

interiot

INTEROPERABILITY
OF HETEROGENEOUS
IOT PLATFORMS.

D3.3

Methods for Interoperability and Integration - Final

June 2018

INTER-IoT

INTER-IoT aim is to design, implement and test a framework that will allow interoperability among different Internet of Things (IoT) platforms.

Most current existing IoT developments are based on "closed-loop" concepts, focusing on a specific purpose and being isolated from the rest of the world. Integration between heterogeneous elements is usually done at device or network level, and is just limited to data gathering. Our belief is that a multi-layered approach integrating different IoT devices, networks, platforms, services and applications will allow a global continuum of data, infrastructures and services that can enable different IoT scenarios. As well, reuse and integration of existing and future IoT systems will be facilitated, creating a de-facto global ecosystem of interoperable IoT platforms.

In the absence of global IoT standards, the INTER-IoT results will allow any company to design and develop new IoT devices or services, leveraging on the existing ecosystem, and bring get them to market quickly.

INTER-IoT has been financed by the Horizon 2020 initiative of the European Commission, contract 687283.

Methods for Interoperability and Integration - Final

Version: 2.2
Security: Public
June 30, 2018

The INTER-IoT project has been financed by the Horizon 2020 initiative of the European Commission, contract 687283



Disclaimer

This document contains material, which is the copyright of certain INTER-IoT consortium parties, and may not be reproduced or copied without permission.

The information contained in this document is the proprietary confidential information of the INTER-IoT consortium (including the Commission Services) and may not be disclosed except in accordance with the consortium agreement.

The commercial use of any information contained in this document may require a license from the proprietor of that information.

Neither the project consortium as a whole nor a certain party of the consortium warrant that the information contained in this document is capable of use, nor that use of the information is free from risk, and accepts no liability for loss or damage suffered by any person using this information.

The information in this document is subject to change without notice.

Executive Summary

The aim of Deliverable 3.3, entitled “Methods for Interoperability and Integration - Final”, is to provide accurate documentation of the work done in the last implementation phase of INTER-Layer interoperability mechanisms. This deliverable is the third version of a series of three (i.e. preceded by D3.1 and D3.2). From now on, all the improvements made to the components developed in WP3 will be tracked through the official INTER-IoT web documentation. Thus, the current deliverable provides information about the final component developments, release features, documentation elaborated, software distribution plan and extensibility of the solutions. It reports the technical work performed in all WP3 tasks, T3.1 (Definition and Analysis of Methods for Device Layer Interoperability and Integration, M5-M30); T3.2 (Definition and Analysis of Methods for Networking Layer Interoperability and Integration, M5- M30); T3.3 (Definition and Analysis of Methods for Middleware Layer Interoperability and Integration, M5-M30); T3.4 (Definition and Analysis of Methods for Application Service Layer Interoperability and Integration, M5-M30), T3.5 (Definition and Analysis of Methods for Data and Semantics Layer Interoperability and Integration, M5-M30) and T3.6 (Definition and Analysis of Methods for Cross-Layer Interoperability and Integration, M13-M30).

D3.1 and D3.2 provide a literature review, describe the architecture, components and use cases, the progress of implementation, the API provided and the demonstration of methods of interoperability. In D3.3, focus has been expanded to report about implemented features, documentation, source code and releases. Firstly, offering a complete overview of progress since D3.2 and relation with other WPs. Secondly, explaining the general details of Inter-Layer software distribution and documentation. Thirdly, providing for each INTER-Layer component, an introduction with mentions to progress since the release described in D3.2, the list of component features, approaches to extensibility and a release and distribution plan. Finally, the document explains the ethical considerations across all INTER-Layer components.

As already reported in the aforementioned previous versions of this deliverable, the developments have been based on the layered architecture description provided in the Description of the Action, requirements described in D2.3 and subsequent updates (INTER-IoT Requirements and Business Analysis, M9). The work has been further based on use cases and scenarios described in D2.4 (Use cases and scenarios, M12) in order to be in line with the proposed pilots.

INTER-Layer is an instantiation (reference implementation) of the INTER-IoT Reference Architecture, which is explained in D4.2 (Final Reference IoT Platform Meta-Architecture and Meta Data Model). The exposed API and extensibility mechanisms of INTER-Layer are integrated in INER-FW and INTER-API, as described in D4.3 (Interoperable IoT Framework Model and Engine v1) and D4.5 (Interoperable IoT Framework API and Tools v1). As INTER-API is a result of the REST-like design of the INTER-IoT layered interoperability stack, providing a unique approach and experience to call the different interoperability APIs developed in WP3. The relation of WP3 with these tasks is reported in deliverables D4.3 and D4.6 (deliverable that merged D4.4 and D4.5). Finally, through interaction with WP4 tasks related to implementation of INTER-FW and INTER-API, extensibility and provision of APIs has been defined and implemented. The relation between the other work packages (WP4, WP5, WP6, WP7 and WP8) is provided in section 1.2.

This document is the last report about the activities of WP3, even if new software releases are provided, these will be improvement iterations over original developments carried out during INTER-IoT WP3 and formally will be assumed by WP6 in INTER-IoT and by the community in further projects.

List of Authors

Organisation	Authors	Main contributions
UPV	Eneko Olivares, Andreu Belsa, Jara Suárez de Puga, Carlos Enrique Palau	Overall coordination, Sections: 2.1, 2.2, 2.3, 2.4 3.1, 3.2, 3.4, 3.6, 5
RINICOM	Eric Carlson	Internal review. Sections: 1.2, 3.2, 4
XLAB	Flavio Fuart, Damjan Murn, Gašper Vrhovšek	Sections: 1, 1.1, 2.1, 2.4, 3.3, 3.5
SRIPAS	Katarzyna Wasielewska-Michniewska	Sections: 2.2, 2.3, 2.4, 3.3, 3.4, 3.5
TU/e	Tim van der Lee	Sections: 3.2, 5
SABIEN	Gema Ibáñez	Sections: 4
NEWAYS	Dennis Engbers, Johan Schabink	Internal review. Sections: 1.2, 3.1
PRODEVELOP	Miguel Ángel Llorente	Sections: 1.2, 3.3, 3.5, 3.6
UNICAL	Pasquale Pace, Giuseppe Caliciuri	Sections: 1.2, 2.3, 3.4

Change control datasheet

Version	Changes	Pages
0.0	Formatting, Inter-IoT template	6
0.1	Table of contents and assignments	12
0.2	Introduction section	15
0.3	Distribution section, first draft	19
0.4	Layer sections, first draft	25
1.0	Ethics section	29
1.1	Distribution section	32
1.2	Layer sections	37
1.3	Executive summary	39
2.0	Ready for internal review	44
2.1	Review comments addressed	48
2.2	Final version	49

Contents

Executive Summary	6
List of Authors	7
Change control datasheet	8
List of Figures	11
List of tables	13
Acronyms	15
1 Introduction	17
1.1 Progress since D3.2	17
1.2 Relation with other Work Packages	18
1.2.1 Relation with WP4	18
1.2.2 Relation with WP5	19
1.2.3 Relation with WP6	19
1.2.4 Relation with WP7	19
1.2.5 Relation with WP8	19
2 Software Distribution and Documentation	21
2.1 Source code	21
2.2 Documentation	22
2.3 Binary distribution	25
2.4 Release Summary	26
3 INTER-Layer Components	30
3.1 D2D solution	30
3.1.1 Release features	30
3.1.2 Extensibility	31
3.1.3 Release and distribution plan	31
3.2 N2N solution	32
3.2.1 Release features	33
3.2.2 Extensibility	33
3.2.3 Release and distribution plan	34
3.3 MW2MW solution	34
3.3.1 Release features	35
3.3.2 Extensibility	37
3.3.3 Release and distribution plan	37
3.4 AS2AS solution	37
3.4.1 Release features	38

3.4.2	Extensibility	39
3.4.3	Release and distribution plan	40
3.5	DS2DS solution	41
3.5.1	Release features	41
3.5.2	Extensibility	42
3.5.3	Release and distribution plan	42
3.6	Cross-Layer solution	42
3.6.1	Layer security integration	43
3.6.2	Layer Interactions	43
3.6.3	Virtualization and Clusterization of layers	43
3.6.4	Cross-Layer as a transversal component	44
4	Ethics	45
4.1	Introduction	45
4.2	Ethics and INTER-Layer	45
4.2.1	Data types	45
4.2.2	Requirements for ethical data processing	46
5	Conclusions	48

List of Figures

Figure 1: Private source code repository based on Gogs	22
Figure 2: INTER-IoT documentation site example	23
Figure 3: Continuous development infrastructure	24
Figure 4: INTER-IoT private binary repository based on Sonatype Nexus	25
Figure 5: INTER-IoT private docker registry	26

List of Tables

Table 1: Current documentation sites summary	23
Table 2: Documentation version scheme example	24
Table 3: Summary table of INTER-Layer D2D component distribution	27
Table 4: Summary table of INTER-Layer N2N component distribution	27
Table 5: Summary table of INTER-Layer MW2MW component distribution	28
Table 6: Summary table of INTER-Layer AS2AS component distribution	28
Table 7: Summary table of INTER-Layer DS2DS component distribution	29

Acronyms

AAL	Active-Assisted Living
ACP	Access Control Policy
ACR	Access Control Rule
API	Application Programming Interface
AS2AS	Application & Services Interoperability
CLI	Command-line Interface
CASE	Computer Aided Software Engineering
D#.#	Deliverable number #.# (D2.1 deliverable 1 of work package 2)
DS2DS	Data & Semantics Interoperability
D2D	Device Interoperability
EC	European Commission
EU	European Union
GA	Grant Agreement
GNU	GNU's Not Unix
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
H2020	Horizon 2020 Programme for Research and Innovation
INTER-FW	INTER-IoT Interoperable IoT Framework
INTER-IoT	Interoperability of Heterogeneous IoT Platform
INTER-Layer	INTER-IoT Layer integration tools
INTERMW	INTER-IoT Middleware
IoT	Internet of Things
IoT-EPI	IoT-European Platforms Initiative
JSON-LD	JavaScript Object Notation used for serialization of Linked Data
JVM	Java Virtual Machine
M#	#th month of the project (M1=January 2016)
MDM	Meta-Data Model
MW2MW	Middleware Interoperability

N2N	Network Interoperability
OF	OpenFlow
OFDM	Orthogonal Frequency Division Multiplexing
OS	Operating System
OSGi	Open Services Gateway initiative
OVSDB	Open vSwitch Database Protocol
OWL	Web Ontology Language
PC	Project Coordinator
PCC	Project Coordination Committee
RA	Reference Architecture
RDF	Resource description Framework
REST	Representational State Transfer
RM	Reference Model
QoS	Quality of Service
SDN	Software Defined Networks
SDR	Software defined Radio
SQL	Structured Query Language
TLS	Transport Layer Security
UDP	User Datagram Protocol
UI	User Interface
URI	Uniform Resource Identifier
XACML	eXtensible Access Control Markup Language
WP	Work Package

1 Introduction

This deliverable is an evolution of D3.1, *Methods for Interoperability and Integration v.1* and D3.2, *Methods for Interoperability and Integration v.2*. It reports about final outcomes of research and development efforts elaborated in the previous documents. The fact that this, third, deliverable reports about implemented features, documentation, source code and releases makes it a self-standing document.

A reader interested in using the software developed in WP3 can find all the needed information in this document and will not need to refer to the previous versions. The authors also made extensive effort to publish the information needed for usage and further development in standard approaches of publishing documentation and source code. This means, that the main entry for future users will be the selected open source code repository and document repository, GitHub and Read the Docs, explained in chapter 2.

On the other hand, a more curious reader, interested in the theoretical background and reasons for specific technological choices, will reach for D3.1 and D3.2.

In order to position this report in the project's perspective, an overview of progress since D3.2 (Section 1.1) and relation with other WPs (Section 1.2) is provided below. The details of Software Distribution and documentation are provided in Section 2. In Section 3 for each INTER-Layer component we provide: an introduction with the progress made since the release described in D3.2, the list of component features, approaches to extensibility and finally, a release and distribution plan. Ethical considerations are elaborated in a separate section across all INTER-Layer components. In the conclusion we summarise the main points of the work performed in this WP.

1.1 Progress since D3.2

In this final phase of technical developments, most of the effort across all INTER-Layer components was devoted to further development of test procedures, refinement of API calls, integration within Cross-Layer and WP4, provision of documentation and development of deployment procedures. The source code of all layers is kept in the project's Gogs repository, until all components are equipped with the correct license texts, at which point they will be moved to Github. In order to support the documentation efforts, all the necessary infrastructure to streamline the creation of comprehensive documentation has been put in place (Section 2.2). A process to release binary distributions is in place (Section 2.3), together with a first set of software releases (Section 2.4).

At the time of submission of D3.2, the main architectural choices were made and most of the core

INTER-Layer functionalities were already developed. In this final phase of technical developments, some components have seen further extension of their features and services they offer. The specifics are further elaborated in INTER-Layer Component introductory sections for each layer separately.

1.2 Relation with other Work Packages

The development activities performed in WP3 are devoted to the layer infrastructure of the INTER-IoT interoperability approach. This means that the core of the full interoperability stack provided in INTER-IoT is technically designed and developed within WP3. Thus, the activities have multiple links with other work packages. Although the technical work has been managed to keep it as much independent as possible. In order to make the most of the research activities planned in the tasks, the tight relation between each layer and the rest of the work packages has been a constant in the technical, management and dissemination activities across the project.

1.2.1 Relation with WP4

One of the most related work package in technical terms to WP3 is WP4. Within this set of tasks, two important branches can be differentiated: 1) the abstraction and theoretical description activities carried out in tasks 4.1 and 4.2; and 2) the homogenization and user experience enhancement activities represented by the design and implementation of a transversal framework and API performed in 4.3, 4.4 and 4.5.

For the first family of activities, the development of layers has bidirectional feedback. The first design of the INTER-IoT Reference Model (RM), Reference Architecture (RA) and Meta-Data Model (MDM), delivered in Month 13 under the deliverable 4.1, was an initial theoretical background and architectural approach for the development of each layer interoperability mechanism, serving as a general reference to ensure the compatibility, de-coupling, complementariness and completeness of the solutions proposed. In the refined version of the RA, RM and MDM, delivered in M24 in the document D4.2 the development of the layer infrastructure played a role, influencing in the final version of these abstractions, similarly, this work provided a theoretical background for the last developments of WP3.

For the second part of activities of WP4, the layers have a key influence, since the purpose of the framework and INTER-API is to ease the independent use of the layers, to homogenize the user and developer experience and provide a seamless process to build a complete IoT interoperability solution based on the use of the interoperability layers. More specifically, INTERFW has developed a graphical web based framework (called configuration and management framework) with a server-side application that supports key activities for the layers such as user management, identification and authorization management, orchestration of API calls and layers management. The INTER-API is a result of the REST-like design of the INTER-IoT layered interoperability stack, providing a unique approach and experience to call the different interoperability APIs developed in WP3. The relation of WP3 with these tasks is reported in deliverables D4.3 and D4.6 (merged D4.4 and D4.5).

1.2.2 Relation with WP5

WP5 is mainly related to WP3 on the implemented layered approach. In particular, T5.2 defines a set of pre-instantiated interoperability-oriented patterns which are tailored on and eventually implemented via the APIs at the various layers (*i.e.* D2D, N2N, MW2MW, AS2AS, DS2DS) of INTER-Layer and through INTER-FW.

T5.3 is mainly devoted to the implementation of the INTER-CASE Tool consisting of different phases. In the *Analysis* phase, the concept of integration point is exactly mapped to one of the WP3 layers at which a given integration requirement is located. Also, the final output of the *Analysis* phase (*i.e.* the Goal Oriented Model) includes a list of identified categories of integration (*i.e.* layers).

The *Design* phase is focused on the definition and instantiation of patterns which are, as aforementioned, tailored to INTER-Layer. Finally, the relation with WP3 emerges at the *Maintenance* phase, where a list of known bugs and future evolution points of the integrated platform are identified according to given affected layers.

1.2.3 Relation with WP6

Work Package 6 is, basically, the prove that WP3 is done according to requirements and use cases which are specified in WP2. In WP3, all the components developed form the INTER-Layer solution and they cover the set of requirements and use cases. WP6 entails the integration of all the separate modules and components into working systems in the form of pilots. The pilots will prove during WP6 that the implementation in WP3 was successful and according to specification. This will be done in the form of unit testing, integration testing, Factory Acceptance Testing (FAT) and finally Site Acceptance Testing (SAT). The modules which form the components are unit tested during build. The component will be integration tested on functionality, performance, interfaces, etc. Next, the system will undergo the FAT to prove that the system is ready for deployment at the installation site. Finally, the system will undergo the SAT test which will prove that the system operates as specified at the installation location of the pilots. The unit tests and integration testing are part of the WP3 package, the FAT and SAT are part of the WP6 package, all tests combined will prove the INTER-IoT solution as defined in WP2.

1.2.4 Relation with WP7

Work Package 7 is the formal evaluation of all aspects of the project. This includes the exploitation, pilots, impact, interoperability and ethical, societal, gender and legal evaluation. In relation to WP3, the technical evaluation will focus on KPIs related to specific pieces of technology developed in INTER-layer. Additionally, the process evaluation will review the system and tools used to control and manage all aspects of WP3 with the outcome being D7.3 where KPIs are reported.

1.2.5 Relation with WP8

Work Package 8 is focused on Impact Creation. In relation to WP3, many of the pieces of technology developed in INTER-Layer are whole or parts of products which feature in the INTER-IoT business

and exploitation plans. Understanding their capabilities and unique selling points allows for the development of convincing business cases and value propositions. Additionally, the INTER-layer code base will be Open source and as such will be central to the Freemium business model developed as part of INTER-IoT.

2 Software Distribution and Documentation

2.1 Source code

Since the beginning of the project, source code has been stored in our own private Git repository server¹ (Figure 1). Although this repository server infrastructure was deployed and maintained as part of WP3, it is used by all other Work Packages within the project. For that reason, even that WP3, WP4 and WP5 end in M30, our private Git repository will be maintained until the end of WP6 and WP7. Once we have a positive evaluation of our first release, we will start the process of migration to the selected public repository (Github²).

Before moving any code repository to Github, and as part of WP8 work, every piece of code and software component has to be fully covered by the chosen license (Apache 2.0³) and all dependencies need to be double checked for their compliancy with the selected license.

Once a code repository is moved from our private repository to Github, it will be deactivated from our private repository server in order to avoid conflicts with multiple remote repositories. When a code repository is migrated to Github it may change the name to something more suitable, but all of them will be added to the Inter-IoT project in Github⁴ and all original commits will be kept.

¹<https://git.inter-iot.eu>

²<https://github.com>

³<https://www.apache.org/licenses/LICENSE-2.0>

⁴<https://github.com/interiot>

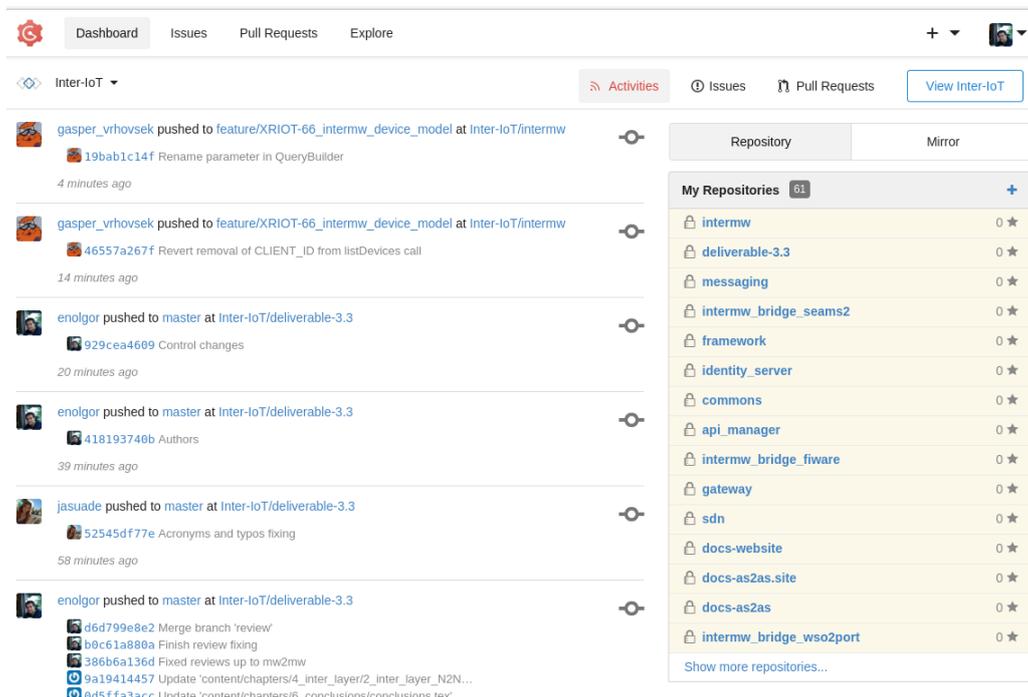


Figure 1: Private source code repository based on Gogs

2.2 Documentation

As part of WP3 technical work, the infrastructure that supports all INTER-IoT documentation generation and publishing has been created and deployed. Since the majority of the tasks performed during the technical stages of INTER-IoT project have been mostly condensed in this Work Package, this infrastructure has been originally deployed to support INTER-Layer documentation but has been extended to support INTER-FW from WP4 and INTER-CASE from WP5, and can be extended to cover any INTER-IoT component that needs a separate documentation site.

This documentation site is self-hosted for the moment in the projects documentation, site⁵ but once all INTER-IoT source code has been migrated to Github (as explained in the previous section), all documentation will be also migrated to the selected documentation hosting site (Readthedocs⁶).

Currently, self-hosted documentation follows this scheme:

`https://docs.inter-iot.eu/docs/<component>/<version>/`

Where `<component>` points to the correspondent INTER-IoT component, currently:

A special case is the `hub` documentation site. It acts as the main documentation portal, with general information about the INTER-IoT project, explaining how and why it's separated in different components and linking to each of the software component specific documentation sites.

In the documentation site address scheme explained above, `<version>` refers to the related software component release version, with two special cases: `stable` will always point to the current release

⁵<https://docs.inter-iot.eu>

⁶<https://readthedocs.org/>

<component>	WP	Description
gateway	WP3	D2D Layer: Gateway documentation
n2n	WP3	N2N Layer: SDN and SDR documentation
intermw	WP3	MW2MW Layer: INTERMW documentation
as2as	WP3	AS2AS Layer: Node-RED based AS2AS tool documentation
ipsm	WP3	DS2DS Layer: IPSM documentation
framework	WP4	INTER-FW documentation
intermeth	WP5	INTER-METH and INTER-CASE documentation
hub	WP8	INTER-IoT (global) documentation site

Table 1: Current documentation sites summary

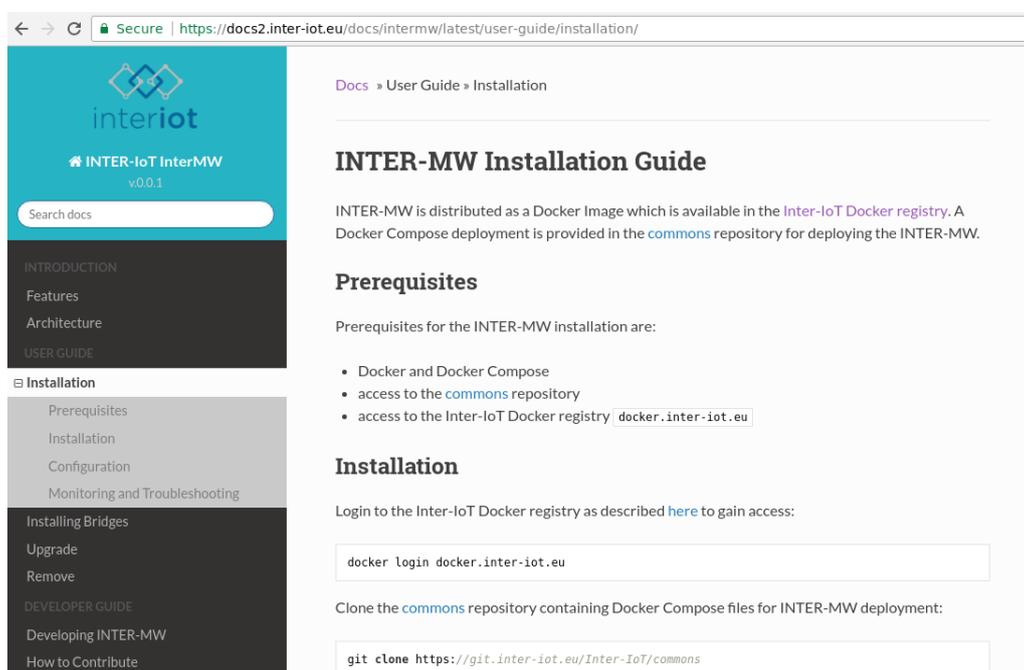


Figure 2: INTER-IoT documentation site example

of the component and `latest` will always point to the current development version of the software component. An example documentation site can be seen in Figure 2

The selected tool to create the documentation sites has been Mkdocs⁷ due to its simplicity to use, open-source nature and because it uses Markdown as the format to write documentation. Up to this point, a lot of documentation was already written in Markdown format (our private code repository, Gogs, as well as Github use this format for the README documents, Wiki, Issues and Pull Request descriptions) it was straightforward to reuse the already written documentation.

This tool (Mkdocs) takes all the markdown files (interlinked), images and other resources to create a documentation website. For this reason, we can consider the markdown files as the "source code" of the documentation and the website as the "compiled" result of this documentation source. Thus, this documentation source is also tracked in our code repository server in a separate repository for each

⁷<https://www.mkdocs.org/>

documentation site.

In order to track correctly the different documentation versions that refers to the software component specific version, git branches and commit tags are used. In the following table can be seen an example of this documentation versioning schema and how it relates to the specific git branch and tag:

<version>	Component Version	Description	Branch	Tag
1.0.0	1.0.0	old release	master	1.0.0
2.1.0	2.1.0	current release	master	2.1.0
stable	2.1.0	current release	master	HEAD
latest	2.5.2	current development	dev	HEAD

Table 2: Documentation version scheme example

In order to have a comfortable working environment to write documentation, the CI (Continuous Integration) setup that was deployed as part of WP3 development environment has been used (see Figure 3). In this case, every time a commit is made to the dev or master branch (HEAD pointer changes the commit hash) of any of the documentation source repositories, Jenkins starts the compilation process and publishes the result to the documentation hosting server in the corresponding version site. The result of the compilation is published in the INTER-IoT #builds slack channel and any failure is notified to the committer by mail with the compilation log file.

When a tagged commit in the master branch is made, the compiled result is also published to the documentation hosting server but with the tag as the version key, in order to keep the structure that has been previously explained.

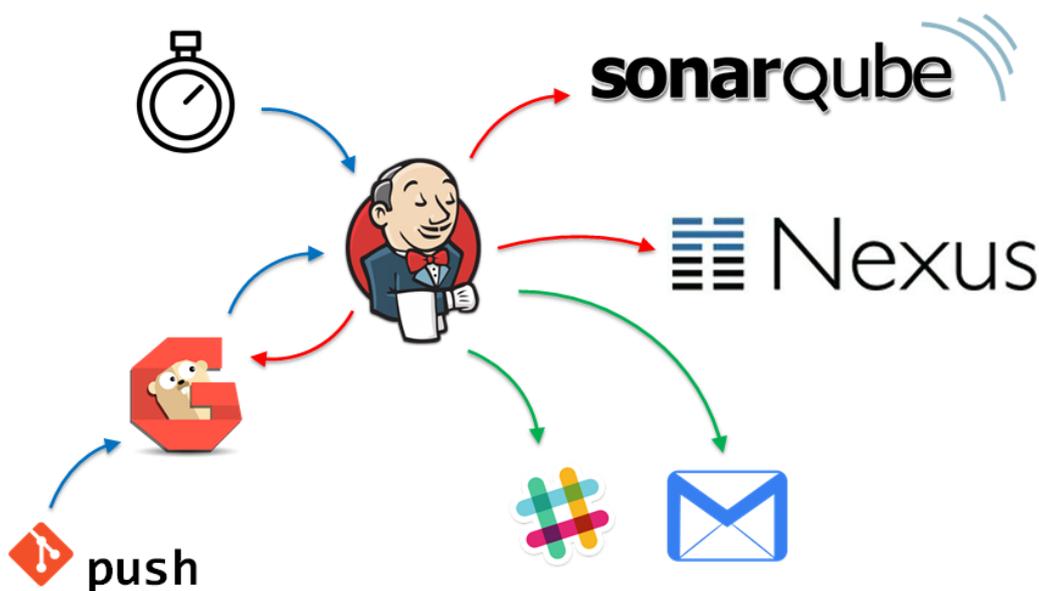


Figure 3: Continuous development infrastructure

2.3 Binary distribution

The main result of INTER-Layer will be the compiled binary software components developed in the different layers. Each software component could have different distribution packages but the following rules are applied:

- Each software component must have a portable zip package with the released binaries, attached to the specific commit tag in the code repository as a release. This zip package should be OS independent or else multiple versions of the portable zip package for the most important OS must be released.
- Each zip file will also be pushed to any release repository available in our private binary repository server, Nexus⁸ (Figure 4).
- Each release must have a specific version of the documentation, and that documentation must contain links to download those binary packages. The documentation has to be clear enough for a successful installation of each distribution release package.
- If, in any case, additional distribution packages are deployed in each release (most of them will have a Docker image available in our private Docker repository server⁹, Figure 5), they must be documented and referenced.

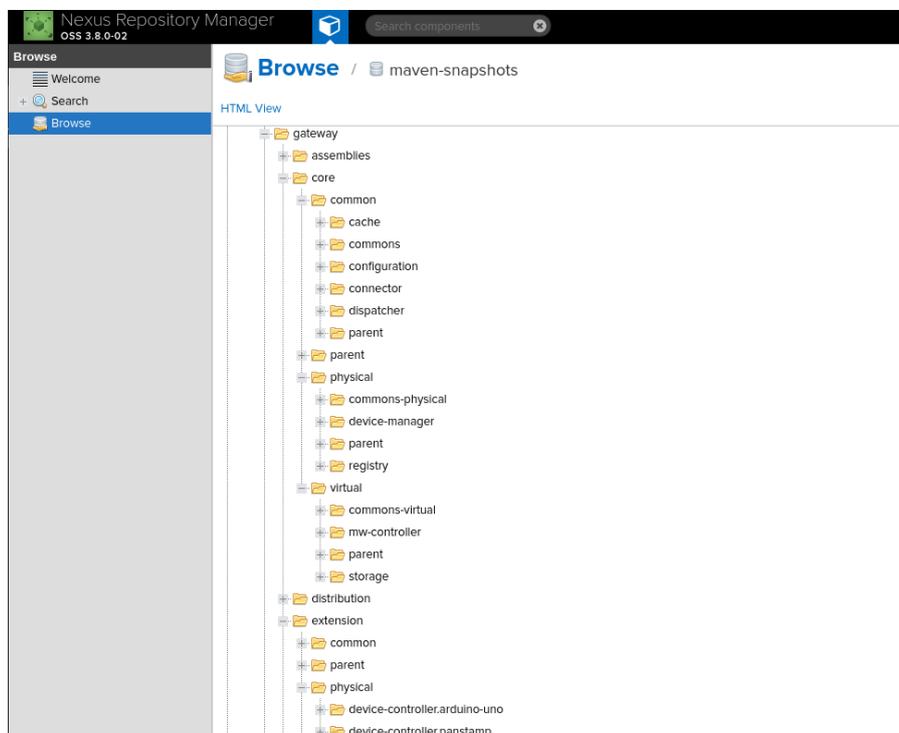
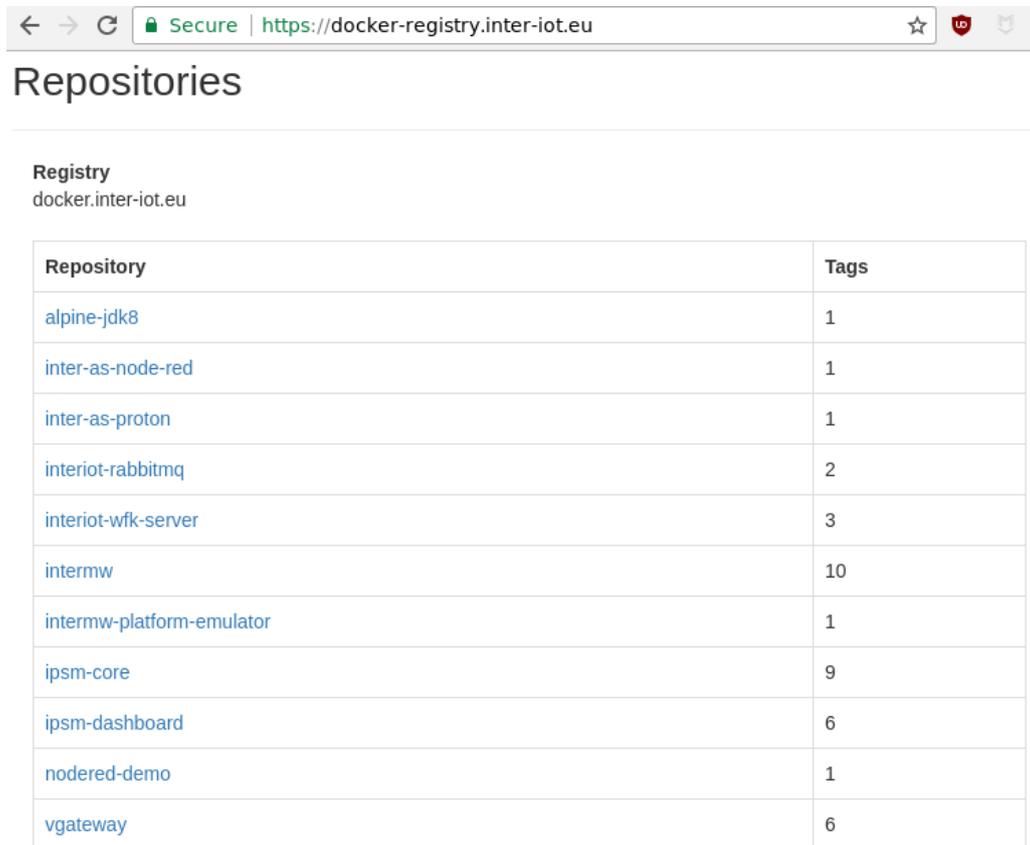


Figure 4: INTER-IoT private binary repository based on Sonatype Nexus

⁸<http://nexus.inter-iot.eu>

⁹<https://docker-registry.inter-iot.eu>



Registry
docker.inter-iot.eu

Repository	Tags
alpine-jdk8	1
inter-as-node-red	1
inter-as-proton	1
interiot-rabbitmq	2
interiot-wfk-server	3
intermw	10
intermw-platform-emulator	1
ipsm-core	9
ipsm-dashboard	6
nodered-demo	1
vgateway	6

Figure 5: INTER-IoT private docker registry

2.4 Release Summary

In this section the release information summary table for each INTER-Layer component is given, including component name, version, source code, binaries, documentation, Docker image and dependencies.

D2D	Physical Gateway	Version	1.0.0
		Source Code	https://git.inter-iot.eu/Inter-IoT/gateway
		Binaries	http://nexus.inter-iot.eu/repository/maven-releases/eu/interiot/gateway/distribution/physical-gateway/1.0.0/physical-gateway-1.0.0-dist.zip
		Documentation	https://docs.inter-iot.eu/docs/gateway/1.0.0/
		Docker Image	—
		Dependencies	—
	Virtual Gateway	Version	1.0.0
		Source Code	https://git.inter-iot.eu/Inter-IoT/gateway
		Binaries	http://nexus.inter-iot.eu/repository/maven-releases/eu/interiot/gateway/distribution/virtual-gateway/1.0.0/virtual-gateway-1.0.0-dist.zip
		Documentation	https://docs.inter-iot.eu/docs/gateway/1.0.0/
		Docker Image	docker.inter-iot.eu/vgateway:1.0.0
		Dependencies	—

Table 3: Summary table of INTER-Layer D2D component distribution

N2N	OpenFlow switch	Version	2.9.2
		Source Code	https://git.inter-iot.eu/Inter-IoT/sdn
		Binaries	—
		Documentation	https://docs.inter-iot.eu/docs/n2n/latest/
		Docker Image	—
		Dependencies	The virtual switch can be exchanged with other virtual switch, but taking into account the following dependencies/issues: (a) controller and switch communicate with v1.3 of Openflow protocol, (b) switch must have OVSDB protocol implemented to configure queues and meters, (c) mechanism of queues and metering has to be implemented within the switch.
	Controller	Version	1.0.0
		Source Code	https://git.inter-iot.eu/Inter-IoT/sdn
		Binaries	http://nexus.inter-iot.eu/repository/raw-releases/eu/interiot/n2n/sdn_controller/1.0.0/sdn_controller-1.0.0.zip
		Documentation	https://docs.inter-iot.eu/docs/n2n/latest/
		Docker Image	—
		Dependencies	Applications: topology, qos, Inter IoT switch.

Table 4: Summary table of INTER-Layer N2N component distribution

MW2MW	Core Engine	Version	2.1.0
		Source Code	https://git.inter-iot.eu/Inter-IoT/intermw
		Binaries	http://nexus.inter-iot.eu/repository/maven-public/eu/interiot/intermw/mw.api.rest/2.1.0-SNAPSHOT/mw.api.rest-2.1.0-20180620.152417-27.war
		Documentation	https://docs.inter-iot.eu/docs/intermw/latest/
		Docker Image	Docker Compose with containers: docker.inter-iot.eu/intermw , rabbitmq:3.7-management-alpine , daxid/parliament-triplestore
		Dependencies	Internally dependent on RabbitMQ and Parliament triple store. Optional dependency on IPSM (if semantic translation is needed).
	INTERMW Bridges	Version	—
		Source Code	Each bridge developer manages their own GIT repository. Currently hosted on the InterIoT Git repo.
		Binaries	—
		Documentation	https://docs.inter-iot.eu/docs/intermw/latest/
		Docker Image	—
	Dependencies	INTERMW Core Engine	

Table 5: Summary table of INTER-Layer MW2MW component distribution

AS2AS	Application and service solution	Version	1.0.0
		Source Code	https://git.inter-iot.eu/Inter-IoT/interas
		Binaries	http://nexus.inter-iot.eu/repository/raw-releases/eu/interiot/as2as/as2as-dist/1.0.0/as2as-dist-1.0.0.zip
		Documentation	https://docs.inter-iot.eu/docs/as2as/latest/
		Docker Image	docker.inter-iot.eu/inter-as-node-red:latest
		Dependencies	Flow repository and nodes repositories are integrated in INTER-FW. In principle, separate deployment of this components is not needed. To achieve the multiuser, multiple instances, scalability, several instances of Node-RED should be deployed. These instances are going to be managed by INTER-FW. For details about the deployment of different instances (running flows) of Node-RED on the same host, see Deliverable 3.2.

Table 6: Summary table of INTER-Layer AS2AS component distribution

DS2DS	IPSM	Version	1.0.0
		Source Code	https://git.inter-iot.eu/Inter-IoT/ipsm-core
		Binaries	http://nexus.inter-iot.eu/repository/raw-releases/eu/interiot/ds2ds/ds2ds-dist/0.5.4/ds2ds-dist-0.5.4.zip
		Documentation	https://docs.inter-iot.eu/docs/ipsm/latest/
		Docker Image	Docker Compose with containers: wurstmeister/zookeeper, wurstmeister/kafka, docker-registry.inter-iot.eu/ipsm-core
		Dependencies	Internally dependent on: Apache Kafka, SQLite DB.
	IPSM Dashboard	Version	1.0.0
		Source Code	https://git.inter-iot.eu/Inter-IoT/ipsm-dashboard-deployment
		Binaries	http://nexus.inter-iot.eu/repository/raw-releases/eu/interiot/ds2ds/ds2ds-dashboard-dist/1.1.3/ds2ds-dashboard-dist-1.1.3.zip
		Documentation	https://docs.inter-iot.eu/docs/ipsm/latest/
		Docker Image	docker-registry.inter-iot.eu/ipsm-dashboard
		Dependencies	IPSM

Table 7: Summary table of INTER-Layer DS2DS component distribution

3 INTER-Layer Components

The content of this section provide a specific delta from the previous version of this deliverable (D3.2) and has been done in concordance with the solutions provided there. Also, this delta has followed the guidelines provided by architectural work packages and requirements (D2.3 and D4.3). This section will present the improvement of the interoperability solutions presented at the beginning of WP3 and the implementation status, as well as the steps to follow in the next stages of implementation.

3.1 D2D solution

In previous versions of this deliverable (D3.1 and D3.2) the architecture, components and functionalities where described. Progress from the last deliverable has been focused on refining the existing Gateway components, adding new features (such as automatic reconnection, new management commands, etc.), as well as adding more extensions to the physical and virtual parts of the gateway. Some refactoring was also done in order to prepare for the packaging and installation of the gateway distribution.

The most important progress has been achieved in the following areas: new device controller extensions (Modbus, UDP), rules engine extension, discovery extension, storage and cache extensions and the decoupling of all module interfaces. Decoupling all the module interfaces helped, for example, in the creation of another connector implementation based on simple http request and MQTT messages instead of a permanent WebSocket connection (relevant for a future physical gateway implementation on mobile devices, where a permanent connection is difficult to maintain).

Work also focused on improving the UI integration with INTER-FW and creating an automatic command line gateway installer with extension for browsing support.

Development versions started from 0.0.1 up until 0.4.X, 0.5.X is the current development version tagged as pre-release since this minor version is devoted only to clean-up, add license headers, etc. First release candidate version is 1.0.0, next version, 1.1.0, will contain minor fixes and all licensing information.

3.1.1 Release features

The first release of the D2D Gateway will include the following features:

- Decoupled Physical and Virtual Gateway
-

- Runs in any OS with a JVM
- Both are modular, consisting in core (mandatory) and extension (plugin) modules
- Core modules can also have different implementations (interface is decoupled)
- Physical features:
 - Support of multiple simultaneous physical device controllers, access networks and protocols
 - Easily extendable with custom device controllers
 - Support for sensors (passive and active) and actuators
 - Managed remotely (through the virtual gateway) or with a CLI (Console extension)
 - Support of multiple connector modules (Websocket, HTTP/REST, UDP Datagrams)
- Virtual features:
 - Support of multiple IoT middleware platforms (only one active)
 - Database extension for device data collection
 - API extension for remote management and query
 - Rules Engine extension for simple rules defined in SQL-like expressions and Javascript instructions
 - CLI for management (Console extension)
 - Support of multiple connector modules (Websocket, HTTP/REST, UDP Datagrams)

3.1.2 Extensibility

The D2D Gateway was built from the beginning with a clear approach towards extensibility. This approach followed OSGI recommendations for a clear decoupled and modular system. These extensions can be developed to work in virtual and/or physical parts of the Gateway. Depending on the case, the Gateway exports all available interfaces in the following packages: "commons", "commons-virtual" and "commons-physical".

Typically, physical extensions are centered in providing support for other device access network and protocols creating new device controllers while virtual extensions are centered in creating new middleware controllers to provide support for more IoT platforms. Common extensions provide utility for the gateway management, configuration, etc.

3.1.3 Release and distribution plan

The following releases are planned for the D2D Gateway:

- *Closing of WP3 activities.* This is the main release of the Gateway and will include all the features stated previously. This release will have version 1.0.0 but will not be ready for public distribution, but for internal usage.

- *License-ready release*. This release will have version 1.1.0 and will contain all licensing modifications and files in order to be ready for public distribution. Any minor fix will also be included.
- *Improved Release 1*. Will include all fixes, features and improvements that come from the feedback on the usage during the pilots deployment (WP6).
- *Improved Release 2*. This release will include any modifications and improvements from the results of the technical evaluation (WP7).

3.2 N2N solution

As explained in previous versions of this deliverable (D3.1 and D3.2), the network interoperability solution is based in two approaches. Firstly, the more physical approach based in Software Defined Radio is being developed to allow bespoke integration in this developing area. And secondly, a more data link and network oriented solution created using Software Defined Network architecture and components.

Regarding SDN, in the first phase, at the time of D3.2 submission, the installation, configuration and modification of modules for the SDN network solution were already implemented. The second phase of the work has focused on security and testing. Moreover, the design of the UI for communication with the controller has been improved to show the data and information of the network in a more desirable format. Additionally, the interaction between virtual gateway and SDN has been further refined in order to be applied to different use cases based on virtualization.

The improvements and testing performed over the Network solution include:

- Security analysis to implement solutions to preserve the privacy through the channel (implementing TLS)
- Inclusion of monitor modules to secure the operations within the controller (analysis tools as Snort)
- Design of scenario to test the performance and the KPIs designed for the last phases of the project

Finally, different scenarios have been defined in order to obtain the results of the KPIs defined at the beginning of the evaluation process.

We must keep in mind that the first stage of the development process has been concluded but, future iterations for review and improvement of the solution will be performed, here we present the first release features that network solution provide to solve the interoperability problem.

Regarding SDR, initial work addressed development of a bi-directional packet based OFDM system. Interfaces were developed for GNU radio modules to pipe Ethernet traffic in and out. The second phase focused on increased data throughput, changing transmit and receive frequencies and reducing dependence on GNU radio modules so the full solution can be deployed on Zync FPGA processor. This process is designed to increase modularization, data throughput and size, weight and power (SWaP). A key part of the COFDM system is the forward error correction code FEC. We have reviewed and are in the selection process of forward error correction (FEC) codes. The primary candidates under review are Polar, Turbo, and LDPC. Development of a full SDR is beyond the scope

of the INTER-IoT project. As such, documentation will reflect the work done. However, should future users of INTER-IoT wish to utilize their own SDR solutions, entry into the gateway will be feasible.

3.2.1 Release features

List of feature of the SDR network solution:

- Bi-directional packet based OFDM link
- Customizable frequency setting

The SDR component of INTER-IoT will offer an additional access point to the INTER-IoT gateway.

List of features of the SDN network solution:

- Network management
- Topology discovery
- QoS application
- Network status monitoring

When deploying an INTER-IoT solution, components such as devices, middlewares and/or INTER-IoT gateways can be inter-connected with SDN. SDN offers a centralized point to control and adapt the network. The network component of INTER-IoT provides a set of tools to configure and adapt the SDN network layer according to the requirements of any deployment.

- Virtual INTER-IoT customized switches can be monitored and configured. The interface provides the ability to add/delete/modify each flow entry. Statistics can be gathered for the switches, providing information and statistics about the flows, ports, or table of flows.
- The N2N layer also provides a visualization tool in order to visualize the entire deployment. Combined with the previous point, these two features ensure a full control over the virtual network.
- The network layer provides a QoS API in order to satisfy the potential QoS requirements of the deployment. When using the QoS API of INTER-IoT, the developer can add/delete/monitor rules, queues and meters. Rules determine whether the specified traffic is assigned to a certain queue or meter. Queues are designed to provide a guarantee on the rate of flow of packets placed in the queue. Different queues at different rates can be used to prioritize specific traffic. Meters complement the queue framework already in place by allowing for the rate-monitoring of traffic prior to output.

3.2.2 Extensibility

Network extensibility typically refers to the ability to include new components such as new hardware appliances such as hubs, switches, routers, etc. or services such as routing, firewalling, load balancing and so on, into the network, allocating these new elements easily, without affecting the performance and good operation of the network. In case of software defined network this approach is closer to the introduction of new services as the virtualization of the functions typically done in specialized hardware is performed together with the programmability of the network. Extensibility

normally involves multiple vendors and deployment architectures. So the ability to integrate the solution with these vendors provide advantages in comparison with traditional networks. Moreover, it provides a single point to manage multiple network heterogeneous devices, creating the desired interoperability. To have in consideration this growth of the network; for one side to include new services, enlarging its features and, for the other side, to extend the network cover of new different nodes, several approaches have been studied.

Implementing extensibility in the INTER-IoT network solution is something that was included already with the deployment of the SDN architecture, as this architecture allows and facilitates the inclusion of new network services and or nodes.

- Custom creation of network up to many bridges thanks to OVS.
- Extensibility based in the insertion of new types of switches.
- Extensibility based in the interconnection with other controllers.
- Extensibility based in the possibility of develop more network application using, controller as a base.

3.2.3 Release and distribution plan

As opposed to the developed modules belonging to the other layers, the network solution is not dockerized as must create the infrastructure where the other modules can run and communicate. The SDN controller can be isolated in a container, but in this case, due to scalability, extensibility and customization requirements we decided to not use containers. For that reason, the virtual switch and the controller must be installed independently (also they are independent in case you want to choose another switch for the data plane), and run both switch and controller with its respective applications. To facilitate this, a script for downloading, run and configuration of the solution as been implemented.

Finally, the network solution is going to be released open source under Apache 2.0 license. The following releases are planned:

- *Closing WP3 activities* - full implementation of core functionalities as planned for the execution of T3.2. Internal release, available to project partners and open call projects.
- *License-ready release* - containing all licensing information and ready for public distribution.

3.3 MW2MW solution

The MW2MW solution is described in D3.1 and D3.2. While the basic functionality to support the INTER-Health pilot and kick-start of the open call projects was provided at the time of D3.2 submission, the full release is provided at the time of submitting of D3.3. In addition to those features, some further refinements have been performed as result of feedback from pilots ant evaluation tasks. We refer to this M30 release as MW2MW V2 in the text that follows. The release is tagged as V2 as it contains substantial advancement from the previous version as well as some changes that are not backward-compatible.

In the *Services* section the device registry has been extended in order to store all relevant device types and meta-data. This component also orchestrates and executes the new Discovery mechanism that populates and maintains the device registry by obtaining devices information through Bridges.

Device Actuation has been implemented. It allows performance of actuation actions on IoT Devices.

As a result of feedback from the implementation of pilots and additional internal evaluation, we concluded that for most usage scenarios, the full exposure of JSON-LD structures in the REST API is too demanding for most developers. Therefore, INTERMW now offers two REST API interfaces. The basic interface that supports most of the usage scenarios abstracts the usage of JSON-LD and allows interaction with the INTERMW through simple REST API calls. On the other hand, in order to exploit the full potential of the system, a separate REST API endpoint that supports JSON-LD is provided.

A REST API callback has been provided. In the first phase of the project the results had to be obtained through a HTTP pull call, while now both approaches are implemented: HTTP pull and push. This allows adaptation to specific customer needs (performance, network topology).

Bridges for the following platforms have been either newly implemented or further developed: FIWARE, WSO2, Seams2. They are mostly related to use-case scenarios in the INTER-LogP Pilot. Some fixes to UniversAAL and BodyCloud bridges used in the INTER-Health scenario have been delivered also.

As what regards bridge development in Open Calls, they will be reported in relevant WP6 reports. Support has been provided to the following projects: *INTER-OM2M*, *SensiNact*, and *Semantic Middleware (ITIA-CNR)*.

The provision of documentation, examples and testing procedures for the development of Bridges have been identified as a critical task in order to achieve a wide coverage of connected platforms. For this reason, through continuous support to Open Calls and Pilots developers, the documentation and code has been optimized in order to help bridge developers as much as possible.

INTERMW is provided as Docker image, in-line with cross-layer efforts to unify the deployment architecture.

3.3.1 Release features

The V2 release provides core functionalities related to facilitation of interoperability among IoT Middleware platforms, as well as the provision of a common abstraction layer to provide access to platform features and information.

The MW2MW V2 release provides the following features:

- *Common ontology.* INTERMW uses the common INTER-IoT ontology (GOIoT) to represent all messages routed through the system.
- *Middleware abstraction.* Common abstraction layer unifies the view on all interconnected platforms, devices and services.
 - *Device registry.* The MW2MW solution maintains a registry of all devices maintained by platforms attached to it. The registry contains meta-information about devices. This allows the implementation of an efficient querying mechanism and seamless operations and implementation of additional services across platforms.

- *Discovery mechanism.* Maintenance of the Device registry is not a trivial task, as there are several approaches utilized by IoT platforms to provide meta-data about attached devices. MW2MW implements several discovery strategies that can be used to populate the registry: full-query at regular time intervals, difference query at regular time intervals or, with more advanced IoT platform implementations registry updates with callbacks.
- *Actuation and Observations.* MW2MW supports actuation and subscription to observations as core IoT platform functionalities.
- *Virtual devices management.* Virtual devices management (create, update, delete) is implemented for platforms that support this functionality.
- *MW2MW REST API.* Implementation of a REST API interface further extends the usability of this abstraction layer by exposing this functionality through a widely used technology.
 - *Results delivery.* Clients can use either a pull or push (URL callback) strategy to get results from the MW2MW layer. The set of functionality provided is same, the choice may depend on client's technical and organisational constraints, like the network topology, availability of REST servers or security policy.
 - *Data format.* Requests and results may be provided in either a simple JSON format, that fulfils the most of the basic user requirements, or in the more complex JSON-LD format that also offers a richer set of functions and full semantic interoperability.
- *MW2MW open architecture for the development of IoT platform bridges.*
 - *Common Java interface.* MW2MW provides a bridge interface that defines common bridge features that have to be implemented: subscriptions, actuations, virtual devices management and discovery. Java annotations are used in combination Java reflection mechanisms in order to dynamically load bridges at runtime.
 - *Syntactic conversion.* One important step in bridges development is the implementation of a syntactic translator to/from platform-specific format and RDF. Generic syntactic translators with examples and some common formats are provided, but in principle a new translator should be provided for each IoT platform type.
 - *Semantic translation.* Semantic translation is performed by the IPSM (described in Section 3.5) module that is bundled with MW2MW deployments.
 - *Unit and integration tests.* A series of unit and integration tests has been provided in order to facilitate the development of bridges. A series of test API calls can be generated in order to test bridge implementations.
 - *Examples and documentation.* As described in Section 2.2 documentation and examples are provided for INTERMW deployments, REST API usage and further developments.
- *Security.* Security has been implemented at two levels:
 - *Integration with API/identity managers.* Authentication and authorization features are provided through integration with the REST API Manager and Identity Manager.
 - *Platform security.* Platform security is a responsibility of bridge developers. In principle, authentication information can be passed through platform registration messages.

- *Dockerized deployment.* The MW2MW core, Parliament™ Triple Store and RabbitMQ can be deployed through a single Compose script. This facilitates the deployment and takes care about correct dependencies among components. An additional compose script for the deployment of both IPSM and MW2MW is provided as well to support simultaneous deployment of both INTER-Layer components.

3.3.2 Extensibility

The main extensibility feature of MW2MW is the development of bridges for new IoT platform types. For this purpose, particular attention has been provided to the definition of unit and integration tests, testing datasets and other validation tools. Detailed development documentation is provided as well.

MW2MW has a build-in abstraction of message broker functionalities, so that a message broker can be replaced in case of customer-specific requirements. However, testing and evaluation has been performed only with RabbitMQ.

The system also supports the addition of new service implementations. For example, functionalities like roaming of devices or updating virtual devices among different IoT platforms could be developed.

3.3.3 Release and distribution plan

The MW2MW component is going to be released open source under the Apache 2.0 license. This however, does not prevent bridge and service developers from releasing their modules under different licensing schemas to accommodate their particular business models.

The following MW2MW releases have been planned:

- *Closing of WP3 activities.* As stated previously, we refer to the release in M30 as V2 of MW2MW. This is the main feature release and provides a full implementation of MW2MW core functionalities as planned for the execution of task T3.3. This release is internal, thus available to project partners and Open Call projects.
- *License-ready release.* This release will contain all licensing information and will be ready for public distribution.
- *Integration-improved release.* This release will take into account lessons learned during the Integration and pilot deployment tasks (WP6).
- *Evaluation-improved release.* This release will take into account the results of the technical evaluation results (D7.2) and related improvements to meet the required technical standards.

3.4 AS2AS solution

The architecture, components, technologies and procedures that establish the interoperability solution were described in deliverables D3.1 and D3.2. The interoperability between services and applications is based on the Flow Based Programming paradigm. An special effort was made to define the steps needed by developers to adapt their IoT platform services in order to be used by the core

solution. In addition, examples of accessible services and interoperability flows between services of IoT platforms were explained.

The progress since the last deliverable implies a final version of Flow Repository and Node Repository. The aim is to provide new functionalities to store, register, describe and access to the developed nodes and designed flows. Offering users the access to a catalogue of IoT Platform services and composed services. New internal functional nodes have been created to provide facilities for interoperability between services. Furthermore, an improvement in the catalog of services available including new nodes and flows that involve new platforms in order to achieve the KPI defined at the beginning of the evaluation process.

The other relevant improvement since D3.2 is the integration of the interoperability solution with INTER-FW. For that reason, to achieve success in the integration process, several elements have been developed. Firstly, layer management mechanisms to allow the management of running instances, including interaction with APIs, Docker and Node-RED. Secondly, the integration of the node repository and the flow repository to facilitate the functionalities of registry, cataloging and discovery. Thirdly, offer to users the elements in a graphical way, to develop, compose and reuse their own services and the created composed services inside the Framework. Finally, develop a full documentation about this interoperability solution.

3.4.1 Release features

The current AS2AS release provides the following features in order to provide a layer of abstraction to perform the interoperability between IoT platform services. As indicated in previous deliverables the main elements of the interoperability solution are the nodes and the flows. The internal components are focused on the access, design and interaction with these key elements.

Components:

- *Modeller*: An available graphical environment to design the service composition. There is an instance of the GUI available for each instance of the core interoperability solution. It is a web environment that allows the graphical creation of flows through the nodes that are available for the instance. Once the design process is complete, the flow is stored in a JSON file.
- *Orchestrator*: The element in charge of validating and executing the JSON file designed by the modeller. It is responsible to make the calls to services and send messages between the nodes. It is the component in charge of keep running the flow and it will send an error message in case there is a problem.
- *Node Registry*: Component to store in a registry the description, configuration and files that define and implement the access to an IoT Platform Services.
- *Flow Registry*: Component to store in a registry the description, configuration and the JSON file that define and implement an interoperability flow.
- *Node Repository*: It provides access to a catalogue of available services of the IoT platforms in order to provide the access to a determinate service through a running instance of the interoperability solution.
- *Flow repository*: It provides access to an available catalog of the flows created, in order to deploy new instances and reuse it.

- *Instance Manager*: Manage the available instances of the solution of each user. It can start, stop and interact with each running instance of the solution. It manage the relation with the API of the core solution and the Docker containers.

Nodes:

- *IoT Platforms Nodes*: They are composed by several HTML and JS files that include the code to access to functionalities of the IoT platform service and provide a graphical interface to configure the access to the service.
- *Translation nodes*: These nodes offer functionalities to perform the translation the messages that are going to go through the IoT Platform services of different platforms. Examples of tasks that perform the node are like syntactic translation, summarize the information or readapt the information to a determinate format. It can be generic, for example a function to parse JSON elements, or specific, for example, adapt a JSON element in Orion format to a plain text format.
- *Functional Nodes*: These nodes execute internally a set of functionalities to facilitate the interoperability between nodes. For example, split information, counter, call to a process, provide order in the execution, listen in a determinate URL waiting for information, create API calls with a determinate body, show information in a dashboard, show information in a map, etc.

Flows:

- *Predefined Flows*: Flows created by users and developers that are stored and can be re-used in another context or domain, with minimal changes.
- *API Flows*: Flow that starts using an HTTP request and ends with an HTTP response. In order to be able to make calls to the flow using the API and receive the response in the body. Using this flow means it is not necessary to use the graphical UI to work with the flow, because you can introduce parameters in the call.

Support functions:

- *Semantic Translation*: A set of nodes dedicated for interaction with IPSM is offered. They allow the user to send messages to any instance of IPSM for translation, and receive translated messages from IPSM. The assumption is that input and output messages have to be in RDF, specifically in JSON-LD message format.
- *Swagger adaptation*: Swagger nodes allow the user to manage an API description, save and use it, without knowing the code to create it. These nodes allow the user to create an API definition, by defining methods and related parameters, responses and supported MIME through a specific node GUI. In the same time, it is possible to test the created API definition. Furthermore, it allows the user to download an API definition and use it. The node classifies the API in different resources and for each resource it is possible to select one of the available methods and test it. The parameters are provided to the specific method through the support of other nodes.

3.4.2 Extensibility

The main elements that facilitate the extension of this solution are the creation of new nodes and flows. Because, the nodes facilitate the use of new services and IoT applications belonging to differ-

ent platforms and the flows allow the creation of new composite services between IoT services and applications.

A service is represented by one or several nodes that access its functionalities. The creation of new nodes is done following the steps indicated in the documentation of this layer, which also appears in deliverables 3.1 and 3.2. These documents indicate how to analyze the services and create the wrapping that facilitates access to the services by the interoperability environment. Using this process, developers can create pairs of HTML and JavaScript files. The HTML files define the graphical interface to access to the service, properties and help information. The JavaScript files implement the functionalities. These files represent the service and must be stored somewhere. For that reason, services must be registered in the Node Repository to access to their properties and be available to interact with other services. In addition, the integration of this component in INTER-FW is important from the point of view of extensibility, because it facilitate the registration of new platforms and services in the interoperability solution.

A flow is a JSON file that represent the interaction of several services and functions to create a new IoT composite service. The creation of flows, is done using the designer component, a browser-based editor that facilitates the creation of new composite services. The execution of flow by the orchestrator generates the JSON file. The documentation explain the steps to generate new composed services, providing examples and recommendations. These flows can be reused and adapted to be used in other domains or locations. The flows are stored and described in the Flow repository. In addition, the integration of this component in INTER-FW facilitate the access to the new services created and its information, the easy execution and the possibility of reuse.

The offered nodes to facilitate the composition offer advantages from the point of view of the extension of the solution. For example, nodes that convert formats or adapt the information offered by a determinate service. These nodes are available to users and programmers to facilitate the interaction between the services in order to create new flows in an easy way. Swagger nodes can help to support the extensibility of the solution speeding-up the porting of services and applications within the core solution thanks to the knowledge of the already created API definition.

Finally, the integration of the Instance Manager with INTER-FW facilitates the deployment of new running instances and the management of the available instances. This facilitate the extensibility, because it is an easy way to start to create new composed services.

3.4.3 Release and distribution plan

The AS2AS open source product is going to be released under the Apache 2.0 license. It is a permissive license whose main conditions require preservation of copyright and license notices. Contributors provide an express grant of patent rights. Licensed works, modifications, and larger works may be distributed under different terms and without source code.

The following releases are planned:

- *Closing of WP3 activities.* Full implementation of core functionalities as planned for the execution of tasks T3.4 and it has been integrated in INTER-FW. It offers an internal release, available to project partners and open call projects.
- *License-ready release.* It contains all licensing information and it is ready for public distribution.

3.5 DS2DS solution

In previous versions of this deliverable (D3.1 and D3.2) the architecture, components, functionalities and APIs were described. This description is up-to-date. Since D3.2 the work has been focused on further testing and fixing bugs / adding extensions in the semantic translation functionality. This has been done in line with creation of new alignments for pilot applications and for open call projects. As a result, the set of prepared alignments is available as sample data for evaluating and testing IPSM.

Besides previously supported channels-based interface for semantic translation (Apache Kafka communication infrastructure using publish-subscribe mode), additional REST-based interface has been added to perform translations in synchronous mode. Request message contains RDF graph to be translated as a sequence of alignment identifiers that should be applied to the graph. Moreover, administrative interfaces for logging configuration have been added.

Integration with external functions called from a transformation included in an alignment cell has been further tested. Work was also been devoted to extending and updating complimentary IPSM Dashboard web application that provides a user-friendly interface for IPSM configuration and testing. The application supports semantic translation with custom sequence of alignments.

A set of installation instructions has been published for IPSM Docker deployment. Additional Docker deployment for IPSM Dashboard application has been prepared.

3.5.1 Release features

List of features for the first IPSM release:

- Semantic translation engine with alignments compiler
- REST API for configuration – alignments and channel management
- Semantic translation in publish-subscribe mode – translation channels defined in communication infrastructure that have topics for consuming and publishing messages with RDF graph; input messages in JSON-LD format with *payload* graph that will be translated, output message in JSON-LD
- REST API for synchronous semantic translation based on a request with RDF graph and sequence of alignment identifiers
- Compatibility with IPSM Alignment Format - alignments need to be expressed in the proposed format (compatible with Alignment API Format) to be consumed by IPSM
- Support for external functions execution from transformations defined in the alignment
- Documentation and a set of sample alignments
- Dockerized deployment with single Docker Compose script for IPSM
- Docker deployment of IPSM Dashboard application

3.5.2 Extensibility

IPSM is an independent component for performing semantic translation based on alignments. The alignment compiling and application functionalities are independent from the deployment environment and can be configured with any alignments in IPSM Alignment Format. Therefore, extending of IPSM applicability can be done by preparing new alignments. As a result, more translation scenarios can be supported. Note that, alignments consist of cells that can be included/excluded from alignment cells execution sequence in a specific order. As a result, one can define alignments that are reusable.

IPSM supports calling of custom functions as part of a transformation defined in an alignment cell. Consequently, another method to extend IPSM is to implement own libraries with functions that should be added to IPSM execution classpath to be visible to alignments applicator.

Finally, the IPSM exposes a REST API so it can be easily used as a component by other applications.

3.5.3 Release and distribution plan

IPSM is going to be released open source under Apache 2.0 license. The following releases are planned:

- *Closing WP3 activities* - full implementation of core functionalities as planned for the execution of T3.5. Internal release, available to project partners and open call projects.
- *License-ready release* - containing all licensing information and ready for public distribution.
- *Integration-improved release* - taking into account lessons learned during the Integration and pilot deployment tasks (WP6).
- *Evaluation-improved release* - taking into account the results of the technical evaluation results (D7.2) and related improvements to meet the required technical standards.

3.6 Cross-Layer solution

Once layered components reached a satisfactory level of maturity, the next step was the identification of common Cross-Layer issues and interaction among layers. Cross-Layer is focused in the aspects that are left out of the scope of a specific layer. These aspects can be defined as transverse elements that affects more than one layer and can be divided in three main areas:

- Layer security integration
- Layer Interactions
- Virtualization and Clusterization of layers

Deliverable 3.2 had the purpose of showing the analysis, design and documentation of the work planned for each area. For that reason, the progress since D3.2 consists in the implementation and integration of the work described in that deliverable.

The following summary list of features correspond to the elements described in Deliverable 3.2, that currently are implemented:

3.6.1 Layer security integration

The main objective was to develop a single security entity for user authentication, access control and identity management as well as securing the access to each layer API under this authentication system plus data encryption. This work has been done together with WP4 efforts since this process covers the whole INTER-API, where each layer exposes its own API.

This solution has been achieved using WSO2's Identity Server and WSO2 API Manager through XACML to define the policies. These policies can cover a variety of different security use cases, for example, filtering an API call by parameters and granting access to specific roles.

3.6.2 Layer Interactions

After the study and definition of the interesting interactions existent between the layers, these are the features developed:

- D2D <> N2N: SDN module for the gateway has been developed. Although it won't be included under the first release.
- D2D <> MW2MW: Through AS2AS nodes (see below). No dedicated component has been developed.
- MW2MW <> DS2DS: IPSM is the component that provides semantic translation in MW2MW.
- D2D <> AS2AS: Creation of a set of nodes compatible with the AS2AS environment to perform control and management of IoT Devices. The functions supported are: Device Status, Device Start, Device Stop, Read device, Write device.
- MW2MW <> AS2AS: Implementation of a MW2MW subscription node. It provides a series of observations from a set of sensors. It can provide information to consumer nodes, mainly, the IoT platform services.
- AS2AS <> DS2DS: It offer a set of nodes dedicated to interaction with IPSM. They allow to send messages to any instance of IPSM for translation, and receive translated messages from IPSM.

3.6.3 Virtualization and Clusterization of layers

Virtualization of the components of each layer simplifies a lot their deployment. Furthermore, clusterization offers an access point to facilitate a centralized management of the virtualized components. The technology selected to perform this virtualization is Docker, through the use of its containers. The layers using this tool involve: D2D, MW2MW, AS2AS and DS2DS since network restrictions of Docker limits the functionalities of an SDN controller for N2N. Those Docker deployments can be consulted in Tables 3 to 7. The main features offered are:

- Dockerization of D2D, MW2MW, AS2AS and DS2DS components.
- Creation of Docker compose files to define and run multi-container Docker applications that includes all the dockerized components that belong to a layer. The purpose is to offer a complete and functional deployment of all the elements of a layer.

- Private Docker registry to store the INTER-IoT Docker images.
- Integration of Docker tools to manage the containers: Docker Swarm and Portainer.

3.6.4 Cross-Layer as a transversal component

Cross-Layer is a transversal component supportive of the other layers, therefore, the main objective is not to provide mechanisms to facilitate the extensibility of the solution, the purpose is to take care of those needs that are not contemplated within the focus of action of the other layers.

Moreover, as Cross-Layer is not a component itself, the release and distribution plan is given by the release specification of each layer, tightly coupled to its characteristic. Then, when a new version of any of the aforementioned layer is delivered, means that, additionally, a previous update of its cross-layer features has been performed.

4 Ethics

4.1 Introduction

Ethics is a central consideration to all INTER-IoT planning and development. As requested at the interim review, an ethical advisory board has been established. This board, within INTER-IoT, continuously reviews ethical issues. The aim of the committee is to ensure that ethical considerations and issues are addressed in the conduct of the research and development work undertaken within the project. The committee seeks to support and encourage the process of ethically conducted research to maintain the safety and well-being of participants and researchers to promote ethical values.

4.2 Ethics and INTER-Layer

INTER-Layer is designed to enable the interoperability of existing IoT systems at different levels. Ethical considerations must be taken into account from two different perspectives: From the systems being connected, and from the inner workings of each layer that provides interoperability. INTER-Layer cannot be responsible for the proper ethical handling of data and security within each connected IoT system, but it can and must assure that such handling in each system will not diminish their ethical considerations by connecting to INTER-Layer. In addition, the inner logic of the components in each layer must comply with the proper ethical handling of data and security, as per the requirements and principles enumerated in the following sections.

4.2.1 Data types

Primary focus of the ethical review of data management focuses on personal data and sensitive personal data. Personal data means data which relate to a living individual who can be identified –

- (a) from those data
- (b) from those data and other information which is in the possession of, or is likely to come into the possession of, the data controller, and includes any expression of opinion about the individual and any indication of the intentions of the data controller or any other person in respect of the individual.

Sensitive personal data means personal data consisting of information as to -

- (a) the racial or ethnic origin of the data subject,
-

- (b) his political opinions,
- (c) his religious beliefs or other beliefs of a similar nature,
- (d) whether he is a member of a trade union,
- (e) his physical or mental health or condition,
- (f) his sexual life,
- (g) the commission or alleged commission by him of any offense, or
- (h) any proceedings for any offence committed or alleged to have been committed by him, the disposal of such proceedings or the sentence of any court in such proceedings.

It is possible that INTER-IoT and INTER-Layer will be used during processing of these types of data, so appropriate controls have to be built into the layer components to enable systems to do this ethically by conforming to the General Data Protection Regulation. This is achieved by assuring the proper security of the communications and data handling within the components of each layer and across layers, as per the security considerations described in previous sections, encompassing the common aspects of integrity, privacy, authentication, authorization of information systems, and the other, more "IoT-specific" ones of pseudonymity, autonomous communication, and semantic querying.

4.2.2 Requirements for ethical data processing

The GDPR requires adherence to 8 principles:

1. Personal data shall be fairly and lawfully processed as defined in the data protection act.
2. Personal data shall be obtained only for one or more specified and lawful purposes, and shall not be further processed in any manner incompatible with that purpose or those purposes.
3. Personal data shall be adequate, relevant and not excessive in relation to the purpose or purposes for which they are processed.
4. Personal data shall be accurate and, where necessary, kept up to date.
5. Personal data processed for any purpose or purposes shall not be kept for longer than is necessary for that purpose or those purposes.
6. Personal data shall be processed in accordance with the rights of data subjects under the data protection act.
7. Appropriate technical and organisational measures shall be taken against unauthorised or unlawful processing of personal data and against accidental loss or destruction of, or damage to, personal data.
8. Personal data shall not be transferred to a country or territory outside the European Economic Area unless that country or territory ensures an adequate level of protection for the rights and freedoms of data subjects in relation to the processing of personal data.

During the process of implementation of each component the developers have been made aware of the above principles, to not introduce unwanted features, workarounds, hot fixes or "hacks" that interfere with them, even if in a temporary fashion, as long as there is the chance that these are

utilized in the pilots real-life deployments. Developers of each module, component and feature of the layers have committed to develop the software so that it does not interpret, handle and/or store the data payload being transferred within the system, so they do not hinder the above principles. Whenever data is indeed interpreted, handled or stored, it only refers, represents or deals with entities that are not individuals (e.g. Sensor devices) and that this data cannot be used to extract or infer personal data about individuals.

5 Conclusions

INTER-IoT envisions a true Internet-of-Things without silos and barriers at any level. From a scientific and engineering perspective, INTER-IoT aims at solutions that allow any device, platform, middleware and application to interact with any other counterpart regardless of their design, architecture or implementation differences. WP3 gathers these integrated mechanisms in a coherent generic and viable solution for interoperability of IoT.

At the core of this solution, there are two concepts that make the solution viable:

- Interoperability in the virtual world is easier than in its physical counterpart: the physical components should be minimized to a minimum required moving much of functionality to the virtual one.
- Extensibility is the only way to ensure compatibility of future IoT systems with legacy and older version systems: physical and virtual components should be able to be added/removed even if their design, architecture and implementation is not yet known.

Based on these fundamental principles, WP3 delivers the following achievements:

- Devices with different network interfaces or other physical functionalities can interact via extensible managed gateways. Each gateway has a minimal physical part and a managed virtual remote counterpart.
- Global scalability of IoT is ensured with software-defined radio and software-defined networks. Though zero configuration is supported, QoS tuning is possible to allow differentiated services.
- Facilitation of interoperability among IoT Middleware platforms, as well as the provision of a common abstraction layer to provide access to platform's features and information.
- Similar features are ported to application and services to ensure interoperability among heterogeneous IoT Platform applications and services.
- A common interpretation of data and information among different IoT systems and heterogeneous data sources, achieving semantic interoperability.

Finally, the above mentioned achievements are linked together thanks to the cross-layer solution provided by WP3. This solution enables interaction between layers, but also security and virtualization through docker to simplify deployments and re-usability. Among all layers, ethics is also considered. INTER-IoT addresses ethical issues in this deliverable and aligns with the EU General Data Protection Regulation (GDPR).

Overall, this document reports the final advancements leading to the release of INTER-Layer components. This document demonstrates that the WP3 is ready for open-source release and that all

INTER-Layer components have reached a state where they can be separately or jointly tested, released, and used. All different aspects of interoperability are addressed, as each layer supports interoperability of its components, and all layers inter-operate smoothly with help of the the cross-layer solution and been accessed and managed from a single point that INTER-FW offers. From a technical point of view, WP3 adopted an iterative approach together with all other work packages, leading to seamless integration with all other INTER-IoT aspects of the project, such as the RA implementation carried out in tasks 4.1 and 4.2, the unification of the API for easy accessibility and integration with other solutions, the INTER-IoT framework, the elaboration of a methodology to create interoperability between IoT solutions and the use case testing performed in WP6.

This deliverable is the third version of a series of three (i.e. preceded by D3.1 and D3.2). For that reason, from now on, all the improvements perform over the components developed in WP3 will be tracked through the official INTER-IoT web documentation.