Integration on Internet of Things: A Semantic Middleware Approach to Integrate Heterogeneous Technologies Seamlessly

Alex C. Olivieri[a], Gianluca Rizzo[b] , Yann Bocchi [c]
[a]*HES-SO, E-mail: alex.olivieri@hevs.ch*
[b]*HES-SO, E-mail: gianluca.rizzo@hevs.ch*
[c]*HES-SO, E-mail: yann.bocchi@hevs.ch*

Summary

The advent of the Internet of Things (IoT) opens a wide spectrum of interesting opportunities for the creation of novel scenarios. By putting in communication devices such as sensors and actuators, in conjunction with mainstream technologies, IoT enables a whole set of new services in almost any domain to be provided, from building automation to Smart City, E-Health and assistive technologies, only to name a few.

Nevertheless, the integration of different protocols and technologies remain to date one of the main challenges in IoT. The proposed solutions have evolved following the heterogeneity increment of the technologies employed for complex scenarios, passing from ad-hoc solutions on a per-connection basis (*N-to-N*), to solutions that use frameworks as middleware (*N-to-1*). *N-to-1* solutions that have shown interesting features use frameworks modeled on the *Publish/Subscribe* (P/S) design pattern. This type of frameworks provides facilities to manage the exchange of communications in presence of large number of heterogeneous communication technologies. However, how to efficiently interface those technologies to a common framework still remains an open issue.

In this work we propose to surround a P/S framework with a layer that allows us to easily interface the various communication technologies with the framework. We describe how to model this layer, and we show how to implement simple adaptors that can be used to interface a technology to the selected P/S framework. We show how our approach makes the integration of heterogeneous technologies easier and how it enables scenarios' reconfigurations even at run time. By using such an approach, designers can focus on the business logic of the scenario rather than on the low level technological details of the integration. We inform the reader about the lessons learned throughout the realization of the proposed approach and about the tasks that we found more challenging in making the final system; by doing so we believe to inculcate into the reader some questions that could lead to some interesting future research in order to address some challenges that remain still open.

1 Introduction

The integration of different technologies has always created great possibilities for the implementation of novel services and the creation of new interesting scenarios. In cyber-physical systems for example, to integrate more and more heterogeneous technologies means to increase the set of available options for their design and implementation. It brings the possibility to exploit the advantages of each technology while offering new options for overcoming their limitations. This perspective is even more fascinating nowadays with the advent of the IoT, which extends the set of available interacting technologies to devices with low computational power.

The evolution of the integration techniques goes from the *N-to-N* approach, where the connections are point-to-point (adopted in pure Machine-to-Machine (M2M) settings, usually with little technological heterogeneity), to the *N-to-1* approach, which typically relies on some form of middleware for the management of all communications.

A predominant type of frameworks employed in this latter approach is based on the publish-subscribe (P/S) design pattern. Such design pattern adopts *decoupling* as fundamental paradigm when interconnecting interacting entities. This decoupling feature allows the framework to create the interconnection among entities based on the data they want to exchange rather than on direct point-to-point interconnection among those entities. These frameworks enable the integration of a large number of different technologies, and they provide good scalability features in context where the load of communications is not too heavy and some latency is accepted. Examples of these situations are web feed systems such as RSS and Atom (Liu et al. 2005).

However, the P/S suitability described above can drop in presence of systems that demand exceptional data loading or real-time systems that demand zero-latency. For dealing with such conditions, there exist advanced P/S architectures that address the limitations of typical P/S systems (Schmidt and ORyan 2003). For example Data Distribution Service (DDS) (Pardo-Castellote 2003) [1] is a standard that aims to enable scalable, real-time, dependable, high-performance and interoperable data exchanges between publishers and subscribers. To overcome these limitations these P/S systems employ P2P communication models (Li et al. 2003). DDS addresses the needs of applications like financial trading, air-traffic control, smart grid management, and other big data applications.

The presence of research aiming to create new solutions based on the P/S paradigm for the IoT, such as MQTT protocol (Hunkeler et al. 2008), reinforces our idea that this type of framework is the suitable one for the IoT domain (Sanchez 2012).

Even though frameworks based on the P/S design pattern, both industrial solutions, such as Xively [2], and academic solutions, such as OpenIoT (Soldatos et al. 2015), show appreciable features for specific domains focused mainly on low computational power devices, they still have two remarkable limitations when dealing with technologies usually employed outside those domains. The first one concerns the capability to easily interconnect the entities with the chosen framework. This is due to the absence of standards that define how these interconnections should be designed. This situation leads to the creation of an ad hoc integration each time the need to interconnect a new entity arises. The second one comes as a consequence of the former, and it concerns the limitations of these framework in addressing the reconfiguration and adaptation of the system when changes on scenarios happen at runtime. The capability of the system to adapt itself to changes is important in the IoT domain, because dynamism and variability are its inner features. Addressing these issues would help provide middleware for the development and management of many of the complex scenarios arising in the IoT.

[1] http://www.omg.org/spec/DDS/1.4/PDF/
[2] https://xively.com/

In this work we propose a stack composed by *de facto* standard solutions as a layer which surrounds a selected P/S framework. This layer greatly facilitates the interconnection of different entities through the framework, regardless which technology they belong to. Moreover, we modelled and deployed the system in such a way that allows the scenarios to be reconfigured at runtime giving the possibility to enable the system to adapt to different situations over time. We present a sample implementation of our solution, and assess it in a motivating scenario.

This work is organized as follows: section 2 introduces a motivating scenario that shows the need of integration, and to which we will refer throughout the article; section 3 provides the state of the art of the approaches for integration, explaining how the main integration challenges have been addressed over time, and highlighting the main open issues. Section 4 presents the main components of our integration layer and the structure of the selected P/S framework. 5 explains how to develop and deploy the adaptors needed to connect the technologies to that framework and it shows how our system supports the configuration of new scenarios in a easy way. Finally, section 6 concludes the chapter testifying to the reader the contribution of our approach in solving the integration problem and it points out some open issues that require further investigation.

2 Motivating Scenario

In this section we introduce a futuristic example where heterogeneous technologies can be connected together effortlessly in order to perform an interesting scenario. This example will be recalled throughout the work and it will be used to make the reader comfortable with the concepts explained.

Jack and John are two friends spending together an afternoon studying at Jack's place. Feeling not really motivated in studying, they decide to play their favourite sport videogame, American Football (NFL), with the Playstation [3]. Given their passion for this sport, they decide also to simulate, within the living room, the real conditions (climatic, environmental etc) of every game they will play. To do so they connect the Playstation to the air conditioning system and to the humidifier present in the living room. This connection will allow the air conditioning and the humidifier to receive the needed pieces of information from the Playstation in order to adapt the living room's environment to the conditions of the game. Since John is a fan of the Green Bay Packers (a team based in Wisconsin), and he chooses it as his team, the home games could be really freezing. For this reason they decide that it would be a good idea to have hot tea when the half time pause will come. To do so they connect the kettle and the Playstation, so that the Playstation will update the kettle about the timing of the game, allowing to be switched on 2 minutes before the pause in order to boil water for the tea. Since Jack's girlfriend Lisa is having an interview for an important job and she said that she would phoned him once finished, Jack decides to connect his smart phone with the Playstation in order to pause the game automatically in case Lisa calls.

Now that everything seems well planned, the two friends start the first match. Suddenly however, they realize that Jack's mother, Marta, is in the basement doing her home chores, and that she would get really upset if she caught them playing. So they decide to place a Kinect device [4] into the corridor just outside the basement, in order to detect Marta when she approaches. They connect the Kinect to the Playstation and the TV in order to switch them off in case Marta shows

[3]https://www.playstation.com/en-us/
[4]https://www.microsoft.com/en-us/kinectforwindows/

up. At last, they feel free to start playing.

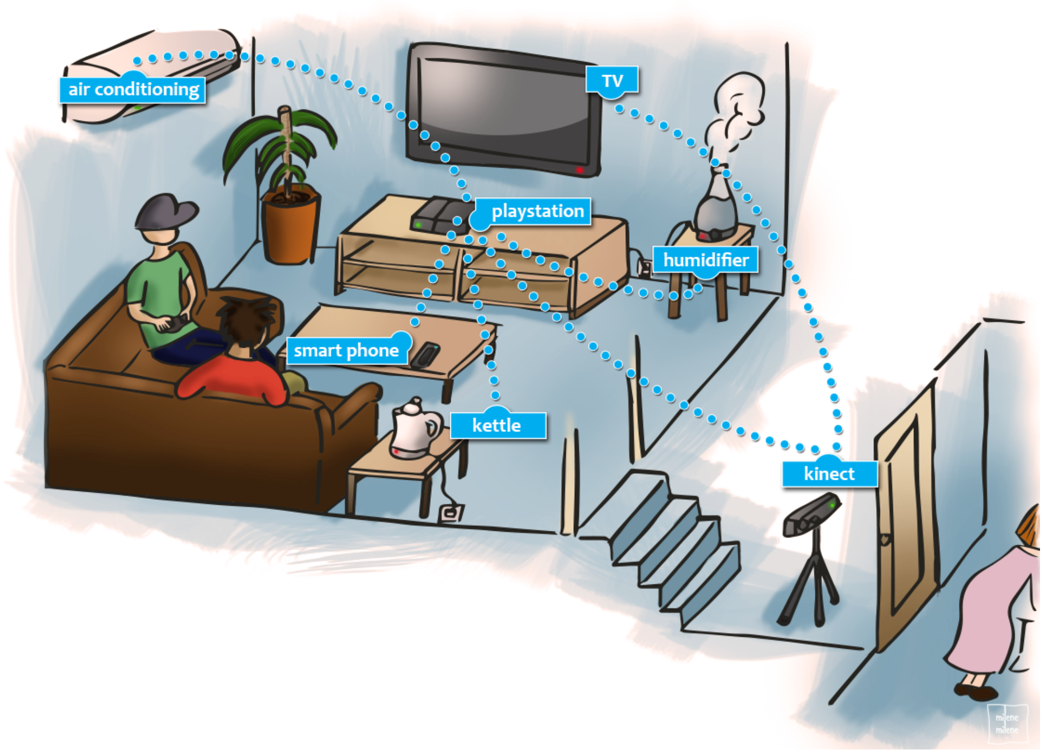


Figure .1: Motivating Scenario

After some hours they finish playing, and even if one of them - the loser - is not happy at all, they really enjoyed the afternoon. The games the two friends played were joyful because they planned everything in order to being able to focus their attention only on their hobby. This was possible only because Jack's house is equipped with the necessary technological devices and due to the possibility to establish, almost effortlessly, the interactions among all the devices employed in the scenario. In the scenario, it seems that in order to implement the desired interactions, the user just needs to wire one device with another as he wishes to create the desired interconnection, treating every entity like a black box.

Nevertheless, nowadays the implementation of such a system, composed of such diverse technologies, and capable of changing configuration at runtime, is very challenging. Even to an inexperienced person of the integration domain can be clear that there are some challenges that have to be faced. For example to connect two technologies means to create a sort of bridge between them in order to make them interoperable, and this work requires a profound knowledge of all technologies employed. Moreover, to establish every point to point connection requires time and the appropriate tools, besides the understanding of what information should be exchanged to design the scenario. These issues make it difficult to model scenarios that really focus on what matters - the business logic of the use case the user is interested in. For business logic we intend all events and actions that can occur during scenarios' execution.

We assume that once the technologies are connected and they can communicate, they are able to perform the business logic designed for the scenario automatically.

3 Current Approaches to Integration in IoT

In this section we analyze in detail the motivating scenario in order to discover and formally define the challenges that the integration poses in the IoT domain. Then we describe the main approaches through which some of these challenges have been addressed, starting from the M2M approaches and finishing with the novel approaches based on frameworks. Finally we describe in detail the frameworks based on P/S design pattern approach, describing all aspects which make them particularly suitable to address the main challenges of IoT integration in comparison with other integration solutions proposed for the same domain.

3.1 Understanding the Integration Issues

Integration in information systems is a *"process or phase concerned with joining different subsystems or components as one large system. It ensures that each integrated subsystem functions as required. System integration (SI) is also used to add value to a system through new functionalities provided by connecting functions of different systems".* [5]

In this section we analyze the integration problem and we understand the main challenges it brings to the IoT domain. To integrate heterogeneous technologies is a problematic task, and to achieve a solution for it requires understanding of what are the main chunks that compose it. To this end, we point out the scenarios the motivating scenario performs and then we apply software engineering techniques to define and analyze the functionalities those chunks imply and then we analyze them.

Firstly, let us define Jack's house as our *ecosystem*. This ecosystem is composed of all technologies and actors that compose our motivating scenario. The motivating scenario shows that to perform the desired behavior we must connect the devices that will communicate between them. Each device can communicate with one or more devices at a time. As a consequence, the number of connections it can be subjected to range from one to an undefined value. The motivating scenario also shows that sometimes there is the need or wish to configure some changes to the scenario at runtime.

If we follow the software engineering theory (Lee 2013) and we compare the above description to the *requirement's specification*, we can apply a tactic that is used to model a component diagram starting by the **nouns** and the **verbs** present in the specification. For the requirement's specification we can highlight the following four meaningful verbs, and deliberate that they are the main challenges that we must face if we want to be able to perform the motivating scenario.

- To Connect
- To Communicate
- To Range
- To Configure

These challenges pose issues that we must address if we want to obtain the integration we need in order to perform not only the use case proposed in the motivating scenario, but even more complex

[5]http://www.techopedia.com/definition/9614/system-integration-si

use cases which can be performed in various situations. In the following sections we analyze such issues in detail.

3.1.1 To Connect - The Connection Problem

In the motivating scenario we often say that Jack and John connect a device to another device; but what does to connect exactly mean to us? When we say connect we refer to the capability of a device to send something to another device. For the moment we are not interested on what is sent, we just want that a device, once connected to another device, is capable to reach it in order to eventually send something.

3.1.2 To Communicate - The Understanding Problem

To establish connections between devices is only a first step. What utility has a connection if something a device sends is alien to the device that receives it. The understanding problem refers to the issue of a mutual understanding of what is exchanged. The receiving device must be able to interpret what the sending device provides to it.

3.1.3 To Range - The Scalability Problem

The motivating scenario contains some devices that are connected only with a device, and others that are connected with more devices. This indicates that it is not possible to know in advance with how many devices each device will be connected to. As a consequence we can assert that every device must be capable, at least in theory, to connect to a range of devices, with no predefined lower or upper bound.

3.1.4 To Configure - The Adaptation Problem

The motivating scenario shows marginally that sometimes there can be a need of changes at runtime. For example Jack and John decide to use the kinect to control Jack's mother movements only at a later time, and not since the beginning. If it would not be possible to configure and to adapt the scenario afterwards, it would be impossible to satisfy the requirements that can come at a later time. Furthermore, if those changes cannot be configured at runtime, the system should be stopped every time a change is requested in order to reconfigure it and to allow it also to manage new situations.

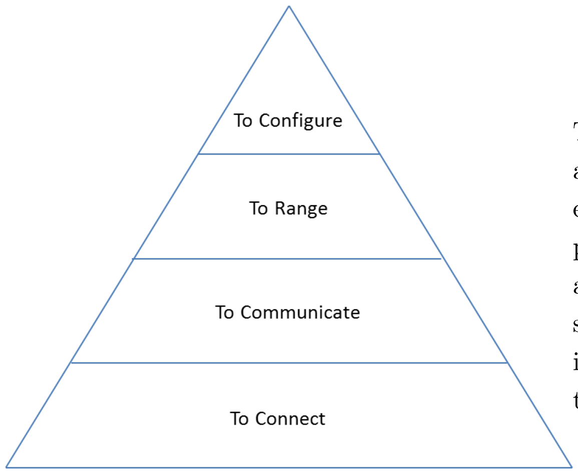


Figure .2: Integration Pyramid

The issues explained in the previous four sections are interrelated, and the order on which they are explained is not random. Without solving the previous, a solution for the following cannot be achieved. Figure .2 highlights this concept by showing that before solving an upper layer of the integration pyramid there is the need of building the lower layers on which it relies.

Now that we have clearly defined the set of challenges that the integration brings and we have described the correlations among these challenges, in the next subsections we will unfold the approaches that have been used to address the integration so far. We will describe which challenges are solved by each approach and we will indicate which challenges are still not solved.

3.2 Approaches for Integration: the History

To integrate heterogeneous technologies has always been a complex task due to the differences that protocols have at different levels (Zorzi et al. 2010). The recent past has seen different approaches proposed to address this issue. These approaches were designed for specific application domains, and their features were really dependent on the specific domain.

One of the first domain where the integration was faced and addressed was the M2M domain. M2M interaction Watson et al. (2004) refers to the information exchange among devices without human assistance. Formerly in the M2M domain the habit was to employ devices belonging to only one technology. In this case the challenges to be faced were mainly related to the connection problem and in a small portion to the understanding problem. The owners of technologies usually provided some ad-hoc solutions that gave the devices the capability to connect and communicate.

When the M2M started to go towards the deployment of scenarios composed by more technologies this approach was no longer effective, because the creation of ad-hoc solutions between each couple of technologies became quickly infeasible (Jung et al. 2012). The next approach used to address the integration was based on the adoption of the so called Universal Gateways (Latvakoski et al. 2014). This approach uses some ideas from the M2M interaction to solve the connection problem, and aimed mainly to solve the understanding problem. The reason for using such gateways is to address the ineffectiveness of ad-hoc solutions in resolving the understanding challenge. A universal gateway is a software or hardware component that provides an internal semantics that acts as a bridge between the semantics adopted by the different technologies and so allowing devices connected to it to exchange data and to understand the pieces of information exchanged. Depending on the real implementation, some gateways can even provide features that address some aspects of the scalability and configuration problem, but only for small context on which they were designed for.

The advent of the IoT brought the need of integration to a new level (Vermesan and Friess 2013). If we consider everything as a machine, starting from simple Building Automation System (BAS) devices arriving to the applications running on Personal Computers (PC), from an integration point of view the IoT can be seen as an extension of the M2M concept. The absence of standardized M2M platforms that would enabled interconnection of heterogeneous technologies (Wu et al. 2011) and the fact that the small devices (*things*) still remain the majority of the things involved in the IoT paradigm, led to the need of finding appropriate solutions for the interoperability. The IoT changed the world of the M2M paradigm, in fact through it, it is now possible to compare everything connected to the internet to a machine that can collaborate to create human-independent systems. To fulfil the integration demanded by this new M2M over IoT paradigm the efforts have been directed towards the identification of an architecure for IoT integration, though specific to some application domains [6] [7]. These architectures aim to provide platforms that allow components to interact in order to take part of scenarios. The integration provided by these solutions is based on two approaches: N-to-N approach and N-to-1 approach (Rao 2013). The N-to-N approach consists in creating a sort of bidirectional interpreter between each couple of technologies in order to make the exchange of information possible. The N-to-1 approach instead is based on the idea of providing a framework (1-component) that acts as integrator for the various technologies that take part in the system (N-components); the components need to be able to communicate with the the common framework.

The N-to-N approach showed to be inappropriate for the IoT domain because it is only suitable for scenarios that comprehend a small amount of heterogeneous technologies (Olivieri and Rizzo 2015). IoT itself aims to use all possible technologies, but this implies to have a grand heterogeneity. Therefore it does not address the scalability problem. Conversely the N-to-1 approach (Isenor and Spears 2007) seems to be the correct solution because it provides all features needed to comply to the scalability property and heterogeneity variety the IoT domain requires.

However, problems arise when the scenarios comprehend a big number of different technologies and they demand high dynamism. This is due the fact that the current solutions based on the N-to-1 approach do not provide the capabilities to easily design scenarios, and even if they tend to ensure the communications, but they do not provide the quality of service some interesting scenarios require. Moreover they let the adaptation problem a challenge to be still addressed.

Different paradigms stand behind the frameworks that act as 1-component in the N-to-1 approaches. For the features proposed by each of these paradigms we believe that the approaches based on the P/S design pattern overcome the others. The reason of our assumption is dictated by its inner quality of addressing the connection problem basing it on the data exchanged rather than on the instantiation of point to point connections between interacting components. In the next subsection we detail why we have this belief.

3.3 Frameworks based on Publish/Subscribe

The P/S framework is based on the homonymous design pattern and it defines a messaging strategy where the senders of messages (the publishers) do not define directly to which receivers (subscribers)

[6]http://www.iot-a.eu/public

[7]http://vital-iot.eu/

the messages will be sent. This strategy is incorporated within a middleware that also provides the features to manage the communications (Eugster et al.). In order to communicate, the interacting entities (publishers and subscribers) must be able to connect themselves to the P/S framework and then to express which information they provide or desire. The information exchange starts when the coupling between sources and destinations of data is established. To do so the framework couples the pieces of information made available by the publisher with the pieces of information requested by subscribers.

To establish the couplings two main matching strategies exist: Topic-based and Context-based. In the topic-based strategy, messages are published to "topics" or named logical channels. Subscribers in a topic-based system will receive all messages published about the topics to which they subscribed, and all subscribers of a certain topic will receive the same messages. The publisher is responsible for defining the topics of messages to which subscribers can subscribe. In the content-based strategy, messages are only delivered to a subscriber if the attributes or content of those messages match constraints defined by the subscriber. The subscriber is responsible for classifying the messages. Some systems support a hybrid of the two strategies; publishers post messages to a topic while subscribers register content-based subscriptions to one or more topics.

The P/S paradigm provides features that fit interaction schemes with large-scale settings, where strong and loose decoupling is required, by abstracting the interacting entities from communication issues. It provides three different kinds of decoupling:

- Space Decoupling: The interacting entities do not need to know the spatial position of the entities they will interact with;
- Time Decoupling: The interacting entities do not need to be actively participating in the interactions at the same time (i.e., if the subscriber entities are off-line when some interesting data is provided, they will receive it anyway when they will be on-line again);
- Synchronization Decoupling: Publishers are not kept waiting while they are producing events and subscribers can be asynchronously notified of the occurrence of an event while they are performing some concurrent activities.

We believe that the P/S design pattern addresses the challenges express in 3.1 in a quite exhaustive manner. In fact it defines how the interacting entities can be connected through a P/S framework and thanks to the time decoupling the publishers do not have to care about the presence of the subscribers when they publish new contents. Moreover it defines how the communications happen once the entities are connected to the framework, because both publishers and subscribers must be able to understand the semantics adopted by the framework. Furthermore it allows the system to range thanks to the space decoupling that hides the addressing information details of the subscribers to the publishers. Further, the three features above mentioned, together with the focus of the framework on data rather than on communications, address a part of the adaptation challenge because they let publishers and subscribers free to enter and exit from the scenario.

The previous reasons contributed to our decision to adopt a framework based on the P/S paradigm when we were requested from the committee of the European Project IoT6 [8] to find a solution to integrate, within the context of IoT, BAS devices and mainstream technologies. For

[8]http://iot6.eu/

mainstream technologies we intend technologies of disparate categories but a large usage, such as personal computers, playstation, mobile phones, etc. The framework that we selected is called *P/S Context Broker - Orion Context Broker* (Orion Context Broker) [9] developed for the FI-WARE European project [10].

3.3.1 Challenges and Open Issues

The P/S frameworks properly address some part of the challenges envisaged, as described in 3.3, however they do not achieve a full integration. As Oracle explains in its white paper *A Brave New Integration World* [11], it is important mainly now that we are moving more and more towards a mobile world, to create solution based on Service Oriented Architecture (SOA) paradigm in order to make systems independent from any vendor or technologies. SOA (Wiederhold 1992) is an architecture for building business applications as a set of loosely coupled black-box components orchestrated to deliver a well-defined level of service by linking together business processes. To obtain this independence, which would mean to reach the integration level desired, the P/S framework should be surrounded by a adaptation layer to provide a standardized access to it. For standardized we mean a layer composed by technologies and/or methodologies that are of common usage nowadays and that can be seen as standard de facto.

If we recall the motivating scenario, there are devices from different technologies that play different roles in it. If the framework that Jack decides to use in his house does not use standardized solutions, each vendor should learn how to interface its devices to the framework, or to hope that some developers will create some piece of code needed to integrate such devices to the framework. If a framework does not provide open and standard interfaces to external systems a two sided problem is born: mainstream technologies vendors (such as Sony) are not interested to interface their products to close environments; developers can find not stimulating to study and learn the details about how to interconnect a device to a close environment. Said this, if the framework Jack uses is a close framework, he could have any technologies he wants, but he will never be able to model the scenario he has in mind. Therefore it is indispensable to equip the frameworks (in our case a P/S framework) of the interfaces needed to easily exploit their features.

Moreover , what if the framework does not allow runtime reconfiguration of the system? This situation would oblige the users to stop it every time a scenario requires changes. As a consequence there will be a pause on the information exchange that maybe is acceptable for some scenarios, but it will be not for sure in others, making it inappropriate for these latter scenarios. In the next section we will describe how we modelled the integration layer and how we connected it to the chosen P/S framework in order to provide standard and user-friendly interfaces for external systems.

4 Design of an Integration Layer for P/S Frameworks

In section 3.3 we mentioned the need for solutions, based on standard and user-friendly interfaces, to address the integration problem in heterogeneous environments. This is a real motivation due

[9] http://catalogue.fiware.org/enablers/publishsubscribe-context-broker-orion-context-broker
[10] https://www.fiware.org/
[11] http://www.oracle.com/us/products/middleware/soa/soa-simplify-enterprise-integration-2209678.pdf

to the desire to make frameworks available to different vendors efficiently and with low efforts. In order to fulfil this guidance we defined an integration layer that surrounds the Orion Context Broker. The aim is to provide interfaces to the external systems, and by doing so to allow them to exploit the features of the framework. In this section we move from the understanding of the integration problem to a solution to address it. We introduce the integration layer that performs the interfacing task, we explain how we designed it and what we implemented and deployed in order to connect it to the framework and to make it suitable for the external systems.

4.1 Integration Layer

The layer we will describe is part of a bigger work named *IoT6 Stack* that was developed for the IoT6 European Project [12]. The IoT6 Stack is the common communication interface amongst all systems present within the IoT6 project. The core elements of the IoT6 Stack are: IPv6 as Internet Protocol, Web services based on CoAP [13] as service layer, the OASIS Open Building Information Exchange standard (oBIX) [14] as object model and JSON [15] as message encoding.

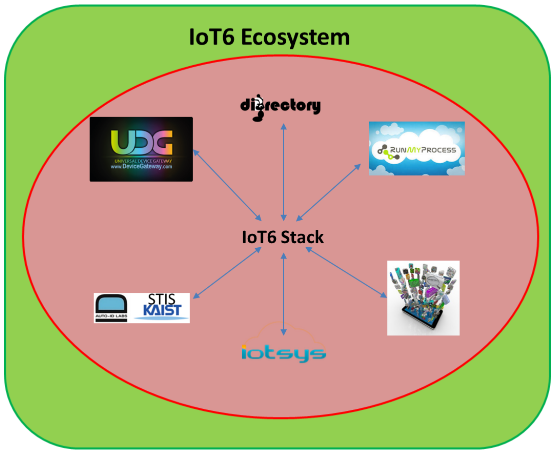


Figure .3: IoT6 Ecosystem

Figure .3 shows how the integration is obtained amid the systems operating within the IoT6 project. The IoT6 stack offers integration functionalities that allow the interconnection of all components under a common communication environment. Each interacting system is connected to the IoT6 Ecosystem through a sort of connector which is based on the IoT6 stack, and which defines the integration between the semantic of the single component and the semantic defined in the IoT6 Ecosystem.

This integration layer was developed mainly for the IoT6 project context, but our purpose is to refine it and to make it useful even outside the IoT6 Ecosystem. For this reason we defined a sub scheme of the IoT6 stack where we ignore oBIX, we replace CoAP with RESTful Web Services [16], we make use of both IPv6 and IPv4 and we preserve JSON.

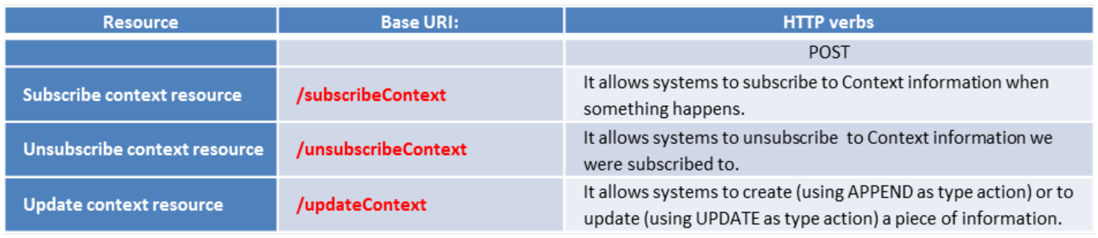
4.2 Adaptors Design

To create the desired integration we decided to equip P/S framework of an integration layer. By definition, in computer programming, frameworks are abstractions in which software providing generic functionalities can be selectively changed by additional user-written code, thus providing

[12] http://iot6.eu/
[13] http://tools.ietf.org/html/draft-ietf-core-coap-09
[14] https://www.oasis-open.org/committees/download.php/38212/oBIX-1-1-spec-wd06.pdf
[15] http://www.json.org/
[16] https://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html

application-specific software [17]. What we have to define at this phase is a way to provide to the interacting systems some tools to exploit the functionalities provided by the selected framework.

We chose a framework based on the P/S design pattern called Orion Context Broker that provides various functionalities called primitives (S.A.U. 2014b), (S.A.U. 2014a), which give to external systems the possibilities to interact with the framework through HTTP POST methods [18]. In order to attain the integration task and to create our prototype we evaluated that we needed only a subset of those functionalities. Table .1 shows the functionalities that we use for our purpose and that are made available to the interacting systems.



Resource	Base URI:	HTTP verbs
		POST
Subscribe context resource	/subscribeContext	It allows systems to subscribe to Context information when something happens.
Unsubscribe context resource	/unsubscribeContext	It allows systems to unsubscribe to Context information we were subscribed to.
Update context resource	/updateContext	It allows systems to create (using APPEND as type action) or to update (using UPDATE as type action) a piece of information.

Table .1: Orion Context Broker Functionalities - Subset of Interest

The idea is to create adaptors that use these primitives to provide easy access to the framework. Figure .4 shows how the integration layer is connected to the Orion Context Broker in order to integrate among them technologies. This layer provides interfaces that the systems must implement in order to participate in scenarios managed by the Orion Context Broker. Our aim is to create standardized self-contained adaptors that the systems can use to interact through the exploitation of the Orion Context Broker.

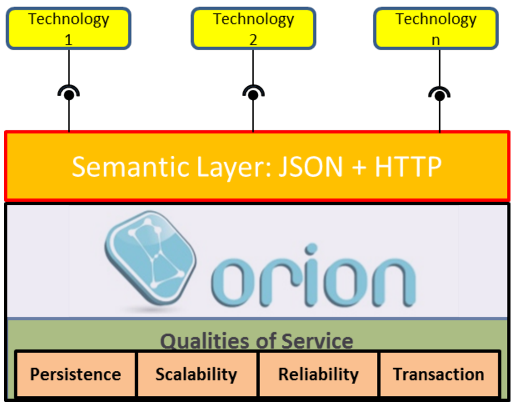


Figure .4: Orion Context Broker with Integration Layer

In order to design these adapters, we decided to adopt the Software Engineering guidelines proposed in (Natali Antonio 2012) and we started by defining the primary use cases that each interacting entity should pass through in order to exchange pieces of information using the framework. Figure .5 shows the use cases (labeled as UC01, UC02, UC03, UC04) that an entity will

[17]https://en.wikipedia.org/wiki/Software_framework
[18]http://www.w3.org/Protocols/HTTP/Methods/Post.html

encounter in performing the information exchange.

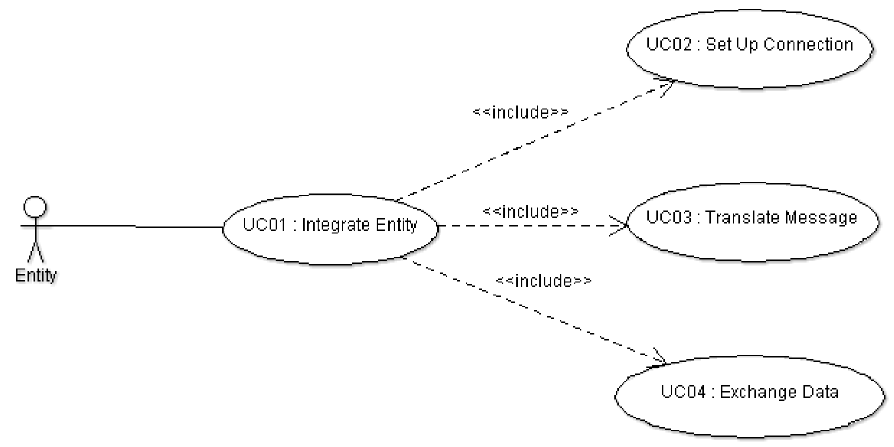
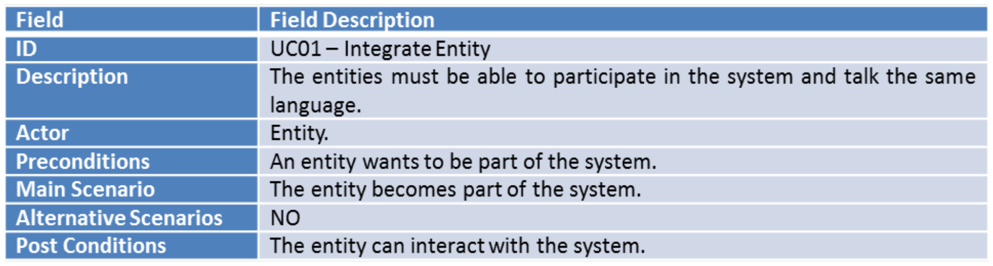
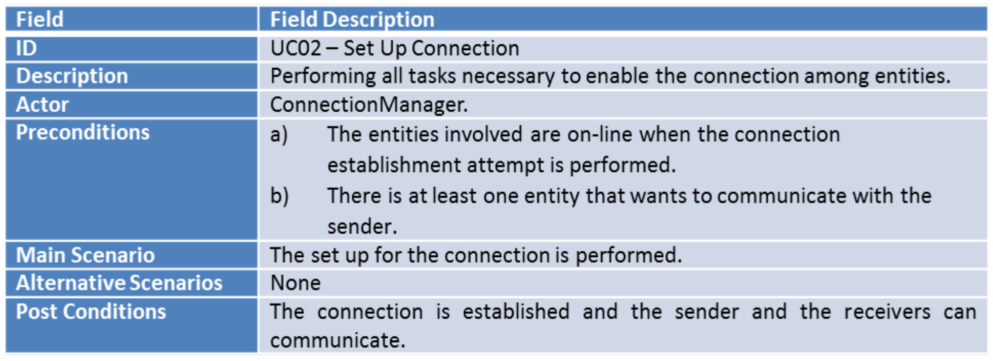


Figure .5: Primary Use Cases

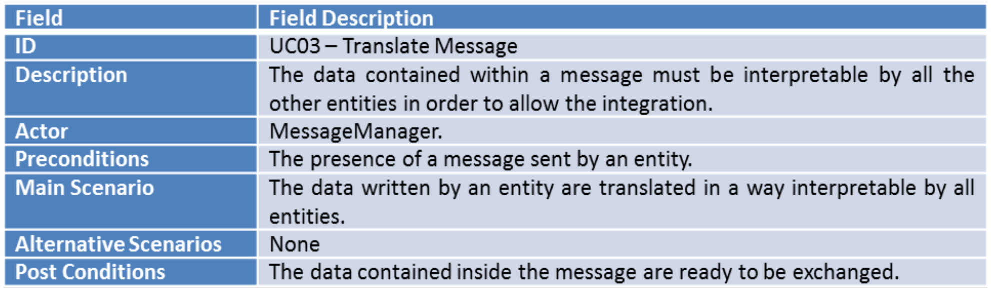
The follows four tables detail the use cases.



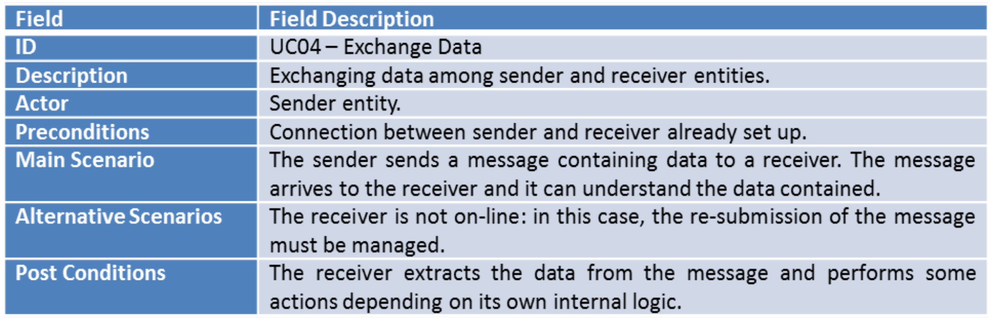
Field	Field Description
ID	UC01 – Integrate Entity
Description	The entities must be able to participate in the system and talk the same language.
Actor	Entity.
Preconditions	An entity wants to be part of the system.
Main Scenario	The entity becomes part of the system.
Alternative Scenarios	NO
Post Conditions	The entity can interact with the system.



Field	Field Description
ID	UC02 – Set Up Connection
Description	Performing all tasks necessary to enable the connection among entities.
Actor	ConnectionManager.
Preconditions	a) The entities involved are on-line when the connection establishment attempt is performed. b) There is at least one entity that wants to communicate with the sender.
Main Scenario	The set up for the connection is performed.
Alternative Scenarios	None
Post Conditions	The connection is established and the sender and the receivers can communicate.



Field	Field Description
ID	UC03 – Translate Message
Description	The data contained within a message must be interpretable by all the other entities in order to allow the integration.
Actor	MessageManager.
Preconditions	The presence of a message sent by an entity.
Main Scenario	The data written by an entity are translated in a way interpretable by all entities.
Alternative Scenarios	None
Post Conditions	The data contained inside the message are ready to be exchanged.



Field	Field Description
ID	UC04 – Exchange Data
Description	Exchanging data among sender and receiver entities.
Actor	Sender entity.
Preconditions	Connection between sender and receiver already set up.
Main Scenario	The sender sends a message containing data to a receiver. The message arrives to the receiver and it can understand the data contained.
Alternative Scenarios	The receiver is not on-line: in this case, the re-submission of the message must be managed.
Post Conditions	The receiver extracts the data from the message and performs some actions depending on its own internal logic.

Table .2: Primary Use Cases Details

The use cases highlights some nouns and verbs that helped us to define a first version of the Domain Model, where we defined all components and interactions needed to connect the entities to the Orion Context Broker in order to obtain the integration required. Figure .6 shows the first version of the domain model that was used as groundwork for the system development.

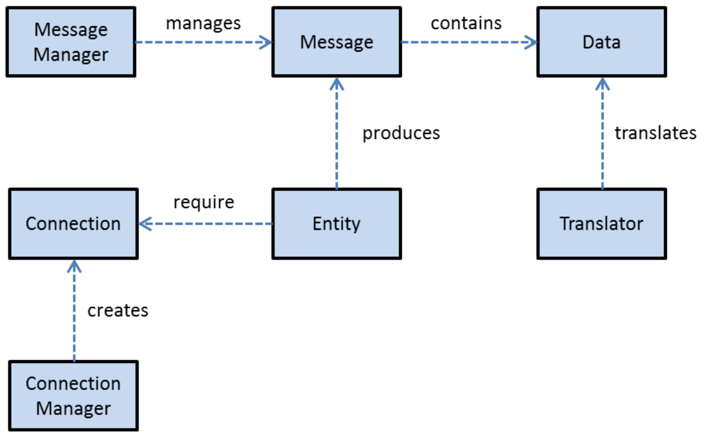


Figure .6: Domain Model

Once that we defined the primary use cases needed to integrate the entities and we modelled the domain model, we decomposed our model into the three dimensions of software (Natali Antonio 2012) to properly design our system: Structure, Interaction and Behavior.

4.2.1 Structure Dimension

To structure our system we started by dividing the components present in the Domain Model in three groups that represent different vertical dimensions as described in (Jacobson 1992). These dimensions are named: Entity, Boundary and Control.

a) Entity: it contains the concrete components which will be used by other components and it is considered as the central point of the project. The following list itemizes the components belonging to the entity dimension:

- Message
- Data
- Connection

b) Boundary: it contains the components that encapsulate the interfaces of the system towards the external world. The following list itemizes the only component belonging to the boundary dimension:

- Entity

c) Control: this is the principal group, and contains the components that mediate between the Boundary and Entity in order to perform the business logic of our system. The following list itemizes the components belonging to the control dimension:

- ConnectionManager
- Translator
- MessageManager

4.2.2 Interaction Dimension

The division of the components performed in the structure dimension gave us the possibilities to bound each component with the correlated use cases in order to discover all interactions present within our integration system. Figure .7 represents the Logical Architecture of the integration layer, and it shows how the components are correlated with the use cases (black solid arrows) and the dependencies between components (red dashed arrows).

The red dashed arrows in Figure .7 highlight the dependencies that are envisaged among the components present in our architecture. The following list itemizes the relevant dependencies needed to accomplish the building logic of the system (some dependencies have been taken out of the list because they are not meaningful within the context of the business logic model):

- Entity - ConnectionManager
- Entity - MessageManager
- MessageManager - Translator

Once the relevant dependencies have been identified, the nature of the system must be decided. This implies determining whether the system should be either a full local one, have only

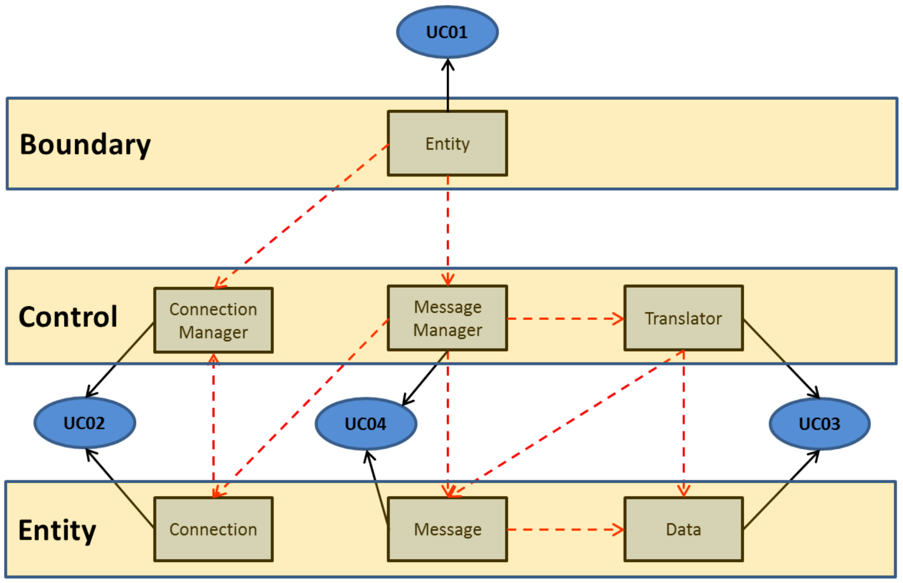


Figure .7: Logical Architecture of the Integration Layer

some components distributed, or be fully distributed. The environment in which the system will be applied leans towards a distributed direction because it is composed of distributed heterogeneous external systems that want to use a middleware in order to exchange data. The solution chosen was one in which a central framework will simulate a fully distributed environment providing a set of interfaces that will be replicated on all nodes that represent external entities. Those interfaces will allow the entities to be part of the system and communicate among them seamlessly. In the follows we analyze these dependencies.

Entity (*E*) - ConnectionManager (*CM*) An *E* depends on a *CM* in order to be connected to the system and to exchange information with other *Es*. Therefore, this interaction starts from *E* and goes towards *CM*. This leads to the following Producer - Consumer decision:

- Producer: Entity
- Consumer: ConnectionManager

Since *E* has to be certain that the *CM* has built the requested connection, a Request/Response (Natali Antonio 2012) is chosen as communication mode between them. This functionality, which we called *connectEntity()*, is provided by the *CM*.

Entity (*E*) - MessageManager (*MM*) An *E* depends on a *MM* to send information it possesses to other *Es*, and to do so it needs the *MM* as the component that takes care of the information flow. Therefore, the communication starts from *E* and goes to the *MM*, leading to the following Producer - Consumer decision:

- Producer: Entity
- Consumer: MessageManager

Since *E* assumes that the *MM* will perform its behavior properly, a response is not needed, hence a Dispatch (Natali Antonio 2012) is chosen as the communication mode between them. This functionality, which we called *dispatchInformation()* is provided by the *MM*.

MessageManager (*MM*) - Translator (*T*) A *MM* depends on a *T* because of the heterogeneity of the entities that will use the final system. Every *E* will produce pieces of information in its own way, and those pieces of information must be adapted to a common semantics in order to create messages that contain data interpretable by all other entities. Therefore, a *MM*, once it gets information to be forwarded to others has to ask for an interpretable representation of that information before dispatching it - this task is performed by *T*. This leads to the following Producer Consumer decision:

- Producer: MessageManager
- Consumer: Translator

Since the *MM* wants the *T* to return the information in an interpretable format, a Request/Response as communication mode is defined between them. This functionality, which we called *produceInterpretableMessage()* is provided by the *T* is defined.

4.2.3 Behavior Dimension

In this subsection, the attention is focused on the business logic. Since the analysis is still being performed, we cannot dig deeply into the internal entities' behavior; therefore the best approach is to define the behavior looking at the components as black boxes, by understanding what they perform, and evaluating eventual inputs and outputs. By doing so, the designer will be free to model all components as he wishes. The components of interest are those belonging to the control group defined in the following subsections.

ConnectionManager (*CM*) The *CM*'s main task is to connect the applicant *E* into the system and to provide it the capability to communicate with the other entities. When the system calls into action the *CM*, it wants it to perform all the steps needed to allow an *E* to be connected to the system. The *E* will have to provide some information necessary to join the system. By doing so, the system will send back all pieces of information *E* needs to participate to the system and to be attainable for external communications.

MessageManager (*MM*) The *MM*'s main task is to send the Message containing the information provided by the sender *E* to the interested *Es*. The *MM* is called into action when an *E* wants to send pieces of information to other *Es*.

Translator (*T*) The *T*'s main task is to format information in an interpretable way and put them into a message. When the system calls into action the *T*, it is because information has to be formatted in order to be interpretable. The *MM* will have to pass the information to the *T*, so that it can send back a message that contains information formatted in an interpretable way.

4.3 Adaptors Implementation and Deployment

The introduction of the Orion Context Broker within the Logical Architecture previously described brings some constraints related to the technologies to be used. These constraints limit the freedom of the modelling phase for interactions and architecture details. The Orion Context Broker is the software framework used in this system to dispatch the information among *Es*, as previously described. It is based on RESTful web services, so the first constraint is that entities are forced to communicate with the framework through RESTful APIs. As a consequence, we must equip *Es* with web interfaces that use HTTP methods. A second constraint is the data format exchanged among the web interfaces and the Orion Context Broker. The Orion Context Broker allows two data formats: XML and JSON - we chose to employ JSON as data format. The Orion Context Broker manages the dispatch of information to the publisher *Es* through the HTTP POST method. This feature constrains the Subscriber *Es* of being equipped with a web service interface in order to receive the information that the Orion Context Broker will dispatch to them.

4.3.1 Boundary Layer

We will now clarify how the *Es* are involved in the three dimensions previously defined:

- Structure: *Es* play a role in interfacing the system with the external world. The external world refers to the real technologies used in this project;
- Interaction: *Es* interact directly with two components of the system - the *CM* through a Request interaction and the *MM* through a Dispatch interaction;
- Behavior: the behavior of *Es* changes depending on whether they act as a publisher or a subscriber. In the case, an *E* acts as a publisher, it will act as a Web Client that will communicate with the Orion Context Broker. However, if an *E* acts as a subscriber, it will act both as a Web Client of the Orion Context Broker and as a Web Service that waits for incoming communications from the Orion Context Broker.

Led by the aforementioned clarifications, we decided to model the publisher and subscriber *Es* using an Abstract class, followed by the concrete classes that extend it (Publisher and Subscriber *Es*). The action of performing the connection to the System is common to all *Es*, and this is the reason why it will be performed by the Abstract class.

4.3.2 Control Layer

Previously we identified the *CM* as the component that allows *Es* to establish the connection with the Orion Context Broker. In order for a connection to be established both the publisher and subscriber *Es* need a Web Resource that allows them to join the Orion Context Broker. Since the communication between *Es* and the *CM* is defined as a Request/Response interaction, when an entity contacts the *CM*, it will respond with a WebResource that is necessary to join the Orion Context Broker.

Once the *CM* is obtained, we need the *MM* to manage the creation of interpretable messages exchanged between the *Es* and to dispatch those messages from publishers to subscribers. The integration of the Orion Context Broker brings a modification to this procedure, whereby the dispatching is not direct, but it is managed by the Orion Context Broker. Therefore, the *MM* provides a publish interface that allows the *Es* to insert information in the Orion Context Broker. In regard to the creation of interpretable messages, nothing changes. The *MM* provides the interfaces needed by *Es* to have some well-formatted information ready to be sent. The Orion Context Broker requires a payload as argument of the HTTP POST method. These payloads can be seen as data that *Es* send to the framework. The payload required by HTTP POST methods will create some consequences when the Data (*D*) component will be modeled.

The messages, for being well-formatted, need the *T* to create interpretable messages. It provides interfaces to allow the *MM* to request the creation of messages containing a JSON Object as D. This is requested in order to be compatible with the Orion Context Broker, and to allow the interpretation of the messages everywhere within the system.

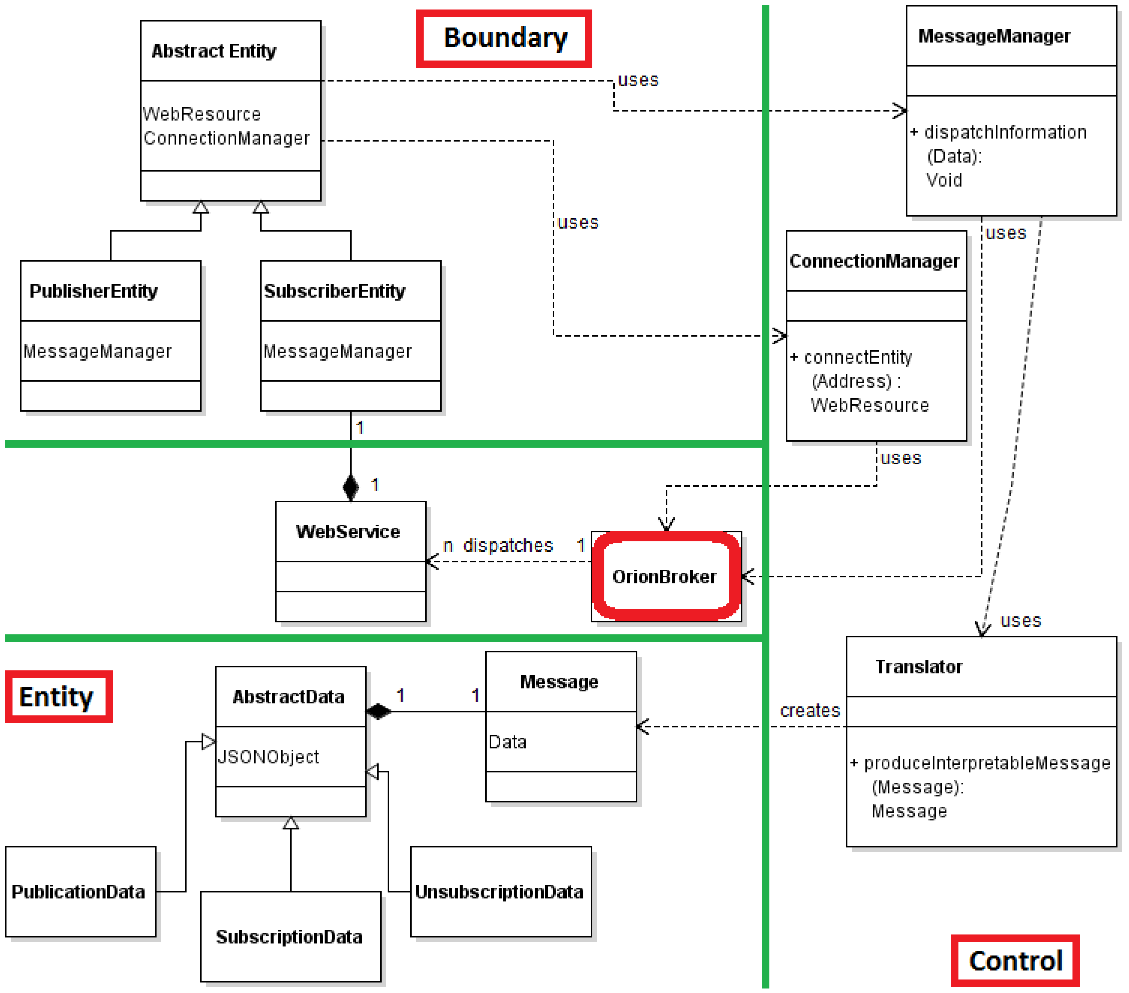


Figure .8: Final Logical Architecture

Figure .8 shows the final Logical Architecture containing all the components and the relationships between them.

4.3.3 Entity Layer

The Message (M) is the real content of the pieces of information exchanged. We decided to use JSON as data format, and by doing so a M conveys a JSON Object that can be coherently exchanged among all entities via the Orion Context Broker.

D is the actual content of the Ms that go from the entities to the Orion Context Broker and from the Orion Context Broker to the Es. Those Ms are needed for various purposes: to publish context information, to subscribe Es to context information and to unsubscribe Es from context information. The D is the full content of the payload inserted inside a M.

In the next section we show examples of adaptors' implementations and we describe how to deploy the entities in our system.

5 Example of Adaptors Implementation and System Deployment

In this section we describe how we can configure the system to manage the motivating scenario. We also show how easy it is to change the characteristics of the scenario, demonstrating how it is possible to mutate totally its configuration in order to support different scenarios.

By recalling the motivating scenario, we can identify all interactions that the external systems have with the Orion Context Broker. In the follows we itemise the interactions in which each external system participates and we detail whether for a piece of information it acts as publisher or subscriber.

a) Environment Condition: this information describes the climatic condition related to the game.
   - Publisher: Playstation
   - Subscriber: Humidifier, Air Conditioning System

b) Time of the match: this information describes the current time of the match.
   - Publisher: Playstation
   - Subscriber: Kettle

c) Call Notification: this information describes Lisa's phone call.
   - Publisher: Mobile Phone
   - Subscriber: Playstation

d) Mother Movement: this information describes that Marta is leaving the basement.
   - Publisher: Kinect
   - Subscriber: Playstation, TV

Figure .9 synthesizes the interactions by highlighting the directions of the information flows and the role that each system plays in the motivating scenario. Black solid arrows indicate information flows established at configuration time while red dashed arrows indicate information flows established at runtime through adaptations in the scenario.

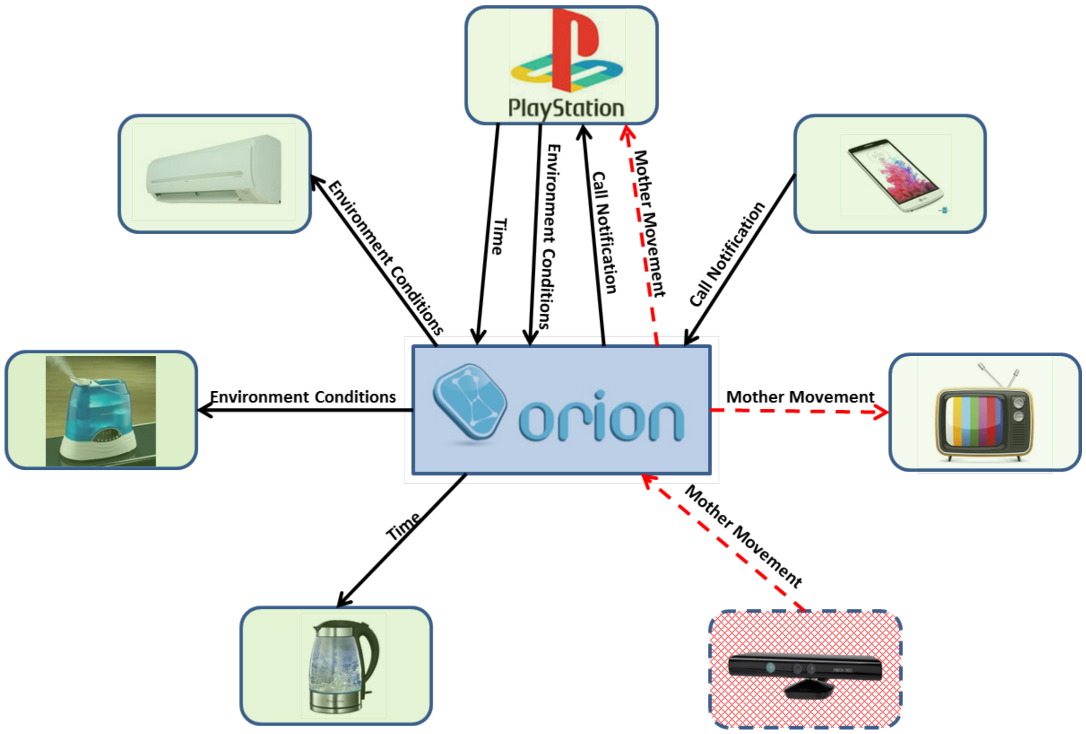


Figure .9: System and Interactions

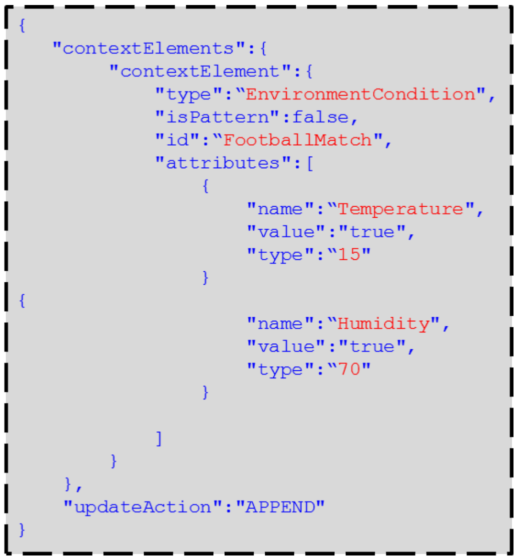
As explained since the beginning, we want the interactions to be based on the data exchanged rather than on the establishment of direct communications between communicating systems. When we decide to subscribe one or more system to some data we do not have to explicitly indicate the publishers of those data. The only task we have to perform is to create subscriptions for those data, without having any information about the providers of those data. This means that we have to create the right adaptors for the publisher system and the subscriber(s) system to let them interoperate correctly with the Orion Context Broker framework. We defined templates for the publisher and subscriber adaptors based on the Logical Architecture showed in figure .8, and below we show some snippets of code that illustrate how these templates must be adapted to interconnect two systems. The aim here is to show how to connect systems, abstracting by their inner business logic - the business logic is out of our interest. We will show few snippets of code to let the user understand how to develop adaptors and to deploy them in order to integrate external systems. Everything could be done without directly implementing the adaptors by hand - we provide special graphical user interfaces to perform these tasks.

Let us take as example the environment conditions information. Here we want to integrate three system: the Playstation, the humidifier and the air conditioning. To do so we have to define three adaptors, a publisher adaptor for the Playstation and two subscriber adaptors, one for the humidifier and one for the air conditioning system.

In the follows two sections we will show how to define the adapters needed to manage the environment condition information flows and then how to add new information flows at run-time.

5.1 Publisher Adaptor

As we said, in order to interact, a publisher entity should format the information following the semantics defined by the framework used as communication middleware.



```
{
   "contextElements":{
        "contextElement":{
            "type":"EnvironmentCondition",
            "isPattern":false,
            "id":"FootballMatch",
            "attributes":[
                {
                    "name":"Temperature",
                    "value":"true",
                    "type":"15"
                }
{
                    "name":"Humidity",
                    "value":"true",
                    "type":"70"
                }

            ]
        }
    },
    "updateAction":"APPEND"
}
```

Figure .10: JSON publisher message

The Orion Context Broker uses JSON as data format, and it bases the couplings of pieces of information on some attributes that are defined within JSON messages. These attributes are the following: element *type* and *id*, attribute *name*. Figure .10 shows the JSON message that a publisher entity (Playstation) sends to the Orion Context Broker in order to update the information related to the environment conditions.

All systems that want to act as publisher entities have to perform the following two tasks:

a) To create a *PublisherEntity* that will allow it to work as Web Client:

```
PublisherEntity publisherEntity = new PublisherEntity();
```

b) To connect themselves to the Orion Context Broker, by creating a WebResource as reference for that connection:

```
WebResource webResource = publisherEntity.connectEntity(url);
```

Once these two tasks are performed they can publish pieces of information into the Orion Context Broker following this three step procedure:

a) To create the pieces of information it wants to publish:

```
PublicationData publishData = new PublicationData();
publishData.setElement("EnvironmentCondition", "FootballMatch");
publishData.setAttribute("Temperature", "15", "true");
publishData.setAttribute("Humidity", "70", "true");
publishData.setAction("APPEND");
```

b) To create a JSON object given the information just created:

```
JSONObject contextPayload = publishData.getData();
```

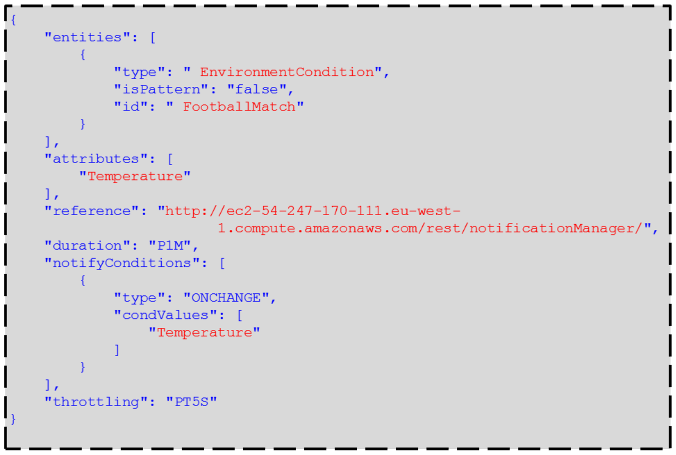
c) To publish that information - its JSON representation - into the Orion Context Broker using the WebResource:

```
publisherEntity.publish(webResource, "updateContext", contextPayload);
```

With this simple piece of code the publisher adaptor is now created and the information concerning the environment variable is available for all systems that desire it. The updating interval is decided by the publisher entity depending on its business logic.

5.2 Subscriber Adaptor

Like for publisher entities, also subscriber entities should format pieces of information as JSON.



```
{
    "entities": [
        {
            "type": " EnvironmentCondition",
            "isPattern": "false",
            "id": " FootballMatch"
        }
    ],
    "attributes": [
        "Temperature"
    ],
    "reference": "http://ec2-54-247-170-111.eu-west-
                  1.compute.amazonaws.com/rest/notificationManager/",
    "duration": "P1M",
    "notifyConditions": [
        {
            "type": "ONCHANGE",
            "condValues": [
                "Temperature"
            ]
        }
    ],
    "throttling": "PT5S"
}
```

Figure .11: JSON subscriber - Temperature

The important attributes that should be declared in a subscriber adaptor are the following: entities *type* and *id*, *attributes*, *reference* and *notifyConditions*. Figure .11 shows the JSON message that a subscriber entity (Air Conditioning System) sends to the Orion Context Broker to request pieces of information about environment conditions.

To become a subscriber entity, a system has to pass through two different phases:

- Web Client phase: this phase is needed to become a member of the system and to express its desire to receive notifications. It is created through a five steps procedure similar to the one described for publisher entities:

  a) To create a Subscriber Entity that will allow it to work as the Web Client:

  ```
  SubscriberEntity subscriberEntity = new SubscriberEntity();
  ```

  b) To connect itself to the Orion Context Broker, creating a WebResource as reference for that connection:

  ```
  WebResource webResource = subscriberEntity.connectEntity(url);
  ```

  c) To create the pieces of information it needs in order to do the subscription:

  ```
  SubscriptionData subscribeData = new SubscriptionData ()
  subscribeData.setAttribute("Temperature");
  ```

```
subscribeData.setEntity("EnvironmentCondition", "FootballMatch");
subscribeData.setDuration(1);
subscribeData.setReference(yourWebServiceURL);
subscribeData.setNotifyCondition("ONCHANGE", "Temperature");
```

In order to subscribe to pieces of information, a Subscriber Entity has to specify the url where it will deploy a Web Service that will manage the notifications. This is specified in the variable *yourWebServiceURL*.

d) To create the JSON object given the information just created:

```
JSONObject contextPayload = subscribeData.getData();
```

A Subscriber Entity that publishes this object intends to retrieve information that has *EnvironmentCondition* as the context type, *FootballMatch* as id, and *Temperature* as condition for the notifications. The Web Service will be listening to the referenced URL and it will receive notification when the value of the attribute changes.

e) To subscribe to that information - its JSON representation - in the Orion Context Broker using the WebResource:

```
subscriberEntity.subscribe(webResource,"subscribeContext",contextPayload);
```

- Web Service phase: this phase is needed to create a RESTful web service that manages the notifications and make the data carried inside pieces of information available for external systems. Subscribers' adaptors contain also a Web Service skeleton that users can use as a starting point for making their own web services. This skeleton provides the REST methods with the right *Headers* for communicating with the Orion Context Broker, abstracting the users from the framework details. The skeleton is made in a way that users can use it directly in their projects. Users just need to replace the italic part of the following URL (present also in figure .11) with their Servlet URL *http://ec2-54-247-170-110.eu-west-1.compute.amazonaws.com/*rest/**notificationManager/**. The access point of the code should be in the web path highlighted in bold. If users already have their own web service and they want to add another web resource, they can replace the web path with their new web resource path. Within the *notificationManager* access point, users should define the business logic that their systems will perform once the web service receives information from notifications.

5.3 Run-Time Modifications

Figure .9 also contains some interactions that are added at runtime (dashed red). These interactions are the ones that are added not at planning time, but during the game, and they are used in order to manage the situation on which Marta leaves the basement. We use them to illustrate how our solution provides support to modify the scenario at run-time. Thanks to our solution to modify a scenario at run-time it is easy and does not require any change to the already interacting systems. To modify it, users must only define/modify the adaptors following the procedures showed in 5.1 and 5.2. Sticking with our motivating example, a publisher adaptor for the Kinect has to be defined in order to send information concerning the *MotherMovement*, and two subscriber adaptors have to be defined in order to receive notifications about this piece of information. Nothing more is needed and the new interactions of a system do not influence the already existing ones.

This approach gives the framework the capability to adapt existing scenarios at runtime and to create new ones when desired providing a great flexibility really necessary in the IoT domain.

6 Conclusions and Future Work

The development of our solution for integrating heterogeneous technologies within the IoT domain followed an approach that tries to address all the challenges the integration brings. In this work we analyzed these challenges in the context of the IoT and we described a general approach to such integration problem.

The approach behind our solution uses some ideas from leading solutions (e.g. the abstraction of real devices with the concept of virtualization (Spiess et al. 2009)) and it refines them to address a more generic integration domain.

We applied our solution to a real scenario that comprehends all envisaged challenges and we described why it provides good support for integrating heterogeneous technologies in IoT.

Nevertheless we think that our solution can be improved and we hope this article paves the way for future work. We mention here a challenge that we faced during the implementation of our system that we believe should be the first problem to be addressed. P/S frameworks couple the pieces of information using approaches based on topic matching or context matching. These approaches work well as long as the interacting entities have knowledge about how to identify those pieces of information, but they can fail in absence of enough knowledge . However, in the IoT domain it is unthinkable to presume that a system that wants to participate to a scenario knows in advance how the pieces of information it is interested about are labeled.

Some researches have already started to study such problem (Zhang et al. 2008) (Liu and arno Jacobsen 2004), but they do not completely address it and leave open some interesting research directions.

List of acronyms with explanation

- BAS - Building Automation System
- CM - Connection Manager
- CoAP - Constrained Application Protocol
- DDS - Data-Distribution Service for Real-Time Systems
- E - Entity
- HTTP - HyperText Transfer Protocol
- IoT - Internet of Things
- IoT6 - Researcing IPv6 potential for the Internet of Things
- IPv6 - Internet Protocol version 6
- JSON - JavaScript Object Notation
- M2M - Machine-to-Machine
- MM - Message Manager
- MQTT - MQ Telemetry Transport
- oBIX - Open Building Information Xchange
- P/S - Publish/Subscribe
- RESTful - Representational State Transfer
- T - Translator
- UC - Use Case

Index of terms as a list

- **Actuator**: device that receives data and subsequently acts properly on a mechanism or system.
- Ad-hoc solution: solution designed for a specific task and it is non-generalizable or adaptable to other purposes.
- Business logic: set of events and actions that models the functionalities that a system provides.
- Cyber-psysical system: system of collaborating computational elements controlling physical entities.
- Decoupling: the separation of previously linked systems so that they may operate independently.
- Design pattern: general reusable solution to a commonly occurring problem within a given context in software design.
- Ecosystem: platform defined by more components that can interact together.
- Entity: component that interact with a system.
- Framework: abstraction in which software providing generic functionality can be selectively changed by additional user-written code, thus providing application-specific software.
- Information System: organized system for the collection, organization, storage and communication of information.
- Internet Of Things: the network of physical objects or ”things” embedded with electronics, software, sensors, and network connectivity, which enables these objects to collect and exchange data.
- Latency: time interval between the stimulation and response.
- Mainstream technology: technology belonging to various categories subjected to large usage
- Middleware: computer software that provides services to software applications beyond those available from the operating system.
- Paradigm: distinct set of concepts or thought patterns, including theories, research methods, postulates, and standards for what constitutes legitimate contributions to a field.
- Point-to-point connection: dedicated connection link between two systems or processes that directly connects two systems.
- Real-time system: system that must process information and produce a response within a specified time, else risk severe consequences, including failure.
- Requirementss specification: description of a software system to be developed, laying out functional and non-functional requirements.

- Scalability: capability of a system to handle a growing amount of work, or its potential to be enlarged in order to accommodate that growth.
- Scenario: the set of interactions that take part in accomplishing a task.
- Sensor: device that detects events and sends data about the events to a system.

Please include an index of terms used in the chapter, in a list format.

Acknowledgment

Thanks to Milene Fauquex for helping with the images, to Michael Barry for his help and to the University of Applied Sciences and Arts Western Switzerland Valais (HES-SO Valais-Wallis), especially the Institute Information Systems in Sierre for their financial and scientific support.

References

Eugster, P. T., P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, ????: The many faces of publish/subscribe. *ACM Comput. Surv.*

Hunkeler, U., H. L. Truong, and A. Stanford-Clark, 2008: Mqtt-s #x2014; a publish/subscribe protocol for wireless sensor networks. *Communication Systems Software and Middleware and Workshops, 2008. COMSWARE 2008. 3rd International Conference on*, 791–798, doi:10.1109/COMSWA.2008.4554519.

Isenor, A. and T. W. Spears, 2007: A conceptual model for service-oriented discovery of marine metadata descriptions. *CMOS Bulletin SCMO*, **35 (3)**.

Jacobson, I., 1992: Object oriented software engineering: a use case driven approach.

Jung, M., J. Weidinger, C. Reinisch, W. Kastner, C. Crettaz, A. Olivieri, and Y. Bocchi, 2012: A transparent ipv6 multi-protocol gateway to integrate building automation systems in the internet of things. *IEEE International Conference on Green Computing and Communications (GreenCom)*, 225–233, doi:10.1109/GreenCom.2012.42.

Latvakoski, J., M. B. Alaya, H. Ganem, B. Jubeh, A. Iivari, J. Leguay, J. M. Bosch, and N. Granqvist, 2014: Towards horizontal architecture for autonomic m2m service networks. *Future Internet*, **6 (2)**, 261–301, doi:10.3390/fi6020261, URL http://www.mdpi.com/1999-5903/6/2/261.

Lee, R., 2013: *Software Engineering: A Hands-On Approach*. SpringerLink : Bücher, Atlantis Press, URL https://books.google.ch/books?id=zdBEAAAAQBAJ.

Li, J., K. Yu, K. Wang, Y. Li, and S. Li, 2003: Peer-to-peer (p2p) communication system.

Liu, H. and H. arno Jacobsen, 2004: A-topss: A publish/subscribe system supporting imperfect information processing. *University of Toronto*, 1107–1110.

Liu, H., V. Ramasubramanian, and E. G. Sirer, 2005: Client behavior and feed characteristics of rss, a publish-subscribe system for web micronews. *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement*, USENIX Association, Berkeley, CA, USA, 3–3, IMC '05, URL http://dl.acm.org/citation.cfm?id=1251086.1251089.

Natali Antonio, M. A., 2012: *Costruire sistemi software. Dai modelli al codice*. Esculapio (Italy), URL http://www.amazon.it/Costruire-sistemi-software-modelli-codice/dp/887488334X.

Olivieri, A. C. and G. Rizzo, 2015: Scalable approaches to integration in heterogeneous iot and m2m scenarios. *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2015 9th International Conference on*, 358–363, doi:10.1109/IMIS.2015.55.

Pardo-Castellote, G., 2003: Omg data-distribution service: architectural overview. *Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on*, 200–206, doi:10.1109/ICDCSW.2003.1203555.

Rao, P., 2013: *Communication Systems*. McGraw Hill Education (India), URL https://books.google.ch/books?id=tRB_AgAAQBAJ.

Sanchez, L., 2012: Pub/sub, the internet of things, and 6lowpan connectivity. [Online; accessed 13-October-2015], urlhttp://www.embedded.com/electronics-blogs/cole-bin/4371184/Pub-sub–the-Internet-of-Things–and-6LoWPAN-connectivity, [Online; accessed 13-October-2015].

S.A.U., T. I., 2014a: Fi-ware ngsi-10 open restful api specification.

S.A.U., T. I., 2014b: Fi-ware ngsi-9 open restful api specification.

Schmidt, D. C. and C. ORyan, 2003: Patterns and performance of distributed real-time and embedded publisher/subscriber architectures. *Journal of Systems and Software*, **66 (3)**, 213 – 223, doi:http://dx.doi.org/10.1016/S0164-1212(02)00078-X, URL http://www.sciencedirect.com/science/article/pii/S016412120200078X, software architecture – Engineering quality attributes.

Soldatos, J., et al., 2015: Openiot: Open source internet-of-things in the cloud. *Interoperability and Open-Source Solutions for the Internet of Things*, I. Podnar arko, K. Pripui, and M. Serrano, Eds., Springer International Publishing, Lecture Notes in Computer Science, Vol. 9001, 13–25, doi:10.1007/978-3-319-16546-2_3, URL http://dx.doi.org/10.1007/978-3-319-16546-2_3.

Spiess, P., S. Karnouskos, D. Guinard, D. Savio, O. Baecker, L. Souza, and V. Trifa, 2009: Soa-based integration of the internet of things in enterprise services. *Web Services, 2009. ICWS 2009. IEEE International Conference on*, 968–975, doi:10.1109/ICWS.2009.98.

Vermesan, O. and P. Friess, (Eds.) , 2013: *Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems*. River Publishers Series in Communication, River, Aalborg, URL http://www.internet-of-things-research.eu/pdf/Converging_Technologies_for_Smart_Environments_and_Integrated_Ecosystems_IERC_Book_Open_Access_2013.pdf.

Watson, D. S., M. A. Piette, O. Sezgen, and N. Motegi, 2004: Machine to machine (m2m) technology in demand responsive commercial buildings. *2004 ACEEE Summer Study on Energy Efficiency in Buildings*, Pacific Grove, CA.

Wiederhold, G., 1992: Mediators in the architecture of future information systems. *IEEE COMPUTER*, **25 (3)**, 38–49.

Wu, G., S. Talwar, K. Johnsson, N. Himayat, and K. Johnson, 2011: M2m: From mobile to embedded internet. *Communications Magazine, IEEE*, **49 (4)**, 36–43, doi:10.1109/MCOM.2011.5741144.

Zhang, W., J. Ma, and D. Ye, 2008: Fomatch: A fuzzy ontology-based semantic matching algorithm of publish/subscribe systems. *CIMCA/IAWTIC/ISE*, M. Mohammadian, Ed., IEEE Computer Society, 111–117, URL http://dblp.uni-trier.de/db/conf/cimca/cimca2008.html#ZhangMY08.

Zorzi, M., A. Gluhak, S. Lange, and A. Bassi, 2010: From today's intranet of things to a future internet of things: a wireless-and mobility-related view. *Wireless Communications, IEEE*, **17 (6)**, 44–51, doi:10.1109/MWC.2010.5675777.