Joaquim André Sena Martins Pereira

Graduate in Electrical and Computer Engineering

Enabling IoT in Manufacturing: from device to the cloud

Dissertation for Master Degree in

Electrical and Computer Engineering

Supervisor:	Doutor Ricardo Luís Rosa Jardim Gonçalves, Professor Associado com Agregação, Faculdade Ciências e Tecnologia (FCT), Universidade Nova de Lisboa
Co-supervisor:	Doutor Carlos Manuel de Melo Agostinho, Investigador, Centro de Tecnologia e Sistemas (CTS), UNINOVA

Dissertation committee

President:	Anikó Katalin Horváth da Costa, Doutora, FCT/UNL
Opponents:	António Carlos Bárbara Grilo, Doutor, FCT/UNL
Members:	Ricardo Luís Rosa Jardim Gonçalves, Doutor, FCT/UNL

September, 2016

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Enabling IoT in Manufacturing: from Device to the Cloud.

To Manuela Sena,

Acknowledgments

I would like to thank all the people who, in some way, supported me during the realization of my course and this dissertation.

To my supervisor Professor Doctor Ricardo Gonçalves and co-supervisor Doctor Carlos Agostinho by believing in my capabilities and giving me advices towards the successful completion of this work, and the great opportunity to work in their group.

To all my colleagues at UNINOVA GRIS, especially to José Ferreira that helped and believed in me all this time, pushing me to do a better work and to successfully complete this dissertation.

To my family, that never gave up on me during the end of this phase of my life, which supported me all these years. To my brother that helped me during the time I was writing this dissertation.

Finally, to my colleagues and friends, who shared all the hard work during the course. Thank you André Coelho, João Nuno, João Lourenço, João Ralo, Nuno Ramos, Renato Paulino, Gonçalo Oliveira and Flávio Silva for all these years together and the great events we passed together.

The research leading to these results has received funding from the European Union HORIZON 2020 Programme under grant agreement n° 636909 (www.c2net-project.eu).

Abstract

Industrial automation platforms are experiencing a paradigm shift. With the new technologies and strategies that are being applied to enable a synchronization of the digital and real world, including real-time access to sensorial information and advanced networking capabilities to actively cooperate and form a nervous system within the enterprise, the amount of data that can be collected from real world and processed at digital level is growing at an exponential rate. Indeed, in modern industry, a huge amount of data is coming through sensorial networks embedded in the production line, allowing to manage the production in real-time. This dissertation proposes a data collection framework for continuously collecting data from the device to the cloud, enabling resources at manufacturing industries shop floors to be handled seamlessly. The framework envisions to provide a robust solution that besides collecting, transforming and managing data through an IoT model, facilitates the detection of patterns using collected historical sensor data. Industrial usage of this framework, accomplished in the frame of the EU C2NET project, supports and automates collaborative business opportunities and real-time monitoring of the production lines.

Keywords: Internet of Things, Cloud Computing, Data Collection, Virtualization

Resumo

As plataformas de automação industrial estão a sofrer uma mudança de paradigma. Com as novas tecnologias e estratégias que estão a ser aplicadas para permitir a sincronização do mundo digital e real, incluindo acesso em tempo real à informação sensorial e a recursos rede avançados a cooperar ativamente formando um sistema nervoso dentro da empresa, a quantidade de dados que podem ser coletadas de mundo real e tratados a nível digital está a crescer a uma taxa exponencial. De facto, na indústria moderna, a enorme quantidade de dados que entra através das redes sensoriais incorporados na linha de produção, permite gerir a produção em tempo real. Esta dissertação propõe uma framework de recolha de dados para coletar continuamente dados de dispositivos para a cloud, permitindo que os recursos na linha de produção da industria sejam facilmente integrados. A framework prevê que para fornecer uma solução robusta que, além de coletar, transformar e gerir os dados através do uso de um modelo que suporta a Internet das Coisas, facilitará também a deteção de padrões usando os dados históricos recolhidos pelos sensores. O uso industrial desta framework, realizada no âmbito do projeto europeu C2NET, tem como objetivo suportar e automatizar oportunidades de negocio de colaboração e de monitorizar em tempo real as linhas de produção.

Palavras-chave: Internet das Coisas, Cloud Computing, Coleção de Dados, Virtualização.

Table of Contents

Table of Figures

Table of Tables

Acronyms

Acronyms	Definition
API	Application Programming Interface
CEP	Complex event processing
CM	Company Middleware
DCF	Data Collection Framework
EDA	Event Driven Architecture
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure-as-a-service
IaaS	Infrastructure as a Service
ICT	Information and Communication Technologies
IETF	Internet Engineering Task Force
IoS	Internet of Services
IoT	Internet of things
IoT-A	Internet of Things – Architecture
IP	Internet Protocol
JSON	JavaScript Object Notation
M2M	Machine to Machine
MQTT	Message Queuing Telemetry Transport
MVC	Model-View-Controller
MVVM	Model-View-Viewmodel
NIST	National institute of Standards and Technologies
OWL	Web Ontology Language
PaaS	Platform as a Service
RM	Resource Management
RM UI	Resource Management User Interface
SaaS	Software as a Service
SensorML	Sensor Model Language

SME	Small Midium Enterprises
SOA	Service Oriented Architecture
SPA	Single page application
SSNO	Semantic Sensor Network Ontology
SUT	System Under Test
SWE	Sensor Web Enablement
TCP	Transmission Control Protocol
TTCN	Three and Tabular Combined Notation
UI	User Interface
W3C	World Wide Web Consutium
WWW	World Wide Web

1 Introduction

The potential benefits of Internet of Things (IoT) are almost limitless and its application is changing how we see our future. The Internet of Things is incorporating more and more our private and public-sector organizations to manage assets, optimize performance, and develop new business models. As a vital instrument to interconnect devices and act as generic enabler of the hyper-connected society, the Internet of Things has great potential to support an advanced society, to improve the energy efficiency and to optimize production processes.

Manufacturing corporations have a huge amount of data and complex business process. The necessity of handle and integrate this amount of data brings huge opportunities for companies to improve their line of production, monitoring capabilities, collaborative opportunities and also to support their real-time decision making skills. With this in mind, companies generally use private solutions that are closed from collaborative networks such as supply chains, and where instead of taking advantage of that data, emails still play the big role of information exchange. However, because of their production and distributions units' enterprises and their information systems require more and more resources that may face compatibility issues with new IT software and systems (Gubbi, Buyya & Marusic 2013).

The emergence of new technical solutions is creating a shift from basic manufacturing towards technology based and value added production which encourages manufacturers to reconsider how they operate. Rather than performing all task in-house, companies are building the capabilities to design and integrate system, for collaboration with networks of suppliers and distributors. Along the past few years, simple solutions of cloud computing provided the necessary computing services for this change, supporting also in massification of Internet of Things (IoT)

devices in manufacturing machines, facilities and fleets. Hence Industrial automation platforms are re-experiencing a paradigm shift. With these new technologies and strategies that are being applied to enable a synchronization of the digital and real world, including real-time access to sensorial information and advanced networking capabilities to actively cooperate and form a nervous system within enterprises and outside, the amount of data that can be collected from real world and processed in a digital level is growing tremendously (Hermann et al. 2015).

The industrial shift of paradigm where the Internet of Things has become a reality, coined a new concept: "Industry 4.0" that describes the organization of production processes based on technology and devices autonomously communicating with each other along the value chain. The impact of the Industry 4.0 and how benefits are realized will fluctuate among industries, but increased adoption of the concept by the industry will force traditional business models to change, leading to the emergence of new models. Hence Industry 4.0 brings new technological opportunities with interoperability, virtualization, decentralization, service orientation and modularity as the six main features expected to be provided (European Parliamentary Research Service 2015). This is the motivation behind of this dissertation, contributing to advance the research on the Industry 4.0 and IoT paradigms by proposing a cloud based framework to support a real time data collection and virtualization from real world devices, in order to evidence how IoT can enhance production lines and also lead to a shift of business model.

1.1 Research Method

Before any approach can be taken towards the solution of a given problem, it is important for an adequate research method to be defined. For the purposes of this work and the developments here presented, a methodology based on the scientific method is used, which consists in a set of techniques used to gather information related to a pre-defined subject, the formulation and testing of hypotheses according to the information obtained, and finally the retrieval of conclusions concerning the tests performed (Murray 1999). Although there are several variations concerning the scientific method, these form the basic set of elements upon which all the variants are based on.

After evaluating the subject presented in this master dissertation work, a research methodology was followed as suggested in (Schafersman 1997) and is displayed on **Figure 1-1.**

In the introduction of this dissertation, it is presented the **Problem Characterization**, which comprises the identification of a meaningful question or problem, the type of factors that may be involved, and possible ways to address the issue. It is a significant part of the methodology as all the research work will be centred on the selected problem. Then in the state of the art chapter comes the **Information Gathering** which, as the name implies, consists in gathering and retrieving relevant information about the issue being treated, enabling, consequently, the **Formulation of Hypotheses** described in the conceptual solution**.** This step consists in analysing the retrieved information and proposing a valid and testable solution to the problem. Once a

solution is proposed, it is time to validate it by **Testing**. This produces the necessary results that allow the **Evaluation** of the proposed answer and whether it serves its purpose or not. If it fails to do so, then it may be needed to perform further testing or to go back and re-formulate a new hypothesis. If it succeeds, then the solution is valid and final **Conclusions** can then be retrieved.

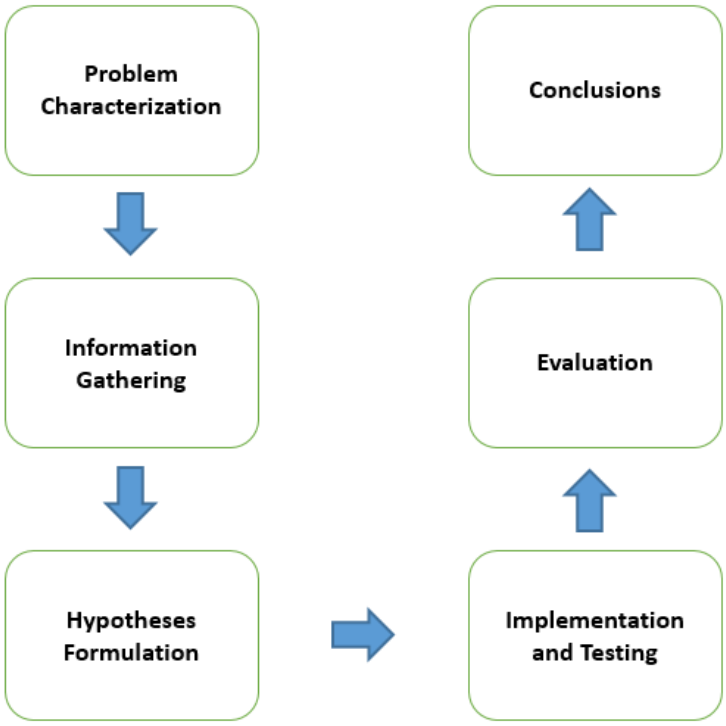

Figure 1-1 - Adopted Research Method

1.2 Research Framework and Questions

This work envisages to discuss and propose an architecture and the creation of a framework for developing key functionalities to the IoT paradigm in the manufacturing environment, as indicated in the dissertation title. The research framework addresses the technical issues to gather manufacturing data using IoT and its real-time availability to the cloud so that it can be used to optimize collaborations and provide real-time monitoring capabilities.

Consequently, the questions that arise in relation to the chosen subject are associated with the capability and feasibility to provide a IoT framework to factories, that integrates and scales easily, allowing an easy adaptation in order to improve process inside and outside factories. Considering modern factories where human hand has been the pillar of its success and the lack of trust from manufactories in IoT technologies, the following question arises:

- How is it possible to design and develop a framework that endow production lines with real time data collection technologies and how they can benefit from the use of cloud solutions?

As this dissertation seeks to contribute to the development of the IoT paradigm in industry, this question forms the base on which the research work here presented is based on.

1.3 Hypothesis

Considering the research questions previously identified and the state of the art analysis of section 2, it is conceivable to anticipate that if IoT solutions could be implemented and used to improve production lines through real time monitoring and adaptability capabilities, could lead to a shift of paradigm where terms like cloud computing and automation will be key for improve and monitor quality over the production and increase collaboration opportunities.

1.4 Dissertation Outline

This master dissertation strives to develop an architecture to aid IoT data collection from the device to the cloud in manufacturing, and address possible improvements that its adoption can bring. This dissertation is outline according to the following topics:

- **Chapter 1** introduces the main context and motivations behind this work, as well as the research methods used to approach it;
- **Chapter 2** describes the concepts of Internet of Things, Middleware, Cloud and Fog computing;
- **Chapter 3** discusses an architecture for supporting data collection and management of IoT and describes an implementation of one of its modules from the technologies chosen to the validation across a manufactory real scenario;
- **Chapter 4** discusses the conclusions obtained in regard to the results, and propose some future work that should be performed to further validate and progress on the developments here achieved.

2 State-of-the-Art

In order to propose a solution, the problem of the IoT management and data collection in manufacturing environments, is necessary to define main concepts and analyse the last proposed methodologies and solutions for issues concerning IoT. This chapter research and analyses concepts such as Internet of Things, Middleware, Cloud and Fog Computing are highlighted, facing their challenges regarding the research framework and research questions.

2.1 Internet of Things

The Internet of Things (IoT) is one of the most exciting trends in the recent history of technology (see **Figure 2-1**), and has become a common paradigm of modern Information and Communication Technologies (ICT) (Gubbi, Buyya, Marusic, et al. 2013). Following the chain of personal computers, World Wide Web (WWW) and mobile phones on the way that society works and communicate. The IoT is currently at the top of the Gartner's hype of expectations, promising to unlock several advancements to both end users and industrials (e.g. Industry 4.0).

Gershenfeld and et al. (2004) defined IoT as a concept and a paradigm that considers pervasive presence in the environment of a variety of things/objects/devices, which through wireless and wired connections and unique addressing schemes, are able to interact with each other and cooperate with other things/objects/devices to create new applications/services and reach common goals.

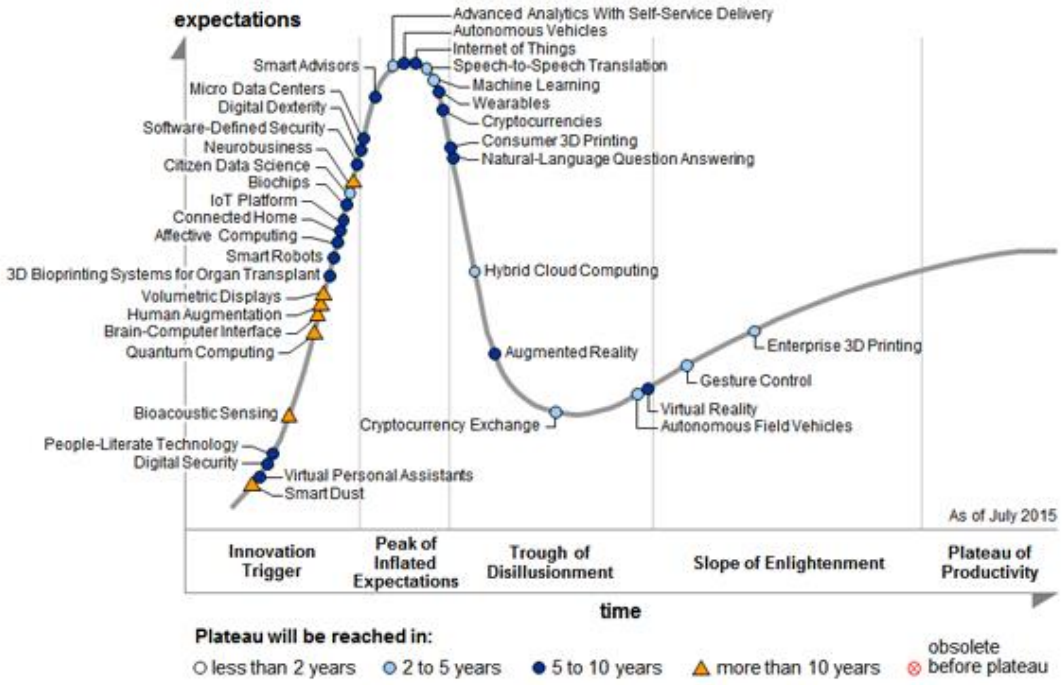

Figure 2-1 - Hype Cycle for Emerging Technologies, 2015 (Gartner's 2015)

The adoption of the IoT paradigm promises to bring huge benefits (**Figure 2-2**), to the industry and to the cities. With the continuous acceptance of this paradigm new technologies are appearing, enabling applications to synchronize the digital and real world, including real-time access to sensorial information and advanced networking capabilities to actively allow cooperation between industries (Agostinho et al. 2015). With the rise of these technologies, industry is encouraged to reconsider how to assembly, manage and operate their production lines.

Rather than performing all tasks in-house, companies are building collaboration systems with the network of suppliers (Thornton 2010). With this in mind, Evans & Annunziata (2012), have presented the view of the IoT paradigm on industry, where intelligent machines are enabled with new ways of connecting to other machines, facilities, fleets and networks though advanced sensors/actuators, controllers and software applications.

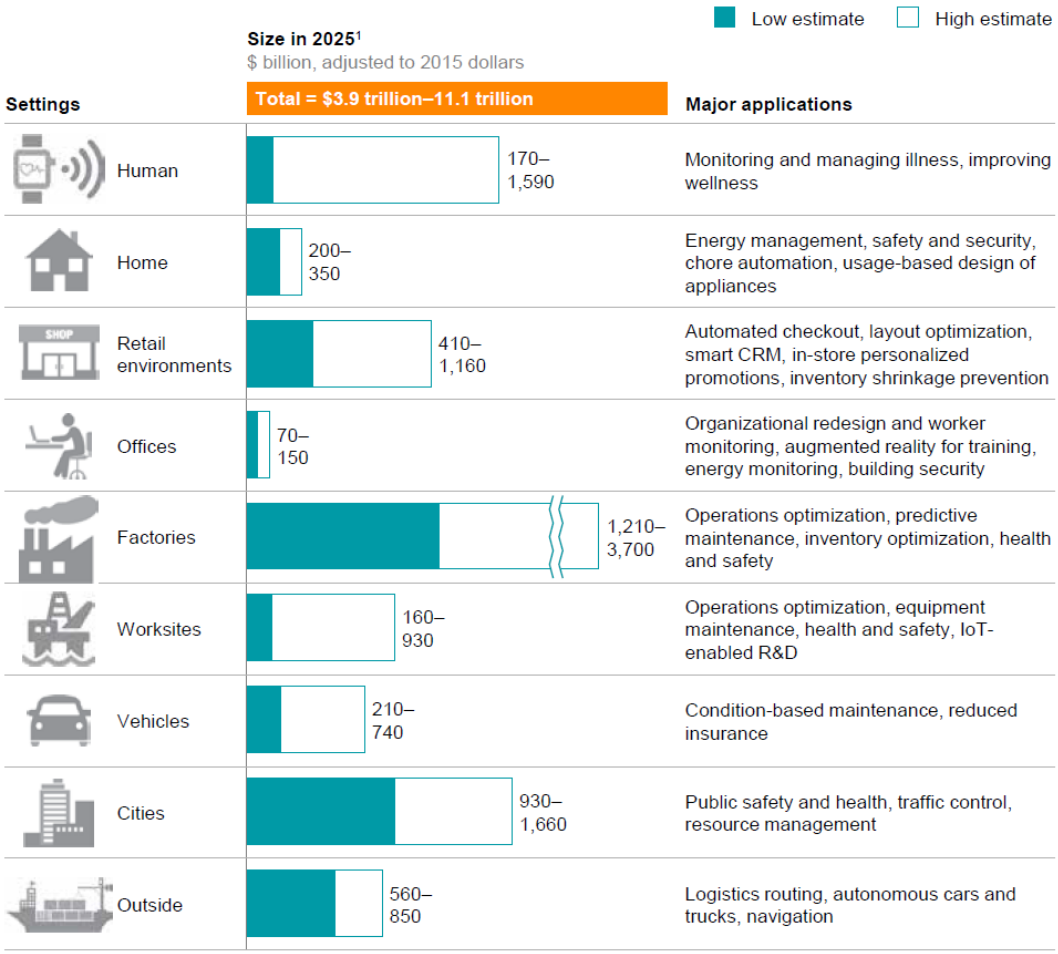

Figure 2-2 - Potential economic impact of IoT in 2025. (Manyika et al. 2015)

Although, is possible to identify by the literature, the in numerous advantages of the use of these paradigm and technologies to nowadays high societies, it is also possible to identify some challenges of its use (see **Table 2-1**). For instance, in manufacturing company's benefits such as communication, automation, control and monitoring are considered a huge benefit allowing to improve the production quality and to spare in human and material resources, although it also brings questions such as: "Who can access the information?"; "Where is the human intervention?"; "How is possible to create these type of systems since the automation process is to complex?"; "Why should I change if now my production is working correctly?".

Table 2-1 - IoT Advantages and Challenges

Advantage	Challenges
Communication - physical devices are able to stay connected with total availability and lesser inefficiencies.	**Complexity** - As with all complex systems, there are more threats and possibilities of failure.
Automation and Control - Without human intervention, the machines are able to communicate with each other leading to	**Privacy/Security** – Constant monitoring and analysis of our data can bring huge breach of our privacy and also security problems in the data exchange.

faster and interrupted results.	
Information and Monitoring - more information can be collected and awareness helps making better decisions.	**Safety/Trust** – Since is nearly impossible to give 100% automation, trust and safety over the process, there is still the need to use the consumer/operator to verify/monetarized if the desired behaviour occurs.
Efficient -The M2M interaction provides better efficiency, hence; accurate results can be fast obtained.	**Technology Controlling Life** – Our lives will be increasingly controlled by technology, and will be dependent on it. There is no way to know where it will stop.

In the next chapters, some models and technologies are presented and discussed envisioning to maximize the advantages and tackle the challenges of the use of future IoT systems.

2.1.1 IoT Models

OASIS foundation defined, reference model has an "*abstract framework for understanding significant relationships among the entities of some environment, and for the development of consistent standards or specifications supporting that environment. A reference model is based on a small number of unifying concepts and may be used as a basis for education and explaining standards to a non-specialist*"(Anon 2016). Envisioning the IoT paradigm, some reference models are already suggested (e.g. IoT-A, Semantic Sensor Network Ontology (SSNO), etc.). These models, despite trying to answer the challenges presented by IoT, tackle them from different perspectives: a more generic perspective, as IoT-A, an information/data exchange perspective as SSNO, and other works in progress.

2.1.1.1 Internet of Things - Architecture

Internet of Things – Architecture (IoT-A) is a European Lighthouse Integration Project addressing the standardization of an IoT architecture and model. It mainly focuses on developing an architectural reference model, along with security, management, and protocol-level interaction of the various components of the architecture for fostering a future Internet of Things (EC 2013).

This reference model aims to establish interoperability for IoT architectures and IoT systems. It consists of four sub-models **Figure 2-3** (IoT-A 2011):

- **Domain Model:** it is the foundation of the reference model that introduces the main concepts of the IoT objects and the relations between these. It also defines basic attributes of these objects, such as name and identifier. Furthermore, the domain model defines relationships between objects, for instance "instruments produce data sets". In the proposed model (**Figure 2-3**), through the colour scheme is possible to categorize concepts/objects (green colour) such has "*Virtual Entity*", "*Resource*" and "*Service*", etc., that are more focused on IoT software, hardware concepts (blue colour), like "Devices", "Actuators", "Tag" and "Sensors" and the representation of concepts such as "User",

"Physical Entity" and "Augmented Entity" that can be present both on hardware or/and software (beige colour), also in the user concept is present the distinction of usage by an human user or by a device usage;

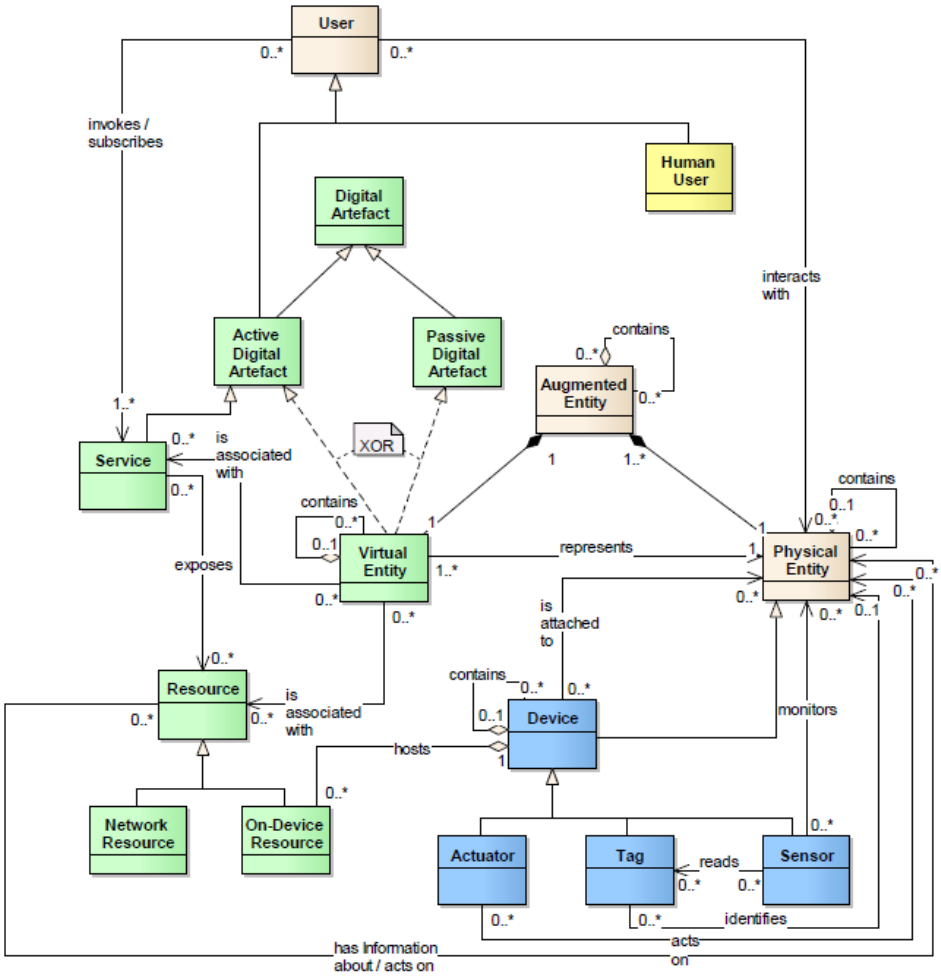

Figure 2-3 - IoT-A Domain Model (IOT-A 2011)

- **Information Model** defines the structure (e.g. relation, attributes) of all data that is handled in a system at a conceptual level. This includes the modelling of the main concepts for information flow, storage and how they are related, which allows us to model all the concepts that are explicitly represented and manipulated in the domain model (**Figure 2-4**);

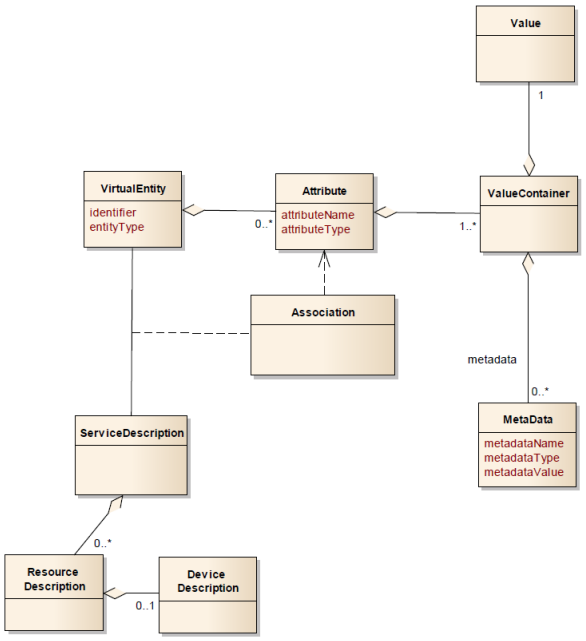

Figure 2-4 - Virtual Entity Information Model (IOT-A 2011)

- **Functional Model,** has two main purposes, the first one is to allow the complexity of a system to be decomposed in smaller and more manageable parts, the second one is to understand and illustrate their relations. **Figure 2-5** depicts the details of the virtual entity functionalities such as the characterization of the services, attributes and also the values collected by these type of entities;

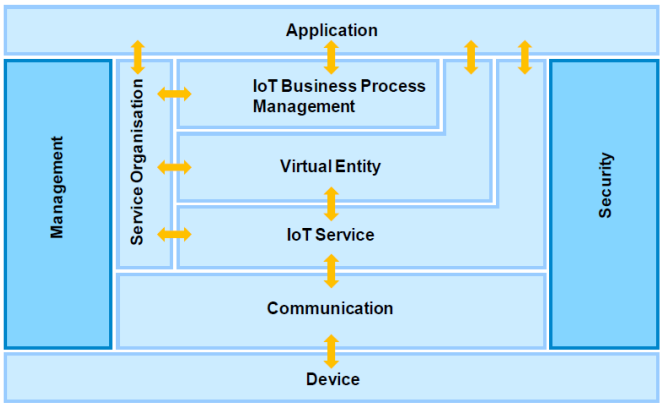

Figure 2-5 - Functional Model (IOT-A 2011)

- **Communication Model** describes the main communication paradigm for connecting entities, as defined in the domain model, as well as to manage them in order to achieve the communication features required for the IoT.

Through the reference model, IoT-A provides guidelines to design and implement IoT systems that answer challenges such as heterogeneity, interoperability, scalability, manageability, mobility, security, privacy and reliability.

2.1.1.2 Semantic Sensor Network Ontology

Semantic Sensor Network Ontology (SSNO), is a W3C project (OGC 16-079), that answers the need for a domain-independent and end-to-end model for sensing applications by merging sensor-focused, observation-focused and system-focused views. It covers the IoT subdomains which are sensor-specific such as the sensing principles and capabilities. Also it can be used to define how a sensor will perform in a particular context to help characterize the quality of sensed data or to better task sensors in unpredictable environments (Cox 2016).

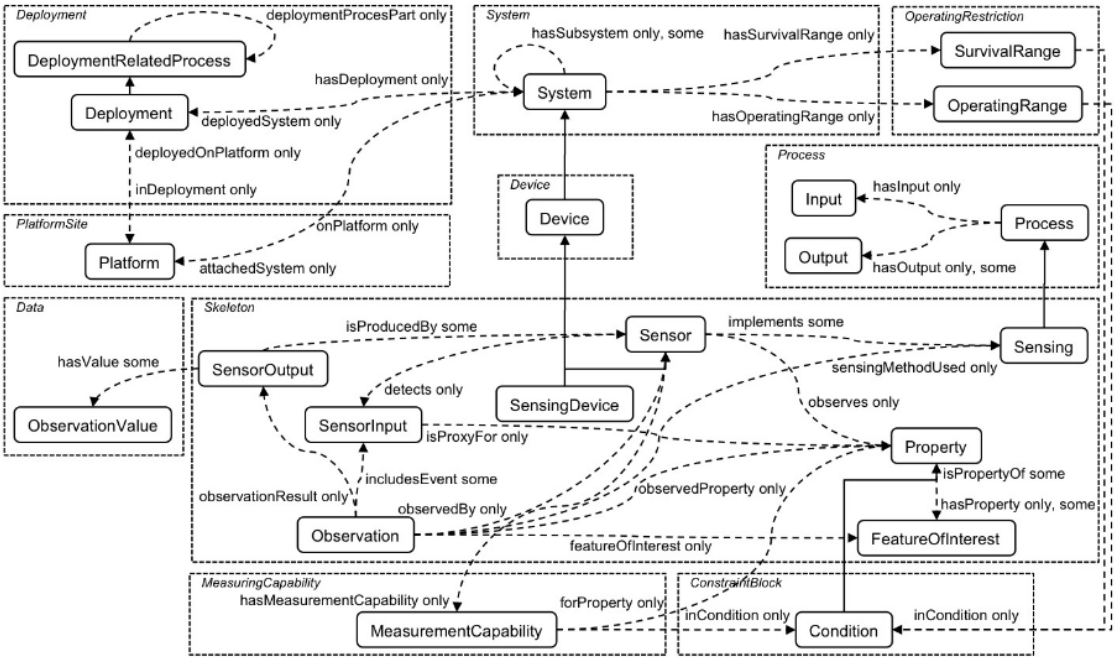

Figure 2-6 - Overview of the SSNO ontology (Anon 2011)

The SSNO, as presented in **Figure 2-6**, can be seen from four main perspectives:

- A **sensor perspective**, with focus on what it senses, how it senses, and what is sensed;
- A **data or observation perspective**, with a focus on observations and related metadata;
- A **system perspective**, with a focus on systems of sensors and there's deployments;
- **Feature and property perspective**, focusing on what senses a particular property or what observations have been made about a property.

These four perspectives allow to describe, classify and reasoning about a sensor, and to check the provenance of measurement and connections of sensors. Also SSNO extends the Sensor Model Language (SensorML) to support semantic annotation of sensor descriptions, services that support the data exchange and the network management (Compton et al. 2012), and IoT-Lite ontology., that proposes a lightweight representation and use of IoT platforms to optimize the excessive processing consuming time when querying the ontology (W3C 2015).

2.1.1.3 OSMOSE Model

The OSMOSE project proposed is an ongoing initiative to design and develop a reference architecture model for managing sensing-liquid enterprises integration. In the scope of OSMOSE project three key concepts are defined (Marques-Lucena et al. 2016):

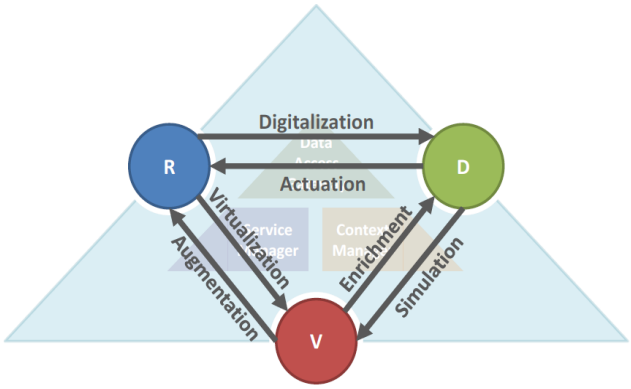

Figure 2-7 - Osmose Process for Inter-World Communication (Felic et al. 2014)

Sensing enterprise, i.e. its smart objects, equipment and infrastructures are able to cooperate actively in order to react to stimuli and form a sort of 'nervous system' with decentralized intelligence. Virtual and physical objects form an ad hoc sensor network in order to fuse multidimensional information captured from different devices and merge the real and the virtual world and to anticipate future decision making (Santucci et al. 2012).

Liquid enterprise, boundaries are fuzzy and blurred in terms of human resources (e.g. employees and partners), markets, products and processes. Its strategies and operational models will make it difficult to distinguish the 'inside' and the 'outside' of the company, such as shared resources between different companies (E. Commission, 2012).

Sensing-Liquid enterprise is described has a composition of the sensing and liquid enterprise concepts. The sensing-liquid enterprise encompasses the real, digital and virtual world each of them enclosed by a semipermeable membrane. These membranes allow controlled exchange of information between worlds. The information flow, presented on **Figure 2-7**, is characterized into three pairs of basic process.

Also, in order to enable communication between different worlds the project presented three different models to unify the representation of heterogeneous "*things*", valid both in the real, virtual and digital world. The entity model of **Figure 2-8**, comprises concepts such as "*Agents*", "*Sensor*", "*Actuators*", "*Probes*", etc., representing the structure of the sensing-liquid enterprise. The event model enables to handle all the event structure details and the process model that is mainly design for support the osmiotic process.

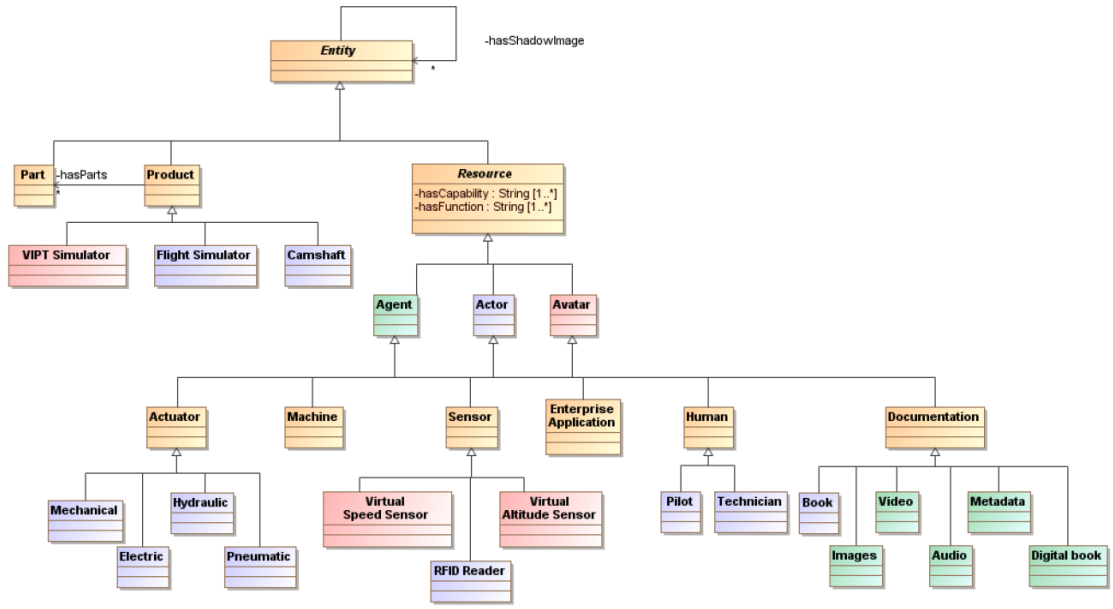

Figure 2-8 - Osmose Entity Model (Felic et al. 2014)

2.1.2 Service Management in IoT

OASIS defines service as *"a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description"* (MacKenzie et al. 2006a). Services, in the scope of IoT, helps customers connect the core of their business to the edge of the network, gain operational efficiencies and drive the creation of new revenue models, products and other services. In order to develop IoT distributed systems, service oriented architecture (SOA) and event-driven architecture (EDA) are proposed to manage IoT services. They are a natural fit due to many of their key characteristics, such as modularity, loose-couplings, and adaptability to support the complexity and constant growing of the systems.

2.1.2.1 Event-Driven Architecture

An event represents something that happens or takes place. It is usually prompted by a change of circumstance or a statement about the world. For example, a sensor value, some piece of information becoming available, an absence of something within a certain time frame, etc. This architecture enables to design and implement almost all the behaviours of a simple IoT

system but in order to design and create a complex manufacturing systems, is also needs to combine the ability to provide services and interfaces to each system component, instead of relying only in event triggered actions.

Event-driven architecture (EDA) defines a methodology for designing and implementing applications and systems in which events transmit between loosely coupled software components and services. An event-driven system is typically comprised of event consumers and event producers. The first subscribe to an intermediary event manager, and the second publish to this manager in order to trigger an action in all the clients subscribed to that specific type of event (Maréchaux 2006). EDA based systems need to support at least three key functions:

- Allow for a component to register the events that it can publish;
- Allow to a component to see what events are being published by other components and allow them to subscribe events they are interested in;
- Provide efficient even handling.

Also, it is possible to characterize the EDA system through three different types of event processing (Michelson 2006):

- **Simple Event Processing** to initiating downstream action(s). Simple event processing is commonly used to drive the real-time flow of work following notable events;
- **Stream Event Processing** both ordinary and notable events happen. Ordinary events are both screened for notability and streamed to information subscribers. Stream event processing is commonly used to drive the real-time flow of information in and around the enterprise enabling in time decision making;
- **Complex Event Processing (CEP)** deals with evaluating a confluence of events and then taking action. The events (notable or ordinary) may cross event types and occur over a long period of time. The event correlation may be casual, temporal, or spatial. CEP requires the employment of sophisticated event interpreters, event pattern definition and matching, and correlation techniques. CEP is commonly used to detect and respond to business anomalies, threats, and opportunities.

2.1.2.2 Service Oriented Architecture

A service, in the Service Oriented Architecture (SOA) context, is a mechanism to provide access to one or more capabilities through a known interface and is executed consistent with constraints and policies as specified by the service description. Despite the benefits of this architecture is possible to anticipate that it not covers all the IoT necessities since unless it is complemented with the mechanism to interact with events.

OASIS (MacKenzie et al. 2006) defines Service Oriented Architecture as a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. SOA offers a powerful framework for matching and combining needs to capabilities. Concepts, such as visibility, that refers to the capacity for those with needs and those with capabilities to be able to see each other; interaction, the activity of using capabilities; and effect, the consequence of using that capability, are key perceptions for the SOA paradigm.

The main drivers for choosing SOA-based architectures are to facilitate the manageable growth of large-scale enterprise systems, to facilitate Internet-scale provisioning and use of services and to reduce costs in organization to organization cooperation. Therefore, SOA means organizing solutions that promotes reuse, growth and interoperability, providing a solid foundation for business agility and adaptability, these are key features for creating an IoT system that envision to support the development of the never ending demand of new feature requests (Huhns & Singh 2005).

2.1.2.3 Service Oriented Architecture 2.0

SOA 2.0 is the term that used to describe the combination of service-oriented architecture and event-driven architecture (Kohli & Silicon 2008). SOA 2.0 is nothing more than a combination between SOA and EDA architectures to form a new event pattern, which provides a richer and more robust level of knowledge and structure. This new business intelligence pattern triggers further autonomous human or automated processing that adds exponential value to the enterprise by injecting value-added information into the recognized pattern which could not have been achieved previously. State of the art project working towards the IoT paradigm frequently adopt SOA 2.0.

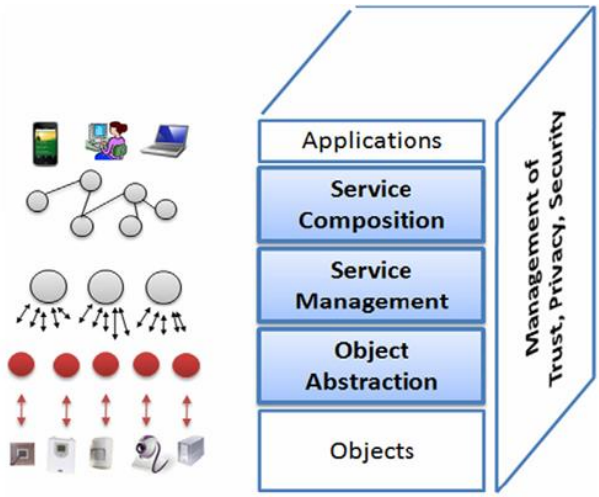

Figure 2-9 - SOA IoT Middleware (Atzori et al. 2010)

Middleware's for IoT as we know it today are designed only with client-server relationship between software modules, with services being subroutines serving clients. However, with the constant growing of IoT this model is not enough due to the lack of real time event communication capabilities. In order to solve this, SOA 2.0 an event-driven architecture (see **Figure 2-9**) was proposed with the objective of deploying software modules related to business components enabling a notification system where alerts and event notifications are transmitted.

2.2 Middleware for IoT

As introduced before, the integration of the IoT concept into the real world is possible through the integration of several enabling technologies. In order to handle with these technologies, a division of the IoT concept through a technical architecture perspective is proposed (Razzaque et al. 2015), dividing it into physical layer, middleware layer and application layer (**Figure 2-10**):

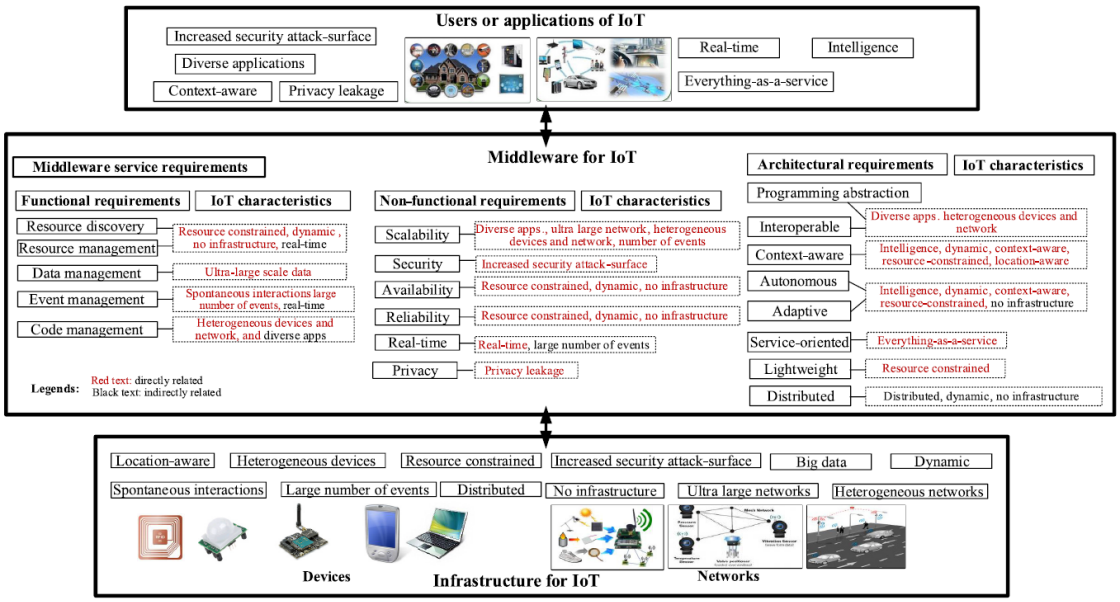

Figure 2-10 – IoT Architecture Requirements (Razzaque et al. 2015)

- **Physical or infrastructure layer** (in the bottom) is the basic networking hardware, such as sensors, actuators, computers that provide the raw data and transmit it to the application layer through the middleware;
- **Middleware layer** (in the middle) or just middleware is a software interposed between the technological and the application level that abstracts the complexities of the system or hardware allowing the developer to focus all his effort on the development of the ap-

plication layer. In the IoT paradigm, this layer is endowed with strong management capabilities in order to allow an easy integration between the use of device and the cloud platform;

- **Application layer** (in the top) is a set of interfaces or/and services responsible for displaying received information, from the middleware, to the user and handles all the user interaction with the network.

In order to design an IoT middleware solutions is necessary to develop management services, which consider real time, availability or scalability concepts, and follow IoT architectural design guide lines (Table 2-2). This approach allow to archive a more robust architecture that envision to tackle challenges such as heterogeneity of the communication technologies and system level technologies (Soma Bandyopadhyay et al. 2011).

Table 2-2 - Services and Architectural Requirements

Services	**Functional**	**Resource discovery**	IoT resources details and services should be discovered automatically without human intervention. For that, when there is no infrastructure network, every device must announce its presence and the resources it offers. Also the discovery mechanisms need to scale well, and there should be efficient distribution of discovery load, given the IoT's composition of resource-constrained devices.
		Resource management	Provide services that manages IoT resources. This means that services such monitoring, allocation/provisioning and management of conflicts in the resource usage are required.
		Data management	Data management should handle the services of data acquisition, data processing, and data storage. This processing may include data filtering, data compression, and data aggregation.
		Event management	Event management transforms simple observed events into meaningful events. It should provide real-time analysis of high-velocity data so that downstream applications are driven by accurate, real-time information, and intelligence.
		Code management	Code management should be supported since is necessary to automate code deployment or perform updates in the device network.
	Non-Functional	**Scalability**	An IoT middleware needs to be scalable to accommodate growth in the IoT's network and applications/ services
		Real Time	A middleware must provide real-time services when the correctness of an operation that supports depends not only on its logical correctness but also on the time in which it is performed.
		Reliability	All the middleware components should remain operational for the duration of a mission, even in the presence of failures.
		Availability	An IoT middleware must be available, or appear available, at all times. Even if there is a failure somewhere in the system, its recovery time and failure frequency must be small enough to achieve the desired availability.

		Security and Privacy	In IoT middleware, security needs to be considered in all the functional and non-functional blocks including the user level application
		Ease-of-deployment	Complicated installation and setup procedures must be avoided
Architectural	**Design Guide Lines**	**Programming abstraction**	The middleware should provide an API for application developers. This API needs to isolate the development of the applications or services from the operations provided by the underlying, heterogeneous IoT infrastructures.
		Interoperable	A middleware should work with heterogeneous devices/technologies/applications, without additional effort from the application or service developer.
		Service-Based	A middleware architecture should adopt a service-based development to offer high flexibility when new and advanced functions need to be added
		Adaptive	A middleware needs to be adaptive so that it can evolve to fit itself into changes in its environment or circumstances
		Context-aware	The IoT's middleware architecture needs to be aware of the context of users, devices, and the environment and use these for effective and essential services' offerings to users.
		Autonomous	Devices/ technologies/applications are active participants in the IoT's processes and they should be enabled to interact and communicate among themselves without direct human intervention.
		Distributed	A middleware implementation needs to support functions that are distributed across the physical infrastructure of the IoT.

Although there are many approaches to design and develop a middleware, there are two that stand: embedded and the cloud middleware.

2.2.1 Device-Embedded Middleware

Embedded system is an engineering artefact involving computation that is subject to physical constraints (reaction constrains and execution constraints) arising through interactions of computational process with the physical world. The key to embedded systems design is to obtain desired functionality under both kinds of constraints. Also embedded system is characterized by running in loop specific and single functioned application; optimization of energy, code size, execution time, weight and dimensions and cost; designed to meet real time constraints; and for the interaction with external world through sensors and actuators to increase the reactivity of the system (Chaqfeh & Mohamed 2012).

Embedded middleware, one of the types of embedded system, allows to control equipment's such as automobiles, home appliances, communication, control and office machines due to their small size and low battery consumption. This type of component is particularly evident in immersive realities, i.e., scenarios in which invisible embedded systems need to continuously interact with human users, in order to provide continuous sensed information and to react to service requests from the users themselves. Since the users are at the centre of the requirement,

this poses many challenges to the current embedded middleware and service technologies for embedded systems designed for simple, static and non-reconfigurable processes (Baldoni et al. 2009), in terms of:

- **Dynamicity:** since devices are no more static and distributed system need to continuously adapt on the basis of the user context, habits, etc., by adding/removing/composing on-the-fly basic elements;
- **Scalability:** in order to support the continuously growth of sensors/devices/appliances that the system need to support;
- **Dependability:** thrust on the system itself in order to users be dependent of it;
- **Security and Privacy:** user are the main focus of the services, so they need to feel that they security and privacy is assured by the system.

Combining large number of sensor allow us to create sensor networks, where is possible to have sensor behaving like embedded middleware, distributed network, or one specific device working like the coordinator of the network, centralized network.

2.2.2 Middleware on the Cloud

Infrastructure-as-a-service (IaaS) provides huge amounts of computing power (e.g. CPU, memory, storage, and bandwidth) on-demand via Web services or APIs (E. Michael Maximilien et al. 2009). Middleware deployed on the cloud is a software platform that sits between an application/device. It makes connections between any two clients, servers, databases or even applications possible (Razzaque et al. 2015). It responds to challenges such as deploy and manage applications, cloud-agnostic application development, and cross-cloud interoperation on a cloud computing platform:

- **Manage deployments in the cloud** feature should enable easy deployments, redeployments, and removal of applications in the cloud using known best practices for heterogeneous application types and development frameworks;
- **Cloud-agnostic application development** allows all application frameworks and libraries to be supported hence supporting the development on many different Web application frameworks and languages;
- **Cross-cloud interoperation** feature allow interoperation across any cloud heterogeneous application framework. That is, small and large enterprises can choose to use many cloud providers rather than standardizing on any one. Enabling redundancy in case of one cloud provider suffers a catastrophic failure allowing your entire operation to be loose couple of the consequence.

Middleware on the cloud and embedded are considered complementary approaches. This complementarity is necessary since each of one do not fully cover problems such as connectivi-

ty, scalability and availability that are necessary for supporting the never ending growth of the IoT devices in the nowadays society.

2.3 Cloud Computing

In this era of Internet of Services (IoS) (Vermesan et al. 2009), enterprises need to change their paradigm to meet the ever changing demand of customers. In response to this, cloud computing paradigm appeared, providing a dynamic infrastructure to build several kinds of services (music, movies, storage, etc.) to customers, allowing each person to customize according to their wishes and needs. The National Institute of Standards and Technologies (NIST) (Mell & Grance 2011) defines cloud computing "*as model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction".* Also NIST divided the cloud model in five essential characteristics (**Figure 2-11**), three service models (**Figure 2-12**), and four deployment models (**Figure 2-13**), which will be used to further describe the state of the art.

2.3.1 Cloud Characterization

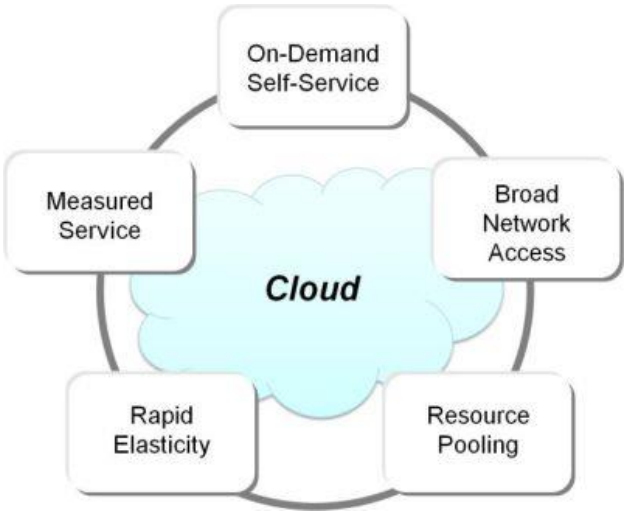

Figure 2-11 - Cloud Computing Five Characteristics (OpenGroup 2013)

On-demand self-service - One of the key features of cloud computing is that computing resources can be obtained and released on the fly, this means that a consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider which can considerably lower the operating cost.

Geo-distribution and ubiquitous network access - Clouds are generally accessible through the Internet and use the Internet as a service delivery network. With this in mind, capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations).

Resource pooling - The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. Such dynamic resource assignment capability provides much flexibility to infrastructure providers for managing their own resource usage and operating costs.

Rapid elasticity - Since resources can be allocated or deallocated on-demand, service providers are empowered to manage their resource consumption according to their own needs. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.

Measured service - Cloud computing employs a pay-per-use pricing model. So, for that cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts), these allow to monitored, controlled, and reported resource usage, providing transparency for both the provider and consumer of the utilized service.

2.3.2 Service Models

Software as a Service (SaaS) - Is a model that provides the capability to the consumer to use the provider's applications running on a cloud infrastructure. These applications are accessible from various client devices through either a thin client interface, such as a web browser, or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings (Jacobs 2005).

***Platform as a Service (PaaS)* -** Is a model where the consumer deploys, his application, onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment. Also the PaaS definition implies that the service provider is responsible for run-time monitoring and management (Keller & Rexford 2010).

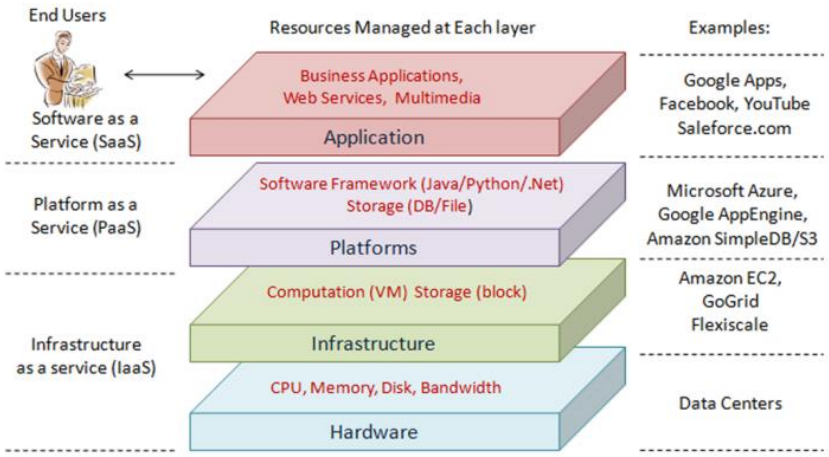

Figure 2-12 - Cloud Computing Service Models (Zhang et al. 2010)

Infrastructure as a Service (IaaS) **-** This model allows the consumer to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls). This model due the availability of high-bandwidth data communications over the Internet makes consumer to instead of owning, managing, and operating your own computer hardware, to use computers that someone else owns, manages, and operates (Bhardwaj et al. 2010).

2.3.3 Deployment Models

Private Cloud - The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers. It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises. Although this solution provides, companies, the benefits of cloud computing, without the restrictions of network bandwidth, security exposures, and legal issues that using external resources might entail, it also carries some disadvantages, such as capital investment, time-to-market since the infrastructure can take up to 6-36 months to be established, and the learning curve that cloud vendors hide it (De Chaves et al. 2011).

Public Cloud - The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider. The main benefit of using a public cloud, as opposed to creating a private cloud, is easy and inexpensive set-up. The provider has done the work needed to create the cloud, the consumer just needs to do an additional amount to configure the resources to be used (Kui et al. 2012).

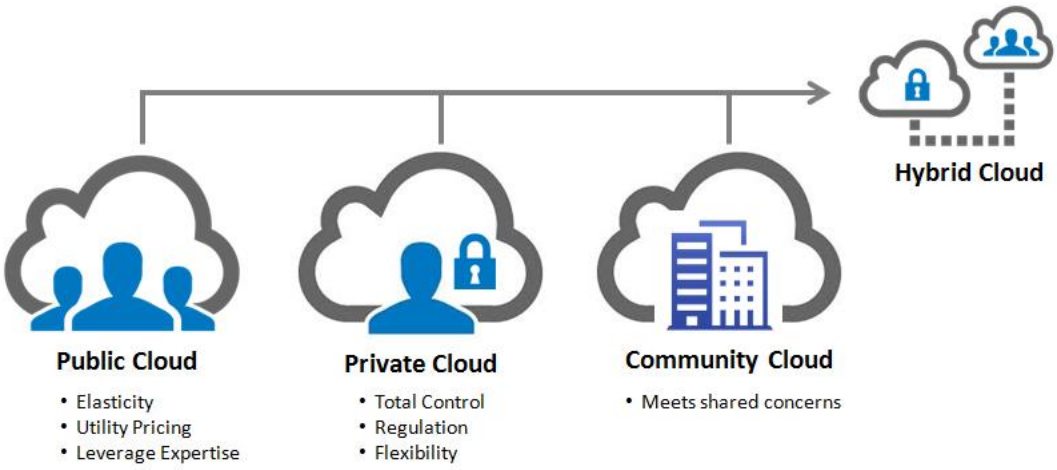

Figure 2-13 - Cloud Deployment Models (Cloud Computing Types 2016)

Community Cloud - The cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organizations that have shared concerns. It may be owned, managed, and operated by one or more of the organizations in the community, a third party, or some combination of them, and it may exist on or off premises. Community cloud, as defined here, has similarities to both private and public cloud. Like private cloud, it can avoid network bandwidth, security exposures, and legal issues that arise from using external resources, and its use can be controlled and managed. Like public cloud, it makes set-up easy for individual organizations, and it provides more efficient use of pooled resources for the whole community than any of its members could achieve individually (Dillon et al. 2010).

Hybrid Cloud - The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds). In this model customers typically host non-business-critical information and processing in the public cloud, while keeping business-critical services and data in their control in the private part of the hybrid (Zhang et al. 2009).

2.3.4 Cloud Solutions

In this sub-chapter and referring to Annex A, a comparison between different cloud services, such as IBM Bluemix, Microsoft Azure, Google Cloud, Digital Ocean, Amazon, OpenStack and WSO2 solutions is presented following the different service models and services provided.

Table 2-3 - Cloud Analysis

	IBM Bluemix	Microsoft Azure	Google Cloud	Digital-Ocean	AWS	OpenStack	WSO2
Service Model							
PaaS	X	X	X		X		X
SaaS	X	X	X		X	X	X
IaaS	X	X		X	X		
Services							
Application/Mobile Services	X	X			X		X
Virtual Machines/Containers	X	X	X	X	X	X	
Business Intelligence	X	X					
Big Data and Analytics	X	X	X		X		X
Storage/Database	X	X	X		X	X	X
Machine Learning	X	X	X				X
Internet of Things	X	X			X		X

Table 2-3 shows that the solutions provided by the IBM and Microsoft offer a full stack of services to the cover all the developer's needs to fulfil their applications, hence this brings a lack of personalization and a rigid use of their services. Platforms such as OpenStack and WSO2 are open source and highly parameterized, there so more oriented to the developer community that envisions to use a set of base functionalities essential to design their new frameworks and cloud solutions.

2.4 Fog Computing

In the past few years, Cloud computing has provided many opportunities for enterprises by offering their customers a range of computing services. However, with the IoT getting more involve in people's life, current Cloud computing paradigm can hardly satisfy their requirements of mobility support, location awareness and low latency. Over the next 10 years, current system architectures will grow increasingly due to the increasing need for high computing capacity. Fog computing architectures will be adopted to analyse timeliness and critical data in edge nodes with the aim of minimizing latency and offloading large amounts of traffic (Bonomi et al. 2014).

Fog Computing, as illustrate in **Figure 2-14**, a concept recently coined by Cisco, extending the Cloud Computing paradigm to be closer to the things that produce and act on IoT data (Bonomi et al. 2012). These devices, called fog nodes, can be deployed anywhere with a network connection: on a factory floor, on top of a power pole, alongside a railway track, in a vehicle, or on an oil rig behaving like middleware devices. Any device with computing, storage, and

network connectivity can be a fog node. Examples include industrial controllers, switches, routers, embedded servers, and video surveillance cameras.

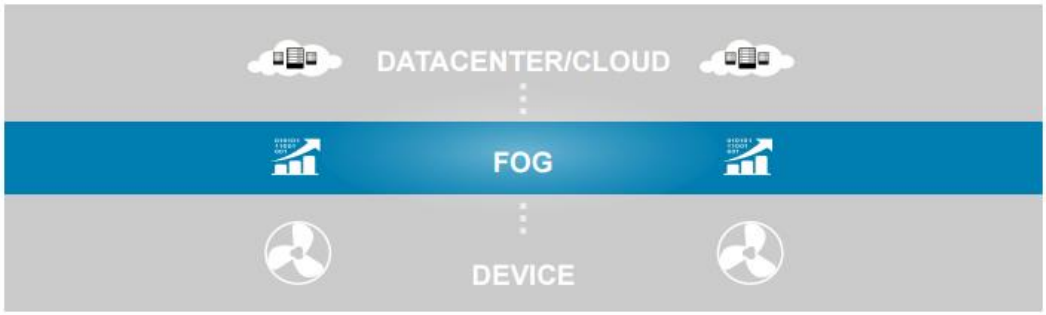

Figure 2-14 - Fog Computing Layer (Cisco Systems 2016)

In contrast to the Cloud, the Fog not only performs latency-sensitive applications at the edge of network, but also performs latency-tolerant tasks efficiently at powerful computing nodes at the intermediate of network. At the top of the Fog, Cloud computing with data centres can be still used for deep analytics (Stojmenovic & Wen 2014). Key Features of Fog Computing including:

- Edge location, location awareness, and low latency;
- Geographical distributed systems (Large-scale sensor networks, Smart Grids);
- Support for mobility;
- Real-time interactions.

3 Framework to Enable IoT in Manufacturing

In the previous chapters, a detailed analysis over the manufacturing companies constant search to enhance and improve their line of production with real-time monitoring and optimize collaboration capabilities is discussed along with the new trends that envision to give an answer to it. Concepts such as cloud and fog computing are in the centre of the IoT revolution in the manufactory. They create the base to build platforms and frameworks that allow to explore the sensing and actuating capabilities of the IoT paradigm, creating nervous system inside production lines and outside company boundaries.

In this chapter, following the SOA and EDA paradigm, an architecture for a platform to manage and collect data from IoT devices is presented, envisioning to handle IoT requirements such as device registration, virtualization, data collection, filtering, aggregation, and security and trust of the data exchanged. Also the architecture designed to be a cloud solution, allowing this way to provide key functionalities, such as scalability, availability and interoperability necessary for handle the continuous growth of the IoT needs.

3.1 Conceptual Solution

A Data Collection Framework (DCF) conceptual solution is designed to be hosted into a cloud-based service ecosystem. Thus to work properly, easy access protocols, data representation formats, well-defined interfaces, user interfaces, and efficient data storage are needed in

order to provide usage of the real-time data collection capability of the IoT devices. Also one of the main considerations in the design of the solution, is that nowadays manufactories request a solution were part of it should be deployed in their physical installations instead of a full cloud solution.

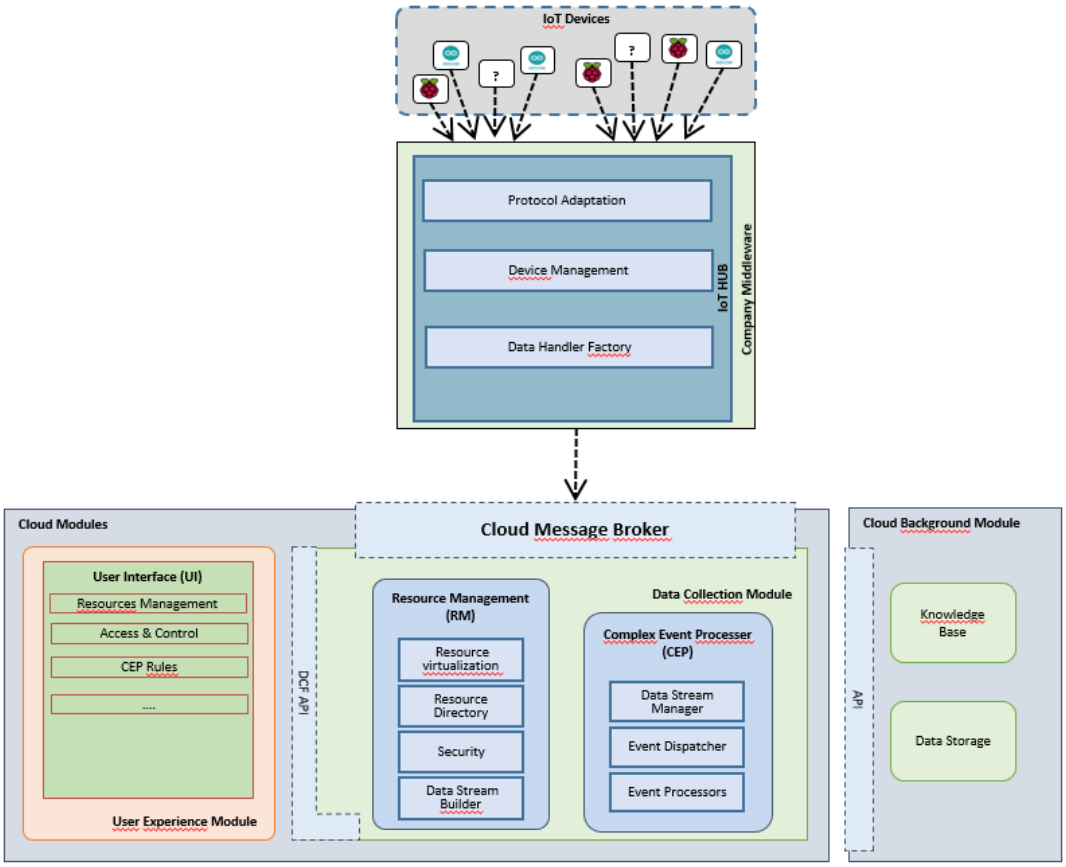

Figure 3-1 - Platform Architecture

Figure 3-1 depicts the proposed architecture to handle IoT needs. It is composed by two components, the first one, called Data Collection Framework (DCF), is deployed in the cloud and is divided in three modules, data collection; backend; UI. The second, Company Middleware, is deployed in the premises of the companies and is composed by a single module, IoT Hub. The design of the architecture is based on the concept of embedded middleware since manufacturing companies see a full cloud computing solution as a possible loss of control of their operations. IoT operation, such as registration, data exchange and event detection, described in IoT-A and SSNO are key functional features handled, also it respects non-functional requirements such as interoperability, adaptability, security and trust in order to provide support to the ever changing necessities of the manufacturing companies and IoT networks. This proposed architecture answers the goal of providing a scalable real-time architecture that support real data collection and monitoring capabilities to the manufacturing production lines.

3.1.1 Company Middleware

The Company middleware(CM) is a collection of physical devices (the IoT Hub) that are deployable in the factory to different systems to cloud components, with the mission of gathering, structure and provide data from the IoT resources in the shop floor with the expected format to DCF. In the scope of the company middleware a data stream is a set of timestamped relations, i.e., each element of the data stream consists of a set of tuples with the data collected by different devices (for instance temperature, humidity, light sensors). The order of the data stream is derived from the ordering of the timestamps and the IoT Hub module provides support to manage and manipulate the timestamps. In this way it is always possible to trace the temporal history of data stream elements collected through it. This allows company middleware to be the central observation tool for the physical world. Due to the dimension of factories multiple IoT Hubs modules can be deployed in is private infrastructure, with direct connection to their IoT networks and systems, but all of them are contained inside the company middleware.

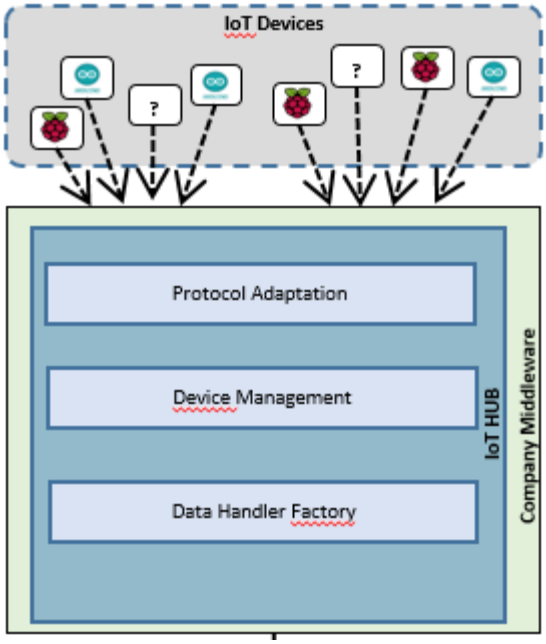

Figure 3-2 - Company Middleware

3.1.1.1 IoT Hub

IoT Hub is the device inside the company middleware which collects data from physical devices and forwards it to the platform. It allows multiple external/internal devices to be connected at the same time and provides a dedicated communication channel to allow continuous data collection from all the devices. The main responsibility of the hub is to take the messages from devices and restructure it in the cloud communication and data model, hence enabling communication between cloud and the IoT devices. In general devices are classified as IoT Hub

Compliant when communicating in the communication standard defined by IoT Hub and Non-compliant when communicating only through their own communication standard. IoT Hub will allow communication through two standards i.e. HTTP 1.1 and MQTT.

- **HTTP 1.1**: First supported protocol is HTTP 1.1 protocol, as defined by IETF RFC-2616. Devices capable of communication via HTTP can directly connect to IoT hub;
- **MQTT**: Second supported protocol is MQTT. The MQTT protocol is based on top of TCP/IP and both client and broker need to have a TCP/IP stack. MQTT connection itself is always between one client (device) and the broker (IoT Hub).

Main Functionalities Envisaged:

- Provide an interface for communication with the IoT devices;
- Conception and management of necessary communication channels and provide message routing;
- Provide implementation for communication protocol conversion between the external protocols and the internal communication protocol;
- Registration and status of the device to provide awareness to the platform;
- Security rules for IoT devices connected to the platform;
- Data pre-processing, filtering, aggregating and merging from different sources, in real time;
- Message representation transformations i.e. creation of data streams.

Sub-Modules:

- **Protocol Adaption:** component is responsible for providing seamless communication between IoT devices located at Shop floor and other components of IoT Hub. Major functionalities to be provided by this component are to allow connectivity for various devices with different communication standards and publish/subscribe paradigm for devices integration;
- **Device Management:** contains much of the core management functionality of the IoT hub and handles communication channels between devices and IoT Hub and IoT Hub and DCF. The respective details of the device like native communication protocol, device type, data type etc. are collected from the meta-data of the device from the details stored in the data base through the DCF resource management module. At the same time other important objective of this component is to guarantee status awareness, secu-

rity and trust over the connected IoT devices;

- **Data Handler Factory:** module will consume the huge amount of data coming from external sources, filtering, aggregating and merging it in real-time allowing applications to consume only to specific/value-added data or pre-processed.

3.1.2 Data Collection Framework

The Data Collection Module, **Figure 3-3**, provides unified services and operational support management functionalities, enabling different applications and end-users to discover, use and activate/deactivate resources, manage their properties and detect events/patterns in the data sent by the CM. In the scope of this dissertation, resources are the entities capable of producing data, i.e. IoT devices such as sensors and actuators. To do so, this module will focus on global model schemes for resources, i.e. IoT-A and SSNO, providing an infrastructure to link them with relevant system/devices and developing a common management tool for configuring, operating, maintaining. These operations are provided by resource management (RM), and detect events from the resources through the complex event processing(CEP) sub-module.

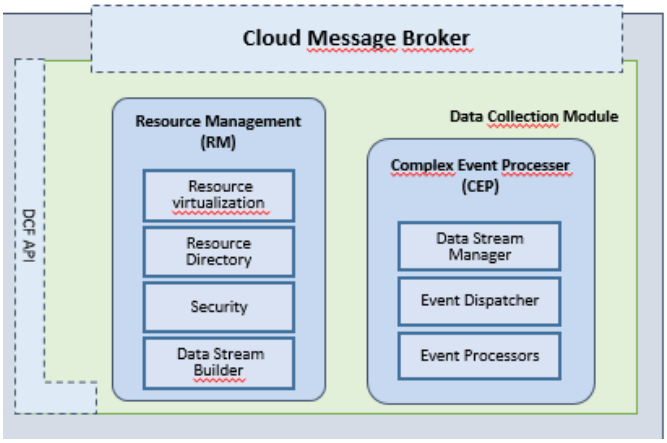

Figure 3-3 - Data Collection Framework

3.1.2.1 Resource Management

This module is responsible for providing core management functionalities over the data sources. It delivers the details of all registered data sources through an interface. It aims to reduce the distance between the physical world and their virtualized representation objects in the network, following three major administrative functionalities:

- **Resource sharing:** users can modify/remove/add virtual resources on the fly during runtime. The component needs to keep track of all on-site physical/computation resources;

- **Failure management:** In case the faults are detected in the physical resource or virtual resources e.g., by runtime exceptions, the Resource Management component should communicate it to the Complex Event module to perform the exception respective contra action. This action should be immediately notified to the end-user so that the effect on the business processes can be monitored;
- **Explicit resource control:** The admin user should be given the rights to specify explicitly computing resources (e.g. memory and processing requirements and restrictions) for the virtualized real resource.

Besides these administrative functionalities, this component will provide the functionalities related to data sharing, creation of new virtualized resourced by aggregating existing resources and manage CEP rules for each device, at the same time this component provides integrated security over data sources.

Main Functionalities Envisaged:

- Allow construction and management of virtual resources;
- Creation of output data streams based on the inputs from the IoT hub to be forwarded to CEP;
- Allow discovery of all the virtual resources based on the requirements of the end-user;
- Provide run-time adaptation on security polices by analysing the behaviour of resources to respond to new and unusual threats in critical services.

Sub-Modules:

Resource Virtualization: provides the implementation for creating, managing, and sharing the virtualization and the configuration of each devices to be shared with the IoT Hub. The configuration details of a virtual resource provide all necessary information required for deploying and using it, including:

- Metadata used for identification and discovery;
- The structure and properties of the data streams which the virtual sensor consumes and produces;
- A declarative SQL-based specification of the data stream processing and aggregation performed by the virtual sensor;
- And additional functional properties related to stream quality management, persistency, error handling and life-cycle management.

At runtime this sub-component is also the point of communication for cloud message broker of the data produce by the virtualized resources.

Resource directory: this provides the API for end-users to persistently store the details about the virtual resources and offers functionalities to discover all virtualized resources based on the requirements of business processes. If the end-user does not find the resources that fulfil the business needs, then new virtual resources are created through the resource virtualization sub-module.

Security: is responsible for providing security and trust in a real-time environment, thus extending the method for building trust and improving security policies. It detects malicious behaviors at runtime due to the learning, adaptation, prevention, identification and answering approaches to achieve adaptive security. This strategy is achieved by the ontology-based security framework.

Data Stream Builder: transforms Hubs data collected and pass it through the backend API.

3.1.2.2 Complex Event Processing

This module deals with the detection and processing of composite events coming from the cloud message broker data streams. It provides a declarative way to detect complex events in real-time and for specifying the interesting event patterns including time, causality, and pattern matching features. Detected patterns are then stored in a knowledge base.

Main Functionalities Envisaged:

- Event pattern detection;
- Event filtering;
- Event aggregation;
- Event transformation;
- Event relationship analysis.

Sub-Modules:

Data stream manager: it is used to manage continuous data streams by handling potentially infinite and rapidly changing. It provides an interface for incoming data feeds then converts the data into event types that the CEP understands, also provides a flexible query processing system, in form of rules, to detect interesting patterns.

Event dispatcher: it acts as a virtual pipe or channel, connecting components that send events with components that receive events.

Event Processors: executes user-defined event processing rules against streams and delivers its results.

3.1.2.3 Cloud Message Broker

The cloud message broker is the access point for remote connections, it provides message validation, transformation and routing, minimizing the mutual awareness that devices should have about the cloud solution in order to be able to exchange messages. Cloud message broker will also provide a publish-subscribe pattern for each company middleware connected, allowing to provide an easy pluggable interface that ensures an easy interoperability and scalability between the hubs, deployed in the shop floor, and the framework deployed on the cloud.

Main Functionalities Envisaged:

- Route messages to one or more of many destinations;
- Transform messages to an alternative representation.

3.1.3 Backend Module

Backend Module, **Figure 3-4**, is a centralized module that provides storage capability to the platform, it is designed with two sub-models that allow to handle the real-time data collection and a more dynamic storage to infer knowledge and provide semantics rules to the platform.

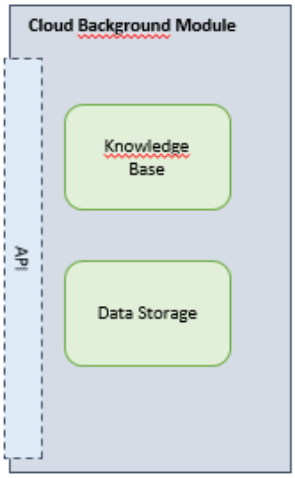

Figure 3-4 - Backend Module

Main Functionalities Envisaged:

- Store Information regarding the IoT device;
- Store data collected by IoT devices;
- Extract knowledge from the data collected by the IoT devices;

- Provide enriched semantics to the platform.

Sub-Modules:

- **Data Storage**, store in real-time all raw data collected from the IoT devices;
- **Knowledge Base**, store semantics and apply rules for extracting knowledge from the data stored in the Data Storage model.

3.1.4 User Interface Module

User Interface Module, **Figure 3-5**, is responsible to provide to the platform a centralized and standardized way to interact with the end-user over the web. Also it is the only module that establishes a public API to allow external developers to extend all the views that the end-user need.

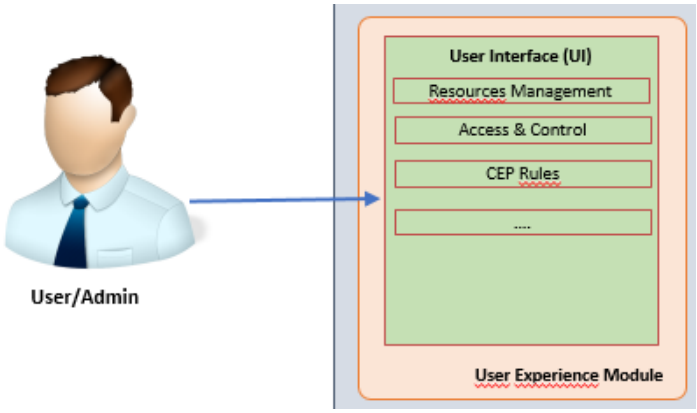

Figure 3-5 - User Interfaces

Main Functionalities Envisaged:

- Centralized and standardized User Interfaces repository and design;
- Display information and manage interactions with the end-user.

3.2 Implementation of the Proof of Concept

The previous section introduced the concepts to design the architecture for collecting data from the IoT devices, as well as the main functionalities envisaged for each module. However, in order to properly achieve the formerly referred functionalities, namely those concerning the resource management module, several technologies had to be evaluated and selected for applications.

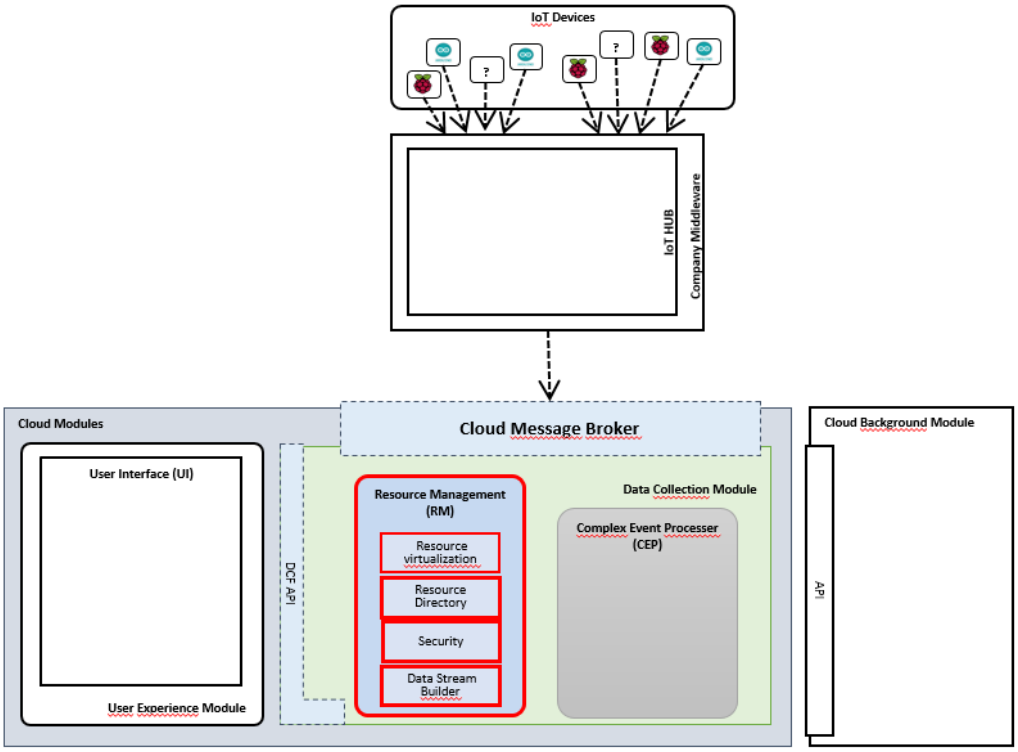

Figure 3-6 - Platform Developed Component

The work developed in this dissertation, **Figure 3-6**, is part of an ongoing research project (C2NET[1]) and due to the size and complexity of the proposed architecture, the following sections describing the dissertation proof of concept implementation will be mainly focused on the resource management module and in its validation. Thus, this section introduces the technologies and how they are used to implement the resource management module, presenting a hypothetical scenario in order to improve readers understanding of module's validation.

3.2.1 Industrial Scenario: An example from Metalworking Sector

Nowadays manufactory is recognizing the lack of real time monitoring systems is a reality, so with this in mind it was choose to validate the proposed framework, focusing on the resource management module, in a real life scenario. The use of the manufacture industry scenario, in particular the metalworking SME's, is suggested due to their particular characteristics. They typical are small companies, with little resources and a little or none IoT technologies. As an example, in the Metalworking SME's it is necessary to reduce time detection of non-conformity products during the production process. This mechanism can help the companies reduce the quantities of waste and non-conforming products that may arise from errors in the setup of machines. **Figure 3-7** illustrates a set of steps to be conducted as a validation story board:

1 http://c2net-project.eu/

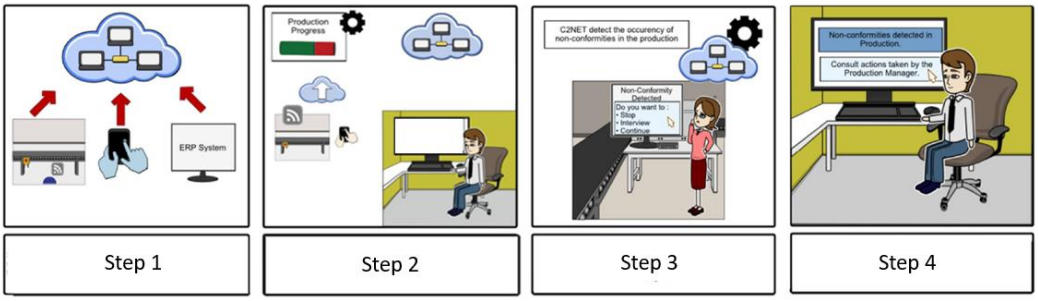

Figure 3-7 - Storyboard for Management of Non-Conformity Scheduling Plans.

- **Step 1:** The framework will allow to configure the sensors and actuators in order to start the production and to define parameters to detect non-conformities in real-time;
- **Step 2:** The framework collects information about the shop-floor production directly via sensor network;
- **Step 3**: If non-conformities are detected in the quality control by direct matching between what was expected in the production and what is actually being produced, the architecture will rise an event;
- **Step 4:** The framework will allow to re-configure the sensors and actuators in order to correct non-conformities detected in real-time.

This validation scenario is expected to be translated into a reduction of waste and non-conform products that are created during the production that derivate from flaws in the machinery setups, or that arise from faulty materials, etc. Also this scenario, in future steps, could rise collaborative opportunities with other companies (garbage, transportation, logistics, maintenance, etc), for instance when the production finish a notification could be automatically send to the garbage company to collect the debris produced by the production process, or a negotiation process may be triggered to transport the final product to the customer.

3.2.2 Technologies Selection

In this section core technologies for implementing the resource management, such as nodeJS, ExpressJS, and AngularJS, along with technologies suggested to implement other modules, such as Apache Stream for the CEP module and the RabbitMQ for both the cloud broker module and the CM are presented. These technologies are suggested based on an event oriented technologies, to answer IoT architecture needs, and grounded on the fact that state of the art technologies are more oriented to it than standardized and well established ones.

NodeJS[2]: is a JavaScript runtime built on Chrome's V8 JavaScript engine, it uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Also Node.js pro-

[2] https://nodejs.org/en/

vides with him a package ecosystem, npm[3]. Node.js specializes in performing and scaling well for low-CPU, highly I/O-bound operations comparing with JAVA[4] or C#[5] runtimes.

ExpressJS[6]: Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. This allowed to create the resource management API and provide endpoint for each of its functionalities. This technology choice is strongly coupled with the choice of the native develop language, if your choice was JAVA you should consider Spring [7]or Play[8] frameworks or in case of C# you should consider to use the .NET WebAPI framework[9].

AngularJS[10]: is a front-end web application framework to address many of the challenges encountered in developing single-page applications (SPA). It aims to simplify both the development and the testing of such applications by providing a framework for client-side model–view–controller (MVC) and model–view–viewModel (MVVM) architectures, along with components commonly used in rich Internet applications. This allowed to enrich and endow the RM UI with dynamic capabilities regarding present static content to the user. Other ways of developing SPA's are ember.js[11] or backbone.js[12], however angular was chosen due to his innovative approach for extending HTML[13] and easy use.

Apache Stream[14]: is suggested to the CEP module, brings Spark's language-integrated API to stream processing, letting the user write streaming jobs the same way he writes batch jobs, also spark streaming lets the user join streams against historical data, or run ad-hoc queries on stream state and is design to supports fault tolerance capabilities. These features and the performance that apache stream shares made it the best fit for answering the requirements of the CEP module, leading to his adoption instead of solutions such as Storm[15] or Flink[16] that only offer a partial cover

RabbitMQ[17]: has a broad adoption and a lot of community support. It proposes an adapter for MQTT making it suitable for an IoT project. It also has a good interoperability with Spark, which is another advantage regarding integration of the different components of architecture. With this reason in mind, this technology is suggested for supporting the development of

[3] https://www.npmjs.com/
[4] http://java.com/
[5] https://msdn.microsoft.com/en-us/library/67ef8sbd.aspx
[6] https://expressjs.com/
[7] https://spring.io/
[8] https://www.playframework.com/
[9] http://www.asp.net/web-api
[10] https://angularjs.org/
[11] http://emberjs.com/
[12] http://backbonejs.org/
[13] https://developer.mozilla.org/pt-PT/docs/Web/HTML
[14] http://spark.apache.org/streaming/
[15] http://storm.apache.org/
[16] https://flink.apache.org/
[17] https://www.rabbitmq.com/

the cloud broker and the Hub protocol adaption module, since it is a light implementation comparing with Kafka[18] and ActiveMQ [19].

After the core technologies being decided it is necessary to define the deployment environment. Through the analysis in section 2.3.4, IBM Bluemix[20] and Microsoft Azure[21] cloud solutions provide the largest set of services and functionalities, however their prices are to elevated and their solutions to closed for the requirements of the architecture proposed, leading this way to use more economic and customizable solutions with a more selective set of features such as OpensStack[22] and WSO2[23].

3.2.3 Resource Management Implementation

In order to provide a more detailed explanation of the implementation of the resource management module functionalities, described in Section 3.1.2.1, the contributions provided to the implementation of this module were divided in four core concepts: the IoT model where all the information regarding the devices and data from it is instantiated; the virtualization flow offered by the module; the API to access key functionalities such as creating, deleting and access the information of the devices, and also an implementation of a user interface (UI) that allow us to offer a visual way to the user to validate the use of the API for the scenario presented on section 3.2.1.

3.2.3.1 IoT Model for the backend module

The resource management module need to handle the bulk of the data from several sensor networks. In order to make sense of the huge amount of data such networks output, and given the diversity of possible available sensor systems, there is a need to specify a formal model that allows reasoning to scale. This model, inspired in the IoT-A and SSNO design principles, is formalized using the language formalism of OWL 2[24] endowing the model with its advanced reasoning capabilities and tools available. Also for reasoning the following major concepts where defined:

[18] http://kafka.apache.org/
[19] http://activemq.apache.org/
[20] http://www.ibm.com/cloud-computing/bluemix/
[21] https://azure.microsoft.com/
[22] https://www.openstack.org/
[23] http://wso2.com/
[24] https://www.w3.org/TR/owl-primer/

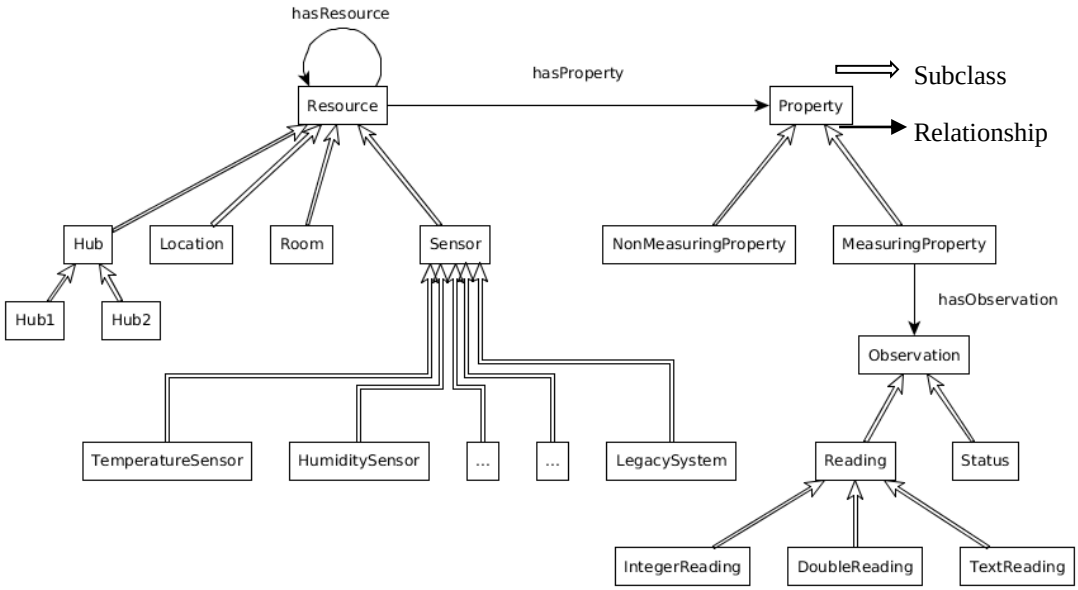

Figure 3-8 - Ontology for IoT Resource Management

- **Resource:** is a hierarchical logical representation of an IoT network component. This could be anything from the identification of the company, to a location, to a sensor device. Constraints on specific classes are defined allowing to inference on a very basic level. In this model a location/room is a purely hierarchical resource, a sensor is any resource that has a measuring property and a hub is any resource that has sub resources associated. It is also possible to specify a resource as virtual;
- **Property**: is a known pair key-value of a specific instance resource. It includes the measuring capabilities of sensors in this concept and distinguish them from non-measuring properties;
- **Observation**: is any measurement reported by a measuring property. It is classified observations as sensor readings and status events.

ss holds a graphical representation of the proposed ontology with the specific classification. The ontology is dynamic enough to handle the classification of new resource types. Conceptually, the information kept in the resource management ontology is abstractly divided into two categories:

- General classification that concerns itself with establishing the properties that are shared among entities of the same class (typically represented as subclasses in the ontology);
- Instantiation, which concerns itself with describing the physical deployment of the network and keeping the individual properties of each entity (typically represented as individuals in the ontology).

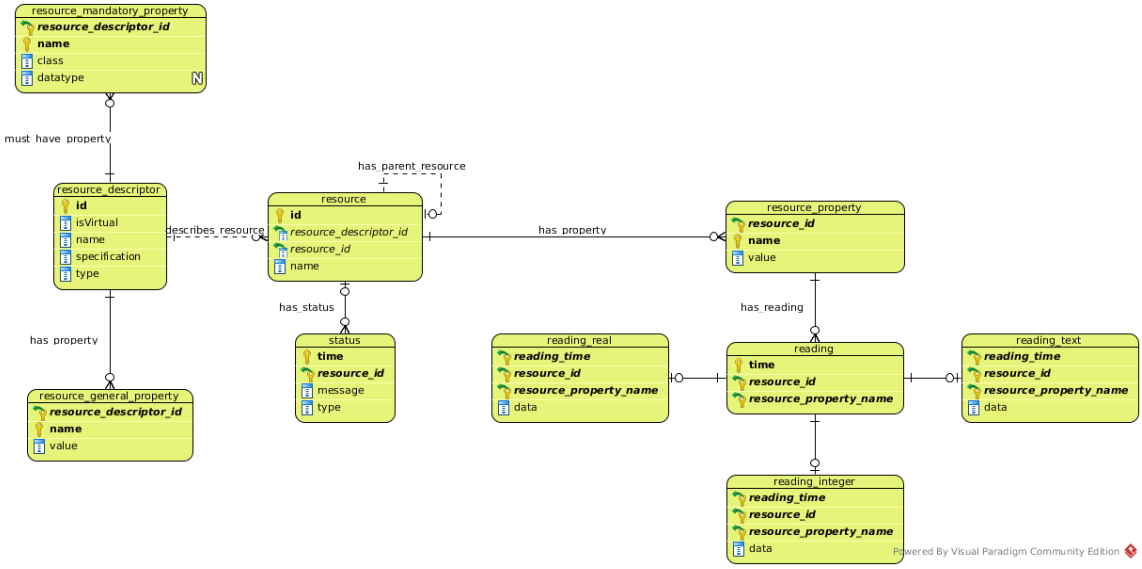

Figure 3-9 - Database model for handling dynamic ontology generation

Thanks to the ontology instantiation, the data extraction of knowledge is possible combining a well-defined set of rules and the Entity-Relationship model, **Figure 3-9**, describes a very generic model for representing the data storage associated with an IoT network. Concerning the virtualization functionality, is possible to divide the model in two sections, one where information will be maintained, described with the entities with descriptor suffixed and its associated properties, and all other entities that are concerned with the instantiation of the sensor network and observed measurements.

Resource Virtualization Functionality

The process of resource virtualization involves the resource management module and the backend modules. **Figure 3-10** shows the high level sequential flow of interaction between different components for addition of new resource. The process of adding a new resource is initiated at the Resource Management UI by utilizing services provided by Resource Management module. The details of steps in between are explained in **Figure 3-11**.

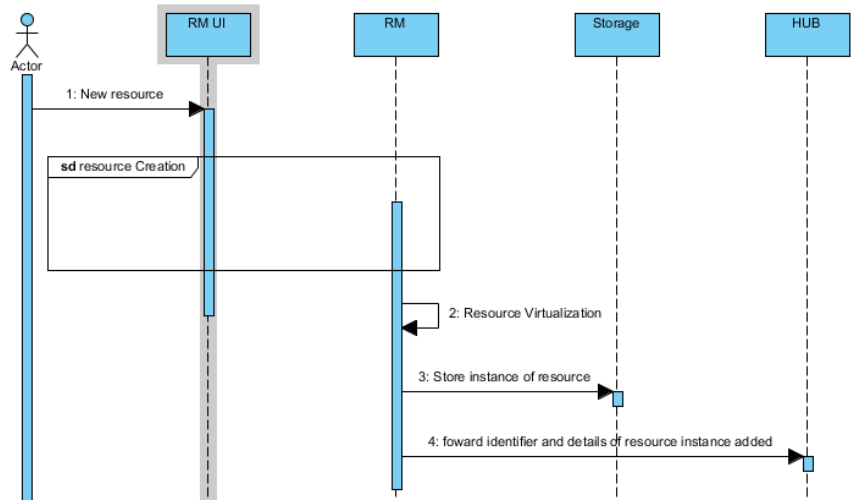

Figure 3-10 - Sequence diagram for addition of new resources

When RM receives the details of the resource to be added, a virtual instance of the resources is created, which can henceforth be used for identification and interaction with the actual resource, irrespective of the type of actual resource. The identification and necessary details, which include access address/port, data filtering constraints, communication protocol, source data format etc. are forwarded to the IoT hubs, so that the hubs can perform identification, authentication and preliminary data pre-processing over the collected data before sending to the DCF. The sequence of steps for creating new resource is as shown in **Figure 3-11**. There are two types of virtualized resources i.e. simple and mashup. Simple resource virtualization is dependent on the simple case as shown in the figure and is dependent on the details provided by the user via the RM UI. While, the mash up virtualization makes use of existing virtualized resources for creating complex resources by using data mashup rule over the data to be collected by two or more resources. Note that even though not shown in the sequence diagrams, the virtualized resources can be edited/deleted/paused etc. to make adapt or change in the functional and behavioural features of the resource. For instance, the requirement on constraints and rule for data handling and event processing can change over time, which the user with the rights can update over the existing virtual instance of the resource.

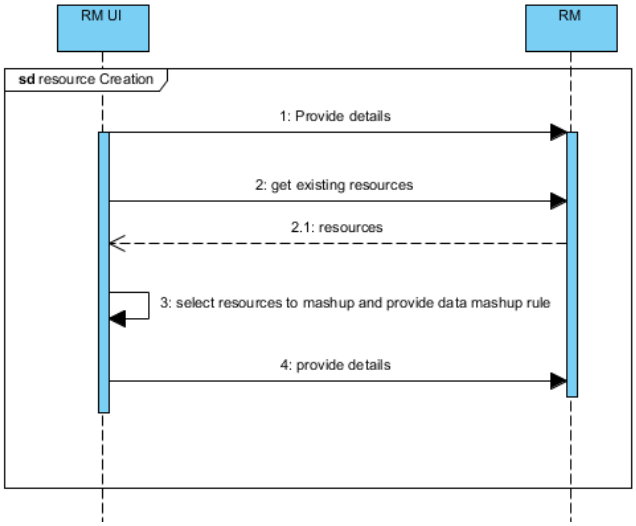

Figure 3-11 - Sequence diagram for resource creation

3.2.3.2 *Application Programming Interface (API)*

- In order to expose the main functionalities of the resource management module, a public application programming interface(API) describing the expected behaviour is provided. The resource management API is design as a webAPI based on the concept of the RESTful concept where all his methods are accessible via standard HTTP methods (GET, POST, PUT, DELETE), it is also defined that the main message exchange language is JSON[25] since it is a more compact and native in the technologies selected. In order to simplify the implementation and to provide a better description to the consumer the webAPI is divided in four different paths/routes:The communication path, **Figure 3-12**, composed by two services one for the interface (‘/com/{hubId}’) where the resource management send the configuration file to the hub and the other concerning the communication link to each hub;

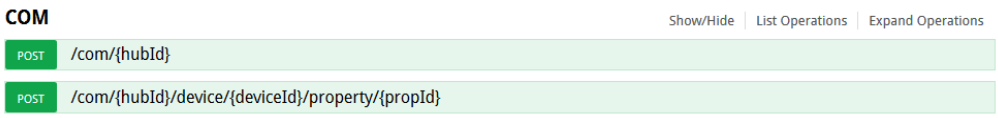

Figure 3-12 - Communication API

- The info API, **Figure 3-13**, describes and provides all the information concerning the hubs, devices and companies that already is in the knowledge base. This way is possible to build knowledge in a dynamic way, allowing the system to scale;

[25] http://www.json.org/

Figure 3-13 - Knowledge API

- The company path,
- **Figure 3-14**, where all the operations to create, update, delete and search concerning the management of the device network, hubs and devices;

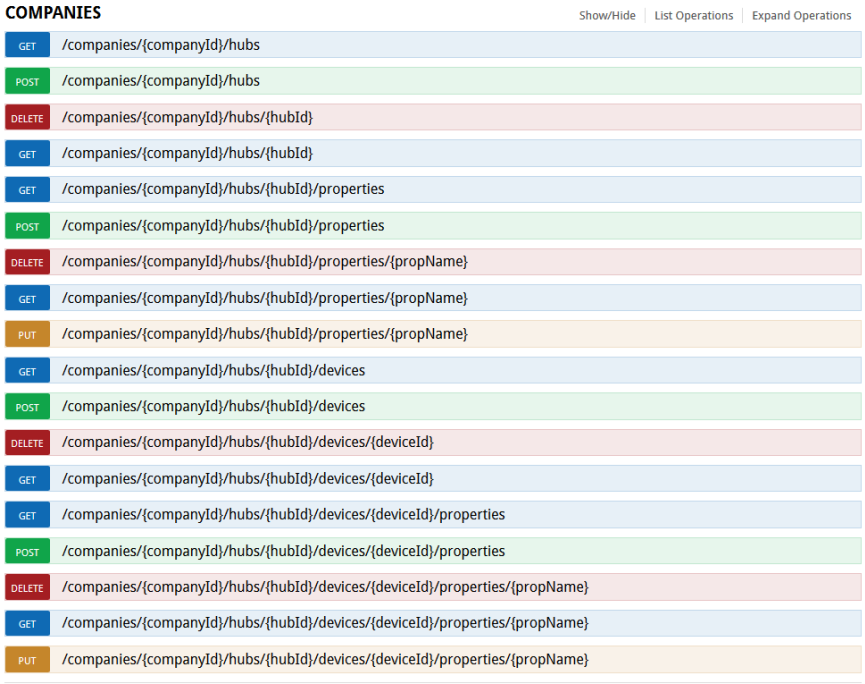

Figure 3-14 - Company API

- The monitoring API, **Figure 3-15**, where all the monitoring methods should be described. In this dissertation there is only one method described, allowing to count how many hubs, devices and message are associated to a specific company.

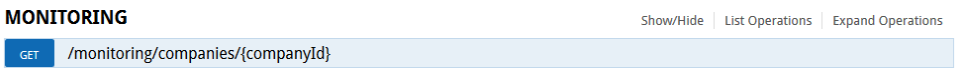

Figure 3-15 - Monitoring API

The use of this webAPI paths/routes enables to distinguish the method operation without the need to read the description, to re-design the UI module according with the needs of each

manufactory without the changing the core functionalities and also a clear separation for extend the monitoring capabilities.

3.2.3.3 User Interface

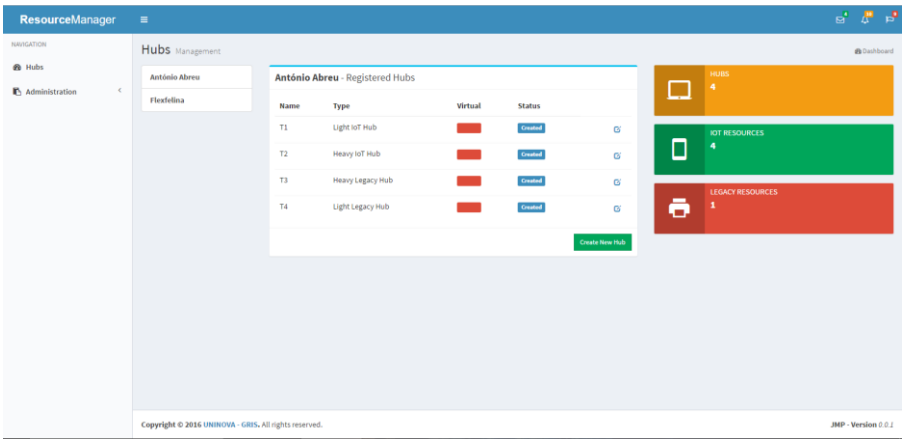

Figure 3-16 - Main Management Interface

User interfaces provide a clear environment for humans to see and use functionalities allowing to being able to understand and validate the events that their systems handle. With this in mind it was developed a simple interface with the main functionalities of the resource management.

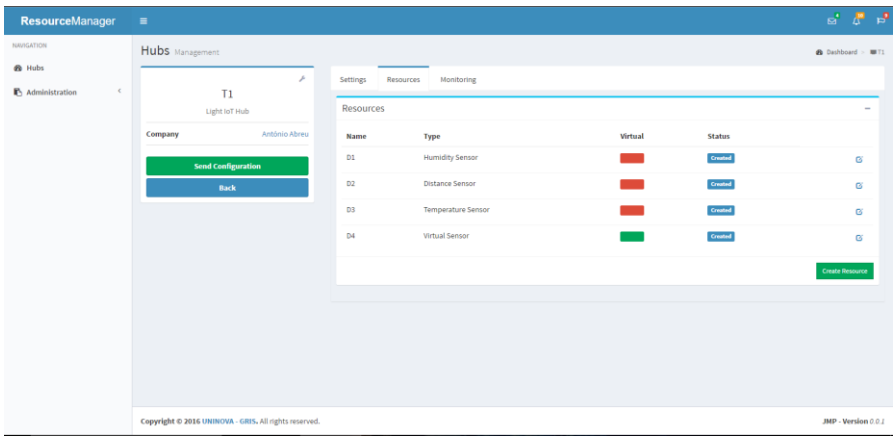

Figure 3-17 - Hub Management Interface

Figure 3-16 depicts the list of hubs that each company has with information regarding which type and the information regarding how many hubs, IoT devices a company have. Also in this interface is possible to create new hubs associated to a company. **Figure 3-17** and **Figure 3-18** the management view is shown, here is possible to manage all the properties of a hub or

device, check their monitoring information and in the case of the hub associate new devices to it.

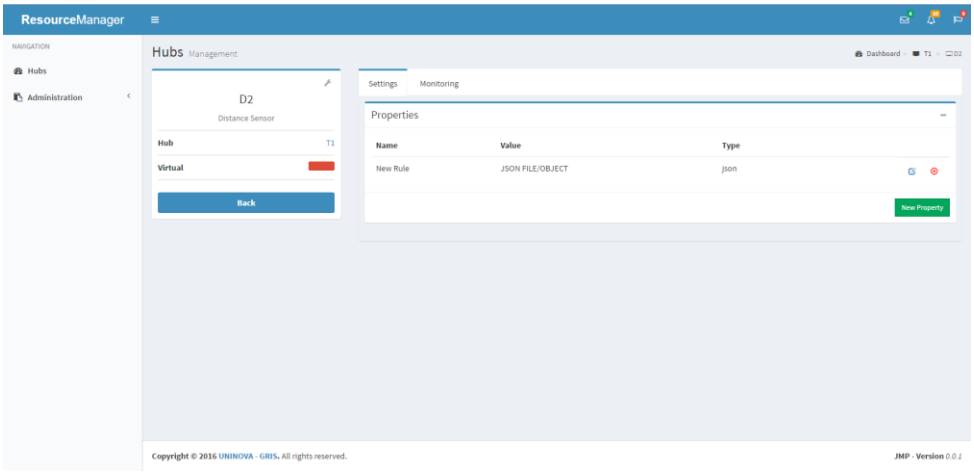

Figure 3-18 - Device Management Interface

At last **Figure 3-19**, shows the interface that allow to manage the virtual device; here is where it is possible to define the aggregation rule for the cep and define what devices will be associated inside this logical representation.

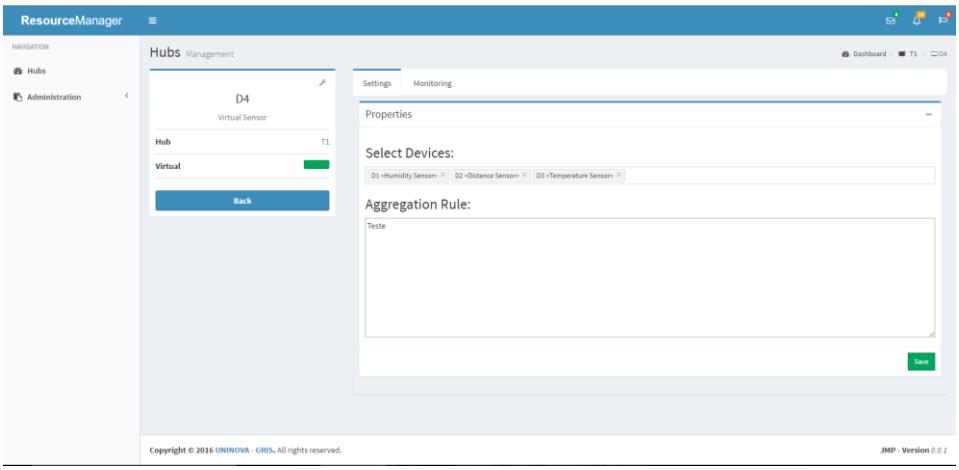

Figure 3-19 - Virtual Device Management Interface

The user interface (UI), presented in this sections, is used in the following section to validate the API provided for enabling IoT management capabilities and also the functional and non-functional requierment that this module should support. It is also important to denote the lack of the resource management monitoring capabilty, but in order to provide them it was needed to implement the CM and the cloud broker to handle data to monitoring.

3.3 Validation of the solution

In this section the proof of concept implementation testing will be addressed, validating the requirements and functionalities of part the system. Testing is the process of trying to find errors in a system implementation by means of experimentation. The experimentation is usually carried out in a special environment, where normal and exceptional use is simulated. The aim of testing is to gain confidence that during normal use the system will work satisfactory: since testing of realistic systems can never be exhaustive, because systems can only be tested during a restricted period of time, testing cannot ensure complete correctness of an implementation. It can only show the presence of errors, not their absence. Also it is important understand and gather feedback from the maturity and complexity of the solution proposed from the scientific and the necessities and objectives desired by the industrial communities. This way it is assured that the platform will have a better compliance in the Industrial market.

In the next sections will be presented TTCN methodology and the tests definitions applied to RM proof-of-concept implementation. In the last section a scientific and industrial context validation is presented.

3.3.1 The Three and Tabular Combined Notation (TTCN)

The Tree and Tabular Combined Notation is a test notation standardized by the ISO/IEC 9646-1 (ISO/IEC, 1994), that concerns the specification of tests for communication systems developed within the framework of standardized conformance testing (ETSI, 2009). In the TTCN, tests behaviour is defined by a sequence of events and specified through tables which are divided in general description, constraints, behaviour and verdict. This sequence can be approached as a tree, containing branches of actions based on the evaluation of the system output after one or a series of executed events, which can be of one of two types:

- Actions, which are preceded by an exclamation point before its brief description, and represent actions performed on the System Under Test (SUT);
- Questions that are preceded by an interrogation point, and represent evaluations of the output of the SUT after one or more actions are completed.

Since the answer can be positive or negative, multiple questions can exist at the same indentation level, covering all possible outputs of the system. Once the TTCN test table is complete, three verdicts based on the sequence of events and the outputs of the system are possible: *Success*, *Failure* or *Inconclusive*. To better illustrate the testing procedure, **Table 3-1** presents a simplified example describing an evaluation description for a phone call.

Table 3-1 - TTCN Table Example

TTCN table test example. **Test Case**		
Test:	Basic Phone Call Connection	
Group:	N/A	
Purpose:	Check if a phone call can be established.	
Comment:	N/A	
Behaviour	**Constraints**	**Verdict**
! Pick up headphone		
? Dialling tone		
! Dial Number		
? Calling tone		
? Connected line		
! Hung up headphone		Success
OTHERWISE		Failure
? Busy tone		
! Hung up headphone		Inconclusive
OTHERWISE		Failure
? Dialling tone absent		Failure

Thus, as one can observe, the table test is structured in a hierarchical manner similar to an IF-ELSE programming statement, and can be textually read as:

1. The user picks up the headphone;
2. Tests if the dialling tone is present;
3. If the dialling tone is present, then the user dials the phone number. Otherwise, if the dialling tone is absent, the verdict is a Failure, due to the inability of establishing a phone call;
4. If there is a calling tone after dialling the number, then the user may test if the line is effectively connected;
5. 5If the line is connected, the user may hang up the headphone and the verdict is set as Success on establishing a phone call, otherwise the verdict is Failure;
6. If the dialling tone is not heard, but a busy tone is present instead, then the user may hang up the headphone and the verdict is set as Inconclusive;
7. Finally, if none of the tones corresponds to the calling or busy tone, then the verdict is defined as Failure on establishing a phone call.

3.3.2 Functional Tests

To address the functional testing of the RM proof-of-concept implementation, in this section, according with the TTCN methodology, functional tests are presented in order to validate if the main functionalities described in section 3.1.2.1 are present and respected in the implementation of the resource management. The validation of the functionalites was achived through the combination of tests, or by a single test. The contruction and management of virtual resources functionality is validated with two tests, as well as the creation of output data streams based on the inputs from the IoT hub. The single test validates the discovery of all resources.

3.3.2.1 Test 1: Create Device

Table 3-2 shows the test where the user creates a new device. This interactions concist in opening the RM UI and check if the device that the user wants to create already exists, if so it will throw a warnig saying that the device already exist otherwise it will be created a new one.

Table 3-2 - TTCN Create Device

Construction and management of virtual resources		
Test:	Create Device	
Group:	N/A	
Purpose:	Check if a device can be created.	
Comment:	N/A	
Behaviour	**Constraints**	**Verdict**
! Open RM UI		
? Device not listed		
! Create Device	Success	
OTHERWISE	Failure	

3.3.2.2 Test 2: Add Property to Device

In the add property to device test, Table 3-3, the user will open the RM UI, search for the device to add the new property. When it is found the user opens the device view adding a new property. This operation has a constrain regarding the name of the property this means that each device is only allow to have one property with that single name.

Table 3-3 - TTCN Add Property to Device

Construction and management of virtual resources	
Test:	Create property
Group:	N/A
Purpose:	Check if a property of a device can be created and is well associated.
Comment:	N/A

Behaviour	Constraints	Verdict
! Open RM UI		
? Found device		
! Open device view		
! Create new	Success	
OTHERWISE	Failure	

3.3.2.3 Test 3: Configuration of the CM

In Table 3-4 is described the steps effectuated to send the configuration from one IoT Hub. This operation is done in the device view where by clicking a button you send the configuration to the device address, added in the ENDPOINT property, and if the communication is correctly done, a notification is presented to the user in the RM. The conclusion of this test was not possible since the IoT hub was not developed under this implementation.

Table 3-4 - TTCN Configuration of the CM

Creation of output data streams based on the inputs from the IoT hub		
Test:	Configuration of the CM	
Group:	N/A	
Purpose:	Check if RM pass-through the configuration of the CM	
Comment:	N/A	
Behaviour	**Constraints**	**Verdict**
! Pick up configuration in the device view		
? Try to communicate		
! send to property (endpoint)		
? Receive confirmation	Success	
OTHERWISE	Failure	
! Notify RM UI		

3.3.2.4 Test 4: IoT Data Exchange with CM

Table 3-5 also describes a test where is possible to check the steps necessary to the system to executed a data exchange with the IoT Hub. However, since the IoT hub implementation was out of the scope of this dissertation, it was impossible to conclude the test and validate the create of data streams feature.

Table 3-5 - TTCN IoT Data Exchange with CM

Creation of output data streams based on the inputs from the IoT hub	
Test:	IoT Data exchange

Group:		N/A
Purpose:		Check if RM receives and stores data incoming from the IoT devices in the CM.
Comment:		N/A
Behaviour	**Constraints**	**Verdict**
? Received data		
! is stored		Success
OTHERWISE		Failure

3.3.2.5 Test 5: Search Device

In the add search device test, **Table 3-6**, the user will open the RM UI, search for a device. When it is found the basic information of the device is show to the user.

Table 3-6 - TTCN Search Device

Discovery of all the virtual Resources		
Test:		Search Device
Group:		N/A
Purpose:		Find a device
Comment:		N/A
Behaviour	**Constraints**	**Verdict**
! Open RM UI		
? User search device		
! User Found device		
! Open Device Detail View		Success
OTHERWISE		Failure

3.3.2.6 Test Results

Table 3-7 - Tests Analysis

Test Name	% Success	Status
Create Device	100%	Approved
Add Property to Device	100%	Approved
Communicate with CM	73%	Need further communication tests and some development in the standard way of do it.
IoT Data exchange with CM	66%	Need further communication test and define the meta-data to be exchange.
Search device	100%	Approved

Through **Table 3-7** is possible to conclude that the device management features, such as searching, creating and editing capabilities are implemented and according with the proof of concept description. It was not yet possible to archive a verdict regarding the real-time monitoring and communication due to some issues with the development of the IoT Hub (out of the scope of the selected proof of concept implementation). Finally, the functionally concerning the. Run-time adaptation of security polices envisaged in the scope was not tested because its implementation is still work in progress.

3.3.3 Scientific Validation

During the research and developments here presented, one scientific publication was made to further contribute to the validation of the proposed architecture. This publication was made in the I-ESA Conference being well received by the Enterprise Interop community and is acknowledged in the C2NET project dissemination. The publication is described as follow:

- Carlos AGOSTINHO, Joaquim PEREIRA, Jean-Pierre LORRE, Sudeep GHIMIRE, Yazid BENAZZOUZ: *"A Distributed Middleware Solution for Continuous Data Collection in Manufacturing Environments"* (AGOSTINHO et al. 2016).

3.3.4 Industrial Validation

Industrial acceptance and validation is equally important as the acceptance by scientific peers. If industry does not approve the concept and the results, most likely they will never be used.

This dissertation is contributing to the developments proposed in the project C2NET project, that is a large research project involving 20 partners across the European Union and founded by the European Commission since 1 January 2015 for the duration of 3 years. The goal of project is the creation of cloud-enabled tools for supporting the SMEs supply network optimization of manufacturing and logistic assets based on collaborative demand, production and delivery plans (see **Figure 3-20)**. Many discussions with the industrial partners have been carried along, both physical and teleconference meetings, involving them in the definition of the concept here proposed, hence providing industrial pilots with data collection functionalities and virtualized resource management. The metalworking SME's case identified in **Figure 3-20** has been a real test bed for the developments.

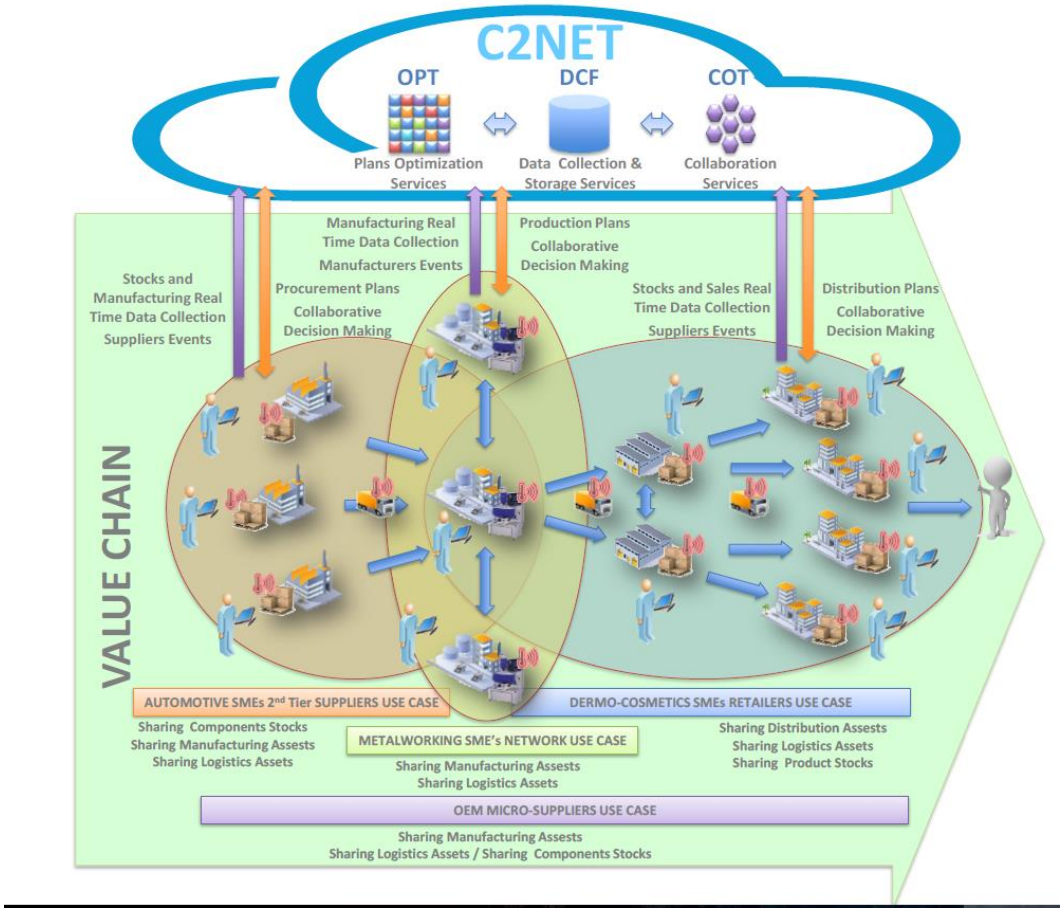

Figure 3-20 - C2NET Goals

3.3.5 Hypothesis Validation

In section 1.3 the hypothesis of this dissertation was defined, and the objectives of this work were presented. The question made in the hypothesis was partial achieved during this dissertation since it was not possible yet to test the full scope of the framework. However, through the development of the resource management, the nervous system, is possible to realize that the framework is one step closer to the objective to provide real-time data collection to the cloud capabilities to the industry, allowing to envision to achieve more complex features such as collaborative networks, real-time monitoring and production line optimization. However, the reaming aspects of the proposed solution and hypothesis have scientifically accepted in the enterprise interoperability community (section 3.3.3) and thanks to fruitful discussion with the partners of c23net project (section 3.3.4).

4 Conclusions and Future Works

4.1 Conclusions

This dissertation aims to present a framework architecture that enables real time data collection, especially concerning the connection to IoT devices and the adoption of Industry 4.0 paradigm. For that, at first a set of concepts related with real world data collection such as "Internet of Things", "IoT Models", "Service in IoT", "Middleware for IoT", "Cloud Computing" and "Fog Computing" are analysed with the purpose to propose a conceptual solution addressing the hypothesis. The main objective was to provide an IoT-based continuous data collection framework from supply network resources. It considers data representation, IoT communication protocols, efficient data storage and user interaction as main requirements. For that the solution was structured in four big modules each of them in charge of handle different task and functionalities:

- **Company Middleware (CM)** is the gateway that allow devices to exchange data with the Data Collection framework. This data can be configurations from the hub/device or data collected by the device from the physical world;
- **Data Collection Framework (DCF)** is design to support all the management functionalities related with the device management and also with the operations, such as detection patterns over the data collected;
- **User Interface** is the module that centralizes all the user experience with the framework, it allows an interaction with the functionalities provided by each module of the framework;

- **Backend module** that provides a centralized storage capability, through a database or a knowledge base system.

Even though a full proposal for a data collection framework is discussed in the conceptual solution, a full implementation was impossible to archive during the time of this dissertation, so it was decided that the implementation should focus on the nervous system of the framework, i.e., the resource management inside the DCF module. Following this decision, the document presents a division of the resource management implementation and validation, where is discussed the model used to store and infer knowledge from the data collected from the IoT devices, the main functionalities and API that the RM should have, and the user interface used for validate the functionalities provided by the RM. This validation consists in a series of functional test alongside with a scientific and industrial.

During the time of the dissertation, many discussions within the C2NET industrial and scientific partners have been carried along, this led to the understanding that although the research discussed in this dissertation is already well addressed by the scientific community, the SME industrial community is some steps behind, where concepts such as IoT and Cloud Computing are still being understood and their advantages comprehended. They are open to test in a closed environment but the lack of standards and regulation, special related with the security, in the IoT and Cloud computing paradigm, are one of the key factors that are leading to a mistrusted relationship with the industries at a more open scale. Hence, is possible to conclude that the framework presented in this dissertation is a “must” if industry was to move towards the industry 4.0 paradigm and the notion of IoT, where all objects are interconnected and contributing to an optimized working environment. However, due to some degree of mistrust, the functions designed to take advantage of Cloud environments, enabling a highly scalable, available and fault-tolerant solution, and to provide a toolkit that supports real time collaborative manufacturing and monitoring capabilities to the production line, are not yet ready to be embraced. Nonetheless, it is possible to say that this dissertation also contributes to helping industries move one step further, understanding how IoT and cloud technologies can enhance their operations by using a production process example to help recognise the benefits of its proof of concept.

4.2 Future Works

With the adoption of these types of solutions, that allow real time data collection, the manufacturing paradigm will forever change. With these changes will arise huge challenges, since industries will demand more intelligent systems. It is from the author's belief that in the future, areas such as artificial intelligence and big data real time data analytics will be key complements for any data collection framework. Also, and in a short-term perspective, it is necessary to design and implement standards for security and interoperability in the IoT paradigm, as this will provide the necessary means for a faster acceptance of the industry regarding this type of solutions. This is a key aspect of the future work, in order to provide to the framework means to respond to the ever changing industrial demand.

References

AGOSTINHO, C. et al., 2016. A Distributed Middleware Solution for Continuous Data Collection in Manufacturing Environments. *I-ESA 2016, 8th International Conference, Interoperability For Enterprise Systems and Applications*.

Agostinho, C. et al., 2015. Model-driven Service Engineering Towards the Manufacturing Liquid-sensing Enterprise. *Proceedings of the 3rd International Conference on Model-Driven Engineering and Software Development, ESEO, Angers, Loire Valley, France, 9-11 February, 2015.*, pp.608–617.

Anon, 2016. OASIS SOA Reference Model. Available at: https://www.oasis-open.org/committees/soa-rm/faq.php [Accessed April 30, 2016].

Anon, 2011. Semantic Sensor Network XG Final Report. Available at: https://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/#Intent_and_Process [Accessed April 30, 2016].

Atzori, L., Iera, A. & Morabito, G., 2010. The Internet of Things: A survey. *Computer Networks*, 54(15), pp.2787–2805. Available at: http://linkinghub.elsevier.com/retrieve/pii/S1389128610001568.

Baldoni, R. et al., 2009. An embedded middleware platform for pervasive and immersive environments for-all. *2009 6th IEEE Annual Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks Workshops, SECON Workshops 2009*, (May 2016).

Bhardwaj, S., Jain, L. & Jain, S., 2010. Cloud Computing : a Study of Infrastructure As a Service (Iaas). *International Journal of Engineering*, 2(1), pp.60–63. Available at: http://ijeit.org/index_files/vol2no1/CLOUD COMPUTING A STUDY OF.pdf.

Bonomi, F. et al., 2014. Big Data and Internet of Things: A Roadmap for Smart Environments. *Studies in Computational Intelligence*, 546, pp.169–186. Available at: http://link.springer.com/10.1007/978-3-319-05029-4.

Bonomi, F. et al., 2012. Fog Computing and Its Role in the Internet of Things. *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pp.13–16. Available at: http://doi.acm.org/10.1145/2342509.2342513\npapers2://publication/doi/10.1145/2342509.2342513.

Chaqfeh, M. a. & Mohamed, N., 2012. Challenges in middleware solutions for the internet of

things. *Proceedings of the 2012 International Conference on Collaboration Technologies and Systems, CTS 2012*, pp.21–26.

De Chaves, S.A., Uriarte, R.B. & Westphall, C.B., 2011. Toward an architecture for monitoring private clouds. *IEEE Communications Magazine*, 49(12), pp.130–137.

Cisco Systems, 2016. Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are. *Www.Cisco.Com*, p.6. Available at: http://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf.

Cloud Computing Types, 2016. Best Cloud Deployment Models & Cloud Computing Pricing Model. Available at: http://cloudcomputingtypes.com/cloud-computing-models/ [Accessed May 22, 2016].

Commission, E., 2012. FInES Research Roadmap 2025.

Compton, M. et al., 2012. Web Semantics : Science , Services and Agents on the World Wide Web The SSN ontology of the W3C semantic sensor network incubator group. *Web Semantics: Science, Services and Agents on the World Wide Web*, 17, pp.25–32. Available at: http://dx.doi.org/10.1016/j.websem.2012.05.003.

Cox, S.J.D., 2016. Ontology for observations and sampling features, with alignments to existing models. *Semantic Web*, Preprint(Preprint), pp.1–18. Available at: http://content.iospress.com/articles/semantic-web/sw214\nhttp://www.semantic-web-journal.net/content/ontology-observations-and-sampling-features-alignments-existing-models-0.

Dillon, T., Wu, C. & Chang, E., 2010. Cloud Computing: Issues and Challenges. *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, pp.27–33. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5474674.

E. Michael Maximilien et al., 2009. Toward cloud-agnostic middlewares. , (May), pp.619–626. Available at: http://doi.acm.org/10.1145/1639950.1639957.

EC, I.-A.P., 2013. Internet of Things - Architecture — IOT-A: Internet of Things Architecture. Available at: http://www.iot-a.eu/public.

European Parliamentary Research Service, 2015. Industry 4.0. Digitalisation for productivity and growth. , (September), p.10.

Evans, P. & Annunziata, M., 2012. Industrial internet: Pushing the boundaries of minds and machines. *General Electric White Paper*.

Felic, A. et al., 2014. First OSMOSE Models and Architecture.

Gartner's, 2015. Gartner's 2015 Hype Cycle for Emerging Technologies Identifies the Computing Innovations That Organizations Should Monitor. Available at: http://www.gartner.com/newsroom/id/3114217 [Accessed February 3, 2016].

Gershenfeld, N., Krikorian, R. & Cohen, D., 2004. *The Internet of things.*, Available at: http://www.cba.mit.edu/docs/papers/04.10.i0.pdf.

Gubbi, J., Buyya, R., Marusic, S., et al., 2013. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7), pp.1645–1660. Available at: http://dx.doi.org/10.1016/j.future.2013.01.010.

Gubbi, J., Buyya, R. & Marusic, S., 2013. Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions. , (1), pp.1–19.

Hermann, M., Pentek, T. & Otto, B., 2015. Design Principles for Industrie 4.0 Scenarios: A Literature Review. , (1).

Huhns, M. & Singh, M., 2005. Service-oriented computing: Key concepts and principles. *Internet Computing, IEEE*, 9(1), pp.75–81. Available at: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1407782.

IOT-A, 2011. Internet-of-Things Architecture Project Deliverable D1 . 2 – Initial Architectural Reference Model for IoT. *Architecture*, (257521), pp.1–97.

Jacobs, D., 2005. Enterprise software as service. *Queue*, 3(6), p.36.

Keller, E. & Rexford, J., 2010. The "Platform as a service" model for networking. *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, (Section 3), p.4. Available at: http://dl.acm.org/citation.cfm?id=1863133.1863137.

Kohli, D. & Silicon, I.B.M., 2008. Understanding Event Driven Architecture. , (1672).

Kui, R., Cong, W. & Qian, W., 2012. Security Challenges for the Public Cloud. *Internet Computing, IEEE*, 16(1), pp.69–73.

MacKenzie, C.M. et al., 2006a. Reference model for service oriented architecture. *Public Review Draft 2*, (October), pp.1–31. Available at: http://lists.oasis-open.org/archives/soa-rm-editors/200511/pdf00001.pdf\nhttp://www.mendeley.com/research/reference-model-for-service-oriented-architectures/\nhttp://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm.

MacKenzie, C.M. et al., 2006b. Reference model for service oriented architecture. *Public Review Draft 2*, (October), pp.1–31.

Manyika, J. et al., 2015. The Internet of Things: Mapping the value beyond the hype. *McKinsey Global Institute*, (June), p.144.

Maréchaux, J., 2006. Combining service-oriented architecture and event-driven architecture using an enterprise service bus. *IBM Developer Works*, (April), pp.1–8. Available at: http://www.immagic.com/eLibrary/ARCHIVES/GENERAL/IBM/I060328M.pdf.

Marques-Lucena, C. et al., 2016. Process Modelling Approach for the Liquid-Sensing Enterprise. *I-ESA'16 Proceedings*, (April), pp.1–12.

Mell, P. & Grance, T., 2011. The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology. *National Institute of Standards and Technology, Information Technology Laboratory*, 145, p.7. Available at: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

Michelson, B.M., 2006. Event-driven architecture overview. *Patricia Seybold Group*, 2.

Murray, R.W., 1999. The scientific method. *Analytical chemistry*, 71(5), p.153A. Available at: http://dx.doi.org/10.1021/ac990210y.

OpenGroup, 2013. Cloud Computing for Business : What is Cloud? Available at: http://www.opengroup.org/cloud/cloud/cloud_for_business/what.htm [Accessed May 22, 2016].

Razzaque, M.A. et al., 2015. Middleware for Internet of Things: a Survey. *Internet of Things Journal, IEEE*, PP(99), p.1.

Santucci, G., Martinez, C. & Vlad-câlcic, D., 2012. The Sensing Enterprise. , pp.1–14.

Schafersman, S.D., 1997. An Introduction to Science: Scientific Thinking and the Scientific Method. *Geology*, pp.1–9. Available at: http://www.muohio.edu/~schafesd/documents/intro-to-sci.htmlx.

Soma Bandyopadhyay et al., 2011. Role Of Middleware For Internet Of Things: A Study.

International Journal of Computer Science & Engineering Survey, 2(3), pp.94–105.

Stojmenovic, I. & Wen, S., 2014. The Fog Computing Paradigm: Scenarios and Security Issues. *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems*, 2, pp.1–8. Available at: https://fedcsis.org/proceedings/2014/drp/503.html.

Thornton, P., 2010. The Global Manufacturing Sector : Current Issues. *Chartered Institute of Management Accountants*, pp.1–11.

Vermesan, O. et al., 2009. Internet of Things Strategic Research Roadmap. *Internet of Things Strategic Research Roadmap*, pp.9–52. Available at: http://sintef.biz/upload/IKT/9022/CERP-IoT SRA_IoT_v11_pdf.pdf.

W3C, 2015. IoT-Lite Ontology. *W3C*. Available at: https://www.w3.org/Submission/2015/SUBM-iot-lite-20151126/.

Zhang, H. et al., 2009. Intelligent workload factoring for a hybrid cloud computing model. *SERVICES 2009 - 5th 2009 World Congress on Services*, (PART 1), pp.701–708.

Zhang, Q., Cheng, L. & Boutaba, R., 2010. Cloud computing: State-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1), pp.7–18.

Annex A – Cloud Service Providers: Service List

Table 4-1 - Bluemix Services

Name:	Bluemix
Provider:	IBM
Type:	Paid
Service Model:	PaaS, SaaS, IaaS
Service Name	**Tools/Services**
Compute	• Virtual Server • Containers • CF applications • OpenWhisk
Network	• Content Delivery • Virtual Private Network
Storage	• Object Storage
Data and Analytics	• Apache Spark • Big Insights for Apache Hoops • Cloudant NoSQL DB • dashDB • DataWorks • Elasticsearch • Geospatial Analytics • IBM DataStage on Cloud • IBM DB2 on Cloud • IBM Graph • IBM Master Data Management Cloud • Insights for Twitter • Mongo DB • PostreSQL

	• Predictive Analytics • Redis • Streaming Analytics • Weather Company Data • ClearDB MySQl Database • ElephantSQL • Namara.io Catalog • Redis Cloud
Watson	• AlchemyAPI • Conversation • Document Conversion • Language Translation • Natural Language Classifier • Personality Insights • Retrieve and Rank • Speech to Text • Text to Speadch • Tone Analyzer • Tradeoff Analytics • Visual Recognition • Cognitive Commerce • Cognitive Graph • Cognitive Insights
Integration	• Secure Gateway • Service Discovery • Service Proxy • Rocket Mainframe Data
DevOps	• Active Deploy • Auto-Scaling • Availability Monitoring • Delivery Pipeline • Globalization Pipeline • IBM Alert Notification • Monitoring and Analytics • Track & Plan • BlazeMeter • jKool • Load Impact • New Relic
Security	• Access Trail • Application Security on Cloud • Key Protect • Single Sign On

	• Adaptive Security Manager
Mobile	• Mobile Analytics • Mobile Application Content Manager • Mobile Client Access • Mobile Foundation • Mobile Quality Assurance • Push Notifications • Kinetise • Testdroid Cloud • Twiio
Internet of Things	• Context Mapping • Driver Behaviour • Internet of Things Platform • IoT for Electronics • Flowthings.io • IQP IoT Code-Free App Development

Table 4-2 - Microsoft Azure Services

Name:	Azure
Provider:	Microsoft
Type:	Paid
Service Model:	PaaS, SaaS, IaaS
Service Name	**Tools/Services**
Compute	• Virtual Machines • Virtual Machine Scale Sets • Azure Container Service • Functions • Batch • Service Fabric • Cloud Services • RemoteApp
Networking	• Virtual Network • Load Balancer • Application Gateway • VPN Gateway • Azure DNS • CDN • Traffic Manager • ExpressRoute
Storage	• Azure Storage • Data Lake Store • StorSimple

	• Backup • Site Recovery
Web & Mobile	• App Service • CDN • Media Services • Search
Databases	• SQL Database • SQL Data Warehouse • SQL Server Stretch Database • DocumentDB • Table Storage • Redis Cache • Data Factory
Intelligence + Analytics	• HDInsight • Machine Learning • Stream Analytics • Cognitive Services • Data Lake Analytics • Data Lake Store • Data Factory • Power BI Embedded
Internet of Things	• Azure IoT Hubs • Event Hubs • Stream Analytics • Machine Learning • Notification Hubs
Enterprise Integration	• Logic Apps • BizTalk Services • Service Bus • API Management • StorSimple • SQL Server Stretch Database • Data Factory
Security + Identity	• Security Center • Key Vault • Azure Active Directory • Azure Active Directory B2C • Azure Active Directory Domain Services • Multi-Factor Authentication
Developer Tools	• Visual Studio Team Services • Azure DevTest Labs • Visual Studio Application Insights • API Management

	• HockeyApp • Developer tools and SDKs • Xamarin • Storage Explorer
Monitoring & Management	• Microsoft Azure portal • Azure Resource Manager • Visual Studio Application Insights • Log Analytics • Automation • Backup • Site Recovery • Scheduler • Traffic Manager

Table 4-3 - Google Cloud Services

Name:	Google Cloud
Provider:	Google
Type:	Paid
Service Model:	PaaS, SaaS
Service Name	**Tools/Services**
Compute	• Compute Engine • App Engine • Container Engine • Container Registry • Cloud Functions
Storage and Databases	• Cloud Storage • Cloud SQL • Cloud Bigtable • Cloud Datastore • Persistent Disk
Networking	• Cloud Virtual Network • Cloud Load Balancing • Cloud Content Delivery Network • Cloud Interconnect • Cloud Domain Name System
Big Data	• BigQuery • Cloud Dataflow • Cloud Dataproc • Cloud Datalab • Cloud Pub/Sub • Genomics

Machine Learning	• Cloud Machine Learning Platform • Vision API • Speech API • Natural Language API • Translate API
Management Tools	• Strackdrive Overview • Monitoring • Logging • Error Reporting • Trace • Debugger • Deployment Manager • Cloud Console • Cloud Shell • Cloud Mobile App • Billing API • Cloud APIs
Developer Tools	• Cloud SDK • Deployment Manager • Cloud Source Repositories • Cloud Endpoints • Cloud Tools for Android Studio • Cloud Tools for IntelliJ • Cloud Tools for PowerShell • Cloud Tools for Visual Studio • Google Plug-In for Eclipse • Cloud Test Lab
Identity & Security	• Cloud Identity & Access Management • Cloud Resource Manager • Cloud Security Scanner • Cloud Platform Security Overview

Table 4-4 - Digital Ocean Services

Name:	DigitalOcean
Provider:	DigitalOcean
Type:	Paid
Service Model:	IaaS
Service Name	**Tools/Services**
Compute	• Droplets
Storage	• Block Storage

Table 4-5 - Amazon Cloud Services

Name:	AWS
Provider:	Amazon
Type:	Paid
Service Model:	PaaS, SaaS
Service Name	**Tools/Services**
Compute	• Virtual Servers • Containers • 1-Click Web App Deployment • Event-Driven Compute Functions • Auto Scaling • Load Balancing
Networking	• Virtual Private Cloud • Direct Connections • Load Balancing • DNS
Storage & Content Delivery	• Object Storage • CDN • Block Storage • File System Storage • Archive Storage • Data Transport • Integrated Storage
Database	• Relational • Database Migration • NoSQL • Caching • Data Warehouse
Analytics	• Business Intelligence • Data Warehouse • Machine Learning • Streaming Data • Elasticsearch • Hadoop • Data Pipelines
Enterprise Applications	• Desktop Virtualization • Email & Calendaring • Document Sharing & Feedback
Mobile Services	• Mobile Development • API Management • Identity • App Testing

	• Mobile Analytics • Development • Notifications
Internet of Things	• IoT Platform • Device SDK • Registry • Device Shadows • Rules Engine
Developer Tools	• Source Code Management • Code Deployment • Continuous Delivery
Management Tools	• Monitoring & Logs • Resource Templates • Usage & Resource Auditing • Dev/Ops Resource Management • Service Catalog • Performance Optimization
Security & Identity	• Access Control • SSL/TLS Certificates • Key Storage & Management • Identity Management • Security Assessment • Web Application Firewall
Application Services	• API Management • App Streaming • Search • Transcoding • Email • Notifications • Queueing • Workflow

Table 4-6 - OpenStack Services

Name:	OpenStack
Provider:	OpenStack
Type:	Open Source
Service Model:	SaaS
Service Name	**Tools/Services**
Object Storage	• Swift
Identity	• Keystone
Compute	• Nova

Networking	• Neutron
Block Storage	• Cinder
Image Service	• Glance

Table 4-7 - WSO2 Services

Name:	WSO2
Provider:	WSO2
Type:	Open Source
Service Model:	PaaS, SaaS
Service Name	**Tools/Services**
API Management	• API Manager • API Cloud
Integration	• Enterprise Service Bus • Data Services Server • Message Broker • Business Process Server • Business Rules Server
Analytics	• Data Analytics Server • Complex Event Processor • Machine Learner
Identity and Security Management	• Identity Server
Service and Application development	• Application Server • App Cloud • Microservices Framework for Java
Management and Governance	• Governance Registry • App Manager • Process Center
Mobile and IoT	• IoT Server • Enterprise Mobility Manager
Foundation Servers and Frameworks	• Dashboard Server • Enterprise Store • Storage Server • Carbon • Developer Studio