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	D4.7 - High Level Design and Specification of RAWFIE Architecture (3rd version)		
Confidentiality	PU	Deliverable type[1]	R
Deliverable File	D4.7	Date	30.04.2016
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	Aberon	vasil.kumanov@aberon.bg
Kakia Panagidi	UoA	kakiap@di.uoa.gr
Giovanni Tusa	IES Solutions	g.tusa@iessolutions.eu
Philippe Dallemagne	CSEM	Philippe.Dallemagne@csem.ch
Damien Piguet	CSEM	damien.piguet@csem.ch
Kiriakos Georgouleas	HAI	Georgouleas.Kiriakos@haicorp.com
Nikolaos Priggouris	HAI	PRIGGOURIS.Nikolaos@haicorp.com

REVIEWERS TABLE

Name	Company	E-Mail
Philippe Dallemagne	CSEM	Philippe.Dallemagne@csem.ch
Giovanni Tusa	IES Solutions	g.tusa@iessolutions.eu

DISTRIBUTION

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

CHANGE HISTORY

Version	Date	Reason for Change	Pages/Sections Affected
0.1	2017-03-08	Start editing 2nd version of the architecture	all
0.2	2017-04-05	Highlighted needed changes for Real time contains, and SFA	Section 3
0.3	2017-04-06	Removed content that did not change	all
0.4	2017-04-10	Updated SFA integration description	Section 3.6
0.5	2017-04-10	Updated descriptions of Resource Explorer Tool, Booking Service, Testbed Manager and Monitoring Manager	Section 4
0.6	2017-04-20	Updated Architectural Overview	Section 3
0.7	2017-04-21	Added descriptions for the Proximity Component	Section 4.2
0.8	2017-04-26	Updated SFA/SAMANT description	Section 3.6
0.9	2017-04-27	Updated Real-time constrains	Section 3.2
0.10	2017-04-28	Updated Message Bus	Section 3.8
0.11	2017-04-30	Version ready for first review	all
0.12	2017-05-01	Review	all
0.13	2017-05-03	Review 2	all
1.0	2017-05-09	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 third 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 design choices and improvements brought during the third design round .

This is the last deliverable of the RAWFIE High Level architecture documents' series. Further changes on high level architectural elements, if any, will be reported in the last deliverable of the RAWFIE components' design and specification documents' series, D4.8.

Keywords:
architecture, components, interactions

Part II: Table of Contents

List of Figures

List of Tables

Part III: Executive Summary

This deliverable describes the final high-level architecture of RAWFIE, with a special attention to the changes that the consortium brought to it during the third design round. It is the third release of this deliverable and, as such, it aims at providing the final architectural overview together with the main changes / updates, avoiding to repeat all concepts and information provided in the previous versions.

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, changed and new components are described and their relation to other components are listed.

The requirement mapping and the state of the art analysis of D4.1 is still valid and not repeated in the deliverable.

Part IV: Main Section

1 Introduction

1.1 Scope and overview of D4.7

D4.7 finalises the RAWFIE architecture that has been elaborated in three phases, with D4.1 and D4.4 as initial and intermediate results. It reflects all the necessary changes that have been done or need to be done based on the experience of the first and second implementation period and the decisions of the third design round.

The section "3 Architectures Overview" of D4.4 was updated with information in the present deliverable. The section "4 Components" of D4.4 is in many parts still valid and it now describes only the components that have been added or changed. Also the section "5 Requirement mapping" of D4.4 is still valid and will not be repeated.

1.2 Relation to other deliverables

D4.7 is an update of D4.4 (and D4.1) so it will share many of the contents with them.

The architectural definition was updated using the detailed updated requirement analysis that was given in D3.3 (which also reflects the experiences of the first and second implementation period.

D4.8 will provide updated detailed components descriptions, based on the architecture defined in D4.7. Therefore, this deliverable aims at providing information about the components and their interfaces only at a high level.

D4.9 will provide information on verification and validation plans and scenarios for the architecture.

2 Overview of changes

This chapter lists the most important changes made in comparison to D4.4:

- Section "Architectures Overview" updated, especially:
  - Interoperability whit SFA
  - Message Bus topology
- Section "Components" only contains the components that have been changed or added

  - Changed: Resource Explorer Tool, Booking Service, Testbed Manager and Monitoring Manager
  - Added: UxV – Proximity Component
- Annex "A: Relevant technologies" (D4.4) and "State of the Art" (D4.1) still valid (no new technologies introduced) and is not repeated.

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 RAWFIE high level architecture after the 3rd design iteration: it provides updates and enhancements to the one presented in the D4.4, 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 second prototype implementation (see WP5). The main design principles are described in the following sub-sections.

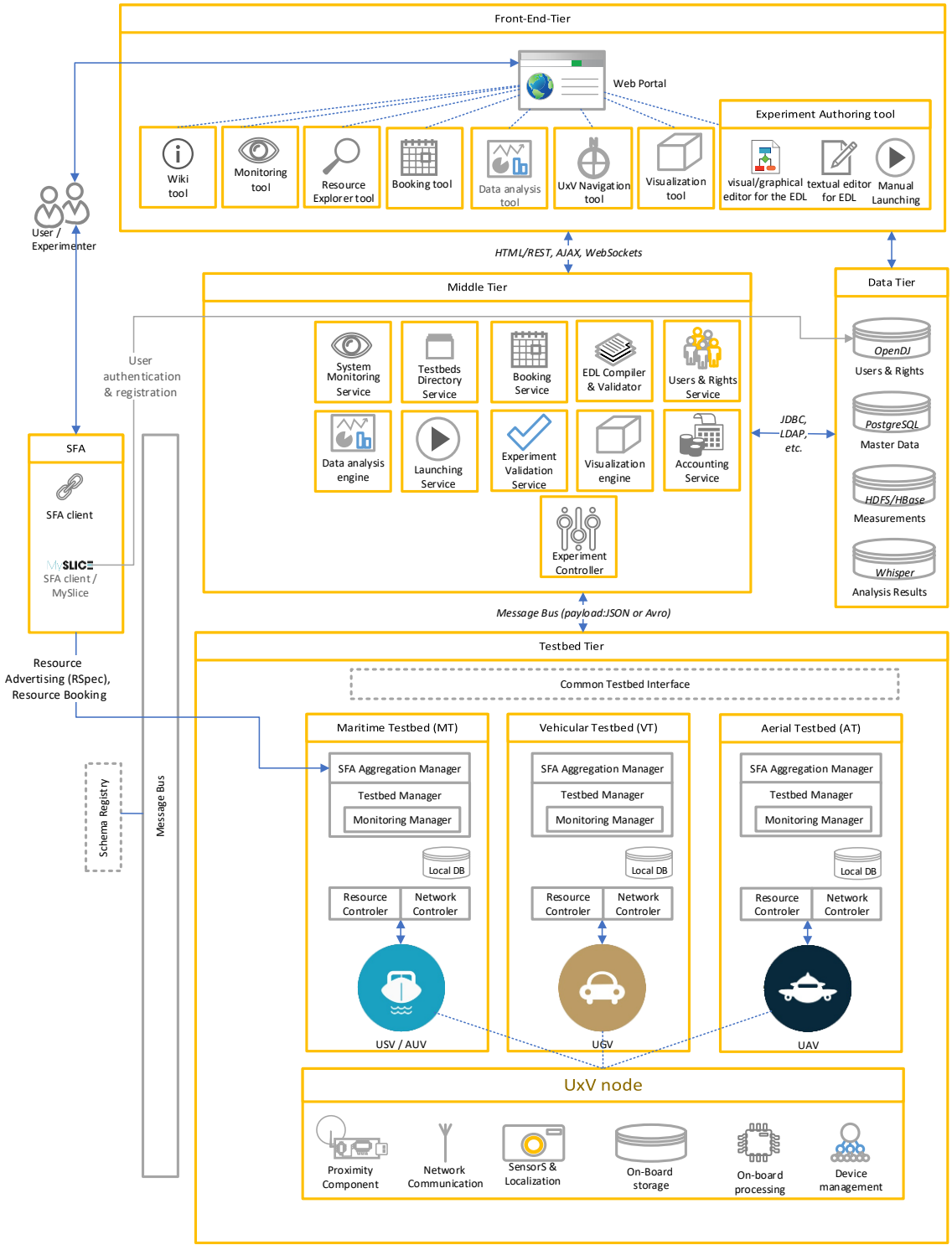

Figure 1 – RAWFIE architecture

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 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, RAWFIE uses a message-oriented middleware (via a Message Bus) where suitable, which offers a convenient communication model providing distribution, replication, reliability, availability, redundancy, backup, consistency, decoupling of components 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. between Middle Tier and Testbed Tier). It can be used for asynchronous notifications and asynchronous method calls / response handling. For example, 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 Engine) as well as for information (including events) that generally addresses multiple components. See also section 3.8 for more information of the communication through the message bus.

Chapter 4 (also of D4.4) gives more information about the components highlighted in Figure 1. Since the present deliverable mostly focuses on providing information about changes with respect to the previous versions, the reader is invited to refer to the corresponding chapters of D4.4 for complete information about all components. As usual, a more detailed and up to date description of interfaces and interactions between the various components will be given in D4.8.

3.2 Real-time constraints and impacts in the architecture

3.2.1 Rationale

The RAWFIE application engages highly dynamic vehicles, such as aerial vehicles or drones (UAVs). This 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 vehicle itself, the experimenter, the resource manager and possibly many others).

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

The consortium continuously identifies and elaborates on time constraints and real time requirements for several types of devices that may be involved in the RAWFIE experiments, in particular 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 also evaluates how these constraints may affect the RAWFIE architecture and the chosen technologies.

3.2.3 Techniques

In a system like RAWFIE, dealing with latency and other real-time operations implies checking for the properties of the components of the system and their behaviour, independently or once integrated as a system.

Therefore, the RAWFIE system is a typical Cyber-Physical Systems (CPS), since it involves concurrent closed loop process for, e.g. controlling the UxVs. As such, its architecture and its components can benefit from the work done in this domain. Most of the work done for process control in critical or real-time applications, which includes a loop involving sensing, transmission, processing and actuation (in the case of RAWFIE, this corresponds to the UxV control), relates to Cyber-Physical systems. CPS covers many requirements, such as timeliness, efficient resource usage, adaptation to changing environments and collective organisation. Like in other CPS systems, RAWFIE envisages the presence of components (e.g Resource Controller implemented by consortium partners or similar proprietary controlling components provided by testbed or UxVs owners), needed to transmit the data collected in the field, analyse the data, collectively take decisions and act on the real world by sending back commands to actuators, hence closing the control loop correctly in terms of computation, time and location.

Many of such systems aim at dealing with real-time constraints, both on the task execution, such as the completion of tasks before specific deadlines, and on the communication, such as bounded transmission of data (from sensors, to actuators, etc.), including over wireless links comprising

multiples hops [29]. Task scheduling can use the traditional scheduling approaches such as Earliest Deadline First or Time-Triggered. The baseline is that communication and task are closely related and the scheduling can be expressed in communication terms [25]. Typical approaches aim at providing reliability, adaptivity and real-time by using a distributed real-time protocol [27] and by decoupling the communication from the local application [26], while keeping temporal interdependencies through communication interactions.

The mobility of the vehicles is intrinsic to RAWFIE. Mobility increases the link variability and drops of wireless links that are typically used for reaching mobile vehicles. As an consequence, mobility calls for appropriate techniques, such as handover (found in GSM, LTE or IEEE 802.11), which is a convenient feature but that introduces additional unpredictable delays. The RAWFIE application may benefit of (and will likely require) end-to-end real-time QoS for UAV, which [28] provides. These results can be used in RAWFIE either directly if the existing technology is adopted or indirectly if the technique is adapted to the case of RAWFIE and implemented for it.

Constraints identification. 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 issuing 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. However, the design of RAWFIE platform reduced the time constraints that could be imposed by UxV manufacturers. One of the basic principle of RAWFIE is that the device executes commands coming from Resource Controller sequentially, i.e. while a device is in operating mode then either executes a command from Resource Controller or waits at the same waypoint. Other emergency cases may need the device to return to the last waypoint/to the initial point of experiment or the manual operation and safety return "home" from the man-on-the-field (testbed operator). Therefore UxV manufactures haven't rose any time constraints that apply during the operation of the devices.

Constraints specification. 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.

All identified constraints must be translated into time-based conditions, e.g. including 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).

Note that the fallback mechanisms may be implemented in components other than the detector of the condition. Hence, the architecture should allow for notification mechanisms across the RAWFIE components, at least those participating to a given experiment and across those of a given testbed. This also requires that the interfaces of the services provision for such notification parameterisation (e.g. parameters or subscription to events).

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.

The verification of time constraints during the execution of the component services can be implemented using timers, watchdogs, still alive notifications or periodic status exchanges. Sequence numbers and time-stamping of the information, including the information collected by on-board sensors, is a primary and solid basis for checking the order, causality and the time consistency of events and actions [25].

This will allow for the implementation of an event abstraction in RAWFIE.

Time constraints in the tools and services.

Time constraints on the tools or components related to the user interactions are limited to the automatic log-out due to inactivity, that is triggered after 10 minutes of absence of any request received by the server from a given user. This impacts:

- Web Portal
- Wiki Tool
- System Mentoring Tool - IcingaWeb2 page

Some calls of services in the RAWFIE architecture (such as the database) are also constrained by temporal parameters: The elapsed timer (timeout) means that the call was not successful

- REST service calls over HTTP have a response timeout of 1minute.
- SQL connection timeout: 1 minute
- LDAP connection timeout: 1 minute

The actions if a timeout occurs are returning error values in the service calls and showing error pages in the front-end tools.

EDL support for time constraints

The Experiment Description Language allows the experimenters for specifying a few time constraints in the experiment scenarios. It is the case for planning and guidance management. The management of waypoints is available through the time perspective. The parallel execution is not implemented, since this feature has not yet been requested. The execution at pre-defined intervals, like in the SMIL approach, is possible by the invocation of algorithms at specific time intervals. A specific example is given below:

```
Node
            ID rawfie.mst.auv-1
            Route[
            WP<0, +0.0, +0.0, +0.0>
                    WP<1, +2.0, +1.0, +0.0>
                    WP<5, +23.0, +25.0, +0.0>
                    WP<8, -0.0, +12.0, +0.0>
                    WP<10, -12.0, +15.0, +0.0>
                    WP<15, +11.0, -17.0, +0.0>
            ]
            Sensor[Time 7 Name Temperature set Activated]
            DataManagement[Time 8 Algorithm dataReporting(status = 1)]
            DataManagement[Time 15 Algorithm dataReporting(status = 0)]
            Sensor[Time 15 Name Temperature set Deactivated]
~Node
```

The above example exemplifies how to manage the waypoints for a node. The DataManagement commands are examples of the execution at specific intervals.

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] or Hbase API for accessing data stored in the Hadoop Distributed File System (HDFS) [14] and Hbase [30].

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 data exchange with the other components. The component descriptions mention these data exchanges as input and output communication. Since the present deliverable mostly focuses on providing information about changes with respect to the previous versions, the reader is invited to refer to the corresponding chapter 4 of D4.4 for complete information about all components.

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 realised 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]) was chosen for this purpose. In addition, the NoSQL system HBase [30] (running on top of HDFS) was tested to better manage the data sets. It will be further integrated the 3rd implementation iteration. Also Kafka Connect for Hbase [31] will be used to store the messages from the message bus to Hbase.
- *Analysis Results Repository:* uses a dedicated 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 compatibility

RAWFIE is a FIRE project. Therefore, it needs to provide compatibility with other FIRE facilities as well as adheres to the concepts and services prescribed by the SFA architecture. We chose the approach of a transparent integration of SFA Aggregate Manager as described in this section. This will allow RAWFIE platform to be part of other FIRE enabled federations. In order to achieve the required level of SFA compatibility the following actions are foreseen:

1. Implementation of a modified version of SFA Aggregate Manager (AM) at testbed level that will support Geni API V3 [12] and will be able:
    - To support/handle the resource specification (RSpec [20])) needed to list, describe and possibly advertise testbed local resources which are whole UxV systems.
    - To handle reservation of UxV resources at testbed level, a process which significantly differs from resource reservation in other FIRE facilities since it is not an one step process (reservations may need to be in pending state waiting to be approved by a testbed authority).
2. Modification of the MySlice interface responsible for providing access to SFA facilities in order to support the RAWFIE authorization model that prescribes users with certain roles as well as the use of LDAP for storing this info.

SFA compatibility will be achieved in a mostly transparent way ensuring minimal effects to the core RAWFIE modules. Details about the SAMANT project and the integration procedure are given in 3.6.1.

3.6.1 Developments from the SAMANT project

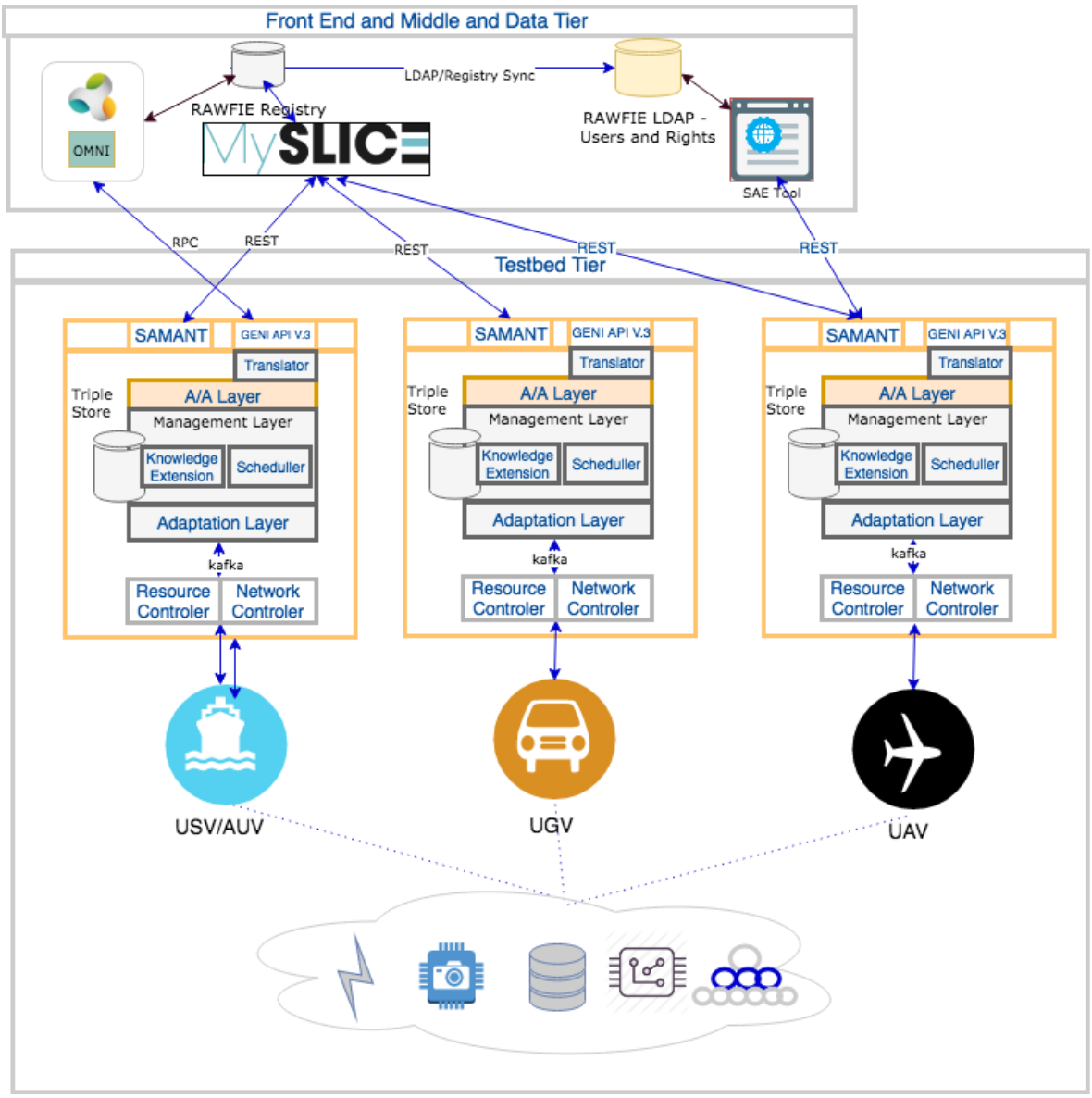

Figure 2: RAWFIE Updated Reference Architecture

The goal of the SAMANT architecture is to provide the appropriate tools and software enhancements at the RAWFIE testbed or federation level, to support functionalities related to resource discovery, booking and reservation, provisioning and release by experimenters, while addressing at the same time the corresponding authentication and authorization issues at the RAWFIE federation. Moreover, it is sufficiently versatile in terms of communication interfaces

and interoperability with external/3rd party tools as well as other testbeds, since it supports the Slice-based Federation Architecture (SFA) [21] protocol for this purpose. It uses XML schemas (GENI RSpecs v3) [20] for describing resources life cycle. SAMANT aims to exploit the benefits of a semantic web approach at the upper tiers of the federation solution, using data semantics, knowledge inference and reasoning. In the context of the SAMANT project, a semantic information model will be adopted for representing and linking RAWFIE resources, by re-using and extending existing standard semantic models, for the federated RAWFIE environment. Moreover adopting the use of a semantic registry repository for testbeds and resources will enable experimenters to find and book resources more easily. For this purpose, SAMANT will adopt and extend the Open-Multinet (OMN) ontology suite [22] [23], the semantic information model for federated infrastructures management. Therefore, based on the framework described in [24], the implementation of an Aggregate Manager will allow the parallel use of ontology-based RSpecs and GENIv3 RSpecs, compatible with existing SFA–based testbed federations, while will support the various functionalities related to lifecycle management of RAWFIE resources and testbed administration requirements.

Figure 2 provides an updated high-level view of the parts of the SAMAMT reference architecture that will support the resource reservation mechanisms of RAWFIE testbeds. The core software module is the SAMANT Aggregate Manager (SAM) which will be based on the (SFA) NITOS[4] Aggregate Manager design principles. One instance of the SAM module will be deployed in each testbed in order to handle the reservation process of the respective resources. The SAM will maintain semantic descriptions of testbed resources to a triple-store database. The SAM will expose two different APIs – a REST API and an XML-RPC API. As it is described in details in the following sections, the REST API exposes functionality based on requirements of all actors involved, allowing the management of the testbed resources from an administrative perspective but also from a client/experimenter perspective. Testbed administrators can add, update, remove, and perform semantic-based queries for the resources of the specific testbed. Experimenters are allowed to utilize the REST API throughout the lifecycle of an experiment in order to query, reserve and release resources. The REST API is expected to be utilized by MySlice[5] tool that provides a web interface and a programmable API from which users can register to the RAWFIE federation and manage aspects of their experiments' lifecycle. The XML-RPC API is GENI V3 compatible thus allowing RAWFIE testbeds to become members of

[4] https://github.com/NitLab/Central_Broker
[5] https://www.myslice.info

SFA federations. This API is expected and able to be utilized by any kind of GENI V3 compliant tools.

3.6.1.1 REST – API

The REST API is tailored to support the discovery, reservation and release functionality for RAWFIE resources. It leverages the OMN-based resource descriptions stored in the local triplets repository (RDF Triples) to provide the users/experimenters with semantically enriched information regarding the resources managed by the respective testbed. From their side, the users are then able to allocate and provision resources that correspond to their experiments' specifications, as well as release these resources when no longer used. An X.509 authentication system is facilitated to authorize the aforementioned actions.

Complementary to this functionality, this API will expose the essential administrative management methods; namely, RAWFIE resource description creation, update and deletion will be supported.

The following table provides a short summary of these methods:

Method		**Description**
experimenter	list_resources	Return information about available resources or resources allocated to a slice.
	describe	Return information about resources allocated to a slice.
	allocate (optional)	Allocate resources as described in a request RSpec argument to a slice with the named URN. On success, one or more slivers are allocated, containing resources satisfying the request, and assigned to the given slice. Allocated slivers are held for an aggregate-determined period.
	renew	Request that the named slivers be renewed, with their expiration extended.
	status	Gets the status of a sliver or slivers belonging to a single slice at the given aggregate.
	shutdown_sliver	Perform an emergency shutdown of a sliver. This operation is intended for administrative use
admin	create	Create a resource description on the respective testbed.
	update	Update a resource description on the respective testbed.
	remove	Remove an existing resource description on the respective testbed.

The MySlice framework will be adapted in order to communicate with each testbed's REST API thus providing the users with RAWFIE federation-wide results. This client-server communication, throughout each reservation's lifecycle, will be based on JSON serialized data.

3.6.1.2 XML-RPC API

The XML-RPC API exposed by SAM, enables interoperability with existing tools for experimenters (e.g. omni, jFed etc) and allows federation with other testbeds or testbed federations that confront to the SFA - GENI API v36 specification. One of the core innovations of the SAMANT project will be the semantic description of experimental resources (UxV domain ontology for OMN suite) and the development of the respective management mechanisms.

In order to be SFA compliant the API method calls and the respective semantic descriptions need to be translated into the respective SFA data models (i.e., RSpec v3). The following RSPEC types will be utilized within the context of SAMANT:

- **Advertisements** will be used to describe the resources available on a testbed. They contain information used by clients to choose resources (components).
- **Requests** will specify which resources a client is selecting from the testbed.
- **Manifests** will provide useful information about the slivers actually allocated a client/experimenter.

In addition, the existing SFA XML-RPC API methods will need to be adapted in order to support the usage of the SAMANT semantic triple store. The following table provides a short summary of the corresponding methods:

Method	Description
get_version	Return the version of the GENI Aggregate API supported by this aggregate.
list_resources	Return information about available resources or resources allocated to a slice.
describe	Return information about resources allocated to a slice.
allocate (optional)	Allocate resources as described in a request RSpec argument to a slice with the named URN. On success, one or more

[6] http://groups.geni.net/geni/wiki/GAPI_AM_API_V3

	slivers are allocated, containing resources satisfying the request, and assigned to the given slice. Allocated slivers are held for an aggregate-determined period.
renew	Request that the named slivers be renewed, with their expiration extended.
status	Get the status of a sliver or slivers belonging to a single slice at the given aggregate.
shutdown_sliver	Perform an emergency shutdown of a sliver. This operation is intended for administrative use

3.6.1.3 MySlice Tool and User registration

Based on the RAWFIE requirement to use LDAP as a source of users' authentication, the architecture of the MySlice frontend has evolved as follows:

- The web login module of MySlice will be modified to authenticate users against LDAP.
- At the first connection to MySlice, the LDAP information about the user will be inserted into MySlice database that will generate an X509 certificate embedded into an XML Credential describing the rights of the user.
- The communication toward the AMs of the testbeds remains the same using X509 certificates.
- A synchronization process will run periodically to disable or remove users form MySlice that were disabled or removed from LDAP.

The SFA model of resources reservation considers compute resources like servers or VMs with an automated approval. In the context of RAWFIE, the reservations of UxVs requires most of the time a manual approval. The reservation status will evolve from a binary state reserved or not, to a multiple steps status (as mentioned in the previous section). Therefore, the scheduler plugin in MySlice will be extended to follow this model.

3.6.2 Resource advertising

The RAWFIE system provides the Resource Explorer Tool for the users to browse the available testbeds and UxVs. SFA also provides a resource advertising mechanisms: The Aggregate Manager provides "RSpec" [20] for each UxV. Both databases (Maser Data repository am AM Triple Store) are kept in sync (see Resource Editing section). The process, which is depicted in Figure 3, is described and in the next paragraph.

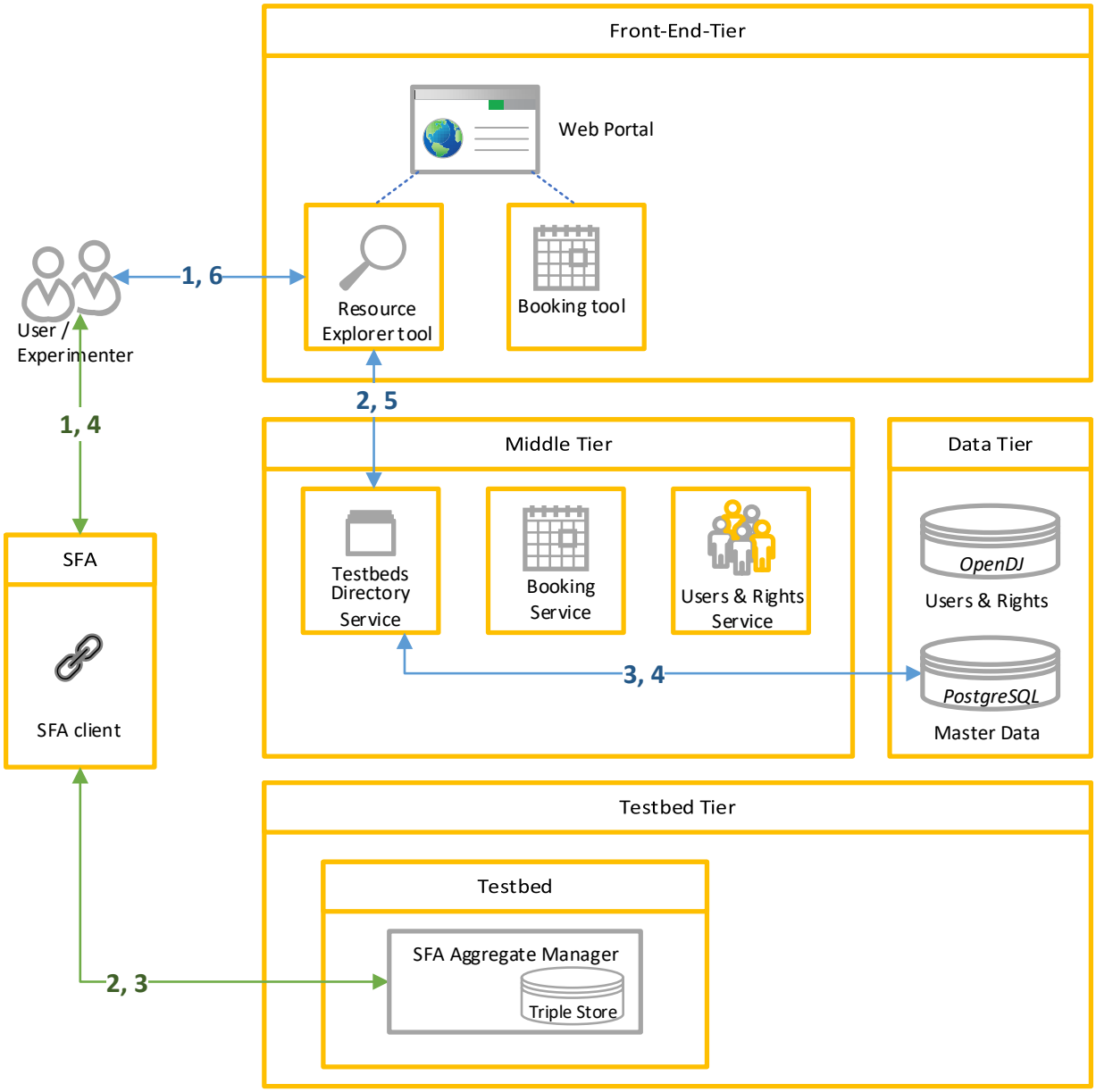

Figure 3 – Resource advertising

RAWFIE Resource advertising:

1. User open the Resource Explorer Tool
2. Resource Explorer Tool loads the data from the Testbed Directory Service
3. Testbed Directory Service queries the data from the Master Data repository
4. Master Data repository returns the data
5. Testbed Directory Service processes and filters the data and returns it to the Resource Explorer Tool
6. Resource Explorer Tool display the data to the user

SFA Resource advertising:

1. User opens its SFA client
2. SFA client requests the AM from the known testbed for RSpecs
3. AM loads the data an retuns it to the SFA client
4. SFA client show the data to the user

3.6.3 Resource editing

SFA Aggregate Manager provides an administration interface/tool that could be used to add/update/delete resources in the internal triple store database. However in the context of RAWFIE this administration tool will be completely disabled. Resource editing/update & deletion will be done locally at the testbed level and will be achieved via the Testbed Manager's UI. RAWFIE web portal will permit only visualization of available resources through the Testbed Directory Service.

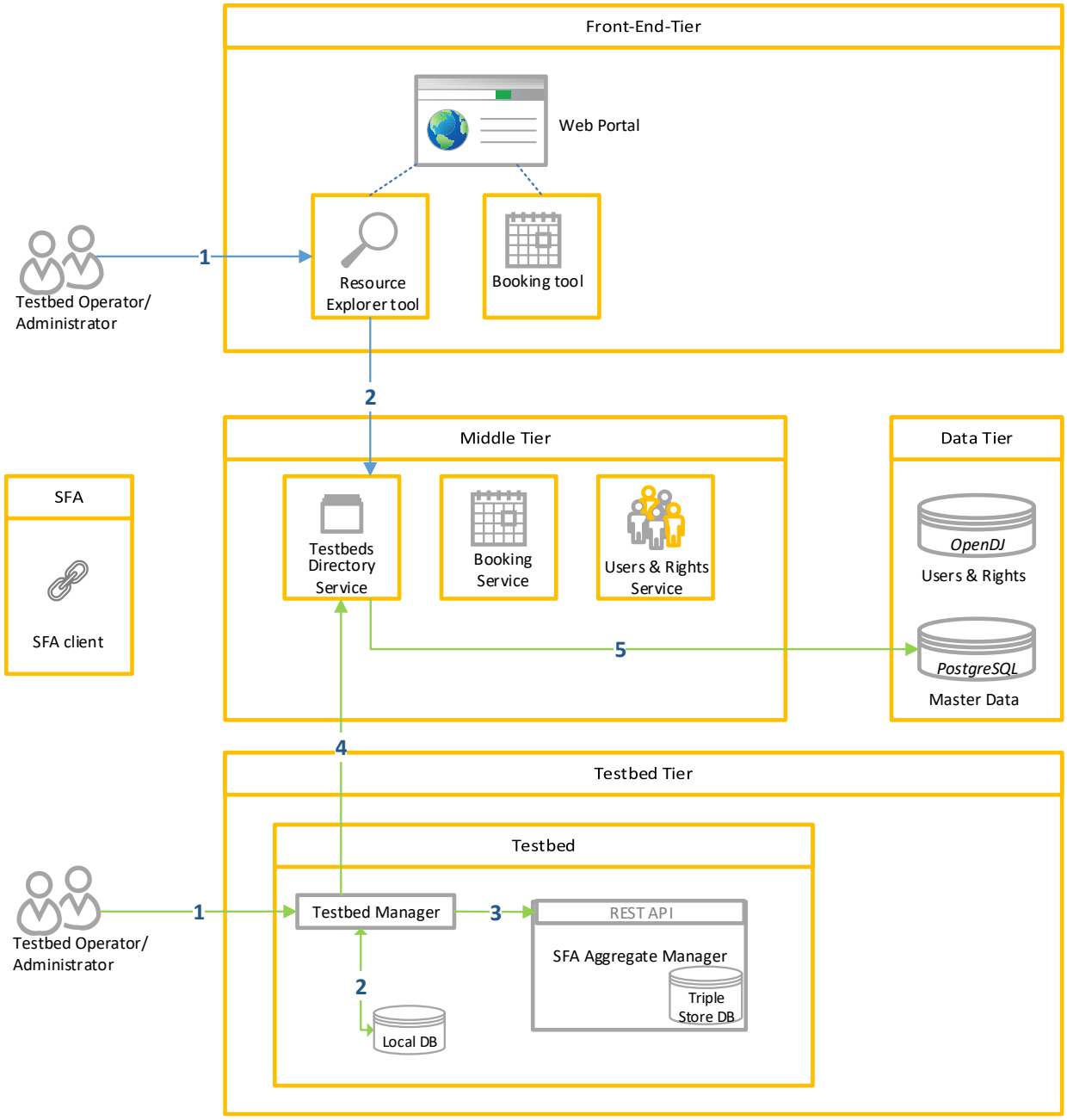

Figure 4 – Resource Edit/Update

RAWFIE Resource editing:

Actions (from Platform Level): No editing of resources from Resource Explorer Tool. Only view is possible by contacting Testbed Directory Service

Actions (from testbed Level):

1. Testbed Operator/administrator sends the changes of the resource to Testbed Manager (via Testbed Manager UI)
2. Local DB is updated
3. Request is forwarded to the AM to update its internal triple store DB
4. Request is forwarded to the Testbed Directory Service
5. Testbed Directory Service will apply the changes to the RAWFIE Master Data repository

SFA Resource editing:

Resource editing/addition/deletion via SFA is disabled in RAWFIE.

3.6.4 Resource booking

Reservation of resources is one of the aspects addressed by the SFA Aggregate Manager (AM). Therefore, the Booking Service will interact with AM in order to ensure that reservations between RAWFIE and the AM triple store database are synchronized. Figure 5 below depicts the extra interactions needed during a booking request.

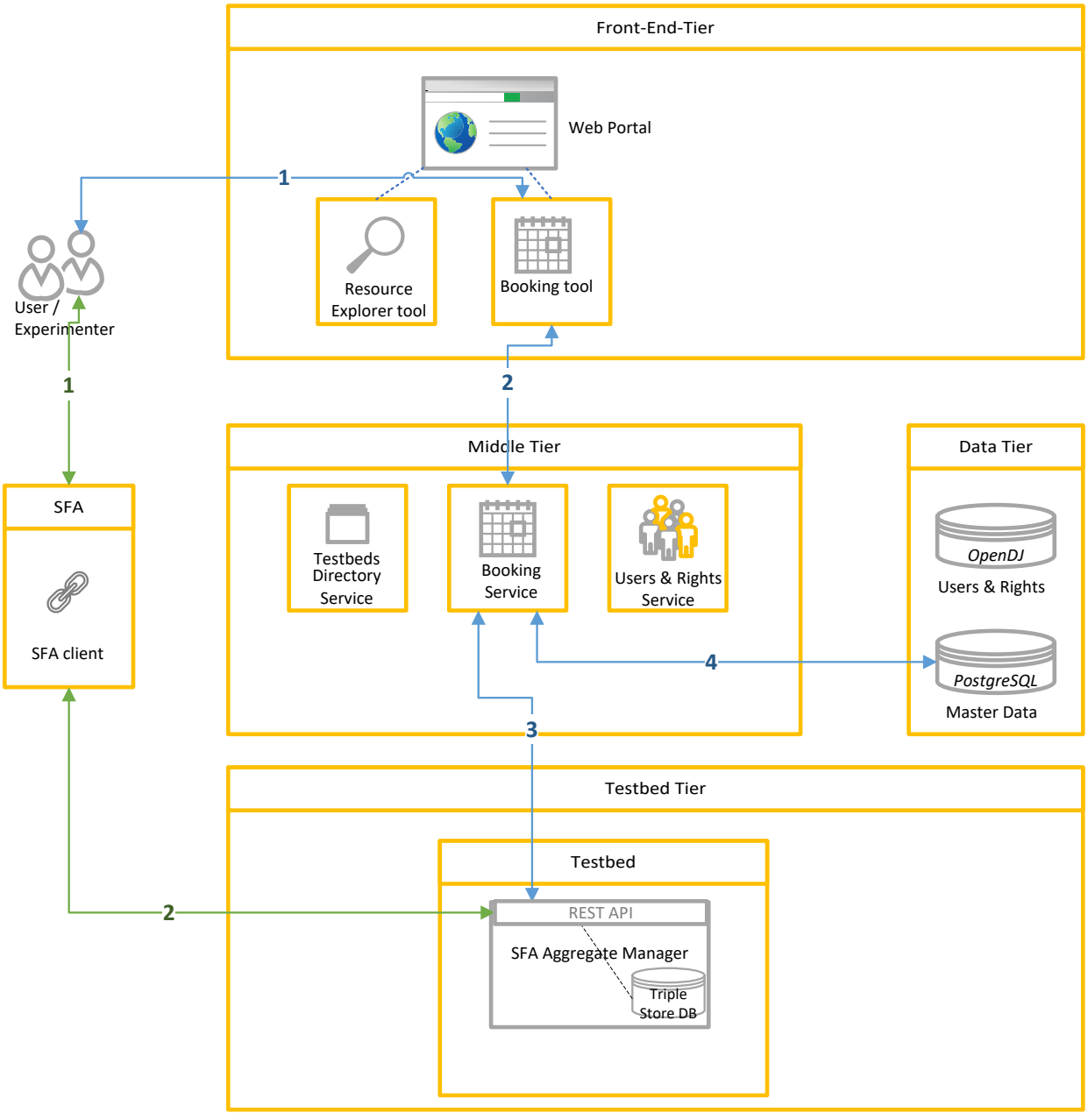

Figure 5 – Resource Booking

RAWFIE Resource booking:

1. User loads the Calendar view and initiates an update or create booking request
2. Booking Tool interacts with Booking Service to retrieve required information from the Master Data Repository
3. Booking Service communicates/synchronizes with AM to retrieve possible AM initiated booking that are not present in Master Data Repository as well as inform about the updated/new booking
4. Booking Service updates/creates the Bookings in the Master Data Repository and returns the data to the Booking Tool

All actions should be performed within a transaction.

SFA Resource booking:

1. User requests the bookings from the SFA Client
2. SFA Client reads the bookings from the AM (directly from internal Triple Store DB)

Note: Booking Service also periodically synchronizes with AM to ensure consistency of reservations with SFA

3.7 Testbed Tier

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

- message bus clients, that is, publishers and consumers of the Message Bus (software perspective),
- secured high bandwidth connection capabilities 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, like start, stop, cancel, etc. (Resource Controller reference implementation),
- Messages related to sending of status and health information for each testbed (Testbed Manager/Monitoring Manager reference implementation),
- Messages related to experiment data/measurements collected during an experiment that need to be analysed by the platform data analytics engine (UxV reference implementation),
- Location information and other dynamic characteristics of the various devices that can be used for coordination, monitoring and visualization purposes (UxV reference implementation).

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 based on the SFA standard and RAWFIE internal resource editing and booking protocol (see section 3.6).

Note that the testbed components implemented in RAWFIE comprise just a reference implementation that may be adopted by Testbed providers. Testbeds facilities belonging to the core RAWFIE project partners will probably use these components as well as the predefined structure. However, external testbeds, including the ones that will be integrated through the Open Calls, will decide on their own whether to use already existing software components by the project, or implement their own (the licence fees for reusing existing RAWFIE reference implementation will be part of the business model).

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 3.7.13.7.1 above)
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 supplyshould operate 24/7. In case of power outage, the tesbed and UxVs must have safety procedures to follow.,
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 D3.3

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 D3.3.

The hereafter-described constraints shall be met by the UxV hardware or by the on-board software suite (directly or 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. The UxVs shall be capable of keeping their positions (request station-keeping within a radius of 5m, except UxVs that have to move forward in order to turn where the radius should be 20 m for watercrafts and 100 m for small aircrafts)
7. UxVs shall periodically publish on-board storage usage, fuel usage, CPU usage,
8. UxVs shall be capable of processing sensor data in order to summarize large sensor data-sets,
9. UxVs shall receive and act upon RAWFIE command messages to control the UxV remotely (e.g. from the Front-end Tier)
10. The UxVs shall implement safe mode, for which it will immediately stop operating, stopping the mission, turning off the actuators. The sensors will continue operating.
11. The UxV dimensions should not exceed 1.0x1.0x1.0m in order to make it possible for a single person to deploy the vehicle on a pool or on the shore. Additionally, the USV weight should not exceed 18 kilograms in its basic configuration, without payload.
12. The operating voltages shall not exceed 48V as a safety measure to minimize damages to humans in case of defaults originated by improper handling of the USVs.

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.

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] was 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. 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.

3.8.1 Benchmarking and dimensioning Message Bus in RAWFIE

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.

During the second development cycle, three different scenarios, corresponding to three different deployment architectures and configurations of the Apache Kafka messages bus, were tested. In all the deploy scenarios the performance of the end-to-end communication using Apache Kafka is considered as one of the most important parameter to check.

The following main configuration and deployment principles were applied for reducing the latency in the publish/subscribe communication mechanism with 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, and it provides much shorter message bus queues dedicated to a particular UxV and much faster response times
- use of a local Message Bus (message broker) installation within each Testbed. This way, 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 (since each broker will just handle its own messages), and the local message bus system can be adjusted to the needs of the Testbed itself. Messages from each local broker can be anyway mirrored to a centralised Kafka broker deployed in the Cloud, so that Middle Tier components which need to access specific messages (e.g. logs or other data for experiments' control) will directly access the central message broker rather than each of the local ones.

The considered deploy scenarios are described in the following, while the actual tests results are reported and discussed in deliverable D6.3.

3.8.1.1 Scenario A- Single centralised Apache Kafka Broker

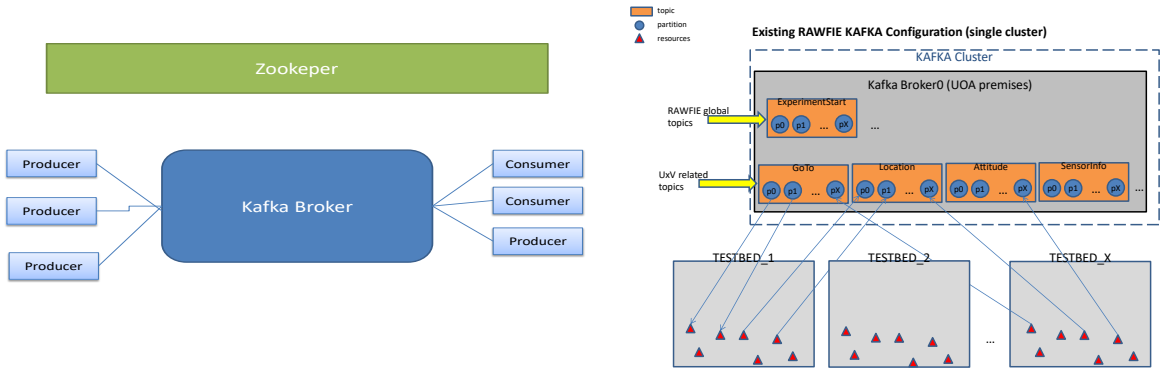

Figure 6 – Scenario A design

Scenario A is based on the initial Apache Kafka configuration infrastructure deployed during the first implementation cycle. A unique, centralised broker controls all the messages exchange between UxVs and RAWFIE services. This unique broker is installed in a single node and creates "a single node - a single broker cluster" as it is described in [32]. Like many publish-subscribe messaging systems, Kafka maintains feeds of messages in topics. Producers write data to topics and consumers read from topics. Since Kafka is a distributed system, topics are partitioned and replicated across multiple nodes. Messages are simply byte arrays and can be stored in object oriented format, like the JSON format that is used by RAWFIE. Each topic can be further discriminated by partitions. It is possible to attach a key to each message in which the producer guarantees that all messages with the same key will arrive to the same partition. A topic partition is the unit of parallelism in Kafka. On both the producer and the broker side, writes to different partitions can be done fully in parallel. So expensive operations such as compression

can utilize more hardware resources. On the consumer side, Kafka always gives a single partition's data to one consumer thread. Thus, the degree of parallelism in the consumer (within a consumer group) is bounded by the number of partitions being consumed. Therefore, in general, the more partitions there are in a Kafka cluster, the higher the throughput one can achieve. This facilitates sending and receiving messages during an experiment execution because all the devices can send their location by using the same message format to the same topic "Location" but in different partition; Resource Controller will read from the same topic the different messages coming from the devices. One of the main disadvantages is that commands for UxVs on one side, and UxVs' measurements on the other side, have to be sent to this single centralised Kafka node located outside of the Testbed, with the possible introduction of round trip delays. Since each UxV device sends to and receive from a different partition, the number of partitions may grow uncontrollably. This may not be scalable because needed partition number depends on the number of resources of all available testbeds and any reconfiguration (i.e. because a new Testbed or new UxVs are added) will affect all testbed providers. In general although it's possible to increase the number of partitions over time, we have to be careful if messages are produced with keys. When publishing a keyed message, Kafka deterministically maps the message to a partition based on the hash of the key. This provides a guarantee that messages with the same key are always routed to the same partition. This guarantee can be important for certain applications since messages within a partition are always delivered in order to the consumer. If the number of partitions changes, such a guarantee may no longer hold. To avoid this situation, a common practice is to over-partition a bit. Basically, it is advised to determine the number of partitions based on a future target throughput, say for one or two years later. Repartitioning can break the semantics in the application when keys are used, so it is advised to avoid it.

3.8.1.2 Scenario B – Multiple Apache Kafka Brokers with the same topics on each different Testbed

In B scenario we are moving from a single node-broker infrastructure to "Multiple Nodes-Multiple brokers cluster". At this installation, we setup multiple brokers of each node which are installed and connected to two zookeepers. Two zookeepers are created that are connected in order to distribute the load of the messages in a more generic cluster of devices.

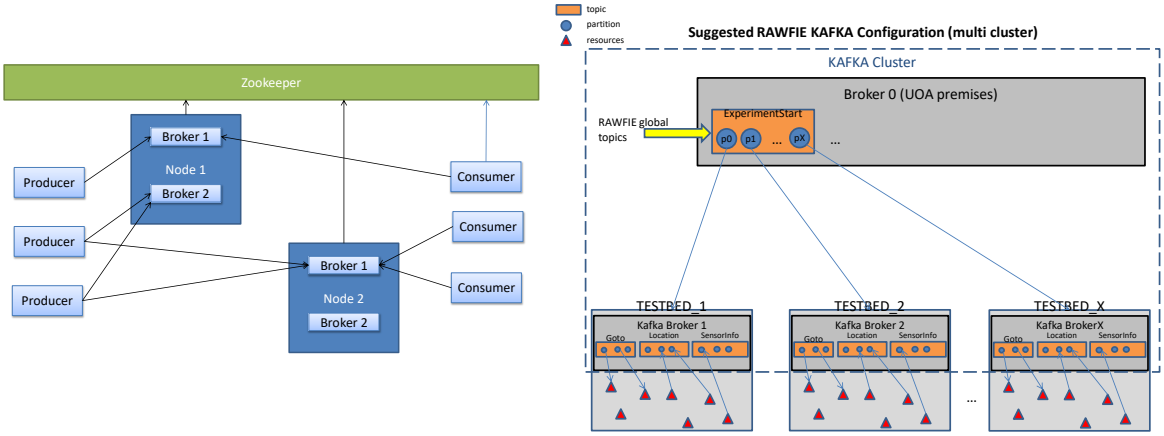

Figure 7 - Scenario B design

Every testbed is part of the Kafka cluster and topics on each testbed broker have the same names of topics in all the other brokers. All brokers are installed in the cluster and the UxV devices on each Testbed send the messages to topics with the same names. With this approach UxV commands avoid the excessive round trip delay of contacting the central Broker. UxV commands respect the boundary of the testbed they belong to and configuration/reconfiguration of partitions per topic is much easier to control since it relates only to specific testbed resources. The repartition problem is still present with this approach even if the messages are distributed in more efficient way.

3.8.1.3 Scenario C - Multiple Apache Kafka Brokers with different topics per testbed

In C scenario, we move from a single node-broker infrastructure to "Multiple Nodes- Multiple brokers clusters" but we go a step further. To overcome the case of repartitioning in case of new testbeds, topics are created with a different name per testbed (i.e prefixing the <testbed_id>_ in each case) only for the messages related with the control of devices. In this scenario, every testbed broker handles topics different from the others and all the topics are mirrored in the main cluster for redundancy purposes. The UxVs need to consider the testbed identifier in order to know where to send or from where to receive in each case (this can be part of their initial configuration when deployed in a testbed). Partitions of topics in other testbeds is not affected by adding or removing devices or even a whole testbed. Each testbed is a micro-system that controls and knows only the devices in it.

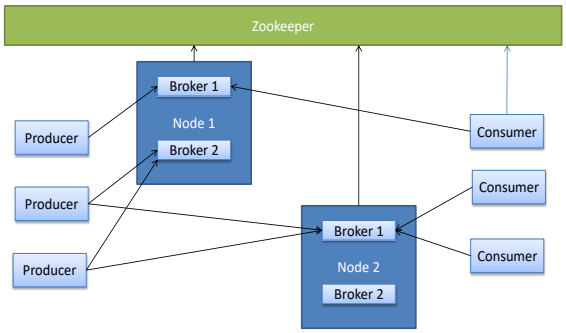

Figure 8 – Scenario C design

4 Components

This chapter describes the changes on components. For components that are not mentioned, the descriptions from D4.4 apply.

4.1 Front End Tier

4.1.1 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.
Provided functionalities	• Visualize Data from the "Testbed, Resources" directory • Provide ability to search and select available resources inside a testbed

Relation to other components	• Testbeds Directory Service (REST/RPC API)
	  ○ IN/OUT ↔ read testbed and UxV data
	• Booking tool (HTTP redirect)
	  ○ OUT → send selected resources

Changes	• Editing and updating of testbed and UxV removed. This is done via the Testbed Manager GUI

4.2 Middle Tier

4.2.1 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 • SFA Aggregate Manager - AM (REST API) o OUT → booking request to be added to AM internal DB o IN ← list of AM bookings to be synchronized in Master Datat Repository
Changes	• Interaction with SFA AM added

4.3 Testbed Tier

4.3.1 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 supports a local storage path (*Local Data Repository*) for storing local resources and configurations and provides mechanisms for the synchronization of local and central data repositories. Testbed Manager will be developed as a stand-alone desktop application with Graphical User Interface enabling the visualization of resources, experiments and all relevant events within testbed boundaries.
Provided functionalities	• Provides a graphical interface for the creation, reading, update and deletion of resources • Propagates additions and changes of resources to the Master Data Repository using Testbeds Directory Service API • Contains the registration log for the experiments in the tested • Periodically receives the status of ongoing experiments • Stores configuration parameters for the UxVs in the relevant Testbed • Keeps statistics about testbed usage • Hosts Monitoring Manager component for presentation of resources parameters during their utilization in experiments
Relation to other components	• Experiment Controller o IN ← Experiment start/cancel information • Resource Controller o IN ← Current experiment status o OUT → Cancelation of experiment in emergency case • Local Data Repository o IN ← UxVs configuration parameters o IN ← Experiments log • Testbeds Directory Service o OUT → Call REST API to synchronize resources between Local and Master Data Repository • SFA Aggregate Manager o OUT → Call REST API to enable creation, update and deletion of resources in SFA Aggregate Manager repository
Changes	• Graphical User Interface support

	• Synchronization of local and central databases during creation, update and deletion of resources

4.3.2 Monitoring Manager

Component	**Monitoring Manager (subcomponent of Testbed Manager)**
Responsible partner	HAI
Parent Component	Testbed Manager
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 checks the current status of the resources in the facility participating in experiments like remaining energy, CPU load, storage usage, etc. • Checks the location and attitude characteristics of the resources participating in experiments • Periodically monitors testbed's CPU load, RAM, storage and network usage and transmits the current status of the Testbed to the System Monitoring Service. • Displays all the above information in the graphical interface of the Testbed Manager
Relation to other components	• System Monitoring Service (via Message Bus) o OUT → testbed status and performance values • UxV Node (via Message Bus) o IN ← Remaining Energy, CPU load, storage usage of UxV o IN ← location and attitude of UxV
Changes	• Added coupling with Testbed Manager for presentation of information in Graphical User Interface at testbed level • Added interactions with UxV nodes via Message Bus

4.3.3 UxV – Proximity Component

Component	**UxV – Proximity Component**
Responsible partner	CSEM
Parent Component	UxV (all UxV components), Message bus
Description	The proximity component allows members of a swarm of autonomous vehicles to discover the presence and possibly interact with each other with very low latency without depending on the RAWFIE middleware or any other ground equipment. In essence, the approach bears similarities with the transponders on commercial airplanes even though

	it offers additional services. The RAWFIE proximity component is based on specific hardware, which is integrated into the UxV in the form of a secondary radio communication interface. That secondary interface is low-power to minimise its impact on the autonomy of the vehicles. The software part of the component implements the protocols and services as well as the interface with the other UxV components.
Provided functionalities	Generic direct data exchange between the UxVs following the producer/consumer and publish/subscribe models. Typical applications realised in partnership with other components: • Detection of the presence and identification of neighbouring UxV, • Exchange of position, course and speed information between a UxV and its neighbours to prevent collisions • Detection of failed UxVs or UxVs disconnected from the main network • Localisation of lost UxVs stopped or grounded due to fuel or battery exhaustion • Data collection on static low power wireless sensors.
Relation to other components	• Serial interface of the hardware with the UxV • Host Command Interface (HCI) protocol on top of the serial interface. • Implementation of a Proximity Delegate component on the UxV to interface the UxV components (UxV Node, UxV Network communications, …) with the proximity component radio. • Depending on the application, the Proximity Component delegate may communicate directly with the message bus (e.g. to publish a list of its neighbours) or internally with the components inside the UxV (e.g. collision avoidance).
Changes	• New

Part V: Annex

Annex A 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
CPS	Cyber-Physical systems
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
HDFS	Hadoop Distributed File System
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 1: 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 2: Notation

Annex B 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

[20] *RSpec Version 3 Specification* - http://groups.geni.net/geni/wiki/RSpecSchema3

[21] Slice-based Facility Architecture. e]. http://opensfa. info/doc/opensfa.html

[22] *The Open-Multinet Upper Ontology - Towards the Semantic-based Management of Federated Infrastructures*, A. Willner, C. Papagianni, M. Giatili, P. Grosso, M. Morsey, Al-Hazmi Y., I. Baldin, "", The 10th International Conference on Testbeds and Research

Infrastructures for the Development of Networks & Communities (TRIDENTCOM 2015), Vancouver, Canada, June 2015.

[23] *DBcloud: Semantic Dataset for the Cloud*, M. Morsey, A. Willner, R. Loughnane, M. Giatili, C. Papagianni, I. Baldin, P. Grosso, Y. Al-Hazmi, "", accepted to appear at CRNET, Infocom 2016.

[24] *Design, architecture and implementation of a resource discovery, reservation and provisioning framework for testbeds*, D. Stavropoulos, A. Dadoukis, T. Rakotoarivelo, M. Ott, T. Korakis, L. Tassiulas,, Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt), 2015 13th International Symposium on. IEEE, 2015.

[25] *Holistic schedulability analysis for distributed hard real-time systems*, Tindell and J. Clark, Microprocessing and Microprogramming, vol. 40, 1994.

[26] *A Stateful Processor Interconnect*, Sutton F. et al., Bolt:,SenSys'15, 2015.

[27] *Adaptive Real-Time Communication for Wireless Cyber-Physical Systems*, Marco Zimmerling, Luca Mottola, Pratyush Kumar, FedericoFerrari, Lothar Thiele, ACM Transactions on Cyber-Physical Systems,1(2), 2017.

[28] *End-to-end real-time guarantees in wirelesscyber physical systems*, Romain Jacob ; Marco Zimmerling ; Pengcheng Huang ; JanBeutel ; Lothar Thiele., IEEE Real-Time Systems Symposium (RTSS), 2016.DOI: 10.1109/RTSS.2016.025

[29] *Real-Time communication and coordination in embedded sensor networks*, John A. Stankovic , Tarek Abdelzaher , Chenyang Lu , Lui Sha , Jennifer Hou, Proc. of the IEEE 91(7), pp. 1002--1022 (2003)

[30] Apache HBase - https://hbase.apache.org/

[31] *Kafka Connect for Hbase* - https://github.com/mravi/kafka-connect-hbase

[32] *Learning Apache Kafka*, second edition, Nishant Garg - 2015