INTER-Meth: A Methodological Approach for the Integration of Heterogeneous IoT Systems

Giancarlo Fortino, Raffaele Gravina, Wilma Russo, Claudio Savaglio, Katarzyna Wasielewska, Maria Ganzha, Marcin Paprzycki, Wiesław Pawłowski, Paweł Szmeja, Rafał Tkaczyk

Abstract The Internet of Things (IoT) is a jeopardized ecosystem in which heterogeneity is intrinsic at all levels, from physical devices to communication protocols till high-level application semantics. The absence of IoT standards increases the complexity of integration and interoperability among heterogeneous platforms. This generates a strong demand for proper methodologies in order to fully support the development of heterogeneous, yet interoperable, IoT systems. To fill this gap, in this chapter the INTER-METH engineering methodology is presented. Developed in the context of the European H2020 INTER-IoT project, INTER-METH supports the integration of heterogeneous IoT platforms from the analysis to the maintenance phase. Its abstract and instantiated process schema are described, with particular focus on the analysis and design phases that are fundamental drivers of the whole integration process. Relevant interoperability design patterns, the building blocks of the design phase, will be discussed. The chapter also presents the INTER-CASE tool associated to the methodology which is useful to guide integrator designers in properly following the INTER-METH workflow. Finally, the chapter shows the proposed methodology and its tool in action, with the practical integration of BodyCloud and UniversAAL platforms adopted in the INTER-Health pilot of the INTER-IoT project.

1 Introduction

The IoT domain is stimulating the interest of academia and industry, thus generating considerable yet uncoordinated research efforts. As a result, high degree of heterogeneity is characterizing IoT scenarios at all levels, obstructing interoperability of IoT devices, systems, and applications [1] and gen-

Giancarlo Fortino
University of Calabria, Italy e-mail: `g.fortino@unical.it`

erating major technological and business development issues. For instance, lack of interoperability causes impossibility to plug third-party IoT devices into existing IoT platforms, makes very hard the development of IoT applications exploiting multiple platforms, discourages the adoption of IoT technology, increases maintenance costs, reduces reusability of existing technical solutions, and as a natural consequence generates user dissatisfaction. To tackle the rapid proliferation of poorly interoperable IoT systems, the H2020 EU-funded project INTER-IoT aimed to design, implement and evaluate methods and tools to enable voluntary interoperability among different IoT platforms by using a bottom-up approach [2]. Indeed, in the absence of global IoT standards, the INTER-IoT results allows IoT stakeholders to design open IoT devices, smart objects, services, and platforms leveraging on the existing ecosystem, and bring them to market quickly. In particular, INTER-IoT is based on hardware/software tools (INTER-LAYER) granting multi-layer interoperability among IoT system layers (i.e., device, networking, middleware, application service, data and semantics), on frameworks for open IoT application and system programming and deployment (INTER-FW), and on a full-fledged engineering methodology for IoT platforms integration (INTER-METH) complemented by its Computer Aided Software Engineering (INTER-CASE) tool.

INTER-METH, that is probably the most peculiar feature of the INTER-IoT project, is the subject of this chapter. INTER-METH 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. INTER-METH, whose relevance is emphasized by the absence in the literature of any other full-fledged methodology for the integration of IoT platforms, is based on a workflow composed by six phases: Analysis, Design, Implementation, Deployment, Testing and Maintenance that are in turn divided into sub-tasks. Each phase generates work-products that are inputs for the successive phase(s). It is worth noting that INTER-METH is domain agnostic by definition, intended to be extensible and customizable, and it can be associated to any specific IoT systems integration approach.

The remainder of the chapter is organized as follows. Sec. 2 provides an overview of available methodologies for integrating IoT systems and making them interoperable. Sec. 3 presents several interoperability patterns that represent the building blocks of the integration design phase. Sec. 4 introduces, phase by phase, the INTER-METH methodology, with particular emphasis on its INTER-IOT instantiation, with the goal 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. Sec. 5 describes INTER-CASE, the tool associated to the methodology which is useful to guide integrator designers in properly following the INTER-METH workflow. Ultimately, in Sec. 6, a practical use of INTER-METH and its INTER-CASE tool is presented by showing the analysis and design workflows applied for the integration of the two platforms adopted in the INTER-Health pilot. Final remarks conclude the chapter.

2 Background

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 Sec. 2.1) and (b) More Specific Methodologies for System Integration (see Sec. 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 (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.

The Waterfall development methodology [11] is a step-by-step guide to the development of software 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. The classic Waterfall methodology is composed of five main steps, in the following sequence: (1) Requirements gathering, (2) System design, (3) Implementation. (4) Verification, and (5) Maintenance. 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.

The VOLERE methodology [12] 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.). The VOLERE methodology provides several templates to deal with the different techniques and activities that it includes.

Interestingly, several development methodologies are built around the concept of Agent and are known as Agent-Oriented Software Engineering (AOSE) methodologies.

Gaia is an AOSE Methodology [13] specifically tailored to the analysis and design of agent-based systems. It 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.

Tropos [14] is an AOSE methodology strongly focuses 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.

ELDAMeth [15] is a methodology specifically designed for the simulation-based prototyping of distributed agent systems (DAS). It is based on an iterative development process covering the modeling, 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. 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 [16], a CASE tool supporting visual modelling and coding of ELDA-based MAS [33, 34], or by (ii) translation and refinement of design objects produced by other agent-oriented methodologies. 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). The developed code could be also mapped onto heterogeneous MAS platforms [38]. -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 [39]. Such evaluation can lead to a further iteration step which starts from a new (re)modelling activity.

Other interesting AOSE methodologies include MaSE [17], Prometheus [18], MESSAGE [19].

General AOSE methodologies deal with providing engineering support to systems modeled 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 of their methods could be generalized and reused 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.

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 (e.g. Gaia, Message, TROPOS, MaSe, Prometheus) and simulation-based (e.g. 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.

2.2 IoT Methodologies

In the following, we analyse currently available IoT methodologies. It is worth noting, however, that they are still at an earlier stage of development with respect to methodologies presented in the previous section.

Despite a variety of research efforts that tackle different specific issues within an IoT systems development process, a full-fledged IoT engineering methodology is still missing. Several studies proposed domain specific best practices, guidelines, checklists, and templates. For instance, Slama et al. [20] and Collins [15] created 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 proposed reference architectures and guidelines to make specific purpose devices interoperable through abstraction data models and high-level software interfaces.

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, to support interoperability and to reduce fragmentation within an IoT context. Notably, IoT-A introduced an Architecture Reference Model (ARM) [21] with the capability of generating architectures for specific systems. A reference model is, according to the OASIS [22] is "an abstract framework for understanding significant relationships among the entities of some 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". Most of the indications provided by IoT-A have inspired AIOTI (Alliance for the Internet of Things) [28] particularly for the domain model. From IOT-A we re-used its functional architecture in INTER-METH Analysis Phase.

Zambonelli [23] 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 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.

Fortino et al. [36] proposed the ACOSOMeth approach for the agent-oriented development of IoT systems [37]. ACOSOMeth uses a model-driven development approach seamlessly covering the analysis, design and implementation phases. ACOSOMeth also enables the development of even complex IoT systems of systems [35].

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. Even though such methodologies have another scope, INTER-Meth took inspiration by borrowing the ideas of:

- meta-modeling approach that is typical of the model-drive development (MDD) approach;
- agent-oriented-like approach (see [23]) allowing to simplify the definition of integration requirements analysis;
- AIOTI [28] and IoT-A [24] 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.

In [26, 25], the addressed IoT interoperability using "model-driven development" tools and techniques. In particular, there are three key contributions:

1. Interoperability models are reusable, visual software artifacts 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);
2. A graphical development tool to allow the developer to create and edit interoperability models and to also execute tests to report interoperability issues;
3. 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. 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.

This research discusses about interoperability of IoT services, systems, and (virtualized) devices and was taken in consideration in our CASE-driven integration methodology (see INTER-CASE) supporting the integration process of heterogeneous IoT platforms. In [27] System of Systems (SoS) integration is considered. SoS is defined as a "set of systems that are cooperating and interoperating while the different systems are simultaneously working as independent entities". Authors proposed methods to improve the way in which an IT architect addresses the integration problem, focusing on how to select the best integration approach in SoS context depending on the features of the environment and systems to be integrated. Some design patterns from popular patterns catalogs are analyzed by the authors who proposed a process for creating SoS based on patterns as a central architectural concept.

2.3 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.

Nevertheless, the Waterfall model can be used, after enhancement, to support INTER-Meth. In fact, the proposed INTER-Meth process is based on an iterative version of the waterfall model, as the basic waterfall is too static. Agent-oriented methodologies are suitable for the development of distributed applications and systems in terms of multi-agent systems. 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.

Regarding the overviewed IoT methodologies, they have been specifically defined for developing IoT systems totally or partially fulfilling the reference requirements for IoT systems development, but they are not 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 that is typical of the model-drive development (MDD) approach;
- The agent-oriented-like approach (see [23]) allowing to simplify the definition of integration requirements analysis;
- AIOTI [28] and IoT-A [24] 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.

3 Design Patterns for IoT systems

With the proliferation of IoT artifacts (devices/platforms/systems/services) the need arises to analyze existing solutions from the software engineering perspective. Specifically, in the context of integration and interoperability new design patterns materialized and required to be analyzed and catalogued. Note that design pattern is understood as a general reusable solution to a problem that recurs repeatedly within a specific context in software design, whereas a pattern catalog is a collection of related patterns, subdivided into a (small) number of categories. Here, we outline the design patterns identified in INTER-IoT project either by defining them from scratch or extending some existing pattern. They address issues that can be extended beyond INTER-IoT project. For more detailed information INTER-IoT deliverable D5.1.

Since so far there have been no formal guidelines to IoT integration, in INTER-IoT we have decided to decompose the the problem into layers: D2D (Device-to-Device), MW2MW (Middleware-to-Middleware), AS2AS (Applications and Services -to- Applications and Services, DS2DS (Data and Semantics -to- Data and Semantics) and CROSS layer relating them. For each of these layers design patterns have been proposed and described using the following template:

- *Pattern name* - unique name of the pattern.
- *Inspired by* - name(s) of pattern(s) that a given one is based on / extends. In most cases, when pre-existing patterns did not fully solve specific problems, new patterns were created, extending existing ones.
- *Related patterns* - other patterns, related to the given one.
- *Intent (summary)* - short description of the goal behind the pattern and the reason for using it (an extension of the "Pattern name", explaining its action/purpose).
- *Problem & Solution* - scenario that illustrates a problem and how the pattern solves it.
- *Applicability* - situations, in which the pattern is usable; context for the pattern.
- *UML representation* - structure of the pattern modeled with a UML diagram (mostly deployment and component diagrams).
- *Implementation* - extension of the "UML representation" property, i.e. description of realization and architecture.
- *Known uses* - an example usage of the pattern within the INTER-IoT pilot installation.

Fields: *UML representation*, *Implementation*, *Related patterns* and *Known uses* have not been included in what follows (find them in INTER-IoT deliverable D5.1).

3.1 D2D patterns

The two following patterns are related to design on device-to-device layer. IoT Gateway Event Subscription describes an approach to data forwarding, whereas D2D REST Request/Response describes approach to communication on D2D layer.

IoT Gateway Event Subscription

Inspired by: (1) “Publish/Subscribe” (IoT Patterns: Design Patterns for Interaction), (2) “Publish-Subscribe Channel” (EIP: Messaging Channels), (3) “Facilitator”, and (4) “Proxy” (Agent Design Patterns: by Kendall).

Intent: D2D gateway allows data forwarding (any type). Flexibility in the D2D layer is achieved by decoupling a gateway into: a physical part that handles network access and communication protocols, and a virtual part dealing with remaining gateway operations and services. Note that, in this way, data providers (communicating within the network) and their identities (known to the virtual layer) can also be decoupled.

Problem & Solution: To provide interoperability between two heterogeneous IoT devices, the solution should establish bidirectional, asynchronous communication with the ability to publish, filter and consume data. Here, the IoT gateway is used as a subscription mechanism. It is an intermediary between IoT artifacts (in D2D communication, between two devices). It allows transmission of data generated by “sensors” to the destination, and asynchronous messaging between artifacts that interact with it. If required, the gateway should perform protocol conversion to enable communication. Senders of messages (publishers) do not program messages sent directly to specific receivers (subscribers). Instead, they publish them, using defined classes, without knowledge of subscribers. Similarly, subscribers express interest in one or more classes and receive only messages of interest (without knowing publishers). Significant is the structure of the message, which should contain subscription information (e.g. message endpoint; see: “D2D REST Request/Response” pattern).

Applicability: Used within event-based communication, when asynchronous data is to be pushed/pulled to/from the gateway.

D2D REST Request/Response

Inspired by: (1) “Request-Response” (Reactive Patterns: Message Flow), (2) “Request-Reply” (EIP: Messaging Patterns), (3) “Request/Response” (IoT Patterns: Design Patterns for Interaction).

Intent: A request/response solution for gateway communication within the D2D layer.

Problem & Solution: IoT Gateway needs to communicate with IoT artifacts. It should be accessible to authorized external elements to enable reception of information and execution of control and configuration orders. For example, the main goal of IoT ecosystems is to allow heterogeneous IoT artifacts to retrieve information. Thus, the artifacts’s middleware should be able to communicate with the IoT gateway to enable needed information flows. Thus, it is desirable to connect IoT artifacts (if possible) through a HTTP/REST API using the Request/Response pattern. This communication pattern allows message exchange, in which a requester (e.g. middleware or gateway) sends a request message to a replier system, which receives and processes the request (e.g. middleware or gateway), ultimately returning a message, in response.

Applicability: Used when communication between the middleware of an IoT artifact and the IoT gateway is performed through a REST API (middleware $\rightarrow$ gateway is typically based on Publish/Subscribe). Also, for management purposes, the gateway will expose a REST endpoint where configuration and management actions can be performed using the Request/Response patterns.

3.2 N2N patterns

On network-to-network layer we have identified one pattern that addresses the problem of orchestration of different SDN network elements.

IoT Pattern for Orchestration of SDN Network Elements

Inspired by: (1) “Software-defined networking (SDN) orchestration”, (2) “Network virtualization (NV)”.

Intent: Monitoring and configuration of SDN elements (virtual-switches) with an orchestrator component (Controller) exchanging flow and control messages. To provide interoperability between different domains connected to a network or between different network topologies and/or configurations.

Problem & Solution: Domain-focus of IoT deployments isolates them from each other. One of approaches to interconnection is, instead of realizing it at the device/gateway level, to move it to upper layers. In particular, at the network layer, interoperability and exchange of information can be achieved by applying pattern that manages elements of the network that provide connection from different domains to the network itself. The pattern is used in development of virtual SDNs, where all elements are virtual resources, or instances, controlled within a central point, or orchestrator. N2N interconnection can then be performed through the SDN. Different networks (in different locations), can be virtually interconnected and belong to a single Virtual LAN. Thus, physical separation of networks becomes “invisible”, thus facilitating elastic definition of needed connectivity.

Applicability: Used when an IoT SDN is deployed, to enable its functionality. Allows total software control over network functions, and transparent N2N interoperability.

3.3 MW2MW patterns

Patterns on middleware-to-middleware layer address the issues of components decomposition, communication infrastructure and messaging between IoT artifacts.

IoT Artifact’s Middleware Simple Component

Inspired by: (1) "Simple component" (Reactive Patterns: Fault Tolerance and Recovery).

Intent: Every IoT artifact is designed to perform (in full) a single task (single responsibility principle, where each class should have only one reason to change).

Problem & Solution: In complex systems with multiple functions, it may be necessary to have these functions performed by different components. Their responsibilities are to be divided recursively, until desired component granularity is reached. This enables testing, debugging and extending complex system more efficiently, simplifying all operations.

Applicability: Basic pattern that can be universally applied. Does not impose level of granularity to be achieved, but indicates that analysis should be performed in order to end up with the best component decomposition. Should be applied recursively, remembering to not to divide components too far, to avoid trivial ones.

IoT artifact's Middleware Message Broker

Inspired by: (1) "Message Broker" (EIP: Message Routing), (2) "Broker" (IoT Patterns: Design Patterns for Interaction).

Intent: Facilitates passing messages between IoT artifacts.

Problem & Solution: In middleware, composed of several independent components, point-to-point connections should be avoided, as they result in multiple interfaces that expose operations of each component. Furthermore, having point-to-point interfaces complicates dynamic reconfiguration, matching of security constraints, and quality of service (QoS) requirements management. Message broker helps to overcome those limitations by enforcing common messaging interface upon different components. This allows components to initiate interactions with other components, no matter their architecture and purposes. Each component communicates directly with the broker only, while within the broker, each component is represented with a logical name, making its internal operation hidden from other components. Crucial is the proper format of message, which consists of the payload and the label, storing information needed by the broker.

Applicability: Central Message Broker receives messages from multiple message producers, determines their destinations (message consumers), and routes them to channels specific for their destinations. Allows decoupling the sender from the destination and maintains central control over the flow of messages. This can be achieved through usage of topics, to which consuming components can: subscribe, and proceed to consume awaiting messages.

IoT Artifact's Middleware Self-contained Message

Inspired by: (1) "Self-contained message" (Reactive Patterns: Message flow), (2) "Messaging Metadata" (SOA Patterns).

Intent: Messages contains complete information needed for execution of a specific action.

Problem & Solution: Within middleware, messages should be "pure and complete" representations of events (or commands), regardless when they are to be interpreted. Each middleware component must always be able to extract from the message, stored there, complete information needed for its routing and interpretation, with only minimal data stored within the middleware components. Each message has distinct set of types associated with it. Each middleware component processes and routes messages based solely on these types. For each message that travels "downstream", there can be a response that travels "upstream". Such messages might, for example, include additional response message type. Matching messages that go downstream with responses that go upstream can be done through remembering and distinguishing different chains of messages (conversations).

Applicability: Allows middleware components to be "contextfree", storing only a minimal information needed for message processing and routing. Can be also employed when there is no need to reference past messages, except for responses, and even then, these are only semantically linked to original messages (could exist without original messages).

IoT Artifact's Middleware Message Translator

Inspired by: (1) "Message Translator" (EIP: Message Transformation), (2) "Data Format Transformation" (SOA Patterns).

Intent: Translation of messages to/from IoT artifact's middleware internal message format and platform's proprietary data models and data formats.

Problem & Solution: The purpose of the middleware is to pass information between different IoT artifacts. However, artifacts produce/consume messages in "their" formats and data models. Hence, to make sense of exchanged messages, they have to be syntactically and semantically translated. A message translator enables conversion between proprietary data models and data formats, used by artifacts, and the internal data model and data format, used by IoT middleware components.

Applicability: Enables interoperability between different platforms without the need to introduce translations between every possible pair of platforms, i.e. translation into and out of the common INTER-IoT data model. Semantic translation from and into the internal message format is done by a dedicated IoT semantic translation component, while syntactic translation is completed in bridges to/from artifacts, as only they know the internal data syntax.

3.4 AS2AS patterns

On applications and services-to applications and services layer design patterns address the issues of service composition, orchestration and discovery.

AS2AS Flow-based Service Composition

Inspired by: (1) "Flow–based programming".

Intent: Generate execution flow that allows interoperation and composition of services from different IoT artifacts.

Problem & Solution: Pattern necessary to define execution flow that allows specific sequence of execution of multiple IoT services. Flow–based programming (FBP) defines applications and services as networks of "black box" processes, which exchange data across predefined connections by message delivery, where connections are specified externally to processes. Considered pattern allows creation of sequential execution flows using those services, thus allowing composition among different IoT services. Black boxes that represent IoT services can be linked by wiring the output of a service with the input of a different one (output messages from a service are routed to another service input). Thus, by wiring IoT services execution flow can be instantiated.

Applicability: Used in black box representation of IoT services to be interconnected through an FBP link, generating a flow.

AS2AS Service Orchestration

Inspired by: "Service Orchestration" (SOA Patterns).

Intent: To specify orchestration of services to facilitate interactions among different IoT services.

Problem & Solution: Cooperating, diverse, heterogeneous IoT artifacts involve huge number of different services that have to work together. Important is not only the message flow from point(s) to point(s) but also triggering necessary actions (during the flow). The common problem is that existing processes/actions are duplicated (not reused). This pattern allows union and orchestration of heterogeneous IoT services, creating a specific process. The main idea is to define a set of IoT nodes, i.e. services and interfaces that run within the integrated platforms. Internal, central, element wires nodes necessary to handle the specific task and controls processes.

Applicability: Reuse of process fragments. Orchestration enables composition of IoT service workflows, based on services from IoT artifacts.

AS2AS Discovery of IoT Services

Inspired by: (1) "Discovery" (IoT Patterns: Design Patterns for Interaction), (2) "Enterprise inventory" (SOA Patterns).

Intent: Registering and claiming specific services, used by the artifacts (within the IoT ecosystem).

Problem & Solution: Multiple IoT services, from different IoT platforms, provide a wide range of functionalities that have to be discoverable, to be aware of them and to use them. This pattern enables registration of services (in a service catalog), in order to find them and (potentially) use through an AS2AS layer solution. Here, only registered services, indicating their associated features, can be discovered.

Applicability: Pattern for providing service interoperability via registration and service retrieval. Applied to services that run within the IoT ecosystem, and used by other IoT artifacts.

3.5 DS2DS patterns

On data and semantics-to-data and semantics layer pattern address the problem of semantic translation, specifically how to persist translation rules and how to organize the translation process.

Alignment-based Translation Pattern

Inspired by: (1) “Message Translator” (EIP: Message Transformation), (2) “Data Model Transformation” and (3) “Metadata centralization” (SOA Patterns), (4) “Market Maker” (Agent Design Patterns).

Intent: Semantic translation of RDF messages exchanged between IoT artifacts, based on alignments (sets of correspondences) defined between artifacts’ ontologies.

Problem & Solution: Building the IoT ecosystem involves combining existing solutions, which (likely) belong to different owners and have been developed using different technologies (e.g. Web Services “combined with” a graph database, communicating using JSON messages, to exchange information with application that uses XML messages). Consequently, they differ both on syntactic and semantic level. Interoperation between artifacts should be achieved regardless of the underlying technology. Without loss of generality, we assume RDF message format, since other formats can be transformed to RDF. Semantics of messages is artifact specific (ontology can be natively supported, or semantics, e.g. expressed in XSD, can be lifted to an OWL ontology). Hence, for semantic interoperability, a method for defining correspondences should support mapping between specific URIs, parts of the RDF structure, transformations etc. The component implementing the translation should provide interfaces to submit messages to be translated and publish translated messages.

Applicability: Providing semantic translation between RDF messages exchanged between heterogeneous IoT artifacts. Translation, based on one-to-one translation (alignment), should be possible to define for any two artifacts.

Translation with Central Ontology

Inspired by: (1) “Message Translator” (EIP: Message Transformation), (2) “Data Model Transformation” and (3) “Metadata centralization” (SOA Patterns), (4) “Market Maker” (Agent Design Patterns).

Intent: Semantic translation of RDF messages exchanged between IoT artifacts, where one involves central/common data model.

Problem & Solution: To provide common understanding in the semantic translation process a modularized central ontology can be created on the basis of IoT and domain ontologies. Here, a domain ontology is a conceptual model for a specific domain, e.g. transportation, health, etc. IoT ontology describes different IoT aspects, e.g. platforms, devices, sensors, services, etc. Central ontology should reuse / be based on existing standards (e.g. SSN, SOSA, SAREF, etc.). This approach is highly scalable: it is possible to add artifacts to the existing IoT ecosystem by instantiating translations with the central ontology (i.e. creation of alignments; see above). This approach requires less preparation/work, as only a single “point of joining” has to be instantiated. Furthermore, the long-term maintenance is simplified, as changes in a single platform require localized adjustments only. The component implementing the translation should provide interfaces to submit messages to be translated and publish messages after translation (realizable via an appropriate pattern, above).

Applicability: Providing semantic translation between multiple heterogeneous IoT artifacts that are to exchange RDF messages.

3.6 CROSS-layer patterns

IoT SSL CROSS-Layer Secure Access

The CROSS-Layer pattern addresses issue that is common to the IoT-based systems regardless of the layers, i.e. security.

Inspired by: (1) Security Patterns, (2) IoT Patterns: Design Patterns for IoT Security.

Intent: Ensuring security of interactions with external interfaces (i.e. APIs) of every layer of the IoT ecosystem.

Problem & Solution: As IoT architecture is composed by diverse layers, access to each of them, as well as interactions between them, must be secure. To ensure a sufficient level of security on each of the IoT layers, different security mechanisms can be implemented: authentication of credentials, use of authentication tokens, or Secure Sockets Layer (SSL). In an IoT ecosystem, layer access will be secured with the SSL that employs the IoT SSL pattern. Every IoT layer exposes a REST API that represents an external interface accessible to the outside actors, such as other IoT layers, users, or IoT artifacts. To enable use of such APIs to only allowed actors the access is secured through SSL. REST APIs are accessible through a browser, which should provide a trusted certificate. Only after the establishment of a secure connection authentication through login will be allowed, to open the access to the layer API. Further operations on the layer API will be done using this secure connection.

Applicability: Pattern applied in interactions of any actor with the IoT environment layer’s APIs. Access can also be done internally between pairs of different layers.

4 INTER-METH

In this section, we will briefly introduce the abstract process of our INTER-METH methodology. Then, we will describe in more details its INTER-IoT instantiation, with particular attention to the Analysis and Design phases.

4.1 INTER-METH Abstract Process

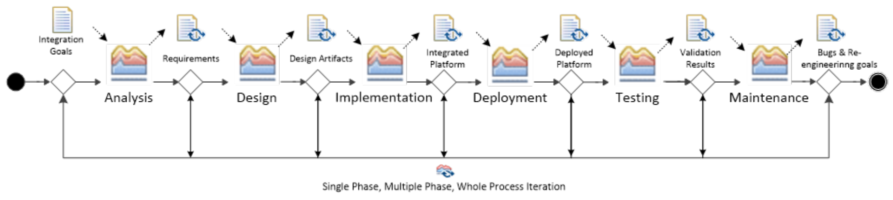


Fig. 1 INTER-METH Abstract Process Schema.

The engineering methodology INTER-METH aims at supporting the integration process of heterogeneous IoT platforms to obtain interoperability and support implementation and deployment of IoT applications on top. In this section, we introduce the abstract process of INTER-METH whose SPEM[1]-based schema is shown in Fig. 1. The process is envisioned as iterative waterfall and is composed of six main phases, each of which is divided into tasks and produces workproducts that are inputs for the successive phase. In particular:

- *Analysis Phase* supports the definition the IoT platform (non-)functional integration requirements.
- *Design Phase* produces both artifacts (e.g., diagrams) and programming/management patterns to fulfill the elicited requirements and define the integration design.
- *Implementation Phase* drives the implementation of the design workproduct to obtain the full-working (hardware/software implemented) integrated IoT platform.
- *Deployment Phase* involves the support to the operating set-up and configuration of the integrated IoT platform.
- *Testing Phase* defines the performance evaluation tests to validate the integrated platform according to the functional and non-functional requirements.
- *Maintenance Phase* manages the upgrade and evolution of the integrated system.

In detail, at the Analysis Phase, on the basis of the Integration Goals (representing high-level integration requirements), each platform is analysed according to a shared reference architecture model. Functional and non-functional integration requirements of the platforms are hence formalized in a document whose format will be specified according to the specific instantiation of the abstract INTER-METH methodology (see Sec. IV). On the basis

[1] OMG, SPEM, and O. M. G. Notation. "Software&systems process engineering meta-model specification." OMG Std., Rev 2 (2008).

of the elicited requirements at the Analysis Phase, an initial design specification is produced for each layer and iteratively refined at the Design Phase. The final workproduct is a formalized specification containing the design of the integration of the IoT platforms/systems to be interconnected/integrated. This full-fledged design specification is actually implemented through multiple refinement steps at the Implementation Phase according to the integration specifications, aiming at obtaining the final integrated platform. The final workproduct is the integrated platform, that will be based on the specific IoT platforms/systems to be integrated/interconnected and that will be deployed according to deployment goals and requirements at the Deployment Phase. Hence, the deployed platform is set-up according to the defined configuration and run specifications. The integrated and deployed platform is then executed and validated through testing according to well-defined test cases at the Testing Phase: specifically, functional and non-functional test cases (previously defined to respectively validate functional and non-functional requirements) are executed by the platform and results collected according to formalized analysis documents. Finally, to maintain and allow the evolution of the integrated IoT platform, the Maintenance Phase aims at identifying both bugs and evolution points, activities which imply to totally/partially re-execute the integration process.

4.2 INTER-METH instantiated on INTER-IoT

The INTER-METH abstract methodology has been customized for the INTER-IoT approach [2] with the aim of showing how the integration process between two or more heterogeneous IoT platforms/systems can be concretely carried out by exploiting the INTER-METH guidelines and the two INTER-IoT products INTER-LAYER and INTER-FW. INTER-LAYER is a set of interoperability solutions dedicated to each specific INTER-IoT architectural layer (Device-to-Device D2D, Networking-to-Networking N2N, Middleware-to-Middleware MW2MW, Application&Services-to-Application&Services AS2AS, Data&Semantics-to-Data&Semantics DS2DS). Thanks to its layered approach, INTER-LAYER makes the interoperability flexible and allows it to reflect the interests/needs of the stakeholders, integrators or application developers. INTER-FW, instead, allows the development of new applications and services by customizing INTER-METH and exploiting INTER-LAYER. Indeed, INTER-FW is a global framework for programming and managing interoperable IoT platforms by means of specific programming interfaces and interoperability tools for every architectural layer. As in Sec. 4.1, for each phase of the instantiated process, an overall description is reported along with a brief description of the performed tasks and obtained workproducts.

- *INTER-IoT-based Analysis Phase*: given two or more heterogeneous IoT platforms to be integrated according to certain Integration Goals, these are analyzed according to the INTER-IoT RA that is, as previously reported in Sec. 2, based on IoT-A [4]. Integration requirements are then defined for each architectural layer (according to some iterative tasks that will be elicited in Sec. 4.2.1) and formalized in the INTER-Goal Oriented Model (INTER-GOM) as final workproduct.

- *INTER-IoT-based Design Phase*: for each layer, on the basis of the elicited requirements reported in the INTER-GOM, an initial INTER-IoT Design Pattern is produced and iteratively defined by exploiting the INTER-LAYER product (D2D, N2N, MW2MW, AS2AS, DS2DS), thus producing five instantiated INTER-IoT Design Patterns. On the basis of these Patterns, a full-fledged specification is iteratively produced as workproduct by incorporating also the CROSS-LAYER and INTER-FW INTER-IoT Design Patterns. These patterns are integration solutions structured according to certain templates (describing pattern's main properties, e.g., pattern name, its intent, its known uses) and formalized through XML files providing domain-specific guidelines to be exploited for driving the INTER-IoT-based Implementation Phase.
- *INTER-IoT-based Implementation Phase*: it consists in the (i) configuration of the components of the Integrated Platform by means of the INTER-FW; (ii) potential extension of the components of the Integrated Platform (e.g., a new functionality enabled by the integration needs to be implemented); and (iii) implementation, in terms of software bridges connected to INTER-LAYER, of the INTER-IoT based design patterns. The final output workproduct is the INTER-IoT-based Integrated Platform (if needed, an ontology alignment for finding correspondences among entities and sub-structures from different ontologies can be performed). Indeed, at this point, the heterogeneous IoT platforms are integrated according to the INTER-IoT approach, thus obtaining interoperability among them and allowing implementation and deployment of IoT applications on top of them.
- *INTER-IoT-based Deployment Phase*: the following six tasks have to be performed for the deployment of the Integrated Platform according to deployment goals and requirements: (i) Platform Configuration, which aims to instantiate and deploy an IoT Platform Middleware in the INTER-FW; (ii) Gateway Configuration, focused on the device to device interoperability, addressed in the scope of INTER-IoT in the D2D Layer; (iii) Networking Configuration, which achieves network interoperability via network virtualization to support the needs of the N2N Layer; (iv) Application Services Configuration, which includes a graphic tool for service orchestration, namely the underlying interoperability mechanism for AS2AS Layer; (v) Semantics Configuration, which manages all the processes and mechanism of DS2DS Layer enabling ontologies interoperability; (vi) User Management Configuration, to configure and manage the users of the INTER-FW and their authorized access to the IoT resources connected in INTER-IoT.

- *INTER-IoT-based Testing Phase*: the integrated, configured and deployed platform is executed and validated through well-defined test cases. In particular, to determine if the requirements of a specification are met, Factory Acceptance Testing (FAT) task is performed by simulation whereas Site Acceptance Testing (SAT) task takes place after integration at the customer site and tests if the solution has been correctly integrated into the customer's environment. The Defect Reporting task, instead, is in charge of identifying and reporting issues emerged in FAT and SAT tasks as well as implementing, integrating and testing the related solutions. FAT and SAT documents are the output workproducts of such phase.
- *INTER-IoT-based Maintenance Phase*: it allows the maintenance and evolution of an integrated and already deployed IoT platform. It first identifies bugs and/or evolution points at each layer as well as at cross-layer of the integrated platform (i.e., at INTER-LAYER) and framework level (i.e., at INTER-FW), and then at actually develops the required changes.

4.2.1 INTER-IoT-based Analysis Phase

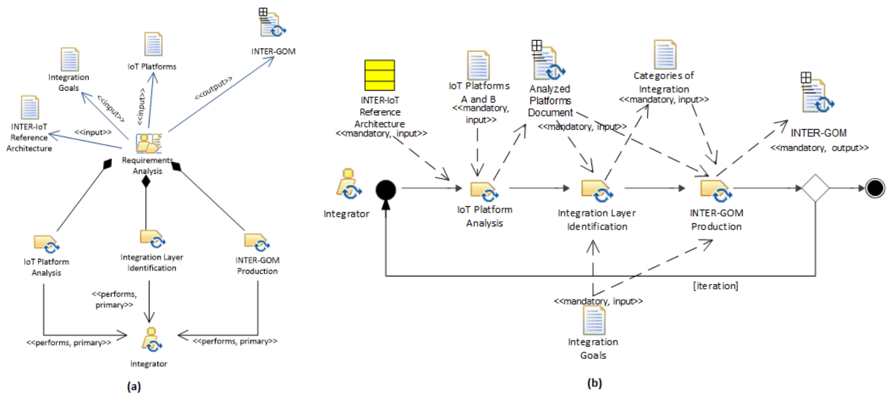


Fig. 2 The INTER-IoT-based Analysis phase: (a) Requirement Analysis activity overall description and (b) its workflow.

The INTER-IoT-based Analysis Phase involves the Requirement Analysis activity, which is described in Fig. 2(a) in terms of tasks, roles, and workproducts as well as in terms of workflow in Fig. 2(b). The Requirement Analysis comprising the following three main tasks that are performed by the integrator: the IoT Platforms Analysis, the Integration Layer Identification, and the INTER-GOM Production.

The *IoT Platforms Analysis Task* receives two or more heterogeneous IoT systems as inputs and, according to the INTER-IoT RA, produces the Analyzed Platforms Document. This step allows heterogeneous IoT platforms with even notably different architectures to be compared by means of a common set of architectural solutions and building blocks. In particular, INTER-IoT RA is depicted in Fig. 3 and consists in the following Functional Groups (FG):

- Service Interoperability FG, supporting the AS2AS interoperability through the definition and execution of new compound services that make use of already existing services in the underlying IoT Platforms; it uses services from different IoT Platforms and create new services based on them.
- Semantics FG, addressing the challenges related to semantic interoperability of IoT Platforms; it provides support for the Service Interoperability FG, the Platform Interoperability FG and the Device Interoperability FG.
- Platform Interoperability FG, interacting with the different IoT Platforms to be interconnected; it is the responsible for accessing the IoT Platforms.
- Device Interoperability FG, addressing the challenges of making legacy devices and non-real IoT Platform interoperable with other IoT Platforms and systems.
- Device Access FG, that 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, considering all the functionalities 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.

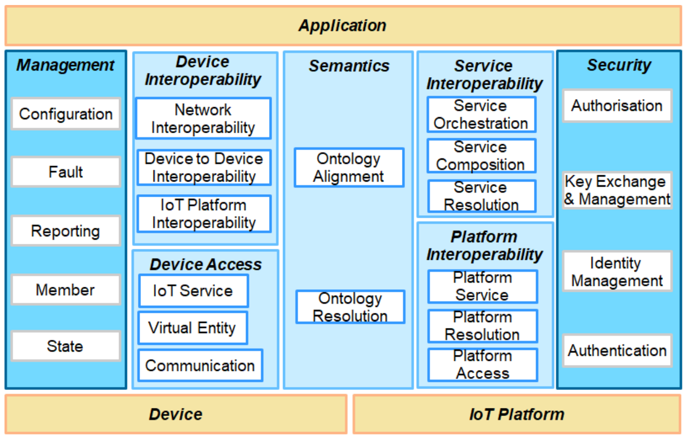


Fig. 3 INTER-IoT Reference Architecture schema.

The *Integration Layer Identification Task* receives the Analyzed Platform Document and a set of Integration Goals as input. 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. Although a layer-oriented approach is more challenging than an application-level approach, it has a higher potential to deliver tight bidirectional integration among heterogeneous IoT platforms, thus providing flexibility, modularity, higher performance, reliability, and security. This layer-oriented solution is achieved through INTER-LAYER and includes several interoperability solutions dedicated to specific layers. The INTER-LAYER architecture is reported in Fig. 4 and comprises the following six layers: (i) Device allows the seamless inclusion of novel IoT devices and their interoperation with already existing ones; (ii) Network(ing) aims to provide seamless support for smart objects mobility and information routing; (iii) Middleware enables seamless resource discovery and management system for the IoT devices in heterogeneous IoT platforms; (iv) Application&Services enables the use of heterogeneous services among different IoT platforms; (v) Semantics&Data allows a shared interpretation of data and information among heterogeneous IoT systems and data sources, achieving semantic interoperability; and (vi) CROSS-LAYER covers and guarantees non-functional aspects that must be present across all layers: trust, security, privacy, and quality of service (QoS).

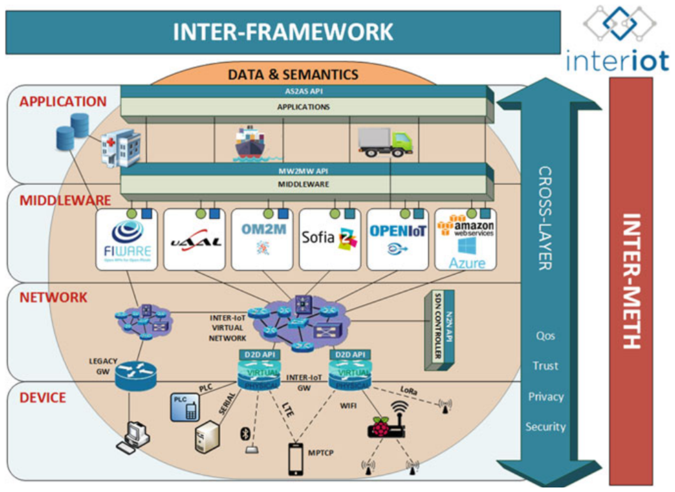


Fig. 4 The INTER-LAYER approach schema

Finally, the *INTER-GOM Production Task* receives the Analyzed Platform Document, the Integration Goals and the Categories of Integration, and produces the INTER-GOM. This task can be iterated one or more time, thus obtaining the final INTER-GOM that will represent the formal requirements model and drive the INTER-IoT-based Design Phase. In particular, the INTER-GOM comprises the following components, according to its Metamodel depicted in Fig. 5(a): (i) the Analyzed Platform Document, which compares the platforms using the shared the INTER-IoT RA to identify their integration points; and (ii) one or more Integration Points IP [14], which put together parts of the platforms according to the CoI and one or more Requirements. Requirements represent the states to be achieved by the IP; they are progressively refined into intermediate goals, until the process produces actionable goals/tasks that need no further decomposition and can be executed. According to the Analyzed Platform Documents and the IP, the GOM is defined following an iterative procedure as shown in the activity diagram of Fig. 5(b). After having analyzed IoT platforms/systems whose

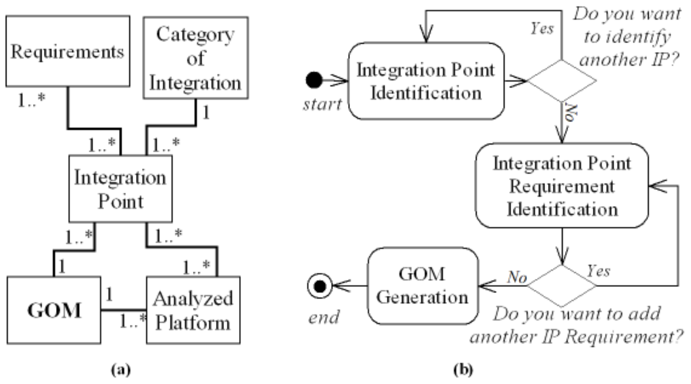

Requirements
Category of Integration
1..*
1
1..*
1..*
Integration Point
1..*
1..*
1
1..*
GOM
1
1..*
Analyzed Platform
(a)
Do you want to identify another IP?
Yes
start
Integration Point Identification
No
Integration Point Requirement Identification
GOM Generation
No
Yes
end
Do you want to add another IP Requirement?
(b)

Fig. 5 (a) INTER-GOM Metamodel and (b) UML Activity Diagram of INTER-GOM Production.

formal integration requirements are reported in INTER-GOM model, process is ready to move toward the INTER-IoT-based Design phase.

4.2.2 INTER-IoT-based Design Phase

Given two or more IoT platforms/systems whose formal integration requirements are reported in INTER-GOM model, a set of INTER-IoT Design Patterns have to be produced (see Sec.3), by instantiation, on the basis of the available pattern templates (semi-instantiated patterns). For each layer, on the basis of the elicited requirements reported in the INTER-GOM, an initial INTER-IoT Design Pattern is produced and then usually iteratively refined. The Integration Design activity, which is the only main activity of the INTER-IoT-based Design phase, is subdivided into two subtasks that are performed by the Integrator according to the workflow depicted in Fig. 6: (i)

INTER-IoT Layer Integration Design Specification and (ii) INTER-IoT Full Integration Design Specification. On the basis of the INTER-GOM and Analyzed Platforms Document, for each INTER-IoT Layer a layer integration specification is iteratively defined by exploiting the INTER-LAYER product. Such task will produce instantiated INTER-IoT Design Patterns (see Section 3), one for each INTER-IoT layer. Then, 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 1.2, where is reported also their related INTER-IoT layer and inspiring Pattern Catalog (if any).

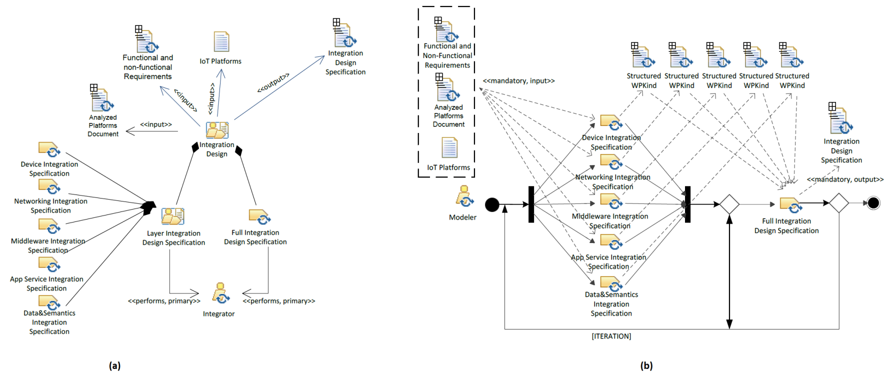

Functional and non-functional Requirements
IoT Platforms
Integration Design Specification
<<input>>
<<input>>
<<output>>
Analyzed Platforms Document
<<input>>
Integration Design
Device Integration Specification
Networking Integration Specification
Middleware Integration Specification
App Service Integration Specification
Data&Semantics Integration Specification
Layer Integration Design Specification
Full Integration Design Specification
<<performs, primary>>
<<performs, primary>>
Integrator
(a)
Functional and Non-Functional Requirements
Analyzed Platforms Document
IoT Platforms
Modeler
<<mandatory, input>>
Structured WPKind
Structured WPKind
Structured WPKind
Structured WPKind
Structured WPKind
Integration Design Specification
<<mandatory, output>>
Device Integration Specification
Networking Integration Specification
Middleware Integration Specification
App Service Integration Specification
Data&Semantics Integration Specification
Full Integration Design Specification
[ITERATION]
(b)

Fig. 6 The INTER-IoT-based Design phase: (a) Design activity overall description and (b) its workflow.

The work product of this phase is a set of instantiated INTER-IoT Design Patterns to be exploited for driving the implementation phase.

4.2.3 INTER-IoT-based Implementation to Maintenance Phases

The Implementation phase is the third step of our methodology; it takes in input two or more IoT platforms whose integration has been designed according to the instantiated INTER-IoT Design Patterns. The objective of this phase is to concretely integrate/interconnect the considered IoT platforms by physically implement the instantiated patterns according to successive refinement steps that involve to (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) implement, in terms of software bridges connected to INTER-LAYER, the INTER-IoT based INTER-LAYER design patterns. The final output work-product is the INTER-IoT-based Integrated Platform. At this point, the heterogeneous IoT platforms/systems are integrated according to the INTER-IoT approach.

The Deployment phase is the fourth step of the methodology and follows the Implemention phase. The objective is the deployment of the integrated and implemented platform. 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).The INTER-IoT based Deployment activity, which is the only main activity of the Deployment phase, comprises six main tasks, described in the following.The Platform Configuration task 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). To instantiate and deploy an IoT Platform Middleware in the INTER-FW to enable middleware interoperability, the following steps are followed: 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. The Gateway Configuration task 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 the Deliverable 5.1). To add a new gateway, the following steps are followed: 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. The Networking Configuration task focuses on network interoperability, achieved via network virtualization and the "Virtual Network Orchestration" pattern is configured and managed.The Application Services Configuration task includes a graphic tool for service orchestration. 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. In the Semantics

Configuration task, configurable parameters and processes related to the semantics interoperability are configured for the deployment of interoperable IoT platforms. The last task of the Deployment phase is called User management Configuration and contains configurations valid for all the previous modules; 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. The final work-product of the deployment phase is, thefore, a configured and deployment integrated IoT platform.

The fith step of our methodology is the Testing phase. 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 [29]. 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 [30]. 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 tasks: Factory Acceptance Testing (FAT) and Site Acceptance testing (SAT). 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 and deployed in the Deployment Phase. The FAT task 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. The SAT task 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. During the SAT testing process the actual deployed system is tested and proven.

Maintenance is the final phase of our methodology with the objectives to maintain and track the evolution of the integrated IoT platform during time. Maintenance is referred to the identification of a list of bugs and/or a list of evolution points at specific INTER-IoT layers and/or products and to the consequent correction of bugs and/or implementation of new functionalities. The Maintenance activity is subdivided into two main tasks. Change Identification task aims at identifying bugs and/or evolution points of the integrated platform. Change Implementation task is the actual development 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.

5 INTER-CASE

INTER-METH is supported by a CASE (Computer Aided Software Engineering) tool called INTER-CASE that helps supporting each aforementioned phase of the integration process and specifically provides the following functionalities:

- Support for workflow execution in each phase;
- Web-based Graphical facilities;
- XML-based project data repositories.

INTER-CASE is specifically intended to guide the IoT integrator in properly following and applying the integration workflow of the INTER-IOT instantiated INTER-METH. It is particularly effective to keep trace of integration choices at each phase of the methodology, so to favor and simplifying documentability of the IoT platforms integration project. In addition, it supports the reduction of inconsistencies during specification refinements from one step to the following, with warning and error messages provided to the integrator when inconsistent conditions are detected.

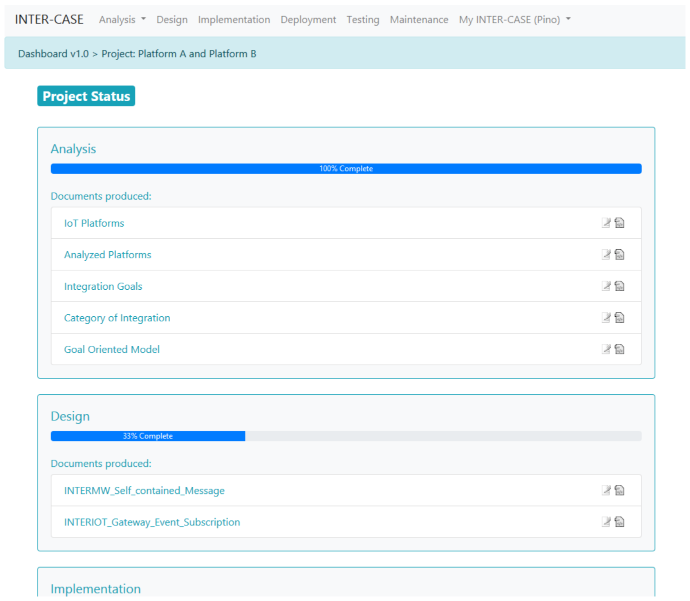
A simple, intuitive graphic user interface (GUI) characterizes the INTER-CASE Tool. It is composed of a:

- Navigation bar, with a menu that allows the integrator user to interact with the application
- Dashboard message bar, that displays the opened project
- Container, in which all the documents are presented to the user
- Footer, that contains application and copyright information

The use of the INTER-CASE Tool is allowed after authentication, by entering a username and password in a traditional-looking form. Each authenticated user can choose to open a previously saved integration project or create a new one from scratch. Obviously, the user can modify or delete an existing project.

Each project has a status page showing the integration project summary at a glance. For each phase, a card contains a list of documents produced. In every card, the Integrator user can edit or export a document by using the specific buttons placed to right of the name of every document. In case a produced document contains inconsistencies (e.g. platforms name mismatch across documents), an alert icon will be shown so to allow the Integrator with quick visualization of the issue. Figure 7 depicts an example of Project Status in which only the Analysis and Design phases are displayed.

A menu to define the activities of a given phase becomes accessible to the integrator user once all tasks related to the previous phase are complete. In the following, we describe the various INTER-CASE functionalities for each phase of the INTER-METH methodology.



INTER-CASE Analysis Design Implementation Deployment Testing Maintenance My INTER-CASE (Pino)

Dashboard v1.0 > Project: Platform A and Platform B

Project Status

Analysis

100% Complete

Documents produced:

IoT Platforms

Analyzed Platforms

Integration Goals

Category of Integration

Goal Oriented Model

Design

33% Complete

Documents produced:

INTERMW_Self_contained_Message

INTERIOT_Gateway_Event_Subscription

Implementation

Fig. 7 INTER-CASE Project Status page

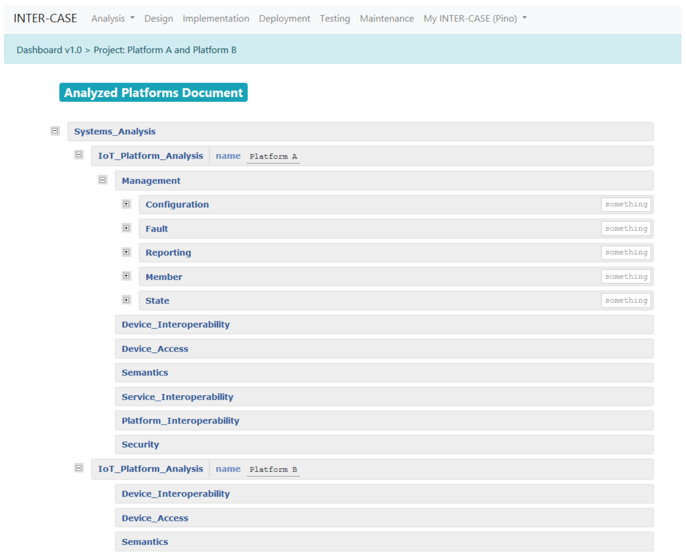
5.1 Analysis Phase

Through the Analysis menu, the analysis phase can be carried out. It is composed by five tasks. The completion of a task enables the access to the following one.

The first task is the IoT Platforms definition, where the integrator user defines generic information about the platforms to integrate, such as the platform type, the platform owner, the ontology type used.

The second task is the Analyzed Platforms Document (APD) definition. In this task the Integrator user has to analyze the platforms to integrate in terms of the INTER-IOT platform reference model described in the Deliverable D4.1. The APD, exemplified in Figure 8, is a fundamental document in the integration project because represents the basis to analyze the platforms according to a homogeneous representation, which is necessary to identify the integration points among the platforms. The platforms in this document must be obviously the one identified in the IoT Platform document, so INTER-CASE automatically fills in the Platform Name field, although the Integrator could still edit the pre-compiled information; in case of information mismatch with the IoT Platform document, however, the Tool will report a warning to the Integrator user.

The third task is the Integration Goals definition where the integrator user identifies high-level integration requirements and objectives.



Systems_Analysis
- IoT_Platform_Analysis — name: Platform A
  - Management
    - Configuration: something
    - Fault: something
    - Reporting: something
    - Member: something
    - State: something
  - Device_Interoperability
  - Device_Access
  - Semantics
  - Service_Interoperability
  - Platform_Interoperability
  - Security
- IoT_Platform_Analysis — name: Platform B
  - Device_Interoperability
  - Device_Access
  - Semantics

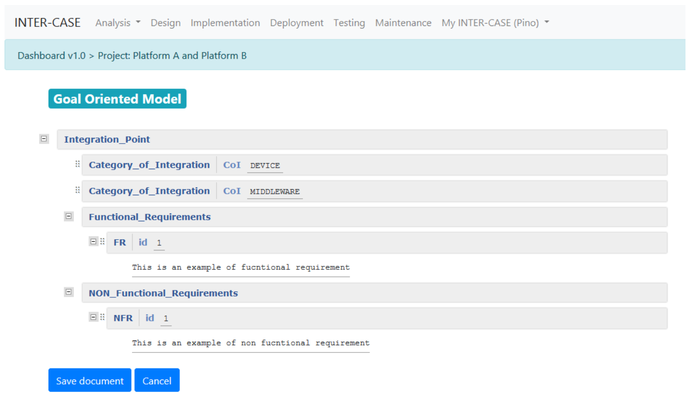
Fig. 8 Example of Analyzed Platforms Document

The fourth task is the Category of Integration definition. The integrator user identifies the integration layers, selecting them among those defined in INTER-Layer.

The last task generated the final output of the Analysis phase: the Goal Oriented Model (GOM) document. In this task the integrator user refines, in terms of functional (FR) and non-functional requirements (NFR) and according to the APD, the identified integration goals. The Goal Oriented Model document also includes the list of Category of Integration document produced in the previous step. In Figure 9 an example of the GOM document is depicted.

5.2 Design Phase

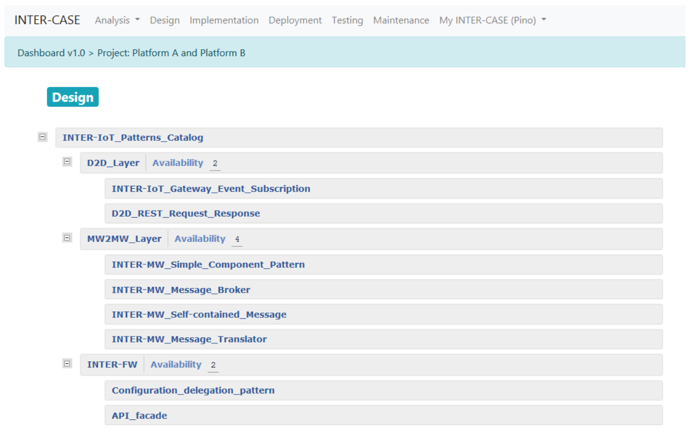
INTER-CASE enables the Design phase when the integrator completes the Analysis phase. The page showed to the user is generated by the application according to the GOMl document defined in the previous Analysis phase. Based on the identified layers of integration, the integrator user must instantiate the design patterns proposed in this phase. The instantiation of a pattern occurs by selecting the specific pattern template, and involves opening the page of the corresponding pre-instantiated pattern (see Section 3). For a complete description of each identified and pre-instantiated pattern, the interested reader can refer to the Deliverable D5.2 of the INTER-IOT



Integration_Point
Category_of_Integration CoI DEVICE
Category_of_Integration CoI MIDDLEWARE
Functional_Requirements
FR id 1
This is an example of fucntional requirement
NON_Functional_Requirements
NFR id 1
This is an example of non fucntional requirement

Fig. 9 INTER-CASE Goal Oriented Model

project. Figure 10 depicts an example of the Design document in which both middleware layer and device layer pattern are proposed.



INTER-IoT_Patterns_Catalog
D2D_Layer Availability 2
INTER-IoT_Gateway_Event_Subscription
D2D_REST_Request_Response
MW2MW_Layer Availability 4
INTER-MW_Simple_Component_Pattern
INTER-MW_Message_Broker
INTER-MW_Self-contained_Message
INTER-MW_Message_Translator
INTER-FW Availability 2
Configuration_delegation_pattern
API_facade

Fig. 10 INTER-CASE Design Phase: list of identified patterns

5.3 Implementation to Maintenance

INTER-CASE also supports the following integration phases, from Implementation to Maintenance.

In the Implementation phase the integrator user creates a document to specify the information related to the public or private repository (or repositories) of the software source code under development.

In Deployment phase, INTER-CASE requests the integrator to specify information related to integrated platform deployment in terms of configuration parameters of the INTER-FW product (see Deliverables D4.x).

The Testing phase the integrator user creates a systematic document with the relevant tests to be carried out on the integrated platform. For each defined test, the integrator has to specify the objectives, the requirement to validate (preferably also including NFRs identified in the Analysis phase), the tools necessary to execute the test, the test strategy, and of course the expected results of the test execution.

Finally, the Maintenance phase allows to create a "live" document in which the integrator inserts. from time to time, notes related to discovered bugs (including possible resolution actions) and evolution points which essentially represent future developments and functionalities of the integrated platform.

6 The INTER-Health use case: from Analysis to Design

To exemplify the use of INTER-METH for the integration of two heterogeneous IoT platforms, we will summarize the work done in the context of INTER-Health (see Deliverable D6.3), one of the two pilots developed in INTER-IOT, with particular focus on the Analysis and Design phases that are carried out and documented through our INTER-CASE tool.

The integration scenario, shown in Figure 11 involves two different IoT platforms, BodyCloud [31] and UniversAAL [32], that need to be integrated to develop the INTER-Health Pilot.

To execute the INTER-METH workflow with INTER-CASE, the integrator logs in the tool and creates a new Integration Project first. The Analysis phase starts with the definition of the two platforms: each one is characterized by name, type, ontology, and owner.

Then, IoT Platforms Analysis is carried out by representing BodyCloud and UniversAAL using the common, homogeneous model: the INTER-IoT Reference Architecture (depicted in Figure 3 and described in Deliverable D4.1). The representation of this model (called Analyzed Platform Document) is reported in Figure 12.

The next task requires to define the high-level integration goals. In particular, for INTER-Health the following goals were identified by the integrator:

- IG1: Data generated from the two platforms have to be shared and transparently accessed from both platforms;
- IG2: Common notion of the users registered to each platform;
- IG3: Subscription support from one platform to the other to be notified upon the availability of new data of a given user.

After the elicitation of the integration goals and by studying the possible integration points with the support of the Analyzed Platform Document, the

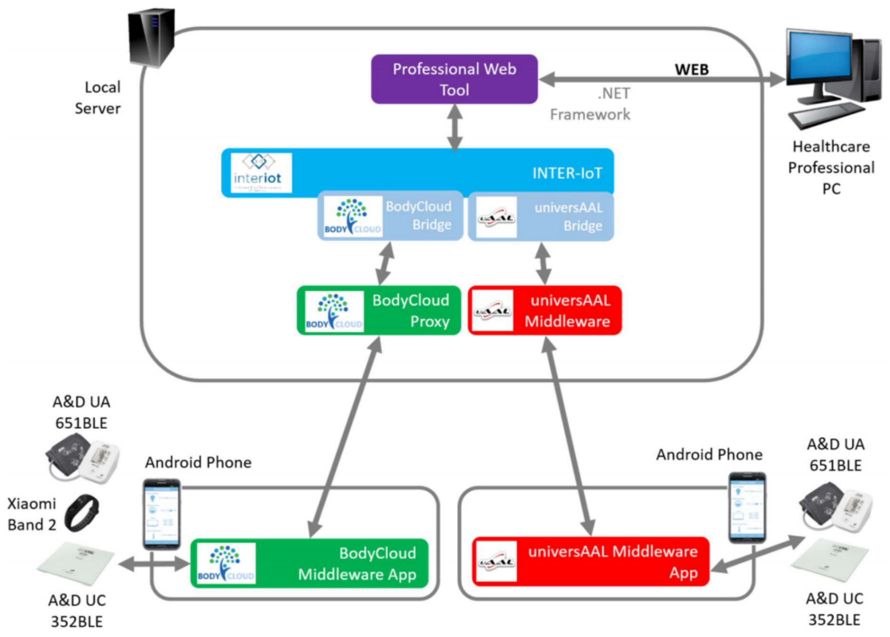
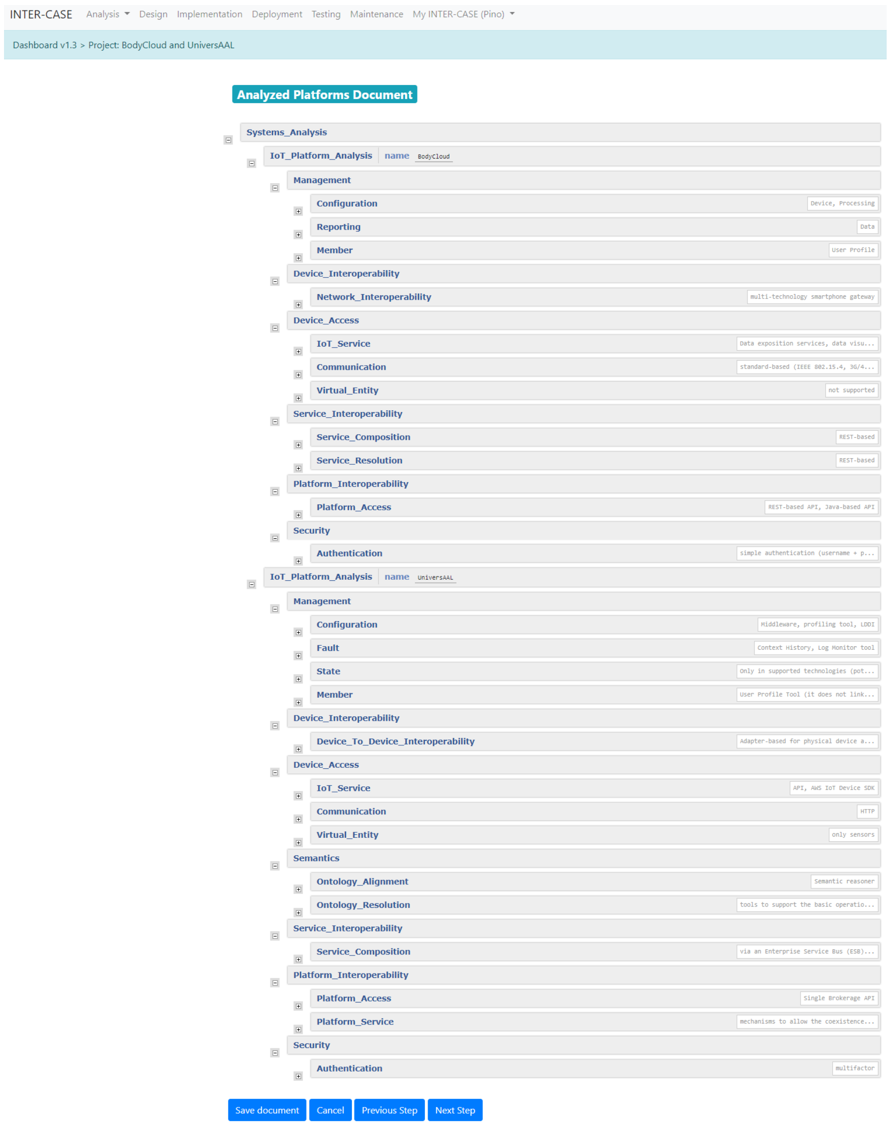


Fig. 11 INTER-Health integration scenario

integrator user made the choice to integrate BodyCloud and UniversAAL at the "MIDDLEWARE" layer among the INTER-LAYER stack.

The final task of the Analysis phase is the INTER-GOM Document production. The high-level integration goals are refined in functional (FR) and non-functional (NFR) requirements; for INTER-Health the result of such work led to the definition of the following:

- FR1: Common knowledge and sharing of users IDs
- FR2: Syntactical and semantic translation of messages (BodyCloud uses JSON while UniversAAL is based on RDF)
- FR3: Subscription management to topics (e.g. messages generated from a given user)
- FR4: User Access Gateway for Patients. The main functionalities for patients are: Access to services (providing username and password); Setting Profile communication and devices pairing; Managing measures on the device and releasing them to the gateway which stores them on a local database; Possibilities of inserting measures manually; Sending measures to the platform.
- FR5: Definition of reference meaning for health information. Health information can be detected using different devices according to different way of measurement (unit of measure that could differ from country to country and also depending on devices manufacturers). To use same information



```
Systems_Analysis
  IoT_Platform_Analysis  name  BodyCloud
    Management
      Configuration                      Device, Processing
      Reporting                          Data
      Member                             User Profile
    Device_Interoperability
      Network_Interoperability           multi-technology smartphone gateway
    Device_Access
      IoT_Service                        Data exposition services, data visu...
      Communication                      standard-based (IEEE 802.15.4, 3G/4...
      Virtual_Entity                     not supported
    Service_Interoperability
      Service_Composition                REST-based
      Service_Resolution                 REST-based
    Platform_Interoperability
      Platform_Access                    REST-based API, Java-based API
    Security
      Authentication                     simple authentication (username + p...
  IoT_Platform_Analysis  name  UniversAAL
    Management
      Configuration                      Middleware, profiling tool, LDDI
      Fault                              Context History, Log Monitor tool
      State                              Only in supported technologies (pot...
      Member                             User Profile Tool (it does not link...
    Device_Interoperability
      Device_To_Device_Interoperability  Adapter-based for physical device a...
    Device_Access
      IoT_Service                        API, AWS IoT Device SDK
      Communication                      HTTP
      Virtual_Entity                     only sensors
    Semantics
      Ontology_Alignment                 Semantic reasoner
      Ontology_Resolution                tools to support the basic operatio...
    Service_Interoperability
      Service_Composition                via an Enterprise Service Bus (ESB)...
    Platform_Interoperability
      Platform_Access                    Single Brokerage API
      Platform_Service                   mechanisms to allow the coexistence...
    Security
      Authentication                     multifactor
```

Fig. 12 INTER-Health integration: IoT Platforms Analysis.

coming from different systems and going to others, it is mandatory to establish specific criteria to: (i) define a common meaning if it is possible, (ii) determine a correspondence between different data that have the same meaning and different values, (iii) set transcoding tables between different values of the same data.

- NFR1: Application response time. The "navigation" functionalities on different contents by using both Smartphone or Personal Computer to access to the platform, have to guarantee a response time of a few seconds.
- NFR2: Availability of sensor data. Health monitoring data must be accessible from a remote location to facilitate patient triage and inform decision making.
- NFR3: User Authentication to access INTER-Health services. Users shall authenticate to the services using their username and password

The design phase of the INTER-Health integration process involves the requirements elicited in the INTER-GOM model. The integrator instantiates a set of INTER-IoT Design Patterns (see Section 3 and Deliverable D5.2) related to the integration of BodyCloud and UniversAAL platforms. The first step is the choice of the necessary pre-instantiated design patterns, among the available ones that are automatically filtered out by the INTER-CASE tool according to the input from the Analysis phase.

Given the choice of make the two platforms interoperable at Middleware layer, the integration design patterns that are selected belongs to the INTER-Middleware category. Specifically, three patterns will drive the implementation phase. *INTER-MW Message Broker* template is instantiated as follows:

- *Intent*: A component that facilitates passing of messages between decoupled INTER-Health components.
- *Problem and Solution*: Having point-to-point communication interfaces among several interacting components, makes dynamic reconfiguration, matching of different security constraints and QoS, requirements between components difficult. The employment of a message broker, can help to overcome limitations of point-to-point connection and enforce a common messaging interface upon different middleware components.
- *Applicability* : Each Inter-Health component communicates directly only with the message server broker which in turn handles the communications and dispatches messages to the respective destination component.
- *Implementation*: There is a message broker service that analyzes each received messages to check is there exists a subscription for the user that sent the message. If a subscription exists, the broker obtains (from the subscription message initially received) the necessary information to send the current message. If the subscription does not exist, the message is not forwarded to the external middleware components.

INTER-MW Message Translator template is instantiated as follows:

- *Intent*: Syntactical and semantic translation of messages between BodyCloud and UniversAAL.

- *Problem and Solution*: BodyCloud and UniversAAL use different message formats and models, respectively based on JSON and RDF, so direct communication between the two platforms is not possible. An effective solution is to introduce a translator component in the middle that is able to understand both message structures so to translate from the source platform format/model to the destination one.
- *Implementation*: Message translation is actually composed of two steps. First, there is need for syntactic translation to/from proprietary message format which is done in the bridge; the second step is semantic translation of the message and it is done in the Inter Platform Semantic Mediator (IPSM) component of Inter-IoT.

INTER-MW Self-contained Message template is instantiated as follows:

- *Intent*: Each message contains all the information needed to subscribe or unsubscribe to a given topic (or conversation).
- *Problem and Solution*: BodyCloud and UniversAAL need to communicate. To do so, it is necessary to know where one platform has to send the messages to the other platform. Hence, the identified solution is the creation of a subscription mechanism to conversations.
- *Implementation*: A bridge component of INTER-MW will process the messages to perform the proper action (i.e. subscription or subscription) . The bridges then registers a corresponding callback to the action, in case of subscription message.

7 Conclusions

Interoperability among heterogeneous IoT platforms is a complex multi-faced issue which requires effective approaches that are difficult to implement and test. Indeed, it is not straightforward defining boundaries and requirements of the IoT platforms to be integrated as well as controlling the development environment. Traditional software/systems engineering approaches showed poor applicability in the IoT domain and, therefore, ad-hoc and closed integration/interoperability solutions are often adopted. INTER-IOT aimed at addressing the lack of systematic, generalizable methods and tools for IoT platforms interoperability. This chapter, in particular, described INTER-METH, developed in the project, that is first full-fledged, general-purpose engineering methodology completely supporting the integration process among IoT platforms. Particular focus has been given to the INTER-IoT-based Analysis and Design phases, that are fundamental drivers for the integration process. Relevant interoperability design patterns, the building blocks of the design phase, have been discussed.The chapter also presented INTER-CASE, a web-based tool that guides integrator designers to follow the methodology; INTER-CASE supports semi-automatic integration refinements and particularly useful to easily document the choices taken during the integration workflow. The practical use case related to the integration of the two IoT platforms (Body-Cloud and UniversAAL) adopted in the context of the INTER-Health Pilot, is finally shown.

References

1. C. Savaglio, G., Fortino, M., Zhou, "Towards interoperable, cognitive and autonomic IoT systems: An agent-based approach," in: Internet of Things, 2016 IEEE 3rd World Forum on. IEEE, pp. 58-63, 2016.
2. G. Fortino, et al., "Towards Multi-layer Interoperability of Heterogeneous IoT Platforms: The INTER-IoT Approach," Integration, Interconnection, and Interoperability of IoT Systems. Springer, 199-232, 2018.
3. G. Fortino, W. Russo, C. Savaglio, W. Shen and M. Zhou, "Agent-Oriented Cooperative Smart Objects: From IoT System Design to Implementation," in IEEE Transactions on Systems, Man, and Cybernetics: Systems. vol. 48, no. 11, pp. 1939-1956, 2018.
4. A. Bassi, et al., "Enabling Things to Talk: Designing IoT solutions with the IoT Architectural Reference Model," 2013.
5. P. Houser, "Best Practices for Systems Integration" Engineering & Product Excellence, Northrop Grumman Corporation, November 2011.
6. OUSD AT&L., " Systems Engineering Guide for Systems of Systems," Washington, D.C.: Pentagon, August 2008.
7. W. Vaneman, "The system of systems engineering and integration 'Vee' model," Systems Conference (SysCon), 2016 Annual IEEE. IEEE, 2016.
8. K. Gama,L. Touseau, D. Donsez, "Combining heterogeneous service technologies for building an Internet of Things middleware," Computer Communications 35.4 (2012): 405-417.
9. G. Aloi, et al., "Enabling IoT interoperability through opportunistic smartphone-based mobile gateways," Journal of Network and Computer Applications, 81 (2017): 74-84.
10. I. Ishaq, et al., "Internet of things virtual networks: Bringing network virtualization to resource-constrained devices," Green Computing and Communications (GreenCom), 2012 IEEE Intl. Conf. on. IEEE, 2012.
11. Waterfall methodology: https://en.wikipedia.org/wiki/Waterfall_model, last accessed May 2019.
12. Robertson, Suzanne, and James Robertson. "Mastering the requirements process: Getting requirements right". Addison-wesley, 2012.
13. 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.
14. 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.
15. 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.
16. ELDATool, documentation and software, http://lisdip.deis.unical.it/software/eldatool, last accessed May 2019.

17. 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.
18. 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.
19. 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.
20. 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/, last accessed May 2019.
21. Bassi, et al., "Enabling things to talk", Springer, 2013.
22. http://docs.oasis-open.org/soa-rm/RS12.0/soa-rm.pdf, last accessed May 2019.
23. Zambonelli, Franco. "Towards a General Software Engineering Methodology for the Internet of Things." arXiv preprint arXiv:1601.05569 (2016).
24. The IoT-A Unified Requirements list, "http://www.iot-a.eu/public/requirements/copy_of_requirements", last accessed May 2019.
25. 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.
26. Grace, Paul, Brian Pickering, and Mike Surridge. "Model-driven interoperability: engineering heterogeneous IoT systems." Annals of Telecommunications 71.3-4 (2016): 141-150.
27. Kazman, Rick, et al. "Understanding patterns for system of systems integration." System of Systems Engineering (SoSE), 2013 8th International Conference on. IEEE, 2013.
28. Report on High-Level Architecture (HLA) http://ec.europa.eu/newsroom/dae/document.cfm?action=display&doc_id=11812, last accessed May 2019.
29. Acceptance Testing, [Online]. Available: https://en.wikipedia.org/wiki/Acceptance_testing, last accessed May 2019.
30. 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, last accessed May 2019.
31. G. Fortino, D. Parisi, V. Pirrone, G. Di Fatta, BodyCloud: A SaaS Approach for Community Body Sensor Networks, Future Generation Computer Systems, vol. 35, n. 6, pp. 62-79, 2014.
32. UniversAAL website, https://www.universaal.info, last accessed May 2019.
33. Giancarlo Fortino, Wilma Russo, Eugenio Zimeo: A statecharts-based software development process for mobile agents. Inf. Softw. Technol. 46(13): 907-921 (2004).
34. Giancarlo Fortino, Alfredo Garro, Wilma Russo:An integrated approach for the development and validation of multi-agent systems. Comput. Syst. Sci. Eng. 20(4) (2005).
35. Giancarlo Fortino, Claudio Savaglio, Giandomenico Spezzano, MengChu Zhou:Internet of Things as System of Systems: A Review of Methodologies, Frameworks, Platforms, and Tools. IEEE Trans. Syst. Man Cybern. Syst. 51(1): 223-236 (2021).
36. Giancarlo Fortino, Wilma Russo, Claudio Savaglio, Weiming Shen, Mengchu Zhou: Agent-Oriented Cooperative Smart Objects: From IoT System Design to Implementation. IEEE Trans. Syst. Man Cybern. Syst. 48(11): 1939-1956 (2018).
37. Claudio Savaglio, Maria Ganzha, Marcin Paprzycki, Costin Badica, Mirjana Ivanovic, Giancarlo Fortino: Agent-based Internet of Things: State-of-the-art and research challenges. Future Gener. Comput. Syst. 102: 1038-1053 (2020).

38. Giancarlo Fortino, Alfredo Garro, Wilma Russo: Achieving Mobile Agent Systems interoperability through software layering. Inf. Softw. Technol. 50(4): 322-341 (2008).
39. Massimo Cossentino, Giancarlo Fortino, Alfredo Garro, Samuele Mascillaro, Wilma Russo: PASSIM: a simulation-based process for the development of multi-agent systems. Int. J. Agent Oriented Softw. Eng. 2(2): 132-170 (2008).