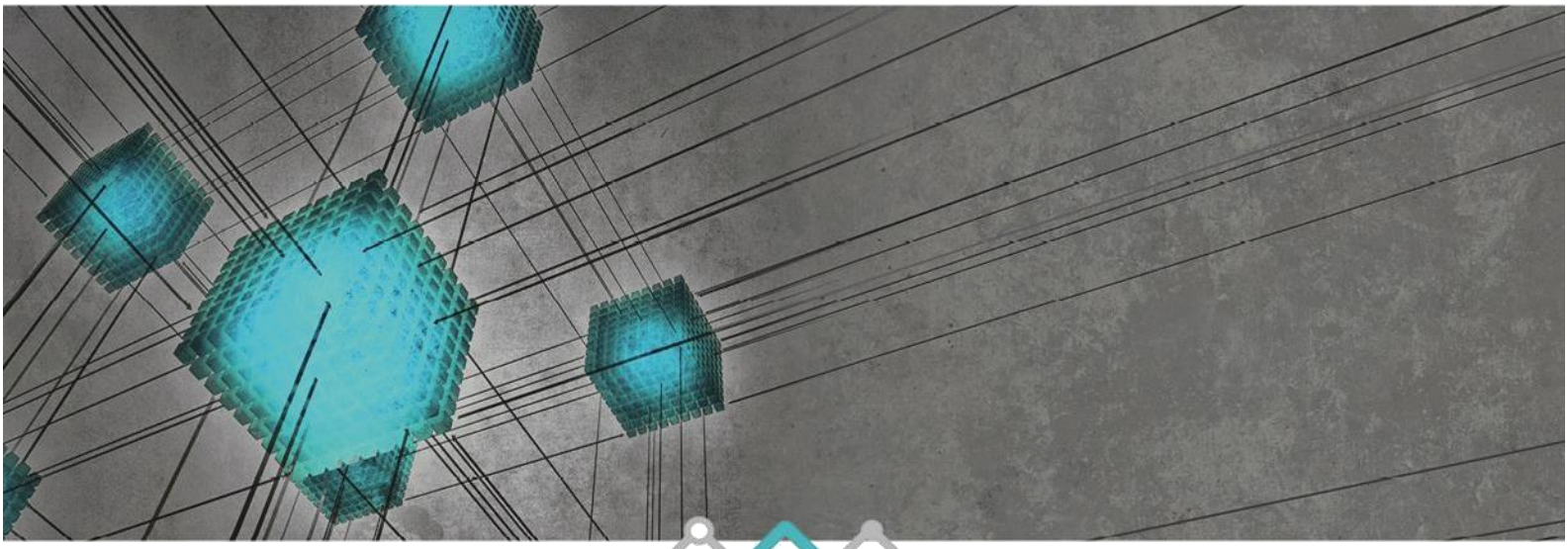

interiot

INTEROPERABILITY
OF HETEREOGENEUS
IOT PLATFORMS.

D5.2

INTER-Meth: Full-fledged Methodology for IoT Platforms Integration

December 2017

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 will enable different IoT scenarios. As well, reuse and integration of existing and future IoT systems will be facilitated, creating a defacto 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.

INTER-IoT

INTER-METH: Full-fledged Methodology for IoT Platforms Integration

Version: Final

Security: Public

31 December. 2017

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

Disclaimer

Executive Summary

INTER-METH is an engineering methodology that aims at supporting the integration process of heterogeneous IoT platforms to (i) obtain interoperability among them and (ii) allow implementation and deployment of IoT applications on top of them. It is widely recognized that using an engineering methodology is fundamental in any engineering application domain (e.g. software engineering, codesign hardware/software, civil engineering, etc). Moreover, the manual and non-systematic application of complex techniques, methods and frameworks would very likely lead to an increase of the degree of errors during integration. Thus, INTER-METH has the objective to avoid such errors. Specifically, INTER-METH is based on a process that defined and offers the following phases (or functionalities of a process):

- Analysis phase will support the definition the IoT platform integration requirements (both functional and non-functional).
- Design phase will produce the design of the integration in terms of design artifacts (e.g. diagrams) of layer interoperability infrastructures and related interfaces (see INTER-LAYER), and INTER-FW programming and management patterns, to fulfill the elicited requirements.
- Implementation phase focuses on driving the implementation of the design work-product to obtain the full-working (hardware and/or software implemented) system.
- Deployment phase involves the support to the operating set-up and configuration of the integrated IoT platform.
- Testing phase will define the performance evaluation tests to validate the integrated platform according to the functional and non-functional requirements.
- Maintenance phase will manage the upgrade and evolution of the integrated system.

Each phase produces work-products that are inputs for the successive phase(s). INTER-METH will be supported by a CASE (Computer Aided Software Engineering) tool (described in D5.3) that will help automating each aforementioned phase of the integration process and will specifically provide the following functionalities: (i) integration guidelines management, (ii) graphical facilities, (iii) engineering patterns automation, and (iv) project data repositories.

This document is organized as follows:

Section 1: provides the aim of the D5.2 deliverable, the definition of basic terms extensively used throughout the document and the main contributions of the T5.2 activity.

Section 2: analyses the state-of-the-art (SOTA) in depth, by dividing it in general-purpose SOTA and IoT system integration-specific SOTA.

Section 3: defines functionalities, requirements, uses cases and scenarios of INTER-METH.

Section 4: defines the abstract INTER-METH methodology for IoT platform integration; “abstract” means applicable to any IoT system integration approach (not only to INTER-IoT).

Section 5: defines the INTER-METH methodology based on INTER-IoT products (especially, INTER-LAYER and INTER-FW) as well as the INTER-PATTERNS, defined in D5.1.

Section 6: discusses ethics related to INTER-METH methodology.

Section 7: lists the bibliography.

Appendix A: provides a short overview of SPEM (Software & Systems Process Engineering Metamodel) used to model INTER-METH.

Appendix B: sketches the ontology alignment procedure used in INTER-IoT/METH.

.

List of Authors

Organisation	Authors	Main organisations' contributions
UNICAL (Task Leader)	G. Fortino, W. Russo, R. Gravina	SOTA analysis, Abstract INTER-METH, Concrete INTER-METH: Analysis, Design, Implementation and Maintenance phases
SRIPAS	K. Wasielewska-Michniewska, R. Tkaczyk	SOTA analysis, Concrete INTER-METH: ontology alignment (supporting the Implementation Phase).
PRO	M. A. Llorente	SOTA analysis, Concrete INTER-METH: Deployment phase.
VPF	P. Gimenez Salazar	SOTA analysis, Requirements and Scenarios.
XLAB	F. Fuart, M. Markovic, M. Gorenc Novak	SOTA analysis, Concrete INTER-METH: Testing. Internal review.
UPV	R. González Usach, Carlos E. Palau	SOTA analysis, Concrete INTER-METH: Deployment. Final Review

Change control datasheet

Version	Changes	Chapters	Pages
0.8	First version of the document	All	45
0.9	Internal review	All	113
1.0	Final version of the document	All	108

Contents

Executive Summary 3
List of Authors 4
Change control datasheet 5
Contents 6
List of Figures 9
List of Tables 11
Acronyms 12
1 Introduction 14
1.1 Aim of the document 14
1.2 Definitions and terminology 14
1.3 Summary of Workpackage Contributions 15
2 State-of-the-Art (SotA) Analysis 16
2.1 Software Engineering Methodologies 16
2.1.1 Waterfall Methodology 16
2.1.2 Volere Methodology 18
2.1.3 Agent-oriented Methodologies 20
2.1.4 IoT Methodologies 26
2.1.5 Agile Methodologies 29
2.1.6 CMMI (Capability Maturity Model Integration) 33
2.1.7 Overall Analysis 34
2.2 Integration Techniques and Methodologies for IoT Systems and Systems of Systems 35
2.2.1 IoT Systems Integration: Model-driven Interoperability 35
2.2.2 System of Systems (SoS) Integration 36
2.2.3 Telecommunication Systems Integration 39
2.2.4 Systems Integration Best Practices 40
2.2.5 IOT-A Methodology 41
2.2.6 Overall Analysis 46
3 INTER-METH Functionalities, Requirements, Use Cases, and Scenarios 47
3.1 Functionalities 47
3.2 Requirements 47
3.3 Use Cases 53
3.3.1 IoT Platform Integration Requirements Analysis (IM1) 53
3.3.2 IoT Platforms Integration Design (IM2) 54
3.3.3 IoT Platforms Integration Implementation (IM3) 54
3.3.4 IoT Platforms Integration Maintenance (IM4) 55

3.4 Scenarios 56
3.4.1 Heterogeneous Platforms Methodology-driven Integration 56
3.4.2 Re-engineering Integrated IoT platforms 57
4 INTER-METH Abstract Process 59
4.1 Phase 1: Analysis of Integration Requirements 60
4.1.1 Activities 60
4.1.2 Work Products 62
4.2 Phase 2: Design of the Systems Integration 62
4.2.1 Activities 63
4.2.2 Work Products 64
4.3 Phase 3: Implementation of the Systems Integration 65
4.3.1 Activities 65
4.3.2 Work Products 66
4.4 Phase 4: Deployment of the Integrated Platform 66
4.4.1 Activities 67
4.4.2 Work Products 68
4.5 Phase 5: Testing of the Integrated Platform 68
4.5.1 Activities 69
4.5.2 Work Products 70
4.6 Phase 6: Maintenance of the Integrated Platform 70
4.6.1 Activities 71
4.6.2 Work Products 72
5 INTER-METH based on INTER-IoT 73
5.1 Phase 1: Analysis 73
5.1.1 Activities 73
5.1.2 Work Products 77
5.2 Phase 2: Design 78
5.2.1 Activities 79
5.2.2 Work Products 81
5.3 Phase 3: Implementation 81
5.3.1 Activities 82
5.3.2 Work Products 83
5.4 Phase 4: Deployment 84
5.4.1 Activities 84
5.4.2 Work Products 88
5.5 Phase 5: Testing 89
5.5.1 Activities 89

5.5.2 Work Products 91
5.6 Phase 6: Maintenance 91
5.6.1 Activities 91
5.6.2 Work Products 93
6 On Methodology and Ethics 94
7 References 96
Appendix A: SPEM in a nutshell 101
Appendix B: Semantic interoperability in the INTER-IoT 104
B.1 Semantic translation as an integration method 104
B.2 Establishing semantic interoperability 105

List of Figures

Figure 2.1.1: Depiction of the six steps in the classic Waterfall Methodology. 17
Figure 2.2.1: Depiction of the six steps in the classic Waterfall Methodology. 19
Figure 2.3.1: Iterative process for prototyping ELDA-based MASs. 25
Figure 2.5.2: Relations between Agile, Lean and Scrum concepts 30
Figure 2.5.2: Kanban board. Source: https://leankit.com/learn/kanban/kanban-board/ 32
Figure 2.6.1: CMMI process 33
Figure 2.8.1: IoT-A: from Reference Models to Software Architecture. 42
Figure 2.8.2: IoT-A architecture generation process. 43
Figure 2.8.3: IoT-A requirements process. 44
Figure 2.8.4: IoT-A workflow for requirements implementation. 44
Figure 2.8.5: IoT-A Methodology correlating Perspectives and Views. 45
Figure 4.1: INTER-METH Abstract Process Schema 59
Figure 4.1.1: The Analysis phase described in terms of activities, roles, and work products 61
Figure 4.1.2: The workflow of tasks of the Requirement Analysis activity 62
Figure 4.2.1: The Design phase described in terms of activities, roles, and work products 63
Figure 4.2.2: The workflow of tasks of the Design Integration activity 64
Figure 4.3.1: The Implementation phase described in terms of activities, roles, and work products 65
Figure 4.3.2: The workflow of tasks of the Integration Implementation activity 66
Figure 4.4.1: The Deployment phase described in terms of activities, roles, and work products 67
Figure 4.4.2: The workflow of tasks of the Integrated Platform Deployment activity 68
Figure 4.5.1: The Testing phase described in terms of activities, roles, and work products 69
Figure 4.5.2: The workflow of tasks of the Integrated Platform Testing activity 70
Figure 4.6.1: The Maintenance phase described in terms of activities, roles, and work products 71
Figure 4.6.2: The workflow of tasks of the Integrated Platform Maintenance activity 72
Figure 5.1.1: INTER-IoT Reference Architecture schema. 74
Figure 5.1.2: The INTER-LAYER approach schema. 75
Figure 5.1.3: The INTER-IoT-based Analysis phase described in terms of activities, roles, and work products 76
Figure 5.1.4: The workflow of tasks of the INTER-IoT-based Requirement Analysis activity 77
Figure 5.1.5: The Metamodel of INTER-GOM 77
Figure 5.1.6: Refinement Process schemes in TROPOS and in INTER-METH 78
Figure 5.1.7: The UML Activity Diagram of INTER-GOM Production 78
Figure 5.2.1: The INTER-IoT-based Design phase described in terms of activities, roles, and work products 80
Figure 5.2.2: The workflow of tasks of the INTER-IoT-based Design Integration activity 80
Figure 5.3.1: The INTER-IoT-based Implementation phase described in terms of activities, roles, and work products 82
Figure 5.3.2: The workflow of tasks of the INTER-IoT-based Integration Implementation activity 83
Figure 5.3.3: Relationships among the different phases (in bold) and work products (in italic) 83
Figure 5.4.1: The INTER-IoT-based Deployment phase described in terms of activities, roles, and work products 88
Figure 5.4.2: The workflow of tasks of the INTER-IoT-based Deployment activity 88
Figure 5.5.1: The INTER-IoT-based Testing phase described in terms of activities, roles, and work products 90
Figure 5.5.2: The workflow of tasks of the INTER-IoT-based Testing activity 91
Figure 5.6.1: The Maintenance phase described in terms of activities, roles, and work products 92
Figure 5.6.2: The workflow of tasks of the Integrated Platform Maintenance activity 92
Figure A.1: The SPEM Process With Method Package Metamodel (from [SPEM08]) 105

Figure B.1: Creating alignment process…… ……107

List of Tables

Table 2.3.1: Comparison among simulation-based methodologies for MAS development..... 26
Table 2.4.1: System-Level requirements (SLR)..... 27
Table 2.4.2: Things-Level requirements (TLR)..... 27
Table 2.4.3: IoT Methodologies Comparison (Y=Yes, P=Partial, N=No)..... 28
Table 2.5.1: Pros and Cons of Agile Methodologies..... 32
Table 2.7.1: Analyzed Methodologies and their main characteristics..... 34
Table 2.8.1: Data and Control choices in SOS (Based on Table 1 from [1])..... 37
Table 2.8.2: Summary of template for information pattern matrix (Based on Table 4 from [1])..... 37
Table 2.9.1: Analyzed Methodologies and their main characteristics..... 46
Table 2.2.1: Main INTER-METH Requirements..... 47
Table 2.3.1: The main four INTER-METH Use Cases..... 53
Table 4.1.1: Tasks of the Requirements Analysis activity..... 61
Table 4.2.1: Tasks of the Integration Design activity..... 64
Table 4.3.1: Tasks of the Integration Implementation activity..... 66
Table 4.4.1: Tasks of the Integrated Platform Deployment activity..... 68
Table 4.5.1: Tasks of the Integrated Platform Testing activity..... 69
Table 4.6.1: Tasks of the Integrated Platform Maintenance activity..... 71
Table 5.2.1: INTER-IoT Design Patterns, their related INTER-IoT layer and inspiring Pattern Catalogue..... 81

Acronyms

Acronym	Definition
GA	Grant Agreement
WP	Work Product
AOSE	Agent-Oriented Software Engineering
IoT	Internet of Things
ISO/IEC	International Organization for Standardization (ISO) & the International Electrotechnical Commission (IEC)
SPEM	Software & Systems Process Engineering Metamodel Specification
API	Application Programming Interface
DTE	Data Terminal Equipment
SotA	State of the Art
UML	Unified Modeling Language
MaSE	Multiagent Systems Engineering
JDE	JACK Development Environment
PDT	Prometheus Design Tool
MAS	Multi-Agent System
EI	Electronic Institutions
JAMES	Java Based Agent Modeling Environment for DEVS-based Simulation
DEVS	Discrete Event Systems Specification
CaseLP	Complex Application Specification Environment Based on Logic Programming
SpiM	Stochastic PI-calculus Machine
MEASURE	Mission Effectiveness Analysis Simulator for Utility, Research and Evaluation
HLA	High Level Architecture
MDE	Model Driven Engineering
IDK	INGENIAS Development Kit
DAS	Distributed Agent System
Hw	Hardware
Sw	Software
SPM	Sprint Planning Meeting
CMMI	Capability Maturity Model Integration
ARC	Appraisal Requirements for CMMI
SCAMPI	Standard CMMI Appraisal Method for Process Improvement
SEI	Software Engineering Institute
RM-ODP	Reference Model of Open Distributed Processing

FSM	Finite State Machine
IP	Internet Protocol
REST	Representational State Transfer
SOA	Service Oriented Architecture
SoS	System of Systems
SSE&I	System of Systems Engineering and Integration
ESB	Enterprise Service Bus
SI	Systems Integration
SEIT	Systems Engineering, Integration and Test
SAS	System Architecture Skeleton
VDD	Version Description Document
ARM	Architecture Reference Model
CASE	Computer Aided Software Engineering
IM	INTER-Meth
GOM	Goal-Oriented Model
FG	Functional Group
R&D	Research & Development
CoI	Categories of Integration
QoS	Quality of Service
IP	Integration Point
UI	User Interface
DNS	Domain Name Service
IPSM	Inter Platform Semantic Mediator
AF	Alignment Format
SAT	Site Acceptance testing
FAT	Factory Acceptance Test
ISTQB	International Software testing Qualifications Board

1 Introduction

1.1 Aim of the document

The aim of the following document is the definition of INTER-METH based on the SPEM notation.

This document is realized within "T5.2: Definition of a Full-fledged Methodology for IoT Platforms Integration (INTER-METH)" task:

[from the Project GA]: "*In this task, we aim at creating a full-fledged methodology for the integration of IoT platforms so making them interoperable. The methodology will be based on an iterative process organized in iterable phases that will systematically incorporate and use all the outcomes of WP2, WP3, WP4 and Task 5.1. In particular, the integration process will consist of the following software engineering standard phases: (i) Requirements Analysis: in which, the outcomes will be a complete set of functional and non-functional requirements according to which IoT platforms (as obtained in WP2) will be integrated. Here, the functional and non-functional test use cases will be also defined; (ii) Design: which will produce a complete design of the IoT platforms integration at each layer and across the layers. The resulting design of the integrated interoperable IoT platforms will be compliant to the reference meta-architecture for interoperable IoT platforms defined in WP4-Task 4.1. The design will also include IoT platforms data integration according to the data metamodel developed in WP4-Task 4.2; (iii) Implementation: aimed at implementing the design object produced in the design phase. The implementation will be based on the INTER-FW framework APIs and tools, developed in WP4-Task 4.3-4.4; (iv) Deployment and Testing: the integrated IoT platform will be deployed and tested (from both functional and non-functional perspectives) according to the scenarios elicited in WP2; (v) Maintenance: this phase will support debug and evolution of the developed integrated platform.*"

1.2 Definitions and terminology

Term	Definition
Software development methodology	Also known as a system development methodology, software development life cycle, software development process, software process is a splitting of software development work into distinct phases (or stages) containing activities for better planning and management.
Software process	A structured set of activities required to develop a software system.
Process model	A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective.
Phase	Phase of a software process containing one or more activity. E.g. Specification phase – defining what the system should do.
Activity	Fundamental building blocks of processes, e.g. specifying a data model, designing a user interface, etc.
Telecommunication system / Network system	a communications system is a collection of individual communications networks, transmission systems, relay stations, tributary stations, and data terminal equipment (DTE) usually capable of interconnection and interoperation to form an integrated whole.
Internet of Things	it is a system of interrelated computing devices, mechanical and digital machines, objects, animals or people that are provided with unique identifiers and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction.

System Integration	In engineering, system integration is defined as the process of bringing together the component subsystems into one system and ensuring that the subsystems function together as a system. In information technology, systems integration is the process of linking together different computing systems and software applications physically or functionally, to act as a coordinated whole.

1.3 Summary of Workpackage Contributions

INTER-METH is, to the best of our knowledge, the first methodology for IoT system integration. According to our SotA analysis (see section 2), we can state that the lack of methodological approaches will slow down IoT interoperability even in presence of “many” technological solutions. The Key Concept is therefore to associate a methodological approach to a more technical solution. Thus, INTER-IoT will associate INTER-METH (including INTER-PATTERNS, described in D5.1, and driven by INTER-CASE, which will be fully described in D5.3) to INTER-LAYER and INTER-FW in order to allow an easier adoption of INTER-IoT technology by a System Integrator (specifically an IoT system integrator). This association will be implemented by keeping independent INTER-LAYER and INTER-FW from INTER-METH, while the INTER-IoT instantiation of INTER-METH will reuse specific results from INTER-LAYER and INTER-FW. Finally, INTER-CASE will make the application of INTER-METH more effective as it will provide semi-automatic support for its application. INTER-CASE will be documented in D5.3.

In particular, the contributions of this D5.2 deliverable are organized as follows:

- Section 2 analyzes the state-of-the-art in depth in order to identify characteristics of existing methodology that could be useful to support the definition of INTER-METH.
- Section 3 reports and discusses a set of functionalities, requirements, use cases, and scenarios for INTER-METH defined in WP3 and refined here.
- Section 4 fully defines the abstract INTER-METH methodology that could be re-used by whatever specific approach for IoT interoperability.
- Section 5 customizes the abstract INTER-METH for the INTER-IoT approach.

Finally, it is worth noting that INTER-METH will be exemplified in a real use case, referring to the INTER-HEALTH use case (see WP6), in D5.3 along with the implementation of INTER-CASE and its application. Thus, in D5.2 we only provide the methodological definition of INTER-METH.

2 State-of-the-Art (SotA) Analysis

The aim of this section is to provide background to the research and design of methodologies for interoperable IoT systems and their integration. State-of-the-art (SotA) analysis includes discussion of: (i) the definition of methodologies, (ii) relevance of the reviewed methodologies in the IoT domain, and (iii) characteristics of such methodologies useful for defining INTER-METH.

Specifically, they are organized in two main categories: (a) General-Purpose Software Engineering Methodologies (see section 2.1) and (b) More Specific Methodologies for System Integration (see Section 2.2).

For each category, we provide an overview and finally an overall analysis towards INTER-METH definition, i.e. the characteristics of surveyed methodologies useful to support the definition of INTER-METH.

2.1 Software Engineering Methodologies

In software engineering, a software development methodology[1] (also known as a system development methodology, software development life cycle, software development process, software process) is a splitting of software development work into distinct phases (or stages) containing activities with the intent of better planning and management. It is often considered a subset of the systems development life cycle. The methodology may include the pre-definition of specific deliverables and artefacts that are created and completed by a project team to develop or maintain an application.

Common methodologies include waterfall, prototyping, iterative and incremental development, spiral development, rapid application development, extreme programming and various types of agile methodologies. Some people consider a life-cycle "model" a more general term for a category of methodologies and a software development "process" a more specific term to refer to a specific process chosen by a specific organization. For example, there are many specific software development processes that fit the spiral life-cycle model.

2.1.1 Waterfall Methodology

The Waterfall development methodology [W1] is a step-by-step guide to the development of software (or possibly other) systems. Briefly, it focuses on gathering the features of the final product, designing it, and then implementing it following that design. The term depicts the idea that only once a higher step in the process is complete (full of water), it will spill its results in the following step below itself.

This engineering method is one of the first such methods used for the structured production of software solutions, and is still often used in several industries, though the so called ‘agile’ methods have displaced the Waterfall method in several areas, in particular in rapidly evolving systems and start-ups.

[1] https://en.wikipedia.org/wiki/Software_development_process

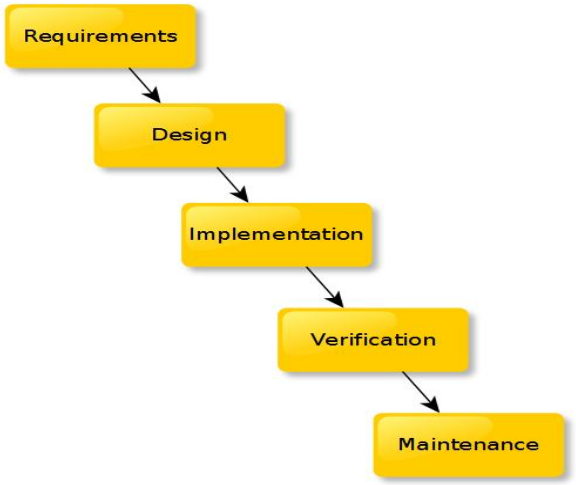

Figure 2.1.1: Depiction of the six steps in the classic Waterfall Methodology.

The 6 main steps of the classic Waterfall methodology (see Figure 2.1.1) are the following:

1. Requirements gathering
2. System design
3. Implementation
4. Verification
5. Maintenance

1. Requirements gathering

These steps refer to the gathering and documenting of the desired features, functional and nonfunctional requirements of the intended solution. It usually includes very detailed documents explaining use-cases and expected interactions between the users and the system, often in representations such as the UML standard.

2. System design

Following the complete set of requirements, an architectural system is designed in order to fulfil them in an efficient manner. Architectural patterns proven in practice to fulfil certain criteria are often used, and there is a lot of bibliography on the design of systems architectures for different nonfunctional requirements, such as the Enterprise Bus, Consumer/Producer and lately also patterns to cope with Big Data and elasticity.

System designs can be depicted by means of an Architecture description language, and different types of diagrams, such as the Zachman Framework, RM-ODP views, Enterprise Architecture views, which in turn use different types of UML diagrams to describe different aspects of the final solution, such as the:

- Logical View (using Class, Activity or State Diagrams),
- Development View (Component and Package diagrams),
- Process View (Activity and Concurrency diagrams),
- Physical View (Deployment and Topological diagrams)
- and sometimes a Use-Case View (Use-Case Diagrams)

Thus, a general architectural design can be enriched with clear descriptions of its components, their expected behaviour and planned deployment strategy.

3. Implementation

Making use of the selected programming languages, frameworks and deployment mechanisms, the implementation is responsible for creating the actual software solution based on the system architectural design. Different components of the final system can use very different technologies, and very complex systems can include deployments in different geographical locations.

There are different and complementary techniques to the implementation phase, such as Test-Driven Development, Contract-based implementation, Micro-services, etc. Regardless of the techniques used, the solution must be accompanied by documentation describing the behaviour of each component and functionalities of the expected deployment, and its source code must contain a clear description of its implementation (which will be particularly necessary during the Maintenance phase).

4. Verification

Even though the implementation phase already performs some software validation, usually in the format of unit-tests, in the Waterfall methodology the verification of the functional and nonfunctional requirements of the system is performed at the end of the implementation phase, when the entire system can be deployed and evaluated as a whole.

Integration testing and particularly System testing are performed on the final deployed solution to validate the correct functioning of the system as part of the Quality Assurance process. Finally, an Operational acceptance test against the defined use-cases corroborates that the system correctly performs the actions as described in the functional requirements, and the acceptance criteria for the nonfunctional requirements are met.

5. Maintenance

Once a system is verified and deploy into production, the maintenance phase begins. This phase includes not only fixing potential errors in the implementation, and might include the following (as per ISO/IEC 14764):

- Corrective maintenance: refers to the correction of behaviour not fulfilling the specification, often referred as “bug fixing”.
- Adaptive maintenance: are modifications to the system that adapts its behaviour to new circumstances or changes in the environment.
- Perfective maintenance: is performed in order to improve the maintainability of the code, or to improve the performance of the system related to nonfunctional requirements.
- Preventive maintenance: tries to solve non-critical operational faults that represent a potential hazard, particularly issues related to scalability and security.

Analysis

The waterfall model even though very simple, after enhancing, can be used to support INTER-Meth. In fact, in the GA, the proposed INTER-Meth process is based on an iterative version of the waterfall model, as the basic waterfall is too static.

2.1.2 Volere Methodology

The VOLERE methodology [RS12] helps to describe, formalize and track the project market analysis, requirements, use cases and scenarios in an explicit and unambiguous manner.

VOLERE has been used by thousands of organizations around the world in order to define, discover, communicate and manage all the necessary requirements for any type of system development (e.g.

software, hardware, commodities, services, organizational, etc.) VOLERE can be applied in almost all kinds of development environments, with any other development methods or with most requirements tools and modelling techniques. To produce accurate and unambiguous requirements, the VOLERE methodology uses techniques that are based on experience from worldwide business analysis projects, and are continually improved.

The VOLERE methodology provides several templates to deal with the different techniques and activities that it includes. In a quick view, the VOLERE Requirement Process that this methodology suggests can be summarised as in Figure 2.2.1.

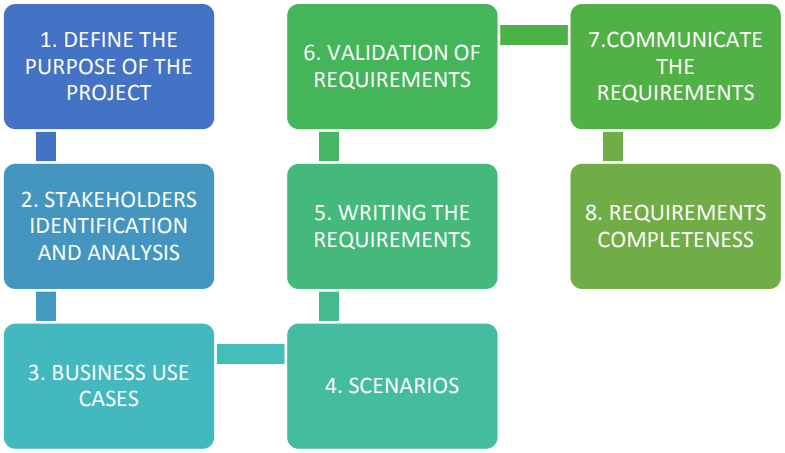

Figure 2.2.1: Depiction of the six steps in the classic Waterfall Methodology.

Analysis

The Volere requirements techniques were developed to answer the need for a common language for discovering requirements and connecting them to solutions. The language needs to be understandable by business people, customers, business analysts, engineers, designers, suppliers, testers or anyone else whose input is needed. All of these people have different skills and, not surprisingly, different views of what is important. A language intended for all of these people must recognise the differences in peoples' viewpoints and yet have a consistent way of communicating and tracing the relevant knowledge. This realisation that requirements is a socio-technical discipline has a strong influence on the development of the techniques.

The consequent application of the VOLERE methodology is not only useful in the initial phases of the project but it is also helpful in specifying a reference point for the later stages. During the implementation and management, it can be used to track and evaluate the progress of the individual work packages and the overall project. Besides being efficient and easy to use, the VOLERE methodology provides a mechanism for all partners to specify requirements, needs, use cases and scenarios in a standard format. Thereby, specifying additional context of an element such as the rationale and the acceptance criteria helps to build a common understanding of the overall system. Furthermore, defining priorities helps to clarify the focus of the project.

2.1.3 Agent-oriented Methodologies

General AOSE (Agent-Oriented Software Engineering) methodologies

2.1.3.1 GAIA

Gaia is a Methodology [ZJW03] specifically tailored to the analysis and design of agent-based systems. Before giving an overview of Gaia Methodology, it is worth commenting on the characteristics of domains for which Gaia is appropriate. Gaia is appropriate for the development of systems with the following main characteristics:

- Agents are coarse- grained computational systems, each making use of significant computational resources (think of each agent as having the resources of a Unix process).
- It is assumed that the goal is to obtain a system that maximises some global quality measure, but which may be sub-optimal from the point of view of the system components. Gaia is not intended for systems that admit the possibility of true conflict.
- Agents are heterogeneous, in that different agents may be implemented using different programming languages, architectures, and techniques. Gaia makes no assumptions about the delivery platform;
- The organisation structure of the system is static, in that inter-agent relationships do not change at run-time.
- The abilities of agents and the services they provide are static, in that they do not change at runtime.
- The overall system contains a comparatively small number of different agent types (less than100).

Gaia is intended to allow an analyst to go systematically from a statement of requirements to a design that is sufficiently detailed that it can be implemented directly. It is worth pointing out that Gaia authors view the requirements capture phase as being independent of the paradigm used for analysis and design. For this reason Gaia does not deal with the requirements capture phase but it considers the requirements statement as an input for the methodology. Analysis and design can be thought of as a process of developing increasingly detailed models of the system to be constructed moving from abstract to increasingly concrete concepts. Abstract entities are those used during analysis to conceptualise the system, but which do not necessarily have any direct realisation within the system. Concrete entities, in contrast, are used within the design process, and will typically have direct counterparts in the run-time system.

2.1.3.2 TROPOS

Tropos [GMP02] is an agent-oriented software development methodology created by a group of authors from various universities in Canada and Italy. One of the significant differences between Tropos and the other methodologies is its strong focus on early requirements analysis where the domain stake-holders and their intentions are identified and analysed. This analysis process allows the reason for developing the software to be captured. The software development process of Tropos consists of five phases: Early Requirements, Late Requirements, Architectural Design, Detailed Design and Implementation.

2.1.3.3 MaSe

The goal of Multiagent Systems Engineering (MaSE) [DWS01] is to provide a complete-lifecycle methodology to assist system developers to design and develop a multi-agent system. It fully describes the process that guides a system developer from an initial system specification to system implementation. This process consists of seven steps, divided into two phases: Analisys (Capturing Goals, Applying Use Cases, Refining Roles) and Design (Creating Agent Classes, Constructing Conversations, Assembling Agent Classes, System Design). MaSE has extensive tool support in the

form of agent Tool. Its latest version implements all seven steps of MaSE. It also provides automated support for transforming analysis models into design constructs.

2.1.3.4 Prometheus

The Prometheus [PW02] methodology is a detailed AOSE methodology that is aimed at non-experts. It has been successfully taught to and used by undergraduate students. Prometheus consists of three phases: System specification (determining the system's environment, and determining the goals and functionality of the system.), Architectural design (defining agent types, designing the overall system structure, and defining the interactions between agents) and Detailed design (defining capabilities, internal events, plans and detailed data structure for each agent type identified in the previous phase).

Prometheus is supported by two tools:

(a) The JACK Developmen tEnvironment (JDE), developed by Agent Oriented Software (www.agent-software.com) includes a design tool that allows overview diagrams to be drawn.

(b) The Prometheus Design Tool (PDT) provides forms to enter design entities. It performs cross checking to help ensure consistency and generates a design document along with overview diagrams.

2.1.3.5 MESSAGE

The MESSAGE methodology [GGM05] covers MAS analysis and design and is intended for use in mainstream software engineering departments. MESSAGE integrates into a coherent AOSE methodology basic agent-related concepts (such as organisation, role, goal, interaction, and task) that have so far been studied largely in isolation. The MESSAGE notation extends the UML with agent knowledge-level concepts and diagrams with notations for viewing them. The proposed diagrams extend UML class and activity diagrams. The MESSAGE analysis and design process is based on the Rational Unified Process (RUP) [A21]. The methodology distinguishes high-level from detailed design. An organisation-driven approach is presented for detailed design, where the global architecture of the MAS is derived from the structure and behaviour of the organisations that interact.

2.1.3.6 A comparison

General AOSE methodologies deal with providing engineering support to systems modelled as multi-agent systems. Thus, they could be used for general software development support (from analysis to implementation) for implementing or re-engineering software systems. Nevertheless, some method they provide could be reused, more in general, to support integration of systems. For instance, TROPOS proposes a goal-oriented analysis that could be reused to elicit integration requirements among different components/part of systems/systems. In particular, we reused TROPOS to identify/refine integration goals among IoT platforms.

Simulation-based Agent-oriented methodologies

The validation phase of software systems can be founded on several techniques such as formal methods, testing and simulation. In particular, testing requires the real deployment of the software system under-development whereas formal methods and simulation don’t require the real deployment. Such considerations can drive the choice of the adopted validation technique especially if the target execution environment is distributed as it occurs in the case of agent-based systems executing in Internet-like environments. In fact, agent-based systems are typically constituted by a huge number of agents executing in a large scale system so testing could be a very inefficient validation technique; conversely, simulation and formal methods validation techniques can be effectively used to validate functional and not functional requirements of agent-oriented designs before their implementations and deployments. Although formal methods are recognized as suitable validation techniques of agent-based systems, simulation-based ones have recently emerged.

In fact, to date a few multi agent-based systems development processes have been proposed in the literature that incorporate simulation to support the agent-based system development lifecycle with the main focus on the validation and performance evaluation of the designed solution. In the following, we briefly describe some interesting approaches for the development of agent-based systems that explicitly incorporate simulation such as Electronic Institutions [S04], DynDEVS/James [RU04], CaseLP [MMZ99], GAIA/MASSIMO [FGR05], TuCSon/pi [GVO05], Joint Measure [SBZH01], and Ingenias/RePast [PS06].

2.1.3.6 Electronic Institutions

In [S04] an integrated development environment for the engineering of multi-agent systems (MASs) as Electronic Institutions (EI) is presented. EIs provide a computational analogue of human organizations in which intelligent agents playing different organizational roles and interact to accomplish individual and organizational goals; authors define an EI as a performative structure of multi agent protocols along with a collection of normative rules that can be triggered off by agents' actions performed through speech acts.

The development environment, aimed at facilitating the iterated and progressive refinement of the development cycle of Eis, is composed of a set of tools supporting the design, validation through simulation, development, deployment and the execution of Eis. In particular, the simulation tool SIMDEI, allows for the animation and analysis of the rules and protocols specification in an EI. Moreover, SIMDEI supports simulations of Eis with varying populations of agents to conduct what-if analysis. The institution designer is in charge of analyzing the results of the simulations and returning to the design stage if they differ from the expected ones.

2.1.3.7 DynDEVS

In [RU04] a modeling and simulation framework (DynDEVS) for supporting the development process of MAS from specification to implementation is proposed. Authors advocate the use of controlled experimentation in order to allow for the incremental refinement of agents while providing rigorous observation facilities. The exploited framework is JAMES (Java Based Agent Modeling Environment for DEVS-based Simulation) which is aimed at an agent-oriented model design and execution supporting a modular and flexible construction of experimental frames for MASs [US98]. In particular, JAMES aims at exploring the integration of the agents paradigm within a general modeling and simulation formalism for discrete-event systems, DEVS (Discrete Event Systems Specification). Such formalism lends itself to an object-oriented model design and execution, facilitating the construction of experimental frames.

2.1.3.8 CaseLP

In [MMZ99] a logic based prototyping environment for multi-agent systems, CaseLP (Complex Application Specification Environment Based on Logic Programming) is presented. Authors propose an architectural description language which can be adopted to describe the prototype at the system specification level, in terms of agent classes, instances, their provided and requested services and communication links. Moreover, at the agent specification level, authors propose a rule-based not executable language which can easily define the behavior of reactive and proactive agents; with respect to the implementation of prototype, authors propose a platform-independent prolog-based language in which new primitives have been defined such as communication capabilities and safe state updates.

It is worth pointing out that from a simulation point of view, CaseLP is a time-driven centralized simulator with a global time known from all the agents in the system. Moreover, CaseLP integrates simulation tools for visualizing the prototype execution and for collecting the related statistics; more in detail, the CaseLP visualizer tool provides documentation about events that happen at the agent level during the MAS execution. Developers according to their needs can instrument the code of some agents after it has been loaded by adding probes to the code of agents. In this way, events

related to state changes and /or exchanged messages can be recorded and collected for on-line and/or off-line visualization.

2.1.3.9 TuCSoN/ π-calculus

In [GVO05] authors promote the use of formal tools, such as Stochastic π-Calculus process algebra, for simulating the dynamics of self-organizing multi agent systems through higher-level models defined at the early stages of design. In particular, authors assert that this approach appears to be almost unavoidable in order to foster evolving ideas and design choices, and to effectively tune parameters of the final system. Moreover, authors promote the use of SpiM (Stochastic PI-calculus Machine) which can be effectively used to simulate the Stochastic π-Calculus specifications and to track the dynamics of global system properties in stochastic simulations, validating design directions, inspiring new solutions, and determining suitable system parameters.

2.1.3.10 Joint Measure

In [SBZH01] a layered architectural framework to support agent-based system development in a collaborative, multidisciplinary engineering setting is proposed. In particular, such framework supports incremental specification, design, implementation, and simulation of agent-based systems. As authors distinguish between performing agents (executing in real-world settings) and simulated agents (simulated in a virtual environment), the proposed framework is intended to form the basis for environments that support development of agents, in both performance and simulation modes, as well as in hybrid combination (both performing and simulated agents interacting at the same time). Authors assert that for complex systems (e.g., distributed agent-oriented systems) adopting a single architectural framework, both for simulation and system development, can offer key advantages such as enable designers to study competing/alternative designs by employing a mixture of “simulated” and “performing” agents and, therefore, support migration from simulation design to operational design. The simulation is enabled by Joint MEASURE (Mission Effectiveness Analysis Simulator for Utility, Research and Evaluation) which is built upon DEVS/HLA, a generic HLA-compliant distributed simulation environment. It is worth noting that although Joint MEASURE affords a baseline to consider the requirements for agent development simulation environments, it is not intended to focus on agents per se.

2.1.3.11 INGENIAS/RePast

In [22] authors propose an agent-oriented methodology aimed to support modelling and simulation of social systems based on MASs. In particular, with respect to the modelling, they propose the use of an agent-oriented modelling language to specify MAS models representing complex social systems. Whereas, with respect to the simulation, authors propose the use of simulation toolkits (RePast, MASON, etc) that execute code obtained through MAS models transformation according with Model Driven Engineering (MDE) practices. This methodology is supported by a set of tools belonging to the IDK (INGENIAS Development Kit), which facilitates the edition of models and the definition of transformations for automatic code generation. Currently, the defined software component of IDK aimed at automatic code generation implements the mapping from INGENIAS models to RePast simulation toolkit.

2.1.3.11 GAIA/MASSIMO

In [FGR05] an integrated approach for the development and validation through simulation of MASs is proposed. Such approach centers on the instantiation of a software development process which specifically includes a simulation phase that makes it possible the validation of a MAS before its actual deployment and execution. In particular, the requirements capture is supported by a goal-oriented approach based on TROPOS methodology, the analysis and design phases are supported by the Gaia methodology, the detailed design phase is supported by the Agent-UML and the Distilled State-Charts formalisms, the implementation phase is supported by the MAO Framework, the simulation phase is enabled by a Java-based event-driven simulation framework, and the

deployment phase is supported by the MAAF framework which allows for the adaptation of the MAO Framework to programming abstraction provided by specific Java-based agent platforms.

2.1.3.12 ELDAMeth

ELDAMeth [FR12] is a methodology specifically designed for the simulation-based prototyping of distributed agent systems (DAS). It is based on the ELDA agent model and related frameworks and tools [F08, F10], and on an iterative development process covering the modelling, coding and simulation phases of DAS. ELDAMeth can be used both stand-alone and in conjunction/integration with other agent-oriented methodologies which fully support the analysis and (high-level) design phases. In particular, the development process of ELDAMeth (see Figure 2.3.1) consists of the following three phases:

- The Modeling phase produces an ELDA-based MAS design object that is a specification of a MAS fully compliant with the ELDA MAS meta-model (MMM). This design object can be produced either by (i) the ELDA-based modeling which uses the ELDA MMM and the ELDATool [A17], a CASE tool supporting visual modelling and coding of ELDA-based MAS, or by (ii) translation and refinement of design objects produced by other agent-oriented methodologies such as PASSI [C05], GAIA [WJK00], MCP [F09], etc. In particular, while the translation process centers on (semi) automatic model transformations based on the MMM of the employed methodology and the ELDA MMM, the refinement process is usually carried out manually by the ELDA-based Modeler by using the ELDATool.
- The Coding phase produces an ELDA-based MAS code object which is a translation of the ELDA-based MAS design object carried out manually or automatically (by means of the ELDATool) according to the ELDAFramework, which is a set of Java classes formalizing all the modelling abstractions of the ELDA MMM.
- The Simulation phase produces the Simulation Results in terms of MAS execution traces and calculation of the defined performance indices that must be carefully evaluated with respect to the functional and non-functional requirements. Such evaluation can lead to a further iteration step which starts from a new (re)modelling activity. In particular, the Simulation Results come from the execution of the ELDA-based MAS simulation object carried out through the ELDASim engine (ELDASim is a Java-based event-driven simulation framework for ELDA agents). The ELDA-based MAS simulation object is obtained by synthesizing the ELDA-based MAS code object with the simulation parameters and performance indices, defined on the basis of the requirements, by means of the ELDASim framework.

A comparison

Although all the overviewed methodologies offer different approaches to the modelling and simulation of MAS, they are centered on the use of simulation to validate and evaluate the design of the MAS under-development. Actually, few of them represent a full-fledged methodology (i.e. covering all the MAS development lifecycle) for the simulation-driven development of general-purpose MAS. Moreover, only INGENIAS and EI offer visual modelling tools for supporting the development process. In Table 2.3.1, the methodologies are compared with respect to the key features for the provision of an effective development of distributed agent systems: (i) Agent model (Mobility, Light-weight Reactive/Proactive agent behavior, Multi-coordination); (ii) Methodology (Dynamic validation methods before implementation, Integrability with other methodologies); (iii) CASE Tool. In particular, the ELDAMeth methodology supports all the mentioned features.

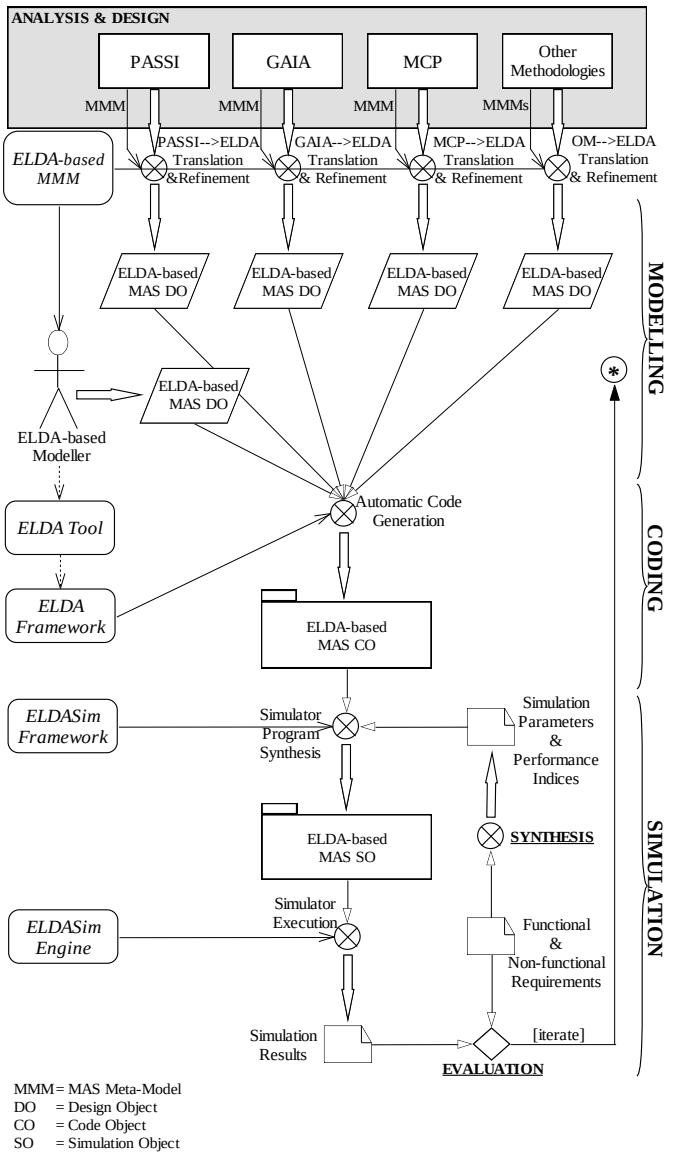

Figure 2.3.1: Iterative process for prototyping ELDA-based MASs.

Analysis

Agent-oriented methodologies are suitable for the development of distributed applications and systems in terms of multi-agent systems. They can be categorized in basic (Gaia, Message, Tropos, Mase, Prometheus) and simulation-based (TucSon/Pi, EI, DynDEVS, Gaia/Massimo, JM-DEVS/HLA, Ingenias, CaseLP, ELDAMeth). However, they do not aim at supporting (hw/sw) distributed systems integration, thus they cannot directly support the definition of INTER-METH. Nevertheless, some of the techniques proposed by such methodologies could be reused as basic techniques for defining INTER-METH:

- Goal-oriented analysis (from Tropos) to analyse integration goals;
- Agent-oriented domain conceptualization (from Gaia and ELDAMeth), to formalize integration requirements in the form of a high-level agent system design;
- Simulation-based validation (from simulation-based methodologies) to validate integration (i.e. the high-level agent system design) before its implementation.

Table 2.3.1: Comparison among simulation-based methodologies for MAS development.

	Agent Model			Methodology		CASE Tool
	Multi-Coord.	Light-Weight R/P Behavior	Mobility	**Dynamic validation through simulation**	Integrability	
TuCSon/Pi				X		
EI				X		X
DynDEVS		X	X	X		**partial**
GAIA/Massimo		X	X	X	X	
JM-DEVS/HLA				X		
Ingenias/Repast				X		X
CaseLP		**partial**		X		X
ELDAMeth	X	X	X	X	X	X

2.1.4 IoT Methodologies

In the following, we analyse currently available IoT methodologies. We should state that they are still at an early stage of development with respect to methodologies presented in the previous section, some of which are indeed well developed and diffused.

2.1.4.1 Requirements for IoT Systems Development

IoT systems are composed of many distributed and interacting components that are usually heterogeneous in terms of hardware devices, communication protocols, software interfaces, produced data, and semantics. To effectively support their development, general and specific requirements need to be defined [I1]. While the general requirements allow effective and flexible middleware for facilitating IoT system programming, the specific requirements are purposely defined for a target IoT system by considering its specific application domain. In the following, we focus on the former that are common to all IoT systems. In particular, we group such requirements in two categories: *System-level* (Table 2.4.1), which includes requirements related to the whole distributed system and its development, and *Things-level* (Table 2.4.2), which encompasses requirements particularly referring to the “things”, such as RFID, sensors, actuators, smart objects, mobile devices, machines, and robots, in an IoT system.

2.1.4.2 IoT Engineering Methodologies

Despite a variety of research efforts that tackle different specific issues within an IoT systems development process, a full-fledged IoT methodology is missing. There are lots of studies that, instead of providing a proper methodology, collect domain specific best practices, guidelines, checklists and templates. For example, Slama et al. [SPMB] and Collins [FR12] build up a repository of technology-dependent solutions coming from the experience in the industrial/business world and specifically directed to the IoT makers and enterprises. In fact, they propose reference architectures and guidelines to make specific purpose devices interoperable (TLR1) through abstraction data models (SLR5) and high-level software interfaces (SLR3). Differently, some researchers present general-purpose approaches. IoT-A [B13] is a systematic collection of architectures, reference models, common definitions and guidelines that can be used to derive a concrete IoT architecture.

Table 2.4.1: System-Level requirements (SLR)

Requirement	Description
SLR_1: Hardware Devices (Virtualization)	Devices are usually heterogeneous; in order to facilitate their use, abstractions are needed to *virtualize* and let them be used, as they are homogeneous by following a kind of a "plug&play" paradigm [V13].
SLR_2: Communications (Abstractions)	Software components and devices need to communicate with each other. Communication abstractions are needed to make them interact and cooperate, independently from the available low-level network protocols [M12].
SLR_3: Software Interfaces	As software interfaces are usually heterogeneous, they need to be generic and standardized through higher level mechanisms such that their use is straightforward. Thus, software components based on such high-level interfaces can be seamlessly accessed [BS11].
SLR_4: Physicality (Self and Context Awareness)	Hardware and software components in IoT systems and entire IoT systems themselves are intrinsically situated. This implies that they have static or dynamic locations and refer to one or multiple contexts during their lifecycle. Abstractions are therefore needed to capture the concepts of location and context, as they are useful in the design and implementation of IoT systems [P14].
SLR_5: Data (Abstraction)	Different hardware and software components, e.g. sensors, machines, smart objects, and mobile apps, usually produce data according to different modalities, formats and types. Thus, abstractions are needed to formalize data streams generated by such components. Continuous data streams, discrete data and sporadic events should be defined under a common framework. Moreover, the representation of data types needs to be standardized as it would allow interoperability in data exchange among heterogeneous components [K08].
SLR_6: Development Process (Methodology)	To analyze, design and implement IoT systems, suitable software engineering methods and tools need to be defined. They should be able to effectively model IoT systems by using high-level modeling abstractions and fully support their design, implementation, deployment and management [Z16].

Table 2.4.2: Things-Level requirements (TLR)

Requirement	Description
TLR_1: Heterogeneity and Interoperability	Applications that use "things" should be programmed independently from vendors-specific "things". For instance, if an application is based on a "smart chair", it should be able to use smart chairs built by different vendors. Moreover, applications should be able to exploit "things" to be built in the future. This implies to adopt a standardized approach or, if not applicable (standardization is a very long process), to exploit software layering-based dynamic adaptation techniques between application and the "things" levels [AIM10].
TLR_2: Augmentation Variation	"Things" usually provide a set of services that can vary in quantity and types both among different "things" and among similar "things". In particular, different "things" can provide same services whereas two similar "things" can provide different services. Thus, "things" cannot be crisply classified only by their type and may expose non-standard interfaces. Augmentation variation of "things" is an important requirement as it defines how "things" can modify their augmentation by providing diversified services that can change during their lifecycle. This implies to design not only methods to dynamically add/modify/remove "things" services but also how they are actually furnished [K10].
TLR_3: Decentralized Management	An effective management of "things" is crucial in IoT applications where tons of distributed "things" could potentially interact with each other and/or be used to fulfill a final goal. Applications and "things" should be therefore able to dynamically adapt as "things" could continuously change for different purposes (augmentation variation, mobility, failures, etc.). Thus, the matching among "things" services and application requirements should be often done at run-time. Discovery services are therefore strategic in such a dynamic context to find and retrieve "things" according to their static and dynamic properties [M12].
TLR_4: Dynamic Evolution	Applications and "things" should be simply and rapidly prototyped and upgraded through proper programming abstractions. The evolution can be driven by programming, learning, or both. In particular, evolution by learning is usually based on smart self-evolving components (application-level components and smart "things") able to self-drive their evolution on the basis of some learning models [K10].

By means of different views, perspectives and metamodels, IoT-A aims to offer a unified approach to the development of IoT systems, in order to promote cross-domain interaction (SLR2), to support interoperability (TLR1) and to reduce fragmentation within an IoT context. Most of the indications provided by IoT-A have inspired the AIOTI (Alliance for the Internet of Things) [AIOTI] reference models, specifically the domain model. The latter describes IoT entities and their relationships, by eliciting all the TLRs in the SO analysis phase. Zambonelli [Z16] proposed a software engineering methodology centered on the main general-purpose concepts related to the analysis, design and implementation phases of IoT systems and applications. Such concepts are used to identify the key software engineering abstractions (SLR1, and SLR3-SLR5) as well as a set of guidelines and activities that may drive the IoT systems development. The envisioned methodology, however, lacks the definition of models and tools to represent different conceptual and software artifacts. In brief, the existing methodologies neither completely support the TLRs and SLRs, nor cover the entire development process. In [FGRS15], authors propose a metamodel-based engineering approach for the systematic development of SOs, from analysis to implementation. In particular, (i) to support the analysis phase and therefore to model the relevant SO aspects, the Smart Object High-Level Metamodel has been introduced; (ii) to support the design phase and to model the functional components of the system, their relationships and interactions, the ELDA-based Smart Object Metamodel (where ELDA stands for Event-driven Lightweight Distilled statecharts-based Agent) and the ACOSO-based Smart Object Metamodel (where ACOSO stands for Agent-based Cooperating Smart Objects), respectively based on the ELDAMeth meta-model [F10] [FR12] and on the ACOSO framework [BPR01] [I19], have been exploited; (iii) to support the implementation phase, the ACOSO-based Agent Smart Object Metamodel has been specialized with respect to the JADE platform, resulting in the JACOSO Metamodel. Such methodology fulfills all identified requirements for IoT systems development (see Section 2.4.1).

Table 2.4.3: IoT Methodologies Comparison (Y=Yes, P=Partial, N=No)

IoT Methodologies [references]	Development phase (Analysis, Design, Implementation)			System-Level-Requirements						Things-Level-Requirements			
	A	D	I	SLR 1	SLR 2	SLR 3	SLR 4	SLR 5	SLR 6	TLR 1	TLR 2	TLR 3	TLR 4
[SPMB]	P	N	P	N	N	Y	N	Y	P	Y	Y	N	N
[CT]	P	N	P	N	N	Y	N	Y	P	Y	Y	N	N
[AIOTI]	Y	P	N	Y	Y	N	N	Y	P	Y	Y	N	Y
[B13]	Y	P	N	Y	Y	N	N	Y	P	Y	Y	N	Y
[Z16]	P	Y	N	Y	N	Y	Y	Y	P	Y	P	P	N
[FGRS15]	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Analysis

Although the overviewed methodologies have been specifically defined for developing IoT systems totally or partially fulfilling the reference requirements for IoT systems development, they have not been devised for IoT systems integration and interconnection. Thus, even though some of the ideas on which they are founded could be reused, such methodologies (as the reviewed agent-oriented methodologies) have another scope. INTER-Meth could borrow mainly:

- The meta-modeling approach (see ACOSO-Meth [FGRS15]) that is typical of the model-drive development (MDD) approach;
- The agent-oriented-like approach (see [Z16]) allowing to simplify the definition of integration requirements analysis;

- AIOTI [AIOTI] and IoT-A [I1] meta-models to have reference IoT architectures during the integration process, in order to align the meta-models of the IoT systems/platforms to be integrated/interconnected or made interoperable to the reference meta-model.

2.1.5 Agile Methodologies

Agile software development refers to a set of methods and practices based on principles expressed in Agile Manifesto[2]. Agile, meaning the ability to create and adapt to changes in the uncertain environment, was applied as a term characterizing the collection of methodologies in 2001, and since then it has been widely promoted (e.g. by Agile Alliance[3]) and applied also to other areas (e.g. marketing). The common features of agile methods include: close collaboration between the development team and business stakeholders, frequent delivery of business value, self-organizing cross-functional teams, quick response to changes and continuous development and integration (short feedback loop and adaptation cycle). Specifically, Agile is a time-focused, iterative philosophy that allows to build a product incrementally and deliver it by small increments. Agile projects can be adapted and modified at almost any step as a result of received feedback or changed market conditions.

The twelve principles behind Agile Manifesto are the following[4]:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity--the art of maximizing the amount of work not done--is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile development has been recognized as highly suited to certain types of environments, including small teams of experts working on innovative projects.

Lean[5] is a philosophy (mindset) that was adapted from manufacturing (originally developed by Toyota in the 1950's) to software development by Mary and Tom Poppendieck[6] in 2003. Agile and Lean philosophies have much in common – adaptive planning and team-orientation. Indeed, Agile has been influenced by Lean manufacturing from the beginning (Lean is considered a wider

[2] http://agilemanifesto.org/iso/en/manifesto.html
[3] https://www.agilealliance.org/
[4] http://agilemanifesto.org/iso/en/principles.html
[5] https://books.google.pl/books?id=hQk4S7asBi4C&pg=PA182&redir_esc=y&hl=en
[6] http://www.poppendieck.com/

approach). Lean aims at using fewer resources (e.g. time, money) and eliminating waste while maximizing customer value.

Lean software development is driven by seven principles:

1. Eliminate waste – everything that does not add value to the customer is treated as a waste
2. Amplify learning - continuous learning process based on iterations (writing code and testing)
3. Decide as late as possible – options-based approach enabling adaptation and correction of mistakes; decisions based on facts and not predictions
4. Deliver as fast as possible – short iterations ended with product increment delivery result in sooner feedback that can be considered for the next iteration
5. Empower the team – motivated team with influence on the project planning
6. Build integrity in – the components of the system work together well which is justified by automated testing
7. See the whole – the need to think about the system not only as a set of components, but as a product of their interactions

Even though practices for Lean software development differ from Agile software development practices, there are some similarities. Comparison of Agile and Lean is given here[7].

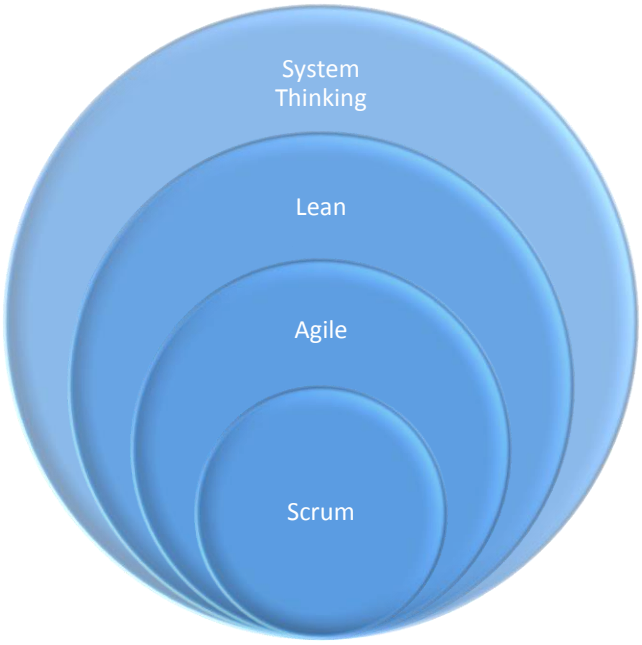

Figure 2.5.2: Relations between Agile, Lean and Scrum concepts[8]

Scrum

According to Jeff Sutherland Scrum is „a framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible

[7] https://realtimeboard.com/blog/choose-between-agile-lean-scrum-kanban/#.WA7WmiRXJ-B
[8] https://agileinsights.wordpress.com/tag/agile-development/

value"[9]. The guidelines of Scrum are summarized in the Scrum Guide[10], where roles of team members, events, artifacts and definition of done are described.

The three roles in the Scrum framework are:

1. Product Owner – represents stakeholders (acts on behalf of customers)
2. Scrum Master – ensures that Scrum framework is followed, removes impediments
3. Development Team – cross-functional and self-organizing team that is responsible for delivering potentially shippable increments

The Scrum workflow is based on sprints (iterations) that form a basic unit of development. They are usually timeboxed to two weeks. Sprint starts with a sprint planning meeting (SPM) during which a sprint backlog is created that identifies the work for the sprint, and make an estimated commitment for the sprint goal. Each sprint ends with a sprint review and sprint retrospective that aim at reviewing progress and identifying improvements for the next sprints. Scrum emphasizes that a shippable product increment (defined by sprint goals and definition of done) should be released at the end of each sprint. Each day at a specified time during a sprint, the team holds a daily scrum (short stand-up meeting) during which each member summarizes work done yesterday, plan for today and encountered problems.

Scrum artifacts include:

1. Product backlog - prioritized list of high-level requirements (user stories)
2. Sprint backlog - prioritized list of tasks to complete during the sprint (selected from the product backlog)
3. Product Increment - the sum of all product backlog tasks completed during a sprint, integrated with the work of all previous sprints

The extension to core artefacts allow visual tracking of progress – sprint burn-down chart, release burn-up chart. Scrum is promoted mostly by Scrum.org[11] and Scrum Alliance[12] organizations. Scrum can be scaled by implementing Scrum of Scrums (metaScrum)[13]. It allows to operate with a large group of people forming multiple teams working on the same product by allowing them to discuss progress on their interdependencies and focusing on how to coordinate delivering and integration.

Kanban

Kanban is a lean methodology that tracks 'to do – in progress – done' activities, however it limits them by the number of concurrent 'in progress' activities (the number is defined by the team manager and cannot be exceeded). Kanban uses visual work management (Kanban boards) to increase communication and collaboration. Team members pull work (tickets/cards) as they have capacity, by moving cards between swimlanes. Board design depends on the context and can vary significantly e.g. tickets can be grouped by different criteria, additional vertical and horizontal divisions may be introduced.

[9] http://scrumguides.org/scrum-guide.html
[10] http://scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-US.pdf#zoom=100
[11] https://www.scrum.org/
[12] https://www.scrumalliance.org/
[13] https://www.agilealliance.org/glossary/scrum-of-scrums/

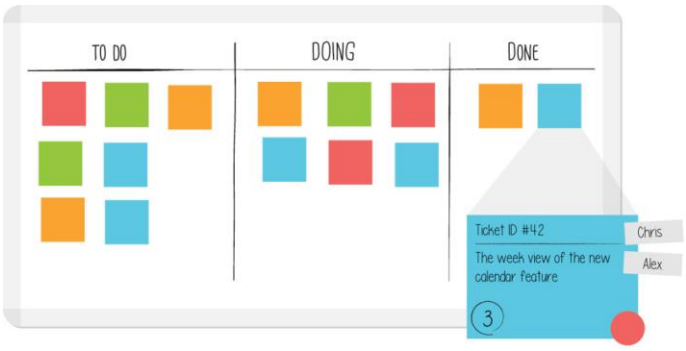

Figure 2.5.2: Kanban board. Source: https://leankit.com/learn/kanban/kanban-board/

Kanban methodology focuses on continuous learning and improvement by calculating metrics, analyzing (performing retrospectives) and optimizing the flow.

Scrum and Kanban share some characteristics as Agile and Lean methodologies, but there are also some significant differences[14]. They both deliver software early and often, utilize pull scheduling, limit work in progress and promote transparency. Kanban, as opposed to Scrum does not require specific roles, estimation and timeboxing. It works well with maintenance projects.

Analysis

Agile methods are designed for complex systems development with dynamic and non-deterministic characteristics, where accurate estimates and predictions are often hard to be made in early stages. See Table 2.5.1 for a comparison.

Table 2.5.1: Pros and Cons of Agile Methodologies

Pros	Cons
Adaptive rather than predictive. Changes are easier accepted and accommodated during the project.	May be problematic when there is a need to describe what will happen in the project in the late future. Planning is less concrete because it is based on timeboxed delivery.
Agile is very beneficial for projects where the end-goal is not clearly defined. The final product shape will come to light as project progresses and requirements evolve.	More emphasis put on working software than documentation. Team may feel that it is less important to focus on documentation. However, documentation can also be prepared and delivered within iterative cycles.
Fast and high-quality delivery. In each iteration an artefact is released that is tested and can be further integrated.	Team commitment and active involvement is required to succeed.
Continuous improvement and maximizing the delivered value by retrospective and re-planning done after each iteration.	Agile requires that clients participate proactively. In special cases they can be replaced by Product Owner.
Well identified pitfall that should be avoided in order to succeed.	Dedicated terminology that has to be used e.g. scrum, sprint, velocity, points.
Applicable to various projects from different fields (the basis are 12 Agile principles that need to be followed).	Small and maintenance projects are not ideal for Scrum methodology.
Scalable methodology – Scrum of Scrums.	
Many management tools that support working in Scrum or Kanban methodologies.	
Transparency in terms of requirements, tasks and costs communication.	
Strong team interaction.	

[14] https://www.crisp.se/file-uploads/futureofagile-slides-henrikkniberg.pdf

2.1.6 CMMI (Capability Maturity Model Integration)

CMMI (Capability Maturity Model Integration) [C1] is a framework of best practices for process improvement. The current version, CMMI-DEV, describes best practices in managing, measuring and monitoring software development processes. The CMMI model does not describe the processes themselves; it describes the characteristics of good processes, thus providing guidelines for companies developing or honing their own sets of processes.

CMMI is a methodology which aims to take projects or teams from level 1, "chaotic", to a higher level, ideally but not necessarily level 5, "optimizing".

It consists of various capabilities, each of which is assigned to a specific level. For example, CMMI level 2 requires the Project Planning capability. The levels are basically:

1. Chaotic, no real control.
2. Managed, processes at project level, mostly reactive.
3. Defined, processes at organization level, proactive.
4. Quantitative, processes measured and controlled.
5. Optimizing, feedback loops and continuously improving.

An organization cannot be certified in CMMI; instead, an organization is appraised. Depending on the type of appraisal, the organization can be awarded a maturity level rating (1-5) or a capability level achievement profile.

Many organizations find value in measuring their progress by conducting an appraisal. Appraisals are typically conducted for one or more of the following reasons:

1. To determine how well the organization's processes compare to CMMI best practices, and to identify areas where improvement can be made
2. To inform external customers and suppliers of how well the organization's processes compare to CMMI best practices
3. To meet the contractual requirements of one or more customers

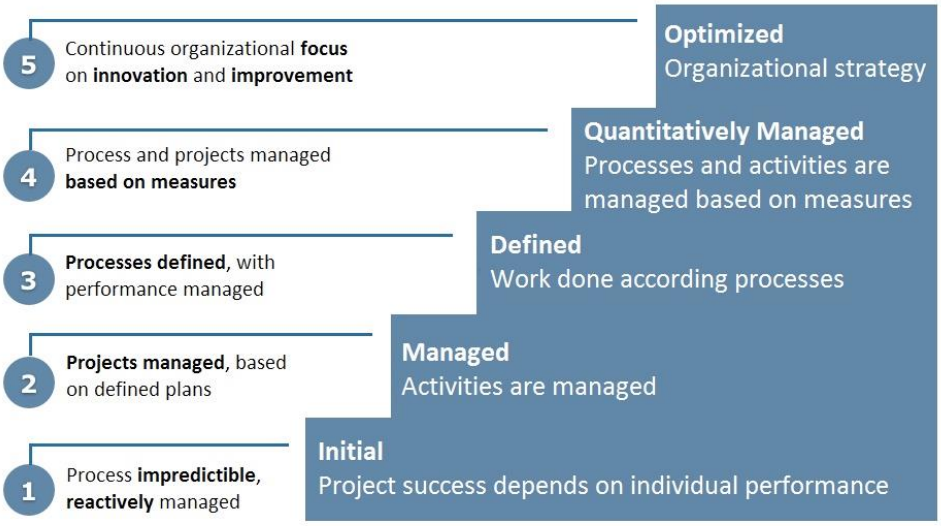

Figure 2.6.1: CMMI process

Appraisals of organizations using a CMMI model must conform to the requirements defined in the Appraisal Requirements for CMMI (ARC) document.

The Standard CMMI Appraisal Method for Process Improvement (SCAMPI) is an appraisal method that meets all of the ARC requirements. Results of a SCAMPI appraisal may be published (if the appraised organization approves) on the CMMI Web site of the Software Engineering Institute (SEI). CMMI is completely compatible with other methodologies, since it relates to process improvement, not determining specific tools to measure, manage or refine the project's development variables. Instead of specifying processes, CMMI is a model defining a collection of the activities to be expected of an organization as it sets out to improve its processes. Thus, CMMI must be implemented instead of applied.

Analysis

Once the INTER-METH process is defined, CMMI could be used to improve it, whereas SCAMPI could be used to appraise it according to the standard Appraisal Requirements for CMMI (ARC) procedure.

2.1.7 Overall Analysis

Methodologies: Summary

The first phase the SOTA analysis related to software engineering methodology for IoT identified the methodologies and types of methodologies reported in Table 2.7.1.

Table 2.7.1: Analyzed Methodologies and their main characteristics

Methodology	Interesting Characteristics
Waterfall	Well-defined and easily applicable process
Volere	Support for systematic elicitation of requirements based on use cases and scenarios
Agent-oriented Methodologies	- Conceptualization based on the Agent Paradigm - Easy analysis of requirements based on goals - Exploitation of simulation for system validation
IoT Methodologies	The use of reference metamodels such as AIOTI, IoT-A, ACOSO-Meth for IoT System Development
Agile/Scrum	Agile process useful for complex projects with dynamic requirements evolution
CMMI	Support for Process Improvement and associated appraisal method SCAMPI (Standard CMMI Appraisal Method for Process Improvement)

An analysis toward INTER-METH

Although the analyzed methodologies have been specifically defined for supporting the development of software systems, they have not been specifically devised for systems integration and/or interconnection.

Thus, even though some of the ideas on which they are founded could be reused, such methodologies (as the reviewed agent-oriented methodologies) have another scope.

However, INTER-METH could profit from some of their characteristics:

- *Waterfall* (Section 2.1.1): INTER-METH could be based on the abstract Waterfall process as proposed in the GA of INTER-IoT. However, the process should be made iterative and partly agile.
- *Volere* (Section 2.1.2): INTER-METH could profit from the Volere specification of requirements, needs, use cases and scenarios in a standard format.

- *Agent-oriented methodologies* (Section 2.1.3)*:* The agent-oriented-like approach can simplify integration requirements analysis (see TROPOS). The meta-modeling approach (see ACOSO-Meth) can support high-level and detailed-design of IoT systems integration. Moreover, agent-based simulation could be used to validate system integration.
- *IoT systems development methodology/approaches* (Section 2.1.4): AIOTI and IoT-A meta-models could be used as the reference IoT architectures during the integration process. Specifically, in order to align the meta-models of the IoT system/platform to be integrated/interconnected or made interoperable to the reference meta-model.
- *Scrum* (Section 2.1.5): It has in general pros and cons with respect to more static methodologies. The possible benefits towards INTER-METH definition should be identified.
- *CMMI* (Section 2.1.6): Once the INTER-METH process is defined, CMMI could be used to improve it, whereas SCAMPI could be used to appraise it according to the standard Appraisal Requirements for CMMI (ARC) procedure.

2.2 Integration Techniques and Methodologies for IoT Systems and Systems of Systems

2.2.1 IoT Systems Integration: Model-driven Interoperability

In the referred papers [GPS16][GP14], the main aim is to collect and analyse data from IoT world. This requirement brings a common problem, namely Interoperability. Things are highly heterogeneous networked devices employing different protocols (HTTP, MQTT, DDS) and different data formats (binary, XML, JSON). Such devices can then be composed with cloud-based infrastructure services or Cloud Platform services for further processing, analysis and storage. Interoperability is managed in an ad-hoc manner, but remains a significant burden on developers to understand and identify interoperability problems and then implement and test solutions accordingly.

In their research, they seek to reduce this burden using "*model-driven development*" tools and techniques. In particular, there are three key contributions:

- *Interoperability models are reusable*, visual software artefacts that model the behavior of services in a lightweight and technology independent manner; these models are a combination of architecture specification and behavior specification and are based upon Finite State Machines (FSM);
- A graphical development tool to allow the developer to create and edit interoperability models and to also execute tests to report interoperability issues;
- The Interoperability monitoring and testing framework captures systems events (REST operations, middleware messages, data transfers) and transforms them into a model specific format that can be used to evaluate and reason against required interoperability behavior.

Hence, such *model-driven development* approach allows the developer to create, use and re-use "models of interoperability" to reduce development complexity in line with the following requirements to ensure interoperability is correctly achieved.

To evaluate the tool, they utilize a case-study based approach:

- FIWARE provides a marketplace of independently developed Future Internet Services (approximately 30) that can be composed to build IoT and cloud applications; these are loosely coupled REST services without formal interface or behavioral specifications, and hence achieving interoperability remains a significant task for developers.

Different stakeholders are defined in the engineering methodology:

- **Interoperability testers** create new IoT applications and services to be composed with one another;
- **Application developers** model the interoperability requirements of service compositions. They create interoperability models to specify how IoT applications should behave when composed. These models are re-usable abstractions that can be edited, shared and composed;
- **Service or API developers** model the compliance requirements of their new service API;
- **Specification compliance testers** test compliance of their specification implementation.

With these models it is then possible to automate the interoperability testing processes and better support service composition. This approach explores models that focus solely on interoperability; the specification of the exchanges between IoT services with rules defining the required behavior for interoperability to be guaranteed. There are two types of model:

- The interoperability model used by application developers and interoperability testers;
- Specification models and compliance testers.

An interoperability model is specified as a finite state machine. A state represents a state of a distributed application waiting to observe an event. A transition represents a change in state based upon an observed event matching a set of rules regarding the required behavior. A compliance model is specified as a finite state machine using the same elements as the interoperability model and added two extra elements:

- Trigger state: this is an active state as opposed to an observing state;
- Trigger transition: this is a transition from one state of the distributed system caused by the sending of a new message.

Analysis

This research could provide some contribution to the INTER-IoT project regarding how an application interoperate with cloud and IoT services and then how easy it is to migrate a given application to a new service. In particular, it discusses about interoperability of IoT services/systems/(virtualized)devices and is fundamental in the CASE-driven integration methodology (INTER-METH) supporting the integration process of heterogeneous IoT platforms to obtain interoperability among them and allow implementation and deployment of IoT applications on top of them. Then an added value is the case study based on FIWARE, because the approach is based on the deployment of enablers and integration of services. It includes several components to integrate IoT and services. The middleware solutions allowing multiple technologies to be deployed and then supporting developers address further application and middleware interoperability problems. Specifically, the modelling of the interoperability requirements can be used by INTER-METH to define/formalize the integration requirements of IoT applications/systems/platforms. As INTER-METH is based on another method, this is indeed taken as future work for INTER-METH.

2.2.2 System of Systems (SoS) Integration

As in [KR13]: “SoS is a set of systems that are cooperating and interoperating while the different systems are simultaneously working as independent entities (and thus not only as parts of the integrated system).”

As in [MMW96]: SoS is an assemblage of components which individually may be regarded as systems, and which possesses five additional properties:

1. Operational Independence of the Components
2. Managerial Independence of the Components

3. Evolutionary Development
4. Emergent Behavior
5. Geographic Distribution

As in [SOS6]: “A set or arrangement of systems that results when independent and useful systems are integrated into a larger system that delivers unique capabilities”.

Unlike traditional systems where systems are created based on a specific user requirements, SoS is composed of multiple constituent systems at various stages within their lifecycles (e.g. emerging systems, systems in-development, legacy systems) to satisfy needed mission capabilities. Differences between SoS and traditional systems are summarized in Table 1 in [VK16].

Summary of [KR13]

Authors propose methods to improve the way in which an IT architect addresses the integration problem. In classic publications such as [C02] and [HW04], generic approaches to integration problem are presented (often making implicit assumptions). Here, authors focus on how to select the best integration approach in SoS context depending on the features of the environment and systems to be integrated. In case of SoS the integration can happen on different layers e.g. database, UI, business logic, for each some developments (e.g. SOA for services) are already available. The amount of available generic and layer specific design patterns makes it more difficult to choose the best solution.

In this paper, a technique is presented in which the *architect needs to perform initial analysis based on predefined categories*. The classification helps to understand and describe design constraints for SoS.

First of all, the form of integration to be achieved will influence the form of the design pattern chosen. Two main aspects of integration shall be considered: how *data* will be shared and how *control* will be managed (See Table 2.8.1).

Table 2.8.1: Data and Control choices in SOS (Based on Table 1 from [1])

Control / Data	**Shared**	**Isolated**
Hierarchical	Information system based on call-return paradigm with shared memory	Traditional information system based on call-return paradigm
None	Data-centric system e.g. warehouse	Agent-based system

Next, the categorization summarized in Table 2.8.2 is proposed.

Table 2.8.2: Summary of template for information pattern matrix (Based on Table 4 from [1])

	Attribute	***Possible Values***
SoS Context	SoS Scope	System of Systems, System
	Development Context	Greenfield, Brownfield, Closed Source
	Integration Purpose	One-Directional Information Exchange, Bi-Directional Information Exchange, Control, Negotiation
	LISI (Levels of Information Systems Interoperability)	Isolated, Connected, Functional, Domain, Enterprise
	PAID	Procedures, Applications, Infrastructure, Data
Technical Categorization	Integration Level	Information Exchange, Basic Behavior Interaction, Complex Behavior Interaction, User Interface Sharing
	Data Abstraction Level	Structural, Syntactic, Semantic

	Data Level Integration	File-Transfer, Message-Exchange, Streams, Common Data
	Interaction Style	Send, Call, Call-Return, Call/Call-Back, Time-Based, Multi-Call Protocols
	Quality Attributes of Integration	Reliability, Performance, Security, Availability, Interoperability, Scalability, Manageability, Consistency

Finally, a few design patterns from popular patterns catalogs are analyzed with respect to the proposed classification. They were assigned values for the attributes in the pattern integration matrix template. In this way, on one hand, the integration context can be described and on the other patterns can be analyzed, compared and chosen to match the context. From this approach emerges a process for creating SoS based on patterns as a central architectural concept. Authors suggest the following sequence of steps:

1. *"Choose/prioritize architecturally significant requirements, focusing on the driving quality attributes.*
2. *Catalog the technological, budget, schedule, and organizational constraints that will limit the set of possible architectural options.*
3. *Choose data and control strategies that reflect the requirements of the interoperating systems.*
4. *Based on the requirements, constraints, and data/control strategies, choose one or more patterns.*
5. *Tailor and combine any chosen patterns.*
6. *Finally, choose technologies to implement and realize these patterns. Enterprise frameworks, for example, will realize many of the patterns described in this report at relatively low-cost to the project."*

Summary of [DJ11]

This paper presents *time-sequence wave model* representing SoS systems engineering (SE) based on guideline by U.S. Department of Defence on SE of SoS [6]. Main characteristics of the model are:

- Multiple overlapping iterations of evolution
- Ongoing analysis
- Continuous input from external environment
- Architecture evolution
- Forward movement with feedback

Model elements are:

- Initiate SoS – fundamental information to start the SoS process,
- Conduct SoS analysis – analysis of the „as is" SoS by establishing SoS baseline and developing initial plans for SoS evolution,
- Develop and evolve SoS architecture – adjustment of the technical framework to address SoS evolution and identify risks,
- Plan SoS update – plan the next SoS update cycle,
- Implement SoS update – monitoring, implementation and testing,
- Continue SoS analysis – ongoing analysis as a basis for future SoS evolution.

Authors have also identified the artifacts that are associated with the model along with the information where they are created, updated and used in the wave model. This publication is complimentary to [6] where SoS core elements are defined and application of system engineering processes in the context of SoS and identified focus area are described.

Summary of [VK16]
Author of the paper introduces System of Systems Engineering and Integration (SSE&I) methodology that describes planning, analyzing and integrating of systems into SoS. The process is based on the *V-Model* that is adapted to represent SoS lifecycle. The following phases were identified:

1. SoS Architecture and Requirements Development – user needs are defined; includes capability collection/customer interface, capability assessment and analysis, SoS architecture development and analysis, SoS requirements and allocation;
2. Systems Design and Development – engineering activities performed within the constituent systems;
3. SoS Governance and Analysis – establishing the set of rules, policies, and decision-making criteria;
4. Mission Assurance; activities complementary to SoS Architecture and Requirements Development including SoS interoperability and certification, deployment, operations and maintenance; this phase assures constant collaboration and coordination of activities for the successful development and integration of the SoS.

Detailed descriptions of the phases are available in [VK16].

Analysis
While methods from [KR13] shall be considered to classify available design patterns that can be applied in SoS / INTER-IoT context, methods in [DJ11, SOS6, VK16] present approaches to SoS engineering and methodology that are based on the V-Model [SOS7] that is not adopted by INTER-METH.

Therefore, our analysis of INTER-IoT features has shown that it shares many characteristics with the concept of SoS. It enables interoperability between systems that operate independently, which results in the development of additional functions that do not reside in any of the integrated components. These characteristics challenge the application of traditional software engineering approaches, since it is more difficult to define boundaries and requirements, and to control the development environment.

2.2.3 Telecommunication Systems Integration

There are three main methods for telecommunication system integration: horizontal integration, vertical integration, and star integration. For the specific case of network and communication system integration, from the IoT environment perspective, it is desirable to achieve seamless integration and avoid the existence of silos. Therefore system integration should be done by means of the method of horizontal integration. To achieve a true Internet of Things we need to move away from such small-scale, vertical application silos, towards a horizontal infrastructure on which a variety of applications can run simultaneously. This is only possible if connecting a thing to the Internet of Things becomes as simple as plugging it in and switching it on. Such plug and play functionality requires an infrastructure that supports it, starting from the networking level.

Horizontal integration - in opposition to vertical integration, “layered”, horizontal integration provides “seamless” integration. Integration based on a horizontal infrastructure on which a variety of applications can run simultaneously.

In the following some well-known methods are listed and commented.

- *Enterprise Service Bus (ESB)* is an integration method in which a specialized subsystem is dedicated to communication between other subsystems. This allows cutting the number of connections (interfaces) to only one per subsystem which will connect directly to the ESB.

The ESB is capable of translating the interface into another interface. This allows cutting the costs of integration and provides extreme flexibility. With systems integrated using this method, it is possible to completely replace one subsystem with another subsystem which provides similar functionality but exports different interfaces, all this completely transparent for the rest of the subsystems. The only action required is to implement the new interface between the ESB and the new subsystem. The horizontal scheme can be misleading, however, if it is thought that the cost of intermediate data transformation or the cost of shifting responsibility over business logic can be avoided.

- By means of the creation of *Virtual Networks*, network and telecommunication systems can be seamless integrated in the overall IoT system, within the Virtual Network. The term "network virtualization" means the operation of multiple networks with different characteristics and functions over the same physical network. The new communication control technology has two key features: (1) The optimal allocation of virtual network capacity based on communication traffic, and (2) The optimization of virtual network routes. This technology will enhance the efficiency of network use in terms of the data communication speed and capacity required for each application, and enable telecommunication operators to provide high-speed communication to a larger number of users. The virtualization technology copes with network congestion caused by a heavy traffic load in a particular part of network by connecting automatically to other types of network, thereby increasing the potential for services to continue operating.
- The *M2M platform and gateways* are required to handle completely new telecommunication protocols, and connect networks reliably. Networks with different technologies require protocol translation. Also it should be considered data format translation. A common data format is an integration method to avoid every adapter having to convert data to/from every other applications' formats, Enterprise application integration (EAI) systems usually stipulate an application-independent (or common) data format. The EAI system usually provides a data transformation service as well to help convert between application-specific and common formats. This is done in two steps: the adapter converts information from the application's format to the bus's common format. Then, semantic transformations are applied on this (converting zip codes to city names, splitting/merging objects from one application into objects in the other applications, and so on).

Other methods for system integration (not devised as deemed not appropriate for IoT environments) are vertical integration and star integration.

2.2.4 Systems Integration Best Practices

Systems Integration (SI) [BP1] is a process of assembly of constituent parts of a system. It includes comprehensive system execution check and a full functional check-out. Together with Systems Test (verification that the system meets requirements and validation that the system performs in accordance with the expectations) and Systems Engineering it forms the Systems Engineering, Integration and Test (SEIT) process.

However, SEIT practices are in most cases highly inconsistent and error-prone, following the so-called Big Bang Integration model. In this model, all components of the system arrive for integration more-or-less simultaneously and tests are conducted either when integration is finished or on the day before delivery, whichever comes first. Thus, integration starts way too late, it is expected to occur instantaneously and its focus is on the components, rather than system capabilities.

Better alternative to the Big Bang Integration model is the *Continuous Integration* model, which establishes integration rhythm that is essentially independent of the development team. It utilizes a dedicated team of systems integration engineers who know the systems requirements. Together with design and test engineers they create a *System Architecture Skeleton (SAS)* very early in the process of the development and use it as the framework for the integration of subsystems and

components. SAS enables incremental subsystem integration (subsystems represent end-to-end function threads that satisfy a set of requirements for capabilities and performance), while it also provides an environment for component verification. With the help of SAS it is possible to demonstrate early, concrete milestones to the customer and it allows witnessed subsystem verification, before the whole system is finished.

The process of integration in the *Continuous Integration model* thus overlaps with development, while its success is based upon completion and verification of integrated subsystems end-to-end against system-level requirements. Internal and/or external automated test tools are created, which greatly reduce the emphasis on man-in-the-loop testing, while the whole process is augmented with requirements-driven testing, which includes stress testing and long mission threads testing.

In order to aid the process of integration, Continuous Integration model partitions specifications and procedures into three categories; *requirements, design and verification*. Requirements are divided into system requirements and derived requirements, design is divided into high-level and detailed design, while verification is divided into system requirements verification and component checkout. In the case of simple and low-risk items, these specifications and requirements are simple and coarse, while for complex and/or high-risk items, these are more detailed.

As is the case for development process, the Continuous Integration process also runs independently of component checkout and release for subsystem integration. *Software Version Description Document (VDD) accompanies each new release, as well as informal readiness integration review*. The focus is on successful integration and end-to-end verification against system-level requirements of each new release, to the extent that is possible within the SAS and currently integrated subsystems. Stress testing and long mission thread testing also commences in parallel to verification; the former includes processor and storage measurements in various system conditions, 'corner condition' testing, testing system operation beyond required system capacities and endurance /'woodpecker' testing, while the latter consists of testing in operationally relevant scenarios and user testing (in the form of focus groups) – in order to ensure suitability.

The system is ready, when all subsystems are integrated, when all subsystem threads are exercised successfully, and the system is robust and ready for formal requirements verification.

Analysis

In the context of INTER-IoT we can reuse the notion of continuous integration and testing, parallel to the development. Alongside with continuous integration we could also have continuous stress testing and verification, which the resulting system at that stage of development conforms to the requirements. *Continuous* integration process continues independently from component checkout and releases for subsystem integration.

We can introduce the notion of *SAS (System Architecture Skeleton)*, around which we can form our CASE tool for automated application of INTER-METH methodology.

Requirements traceability will be maintained between requirements, design and verification specifications at all levels of development and integration; through the *Informal Integration Readiness Review*.

2.2.5 IOT-A Methodology

A major benefit of the IoT-A ARM (Architecture Reference Model) [B13] is the capability of generating architectures for specific systems. This architecture generation is done by providing best practices and guidance for helping translating the ARM into concrete architectures. The benefit of such a generation scheme for IoT architectures is not only a certain degree of automatism of this process, and thus the saved R&D efforts, but also that the decisions made follow a clear, documented pattern.

Reference Model and Reference Architecture Definition

A reference model is, according to the OASIS [IOTA1] definition: “A reference model is an abstract framework for understanding significant relationships among the entities of some environment. It enables the development of specific reference or concrete architectures using consistent standards or specifications supporting that environment. A reference model consists of a minimal set of unifying concepts, axioms and relationships within a particular problem domain, and is independent of specific standards, technologies, implementations, or other concrete details. A reference model may be used as a basis for education and explaining standards to non-specialists”.

While an architectural reference model (ARM) is a term extensively used in IOT-A and defined in this document as an architectural pattern in [B07]: “Architectural reference model is a description of element and relation types together with a set of constraints on how they may be used.”

Finally, the definition of reference architecture used in this document is the following (also taken from [B07]): “A reference architecture is a reference model mapped onto software elements (that cooperatively implement the functionality defined in the reference model) and the data flows between them”.

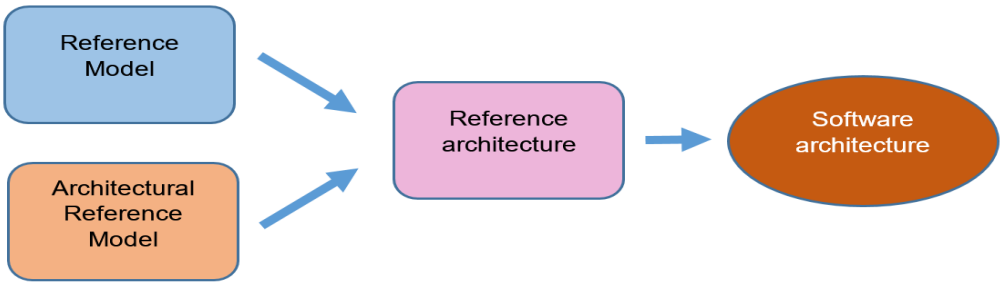

Figure 2.8.1: IoT-A: from Reference Models to Software Architecture.

Architecture Generation

The process described in [IOT3] is used to describe each of the following seven views:

- Physical Entity view
- Deployment view
- Operational view
- IoT Context view
- IoT Domain Model
- Functional view
- Information view

And the process to execute is:

1. Create Physical Entity view. It refers to the Physical Entity in the IoT Domain Model [IOT3]. A Physical Entity is “any physical object that is relevant from a user or application perspective”. **IoT-A does not provide a solution for this definition**. Activities related to this are the analysis of devices to be connected, the definition of the type of measurements and the physical communication links to use.

2. Create IoT Context view. The context view describes “the relationships, dependencies, and interactions between the system and its environment (the people, systems, and external entities with which it interacts)”. The concerns addressed by the context view are:
   - System scope and responsibilities
   - Identity of external entities and services and data used

- Nature and characteristics of external entities
- Identity and responsibilities of external interfaces
- Nature and characteristics of external interfaces
- Other external interdependencies
- Impact of the system on its environment
- Overall completeness, consistency and coherence

According to the IOT-A deliverables, the IoT Context View is composed of the context view and the IoT Domain Model. It provides a semantic and ontological overlay for the context view in that it provides guidance on which entities make up an IoT system and how they relate to each other. It also helps to identify system boundaries, which is one of the main questions to be addressed in the context view.

3. Requirements process: For IOT-A, the requirements can be categorized according to the action that the reference raised: 1) view requirements (i.e. requirements that directly inform one of the architectural views), 2) qualitative requirements and 3) design constraints. This process is supported by the definition of the so-called “Unified Requirements” template and follows extended methodologies such as ISO 9001. The requirements are named in that way because they follow a standard way of definition such as a template.
4. Derive other views: IOT-A suggest to derive from the previous activities, at least the following views: functional view, the information view, the operational view and the deployment view. This activity is contingent on the requirements process and is also guided by the Physical Entity View and the IoT Context View. For instance, the IoT Context View might indicate that, due to the different ownership of parts of the system, a communication firewall is needed (functional view). In another example, the Physical Entity View might indicate that, due to the fragility of the Physical Entity, all devices attached need to be installed all at once (deployment view).

Fig. 2.8.2 (extracted from [IOT3]) depicts the architecture generation process (and where the steps are described in the mentioned book).

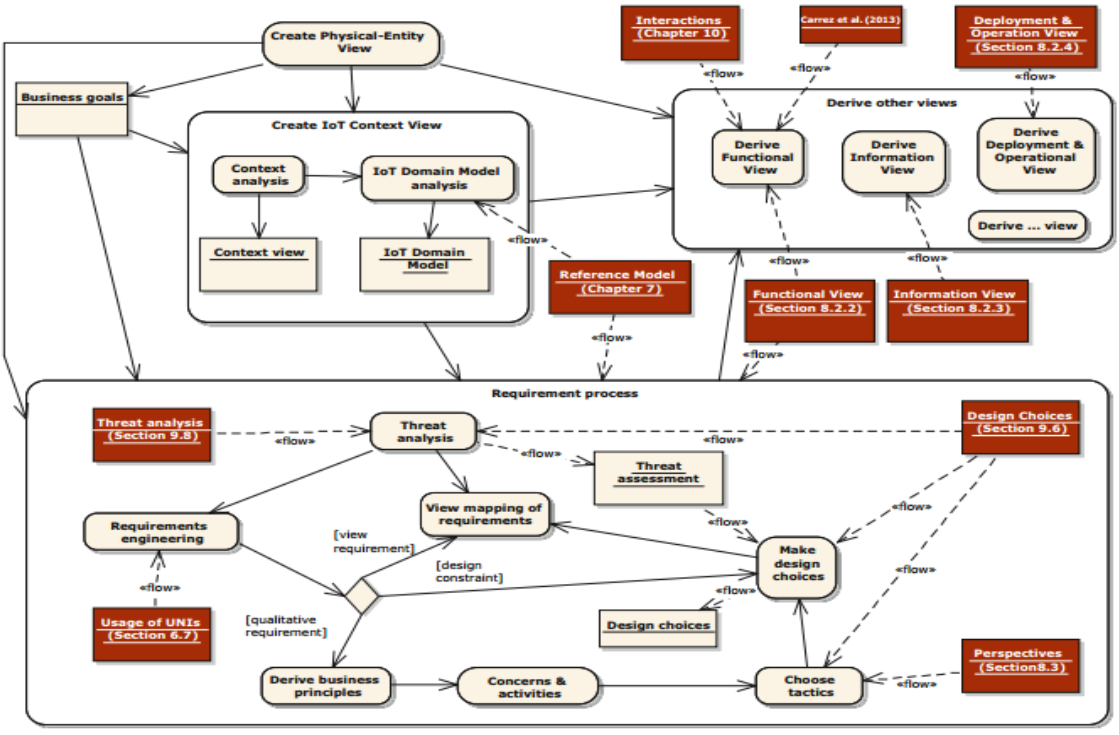

Figure 2.8.2: IoT-A architecture generation process.

Fig. 2.8.3 describes the requirements process.

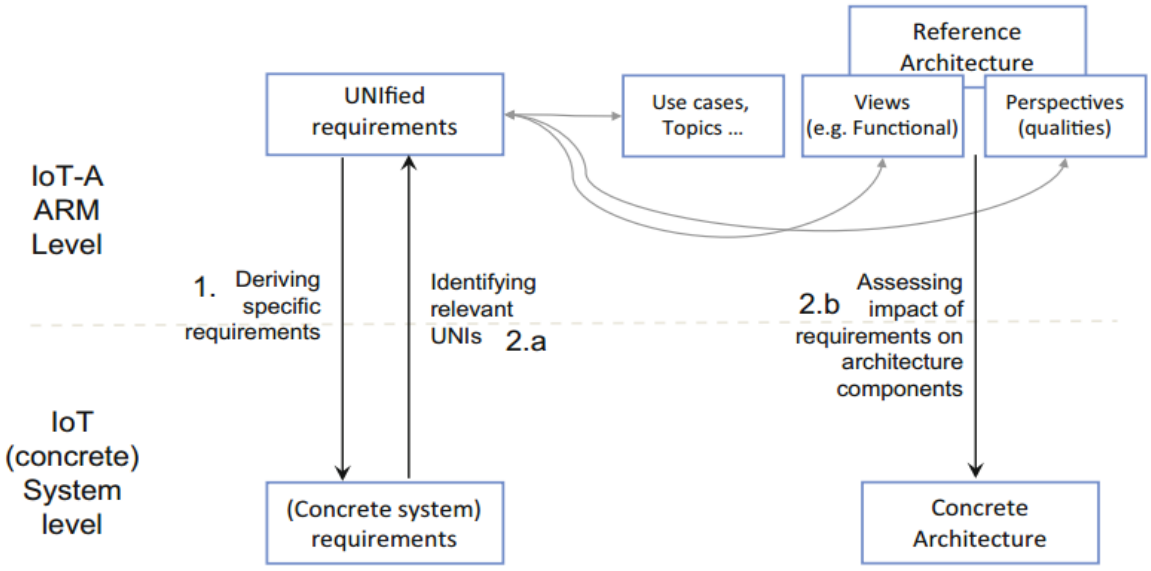

Figure 2.8.3: IoT-A requirements process.

In the workflow depicted in Fig. 2.8.4, the process of how to use the requirements in the different views construction is defined.

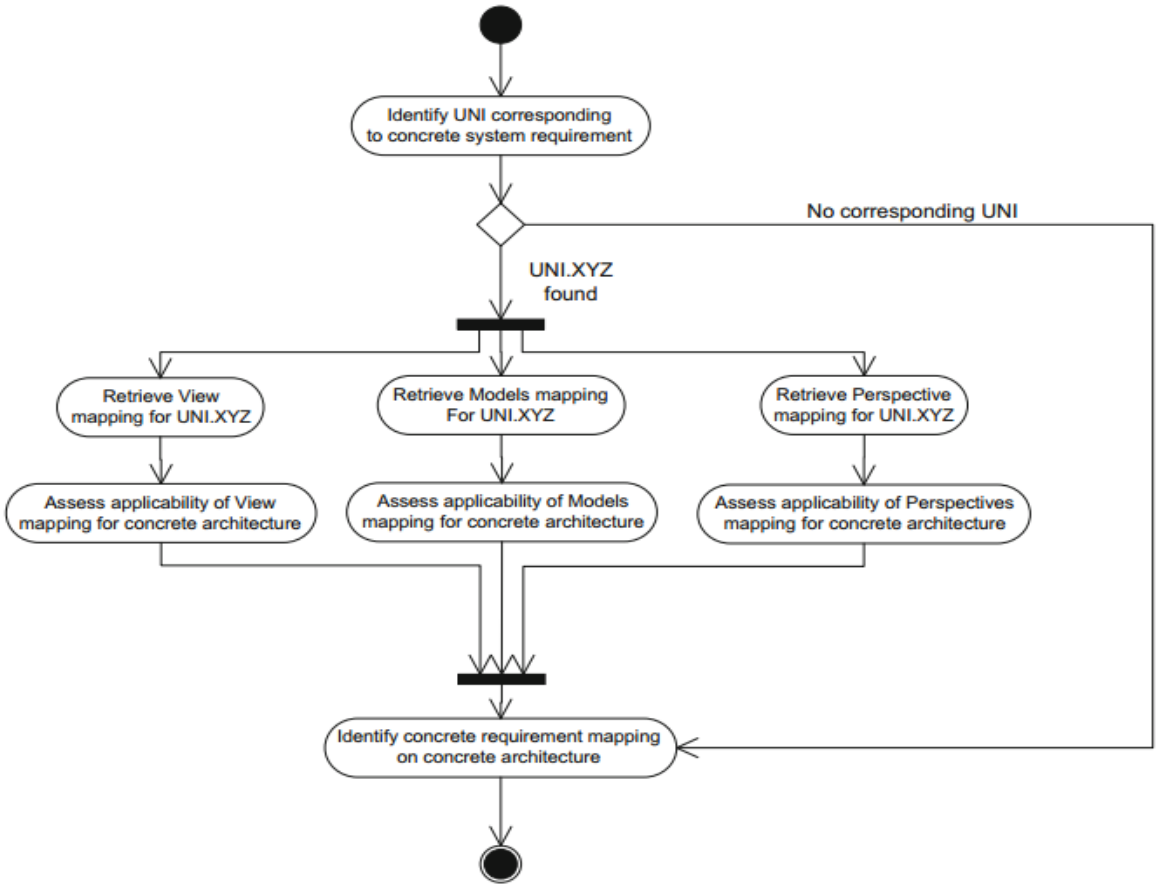

Figure 2.8.4: IoT-A workflow for requirements implementation.

Design choice and development methodology

The IOT-A methodology classifies the design choices into the following depending on the perspectives they address. A perspective is a collection of activities, tactics, and guidelines that are used to ensure that a system exhibits a particular set of related quality properties that require consideration across a number of the system's architectural views.

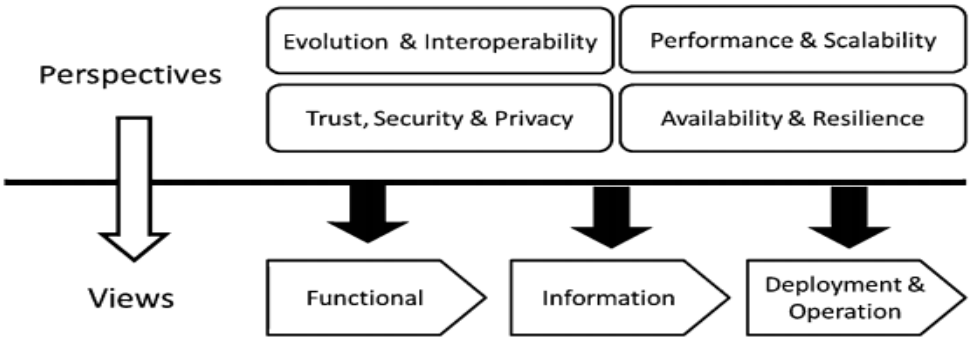

Figure 2.8.5: IoT-A Methodology correlating Perspectives and Views.

Figure 2.8.5 depicts the relation of the IOT-A defined perspective and the architectural views. These perspectives are:

- *Evolution and interoperability*. This perspective addresses the fact that requirements change and software evolves sometimes rapidly. The activities related to this perspective are:
    - Characterize the evolution needs
    - Assess the current ease of evolution
    - Consider the evolution trade-offs
    - Rework the architecture

  And the tactics related:
    - Create extensible interfaces, Apply design techniques that facilitate change
    - Apply metamodel-based architectural styles
    - Build variation points into the software, Use standard extension points
- *Performance and scalability*. The tactics related to this perspective are:
    - Replication
    - Prioritize Processing
    - Distribute Processing Over Time
    - Minimize Used of Shared Resources
    - Reuse Resources and Results
    - Scale Up or Scale Out
    - Degrade Gracefully
- *Trust, security and privacy*, with the following tactics associated:
    - Harden Root of Trust (trust)
    - Ensure High Quality of Data (trust)
    - Infrastructural Trust and Reputation Agents (trust)
    - Provide High System Integrity (trust)
    - Avoid Leap of Faith (trust)
    - Subject Authentication (security)
    - Use Access Policies (security)
    - Secure Communication Infrastructure (security)
    - Secure Peripheral Networks (security)
    - Pseudonymisation (privacy)

    - Avoid Transmitting Identifiers in Clear (privacy)
    - Minimize Unauthorized Access to Implicit Information (privacy)
    - Enable the User to Control the Privacy Settings (privacy)
    - Privacy-Aware Identification (privacy)
- *Availability and resilience*
    - Use High Availability Clustering
    - Load Balancing
    - Etc.

Analysis

IOT-A ARM and methodology can be used to support INTER-METH. Specifically, we re-used the functional architecture (Functional View) during the INTER-METH *Analysis Phase*.

2.2.6 Overall Analysis

Techniques and Methodologies: Summary

The second phase of the SOTA analysis, related to systems integration methodologies for IoT and SoS, identified the methodologies and techniques in Table 2.9.1.

Table 2.9.1: Analyzed Methodologies and their main characteristics

Methodology/Techniques	**Interesting Characteristics**
Model-driven Interoperability	- Interoperability models for IoT systems/services integration
System of Systems (SoS) Integration	- Methods for the classification of available design patterns for SoS engineering/integration - Approaches to SoS engineering and methodology that are based on the V-Model
Telecommunication Systems Integration	- Enterprise service bus - Systems Virtualization
Systems Integration Best Practices	- *Continuous* Systems Engineering, Integration and Test (SEIT) process - *System Architecture Skeleton (see also IOT-A Methodology)*
IOT-A Methodology	- Architecture Reference Model (ARM) - Unified Requirements - Architectures generation model

An analysis toward INTER-METH

The analyzed methodologies and techniques directly address the issue of systems integration by adopting different approaches. Some of them are purposely related to IoT systems integration but they do not provide any systematic methodology that, starting from two or more systems to integrate, provides a clean process (along which tools for each phase) to support the integration.

However, INTER-METH could profit from or incorporate some of their characteristics:

- IoT Systems Integration: Model-driven Interoperability: *Interoperability models.*
- System of Systems (SoS) Integration: *Approaches to SoS engineering and methodology that are based on the V-Model*.
- Telecommunication Systems Integration: *ESB and virtualization techniques*.
- Systems Integration Best Practices: *Continuous SEIT and System Architecture Skeleton.*
- IOT-A Methodology: *ARM, Unified requirements and architectures generation techniques*.

3 INTER-METH Functionalities, Requirements, Use Cases, and Scenarios

3.1 Functionalities

INTER-METH is an engineering methodology that aims at supporting the integration process of heterogeneous IoT platforms to (i) obtain interoperability among them and (ii) allow implementation and deployment of IoT applications on top of them. It is widely recognized that using an engineering methodology is fundamental in any engineering application domain (e.g. software engineering, co-design hardware/software, civil engineering, etc). Moreover, the manual and non-systematic application of complex techniques, methods and frameworks would very likely lead to an increase of the degree of errors during integration. Thus, INTER-METH has the objective to avoid such errors. Specifically, INTER-METH is based on a process that defines and offers the following phases (or functionalities of a process):

1. Analysis phase enables the definition of the IoT platform integration requirements (both functional and non-functional).
2. Design phase produces the design of the platform integration in terms of: a) design artifacts (e.g. diagrams) of layer interoperability infrastructures and related interfaces (see INTER-LAYER), and b) INTER-FW programming and management patterns, to fulfil the elicited requirements.
3. Implementation phase focuses on driving the implementation of the design work-product to obtain the fully-working (hardware and/or software implemented) system.
4. Deployment phase involves the support to the operating set-up and configuration of the integrated IoT platform.
5. Testing phase defines the performance evaluation tests to validate the integrated platform according to the functional and non-functional requirements.
6. Maintenance phase manages the upgrade and evolution of the integrated system.

Each phase produces work-products that are inputs for the successive phase(s). INTER-Meth will be supported by a CASE (Computer Aided Software Engineering) tool that will help automating each aforementioned phase of the integration process and will specifically provide the following functionalities: (i) integration guidelines management, (ii) graphical facilities, (iii) engineering patterns automation, and (iv) project data repositories.

3.2 Requirements

The INTER-METH-related requirements elicited in the INTER-IoT requirements analysis phases are reported in Table 2.2.1. Details of the requirements are given below (more information can be found in D2.3).

Table 2.2.1: Main INTER-METH Requirements

ID	Name	Type
74	Ontology support	Functional
162	Specification of unique attributability	Functional
161	Model-driven support	Functional

160	CASE-tool support	Functional
159	Development support for systematic IoT platforms integration/interconnection	Functional
120	Effectiveness and Optimization	Non Functional
119	Maintainability	Non Functional
118	Legal and licensing issues avoidability	Non Functional
117	Security and Trust Management	Functional
116	Privacy	Functional
115	Scalability Preserving	Non Functional
114	Extensibility and Customization	Non Functional
113	Error minimization	Non Functional
112	Compliance	Non Functional
111	Documentation	Non Functional
110	Usability	Non Functional
109	Stability	Non Functional
108	Open Source	Non Functional

74: Ontology support

Category: Functional; Type: Semantics; Priority: Must

Rationale: INTER-METH should provide support for the management of the ontologies of heterogeneous IoT platforms.

Description: INTER-METH provides mechanisms to support through design semantic interoperability between: (i) platform(s) with explicitly defined ontology (or, at least, taxonomy); (ii) platform(s) with no explicitly defined ontology/taxonomy.

Acceptant criteria: INTER-METH supports the design of inter-platform semantic interoperability.

162: Specification of unique attributability

Category: Functional; Type: Functionality; Priority: Should

Rationale: Within a unique system of transparently interconnected IoT platforms, it should be possible to identify the different components affecting or contributing to the integrated system functionalities.

Description: INTER-METH is the methodology supporting the interconnection/interoperability of heterogeneous IoT platforms. However, it may happen that for different reasons there is the need of identifying the single platform contribution to the system functionalities (e.g. two interconnected platforms provide the same service but the user have to choose a particular service provider).

Acceptant criteria: IoT platforms identities and functionalities are distinguishable.

161: Model-driven support

Category: Functional; Type: Interoperability; Priority: Must

Rationale: The exploitation of common meta-models increases the compatibility between systems and simplifies the development processes.

Description: Model Driven Engineering (MDE) raises the level of abstraction in systems/programs specifications, facilitates the understandability of the main system concepts and increases automation in systems/programs development. INTER-METH approach is based on meta-models that are defined at different levels of abstraction to support the development phases of analysis, design, integration, and validation.

Acceptant criteria: Meta-models are available to support each development phase.

160: CASE-tool support

Category: Functional; Type: Requirement; Priority: Medium

Rationale: A CASE tool supporting the integrators (or developers of the platform integration) all over the integration process may facilitate the analysis, design, integration, realization and testing of high-quality and maintainable integrated heterogeneous IoT platforms.

Description: INTER-METH is supported by INTER-CASE (Computer Aided Software Engineering tool for integration) in order to foster the effective and efficient integration of different heterogeneous IoT platforms. It will help to automate each phase (analysis, design, implementation, deployment, test, maintenance) of the integration process by using INTER-METH, thus providing guidelines, graphical facilities, engineering patterns and methods, and data repositories. The need of a CASE-tool supporting and, if possible, automating such processes has been underlined by several stakeholders (e.g. Telefonica, DGCONNECT, ISECO, DISI-UNIBO, Symblote, etc.).

Acceptant criteria: The CASE-tool is developed according to the above mentioned functionalities.

159: Development support for systematic IoT platforms integration/interconnection

Category: Functional; Type: Interoperability; Priority: Must

Rationale: In order to avoid the proliferation of isolated silos, the IoT platforms integration/interconnection processes must be driven by a well-defined, organic and full-fledged methodology.

Description: It is widely recognized that using an engineering methodology is fundamental in any engineering application domain, since the manual and non-systematic application of complex techniques, methods and frameworks would very likely lead to an increase of the degree of errors during integration. INTER-METH provides a methodology to systematically support the development of voluntary interoperability among heterogeneous IoT platforms (even belonging to different domains). In this direction, by means of different guidelines, models, facilities and tools, INTER-METH supports the systematic IoT platforms integration/interconnection in the development phases of analysis, design and implementation. The needs of a “systematic methodology and a well-defined approach to support IoT interoperability at any level of abstraction and within every application domain" have been underlined by several stakeholders (e.g. THINGS, XLAB, SRIPAS, ABC, VEMCO, etc.).

Acceptant criteria: Integration of different platforms is driven by a common and well-defined methodology.

120: Effectiveness and Optimization

Category: Non Functional; Type: Functionality; Priority: Must

Rationale: INTER-METH must positively impact the IoT platforms integration processes.

Description: As highlighted by several stakeholders (CSE, ISECO, NEWAYS, INFOPORT, UNICAL), INTER-METH is expected to have a high-impact on “reducing development time and cost” but at the same time “enhancing effectiveness, agility and quality”. The IoT platforms integration processes must be optimized and strengthened by the INTER-METH application.

Acceptant criteria: INTER-METH makes IoT platforms integration processes faster, higher quality and lower resource waste.

119: Maintainability

Category: Non Functional; Type: Functionality; Priority: Must

Rationale: Modifications/Updating efforts are reasonable.

Description: The IoT scenario is highly dynamic and INTER-METH aims to play an important role even in the next years. With reference to integrated platforms, the effort needed to isolate and correct defects, to make future focused modifications or to cope with a changed environment/standard must be reasonable.

Acceptant criteria: Modification/Updates do not require unreasonable efforts.

118: Legal and licensing issues avoidability

Category: Non Functional; Type: Legality; Priority: Could

Rationale: Avoid third-part legal hassles.

Description: There may be legal issues involving privacy of information, intellectual property rights, export of restricted technologies, patent-infringement, etc. As suggested by the DISI-UNIBO stakeholder, the methodology should made heterogeneous IoT platforms and products interoperable but in a manner that it "takes care of the all technological, organizational, ethical and legal constraints".

Acceptant criteria: No complaints for patent/license/privacy violation are raised.

117: Security and Trust Management

Category: Functional; Type: Security; Priority: Should

Rationale: The driving idea for including security among the requirements for the INTER-METH product is the concept of "security-by-design" by which data, hardware and software protection should be embedded throughout the entire life cycle of the system development (or integration).

Description: INTER-METH takes into account security-related concerns, namely the set of hardware, software, procedures, and policies components for defending and controlling access to integration of devices, networks, data/information, and software against malicious entities and attacks. Here security is related to the integrated platforms. For example, trust issues refer to scenarios in which different integrated/interconnected platforms cooperate without previous collaboration history. Trust management enables to make sure that the shared data and services/operations are real and trustworthy, especially with crowdsourcing generated data and machine generated data.

Acceptant criteria: INTER-METH guides the integration of IoT systems/platform so preserving the overall security, trustability and protection to their devices, data, information, and software.

116: Privacy

Category: Functional; Type: Privacy; Priority: Should

Rationale: The driving idea for including privacy among the requirements for the INTER-METH product is the concept of "privacy-by-design" by which privacy and data protection should be embedded throughout the entire life cycle of the system development (or integration).

Description: INTER-METH takes into account the constraints deriving from the processing of personal and health data. INTER-METH defines and implements privacy policies to determine which information can be revealed in the integrated platform, who can access to such information, and for what purposes such information may be used.

Acceptant criteria: INTER-METH guides integration of IoT systems/platforms that successfully provides different degree of freedom defining appropriate privacy policies and eventually guarantee high end-to-end privacy levels.

115: Scalability Preserving

Category: Non Functional; Type: Architecture; Priority: Could

Rationale: INTER-METH should drive the integration of heterogeneous IoT systems/platforms in order to obtain the maximum degree of scalability.

Description: Scalability is a fundamental feature of any distributed platform/system. Thus, INTER-METH will provide support by-design to drive an integration of platforms that will result into an integrated platform that will preserve the maximum level of scalability.

Acceptant criteria: INTER-METH supports an integration of heterogeneous platforms that preserves scalability.

114: Extensibility and Customization

Category: Non Functional; Type: Usability; Priority: Should

Rationale: The methodology should be easily extensible and customizable for dealing with the integration of a wide spectrum of (even very different) IoT platforms.

Description: A robust methodology aiming at integrating different IoT platforms should be easily adaptable to any specific platform to be integrated. The extensibility and personalization features of such engineering methodology should allow a simple and fast integration among different heterogeneous non-interoperable IoT solutions.

Acceptant criteria: Fast and easy customization to different IoT platforms.

113: Error minimization

Category: Non Functional; Type: Functionality; Priority: Must

Rationale: Minimize errors (e.g. during system integration) is crucial to develop/implement valuable integrated IoT systems. The methodology should therefore comprise (formal) models and tools to guide designers/integrators in building correct-by-construction integrated system representations.

Description: At methodological level, the way to minimize errors is to follow a “correct-by-construction” approach such that the methodology itself helps the IoT system designer/integrators to construct a (formal) model before that any implementation/integration detail is produced. The model is used to reason about the proposed solutions, ensuring that all the required functional and non-functional requirements will be fulfilled and the correct behaviour of the integrated system exhibited.

Acceptant criteria: Testing procedures should be devoted to validate the correct-by-construction platforms integration and eventually conceptual errors should not be found at all in the designed system model.

112: Compliance

Category: Non Functional; Type: Functionality; Priority: Could

Rationale: Reuse of existing proven working standards.

Description: Over the years, several organizations (ISO, IEEE, IPC, OMG, etc.) provided successful standards and protocols. Some of them may be maintained and exploited from INTER-METH, as different stakeholders highlighted (e.g. OGC, CSE, PRO, Intel, AYAC, Agile, or SPEM). In fact, they may represent useful starting points and represent source of interoperability, avoiding unnecessary duplication of efforts.

Acceptant criteria: INTER-METH complies with standards considered consistent with the methodology itself.

111: Documentation

Category: Non Functional; Type: Usability; Priority: Must

Rationale: Documentation should be available to improve INTER-METH usability.

Description: As several stakeholders have highlighted (e.g. VPF, ABC, SRIPAS), INTER-METH should be supplied with a detailed documentation (in terms of "know-how knowledge, guidance, step-by-step instructions") to improve its usability. Documentation will be based on standard notations for software/system engineering methodologies (e.g. SPEM by OMG).

Acceptant criteria: Complete and detailed documentation is delivered also according to a standard notation.

110: Usability

Category: Non Functional; Type: Usability; Priority: Must

Rationale: Understandability, learnability and attractiveness facilitate a broad adoption of an engineering methodology.

Description: INTER-METH must be as much as possible easy to be understood, learned and used. This implies that the users' training effort must be reasonable. Moreover, INTER-METH must be attractive in order to positively influence other organizations to reach a broad adoption. Several stakeholders such as UPV and GIS (multinational solutions provider for government and institutions) highlighted this important aspect.

Acceptant criteria: The methodology is usable according to well-established usability evaluation procedures.

109: Stability

Category: Non Functional; Type: Functionality; Priority: Should

Rationale: The methodology core will be stable over time and will not need immediate changes.

Description: INTER-METH must be released in a (reasonably) stable version in order to avoid frequent changes and to provide long-term support to IoT systems/platforms integration/interconnection and to integrated/interconnected IoT systems/platforms.

Acceptant criteria: INTER-METH released version is stable.

108: Open Source

Category: Non Functional; Type: Commercial, Usability; Priority: Must

Rationale: European Commission prefers Open Source project.

Description: An extract of the European Commission strategy in matter of software products is here reported: "Software produced by the Commission services, in particular software produced with the objective of being used outside the Commission, will be open sourced and published on the Join up platform and will use the

European Union Public License (EUPL)". In general, open source projects may rely on wide and active communities of users and developers (in particular, THINGS underlines the need of "Open API" to improve usability, extensibility and interoperability). According to this vision the INTER-Meth documentation, the INTER-CASE software tool and related documentation should be open.

Acceptant criteria: Source code of the INTER-CASE tool is available on INTER-IoT website.

3.3 Use Cases

Four main use cases, reported in Table 2.3.1, have been identified for INTER-METH. In the following subsections the use cases will be described in detail. The use cases provide the abstract activities/tasks to be performed in the INTER-METH process phases.

Table 2.3.1: The main four INTER-METH Use Cases

Id	Use Case
IM1	IoT Platform Integration Requirements Analysis
IM2	IoT Platforms Integration Design
IM3	IoT Platforms Integration Implementation (including deployment and testing)
IM4	IoT Platforms Integration Maintenance

3.3.1 IoT Platform Integration Requirements Analysis (IM1)

Description: Given two or more IoT platforms/systems to be integrated, the integration requirements need to be elicited. On the basis of the elicited requirements, the design of the IoT platforms integration could be then carried out.

Involved Platform Layers: Data & semantics, Service, Middleware, Network, Device, Cross-layer

Objectives: To elicit the requirements for the integration of IoT platforms/systems.

Actors: The actors are:

(a) The developer of the integration (aka Integrator), who carries out the integration interconnection of heterogeneous IoT platforms;

(b) The platform Owner, who will obtain the integrated platform;

(c) The involved Platforms to be integrated.

Pre-conditions: Two or more heterogeneous IoT platforms (available) to be interconnected/integrated.

Trigger: The need to integrate/interconnect identified heterogeneous IoT platforms

Expected results: Set of (functional and non-functional) requirements for the integration of the identified IoT platforms/systems

Notes and Issues: The involved IoT platforms should be well-documented and/or the IoT platform developer/s should be available to provide technical support.

Main execution:

1. Each platform is analysed according to the functional and non-functional viewpoints of the five IoT platform layers (device, networking, middleware, application services, data & semantics) and of the cross-layering.

2. According to the Step 1, the requirements of integration among the layers of the platforms to be integrated are defined according to an iterative process.

The execution could be supported by the INTER-CASE tool.

Requirements: [74][159-162][108-120] (see Section 3.2)

3.3.2 IoT Platforms Integration Design (IM2)

Description: Given two or more IoT platforms/systems that have been analysed according to the use case IM1, design specifications have to be produced. On the basis of the design specifications, the implementation of the IoT platforms integration could then be carried out.

Involved Platform Layers: Data & semantics, Service, Middleware, Network, Device, Cross-layer.

Objectives: To define the design specifications for the integration of IoT platforms/systems.

Actors: The actors are:

(a) the developer of the integration (aka Integrator), who carries out the integration interconnection of heterogeneous IoT platforms;

(b) the platform Owner, who will obtain the integrated platform;

(c) the involved Platforms to be integrated.

Pre-conditions: Two or more heterogeneous IoT platforms (available) to be interconnected/integrated, analyzed through the use case IM1.

Trigger: Availability of the specific requirements for the integration of the involved IoT platforms/systems elicited through the use case IM1 (or even through another methodology/process).

Expected results: Set of design specifications for the integration of the identified IoT platforms/systems.

Notes and Issues: The involved IoT platforms should be well-documented (or even open-source) and/or the IoT platform developer/s should be available to provide technical support.

Main execution:

1. For each layer (and cross-layer), on the basis of the elicited requirements in IM1, an initial design specification is produced.
2. Each design specification produced in Step 1 is iteratively refined.
3. A global integration design is defined on the basis of the outcome of Step 2.

The execution could be supported by the INTER-CASE tool.

Requirements: [74][159-162][108-120] (see Section 3.2).

3.3.3 IoT Platforms Integration Implementation (IM3)

Description: Given two or more IoT platforms/systems, whose integration has been designed according to the use case IM2, the integration implementation (deployment and testing/validation) has to be performed. On the basis of the actual deployed and tested implementation, the maintenance of the integrated IoT platforms could then be realised.

Involved Platform Layers: Data & semantics, Service, Middleware, Network, Device, Cross-layer.

Objectives: To integrate/interconnect, deploy and test the IoT platforms/systems to be integrated/interconnected.

Actors: The actors are:

(a) the developer of the integration (aka Integrator), who carries out the integration interconnection of heterogeneous IoT platforms;

(b) the platform Owner, who will obtain the integrated platform;

(c) the involved Platforms to be integrated.

Pre-conditions: Two or more heterogeneous IoT platforms (available) to be interconnected/integrated, whose integration design specification are analyzed through use case IM2.

Trigger: Availability of the design integration specifications of the involved IoT platforms/systems defined through the use case IM2 (or even through another methodology/process).

Expected results: The integrated, deployed and tested IoT platform/system.

Notes and Issues: The involved IoT platforms should be well-documented (or even open-source) and/or the IoT platform developer/s should be available to provide technical support.

Main execution:

1. For each layer (and cross-layer), the design specifications produced in IM2 are actually implemented.
2. On the basis of Step 1, a full-fledged integration (namely Integrated Platform) among the involved IoT platforms/system will be obtained.
3. The Integrated Platform is deployed.
4. The Integrated Platform is validated through testing.

The execution could be supported by the INTER-CASE tool.

Requirements: [74][159-162][108-120] (see Section 3.2)

3.3.4 IoT Platforms Integration Maintenance (IM4)

Description: Given an integrated platform obtained from the integration of two or more IoT platforms/systems, such platform needs to be maintained.

Involved Platform Layers: Data & semantics, Service, Middleware, Network, Device, Cross-layer.

Objectives: To maintain and make evolve an integrated IoT platform.

Actors: The actors are:

(a) the developer of the integration (aka Integrator), who carries out the integration interconnection of heterogeneous IoT platforms;

(b) the platform Owner, who will obtain the integrated platform;

(c) the involved Platforms to be integrated;

(d) the integrated Platform.

Pre-conditions: An integrated platform implemented (and deployed and tested) through use case IM3.

Trigger: Availability of an integrated platform defined through the use case IM3 (or even through another methodology/process).

Expected results: Continuous maintenance of the integrated IoT platform.

Notes and Issues: The involved integrated IoT platform should be well-documented (or even open-source) and/or the IoT platform developer/s and integrator/s should be available to provide technical support.

Main execution:

1. Identification of a list of bugs and/or a list of evolution points.
2. Correction of bugs and/or implementation of new functionalities.

The execution could be supported by the INTER-CASE tool.

Requirements: [74][159-162][108-120] (see Section 3.2)

3.4 Scenarios

In this section, we report two main scenarios: a) integration of two (or more) platforms, illustrating the integration activities to obtain an integrated platform from two heterogeneous ones; b) reverse engineering of the integration of two (or more) platforms by integrating them also at another different integration layer.

3.4.1 Heterogeneous Platforms Methodology-driven Integration

<table>
<tr><th colspan="3">Business Scenario</th></tr>
<tr><td>Scenario ID
METH#1</td><td colspan="2">Scenario Name
Heterogeneous Platforms Methodology-driven Integration</td></tr>
<tr><td rowspan="6">Illustration of system's behaviour in a specific situation, flow of events</td><td colspan="2">Mark is a System Integrator and is in charge to integrate two IoT platforms for exploiting the features of both platforms to develop novel IoT applications. Mark therefore uses INTER-METH, following its process made up of interconnected phases. Each phase is a workflow of task that Mark needs to execute. Thus, Mark needs to: 1-Analysis) analyze the two platforms in order to elicit the functional and non-functional requirements according to the integration goals; 2-Design) design the integration solution based on the elicited requirements; 3-Implementation) implement the design to obtain the integrated platform; 4-Deployment) configure and execute the integrated platform; 5-Testing) test the deployed integrated platform according to test cases to validate functional and non-functional requirements; 6-Maintenance) maintain the integrated platform, i.e. bug fixing and platform evolution.</td></tr>
<tr><td>User/users:</td><td>The user is an IoT System Integrator.</td></tr>
<tr><td>Setting / context</td><td>The context is represented by a System-of-System context based on the IoT platforms to be integrated.</td></tr>
<tr><td>Interacting system</td><td>The technical environment is strongly dependent on the IoT platforms to be made interoperable (or to be integrated).</td></tr>
<tr><td>Users' goals</td><td>The user's aim is to integrate two or more IoT platforms.</td></tr>
<tr><td>Interaction</td><td>The IoT System Integrator uses the methodology. The methodology is not a system with which it is possible to interact. However, INTER-METH will support the IoT System Integrator step-by-step to fulfill its integration goals.</td></tr>
</table>

	Initial status	*The initial conditions are the availability of the IoT platforms to be integrated and the integration goals of the IoT System Integrator.*
	Workproduct	*The workproducts of the methodology are different at the different phases. Considering the whole process: from non-integrated IoT platforms and integration goals to integrated platforms ready to be deployed and tested.*
	Motivation	*To create a more capable and powerful IoT platform starting from two or more heterogeneous ones.*
	Time	*The integration takes place according to specific needs so timing is not fixed but random.*
Interoperability Role	*Interoperability role is by goal, i.e. heterogeneous platforms are integrated according to integration goals. Thus, the interoperability role is the key enabling factor in INTER-METH.*	
Market and usage data available	*To the best of our knowledge, no methodology for IoT systems/platforms integration is available.*	
Business model	*The business model can be built around the IoT System Integrator as a new ICT professional job to support the IoT world.*	
Missing technical know how / input	*None.*	
Partner specific interests	*Definition of a general-purpose methodology for IoT system integration.*	

Product:	**Identified by:**	**Registration Date:**
INTER-METH	*UNICAL*	*1/12/2016* *1/12/2016*

3.4.2 Re-engineering Integrated IoT platforms

Business Scenario	
Scenario ID METH#2	**Scenario Name** Re-engineering Integrated IoT platforms
Illustration of system's behaviour in a specific situation, flow of events	*Mark is a System Integrator and is in charge to an integrated IoT platform, i.e. obtained according to the Scenario METH#1. Specifically, Mark has an additional set of Integration Goals. During the Analysis phase, Mark will identify one or more new integration points according to the integration goals and translate them into new functional and non-functional requirements. With reference to the INTER-IoT approach, the new identified integration goals could refer to one or more layers to be integrated.*
	User/users: *The user is an IoT System Integrator that uses INTER-METH.*

	Setting / context	*IoT Platforms integrated using INTER-METH.*
	Interacting system	*A specific integrated IoT platform.*
	Users' goals	*Upgrade the Integrated IoT Platform by considering additional IoT layers.*
	Interaction	*Use of INTER-METH*
	Initial status	*The Integrated IoT Platform and an additional set of integration goals*
	Workproduct	*All new workproducts to be produced due to the fulfillment of the new integration.*
	Motivation	*To upgrade an integrated IoT with new capabilities.*
	Time	*The integration upgrade takes place according to specific needs so timing is not fixed but random*
Interoperability Role	*Interoperability of the upgraded integrated IoT platform must be kept.*	
Market and usage data available	*To the best of our knowledge, no methodology for IoT systems/platforms integration is available. Thus, no upgrade method for integrated platform is also available.*	
Business model	*The business model can be built around the IoT System Integrator as a new ICT professional job to support the IoT world.*	
Missing technical know how / input	*None.*	
Partner specific interests	*Definition of a general-purpose methodology for IoT system integration.*	

Product:	**Identified by:**	**Registration Date:**
INTER-METH	*UNICAL*	*1/12/2016* *1/12/2016*

4 INTER-METH Abstract Process

The engineering methodology INTER-METH aims at supporting the integration process of heterogeneous IoT platforms to obtain interoperability among them and allow implementation and deployment of IoT applications on top of them. To date, no proposals in the IoT area provided a systematic methodology driving the integration of heterogeneous IoT platforms. It is widely recognized that using an engineering methodology is fundamental in any engineering application domain (e.g. software engineering, co-design hardware/software, systems of systems, civil engineering, etc.). In fact, the manual and non-systematic application of complex techniques, methods and frameworks would very likely lead to an increase of the degree of errors during the integration process.

In this section we introduce the abstract process of INTER-METH whose SPEM-based schema is shown in Figure 4.1. SPEM overview provided in Appendix A supports the reader in understanding SPEM diagrams. The process is envisioned as *iterative waterfall*, including the following six phases: *Analysis*, *Design*, *Implementation*, *Deployment*, *Testing* and *Maintenance*. Each phase produces work-products (requirements, design diagrams, coded platforms, system deployment, and validation results) that are inputs for the successive phase/s. Iteration could involve single phases, set of successive phases or the whole process, thus assuring adaptability to new requirements.

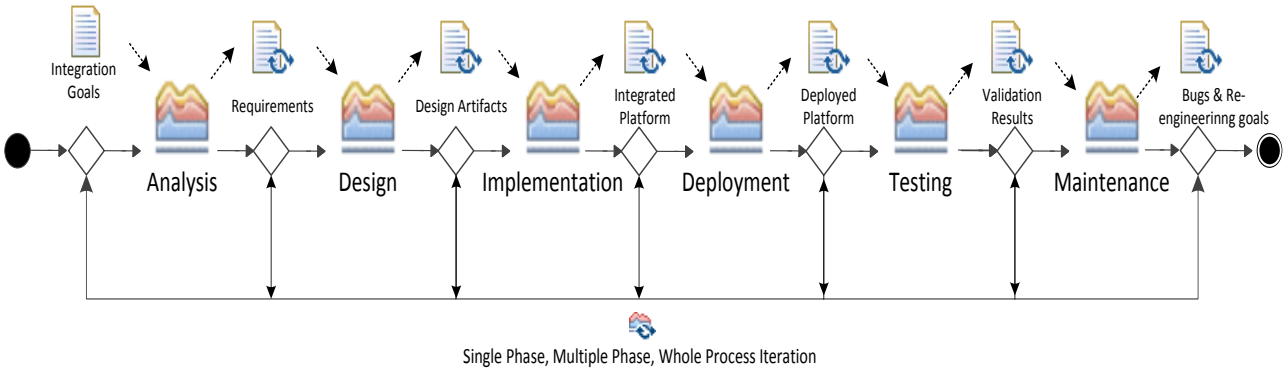

Figure 4.1: INTER-METH Abstract Process Schema

In particular:

- The *Analysis* phase formalizes the integration requirements, both functional and non-functional (e.g. real-timeliness, reliability, security, privacy, trust).
- The *Design* phase produces the design of the integration in terms of diagrams of (i) interoperability layer infrastructures and related interfaces, and (ii) programming and management patterns, to fulfil the elicited requirements.
- The *Implementation* phase focuses on the implementation of the design work-product/s to obtain the full-integrated (hardware and/or software) system.
- The *Deployment* phase involves the definition of the operating set-up and of the configuration of the integrated IoT platform.
- The *Testing* phase allows defining and performing tests to validate the integrated platform according to the functional and non-functional requirements.
- The *Maintenance* phase manages the upgrade and evolution of the system.

The proposed abstract process could be associated to any specific IoT systems integration approach. The instantiation of such process for INTER-IoT (thus strongly connected to INTER-LAYER and INTER-FW) is presented in Section 4.

In the following subsections each phase will be detailed.

4.1 Phase 1: Analysis of Integration Requirements

Description: Given two or more IoT platforms/systems to be integrated, the integration requirements need to be elicited. On the basis of the elicited requirements, the design of the IoT platforms integration could be then carried out.

Objectives: To elicit the requirements for the integration of IoT platforms/systems.

Actors: The actors are:

(a) The developer of the integration (aka Integrator), who carries out the integration interconnection of heterogeneous IoT platforms; the Integrator is an active performer.

(b) The platform Owner, who will obtain the integrated platform;

(c) The involved Platforms to be integrated.

Expected results: Set of (functional and non-functional) requirements for the integration of the identified IoT platforms/systems.

Main execution:

1. On the basis of the *Integration Goals*, each platform is analysed according to the functional and non-functional viewpoints of the five IoT platform layers (device, networking, middleware, application services, data & semantics) and of the cross-layering.
2. According to the Step 1, the requirements of integration among the layers of the platforms to be integrated are defined according to iterative tasks enclosed in activities.

4.1.1 Activities

The Requirements Analysis activity, which is the only main activity of the Analysis phase, is subdivided into three main tasks that are performed by the Integrator (see Figure 4.1.1 and Table 4.1.1) according to the workflow depicted in Figure 4.1.2:

1. *IoT Platforms Analysis*: each platform/systems to be integrated/interconnected is analyzed in terms of the 5 reference layers (device, networking, middleware, application services, and data & semantics) and of the cross-layer functionalities. Such analysis will produce a well-formalized analysis document (Analyzed Platforms Document).
2. *Functional Requirements Elicitation*: On the basis of the Integration Goals document and of the Analyzed Platforms Document, the functional requirements are elicited and included in the Functional Requirements document.
3. *Non-functional Requirements Elicitation*: On the basis of the Integration Goals document, the IoT Platforms Analysis document, and the Functional Requirements document, the non-functional requirements are elicited and included in the Non-functional Requirements document.

The functional and non-functional requirements are finally merged by the task *Requirements Merger* in the Functional and non-functional document final activity work-product.

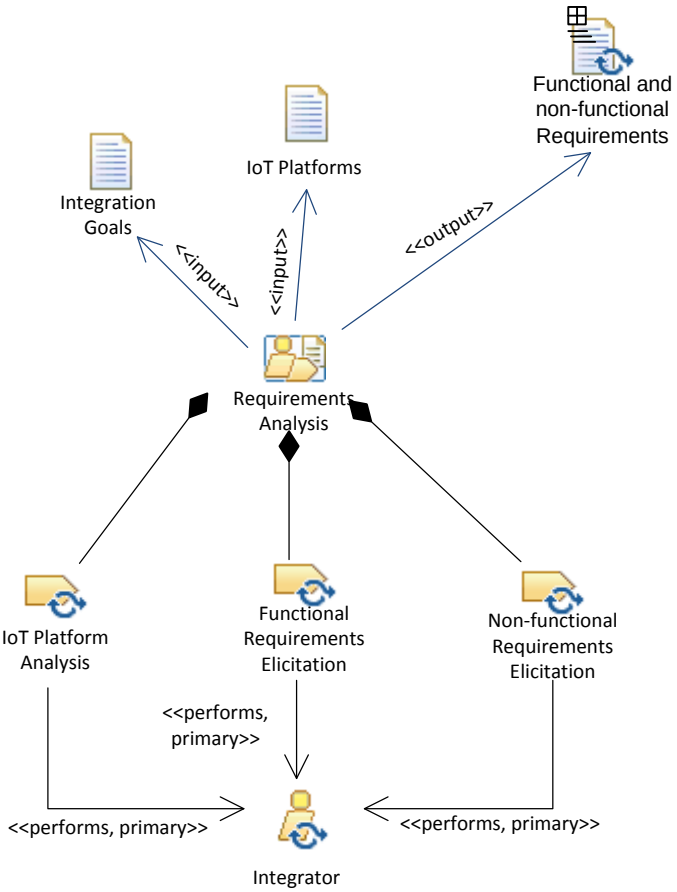

Figure 4.1.1: The Analysis phase described in terms of activities, roles, and work products

Table 4.1.1: Tasks of the Requirements Analysis activity

Activity	Task	Task description	Role involved
Requirement Analysis	IoT Platforms Analysis	Analysis of the platform/systems to be integrated/interconnected	Integrator
Requirement Analysis	Functional Requirements Elicitation	Definition of the Functional Requirements for IoT platforms integration	Integrator
Requirement Analysis	Non-functional Requirements Elicitation	Definition of the Non-functional Requirements for IoT platforms integration	Integrator
Requirement Analysis	Requirements Merger	Merging the function and non-functional requirements into the final work-product	Integrator (or automatic)

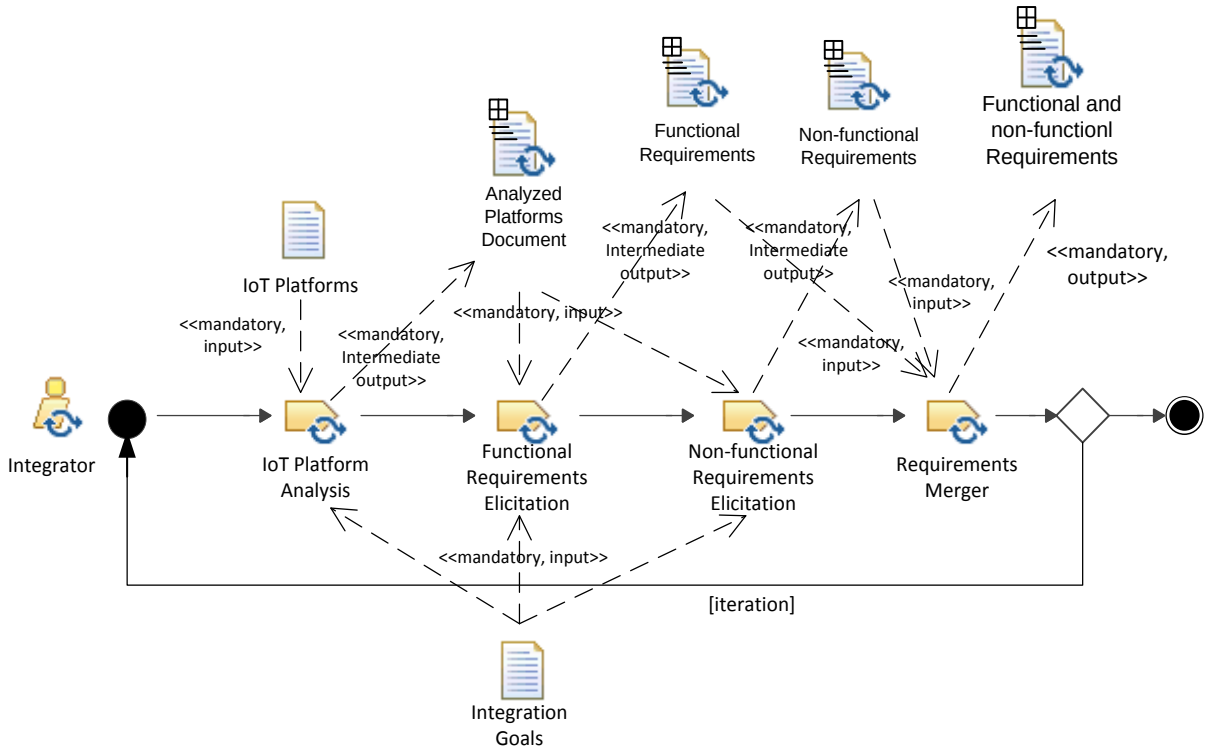

Figure 4.1.2: The workflow of tasks of the Requirement Analysis activity

4.1.2 Work Products

The final output work-product is a formalized document containing the functional and non-functional integration requirements. The document format is not specified now as it will be specified by the specific instantiation of the abstract INTER-METH methodology (see Section 5). Intermediate work-products are: (a) the Analyzed Platforms Document, containing the layer-oriented analysis of the platform to be integrated/interconnected (this work-product will be also used as input in the Design phase); (b) the Functional Requirements Document; (c) the Non-functional Requirements Document. Also the specification format of the Analyzed Platforms Document is not given now.

4.2 Phase 2: Design of the Systems Integration

Description: Given two or more IoT platforms/systems that have been analysed according to the Analysis phase, design specifications have to be produced. On the basis of the design specifications, the implementation of the IoT platforms integration could then be carried out.

Objectives: To define the design specifications for the integration of IoT platforms/systems.

Actors: The actors are:

(a) the developer of the integration (aka Integrator), who carries out the integration interconnection of heterogeneous IoT platforms; the Integrator is an active performer.

(b) the platform Owner, who will obtain the integrated platform;

(c) the involved Platforms to be integrated.

Expected results: Set of design specifications for the integration of the identified IoT platforms/systems.

Main execution:

1. For each layer, on the basis of the elicited requirements in IM1, an initial design specification is produced.
2. Each design specification produced in Step 1 is iteratively refined.
3. A global integration design (including the cross-layer) is defined on the basis of the outcome of Step 2.

4.2.1 Activities

The Integration Design activity, which is the only main activity of the Design phase, is subdivided into two main tasks that are performed by the Integrator (see Figure 4.2.1 and Table 4.2.1) according to the workflow depicted in Figure 4.2.2:

1. *Layer Integration Design Specification*: on the basis of the Functional and non-functional Requirements Document, the Analyzed Platforms Document, and the IoT Platforms documentation, for each layer (device, networking, middleware, application services, and data & semantics) a layer integration specification is iteratively defined. Such task will produce five specifications: device integration specification, networking integration specification, middleware integration specification, application services integration specification, and data & semantics integration specification.
2. *Full Integration Design Specification*: On the basis of the five specification produced in the previous task, a full-fledged specification is iteratively produced, incorporating the cross-layer functionalities and properties.

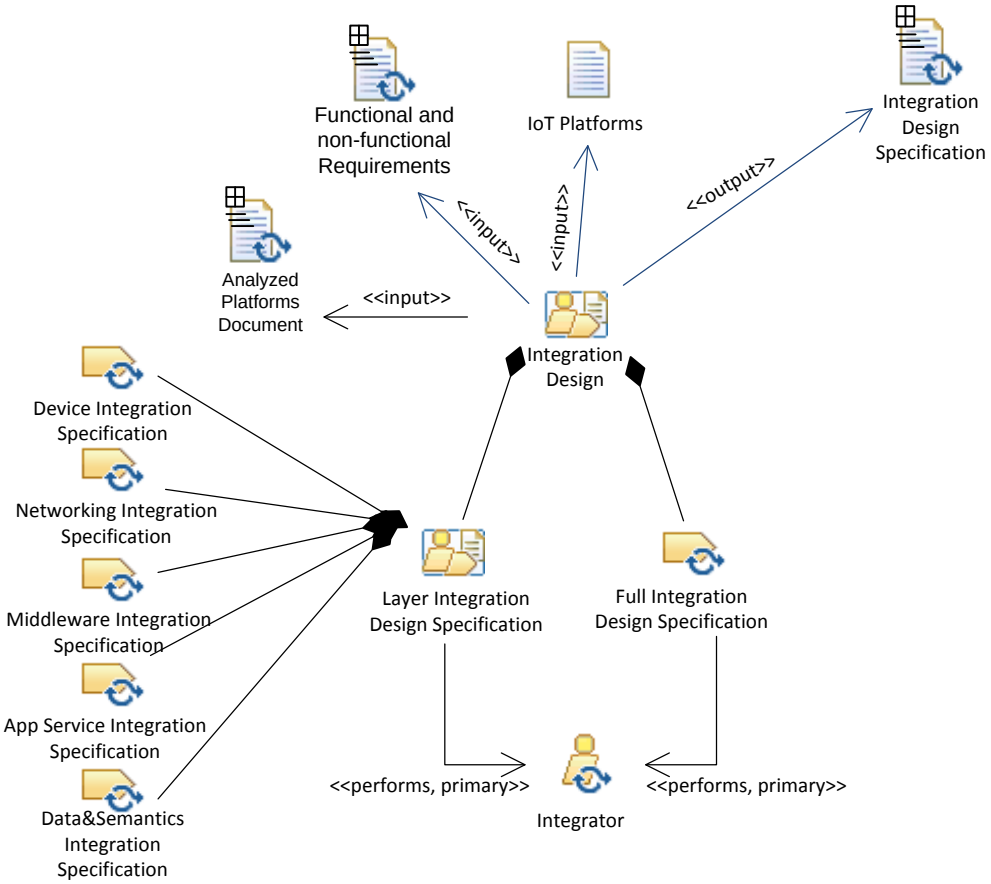

Figure 4.2.1: The Design phase described in terms of activities, roles, and work products

Table 4.2.1: Tasks of the Integration Design activity

Activity	Task	Task description	Role involved
Integration Design	Layer Integration Design Specification	A layer integration specification is iteratively defined for each layer	Integrator
Integration Design	Full Integration Design Specification	A full-fledged integration specification is produced including the five layers and the cross-layering	Integrator

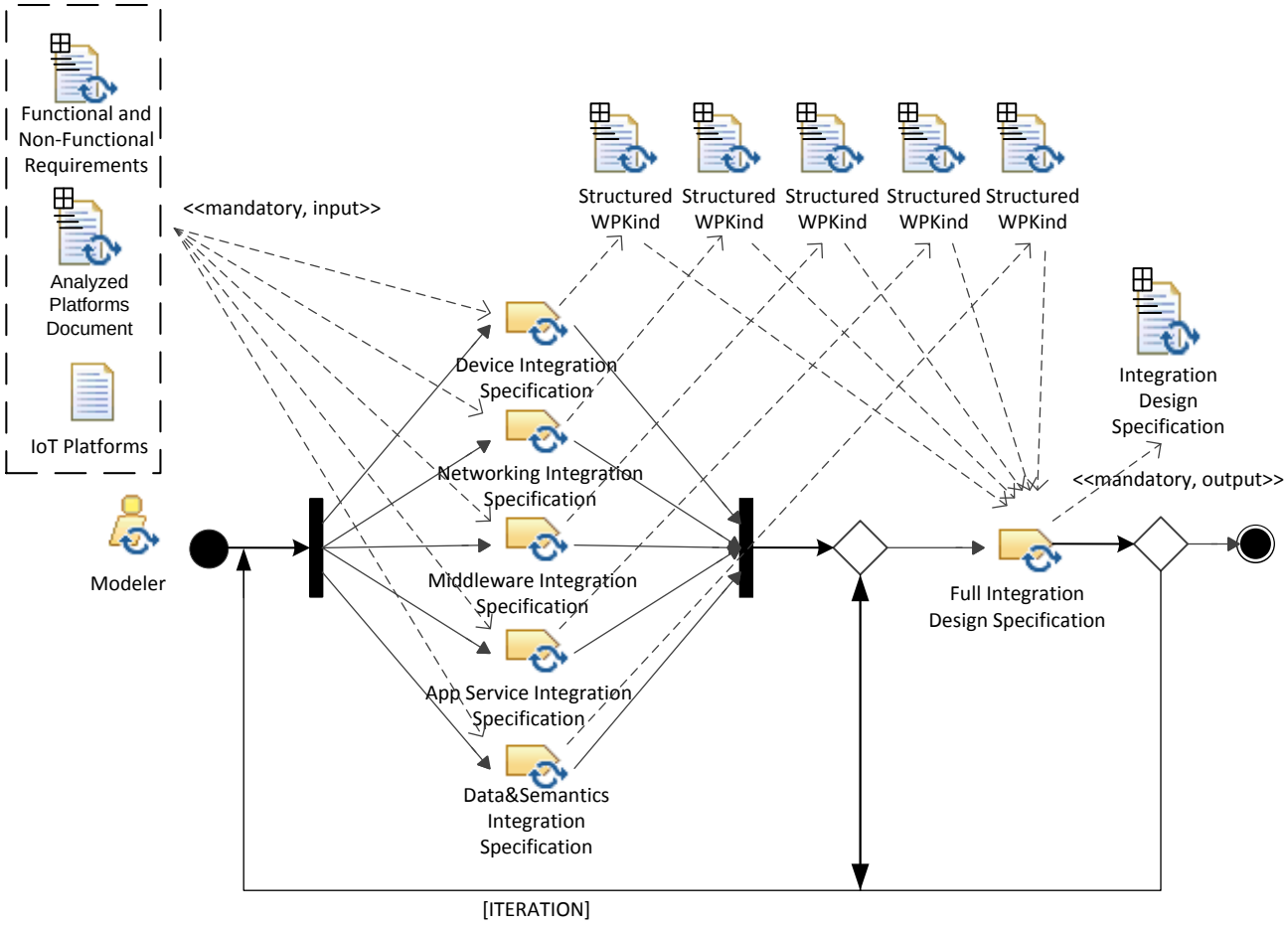

Figure 4.2.2: The workflow of tasks of the Design Integration activity

4.2.2 Work Products

The final output work-product is a formalized specification containing the design of the integration of the IoT platforms/systems to be interconnected/integrated. The document format is not specified now as it will be specified by the specific instantiation of the abstract INTER-METH methodology (see Section 5). Intermediate work-products are the five layer integration specifications. Also the format of such specification is not given now.

4.3 Phase 3: Implementation of the Systems Integration

Description: Given two or more IoT platforms/systems, whose integration has been designed according to the Design phase, the integration implementation has to be performed.

Objectives: To integrate/interconnect the considered IoT platforms/systems.

Actors: The actors are:

(a) the developer of the integration (aka Integrator), who carries out the integration interconnection of heterogeneous IoT platforms; the Integrator is an active performer.

(b) the platform Owner, who will obtain the integrated platform;

(c) the involved Platforms to be integrated.

Expected results: The integrated IoT platform/system.

Main execution:

1. The full-fledged design specification work-product produced in the Design phase is actually implemented according to multiple refinement steps.

4.3.1 Activities

The Integration Implementation activity, which is the only main activity of the Implementation phase, is subdivided into one main task that is iteratively performed by the Integrator (see Figure 4.3.1 and Table 4.3.1) according to the workflow depicted in Figure 4.3.2:

1. *Systems Integration*: on the basis of the Integration Design Specification, the heterogeneous IoT platforms/systems are integrated by the Integrator in an iterative activity (see workflow in Figure 4.3.3) aimed at obtaining the final integrated platform.

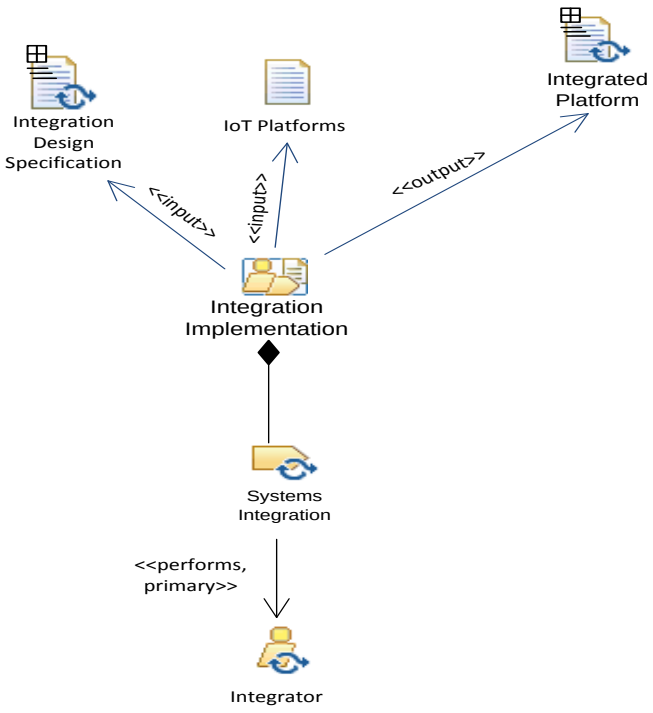

Figure 4.3.1: The Implementation phase described in terms of activities, roles, and work products

Table 4.3.1: Tasks of the Integration Implementation activity

Activity	Task	Task description	Role involved
Integration Implementation	Systems Integration	Heterogeneous platforms are integrated according to the integration specifications	Integrator

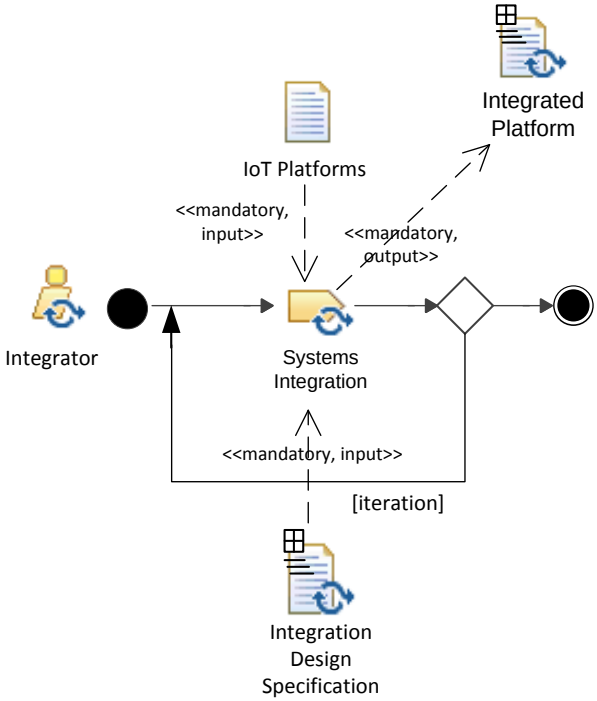

Figure 4.3.2: The workflow of tasks of the Integration Implementation activity

4.3.2 Work Products

The final output work-product is the integrated platform. The integrated platform will be based on the specific IoT platforms/systems to be integrated/interconnected and on the instantiation of the abstract INTER-METH methodology (see Section 5) and thus based specifically on INTER-LAYER and INTER-FW.

4.4 Phase 4: Deployment of the Integrated Platform

Description: Given two or more IoT platforms/systems, whose integration has been implemented according to the Implementation phase, the deployment of the integrated platform has to be performed.

Objectives: To deploy the integrated IoT platforms/systems.

Actors: The actors are:

(a) the developer of the integration (aka Integrator), who carries out the integration interconnection of heterogeneous IoT platforms; the Integrator is an active performer.

(b) the platform Owner, who will obtain the integrated platform; the Owner is an active performer.

(c) the involved Platforms to be integrated.

Expected results: The deployed IoT platform/system.

Main execution:

1. The Integrated Platform is deployed according to deployment goals and requirements.

4.4.1 Activities

The Integrated Platform Deployment activity, which is the only main activity of the Deployment phase, is subdivided into two main tasks that are performed by the Integrator and/or the Owner (see Figure 4.4.1 and Table 4.4.1) according to the workflow depicted in Figure 4.4.2:

1. *Deployment Configuration Definition*: to enable system deployment, the configuration of the integrated platform obtained in the previous Implementation phase has to be defined. The outcome of this task is a full-fledged specification of the configuration of the deployed system/platform.
2. *Deployment & Run*: the Integrated IoT Platform is actually deployed and run.

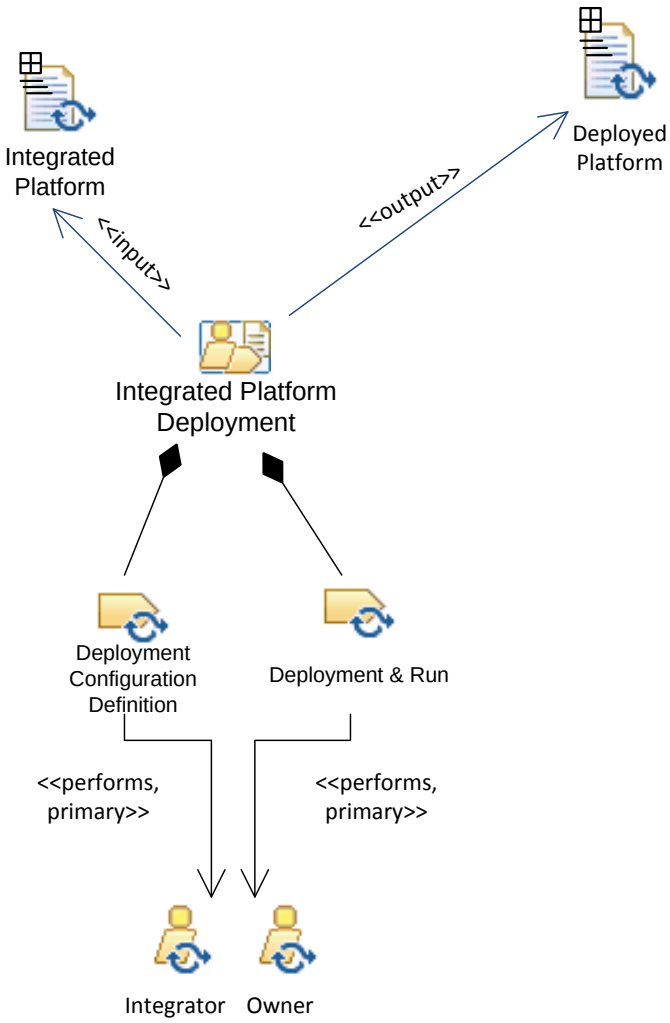

Figure 4.4.1: The Deployment phase described in terms of activities, roles, and work products

Table 4.4.1: Tasks of the Integrated Platform Deployment activity

Activity	Task	Task description	Role involved
Integrated Platform Deployment	Deployment Configuration Definition	Definition of the configuration for the integrated platform deployment	Integrator and/or Owner
Integrated Platform Deployment	Deployment & Run	Actual deployment and execution of the integrated platform	Integrator and/or Owner

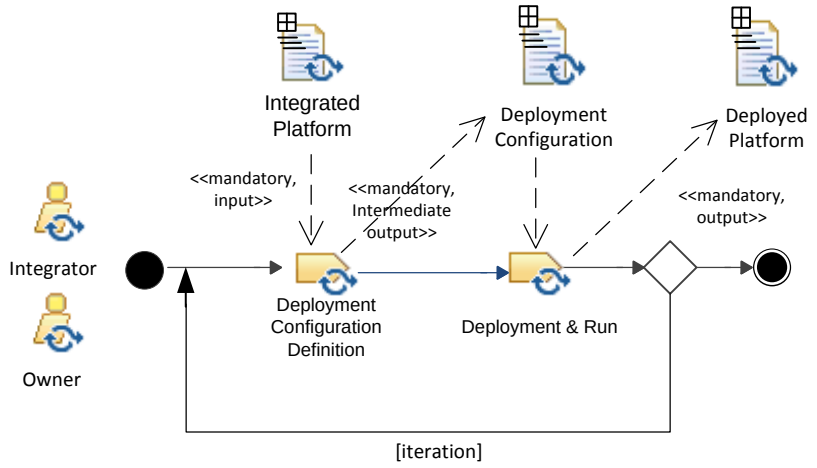

Figure 4.4.2: The workflow of tasks of the Integrated Platform Deployment activity

4.4.2 Work Products

The final output work-product is the deployed platform. The deployed platform will be set-up according to the defined configuration and then run.

4.5 Phase 5: Testing of the Integrated Platform

Description: Given two or more IoT platforms/systems, whose deployment has been carried out according to Deployment phase, the testing/validation has to be performed.

Objectives: To test the deployed integrated IoT platforms/systems.

Actors: The actors are:

(a) the developer of the integration (aka Integrator), who carries out the integration interconnection of heterogeneous IoT platforms; the Integrator is an active performer.

(b) the platform Owner, who will obtain the integrated platform; the Owner is an active performer.

(c) the involved Platforms to be integrated.

Expected results: The test results on the integrated and deployed IoT platform/system.

Main execution:

1. The integrated and deployed platform is executed and validated through testing according to well-defined test cases.

4.5.1 Activities

The Integrated Platform Testing activity, which is the only main activity of the Testing phase, is subdivided into two main tasks that are performed by the Integrator and/or the Owner (see Figure 4.5.1 and Table 4.5.1) according to the workflow depicted in Figure 4.5.2:

1. *Test Cases Definition*: the definition of test cases is fundamental to test the integrated deployed platform defined in the Deployment phase. Specifically, functional and non-functional test cases are defined to respectively validate functional and non-functional requirements.
2. *Platform Testing*: the test cases are executed by the platform and results collected according to well-formalized analysis documents.

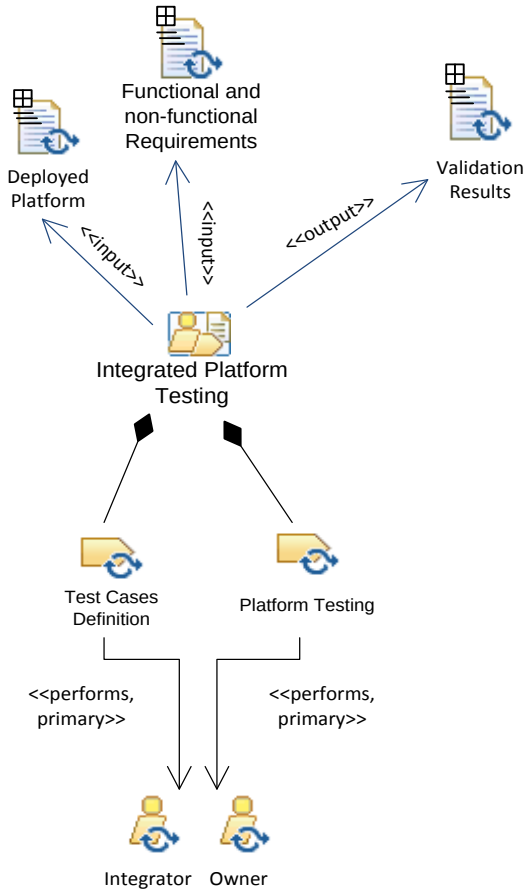

Figure 4.5.1: The Testing phase described in terms of activities, roles, and work products

Table 4.5.1: Tasks of the Integrated Platform Testing activity

Activity	Task	Task description	Role involved
Integrated Platform Testing	Test Cases Definition	Definition of the Test Cases to validate the integrated platform	Integrator and/or Owner
Integrated Platform Testing	Platform Testing	Validation of the platform by executing the defined Test Cases	Integrator and/or Owner

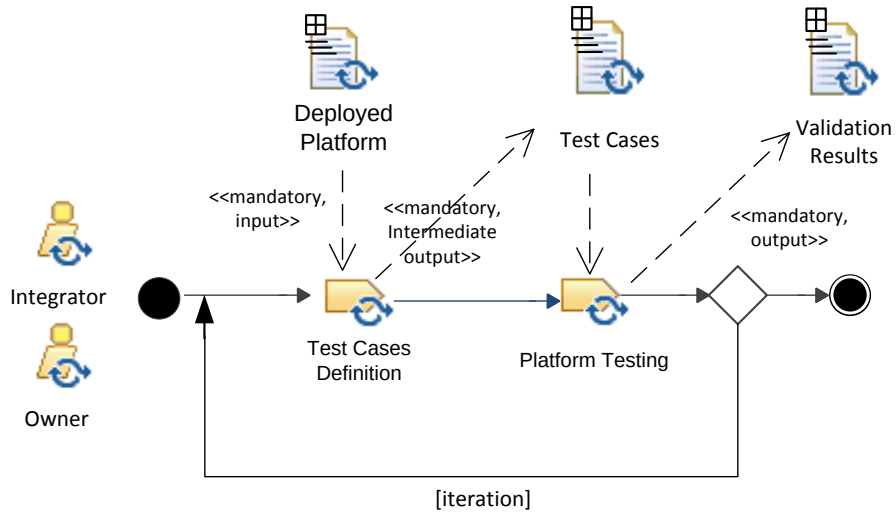

Figure 4.5.2: The workflow of tasks of the Integrated Platform Testing activity

4.5.2 Work Products

The final output work-product is the results of the validation in terms of the defined Test Cases parameters.

4.6 Phase 6: Maintenance of the Integrated Platform

Description: Given an integrated platform obtained from the integration of two or more IoT platforms/systems according to the previous phases, such platform needs to be maintained.

Objectives: To maintain and make evolve an integrated IoT platform.

Actors: The actors are:

(a) the developer of the integration (aka Integrator), who carries out the integration interconnection of heterogeneous IoT platforms; the Integrator is an active performer.

(b) the platform Owner, who will obtain the integrated platform; the Owner is an active performer.

(c) the involved Platforms to be integrated;

(d) the integrated Platform.

Expected results: Continuous maintenance of the integrated IoT platform.

Main execution:

1. *Identification of a list of bugs* and/or *a list of evolution points*.
2. *Correction of bugs* and/or *implementation of new functionalities* (go back to the Analysis or Design phases).

4.6.1 Activities

The Integrated Platform Maintenance activity, which is the only main activity of the Maintenance phase, is subdivided into two main tasks that are performed by the Integrator and/or Owner (see Figure 4.6.1 and Table 4.6.1) according to the workflow depicted in Figure 4.6.2:

1. *Change Identification*: this task aims at identifying bugs and/or evolution points of the integrated platform.
2. Change *Implementation*: actual implementation of the changes, i.e. bug fixing or analysis, design, implementation, deployment, and validation of new functionalities; the latter implies to re-execute, totally or partially, the integration process.

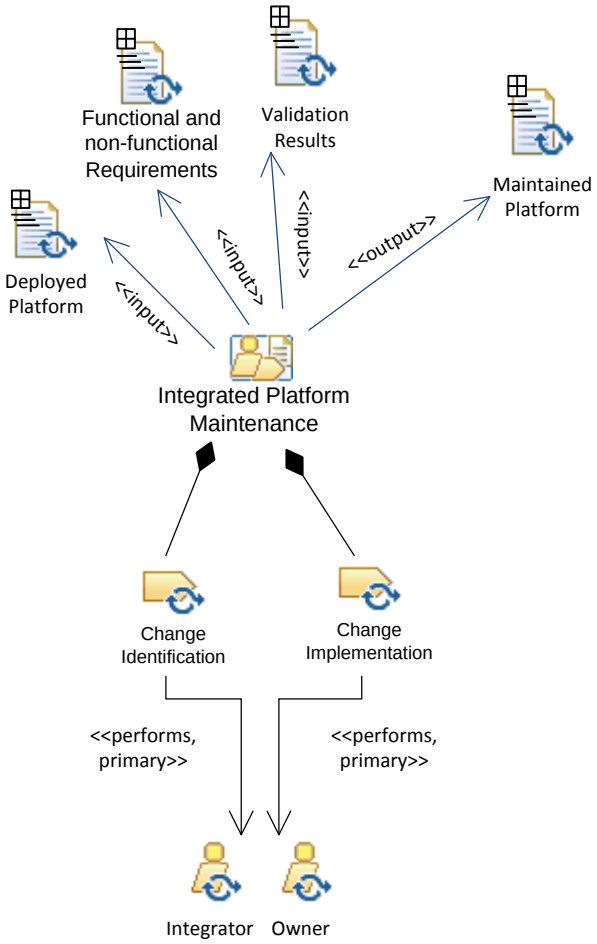

Figure 4.6.1: The Maintenance phase described in terms of activities, roles, and work products

Table 4.6.1: Tasks of the Integrated Platform Maintenance activity

Activity	Task	Task description	Role involved
Integrated Platform Maintenance	Change Identification	Identification of changes in the integrated platform	Integrator and/or Owner
Integrated Platform Maintenance	Change Implementation	Implementation of changes in the integrated platform	Integrator and/or Owner

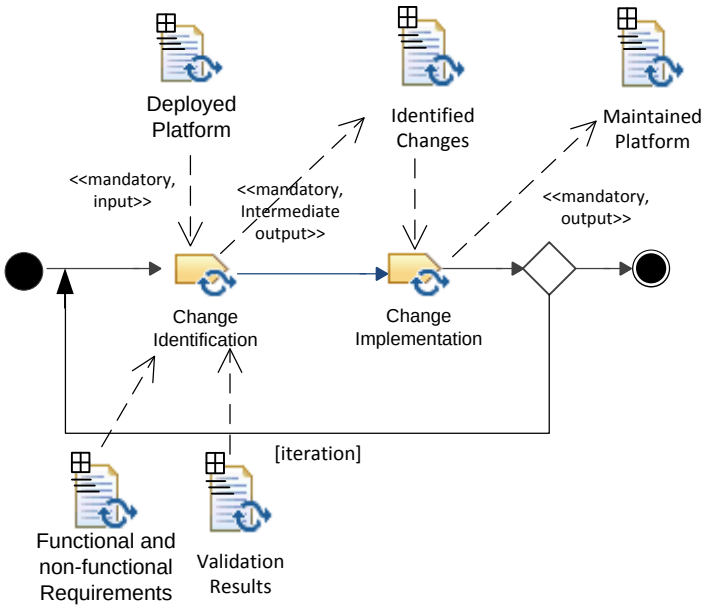

Figure 4.6.2: The workflow of tasks of the Integrated Platform Maintenance activity

4.6.2 Work Products

The final output work-product is the results of the maintenance activity, i.e. bug-fixing and/or evolution of the integrated platform.

5 INTER-METH based on INTER-IoT

In this Section, the INTER-METH instantiation process is shown. In particular, the Abstract phases of *Analysis*, *Design*, *Implementation*, *Deployment*, *Testing* and *Maintenance*, are customized with the aim of showing how the integration process between two heterogeneous IoT platforms/systems can be concretely carried out by exploiting the INTER-METH guidelines and INTER-IoT products, in particular, INTER-LAYER and INTER-FW. As in Section 4, for each phase of the instantiated process, an overall description is reported along with the list of performed activities and obtained work products.

5.1 Phase 1: Analysis

Description: Given two or more IoT platforms/systems to be integrated, the integration requirements need to be elicited. On the basis of the elicited requirements, the INTER-IoT-based design of the IoT platforms integration could be then carried out.

Objectives: To provide a model of the requirements for the integration of IoT platforms/systems.

Actors: The actors are:

(a) The developer of the integration (aka Integrator), who carries out the integration interconnection of heterogeneous IoT platforms; the Integrator is an active performer.

(b) The platform Owner, who will obtain the integrated platform;

(c) The involved Platforms to be integrated.

Expected results: INTER-GOM (Goal-Oriented Model), a final model that will represent the formal requirements models.

Main execution:

1. On the basis of the *Integration Goals*, each platform is analysed according to the INTER-IoT Reference Architecture (see D4.1/D4.2) and the requirements of integration among the layers of the platforms to be integrated are defined according to iterative tasks enclosed in activities.
2. According to the Step 1, the requirements of integration among the layers of the platforms to be integrated are defined according to iterative tasks enclosed in activities.

5.1.1 Activities

The analysis phase of the concrete instantiation consists of three main tasks that are performed by the Integrator (see Figure 5.1.3) according to the workflow depicted in Figure 5.1.4.

IoT Platforms Analysis: receives two heterogeneous IoT systems inputs: A and B. According to the INTER-IoT Reference Architecture, they are analyzed, thus producing the Analyzed Platforms Document. This step allows heterogeneous IoT platforms/systems with even notably different architectures to be compared by means of a common set of architectural solutions and building blocks. In particular, the INTER-IoT Reference Architecture is based on IoT-A (see Deliverable D4.2), which represents the most known and adopted IoT reference architecture. Aiming at the creation of an architectural reference model along with the definition of an initial set of key building blocks for enabling the emerging IoT, the IoT-A reference architecture provides views and perspectives on different architectural aspects that are of concern to stakeholders of the IoT. IoT-A functional architecture consists in the following Functional Groups: Device (technical artefact meant to provide an interface between the digital and the physical worlds), Communication (with regards to

the plethora of communication technologies that the IoT ARM needs to support, the need for a Communication FG is identified), Service Organization and IoT Process Management (requirements expressed by stakeholders regarding the possibility to build services and applications on top of the IoT are covered by the IoT Process Management and Service Organisation FGs), Virtual Entity IoT Service and Application (derived from the main abstractions identified in the Domain Model), Security (to address consistently the concern expressed about IoT Trust, Security and Privacy, the need for a Security transversal FG is identified) and Management (to address consistently the concern expressed about IoT Trust, Security and Privacy).

The INTER-IoT Reference Architecture based on IoT-A is portrayed in Fig. 5.1.1.

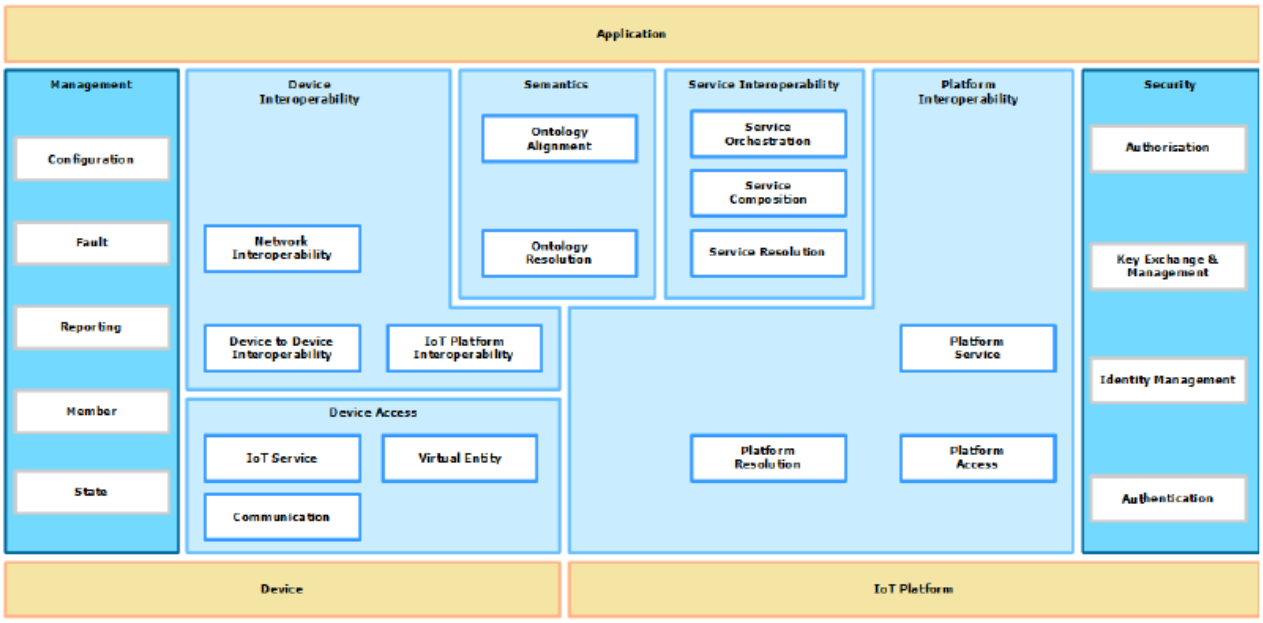

Figure 5.1.1: INTER-IoT Reference Architecture schema.

The INTER-IoT Reference Architecture contains 11 Functional Groups (FGs): the Application FG, the Device FG and the IoT Platform FG are out-of-scope of the INTER-IoT Reference Architecture, because they represent external groups that are going to be interoperated through the INTER-IoT and have been represented in yellow color. The five longitudinal FGs (Device Interoperability, Semantics, Service Interoperability, Platform Interoperability, Device Access) are represented in light blue color. The Management FG and the Security FG are transversal Functionality Groups and are shown in dark blue color. These transversal groups provide functionalities that are required by any of the longitudinal groups. In particular:

- *Service Interoperability FG*: supports the Application & Service to Application & Service (AS2AS) interoperability through the definition and execution of new compound services that make use of already existing services in the underlying IoT Platforms. Its goal is to use services from different IoT and create new services based on them.
- *Semantics FG*: addresses the challenges related to semantic interoperability of IoT Platforms. It provides support for the other FGs dealing with interoperability about IoT: The Service Interoperability FG, the Platform Interoperability FG and the Device Interoperability FG.
- *Platform Interoperability FG*: interacts with the different IoT Platforms to be interconnected. It is the responsible for accessing the IoT Platforms, not for implementing any of the features that the IoT Platforms provide.

- *Device Interoperability FG*: addresses the challenges of making legacy devices and non-real IoT Platform interoperable with other IoT Platforms and systems.
- *Device Access FG*: is responsible for offering a common interface to services and virtual entities that represent and expose functionality of physical devices. It abstracts all the necessary functions for managing the devices and interacting with them.
- Management FG: considers all the functionalities needed to rule the interoperability among different IoT Platforms. It is responsible for initializing, monitoring and modifying the operation of the interoperability among IoT Platforms.
- Security FG: is responsible for ensuring all the security aspects involved in the interoperability of IoT Platforms.

1. *Integration Layer Identification:* receives in input the Analyzed Platform Document and a set of Integration Goals. In output, produces the document Categories of Integration (CoI), a set of INTER-IoT integration layers and of the cross-layering.

The INTER-LAYER (INTER-IoT layered-oriented approach) architecture (see WP4) is reported in Fig. 5.1.2.

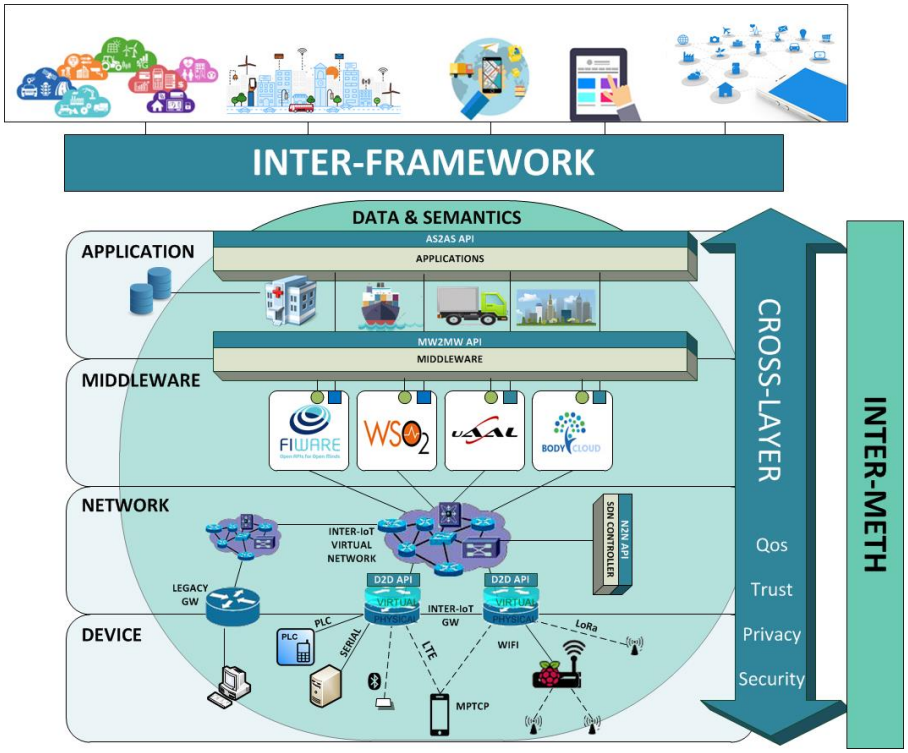

Figure 5.1.2: The INTER-LAYER approach schema.

As extensively reported, INTER-IoT presents a layer-oriented solution for interoperability, to provide interoperability at any layer and across layers among different IoT systems and platforms. This layer-oriented solution is achieved through INTER-LAYER and includes several interoperability solutions dedicated to specific layers. The different layers are:

- *Device*: allows the seamless inclusion of novel IoT devices and their interoperation with already existing ones.

- *Network(ing)*: aims to provide seamless support for smart objects mobility and information routing.
- *Middleware*: enables seamless resource discovery and management system for the IoT devices in heterogeneous IoT platforms.
- *Application & Services*: enables the use of heterogeneous services among different IoT platforms.
- *Semantics & Data*: allows a common interpretation of data and information among different IoT systems and heterogeneous data sources, achieving semantic interoperability.
- *CROSS-LAYER*: covers and guarantees non-functional aspects that must be present across all layers: trust, security, privacy, and quality of service (QoS).

2. *INTER-GOM Production*: receives the Analyzed Platform Document, a set of High-Level Integration Goals and the Categories of Integration, and produces, according to the GOM Meta Model, the INTER-GOM (Goal-Oriented Model) as output. To obtain the final model that will represent the formal requirements model and will drive the INTER-IoT-based Design Phase, the INTER-GOM Production Task can be iterated one or more time.

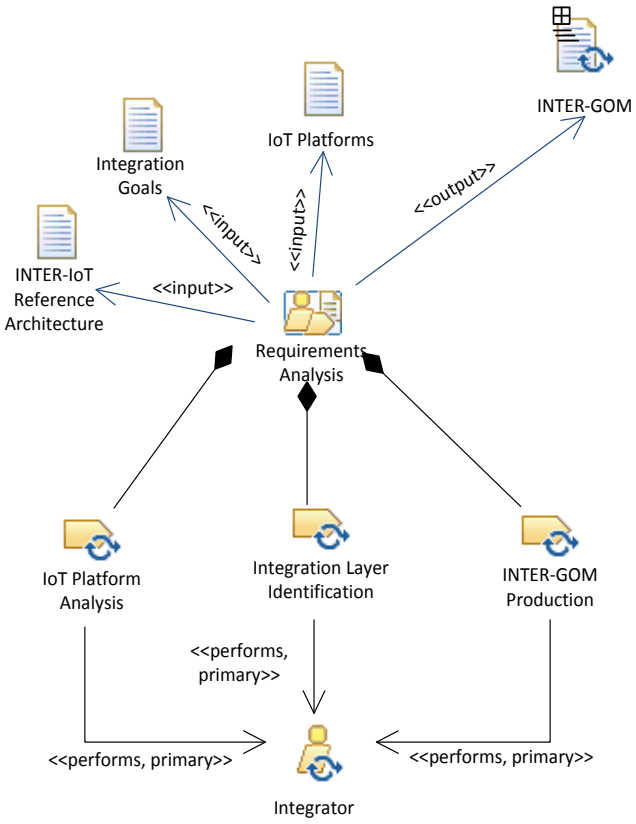

Figure 5.1.3: The INTER-IoT-based Analysis phase described in terms of activities, roles, and work products

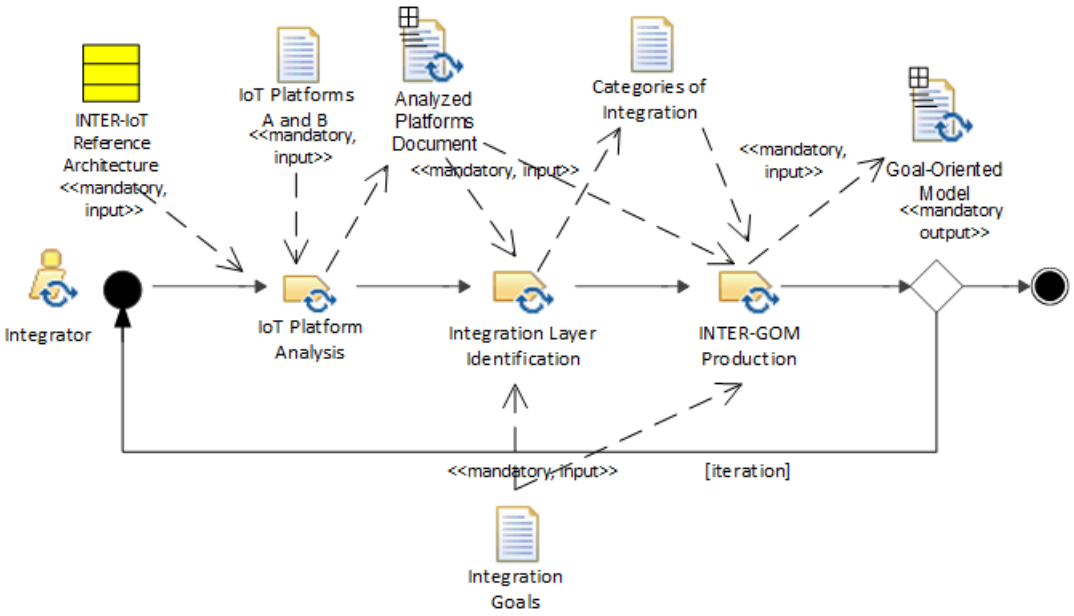

Figure 5.1.4: The workflow of tasks of the INTER-IoT-based Requirement Analysis activity

5.1.2 Work Products

The final output work-product is the INTER-GOM, a final model (compliant to the Metamodel reported in Figure 5.1.5) that will represent the formal requirements model and will drive the INTER-IoT-based Design Phase.

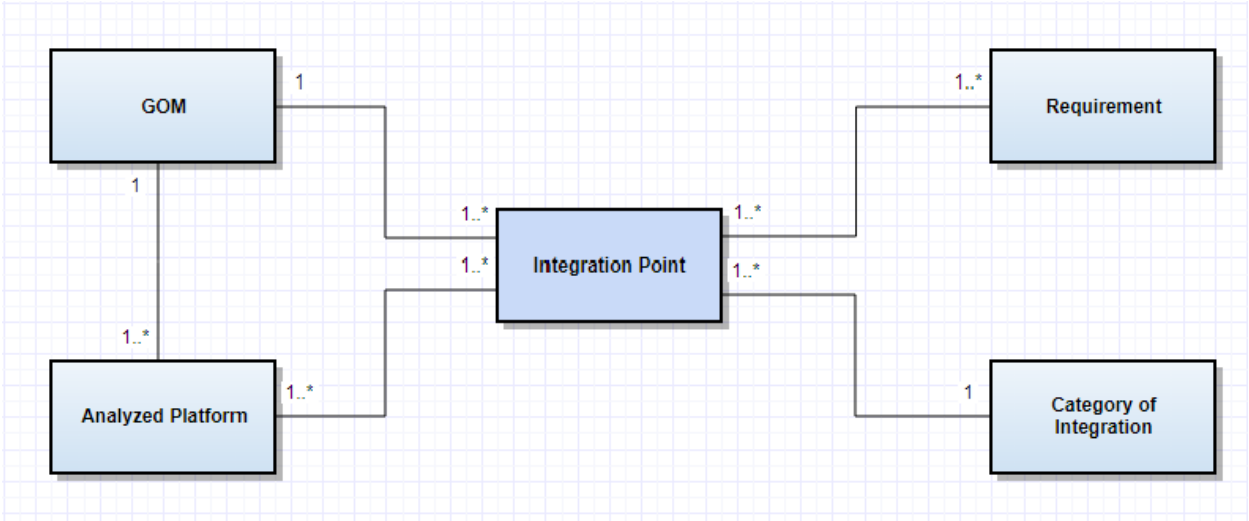

Figure 5.1.5: The Metamodel of INTER-GOM

The Metamodel of INTER-GOM is reported in Fig. 5.1.5 and is composed of four main elements. In particular:

- *Analyzed Platform Document*: comparing the platforms using a homogeneous and shared architectural representation (see above), allows to identify the integration points between the platforms.
- *Integration Point*: Tropos defines the concept of Refinement. A refinement of an Element (e.g., a Goal) is a conjunction of the sub-elements that are necessary to achieve it. We will call these points of refinement, Integration Points (see Fig. 5.1.6). They put together parts of the platforms according to the layer-based architectural model. It is possible to

define a general and a specific level of refinement, according to INTER-IoT Reference Architecture: the first level identifies the *Functional Groups*, the second level the *Functional Components* (see Figure 5.1.1).
For each Integration Point, there are one or more:

- *Requirement*: represents desired states we want the Integration Point to achieve. They are progressively refined into intermediate goals, until the process produces actionable goals (tasks) that need no further decomposition and can be executed. One or more requirements can be associated to a single Integration Point.
- *Category of Integration (CoI)*: identifies the level of interoperability in a set of INTER-IoT integration layers and of the cross-layering.

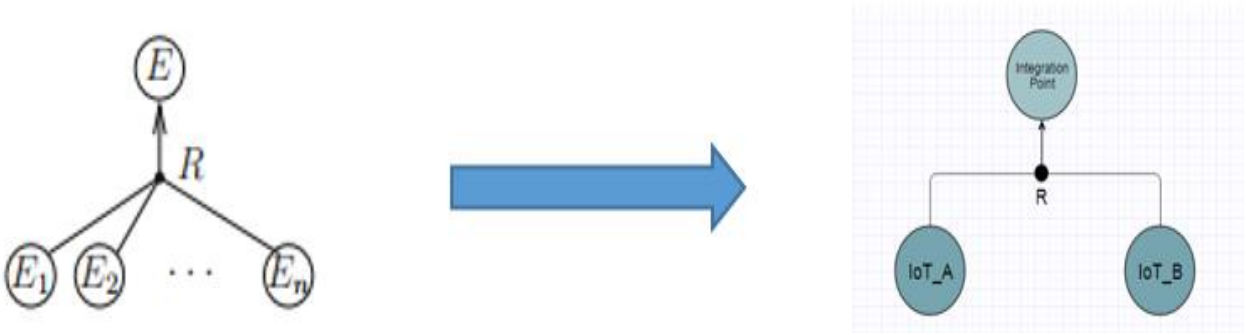

Figure 5.1.6: Refinement Process schemes in TROPOS and in INTER-METH

According to the *Analyzed Platform Documents* and a set of *Integration Points*, the GOM is defined, following an iterative procedure as shown in the activity diagram in Fig. 5.1.7. The execution of the activity diagram is as follows:

1. The first activity is the *Integration Point Identification*. According to the transition conditions, the activity flow loops according to the need of choosing other integration points, otherwise it moves to the next activity: *Integration Point (IP) Requirement Identification*;
2. The second activity identifies one or more IP requirements before proceeding to the final GOM generation;
3. Finally, if no further IP requirement has to be identified, the GOM document is generated and the process ends.

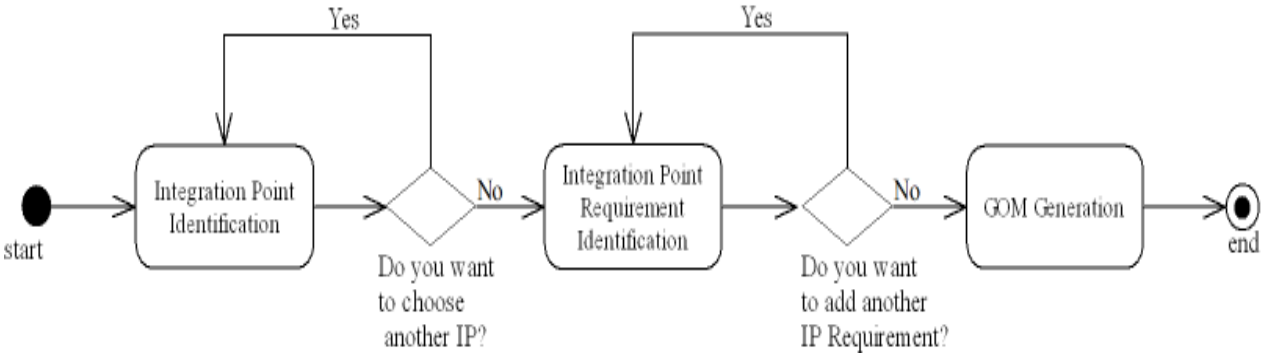

Figure 5.1.7: The UML Activity Diagram of INTER-GOM Production

5.2 Phase 2: Design

Description: Given two or more IoT platforms/systems whose formal integration requirements are reported in INTER-GOM model (generated as output of INTER-IoT-based Analysis Phase), a set of

INTER-IoT DESIGN PATTERNS have to be produced. On the basis of the INTER-IoT DESIGN PATTERNS (documented in D5.1), the implementation of the IoT platforms integration could then be carried out by INSTANTIATION.

Objectives: To define the INTER-IoT DESIGN PATTERNS for the integration of IoT platforms/systems

Actors: The actors are:

(a) the developer of the integration (aka Integrator), who carries out the integration interconnection of heterogeneous IoT platforms; the Integrator is an active performer.

(b) the platform Owner, who will obtain the integrated platform;

(c) the involved Platforms to be integrated.

Expected results: Set of INSTANTIATED INTER-IoT DESIGN PATTERNS for the integration of the identified IoT platforms/systems.

Main execution:

1. For each layer, on the basis of the elicited requirements reported in the INTER-GOM, an initial INTER-IoT DESIGN PATTERN is produced.
2. Each INTER-IoT DESIGN PATTERN produced in Step 1 is iteratively refined.
3. A global integration design (including the cross-layer) is defined on the basis of the outcome of Step 2 (and iteratively refined, if needed).

5.2.1 Activities

The Integration Design activity, which is the only main activity of the INTER-IoT-based Design phase, is subdivided into two main tasks that are performed by the Integrator (see Figure 5.2.1 and Table 5.2.1) according to the workflow depicted in Figure 5.2.2:

1. *INTER-IoT Layer Integration Design Specification*: on the basis of the INTER-GOM and Analyzed Platforms Document, for each INTER-IoT Layer (Device, Network, Middleware, Application, Service, Data & Semantics) a layer integration specification is iteratively defined by exploiting the INTER-LAYER product (D2D, N2N, MW2MW, AS2AS, DS2DS). Such task will produce six INSTANTIATED INTER-IoT Design Patterns, one for each INTER-IoT layer.
2. *INTER-IoT Full Integration Design Specification*: On the basis of the six INSTANTIATED INTER-IoT Design Patterns from the previous task, a full-fledged specification is iteratively produced by incorporating the CROSS-LAYER and INTER-FW INTER-IoT Design Patterns.

In particular, INTER-IoT PATTERNS (or INTER-PATTERNS) are design patterns directly corresponding to the integration solutions already achieved in the WP3 (particularly, according to INTER-LAYER) and WP4 (particularly, according to the INTER-FW for contextualize solutions in different application domains, e.g. m-Health, Transportation and Logistics) and they aim at furnishing well-formalized domain-specific guidelines. Note that WP5 depends on WP3 and WP4 outcomes, while WP3 and WP4 are independent from WP5. The defined set of INTER-PATTERNS comprises the element listed in Table 5.2.1, where is reported also their related INTER-IoT layer and inspiring Pattern Catalog (if any).

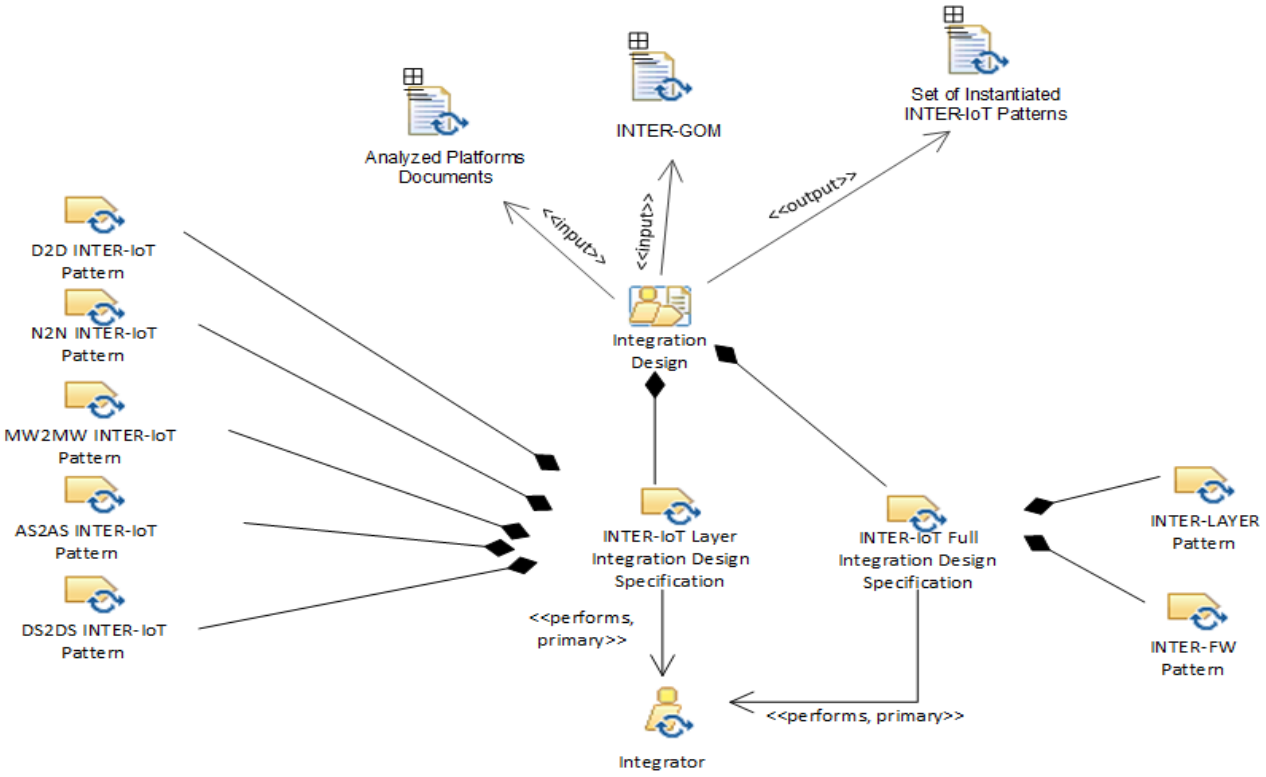

Figure 5.2.1: The INTER-IoT-based Design phase described in terms of activities, roles, and work products

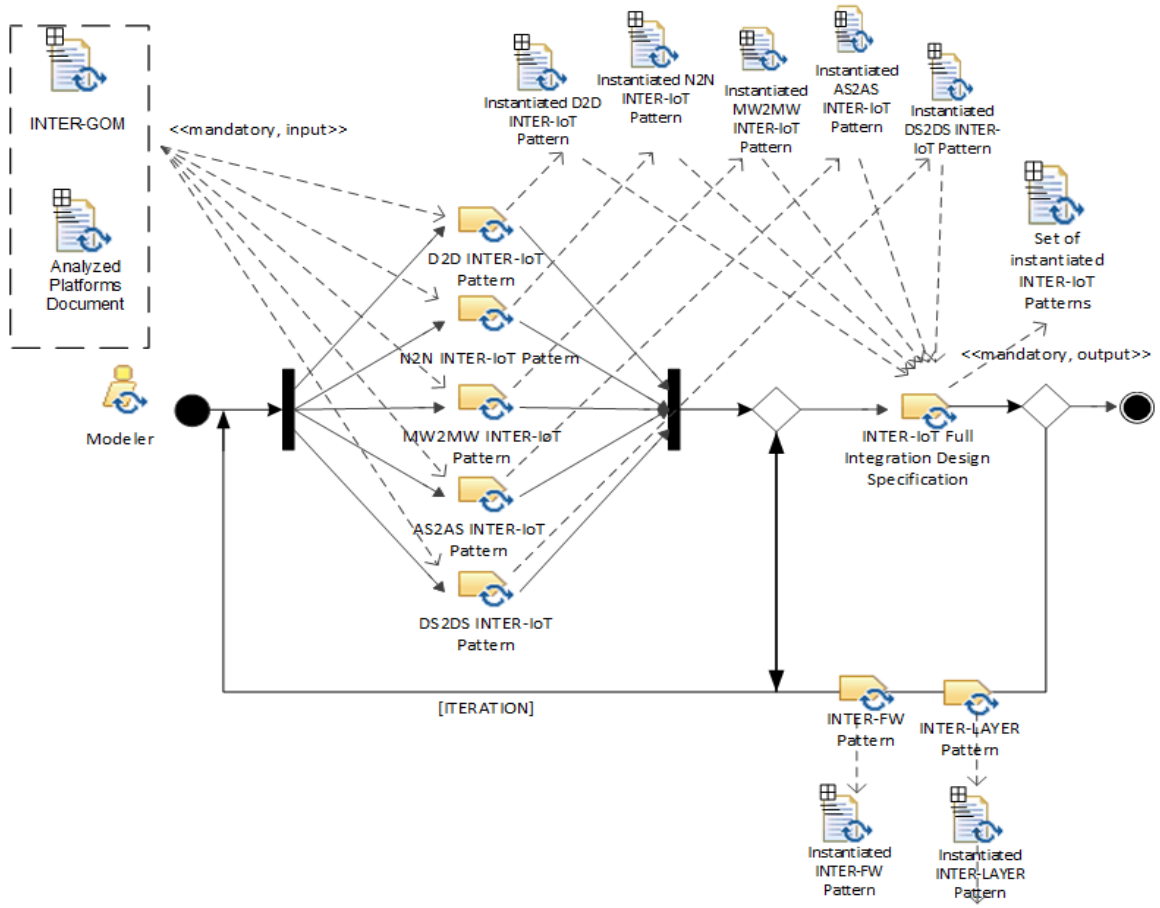

Figure 5.2.2: The workflow of tasks of the INTER-IoT-based Design Integration activity

Table 5.2.1: INTER-IoT Design Patterns, their related INTER-IoT layer and inspiring Pattern Catalogue

Layer	Name of INTER-IoT PATTERN	Inspiring PATTERN CATALOG
D2D	INTER-IoT Gateway Event Subscription.	IoT Patterns, Enterprise Integration Patterns, Agent Design Patterns.
D2D	D2D REST Request/Response.	Reactive Patterns, IoT Patterns, Integration Patterns.
N2N	INTER-IoT Orchestration of SDN Network Elements.	
MW2MW	INTER-MW Simple Component Pattern.	Reactive Patterns.
MW2MW	INTER-MW Message Broker.	Integration Patterns, IoT Patterns.
MW2MW	INTER-MW Self-contained Message.	Reactive Patterns, SOA Patterns.
MW2MW	INTER-MW Message Translator.	Integration Patterns.
AS2AS	AS2AS Flow Based Service Composition.	
AS2AS	INTER-AS2AS Service Orchestration.	Service Orchestration.
AS2AS	AS2AS Discovery of IoT Services.	IoT Patterns, SOA Patterns.
DS2DS	Alignment-based Translation Pattern.	Integration Patterns, Agent Design Patterns.
DS2DS	Translation with central ontology.	Integration Patterns, Agent Design Patterns.
CROSS	INTER-IoT SSL CROSS-Layer secure access	Security Patterns, IoT Patterns.
INTER-FW	Configuration delegation pattern.	Object-oriented Patterns.
INTER-FW	API façade.	
INTER-HEALTH	INTER-Health Pilot Integration.	Enterprise Integration Patterns.
INTER-HEALTH	Integrated deployment in security-constrained environments.	Security Patterns.
INTER-LogP	Geofencing pattern.	

As it could be noted, three patterns catalogs have inspired the INTER-PATTERNS: IoT Patterns (which is obvious choice, because it describes the IoT solutions), Reactive Patterns (mainly because of its responsive and simple solutions) and Integration Patterns Catalog (mainly because of its effective and common message solutions). Regardless to their functionality, the template-based approach of describing all the INTER-IoT PATTERNS comprises twelve properties (as extensively reported in D5.1) *Pattern name, Identifier, Inspired by, Related patterns, Intent, Problem & Solution, Applicability, UML representation, Implementation, Known uses, Identified by, Registration date*.

5.2.2 Work Products

The work product of this phase is a set of INSTANTIATED INTER-IoT Design Patterns to be exploited for driving the implementation phase. Each INSTANTIATED INTER-IoT Design Pattern is described in a xml file according to the template previously reported, containing, among others, the *UML representation* attribute that contains an URL to graphical resources (formal diagrams, e.g., UML diagram, generated through an external tool).

5.3 Phase 3: Implementation

Description: Given two or more IoT platforms/systems whose integration has been designed according to the INSTANTIATED INTER-IoT Design Patterns, the Inter-IoT based integration implementation has to be performed.

Objectives: To concretely integrate/interconnect the considered IoT platforms/systems.

Actors: The actors are:

(a) the developer of the integration (aka Integrator), who carries out the integration interconnection of heterogeneous IoT platforms; the Integrator is an active performer.

(b) the platform Owner, who will obtain the integrated platform;

(c) the involved Platforms to be integrated.

Expected results: The Inter-IoT based integrated IoT platform/system.

Main execution:

1. The INSTANTIATED INTER-IoT Design Patterns produced in the INTER-IoT based Design phase are actually implemented according to successive refinement steps.

5.3.1 Activities

The INTER-IoT based Integration Implementation activity, which is the only main activity of the Implementation phase, includes one main task that is iteratively performed by the Integrator (see Figure 5.3.1) according to the workflow depicted in Figure 5.3.2:

1. *INTER-IoT based Systems Integration*: on the basis of the INSTANTIATED INTER-IoT Design Patterns, the heterogeneous IoT platforms/systems are integrated by the Integrator in an iterative activity aimed at obtaining the final integrated platform. At this step, we can: (i) *configure* the components of the Integrated Platform by means of the INTER-FW; (ii) *extend* the components of the Integrated Platform (e.g., a new functionality enabled by the integration needs to be implemented); and (iii) to *implement*, in terms of software bridges connected to INTER-LAYER, the INTER-IoT based INTER-LAYER design patterns. As results of each these three sub-activities, the Integration Design Specifications are updated.

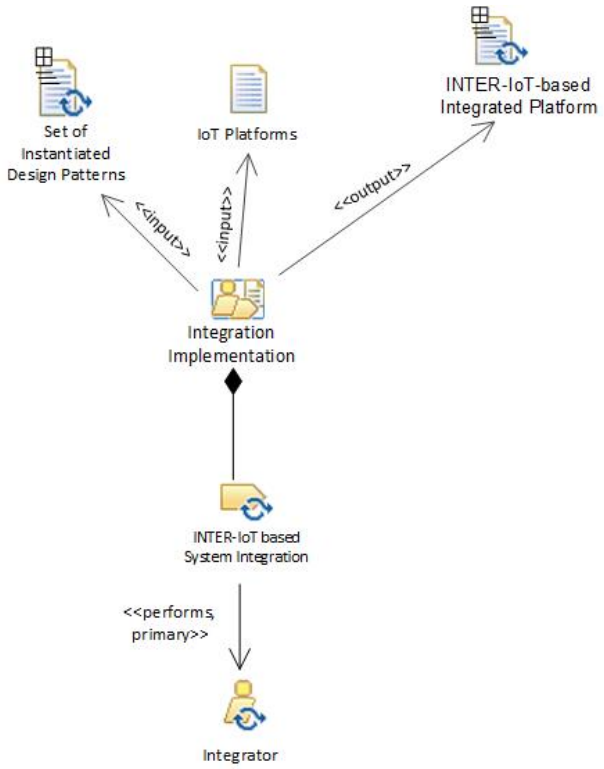

Figure 5.3.1: The INTER-IoT-based Implementation phase described in terms of activities, roles, and work products

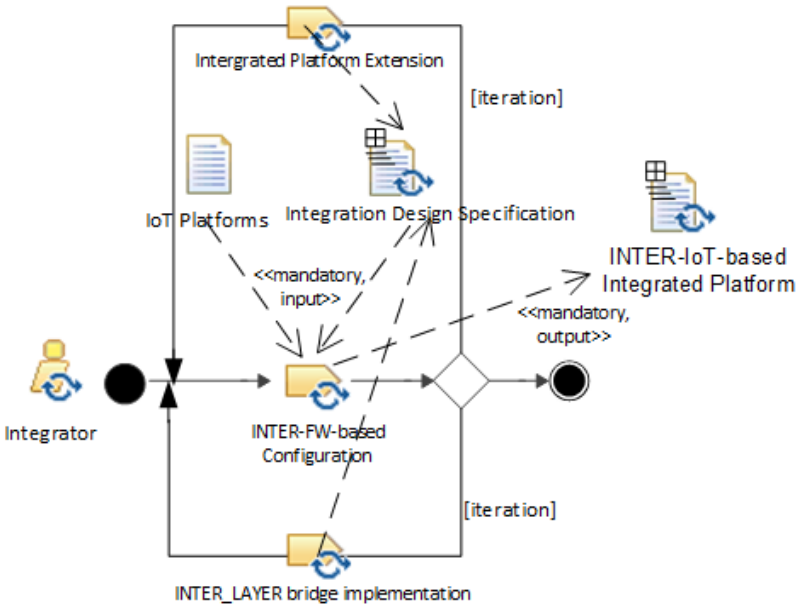

Figure 5.3.2: The workflow of tasks of the INTER-IoT-based Integration Implementation activity

5.3.2 Work Products

The final output work-product is the *INTER-IoT-based Integrated Platform*. If needed, an ontology alignment may be performed at this step through the IPSM tool, which will be presented in detail in D5.3 (INTER-CASE Tool). Such process consists in finding of correspondences between two or more ontologies. The result of this process is an alignment of a set of correspondences between entities (atomic alignment) or groups of entities and sub-structures (complex alignment) from different ontologies. A correspondence can be either a predicate about similarity, called a matching, or a logical axiom mapping. In Appendix B, we provide the methodological base of INTER-IoT ontology alignments.

At this point, the heterogeneous IoT platforms/systems are integrated according to the INTER-IoT approach, thus obtaining interoperability among them and allow implementation and deployment of IoT applications on top of them.

Figure 5.3.3 reports the relationships that are established among the different phases and work products.

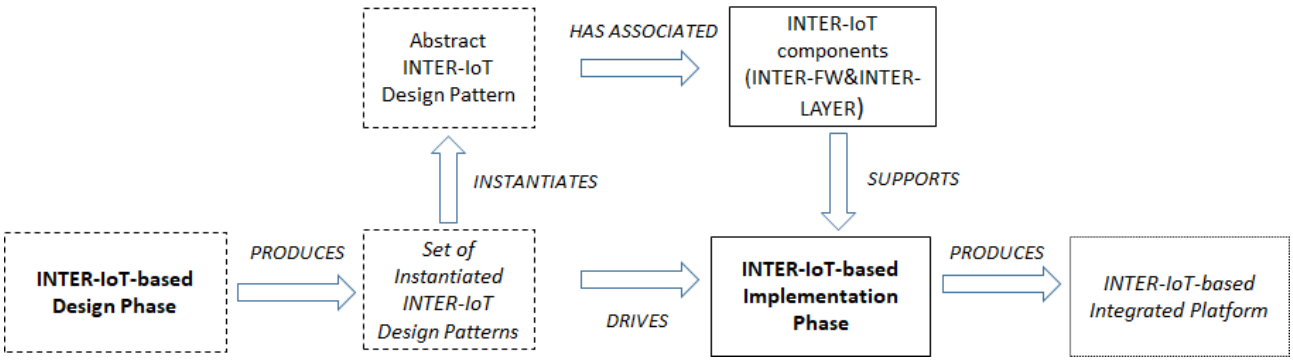

Figure 5.3.3: Relationships among the different phases (in bold) and work products (in italic)

5.4 Phase 4: Deployment

Description: Given two or more IoT platforms/systems, whose integration has been implemented according to the Implementation phase, the deployment of the integrated platform has to be performed.

Objectives: To deploy the integrated IoT platforms/systems.

Actors: The actors are:

(a) the developer of the integration (aka Integrator), who carries out the integration interconnection of heterogeneous IoT platforms; the Integrator is an active performer.

(b) the platform Owner, who will obtain the integrated platform; the Owner is an active performer.

(c) the involved Platforms to be integrated.

Expected results: The deployed IoT platform/system.

Main execution:

1. The Integrated Platform is deployed according to deployment goals and requirements. In particular, the INTER-FW web framework is the entry point to the INTER-IoT Configuration and Management Framework (CMF), one of the three main software results of WP4, together with INTER-API and the Software Development Framework. The use of INTER-FW CMF (from now on, INTER-FW for shortness) is baseline for the IoT Platforms deployment phase in the scope of INTER-METH. Thus, the INTER-IoT-based Deployment Phase of INTER-METH is strongly dependent on INTER-FW. The first version of the INTER-FW has eight sections (or modules), from which six of them are directly involved in the deployment of interoperable IoT Platforms. INTER-IoT treats the five defined interoperability layers in separate concerns, so five of these modules correspond 1-to-1 to these concerns. The sixth module is related to AAA (Accountability, Authorization and Authentication).

5.4.1 Activities

The INTER-IoT based Deployment activity, which is the only main activity of the Deployment phase, comprises six main tasks that are performed by the Integrator/Configurer (see Figure 5.4.1) according to the workflow depicted in Figure 5.4.2 (in particular, the activities are strongly connected to the UI functionalities of the different modules based on INTER-FW and so will be described):

- *Platform Configuration*: this task (connected to the *Platforms* module) deals with the deployment of complete IoT Platforms for interoperating them towards rich applications. Although the technical focus of this module is the deployment of interoperable middlewares of platforms, the whole content (i.e., the platform) has been assumed to the concept of 'middleware' since the IoT platforms are univocally bound to the concept of platform (there are few or none platforms without a middleware, while there are middlewares not linked with specific platforms, though). To instantiate and deploy an IoT Platform Middleware in the INTER-FW to enable middleware interoperability, the following steps are followed:

  0. A bridge of the platform to interoperate must be available. INTER-IoT provides a series of reference bridges. If the platform is not in the list of reference implementations, this must be done following the "*developing new bridges*" instructions that will be publicly available by the end of the project in the project site in GitHub. These instructions will extensively use the Software Development Framework of the project.

  1. Deploy the platform following the deployment instructions of the platform vendor/developer. The platform must be reachable in a public endpoint (via DNS or IP).

2. Access the *Platforms* module (through its GUI)

3. Add a new *Platform*

4. Provide the following information (through filling in a given form), according to the following fields:

4.1. Name: the name of the *Platform* (arbitrary)

4.2. Type: the type of *Platform* selected from the reference implementations in INTER-IoT

4.3. Bridge: the bridge applicable to the platform. One platform can be interfaced through different bridges (e.g. different versions or different natively unavailable method implementations).

4.4. Security: the security approach used to connect with the *Platform*, chosen from a list.

4.5. Owner: the user that allows the property of the data and the responsibility of them. It must be a user already registered in INTER-FW.

4.6. URL and port: the endpoint to reach the public instance of the *Platform*. The GUI allows checking whether or not the introduced endpoint is reachable from the location where INTER-FW is deployed.

5. Save the all the information (i.e. the form). If all the fields are filled properly, the *Platform* should now be available in the INTER-FW environment. The platform will appear in a list in the main section of the *Platforms* module.

To explore the list of devices registered in a platform, its entry can be properly selected in the list of platforms. This view contains two tabs: a) the first entry “Platform” shows the details of the platform; b) the second entry “Entities” has a list of devices connected to the platform. New devices can be registered (e.g. by using the “Add” button). Entities can be explored or deleted from the list.

- *Gateway Configuration*: this task (connected to the *Gateways* module) focuses on the device to device interoperability, addressed in the scope of INTER-IoT in the D2D Layer through the “Gateway Event Subscription” and “REST Request/Response” patterns (as described in D5.1). The steps to add a new gateway are:

1. A gateway with the gateway software of INTER-IoT must be available. The hardware must be compatible with the INTER-IoT Gateway software. The compatibility list will be published in the INTER-IoT Gateway development site in GitHub.

2. Select the *Gateways* module in the INTER-FW

3. Add a new Gateway

4. Providing the following information (by filling in the GUI form), containing the following fields:

4.1. Name: the given name of the physical *Gateway*

4.2. URL and port: the endpoint to reach the public instance of the *Gateway*. The form allows checking if the introduced endpoint is reachable from the location where INTER-FW is deployed.

4.3. Owner: the user that allows the property of the data and the responsibility of them. It must be a user already registered in INTER-FW.

5. Save the provided information. Once the *Gateway* is created, if the data are correct and the gateway is reachable, the gateway will appear in the list of physical gateways. A virtual instance of the gateway is automatically created following the pattern “Digital Twin”.

To explore the list of devices registered in a gateway, select the list of registered gateways in its entry. This view contains three entries: the first “Gateway” shows the details of the gateway; the second “Physical” has a list of devices connected to the platform; the third, “Api” links to the API of the layer interoperability implementation. Entities can be explored or deleted from the list.

- *Networking Configuration*: in this task (connected to the Networking module), the network interoperability, achieved via network virtualization and “Virtual Network Orchestration” pattern is configured and managed, supporting the needs of the N2N layer of INTER-IoT in terms of implementation and deployment. The module of Network is divided in three subsections (or views):

  1. Topology: an informative view aimed to show the emulated network topology.

  2. Statistics: It has two functions: one to manage (read, create, modify, remove) network ports and another to manage network flows.

  3. QoS: where the statistics of the different network devices and segments are displayed. This view offers the possibility to set up new queues and new network rules and related metrics. All these elements can be managed from their respective lists, allowing editing or removing.

- *Application Services Configuration*: this task (connected to the *Application Services* module) includes a graphic tool for service orchestration, the underlying interoperability mechanism for AS2AS Layer (see D3.2). This is one of the less intrusive views in the INTER-FW, since the implementation tool, the open source project “node-red” has a powerful user interface which allows this service orchestration from a visual perspective. The initial view in INTER-FW of the Services section shows a list of already existing workflows.

  1. To create a new workflow, it must be clicked the “Add” button, while existing ones can be modified by clicking their entry in the list or clicking the “Remove” button to delete them.

  2. Adding a new node creates a new instance of the “node-red2 tool enhanced by INTER-IoT.

  3. To set up a new workflow in the UI of the service orchestrator, components must be dragged and dropped to the canvas and connected via virtual wires.

  4. Double clicking each component gives access to the configuration of the services.

  5. To deploy new orchestrated services, the *deploy* button is to be clicked. If no error has been raised, the new workflow will start working immediately.

- *Semantics Configuration*: in this task (connected to the *Semantics* module), all the configurable parameters and processes related to the semantics interoperability (all mechanism present in the DS2DS layer, see D3.2) are configured for the deployment of interoperable IoT platforms. The view contains five sections.

  1. To add a new mediator instance, the IPSM (Inter Platform Semantic Mediator) Configuration tab needs to be selected and then a “Register IPSM instance” created, the URL and port of the instance where the IPSM instance to be deployed will be requested.

2. To create new IPSM alignments, a new alignment instance has to be created through “Add Alignment”. A form asking for the possible combinations will be used. After filling the information, the new alignment will be created. The created alignments can be administered (modify, remove) from the same tab. The catalogue of available alignments is populated from the IPSM authoring tool, currently not available in the INTER-FW, which will be created in the context of the INTER-CASE tool (see T5.3).

3. To set up translation channels (to improve performance), “Add channel” functionality is used. Channels are managed from the same view.

4. To explore the available ontologies, “Ontology repository” tab can be selected. Ontologies can be added by using “Add Ontology”.

- *User management Configuration*: this task (connected to the *User management* module) contains configurations valid for all the previous modules (or sections); in particular, it configures and manages the users of the INTER-FW and the authorized access of them to the IoT resources connected in INTER-IoT. To add a new user, the following steps must be followed:

  0. The current user must have administration permissions.

  1. Select the “Users management” view. In this view, all the users of INTER-FW are listed. From this view they can be managed.

  2. Add a new user.

  3. Complete the form that contains the following fields:

     3.1. Username: the INTER-FW login name for the application.

     3.2. Password: the secret key for the user.

     3.3. Name: the complete name of the user (optional)

     3.4. Email: a valid email for the user. Communications will be issued to this address, so it must be valid. A verification mechanism (validator + two-step sign-up) checks that the user has introduced a valid email.

     3.5. Company: The company of the user.

     3.6. INTER-FW App role: The initial role. The role is applicable for the views that are initially rendered for the users.

  4. Once the user is created, authorization to IoT artifacts connected through INTER-IoT can be granted/revoked by using the specific *User* entry and accessing the “Permissions” tab.

  5. Specific policies for the user can be also assigned by using the *Policies* tab in the same view.

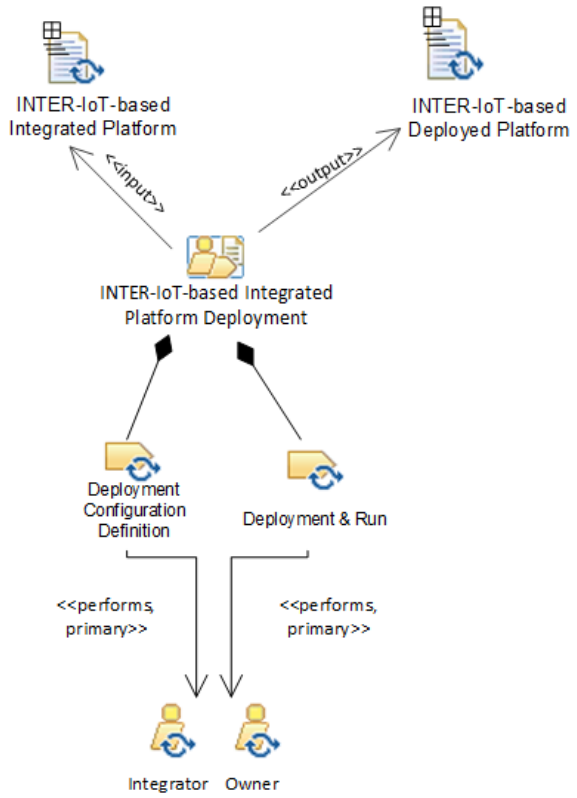

Figure 5.4.1: The INTER-IoT-based Deployment phase described in terms of activities, roles, and work products

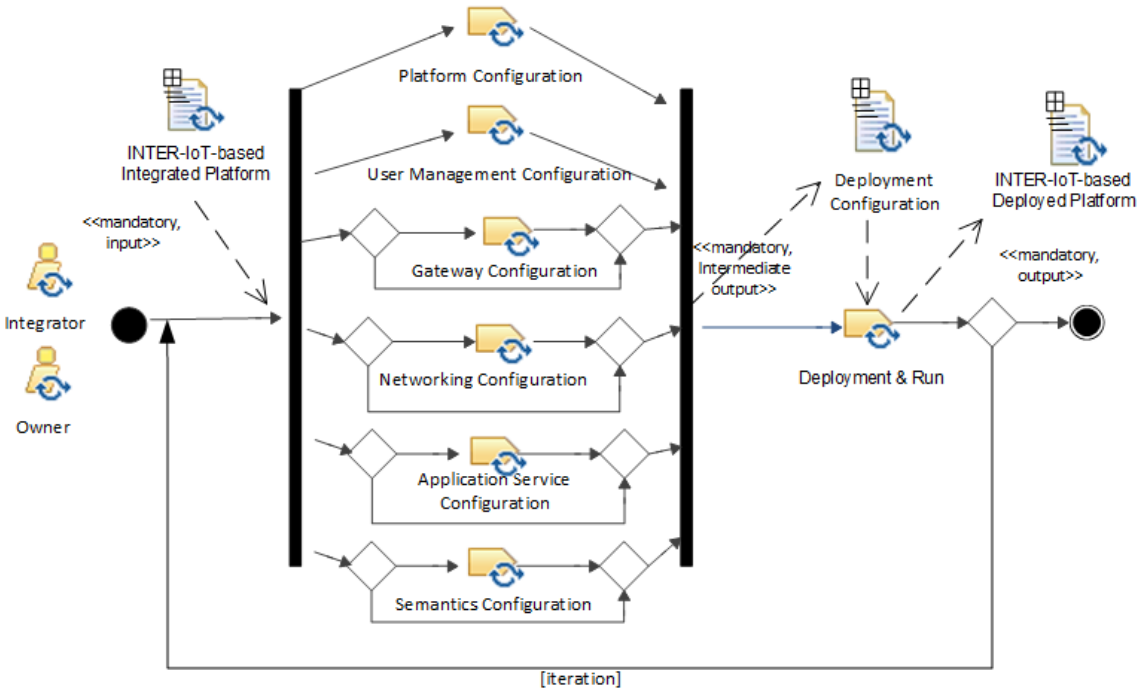

Figure 5.4.2: The workflow of tasks of the INTER-IoT-based Deployment activity

5.4.2 Work Products

The work product refers to the configured and deployment integrated IoT platform.

5.5 Phase 5: Testing

Description: Given two or more IoT platforms/systems, whose deployment has been carried out according to Deployment phase, the testing/validation has to be performed.

Objectives: To test the deployed integrated IoT platforms/systems.

Actors: The actors are:

(a) the developer of the integration (aka Integrator), who carries out the integration interconnection of heterogeneous IoT platforms; the Integrator is an active performer.

(b) the platform Owner, who will obtain the integrated platform; the Owner is an active performer.

(c) the involved Platforms to be integrated.

Expected results: The test results on the integrated and deployed IoT platform/system.

Main execution:

1. The integrated and deployed platform is executed and validated through testing according to well-defined test cases. Acceptance testing is a test conducted to determine if the requirements of a specification are met [TE1]. In systems engineering it may involve black-box testing performed on a system, such as for example for software modules. International Software testing Qualifications Board (ISTQB), which is a software testing qualification certification organisation, defines acceptance as formal testing with respect to user needs, requirements, and business processes conducted to determine whether a system satisfies the acceptance criteria [TE2]. There can be many types of acceptance testing for a system, service or product. The acceptance test can be performed multiple times in the case of defect resolving or when test cases are not executed within a single test iteration. In INTER-IoT acceptance testing is performed in two phases: Factory Acceptance Testing (FAT) and Site Acceptance testing (SAT) [TE3]. Factory Acceptance Test (FAT) and Site Acceptance Test (SAT) are performed to test and evaluate the INTER-IoT-based integrated system implemented in the Implementation Phase (see section 5.3) and deployed in the Deployment Phase (see section 5.4). During development, units and components are tested by the developers. After the integration of the tested and validated components in the system, the FAT will be executed in order to indicate if there are needs for improvements. After the successful execution, the system can be tested in the field and undergo the SAT.

5.5.1 Activities

The Testing activity, which is the only main activity of the INTER-IoT-based Testing phase, is subdivided into four main tasks that are performed by the Integrator/Tester (see Figure 5.5.1 and Table 5.5.1) according to the workflow depicted in Figure 5.5.2:

- *Factory Acceptance Testing (FAT) Task.* The FAT is performed to test and prove the system in a lab environment and tests if solution meets the specifications and if it is functional before it is deployed in the field. FAT tests can be performed by simulation or a functional test. FAT describes the system, test setup, tooling, test strategy, test activities and test results for the lab setup. During FAT testing the readiness of the system is shown to the customer. The readiness of the system is shown to the costumer, which can use the system in actual system setup for the first time. During the testing process, the operators and maintenance personal can get insight into the operated and maintained system. Minor irregularities can be detected which lead to minor changes in the INTER-IoT system before the system is deployed in the field.
- *Site Acceptance Testing (SAT) Task.* SAT takes place after integration at the customer site and tests if the solution has been correctly integrated into the customer's environment and

meets all the requirements. The solutions are tested on: Integration, Performance, User acceptance, conformance to specifications. During the SAT testing process the actual deployed system is tested and proven. SAT process are similar like FAT process, except is describes and tests the system integrated in the customer systems. For each of the use-case/pilots a FAT and SAT testing will be performed.

- *Test Environment Task*. The test environment is usually designed to be identical, or as close as possible, to the anticipated production environment. In order to test the IoT system/framework in “real world” as much as possible, representative test systems are needed where pilot test setups must be created and verified, which includes all facilities, hardware, software, firmware, procedures and/or documentation intended.
- *Defect Reporting Task*. For the defect reporting purpose, each integration project can be planned, tracked and reported in organized manner. During the integration testing new issues can be found, for which it has to be determined whether they are actually a defect or not. The defect has to be resolved by either developer, project manager or software architect. After the defect has been resolved, the solution will be integrated into the next release and tested during the next integration test.

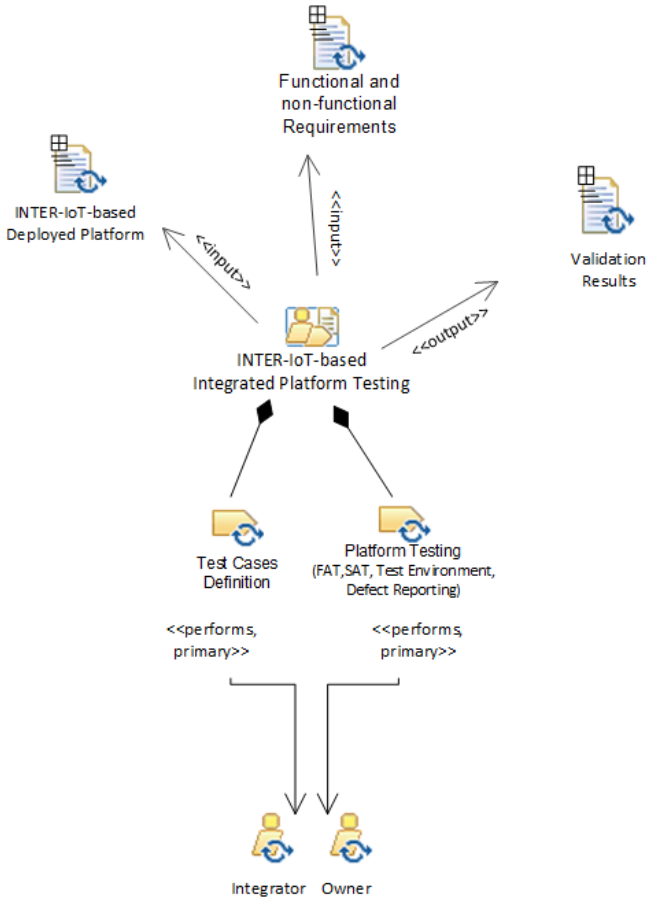

Figure 5.5.1: The INTER-IoT-based Testing phase described in terms of activities, roles, and work products

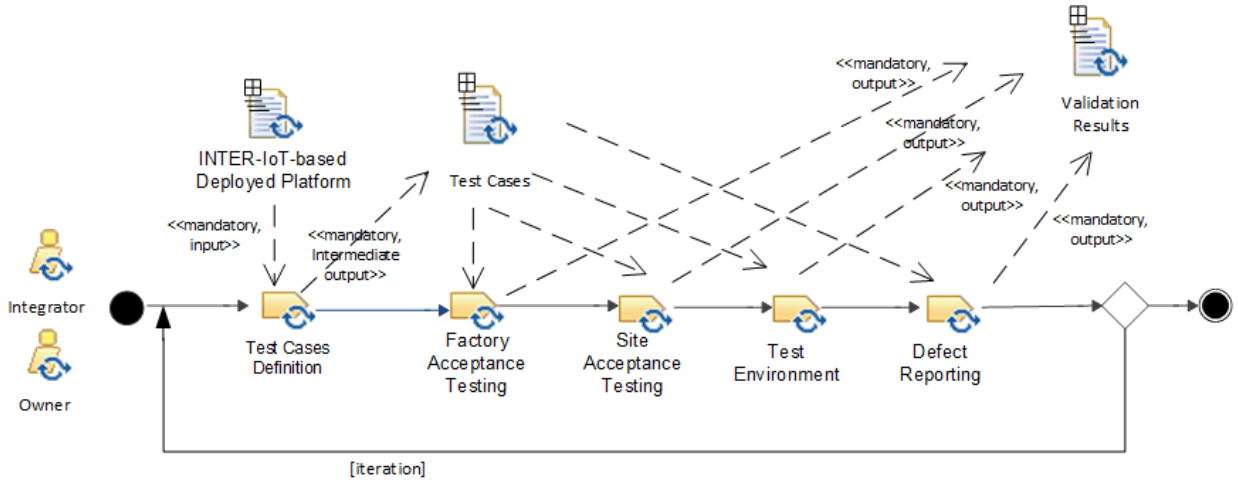

Figure 5.5.2: The workflow of tasks of the INTER-IoT-based Testing activity

5.5.2 Work Products

The FAT and SAT indices documents will constitute the output work products of such phase.

5.6 Phase 6: Maintenance

Description: Given an INTER-IoT-based integrated platform obtained from the integration of two or more IoT platforms/systems according to the previous phases, such platform needs to be maintained.

Objectives: To maintain and make evolve an integrated IoT platform.

Actors: The actors are:

(a) the developer of the integration (aka Integrator), who carries out the integration interconnection of heterogeneous IoT platforms; the Integrator is an active performer.

(b) the platform Owner, who will obtain the integrated platform; the Owner is an active performer.

(c) the involved Platforms to be integrated;

(d) the integrated Platform

Expected results: Continuous maintenance of the integrated IoT platform.

Main execution:

1. *Identification of a list of bugs* and/or *a list of evolution points at specific INTER-IoT layers and/or products*.
2. *Correction of bugs* and/or *implementation of new functionalities* (go back to the Analysis or Design phases).

5.6.1 Activities

The Integrated Platform Maintenance activity, which is the only main activity of the Maintenance phase, is subdivided into two main tasks that are performed by the Integrator and/or Owner (see Figure 5.6.1) according to the workflow depicted in Figure 5.6.2:

1. *Change Identification*: this task aims at identifying bugs and/or evolution points of the integrated platform. Identification will be done at each layer (Device, Networking, Middleware, App Services, Data & Semantics) and cross-layer of the integrated platform (i.e. at INTER-LAYER) and at framework level (i.e. at INTER-FW).
2. Change *Implementation*: actual implementation of the changes, i.e. bug fixing or analysis, design, implementation, deployment, and validation of new functionalities; the latter could imply to re-execute, totally or partially, the integration process.

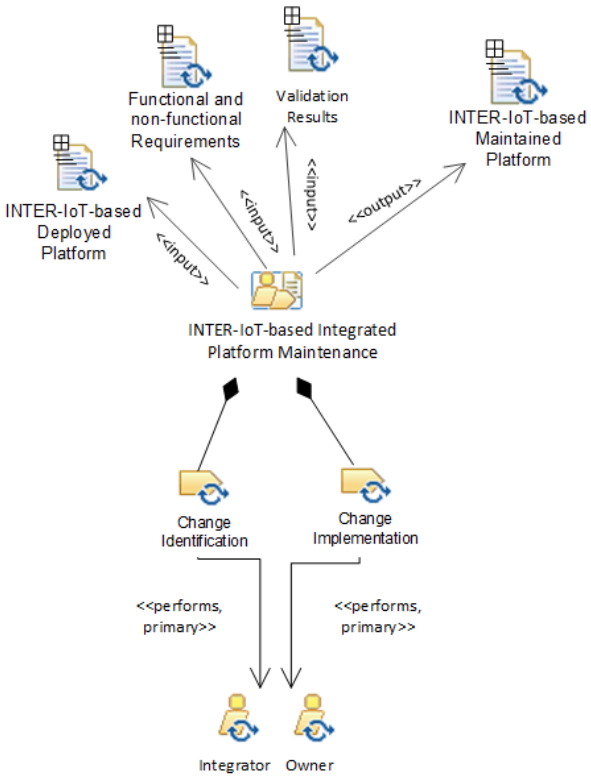

Figure 5.6.1: The Maintenance phase described in terms of activities, roles, and work products

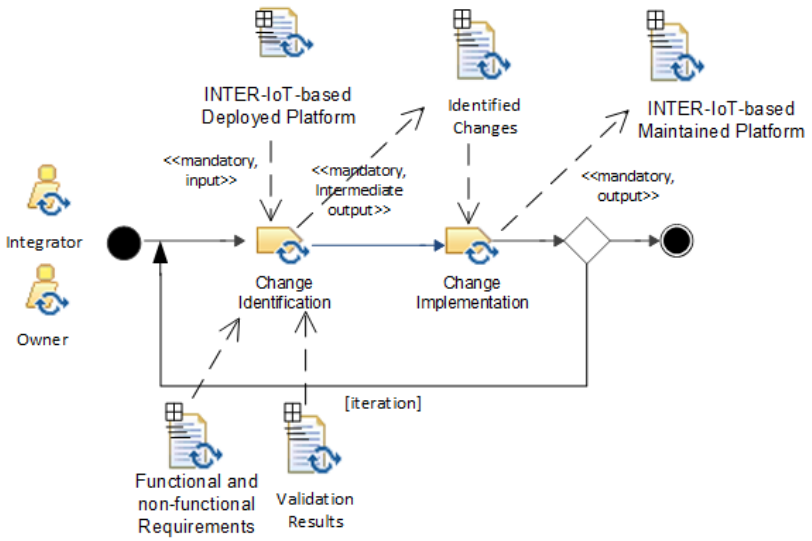

Figure 5.6.2: The workflow of tasks of the Integrated Platform Maintenance activity

5.6.2 Work Products

The final output work-product is the results of the maintenance activity, i.e. bug-fixing and/or evolution of the integrated platform.

6 On Methodology and Ethics

INTER-METH is a general-purpose methodology for supporting the integration of heterogeneous IoT platforms/systems. INTER-METH is the therefore Ethics-agnostic as Ethics come into play when dealing with specific application domains. Nevertheless, when INTER-METH is employed in such specific application domains where Ethics has an important role, Ethics need to be considered in the integration process. Specifically, the concrete instantiation of INTER-METH (see Section 5), i.e. INTER-METH based on INTER-IoT, is mainly centred on the INTER-IoT products, INTER-LAYER and INTER-FW.

Thus, in the following, we summarize Ethics related to INTER-IoT products as strongly related to INTER-METH applications.

INTER-LAYER

INTER-LAYER is designed to enable the interoperability of existing IoT systems at different levels. Ethical considerations are taken into account from two different perspectives:

- From the systems being connected,
- From the inner workings of each layer that provide such interoperability.

INTER-LAYER cannot deal with the proper ethical handling of data and security within each connected/integrated IoT system, but it can and must assure that the connection/integration enabled by INTER-LAYER follows ethical procedures and behavior. In addition, the inner logic of the components in each layer of INTER-LAYER must comply with the proper ethical handling of data and security, as per the requirements and principles described in D3.2.

INTER-FW

INTER-FW is designed to configure, monitor, and manage different layers within INTER-IoT while providing a REST-like API to enable the use of the interoperable platform and INTER-IoT features. To provide this type of functionality, ethical considerations were made on the types of data being used. In particular, the backend design should be guided by end user needs when handling each type of data and reflected in the requirements. Additionally, the possible exploitations of INTER-IoT are purposely wide ranging. This makes it difficult to exhaustively define all possible requirements. The goal then becomes to provide the end user with a customisable system capable of handling all sensitivity levels of data, even though specific contexts could require to introduce new ad-hoc mechanisms. Details on INTER-FW ethics are given in D4.2.

Finally, specific Ethics need to be taken in to account when using INTER-METH in specific and Ethics-sensible application domains like: health-care, people surveillance, people monitoring, etc., i.e. in all contexts where “humans” are central to the IoT systems. Specifically, in INTER-IoT the main use cases INTER-HEALTH and INTER-LOGP, as well as the INTER-DOMAIN use case connected to INTER-LOGP, are considered. They are “human”-centered (INTER-HEALTH more specifically) so the application of INTER-METH in such domains needs to be Ethics aware. This mainly affects the Analysis phase as the Design and Implementation (and Deployment) phases are well-supported by INTER-LAYER and INTER-FW. In particular, specific Ethics could affect the identification of the “Integration Points” and their refinement (see Section 5.1). As an example, consider the integration of two e-Health platforms (such as the ones we employed in INTER-HEALTH: UniversAAL and BodyCloud). In particular, one platform (e.g. BodyCloud) is based on Google App Engine and therefore uses the public Google cloud to store all data whereas the other platform (e.g. UniversAAL) uses a private cloud. According to Ethical requirements, data should be stored only on private clouds/data centers. This affects the identification of the “Integration Point”, therefore resulting in a specific solution: the public cloud of the first platform (e.g. BodyCloud) is not

used and the integration is performed at Middleware layer by integrating the platform through a private cloud-side server, which directly receives data streams from smartphones of monitored people, so bypassing the public Google cloud infrastructure. As one can see, according to domain-specific Ethics, the execution of the Analysis phase of INTER-METH could be affected and therefore directed to be complaint with the Ethics rules to be enforced.

7 References

[A17] ELDATool, documentation and software, http://lisdip.deis.unical.it/software/eldatool.

[A21] https://en.wikipedia.org/wiki/Rational_Unified_Process.

[AB12] “EDOAL: Expressive and declarative ontology alignment language.” http://alignapi.gforge.inria.fr/edoal.html.

[AIM10] Atzori, Luigi, Antonio Iera, and Giacomo Morabito. "The internet of things: A survey." Computer networks 54.15 (2010): 2787-2805.

[AIOTI] AIOTI - Report on High-Level Architecture (HLA) http://ec.europa.eu/newsroom/dae/document.cfm?action=display&doc_id=11812.

[B04] Bresciani, Paolo, et al. "Tropos: An agent-oriented software development methodology." Autonomous Agents and Multi-Agent Systems 8.3 (2004): 203-236.

[B07] Bass, Len. Software architecture in practice. Pearson Education India, 2007.

[B13] Bassi, et al., “Enabling things to talk”, Springer, 2013.

[BP1] Best Practices for Systems Integration, November 2011, Presented by Pete Houser Manager for Engineering & Product Excellence, Northrop Grumman Corporation, www.dtic.mil/ndia/2011system/13007_HouserThursday.pdf

[BPR01] Bellifemine, Fabio, Agostino Poggi, and Giovanni Rimassa. "Developing multi-agent systems with a FIPA-compliant agent framework." Software-Practice and experience 31.2 (2001): 103-128.

[BS11] Bandyopadhyay, Debasis, and Jaydip Sen. "Internet of things: Applications and challenges in technology and standardization." Wireless Personal Communications 58.1 (2011): 49-69.

[C02] Clements, Paul C. "Software architecture in practice." Diss. Software Engineering Institute (2002).

[C05] Cossentino, Massimo. "From requirements to code with the PASSI methodology." Agent-oriented methodologies 3690 (2005): 79-106.

[C1] Cynertia Consulting http://www.cynertiaconsulting.com/en/IT-services/audit-IT-process-improvement-CMMI-ITIL

[CT] Collins, Tom. "A Methodology for Building the Internet of Things.", http://www.iotmethodology.com/.

[D11] David, Jérôme, et al. "The alignment API 4.0." Semantic web 2.1 (2011): 3-10.

[DJ11] Dahmann, Judith, et al. "An implementers' view of systems engineering for systems of systems." Systems Conference (SysCon), 2011 IEEE International. IEEE, 2011.

[DWS01] DeLoach, Scott A., Mark F. Wood, and Clint H. Sparkman. "Multiagent systems engineering." International Journal of Software Engineering and Knowledge Engineering 11.03 (2001): 231-258.

[F08] Fortino, Giancarlo, et al. "Modeling multi-agent systems through event-driven lightweight DSC-based agents." Università della Calabria, Via P. Bucci cubo 41c 87036 (2008).

[F09] Fortino, Giancarlo, et al. "A multi-coordination based process for the design of mobile agent interactions." Intelligent Agents, 2009. IA'09. IEEE Symposium on. IEEE, 2009.

[F10] Fortino, Giancarlo, et al. "Using event-driven lightweight DSC-based agents for MAS modelling." International Journal of Agent-Oriented Software Engineering 4.2 (2010): 113-140.

[FGR05] Fortino, Giancarlo, Alfredo Garro, and Wilma Russo. "An integrated approach for the development and validation of multi-agent systems." International Journal of Computer Systems Science & Engineering 20.4 (2005): 259-271.

[FGRS15] Fortino, Giancarlo, et al. "Towards a development methodology for smart object-oriented IoT systems: A metamodel approach." Systems, Man, and Cybernetics (SMC), 2015 IEEE International Conference on. IEEE, 2015.

[FR12] Fortino, Giancarlo, and Wilma Russo. "ELDAMeth: An agent-oriented methodology for simulation-based prototyping of distributed agent systems." Information and Software Technology 54.6 (2012): 608-624.

[FRR14] Fortino, Giancarlo, Francesco Rango, and Wilma Russo. "Eldameth design process." Handbook on Agent-Oriented Design Processes. Springer Berlin Heidelberg, 2014. 115-139.

[FS18] Fortino, Giancarlo, et al. "Towards Multi-layer Interoperability of Heterogeneous IoT Platforms: The INTER-IoT Approach." Integration, Interconnection, and Interoperability of IoT Systems. Springer, Cham, 2018. 199-232.

[G16] Ganzha, Maria, et al. "Semantic Technologies for the IoT-An Inter-IoT Perspective." Internet-of-Things Design and Implementation (IoTDI), 2016 IEEE First International Conference on. IEEE, 2016.

[G16] Ganzha, Maria, et al. "Tools for ontology matching—practical considerations from INTER-IoT perspective." International Conference on Internet and Distributed Computing Systems. Springer International Publishing, 2016.

[G17] Ganzha, Maria, et al. "Streaming semantic translations." System Theory, Control and Computing (ICSTCC), 2017 21st International Conference on. IEEE, 2017.

[G18] Ganzha, Maria, et al. "Towards semantic interoperability between Internet of Things platforms." Integration, Interconnection, and Interoperability of IoT Systems. Springer, Cham, 2018. 103-127.

[GG17] Ganzha, Maria, et al. "From implicit semantics towards ontologies—practical considerations from the INTER-IoT perspective." Consumer Communications & Networking Conference (CCNC), 2017 14th IEEE Annual. IEEE, 2017.

[GGG17] Ganzha, Maria, et al. "Semantic interoperability in the Internet of Things: an overview from the INTER-IoT perspective." Journal of Network and Computer Applications 81 (2017): 111-124.

[GGM05] Garijo, Francisco J., Jorge J. Gomez-Sanz, and Philippe Massonet. "The MESSAGE methodology for agent-oriented analysis and design." Agent-oriented methodologies 8 (2005): 203-235.

[GMP02] Giunchiglia, Fausto, John Mylopoulos, and Anna Perini. "The tropos software development methodology: processes, models and diagrams." Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1. ACM, 2002.

[GMP02] Giunchiglia, Fausto, John Mylopoulos, and Anna Perini. "The tropos software development methodology: processes, models and diagrams." Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1. ACM, 2002.

[GP14] Grace, Paul, et al. "Taming the interoperability challenges of complex iot systems." Proceedings of the 1st ACM Workshop on Middleware for Context-Aware Applications in the IoT. ACM, 2014.

[GPS16] Grace, Paul, Brian Pickering, and Mike Surridge. "Model-driven interoperability: engineering heterogeneous IoT systems." Annals of Telecommunications 71.3-4 (2016): 141-150.

[GVO05] Gardelli, Luca, Mirko Viroli, and Andrea Omicini. "On the role of simulation in the engineering of self-organising systems: Detecting abnormal behaviour in MAS." (2005).

[HW04] Hohpe, Gregor, and Bobby Woolf. Enterprise integration patterns: Designing, building, and deploying messaging solutions. Addison-Wesley Professional, 2004.

[I1] The IoT-A Unified Requirements list, <<http://www.iot-a.eu/public/requirements/copy_of_requirements>>

[IOTA1] http://docs.oasis-open.org/soa-rm/RS12.0/soa-rm.pdf

[K08] Katasonov, Artem, et al. "Smart Semantic Middleware for the Internet of Things." ICINCO-ICSO 8 (2008): 169-178.

[K10] Kortuem, Gerd, et al. "Smart objects as building blocks for the internet of things." IEEE Internet Computing 14.1 (2010): 44-51.

[K10] Kawsar, Fahim, et al. "Design and implementation of a framework for building distributed smart object systems." The Journal of Supercomputing 54.1 (2010): 4-28.

[KR13] Kazman, Rick, et al. "Understanding patterns for system of systems integration." System of Systems Engineering (SoSE), 2013 8th International Conference on. IEEE, 2013.

[M12] Miorandi, Daniele, et al. "Internet of things: Vision, applications and research challenges." Ad Hoc Networks 10.7 (2012): 1497-1516.

[MAK12] Münch, Jürgen, et al. "Process Modeling Notations and Tools." Software Process Definition and Management. Springer Berlin Heidelberg, 2012. 111-138.

[MMW96] Maier, Mark W. "Architecting principles for systems-of-systems." INCOSE International Symposium. Vol. 6. No. 1. 1996.

[MMZ99] Martelli M., Mascardi, V. and Zini, F., Specification and Simulation of Multi-Agent Systems in CaseLP, Appia-Gulp-Prode Joint Conf. on Declarative Programming, L'Aquila, Italy. M.C. Meo and M. Vilares-Ferro (eds), pp. 13-28, 1999.

[P14] Perera, Charith, et al. "Context aware computing for the internet of things: A survey." Communications Surveys & Tutorials, IEEE 16.1 (2014): 414-454.

[PS06] Pavón, Juan, Candelaria Sansores, and Jorge Gómez-Sanz. "Modeling of social systems with Ingenias." Proc. of 1st Workshop on Multi-Agent Systems and Simulation (MAS&S'06). 2006.

[PW02] Padgham, Lin, and Michael Winikoff. "Prometheus: A methodology for developing intelligent agents." Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1. ACM, 2002.

[RS12] Robertson, Suzanne, and James Robertson. Mastering the requirements process: Getting requirements right. Addison-wesley, 2012.

[RU04] Röhl, Mathias, and Adelinde M. Uhrmacher. "Controlled Experimentation with Agents-Models and Implementations." ESAW. 2004.

[S04] Sierra, Carles, et al. "Engineering multi-agent systems as electronic institutions." European Journal for the Informatics Professional 4.4 (2004): 33-39.

[SBZH01] Sarjoughian, Hessam S., Bernard P. Zeigler, and Steven B. Hall. "A layered modeling and simulation architecture for agent-based system development." Proceedings of the IEEE 89.2 (2001): 201-213.

[SGPPW] Paweł Szmeja, Maria Ganzha, Marcin Paprzycki, Wiesław Pawłowski, and Katarzyna Wasielewska. Declarative Ontology Alignment Format for Semantic Translation.

[SOS6] OUSD AT&L. Systems Engineering Guide for Systems of Systems. Washington, D.C.: Pentagon, August 2008.

[SOS7] http://www.ieee-stc.org/proceedings/2009/pdfs/JOC2249.pdf

[SPEM08] SPEM 2.0 specifications at http://www.omg.org/spec/SPEM/

[SPMB] D. Slama, F. Puhlmann, J. Morrish, R. Bhatnagar, “Enterprise Internet of Things,” available at http://enterprise-Internet of Things.org/book/enterprise-Internet of Things/

[TE1] “Acceptance Testing,” [Online]. Available: https://en.wikipedia.org/wiki/Acceptance_testing. [Accessed 16 11 2017].

[TE2] “Standard glossary of terms used in Software Testing, Version 2.1. ISTQB. 2010.,” [Online]. Available: https://en.wikipedia.org/wiki/Acceptance_testing#cite_ref-ISTQB_Glossary_2-0. [Accessed 16 11 2017].

[TE3] “D6.1 System Integration Plan,” 2017.

[TSGPSN] Rafał Tkaczyk, Paweł Szmeja, Maria Ganzha, Marcin Paprzycki, Bartłomiej Solarz-Niesłuchowski. From relational databases to an ontology – practical considerations.

[US98] Uhrmacher, Adelinde M., and Bernd Schattenberg. "Agents in discrete event simulation." European Simulation Symposium-ESS. Vol. 98. 1998.

[V13] Vlacheas, Panagiotis, et al. "Enabling smart cities through a cognitive management framework for the internet of things." IEEE communications magazine 51.6 (2013): 102-111.

[VK16] Vaneman, Warren K. "The system of systems engineering and integration" Vee" model." Systems Conference (SysCon), 2016 Annual IEEE. IEEE, 2016.

[W1] Waterfall methodology: https://en.wikipedia.org/wiki/Waterfall_model

[WJK00] Wooldridge, Michael, Nicholas R. Jennings, and David Kinny. "The Gaia methodology for agent-oriented analysis and design." Autonomous Agents and multi-agent systems 3.3 (2000): 285-312.

[WJK00] Fortino, Giancarlo, et al. "An agent-based middleware for cooperating smart objects." International Conference on Practical Applications of Agents and Multi-Agent Systems. Springer, Berlin, Heidelberg, 2013.

[Z16] Zambonelli, Franco. "Towards a General Software Engineering Methodology for the Internet of Things." arXiv preprint arXiv:1601.05569 (2016).

[ZJW03] Zambonelli, Franco, Nicholas R. Jennings, and Michael Wooldridge. "Developing multiagent systems: The Gaia methodology." ACM Transactions on Software Engineering and Methodology (TOSEM) 12.3 (2003): 317-370.

Appendix A: SPEM in a nutshell

Software Process Engineering Metamodel (SPEM) [SPEM08] is a meta-modelling language used for the description of development design processes and their components. The SPEM specification defines a set of process modelling elements useful for describing a development design process with the exclusion of all the constraints on the specific application context. The SPEM notation is suitable for a standardized representation of complex and detailed models. The SPEM is based on the idea that "a software development process is a collaboration between abstract entities called process roles that perform operations called activities on tangible entities called work products". This well-defined conceptual model allows to represent every kind of process lifecycle (iterative, incremental, waterfall and so on), SPEM in fact is composed of a breakdown structure allowing to represent the design process; the analysis of the process is divided into categories.

SPEM 2.0 presents a metamodel structured in seven main packages. Only three of them will be described in this section in order to justify their specific use: Process Structure, Method Content and Process With Method. Each of them contains all metamodel classes and abstractions for describing and modelling a design process.

It is worth noting that in SPEM 2.0 the concepts of "Method" and "Process" have a specific meaning, explained in the following paragraphs. That meaning is used for the purpose of standard representation of a design process and of a process fragment.

The **Method Content** package contains all the elements for creating a set of reusable methods, independently from a specific project, in which techniques and concrete realizations of best practices are specified. The aim is to illustrate which are the goals that a method has to reach, which resources are used and which roles are involved.

The **Process Package** is composed of main elements for process modelling: activities, nested in a breakdown structure where the performing Role classes and the input and output Work Product classes for each activity are listed. These elements are used to represents a high-level process that, when instantiated on a specific project, takes method content elements and relates them into partially-ordered sequences that are customized to specific project types.

The **Process With Method Package** contains all the elements for integrating the method content in the process.

To sum up, “**process” provides** a complete breakdown structure for a specific development situation, whereas a **process with methods** relates each method content element (such as Tasks Definition, Work Products Definition, etc.) to an ordered sequence of work steps customized for specific project types. Therefore, at a first level of abstraction, a process is represented in its general structure without any reference to a specific project and without detailing the inner methods of each activity.

SPEM 2.0 presents two level of process representation, both of them aim at modelling the portion of work to be done with required input/output and role involved, but whereas a method is considered at an higher abstraction level, where there is no reference to a specific project and above all it is considered as an auto consistent portion of process, a process is the concrete representation of a specific development situation.

For these reasons SPEM 2.0 is suitable both for the proposed top-down decomposition/ representation of a design process and for the representation of each process fragment; in fact we can use the Method Content Package elements to represent a process fragment and the Process With Method Package elements to represent an existing design process; in fact a development process, in the proposed approach, is composed of process fragments.

In Figure A.1 the metamodel classes composing the Process with Method Package are shown. The central element is the BreakdownElement. In SPEM 2.0 processes are represented with a breakdown structure of Activities that nest BreakDown Elements.

BreakDown Elements definition from Process Package, not shown here [SPEM08], is "the abstract generalization for any type of Process Element that is part of a breakdown structure. Any of its concrete subclasses can be placed inside an Activity that represents a grouping of nested BreakDownElements such as other Activity instances. Task Uses and Role Uses". Thus, Activity represents a basic unit of work within a Process as well as a Process itself.

Role Uses represent a "Role in the context of a specific Activity, it is the performer of a Task Uses and defines a set of related skills, competencies and responsibilities of a set of individuals" and Task Uses define the unit of work that is performed by Roles; a Task Use has a clear purpose in which the performing roles achieve a well-defined goal. They provide complete step-by-step explanations of doing all the work that needs to be done to achieve a goal. The Work Product Uses are the artifacts, produced, consumed or modified by Task Uses1; Roles use Work Products to perform Tasks and produce Work Products in the course of performing Tasks. Method Content Use are the key concept that carries out the separation between process and method content. As it could be noted the Process With Method Package contains the same elements we considered in the proposed process definition, the Activity can be related/mapped to Task Use, The Process Role to Role Use and obviously the Work Product to Work Product Use. These elements together with the rationale of the work breakdown structure, which SPEM is based on, are sufficient enough for describing a design process following a top-down approach, from higher level definition of the work to be done until the details of each task with the role performing it and the artifacts produced.

Besides we used other two elements not included in the process metamodel, they are provided by SPEM and are useful for grouping a set of activities under a common application theme, they are the Process Component: that is a "special Process Package that applies the principles of encapsulation. A Process Component contains exactly one Process represented by an Activity and defines a set of Work Product Ports that define the inputs and outputs for a Process Component", and Phase that represents "a significant period in a project, ending with major management checkpoint, milestone or set of Deliverables. It is included in the model as a predefined special Activity, because of its significance in defining breakdowns."

Figure A.1 portrays which SPEM elements we use and the related relationships that we use for representing a design process, an example will be provided in the following section. A whole process can be divided into process components that groups the activities under a common theme, set of activities are also grouped into phases that impose specific milestones to the work to be done. Techniques, methods and guidelines for each activity are given by tasks that are performed by roles and consumes/produces (has input/output) work products.

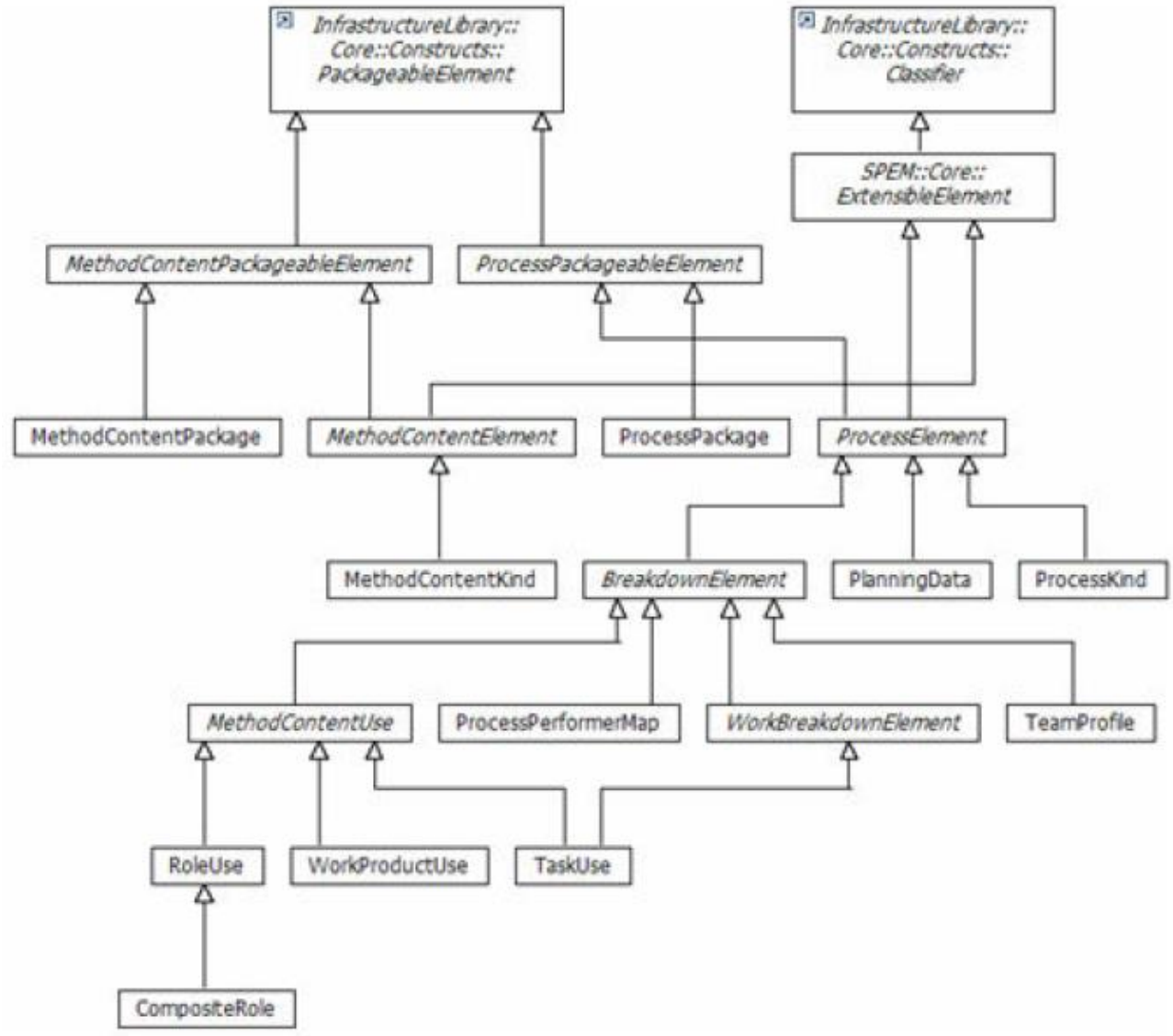

Figure A.1: The SPEM Process With Method Package Metamodel (from [SPEM08])

Appendix B: Semantic interoperability in the INTER-IoT

B.1 Semantic translation as an integration method

The main issue of Internet of Things is lack of interoperability between IoT artifacts (e.g. platforms/systems/applications/devices). Papers [G18][GG17][G16][GGG17][FS18][G17] explain the problem and describe semantic approach as promising solution. Nowadays, there is still no common practice for creating communication interfaces, that can be easily used for integration between artifacts on semantic level. Paper [TSGPSN] shows, that after research and taking into account results of other EU projects, use of semantic technologies is a noteworthy direction for exploration, as an approach integrating heterogeneous data sources (including divergent data models).

INTER-IoT uses alignments, and alignment-based translation as a method for establishing semantic interoperability. Paper [G16] explains the alignment format and differences with other related terms (ontology merging and translation). Ontology alignment refers to finding correspondences between two or more ontologies. The result of this process is an alignment - a set of correspondences between atomic entities or groups of entities and sub-structures from different ontologies. A correspondence can be either a predicate about similarity, called a matching, or a logical axiom - a mapping. Typically used mapping axioms are equivalence and subsumption.

In INTER-IoT, an application of ontology alignments is used to facilitate the message translation. Since the “world of distributed applications” is still ways away from the Semantic Web, and since legacy applications are not based on ontologies and semantic data processing, as a first step approaches to lifting other formats e.g. XML Schema, JSON to the RDF/OWL ontologies should be proposed. For IoT artifacts that use different ontologies (predefined or defined as a result of extraction/formalization), such ontologies can be aligned to build common understanding. The core of this approach in the INTER-IoT environment is Inter Platform Semantic Mediator (IPSM). The aim of this component is to execute translation between source IoT artifact ontology and target IoT artifact ontology. The detailed description can be found in [G17] and Deliverable D3.2. The IPSM infrastructure is based on the concept of data streams and channels, i.e. data flow abstractions. This idea is related to the huge amounts of real-time data that need to be handled within the INTER-IoT environment.

The general idea of INTER-IoT approach to semantic interoperability is described in [G18]. It starts with optional step of lifting other formats to OWL ontologies for all artifacts that are to be joined in an IoT ecosystem. Next step is an instantiation of a central modular ontology. It consists of two elements: (1) core IoT ontology, (2) and modules representing domains of communication within the ecosystem. The basis of the semantic translation is the definition and persistence of alignments. To achieve it, message flows should be known in advance (the platforms that participate in communication and the topics of conversation). INTER-IoT uses IPSM (many instances are allowed) for semantic translation of messages. This is proceeded with syntactic translation that is a functionality of a bridge component (described in Deliverable D3.2) placed between IoT platform and INTER-MW. The translation produces RDF from platform’s native format. Inside INTER-MW, the

IPSM is used to translate from source RDF to target RDF. Specifically, a “double translation” is performed, first to the common ontology, then to the ontology of the receiver.

B.2 Establishing semantic interoperability

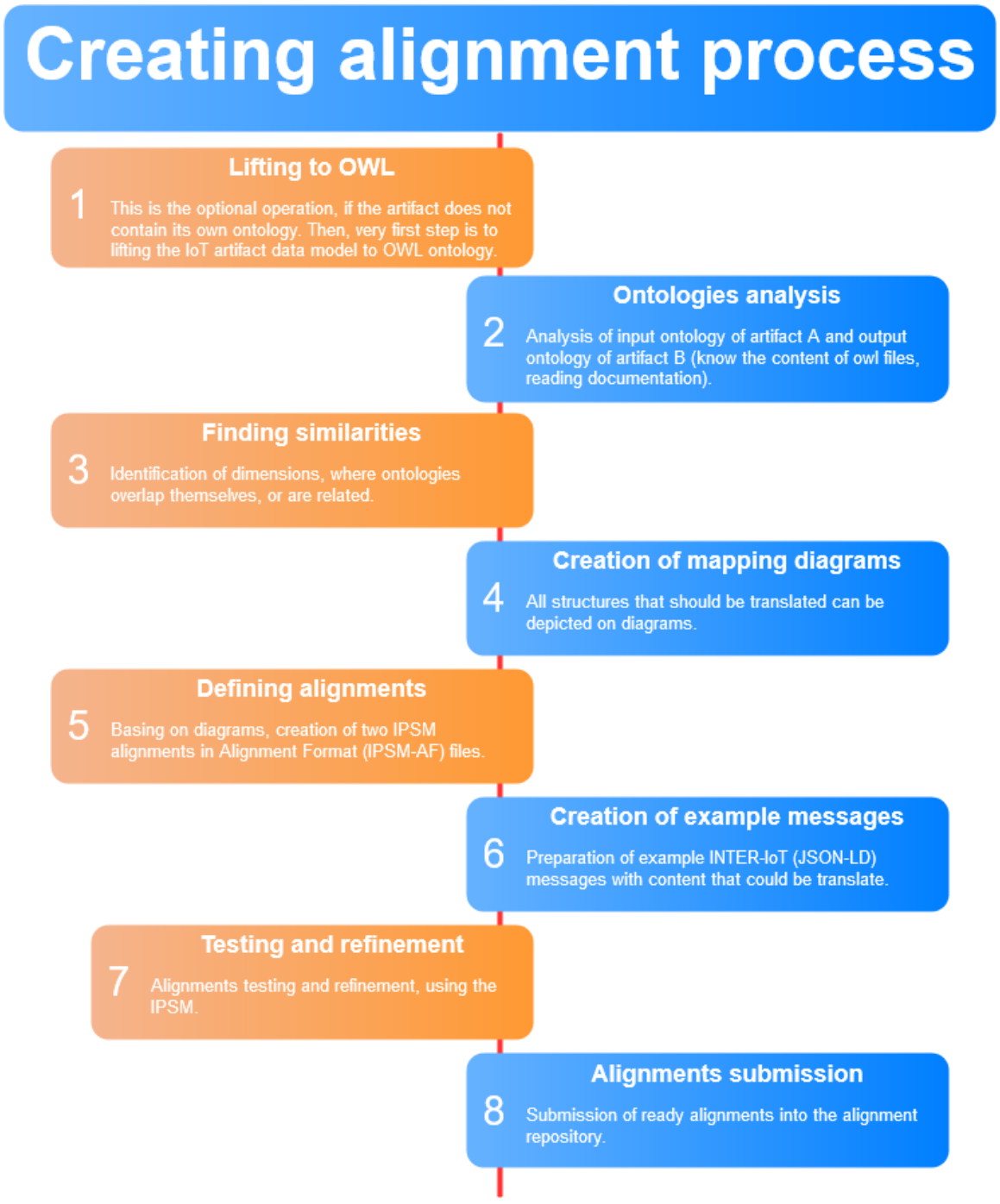

Figure B.1: Creating alignment process

Diagram (see Figure B.1) depicts the whole process of establishing semantic interoperability, necessary in the IoT artifact integration process. It consists of eight main steps, and describes the

path to the point when artifact is ready to join to the INTER-IoT environment (taking into account if it contains the ontology), to the submission of a brand new, tested alignment(s).

Step 1. Lifting to OWL. Nowadays, it is not common, that systems are based on ontologies and semantic data processing. Therefore, the first step in integration process is lifting the IoT artifact to OWL ontology. Papers [GG17], [TSGPSN] and [G16] introduces the methods and tools, that can useful in this operation and also explain common problems, related with it. For instance, there can be many, different sources of data structure, e.g. XML, JSON, relational and non-relational databases, etc. Moreover, there is no universal tool for creating/supporting creating alignment file, that would contain all basic functionalities, necessary in designing process.

Step 2. Ontologies analysis. If the IoT artifacts (that participate in integration process) contains the OWL ontologies, the next step is analysis of input and output ontologies. The clue of this operation is to know the content of owl file (i.e. graph structure: classes, data and object properties, annotations, etc.) and reading documentation.

Step 3. Finding similarities. Step 2 is necessary to know both input and output ontologies, in order to observe the similarities between them, i.e. structures that are overlap or related.

Step 4. Creation of mapping diagrams. All structures that should be translated can be depicted on diagrams.

Step 5. Defining alignments. Basing on diagrams, creation of two IPSM alignments in Alignment Format (IPSM-AF) files. IPSM-AF is inspired by inspired by the Alignment API [D11] and EDOAL [AB12]. First alignment describes the translation rules from source IoT artifact to central ontology. Second alignment describes translation rules from central ontology to output IoT artifact.

Step 6. Creation of example messages. Preparation of example INTER-IoT (JSON-LD) messages with content that could be translated. The benchmarks should contain both input and output content, in order to compare the results of IPSM translation. The auxiliary diagrams, reflecting the content, would be helpful.

Step 7. Testing and refinement. Using the prepared benchmarks, alignments should be tested. Moreover, alignments should be refined, after analysing of the tests results.

Step 8. Alignments submission. Ready and tested alignments can be submitted to alignment repository.

The significant aspect of creating alignment process is to hold on to the established IPSM-AF standard. It is described in details in [SGPPW] and [G17]. The future work of Task 5.3 assumes design and development the IPSM Alignment Editor CASE tool that shall provide support in creating (and editing) alignments, using IPSM-AF format. The main idea is to create an alignment in two ways: (1) in graphical environment, and (2) in the text mode. The main functionalities will be: generating the code on the basis of input properties (e.g. header properties, cells, steps, ontologies sources and targets, etc.), syntax validation, hints, syntax coloring and error checking. The tool will not support the whole aforementioned process, but only Step 5 - Creating alignment. Nevertheless, it is the most significant and sensitive stage of whole operation. Moreover, at present, there is no such existing tool. Therefore, it was decided to create this editor.