steps.

	• 2 The component receives feedback from the units which report severe localization issues. It is worth noting that 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.
Provided functionalities	• The calculation of 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 • 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	• Experiment Controller (Message Bus) o IN ← Experiment Details, start, stop and cancel instructions, desirable trajectory etc. • UxV Node (Message Bus) o IN/OUT ↔ Sends the desirable location and receives the actual position of the vehicles
Changes	• none

Table 31: Resource Controller

4.4.6 UxV node

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	UxVs typically 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 • 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	• Resource Controller o IN ← Provides resources to other components, Provides status on its internal resources. o OUT → Requests external resources. • UxV – On board processing o OUT → Data collected from sensor (samples). o IN ← Results of local data processing. • UxV – On board storage (buffering) o IN ← Data retrieved from local storage. o OUT → Data to be stored in local storage. • Data Tier o OUT → Provides measurement to the Measurements Repository. o OUT → Provides status and various statistics to the Master Data repositories. • Sensors & Localization o OUT → status & samples (with tagging information, such as timestamp), events. o IN ← sampling configuration, commands (start, stop, hold, individual sampling, etc.) • Experiment controller o OUT → Provide status and geographical information about the UxV. • System monitoring o IN ← requests for information. o OUT → Events, status, geographical & other UxV specific information (speed, altitude, etc.)
Changes	• Relation to other components

Table 32: UxV node

4.4.7 UxV - Network communication

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	• Network Controller o IN ← Request communication resources (such as bandwith) or other Quality of Service. o OUT → Provides communication status and statistics. o
Changes	• Relation to other components

Table 33: UxV - Network communication

4.4.8 UxV – Sensors & Localization

Component	UxV – Sensors & Localization
Responsible partner	CSEM (MST, Robotnik)
Parent Component	UxV node
Description	• Provides interfaces to different 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	• UxV o OUT → status & samples (with tagging information, such as timestamp), events. o IN ← sampling configuration, commands (start, stop, hold, individual sampling, etc.) o
Changes	• Relation to other components

Table 34: UxV – Sensors & Localization

4.4.9 UxV – On board storage

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	• UxV o OUT → Data retrieved from local storage. o IN ← Data to be stored in local storage.
Changes	• Relation to other components

Table 35: UxV – On board storage

4.4.10 UxV – On board processing

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	• UxV – On board storage (buffering) o OUT → Data retrieved from local storage. o IN ← Data to be stored in local storage.
Changes	• Relation to other components

Table 36: UxV – On board processing

4.4.11 UxV – Device management

Component	UxV – Device management
Responsible partner	CSEM
Parent Component	UxV node
Description	• Provides network and sensor management for the UxV
Provided functionalities	• Network connection to the base state • Ad-hoc networks • Device sensor controlling
Relation to other components	• Network controller o IN ← Communication configuration & quality of service management.

	o OUT → Communication Status & Statistics. • Resource controller o IN ← Resource reservation requests & configuration. o OUT → Resource reservation response & status.
Changes	• Relation to other components

Table 37: UxV – Device management

5 Requirement mapping

A simple requirement mapping to components was already given in D3.2.

This chapter shows how the system level requirements ("general" ones from D3.2) are addressed with the current architecture. The component specific requirement will be handled in D4.5.

- **PT-GEN-R-001: RAWFIE Platform should adopt Sliced Federated Architecture (SFA)**
  - A SFA service will be develop that implements the SFA interface. So user may also access some RAWFIE services via SFA
- **PT-GEN-R-002: RAWFIE platform shall support various roles with different privileges at every level of access.**
  - The User & Rights Service manages a set of roles for each user. Each role stands for a privilege. The other RAWFIE component have to ask the User & Rights Service, if the given user has the appropriate role to execute the requested operation.
- **PT-GEN-R-003: The RAWFIE Data model should include all basic entities that are used or/and exchanged by the various components of the RAWFIE Platform**
  - All main entities are now in the Master Data repository (users are separated in an LDAP server, but their unique IDs are used in the Master Data repository). It only contains a single relational database schema. Full SQL query capabilities are provided to all Middle Tier components.
- **PT-GEN-R-004: RAWFIE platform shall provide appropriate data storage for persistent information that needs to be reused later**
  - The Data Tier consists of three different specialized repositories. OpenDJ (Directory Server) for user data, PostgreSQL for the master data and HDFS/HBase for the potential huge amounts of measurements. See also section 3.5.
- **TB-GEN-R-001: Each UxV Testbed should provide a Slice Interface for federating their capabilities/resources to the experimenter.**
  - Testbeds need to implement the required the SFA Aggregate Manager Interface prescribed for FIRE compatible testbeds (as described in section 3.6)

- **TB-GEN-R-002: Each Testbed should provide the exact boundaries within which its UxVs can operate**
  - This data is stored in the Master Data repository and will be evaluated by the Experiment Validation Service
  - In the testbed themselves, the navigation service in the Resource Controller will also check if the boundaries are obeyed.
- **TB-GEN-R-003: Testbed areas should at least be able to host/operate multiple UxVs of one or more types**
  - Physical implementation is out of scope of the architecture
  - On the software side, each testbed can have an unlimited amount of UxVs (if appropriate computing power is available)
- **TB-GEN-R-004: Testbed areas environment should be closely monitored**
  - Out of scope of the architecture
- **TB-GEN-R-005: Indoor spaces of a testbed should provide a shielded indoor environment**
  - Out of scope of the architecture
- **TB-GEN-R-006: Testbed facility areas should comprise storing spaces and be able to receive inspect and assemble and/or fix UxVs**
  - Out of scope of the architecture
- **TB-GEN-R-007: Testbed facilities should provide dedicated services for emergency situations.**
  - Out of scope of the architecture
- **TB-GEN-R-008: Testbed areas should provide proper facilities and equipment**
  - Out of scope of the architecture
- **TB-GEN-R-009: Testbed must provide dedicated computational resources**
  - This is a mandatory requirement that a testbed has to fulfil. See section 3.7.2.
- **TB-GEN-R-010: Testbeds should be supported by on-site personnel**
  - Out of scope of the architecture
- **TB-GEN-R-011: Testbeds should conform to all legal regulations and restrictions**
  - Out of scope of the architecture
- **TB-UVG-001: Compliance of UxV to RAWFIE specification and interfaces**
  - This is a mandatory requirement that an UxV has to fulfil. See 3.7.3.

Part V: Annex

Annex A Relevant technologies

In this chapter additional technologies that extent the State of the Art chapter of D4.1 are described.

A.1 HDFS

HDFS (Hadoop Distributed File System) is a Java-based file system that provides scalable and reliable data storage, and it was designed to span large clusters of commodity servers. HDFS is a scalable, fault-tolerant, distributed storage system that works closely with a wide variety of concurrent data access applications. By distributing storage and computation (data analysis computations) across many servers, the combined storage resource can grow linearly with demand while remaining economical at every amount of storage.

HDFS features:

- Considers a node's physical location when allocating storage and scheduling tasks. (Rack awareness, data locality)
- Dynamically diagnose the health of the file system and rebalance the data on different nodes.
- Provides redundancy and supports high availability.
- HDFS requires minimal operator intervention, allowing a single operator to maintain a cluster of 1000s of nodes.

An HDFS cluster is comprised of a NameNode, which manages the cluster metadata, and DataNodes that store the data. Files and directories are represented on the NameNode by inodes. Inodes record attributes like permissions, modification and access times, or namespace and disk space quotas.

The data is split into large blocks (typically 128 megabytes), and each block is independently replicated at multiple DataNodes. The blocks are stored on the local file system on the DataNodes.

The Namenode actively monitors the number of replicas of a block. When a replica of a block is lost due to a DataNode failure or disk failure, the NameNode creates another replica of the block. The NameNode maintains the namespace tree and the mapping of blocks to DataNodes, holding the entire namespace image in RAM.

HDFS, as a core module of the Hadoop framework, enables compatibility with a wide range of Hadoop-related tools, known as the Hadoop ecosystem, that can be installed on top of or

alongside Hadoop to simplify access and processing of data stored in the Hadoop cluster. (Descriptive list of those projects at: https://hadoopecosystemtable.github.io).

To only mention the most relevant for the project's purposes:

- **Apache Spark**: In-memory cluster computing framework used for fast batch processing, event streaming and interactive queries. It is the successor of the Hadoop MapReduce programming model for large scale data processing, which is, like HDFS, a core module of the Hadoop framework. Spark is used by the Data Analysis Engine to perform the data analytical tasks. HDFS is also used by Spark for checkpointing, which is the process of truncating RDD (resilient distributed dataset) lineage graph and saving it to a reliable distributed file system.
- **Apache HBase**: Open source NoSQL distributed database. It runs on top of HDFS and its column-based behavior benefits from the parallelism and distributed behavior provided by HDFS.

A.2 SFA APIs

This section describes the mandatory SFA APIs that will be implemented in the RAWFIE system.

A.2.1 Aggregate Manager API

The Aggregate Manager API is the interface by which experimenters discover, reserve and control resources at resource providers. It does not include resource specific interactions, application level interactions, or monitoring and management functions. Three versions of AM API exist at the moment as AM API v1, 2 and 3, while a 4th version is in a draft state.

The latest stable version of the AM is the GENI AM API v3. A brief description of the methods of it is given in Table 38 as it is defined in [11].

Methods	Functionality
GetVersion	Query static configuration information about this aggregate manager implementation, such as API and RSpec versions supported. Get static version and configuration information about this aggregate. Return includes: • List of versions of the API supported by this aggregate. • List of request RSpec formats supported by this aggregate • List of advertisement RSpec formats supported by this aggregate. • List of supported credential types and versions. • Options on how sliver allocation works.
ListResources	Return a listing and description of available resources at this aggregate. The resource listing and description provides sufficient information for clients to select among available resources. These listings are known as advertisement RSpecs.
Describe	Retrieve a manifest RSpec describing the resources contained by the named

	entities, e.g. a single slice or a set of the slivers in a slice. This listing and description should be sufficiently descriptive to allow experimenters to use the resources.
Allocate	Allocate resources as described in a request RSpec argument to a slice with the named Uniform Resource Name (URN). On success, one or more slivers are allocated, containing resources satisfying the request, and assigned to the given slice. This method returns a listing and description of the resources reserved for the slice by this operation, in the form of a manifest RSpec. Allocated slivers are held for an aggregate-determined period. Clients must `Renew` or `Provision` slivers before the expiration time (given in the return struct), or the aggregate will automatically `Delete` them. Aggregates should implement `Allocate()` as quick, cheap, and not impacting provisioned resources, such that it can be readily undone. `Allocate` is an all or nothing request: if the aggregate cannot completely satisfy the request RSpec, it should fail the request entirely.
Renew	Request that the named slivers be renewed, with their expiration extended. If possible, the aggregate should extend the slivers to the requested expiration time, or to a sooner time if policy limits apply. This method applies to slivers that are `geni_allocated` or to slivers that are `geni_provisioned`, though different policies may apply to slivers in the different states, resulting in much shorter max expiration times for `geni_allocated` slivers.
Provision	Request that the named `geni_allocated` slivers be made `geni_provisioned`, instantiating or otherwise realizing the resources, such that they have a valid `geni_operational_status` and may possibly be made `geni_ready` for experimenter use. This operation is synchronous, but may start a longer process, such as creating and imaging a virtual machine.
Status	Get the status of a sliver or slivers belonging to a single slice at the given aggregate. Status may include other dynamic reservation or instantiation information as required by the resource type and aggregate. This method is used to provide updates on the state of the resources after the completion of Provision, which began to asynchronously provision the resources. This should be relatively dynamic data, not descriptive data as returned in the manifest RSpec.
Perform Operational Action	Perform the named operational action on the named slivers, possibly changing the geni_operational_status of the named slivers. E.G. 'start' a VM. For valid operations and expected states, consult the state diagram advertised in the aggregate's advertisement RSpec.
Delete	Delete the named slivers, making them geni_unallocated. Resources are stopped if necessary, and both de-provisioned and de-allocated. No further AM API operations may be performed on slivers that have been deleted.
Shutdown	Perform an emergency shutdown on the slivers in the given slice at this aggregate. Resources should be taken offline, such that experimenter access (on both the control and data plane) is cut off. No further actions on the slivers in the given slice should be possible at this aggregate, until an un-specified operator action restores the slice's slivers (or deletes them). This

	operation is intended for operator use. The slivers are shut down but remain available for further forensics.

Table 38: GENI Aggregate Manager API Version 2

A detail description of the methods provided by the AM APIs can be found in [12].

A.2.2 Registry API

A Registry is a trusted body that issues certificates to users or authorities and is organized in a tree-based hierarchical naming space. The Registry API is used by the users and authorities in order to be part of a trusted federation. A synopsis of the Registry AP [15] methods is shown in Table 39.

Method	Functionality
GetVersion	Get static version and configuration information about this Registry. Return includes: • The version of the Registry API supported • The root authority HRN of this Registry The URL's of the peers root authorities
Register	Registers an object (Authority, User, Slice) with the registry
Update	Updates an object (Authority, User, Slice) in the registry
Remove	Removes an object (Authority, User, Slice) from the registry
Resolve	Used to learn the detailed information bound to a Registry object (Authority, User, Slice)
List	Lists the set of Registry objects (Authority, User, Slice) managed by the named Authority
GetSelfCredential	A degenerate version of GetCredential used by the client to get his initial credential when he doesn't have one.
GetCredential	Retrieves the credentials corresponding to the named object (Authority, User, Slice)
CreateGid	Creates a signed certificate for the object with the registry

Table 39: Registry API

Annex B Abbreviations

The following table gives the abbreviations used across the RAWFIE projects in the documents and deliverables.

Abbreviation	Meaning
3D	three-dimensional space
ACL	Access Control List
AGL	Above Ground Level
AHRS	Attitude and Heading Reference System
AJAX	Asynchronous JavaScript and XML
AM	Aggregate Manager (of SFA)
AP	Access Point

API	Application Programming Interface
API	Application programming interface
AT	Aerial Testbed
AUV	Autonomous underwater vehicle
B-VLOS	Beyond Visual Line Of Sight
CA	Certification Authority
CAA	Civil Aviation Authority
CAO	Cognitive Adaptive Optimization
CBNR	Chemical Biological Nuclear Radiological
CEP	Circular Error Probability
CPU	Central Processing Unit
CSR	Certificate Signing Request
DETEC	Department of the Environment, Transport, Energy and Communication
DGCA	Directorate General of Civil Aviation
DoA	Description of Actions
EASA	European Aviation Safety Agency
EC	Experiment Controller
ECC	Error Correction Code
ECV	EDL Compiler & Validator
EDL	Experiment Description Language
EDL	Experiment Description Language
EER	Experiment and EDL Repository
EU	European Union
E-VLOS	Extended Visual Line Of Sight
EVS	Experiment Validation Service
FIRE	Future Internet Research & Experimentation
FOCA	Federal Office of Civil Aviation
FPS	Frames Per Second
FPV	First Person View
GAA	German Aviation Act
GIS	Geographic Information System
GNSS	Global Navigation Satellite System
GPIO	General Purpose Input/Output
GPS	Global Positioning System
GUI	Graphical user interface
HD	High Definition
HTTP	Hypertext Transfer Protocol
HW	Hardware
IAA	Irish Aviation Authority
IaaS	Infrastructure as a Service
IDE	Integrated Development Environment
IDE	integrated development environment
IFR	Instrument Flight Rules
IP	Internet Protocol
ISO	International Standards Organization
JDBC	Java Database Connectivity
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
KPI	Key Performance Indicator

LBL	Long Baseline
LDAP	Lightweight Directory Access Protocol
LS	Launching Service
MEMS	MicroElectroMechanical System
MM	Monitoring Manager
MSO	Multi Swarm Optimization
MT	Maritime Testbed
MOM	Message Oriented Middleware
MVC	Model View Controller
NAT	Network Address Translation
NC	Network Controller
NF	Non Functional
ODBC	Open Database Connectivity
OEDL	OMF EDL
OMF	cOntrol and Management Framework
OMF	Orbit Management Framework
OML	ORBIT Measurement Library
OS	Operating System
OTA	Over The Air
P2P	Point to Point
PSO	Particle Swarm Optimization
PTZ	Pan Tilt Zoom
RC	Resource Controller
RC	Resource Controller
RE	Requirement Engineering
REST	Representational state transfer
RIA	Research and Innovation Action
ROS	Robot Operating System
ROV	Remotely Operated Vehicle
RPA	Remotely Piloted Aircraft
RPAS	Remotely Piloted Aircraft System
RPS	Remotely Piloted Station
RSpec	SFA Resource Specification
SaaS	Software as a Service
SAML	Security Assertion Markup Language
SFA	Slice-based Federation Architecture
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SQL	Simple Query Language
SSO	Single-Sign-On
SVN	Apache Subversion
TM	Testbed Manager
TMS	Testbed Manager Suite
TP	Testbed Proxy
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
UI	User Interface
UML	Unified Modelling Language
USV	Unmanned Surface Vehicle

UUV	Unmanned Underwater Vehicle
UxV	Unmanned aerial/ground/surface/underwater Vehicle
VE	Visualization Engine
VT	Vehicular Testbed
VT	Visualization Tool
WCS	Web Coverage Service
WFS	Web Feature Service
WMS	Web Map Service
WPS	Web Processing Service
WSDL	Web Services Description Language
XMPP	Extensible Messaging and Presence Protocol

Table 40: Common abbreviations

The following table gives the notations used in the RAWFIE documents and deliverables.

Notation	Description
D*X.Y*	Deliverable *X.Y* from the DoW
MS*X*	Milestone *X* from the DoW
WP*X*	Work package *X* from the DoW
OC*X*	Open Call *X*
A*X.Y*	Activity number *Y* in Phase *X*
DL*X.Y*	Deadline number *Y* in Phase *X*
M*X*	Project month number *X*

Table 41: Notation

Annex C Glossary

The RAWFIE glossary is made of generic terms, contributed by all partners.

A

Accounting Service

RAWFIE component. Component that keeps track of resources usage by individual users.

Aggregate Manager

SFA term. The Aggregate Manager API is the interface by which experimenters discover, reserve and control resources at resource providers.

Avro

Apache Avro: a remote procedure call and data serialization framework

B

Booking Service

RAWFIE component. The Booking Service manages bookings of resources by registering data to appropriate database tables.

Booking Tool

RAWFIE component. The Booking tool will provide the appropriate Web UI interface for the experimenter to discover available resources and reserve them for a specified period.

C

Common Testbed Interface

RAWFIE component. The set of software and hardware functionalities each Testbed provider should ensure, for the communication with Middle Tier software components of RAWFIE, therefore for the integration with the RAWFIE platform

Component

A reusable entity that provides a set of functionalities (or data) semantically related. A component may encapsulate one or more modules (see definition) and should provide a well defined API for interaction

D

Data Analysis Engine

RAWFIE component. The Data Analysis Engine enables the execution of data processing jobs by sending requests to a processing engine which will perform the computations specified when the analytical task was defined through the Data Analysis Tool to be transmitted to the processing engine for execution.

Data Analysis Tool

RAWFIE component. The Data Analysis Tool enables the user to browse available data sources for subject to analytical treatment as well as previous analysis tasks' outcomes.

E

EDL Compiler & Validator

RAWFIE component. The EDL validator will be responsible for performing syntactic and semantic analysis on the provided EDL scripts.

Experiment Authoring Tool

RAWFIE component. This component is actually a collection of tools for defining experiments and authoring EDL scripts through RAWFIE web portal. It will provide features to handle resource requirements/configuration, location/topology information, task description etc.

Experiment Controller

RAWFIE component. The Experiment Controller is a service placed in the Middle tier and is responsible to monitor the smooth execution of each experiment. The main task of the experiment controller is the monitoring of the experiment execution while acting as 'broker' between the experimenter and the resources.

Experiment Monitoring Tool

RAWFIE component. Shows the status of experiments and of the resources used by experiments.

Experiment Validation Service

RAWFIE component. The Experiment Validation Service will be responsible to validate every experiment as far as execution issues concern.

M

Master Data Repository

RAWFIE component. Repository that stores all main entities that are needed in the RAWFIE platforms. Is an SQL-database

Measurements Repository

RAWFIE component. Stores the raw measurements from the experiments

Message Bus

Also known as Message Oriented Middleware. A message bus is supports sending and receiving messages between distributed systems. It is used in RAWFIE across all tiers to enable asynchronous, event-based messaging between heterogeneous components. Implements the Publish/Subscribe paradigm.

Module

A set of code packages within one software product that provides a special functionality

Monitoring Manager

RAWFIE component. Monitors the status of the testbed and the UxVs belonging to it, at functional level, e.g. the 'health of the devices' and current activity.

N

Network Controller

Manages the network connections and the switching between different technologies in the testbed in order to offer seamless connectivity in the operations of the system.

L

Launching Service

RAWFIE component. The Launching Service is responsible for handling requests for starting or cancellation of experiments.

R

Resource Controller

RAWFIE component. The Resource Controller can be considered as a cloud robot and automation system and ensures the safe and accurate guidance of the UxVs.

Resource Explorer Tool

RAWFIE component. The experimenter can discover and select available testbeds as well as resources/UxVs inside a testbed with this tool. Administrators can manage the data.

Results Repository

RAWFIE component. Stores the results of data analyses.

Resource Specification (RSpec)

SFA term. This is the means that the SFA uses for describing resources, resource requests, and reservations (declaring which resources a user wants on each Aggregate).

S

Schema Registry

A schema registry is a central service where data schemas are uploaded to. As an added benefit each schema has versions with it can convert allowable formats to other ones (e.g.: float to double) It maintains schemas for the data transferred and keeps revisions to be able to upgrade the definitions as with the simple field conversion. Used in RAWFIE for messages on the message bus.

Service

A component that is running in the system, providing specific functionalities and accessible via a well known interface.

Slice Federation Architecture (SFA)

SFA is the de facto standard for testbed federation and is a secure, distributed and scalable narrow waist of functionality for federating heterogeneous testbeds.

Subsystem

A collection of components providing a subset of the system functionalities.

System

A collection of subsystems and/or individual components representing the provided software solution as a whole.

System Monitoring Service

RAWFIE component. Checks readiness of main components and ensure that all critical software modules will perform at optimum levels. Predefined notification are triggered whenever the corresponding conditions are met, or whenever thresholds are reached

System Monitoring Tool

RAWFIE component. Shows the status and the readiness of the various RAWFIE services and testbed

T

Testbed

A testbed is a platform for conducting rigorous, transparent, and replicable testing of scientific theories, computational tools, and new technologies.

In the context of RAWFIE, a testbed or testbed facility is a physical building or area where UxVs can move around to execute some experiments. In addition, the UxVs are stored in or near the testbed.

Testbeds Directory Service

RAWFIE component. Represents a registry service of the middleware tier where all the integrated testbeds and resources accessible from the federated facilities are listed, belonging to the RAWFIE federation.

Testbed Manager

RAWFIE component. Contains accumulated information about the UxVs resources and the experiments of each one of the federation testbeds.

Tool

A GUI implementation to do a special thing, e.g. the “Resource Explorer tool” to search for a resource

U

Users & Rights Repository

RAWFIE component. Management of users and their roles. Is a directory services (LDAP).

Users & Rights Service

RAWFIE component. Manages all the users, roles and rights in the system.

UxV

The generic term for unmanned vehicle. In RAWFIE, it can be either:

USV Unmanned Surface vehicle.

UAV Unmanned Aerial vehicle.

UGV Unmanned Ground vehicle.

UUV Unmanned Underwater vehicle.

UxV Navigation Tool

RAWFIE component. This component will provide to the user the ability to (near) real-time remotely navigate a squad of UxVs.

UxV node

RAWFIE component. A single UxV node. The UxV is a complete mobile system that interacts with the other Testbed entities. It can be remotely controlled or able to act and move autonomously.

V

Visualisation Engine

RAWFIE component. Used for providing the necessary information to the Visualisation tool, to communicate with the other components, to handle geospatial data, to retrieve data for experiments from the database, to load and store user settings and to forward them to the visualisation tool.

Visualisation Tool

RAWFIE component. Visualisation of an ongoing experiment as well as visualisation of experiments that are already finished

W

Web Portal

RAWFIE component. The central user interface that provides access to most of the RAWFIE tools/services and available documentation.

Wiki Tool

RAWFIE component. Provides documentation and tutorials to the users of the platform.

References

[1] *Reference Model for Service Oriented Architecture 1.0,Committee Specification 1*, 2 August 2006 - http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm

[2] *Apache Avro* - http://avro.apache.org/

[3] *Avro RPC Quick* Start - https://github.com/phunt/avro-rpc-quickstart

[4] *Apache Kafka homepage* - http://kafka.apache.org/

[5] *A Relational Database Overview*, oracle.com - https://docs.oracle.com/javase/tutorial/jdbc/overview/database.html

[6] *PostgreSQL homepage*: http://www.postgresql.org/

[7] *Directory Services (LDAP)*, oracle.com - http://docs.oracle.com/cd/A87860_01/doc/ois.817/a83729/adois09.htm

[8] *OpenDJ hompage* - http://opendj.forgerock.org/

[9] *PostgreSQL-Wiki: Replication, Clustering, and Connection Pooling*,- https://wiki.postgresql.org/wiki/Replication,_Clustering,_and_Connection_Pooling

[10] *OpenDJ Administration Guide: Chapter 9: Managing Data Replication* - https://backstage.forgerock.com/#!/docs/opendj/2.6/admin-guide/chap-replication

[11] *GENI Aggregate Manager API Version 3* - http://groups.geni.net/geni/wiki/GAPI_AM_API_V3

[12] *GENI Aggregate Manager API* - http://groups.geni.net/geni/wiki/GAPI_AM_API

[13] *Hadoop WebHDFS REST API* - http://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-hdfs/WebHDFS.html

[14] *Hadoop* - http://hadoop.apache.org/

[15] *Fed4FIRE: D5.1* - http://www.fed4fire.eu/fileadmin/documents/public_deliverables/D5-1_Fed4FIRE_Detailed_specifications_for_first_cycle_ready.pdf

[16] *Kafka Security* - http://docs.confluent.io/2.0.0/kafka/security.html

[17] *Graphite* - http://graphite.wikidot.com/

[18] Whisper http://graphite.readthedocs.io/en/latest/whisper.html

[19] *Messaging Bridge*, from *Enterprise Integration Patterns*, Gregor Hohpe and Bobby Woolf, Addison-Wesley 2013 http://www.enterpriseintegrationpatterns.com/patterns/messaging/MessagingBridge.html