Pedro André da Silva Corista

Licenciado em Ciências da Engenharia

Eletrotécnica e de Computadores

IoT data processing pipeline in FoF perspective

Dissertação para obtenção do Grau de Mestre em Engenharia Eletrotécnica e de Computadores

Orientador: Ricardo Luís Rosa Jardim Gonçalves, Professor Auxiliar com Agregação, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Co-orientador: João Filipe dos Santos Sarraipa, Investigador, UNINOVA

Júri:

Presidente: [Nome do presidente do júri]

Arguente: [Nome do arguente 1]

Vogais: [Nome do vogal 1]

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro 2017

IoT data processing pipeline in FoF perspective

For my family and friends.

Acknowledgements

After an intense period of academic formation, I want to greet all the people that helped and supported me during this time, and that in their own way, contributed to this dissertation.

I would first like to thank my family, specially my parents and brother for the constant support that provided me, either in the good or hard academic times. The motivation that they provided granted me the confidence to grow further.

To my advisor, Dr. Ricardo Gonçalves and my co-advisor Dr. João Sarraipa for giving me the opportunity of working in a GRIS's project and for believing in me and in my capabilities. The door to João's office was always open whenever I had a question about my research, or when I got lost in the writing process.

To all the people that work at GRIS, for supporting me great and being always willing to help. The environment and counselling that they provided, enlightened me during several of my dissertation's steps.

Finally, I would like to thank all my close friends who have never disappointed me during the whole period of the dissertation. Special thanks to my friends who used their own time to improve my writing skills and styles.

Abstract

With the development in the contemporary industry, the concepts of ICT and IoT are gaining more importance, as they are the foundation for the systems of the future. Most of the current solutions converge into transforming the traditional industry in new smart interconnected factories, aware of its context, adaptable to different environments and capable of fully using its resources. However, the full potential for ICT manufacturing has not been achieved, since there is not a universal or standard architecture or model that can be applied to all the existing systems, to tackle the heterogeneity of the existing devices. In a common factory, exists a large amount of information that needs to be processed into the system in order to define event rules accordingly to the related contextual knowledge, to later execute the needed actions. However, this information is sometimes heterogeneous, meaning that it cannot be accessed or understood by the components of the system. This dissertation analyses the existing theories and models that may lead to seamless and homogeneous data exchange and contextual interpretation. A framework based on these theories is proposed in this dissertation, that aims to explore the situational context formalization in order to adequately provide appropriate actions.

Keywords: Internet of Things, Information and Communication Technologies, Sensors, Context, Rules, Data Harmonization

Resumo

Com os desenvolvimentos da indústria contemporânea, os conceitos de ICT e IoT estão a ganhar mais importante devido a representarem os alicerces para os sistemas do futuro. Grande número das soluções existentes converge para a transformação da indústria tradicional em novas fábricas inteligentes interconectadas, cientes do seu contexto, adaptáveis a diferentes ambientes e capazes de usar os seus recursos na totalidade. No entanto, o potencial máximo da manufatura ICT ainda não foi atingido, dado que não existe uma arquitetura ou modelo universal ou padrão, que possa ser aplicado a todos os sistemas existentes e que possa lidar com a heterogeneidade dos dispositivos existentes. Numa típica fábrica, existe uma quantidade grande de informação que precisa de ser processada pelo sistema, para que se possa definir regras de eventos de acordo com o conhecimento contextual relativo, para poder posteriormente executar as ações necessárias. No entanto esta informação é, por vezes, heterogénea, o que quer dizer que não pode ser acedida ou percebida pelos componentes do sistema. Esta dissertação analisa as teorias e modelos de informação de harmonização existentes, que podem permitir a troca e interpretação de informação homogénea. Uma framework, baseada nestas teorias, é proposta nesta dissertação, que pretende explorar a formalização do contexto situacional para conseguir providenciar ações apropriadas, de forma adequada.

Palavras-Chave: Internet das coisas, Sensores, Tecnologias de Comunicação e Informação, Contexto, Regras, Harmonização de Informação

Table of Contents

List of Figures

List of Tables

Table of Acronym

Acronyms	Definition
API	Application Programming Interface
CEP	Complex Event Processing
CORBA	Common Object Request Broker Architecture
CPPS	Cyber-Physical Production System
CPS	Cyber-Physical System
DB	Database
EDA	Event-Driven Architecture
EU	European Union
FI-PPP	Future Internet Private Public Parcery
FoF	Factories of the Future
FP7	Seventh Framework Programme
GE	Generic Enabler
GRIS	Group for Research in Interoperability of Systems
HTTP	Hypertext transfer protocol
I2ND	Interface to Networks and Devices
ICT	Information and Communication Tecnologies
ID	Identification
IEEE	Institue of Electrical and

	Electronics Engineers
IO	In Out
IoT	Internet Of Things
ISO	International Organization for Standardization
IT	Information Tecnology
JSON	JavaScript Object Notation
OS	Operating System
OWL	Web Ontology Language
PIV	Process Identification Value
RDF	Resource description framework
RFID	Radio-Frequenct Identification
SDN	System defined Network
SODA	State Of The Art
TSV	Tab-separated values
UNINOVA	Institute for the development of new technology
VF	Virtual factory
Vf-APPS	Virtual-Factory Applications
Vf-MW	Virtual-factory Middleware
Vf-SK	Virtual-factory System Kernel
W3C	World Wide Web Consortium

1 Introduction

Nowadays world is living the fourth industrial revolution. The previous revolutions introduced the concepts of machinery work, mass production and automation, this one focus on smart machines and interconnectivity. The future industry will tend towards flatter management structures with a more highly skilled and IT literate workforce that will be focused on improving the product's development and performance. Factories are already applying the concepts of internet of things. This way a "network" can be formed between devices, objects and people that cooperate in order to accomplish common goals.

Figure 1-1: Interconnectivity in factories of the future.

It is important to develop a system that allows to join the concepts of internet of future with the existing technology in order to achieve a better performance. Within this system there would exist the necessity of modules responsible for managing the situational awareness. A factory is formed by several components and each of them produces data, regarding the situation that information can be important to generate an action or not. To know when data is important there must be defined a group of rules within a context. This is necessary because sometimes very similar inputs need to generate very different outputs. Imagine for example the statement "today is hot", if a human being hears it during summer will associate it with light t-shirts or even a beach day. However, if someone hears it during winter will relate it to just a warmer day. Because nowadays technology allows to integrate different kinds of technology, it is important to consider that a system must take care of information codified in different ways. This means that a system must have a module within it that is responsible for harmonizing the incoming data and make it understandable for all off its components. Although people do not associate agriculture with industry, the truth is that much of the work done since growing seeds to the harvest can be optimized used concepts that are already used in industry.

With the development of technology, new forms of agriculture are emerging in our society. The use of technology is growing in order to improve agriculture production and have higher profits. To achieve these goals, agricultural producers or distributors use Information Technology and Communication (ICT) to help to manage agriculture. Sensors can be used to enable real time monitoring food's parameters, such as pH, temperature, earth's moisture or oxygen flow. With these sensors and its connection with the Internet, it is possible to monitor all cultivations, even if they are apart, and predict and control its quality and how much food can be sold. With the knowledge of position, its values and seasonal care, tractors can supply plant's daily needs, without human intervention.

Several plants production is increasingly threatened. Climate changes, insufficient available lands, air toxins are some problems that agriculture faces these days. One agricultural solution to address these problems are plant factories. They can be defined as horticulture greenhouses or automated system facilities that have artificially controlled environments in order to produce vegetables and seedling year-round (Hirama, 2015). By using the benefits of smart farming, it is possible to turn the entire growth process automated and leading economies in order to control cost, quantity and quality against the required harvest time.

This work aims to develop an implementation of the models related to system´s situational awareness and data optimization that can be used within a framework that processes fruit information collected during a harvest process. In this dissertation is proposed the creation of a context framework, that can be used to infer about sensor data, based on context and rules. This dissertation's implementation will be done according to the needs of a European project called Vf-OS. This project aims to create an open operating system that can be used as a reference, considering Factories of the Future.To achieve this goal will be presented first the research question/hypothesis and the information that allows to contextualize them and present the information that allows to create a model in order to answer them. On a second phase will be presented a model based on the retrieved information that will allow to collect data to verify the hypothesis.

1.1 Motivation

Within factories there are lots of systems and information that must be exchanged or shared among the different blocks. Therefore exists the need of having a platform to manage all the interactions between the systems. However some platforms with this objective already exists in some industrial sectors, there aren´t platforms based on open and transparent standards.

Current European solutions are not affordable (considering cost and spent time) and aren´t oriented to create the needed standards. The present solutions are not global and even within the same factory can exist a large amount of technologies whose data is not

harmonized and consequently are not interoperable. The existing technologies have been created in order to develop point to point solutions that aim to solve or improve the solution to a single problem instead of considering the whole situation or global factory architecture. The main problem of this situation is that not much of the existing solutions can be adapted to different organizations, because they were designed to fit a particular system. This problems within industrial technologies are even more severe in modern agriculture, since this is a fairly new industrial "branch". Concerning subjects like situational awareness exists a huge gap between the existing theories and implementation.

Therefore exists the necessity to create a flexible solution to support the current industrial agriculture systems. The desired solution shall grant the interoperability of the current technologies, without large implementation costs and maintaining each companies' competitiveness.

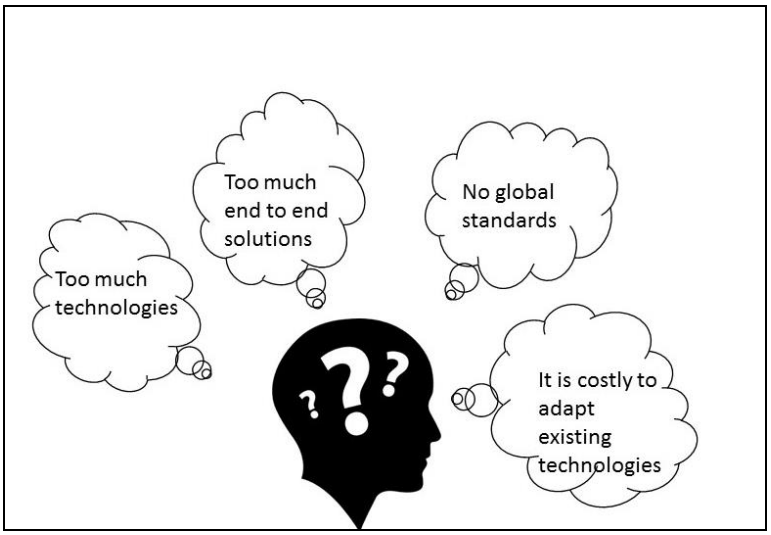


Figure 1-2:Motivation topics

1.2 Research question

The main theme for this dissertation can be generally described as system awareness in IoT systems, which leads to the need to know how a system processes the necessary information and what are the required parameters to do it.

To properly address the procedures of the necessary research work and to emphasize the objective of this dissertation, it is important to formulate the research question will be addressed during the completion of this work.

RQ: "Which IoT data processing methods shall be developed to use a context based framework capable of reasoning data?"

This question will provide some guidance during the development of this theses and its answer will be the result from the system implementation, validation and assessment.

1.3 Hypothesis

Based on the researched information and the research question, that was presented before in this chapter, this thesis is conducted regarding the following hypothesis:

H: "If a system can be compliant with a mental modelling approach, then its processes of contexts formalisation and consequent decisions/actions are enhanced."

This statement will be researched, implemented, tested and validated during the completion of this thesis. The results of the process will be presented and discussed in the following sections.

1.4 Work methodology

With the objective of presenting the best possible outcome from the development of this thesis it is important to define and clarify the steps necessary to proceed on a rich scientific investigation and experimentation. The steps presented were based on the required process stages, thesis supervisors advises and methodology bibliography (Harvard College, 2011) and (Camarinha-matos, 2012).Below are presented the principal adopted steps:

1. Research Questions – It is the first step to be made and, it is related to a specific problem or subject in which the author has interest in. The chosen questions may have as objective to solve a particular matter or to improve existing solutions.

2. Hypothesis – This step deals with the establishment of suggestions that can improve/solve the thematic within research questions.

3. Context Observations – During this step, research that can help to understand or contextualize the research questions is made. It deals with information published by other authors (including former students' articles) and observations that will help to understand the current SoA (State of Art) within the thematic.

4. Design Experiment – During this step an experiment that can be used to test the hypothesis is built. It concerns aspects such as variables control, observation and data collecting.

5. Test Hypothesis (Data Collection) – In this step data is collected using the model defined in the previous step and store it for further analysis and comparison.

6. Analyse Results – In this step the data collected previously is analysed, in order to provide a validation to the hypothesis. If the data analysis proves that the previously stated hypothesis is wrong or lacks information to prove it, the hypotheses must be rethink, according to the research questions.

1.5 Dissertation Outline

After presenting the subject and thematic that lead to the creation of the research question and hypothesis, it is important to explain how this dissertation is going to evolve, and how it will be organized.

Chapter two is the State of the Art, it will provide an overview of current technologies and solutions that have already been researched, concerning the thematic presented in the research question and this dissertation's abstract. In this chapter relevant concepts/theories will also be presented and compared that allow to approach the main thematic of this work.

Chapter three is called Used technologies and System Architecture. In this chapter, will be explained in detail the aims of Vf-OS project and also the applicational scenario in which further work will be developed. The architecture of a system that allows to solve/ answer the research questions/problematics will also be explained in this chapter. The explanation will be composed of a high and low level architecture clarification and guidelines that allow to integrate that architecture with other systems. Throughout this chapter will be taken in account the theories that were discussed in the SoA section.

Vf-APPS Development is the name of chapter four, in this section the development of the framework's prototype is presented, taking in consideration the previously explained architecture.

Results and validation corresponds to chapter five and allows to infer if the hypothesis that was developed for the research question is valid. In this section the prototype running and performed evaluations to the collected results are presented, these tests allow to evaluate the hypothesis. The content of this chapter corresponds to the steps five and six of the methodology.

The final chapter is called Conclusions and Future work, it provides an analogy/comparison between what was studied/researched, what was developed and the test results. This chapter provides an enclosure regarding this dissertations main thematic and highlights some points that may be addressed on future work.

2 State of art

In this section the main articles and literature on which this dissertation will be based will be presented.

The main goal of this dissertation is to develop a framework capable of processing data, based on context, to be used within vf-OS project. Because this project aims to develop an open operating system, to be used regarding industry 4.0, will be firstly explained the main concept regarding factories of the future. The concept of open operating system demands an open software core. For this reason, the second sub-chapter will detail information about the kernel that is one of the lowest software-level components within an open OS . To develop an open operating system, it is necessary to have access to open development tools. The third sub chapter deals with fi-ware enablers that are the main tools that will be used to develop the framework. The last sub-section focuses on the particular subjects and concepts that will be needed to develop the framework.

The main goal of this investigation is to understand the basic and main concerns and concepts important or related to the main project/objective. The research and articles consulted may result on new information which can lead to a change/modification of this chapter

2.1 Factories of the Future

Throughout history industry has been the target of several revolutions that allowed it to improve and adapt to the different times and demands from the market. As pointed by ("The Fourth Industrial Revolution Gains Momentum," 2016), until now existed three industrial revolutions. The first used the steam power to provide the energy to mechanize production. The second was responsible for the start of mass production and was powered by electricity. The third became possible due to the creation of microprocessors and robots that allowed automated production

The fourth revolution is expanding nowadays and merges the technologies that allowed mass and automated production with the internet. As stated in (Schwab, 2015) the fourth industrial revolution is not just about smart and connected machines and systems, its much wider.

According to (Keith Ridgway, Chris. W. Clegg, 2013) Factories will tend towards flatter management structures with a more highly skilled and IT literate workforce focusing more on product design, optimisation, monitoring and controlling of processes. These factories will rely on sensors and devices, that can create an accurate image of the current state within the

factory (like the current position of a product, for example). The information coming from the sensors will be correlated with digital models and simulations resulting on a high efficient task management.

Therefore, the concepts of internet of things and cloud computing can be used, in order to create cyber physical systems that will populate the factories of the future, which will be explained in this chapter. The internet of things is a communication paradigm in which objects can be equipped with transceivers (and respective control logic), for digital communication with each other's and with the users (Atzori, Iera, & Morabito, 2010). This way a "network" is formed between devices, objects and people, that cooperate through addressing schemas, in order to accomplish common goals. There are three main components within the IoT which are hardware, middleware and presentation(Gubbi, Buyya, Marusic, & Palaniswami, 2013). The hardware is composed of the various sensors (RFID, pressure, presence…), actuators (relays for example) and even the hardware used for communication (bluetooth, wireless modules). Middleware is software that acts as interface between the components, allowing communication between objects that wouldn't be able to do it otherwise. Presentation englobes tools that allow visualization and data interpretation and, that can be accessed from different platforms. Thinking about these components, it is easy to create an analogy between them and what is usually found at an industrial environment and that is why it is easy to associate IoT to the factories of the future.

2.1.1 Concepts of factories of future

Based on the findings from the literature review, in (Hermann, M.; Pentek, 2015) are presented the principles that are used to obtain design principles to factories of the future and that are relevant to this work's main goal. These principles are interoperability, virtualization, decentralization and real-time capability and are explained below.

2.1.1.1Interoperability

Interoperability is the ability of two or more systems to communicate and exchange data, even if the languages and models used in their implementation are not the same(Fortineau, Paviot, & Lamouri, 2013). This means that the concept of interoperability is an important factor on factories of the future, because it allows to enable communications between the CPS, the installation network and the people connected through it. An important tool to achieve interoperability is a web-service, as data stored within this service can be consulted by the devices in the network, despite their core languages, as seen in (Karnouskos, Colombo, Lastra, & Popescu, 2010). Older interoperability solutions depended on frameworks such as CORBA, as seen in (Courses, 2014). CORBA was a standard for application development when dealing with heterogeneous environments and platforms, due to its services abstractly defined. When running this framework redirects client's requests to a remote host, without the client needing to know where and how this object is implemented.

Nowadays, new strategies can be applied to achieve interoperability between different systems. As seen in literature reviews, such as (Gerhard Friedrich &, 2005) , (Doctor, Hagras, & Callaghan, 2005) and (Nativi, Mazzetti, & Geller, 2013) some common solutions are:

- Mediated approach: The integration procedures are done by mediators. When running, the systems converts the information source queries (in the source device language) into destiny source queries.

- Agent-based-based approach: When running, an agent-based approach uses mediator interface and source agents. The mediator agents communicate with source and interface agents. Source agents communicate to data sources. Interface agents receive user's queries and send them to the mediator who asks for the response from the source agents.

2.1.1.2Virtualization and Cyber physical systems

The virtualization of the factories of the future can be achieved using cyber-physical systems, that are composed by a physical entity embedded with a cyber entity(Wen & Guo, 2016). The cyber part contains a virtualization of the physical part so that it can replicate virtually any behaviour of the physical machine. Being a virtual/physical system allows the system to easily access and store information in the cloud. CPS also contain wireless embed wireless devices (Bordel Sánchez, Alcarria, Sánchez de Rivera, & Sánchez-Picot, 2016), which allow the various machines to interact and communicate with each other and with the network. The concept of cyber-physical systems is illustrated in figure 2-1.

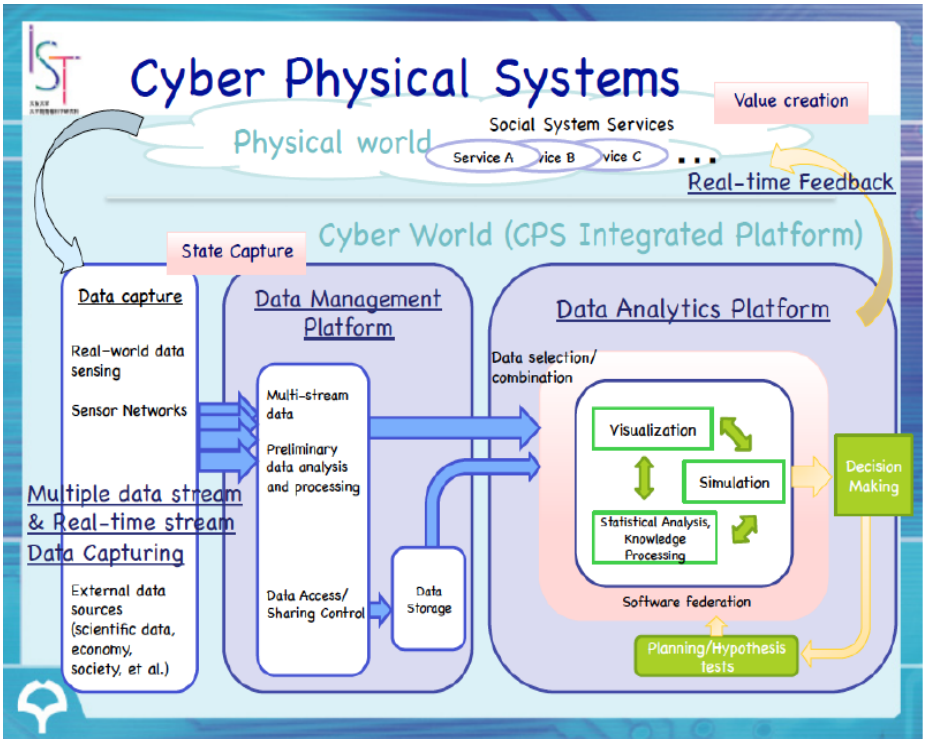


Figure 2-1: Cyber-Physical Systems Concept Illustration (T. Higashino, 2005)

2.1.1.3Decentralization and real-time capability

Nowadays, most factories use centralized control systems to meet their demand and goals. The main problem is that, with current market demand, centralized systems are pushed to their limits, because of their abilities to deal with complex plans, and variable control and logistics systems (Schuhmacher & Hummel, 2016). Decentralized control systems can overcome these problems by distributing operations between the various nodes of the system. In the previous section the characteristics of the CPFs and, their capacity of exchange between them were presented. These characteristics allow for the creation of collaborative platforms where various CPPS groups can obtain holistic information about their tasks and thus learn and be trained on different tasks (Kreimeier, Morlock, Prinz, Krückhans and Bakir, 2014).

The ability to obtain holistic information on certain tasks allows CPPS to make decisions through production control without superordinate control systems (Gräßler, Pöhler and Pottebaum, 2016). Therefore, the various production systems can determine production sequences by themselves, which, combined with real-time data analysis, leads to efficient real-time performance.

2.2 Kernel

According to (Mauerer, 2008) the kernel can be considered an enhanced machine that, in the view of the application, abstracts the computer on a high level. Therefore it can be viewed as a resource manager as it takes care of the various programs that are running concurrently in the system.

According to (Justin Garrison, 2010) it is the lowest level of software in the computer and responsible for interfacing all the applications running, down to hardware. Kernel is also responsible for managing the memory where these applications are loaded. Without the existence of a kernel, or another layer that could perform its roles, the user would need to restart his system every time he wanted to run a different program (as used to happen in first computer systems(brokenthorn.com, 2009)).

To achieve its goals Kernel needs to create and manage processes, allocate memory and manage the physical hardware(The Linux Information Project, 2004).

2.2.1 Processes

One of kernel's major responsibilities is the process management. To understand what this management concerns, it is important to clarify what a process is considered to be. The concept of a process includes, not only the executing program code but also its related

resources (Tobergte & Curtis, 2013). These resources may include memory position, internal kernel data, open files and threads that represent the objects of activity within the process.

In order to store all this information, exists a large structure within the kernel called *task_struct* (M. Tim Jones, 2008), that also contains memory pointers to relevant files and program dependencies. Whenever a program is created memory is allocated to create an instance of the *task_struct.* Within this structure can also been found the current state of the program, the table 2-1 describes the possible states of a process, as seen in (Robert Love, 2005).

Table 2-1 – Kernel process states.

Task	Process Description
TASK_RUNNING	The process can run. This state can either imply that the program is either running or on a run queue.
TASK_INTERRUPTIBLE	The process can't run. This state is applied when a process is waiting for a certain synchronous signal to unblock (change status to TASK_RUNNING)
TASK_UNINTERRUPTIBLE	The process can't run. This state is similar to TASK_INTERRUPTIBLE but can't be interrupted by a signal (however asynchronous signal can be received). It is used in delicate processes like the ones that involve spinning the hard drive disk or spinning heads.
TASK_ZOMBIE	The task has terminated but the process that called the task can still access it until decides that isn't necessary. When that happens the process descriptor is deallocated.
TASK_STOPPED	The process is stopped and is not eligible to run. This status exists to prevent severe crash scenarios and it´s commonly used when a process receives a signal while debugging.

2.2.1.1Process hierarchy and device management

In kernel processes follow a hierarchy since they are born, the first process is *init*, and it is launched during the last part of the booting phase. Every process has a unique *PID* (process identification value) (The Linux Information Project, 2005) that is received when a process is born. This value is sequential, which means that the next process value will be one unit higher than the one that was launched before. Because *init* is the first one to be launched its *PID* is 1 and it is the parent of every processes that it calls. Due to this hierarchy and, to the pointers stored in the *task_struct,* it is possible to track the last called process into the first. This is an

important characteristic, because it provides a very efficient way to locate parents through their children and vice-versa.

2.2.2 Device and drivers management

Kernel is in charge of the device management, that can be achieved with the use of device drivers. As stated in (Rubini & Corbet, 2005) drivers can be considered software modules that make a particular piece of hardware respond to a well-defined internal programming interface. This means that a driver needs to accept requests from the software related to the device it controls, accept requests from the kernel and ensure that the kernel's requests are executed with success. The requests are made through interrupt service routines and parsed using I/O protocols.

To organize this information kernel uses a process called *udev*. This process manages all the drivers and is in charge of supplying a dynamic device directory (dev), with the nodes of the devices that are currently connected to the system (Unnikrishnan A, 2009). Whenever a device is added or removed to the system an *udev* event is triggered and parsed according to *its* rules files. If that event represents a connected device a directory is added into the dev directory (at each directory corresponds a different device), when the device is updated/changed, the information within this folder changes too. Finally if the device is removed, the folder is also removed.

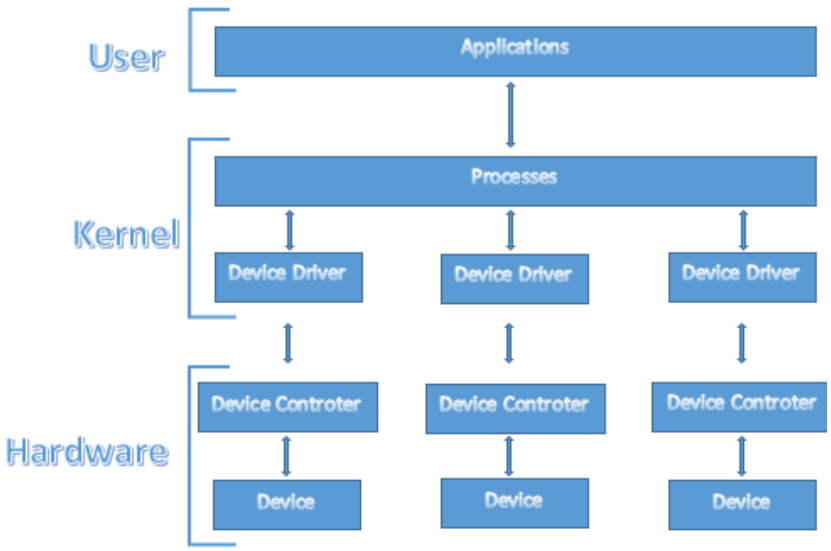


Figure 2-2: Kernel driver management

2.3 FIWARE, the European project

FIWARE is a project that looks for the creations of a core platform on the context of the Future Internet Private Public Partnerships (FI-PPP), that is part of FP7(Seventh Framework Programme) of the European Commission (Tuominen, 2013). The programme's final goal is to develop future internet technology which can be used in smart infrastructures and that will contribute for the technological competitiveness and growth of Europe.

To fulfil its objectives, FI-PPP was divided in three phases that took place between 2011 and 2016 ("Future Internet Public and Private Parceries," 2012). The first phase focused on the development of the project architecture, technology foundation and creation of specific enablers. Phase two concerned on the creation of infrastructure to operate an European network of FIWARE nodes. Phase three focused on probing the vitality of the project and expansion of the use cases. The development structure of FI-PPP can be seen in figure 2-3, it is also possible to verify the existence of important open source development tools, such as fiware (FI-CONTENT).

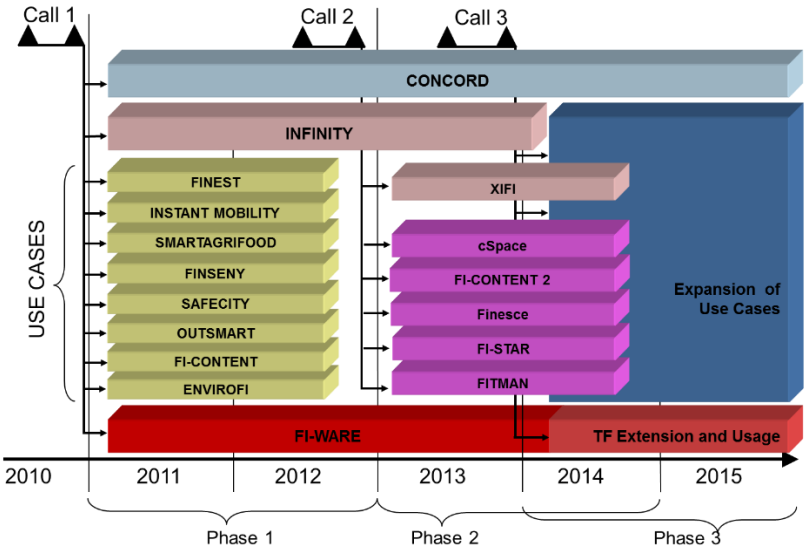


Figure 2-3: FI-PPP architecture("CORDIS Archive : European Commission : CORDIS : FP7 : ICT : Net Innovation : FI-PPP Call 3," 2013).

2.3.1 FIWARE enablers

As it has already been explained, FIWARE was born from the Future Internet Private Public Partnerships programme. With the adoption of FIWARE engineers and developers gained the ability to use its capabilities for the design of Cloud and IOT architectures. Using FIWARE became possible to speed up of the design and implementation of architectures and platforms, as seen in (Fazio, Celesti, Marquez, Glikson, & Villari, 2016).

FIWARE offers many open-source technological resources and is based on elements that are called Generic Enablers (GE). The Enablers offer reusable and shared modules, which include protocols and interfaces for operation and communication, that are able to cover several usage areas among different sectors(Pablo Fernández , José Miguel Santana , Sebastián Ortega & José Pablo Suárez , Conrado Domínguez, 2016).

Although the activators can be applied in several different cases, a single service can not be applied in several areas. This means that there are specific facilitators for a specific area of focus that allows you to divide the modules into technical chapters. According to (Trindade et al., 2016) there are seven different chapters:

1. Cloud hosting
   As one of FI-PPP's objectives was to improve the competitiveness of the European market, it has become important to provide a service to help start-up companies and even small businesses. Providing resources for this area means the improvement of a service that these types of companies were already paying according to their data storage needs. The services provided to the European cloud are business-to-business variables, when a business expands or increases the storage of data needs, it just needs to request more resources / data space. When choosing this type of service, a low initial investment is granted. To support the European cloud services, the specific FIWARE facilitators have been created, according to this chapter.

2. Data context/management
   The rising of industry 4.0 leads to the emergence of smart factories and other business branches that produce a big amount of data. The analysis of this large quantity of information, whether it is structured or unstructured, demands specialized programs or applications. This FIWARE chapter looks forwards to develop Generic Enablers that provide an easier and faster way to create or integrate data analysis algorithms.

3. Internet of things
   As several interoperable services appear it is important to integrate different systems. Considering that the architecture/programming language used among the various systems can be different, there must exist a system component/service that provides the knowledge translation among the systems. This chapter focus on obtaining information and making it available and accessible so that services can profit from the data collected.

4. Applications, Services and Data Delivery
   Under this chapter are developed enablers related to the creation of applications and a services ecosystem that promote sustainable growth and innovation.

5. Security

However the future industry promotes the interoperability between services and systems, it is inevitable that each device/system develops its own individuality. This way it is normal that each service has data transfer and security protocols that grant data integrity. This chapter deals with the enablers that create security features within protocols, allowing communication between different devices.

6. Interface with Networks and Devices(I2ND) Architecture

The I2ND is related to the creation of Generic Enablers that aim to execute a network infrastructure, whose model is standardized. The facilitators in this chapter need to be able to handle very sophisticated operators' terminals, proxies, and infrastructure. The successful management of these components ensures that the I2ND network can be accessed by other potential suppliers.

7. Advanced Web-based User Interface

It is important to provide the final user not only with service functionality but also with an environment that improves the user experience. This is achieved by adding user-friendly layouts, inputs and interaction capabilities, which include 3D graphics and systems virtualization. It is also granted compatibility with traditional web services and the opportunity to use more advanced features.

2.4 System Awareness and Data Harmonization

As far as data awareness is concerned, it is important that the system knows what is around it and what is the concept in which it is working. Data awareness refers to information that a given operator needs to know to make decisions and complete tasks. This concept can be easily understood, if you consider real-world situations. For example, a race car driver does not need to know how his car engine's combustion works, but he needs information about the best gear and maneuvers for each turn in a race circuit. It is also necessary to make sure that all information received can be processed by all elements of that system. As seen in (H. Chen, Finin, & Joshi, 2003) and (H. L. Chen, 2004), there may be many methods for acquiring context information. To choose the right method, it is important to have regard to the architecture design of the target system.

2.4.1 Context models

There are many ways to establish the rules that will define the system´s classification methods, the simplest way is to use key-value definitions. Key values are always composed at least by a pair of two words (Bettini et al., 2010) as one is the attribute key and the other the meaning key. Traditionally, when using key-values definitions, the information will appear in the format <key,value> (Zhao & He, 2009). For example, the word "animal" can have associated an action and the word "bear" can have associated a value. In this kind of services the pair values are stored in tables within the system's memory. A similar approach can be done using markup models, that instead of storing the correspondent pair of keys in tables, tag all the data and store them according to a defined hierarchy(Indulska, Robinson, Rakotonirainy, & Henricksen, 2003). In this case, the hierarchical data structures consist of mark-up tags, with attribute value (Patnaik, 2013).

Another possible solution can be achieved using logic based models as seen in (Strang & Linnhoff-Popien, 2004). These models have the context defined through several facts, statements and rules that, in the presence of the required conditions trigger the system´s response. For example, having the input "a" in the context "x" trigger the response "z".

A more versatile solution is the use of ontology based models that involve knowledge representation languages. Ontologies provide a description about the target world, retrieving its concept´s through classes, members through instances and roles and needed information through relationships (Ian Horrocks, 2006). According to (Wang, Gu, Zhang, & Pung, 2004) ontologies provide advantages when considering knowledge sharing, logic inference and knowledge reuse:

1. Knowledge sharing
   Using ontologies means that system entities such as agents and services have a common set of context, when interacting with each other. This also implies that it is easier for someone that has not high knowledge on a specific system, to make some modifications, since he has only to change the parameters within the ontologies.

2. Logic inference
   Based on the information under the ontology a context-aware system can deduce high-level context from low-level context.

3. Knowledge reuse
   Reusing well-defined ontologies from different do-mains it is possible to build a different ontology without the need to start from zero, as seen in(Benjamins, 2000).

2.4.2 Context frameworks

When creating a model that has context involved it is necessary to make sure that the system levels process that information according to their designed function. As seen in (Stewart & Narasimhan, 2007)it is necessary not only to define the context, but also use levels/layers that take care of gathering storing and sharing it. The information gathering is usually made by sensors who need to have rules about when is necessary to retrieve the necessary information. For example a sensor used in a system that has as objective to collect the solar light intensity, does not need to collect the light values at night even if there is some moonlight. It is also necessary to ensure a mechanism that tolerates delays in a context sharing situation. Storing context is important when referring to a system that needs some added value, besides the real time information, as defended by (Salber & Abowd, 1998). Sometimes the system may need to predict the future cases/reactions and, that decision can be based on older information. The context use defines the situations that are necessary for the system to enable a behaviour or action and to decide whose methods were already discussed in the previous section.

As seen in(Miguel & Miranda, 2014) the most common context architectures are hierarchical, with the possibility of having centralized elements. Having a hierarchical architecture allows to consume less processor and memory resources, however it also leads to a system failure if an element fails. The figure 2-4 illustrates a hierarchical presented in(Mamo & Ejigu, 2014):

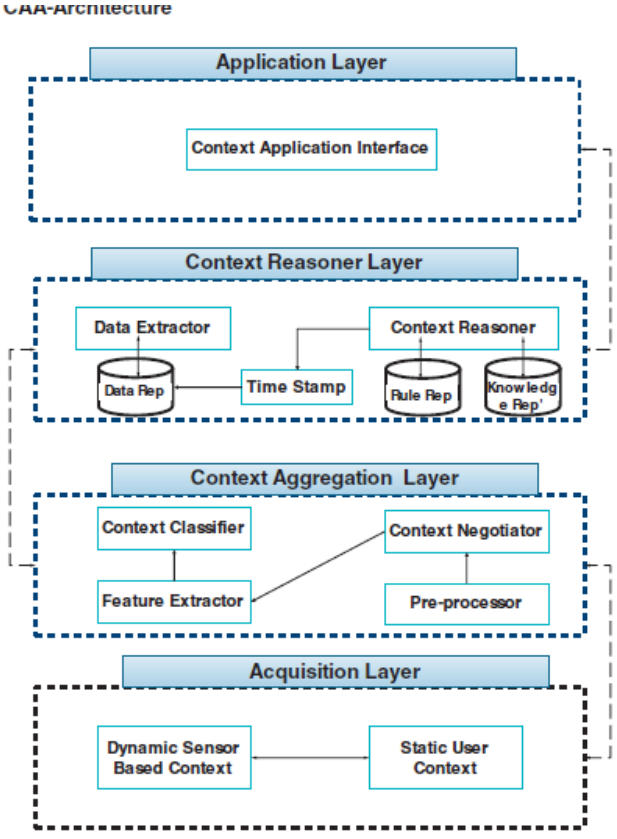


Figure 2-4:Hierarchical context framework(Mamo & Ejigu, 2014)

In this architecture Information is obtained through the acquisition layer. The static user context module collects user information and, the dynamic sensor based context collects the information from sensors. Context aggregation layer processes the raw information from the acquisition layer and organizes it so that the context reasoner layer can easily process the obtained information. The information passes through a pre-processor that binds data from the two modules on the previous layer. Context negotiator determines the validity of retained data and sends it to the feature extractor, that determines the most important information within a block of data. That information is sent to the context classifier block, that combines the information's attributes, creating new context classes or supporting existing ones. The information is then sent to the context reasoning layer where the information is processed based on defined rules and past experiences. Finally, on the application layer the system interacts with user based on the processed information.

2.4.3 Situation awareness

In the previous chapters the mechanisms related to the system awareness were discussed. The levels that a system needs to have, to grant action and decision making, were explained. However, to address the presented hypothesis, it is also necessary to understand what is the logic used to explain how an individual can achieve awareness, from the information at its range. Below will be explained several different mental models, which provide explanations about how individuals achieve awareness, by analysing the available information (Morrison, 2012). The models to be explained are the three level model, the perceptual cycle model and the activity theory model(Panteli & Kirschen, 2015).

In (Endsley, 2000) is described the three level model which uses three levels to achieve situational awareness,. These levels are perception, comprehension and projection and that can be applied to virtual systems. Perception is related to the basic sense of the important information. To perform fast, an individual must filter the important information from the whole data received. Comprehension is about how the system combines, interprets, stores and retains information. The speed of the comprehension level will depend on how well perceived was the information on the previous level. Projection worries about the future situations that can be anticipated from the currently comprehended data. In the model depicted in figure 2-5 ,it is also possible to understand that with the same world information but different experiences it is possible that two equal individuals get different decisions. This explains why sometimes when an individual detects some past flaw, can choose not to do the same mistake, when the same situation occurs and, someone that hasn't witnesses the same flaw can not.

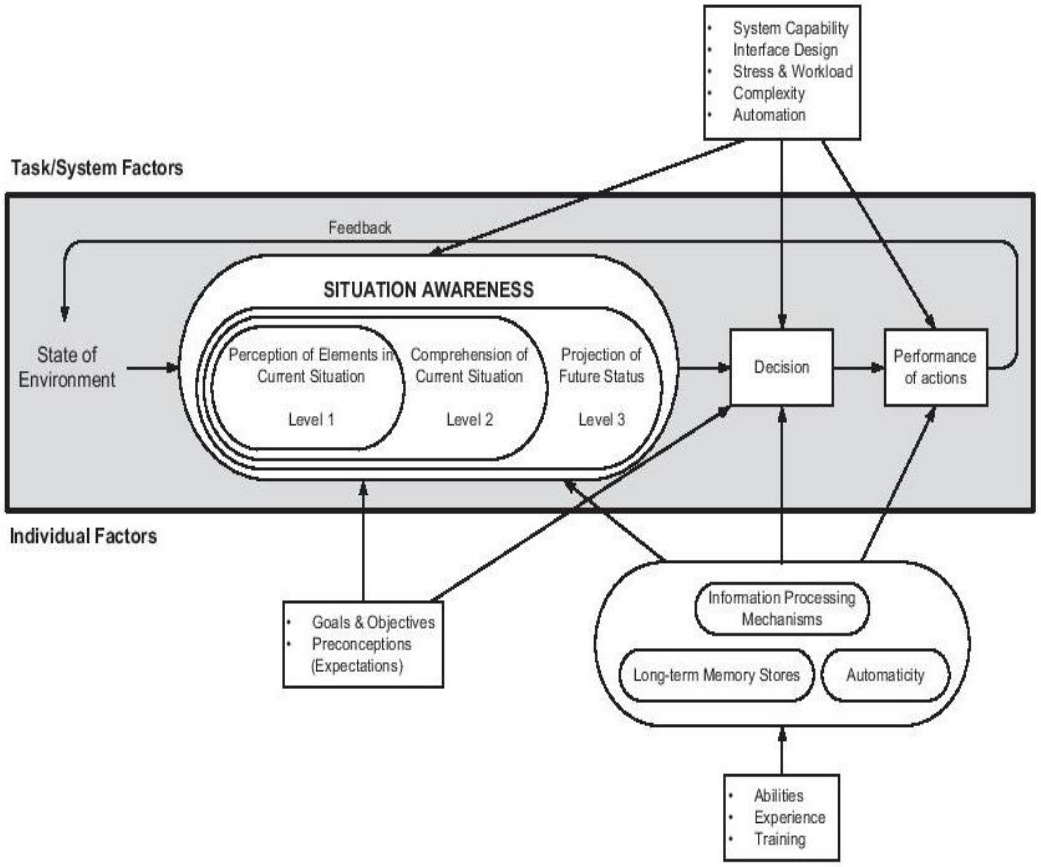


Figure 2-5: The three level module for situation awareness (Panteli & Kirschen, 2015)

Regarding the perceptual cycle model, (Plant, Stanton, & Harvey, 2013) and (Giacobe, 2013) explain that it is based in the interaction of the individual with the world. To use this model there are three important concepts, the environment information, the schema of present environment and perceptual exploration. The environment information is related to the potential information extracted from the system's world. The perceptual exploration concerns the information that can be achieved from exploration (like for example the perception that a surface is wrinkled from touching it) and provides it to perceptual information. The schema of present environment is the big picture model, that is created from the previous images of the world. In this model all the three levels coexist and modify each other, perceptual exploration provides environment information with real time data that it uses to create a world image. By comparing the current world image with the schema of present environment it is possible to make decisions. Whenever the current schema is outdated or does not allow to make the best decision it is rewritten according to the current world picture. The information contained in the schema provides standard decisions or actions that allow perceptual exploration to retrieve the most accurate data.

The activity theory model is based on the idea that the extent to which processes are involved is dependent on the nature of the task and the goals of the individual (Stanton, Chambers, & Piggott, 2001). This means that when using this models exist several sub-modules that take care of small parts of the decision process. The figure 2-6 illustrates this kind of system.

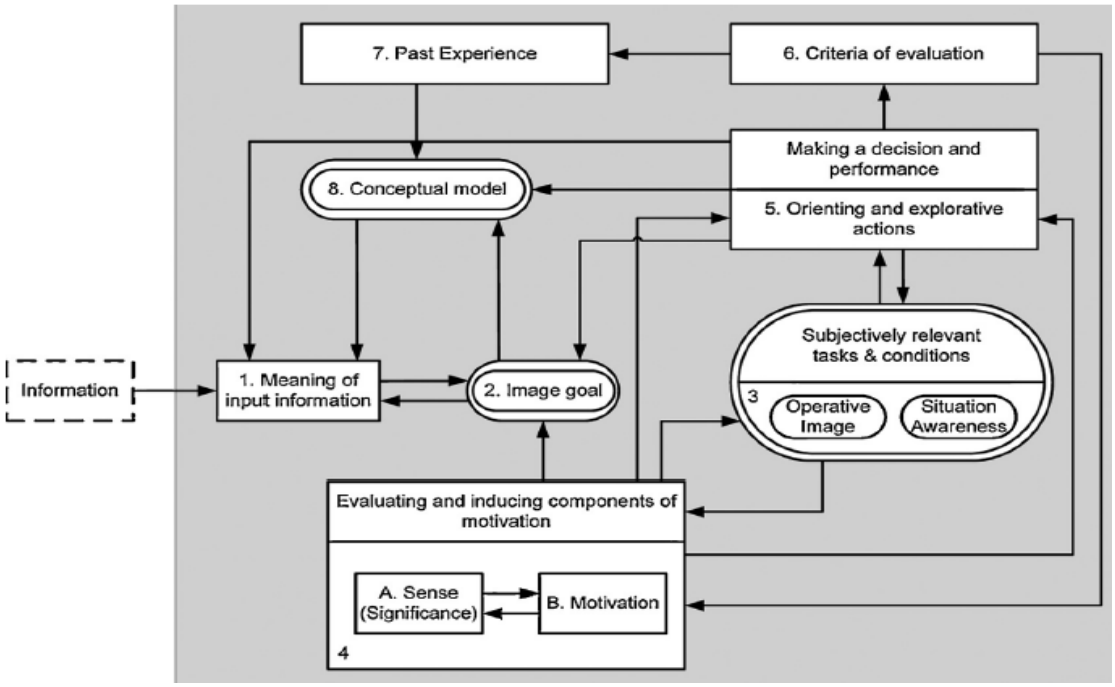


Figure 2-6: Activity theory model diagram (Panteli & Kirschen, 2015)

The new information arrives on block 1 and is interpreted according to the model of the world (block 8),the purpose of the information (block 2) and orientation according to the to the objective of the information (block5). The interpretation of this information goes to block 2. According to this interpretation the individual can decide what are the relevant tasks to process the information (block 4) and their relevance to the world (block 5) based on a criteria evaluation (block 6). The result of the evaluation will direct person's engagement with the world(block5) and new criteria will be developed (block 6). The interaction with the world will be stored as experience (block 7) and inform representation of the world (block 8). This process repeats for new information, at each interaction the conceptual model will be based on each time more knowledge which leads to a more efficient decision making.

2.4.4 Data harmonization levels

As it was already mentioned, sometimes within a system formed by a group of sub-systems exists heterogeneous information. This may happen because not all the systems have the same architecture or were not built in the same language and do not have the ability to exchange data. It is possible to know about data specifications and properties using its metadata. Metadata is simply information about the data itself and has several types, as shown in (Rebecca Guenther, 2003). Descriptive metadata contains information about the type and author, that allow a system to realise if a set of data is compatible with it and therefore verify if there is immediate interoperability.

There are different interoperability levels in which data can be incompatible, that are process, legal, technical, syntax, semantic and query levels (Veeckman et al., 2016). Process level incompatibility occurs when trying to merge two datasets created from two different

processes. Usually this case is more a policy problem than a data problem, since the data can still be read and used. Legal level incompatibilities happen when merging two datasets while one of them has restricted access from its creator. The syntax level is related to whether the structure of information data can be successfully merged. When two different data sources use different data models to store information that looks alike, the merge of the two datasets cannot be done without a prior normalization. Technical level deals with problems related to the means of transport of information. For example, merging information received from bluethoot with text typed, can imply some difficulties because they are different data types. Semantic level takes care of issues related to the ambiguity of identifiers. When for example considering two different datasets that have a client's name list, one can have a field with the complete name and the other two fields, one with the first name and the other with the last. Finally, query level is of major importance when merging information that is split into multiple data files. For example, when merging client's information to create a complete file, information like email can be in one file and cell phone number in another, in this case it is necessary to perform several small queries to gather the information

2.4.5 Data harmonization framework

The data coming from different sources must be harmonized to achieve a common data standard which can be processed by the system. Although exist several methods, usually the process is common to several data types, as it is illustrate in figure 2-7. The process starts with establishing an inventory of the current data requirements, definition of the data collected(1), analysis of the information requirements and data elements(2,3,4) and reconciliation of the data (5) (United Nacions Economic Comission for Europe, 2017). When processing data, each dataset is considered to be formed by several series of statements that are serialized according to a particular syntax(Colpaert et al., 2014). Some syntaxes, like TSV, represent information using tables, where each cell in each row contains the statements. Syntaxes like XML and JSON represent information hierarchically (within a tree structure). Because the syntaxes are different, some of the normalization steps will have to be specialized to a certain syntax.

In (Wc & Corve, 2015) are introduced the basic steps to achieve harmonization. By modelling the system for each of these steps it is possible to convert heterogeneous into homogeneous information.

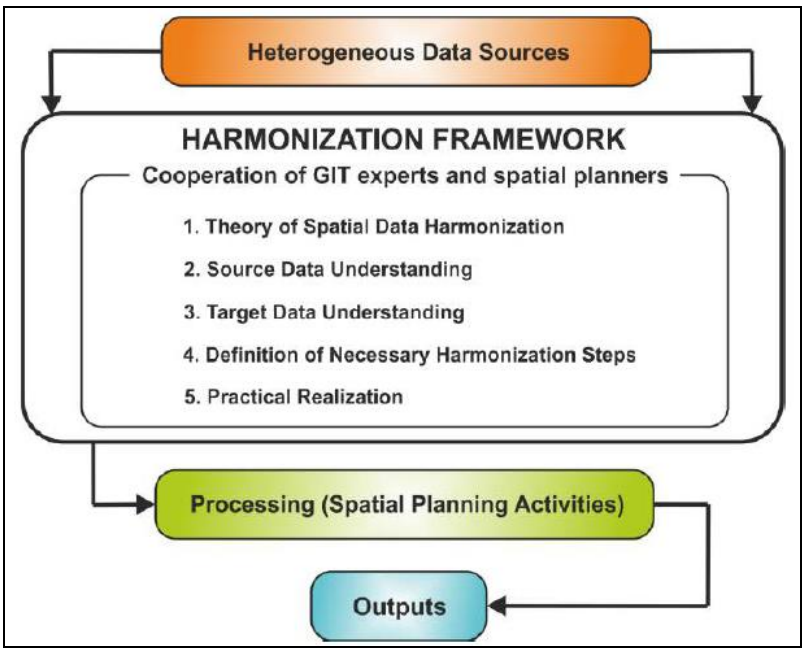


Figure 2-7: Data harmonization process(Wc & Corve, 2015)

The three first steps within the framework are common to most normalization processes, the understanding of source data is done using metadata in-formation, as referred before. The fourth step will depend on how the syntax stores the information and according to that will be created personalized harmonization steps.

2.5 State of the art conclusions

After explaining the theories that allow answering the question of research, it is important to review some of the presented concepts and compare approaches, considering the opinion of the Author on the work to be developed. The first section was related to the factories of the future and explained how the IoT concepts influence industry 4.0. During the explanation of the characteristics of FoF the concept of interoperability was introduced and, the several approaches that can be carried out to exchange data from different systems, were explained from a high level point of view. From the explained approaches (CORBA, mediated approach, agent approach and webservers), the one that best fits this work is the web server solution. Comparing the four approaches, the web server is the easiest way to exchange information, considering that there are several enablers that work in this domain, as seen in the Fiware section (Internet and Cloud Application Chapters). It can also be considered an enabler of the internet chapter based on cloud storage,to transfer information stored over the internet.

The second section explained the importance of a Kernel within an operating system, which was necessary considering the goal of creating an open system. The drivers' management chapter explained how a kernel interacts with peripheric devices. The same

logic can be used by the vf-APPS, when interacting with sensor devices. In this case, the system's applications would request the information, making it understandable to the device, through a software module. The recovered sensor information could be stored using webservers and accessed through Fiware enablers.

In the System Awareness and Data Harmonization section the ways how a system can be aware of the elements that surround him and, how to use that information to react, were explained. The models that allow to describe a system and create context were also explained. Considering a simple framework, key value pars can be used, considering that there is not necessary a hierarchy between the several rules to be applied. The same can not be done with complex frameworks that demand a high level of context description and definition. From the provided models, the Autor considers Ontologies to be the most complete, however, may exist enablers that combine the models' strengths and that allow to reason context.

Several mental model theories were also explained, in order to find one that could be used to create a context framework compliant with the way how individuals achieve awareness. The activity theory model defends that the information gathering and analysis depend on the nature of the task and, on the goals of the individual. This logic is not fit to be used in a system that needs to be able to impartially analyse sensor data. The perceptual cycle model is based in the interaction of the individual with the world. Considering this model's application cases, it would be better used in a machine learning system, because it would provide such a system with tools to grow smarter. Finally, the three level model, uses several levels to achieve situational awareness (perception, comprehension, projection). This model can infer about the analysed data, because it effectively defines the functions of each level in detail. Considering the models that were presented, this is the one that describes with higher detail the necessary steps to achieve awareness. By relying on this model, it is possible to create an analogy for a framework that uses it to obtain context evaluations, based on collected sensor data.

Considering data harmonization, the concepts, that need to be considered when merging information from heterogeneous data sources were presented. The several incompatibility levels were references and explained, however, it is necessary to dig deeper to solve specific problems. For this reason, it is necessary to identify the specific framework's harmonization issues, that may occur, based on the chosen architecture.

3 System Architecture

This section aims to explain the technologies in which the framework is based and also present its architecture. The first chapter introduces an agriculture practical scenario, as the problem to be addressed. To Solve this problem, it is proposed to use a context based framework that is able to interpreter sensor data from crop sensors, based on provided rules. The framework's architecture takes in consideration concepts of the three level mental model explained in the previous chapter. The framework section shows in detail each part of the framework's structure, starting the presentation from a very high level and then explaining each module in detail. Besides an architecture it is necessary to identify a set of specific tools that allow to build it. Orion and Docker sections present in detail these technologies and explains how they can be used to create the framework. When dealing with sensor data, can occur harmonization problems, if the sensor's architecture it is incompatible with the framework's sensor model, this thematic is addressed in the harmonization chapter. Sensor data mismatches section, explains the integration problems that a framework may find, when dealing with different sensor models. Harmonization guidelines section present solutions to integrate different sensor types with the framework, taking in consideration how the information is retrieved. The last part of this chapter has as objective to relate the developed framework's needs with vf-OS, resulting in the final system architecture.

This section aims to explain the technologies on which the structure is based and also present its architecture. The first chapter introduces a practical agricultural scenario, as the problem to be addressed. To solve this problem, it is proposed to use a context-based framework that is capable of interpreting sensor data from harvest sensors, based on rules provided. The framework's architecture takes into account concepts from the three-level mental model explained in the previous chapter. The structure section shows in detail each part of the framework's architecture, starting the presentation from a very high level and then explaining each module in detail. In addition to an architecture, it is necessary to identify a set of specific tools that allow the construction. The Orion and Docker sections explain these technologies in detail and clarify how they can be used to create the framework. When dealing with sensor data, harmonization problems may occur, if the sensor architecture is incompatible with the framework's sensor model, this issue is addressed in the harmonization section. Sensor data mismatch section, explains the integration problems that a structure can encounter, when dealing with different sensor models. The harmonization guidelines section presents solutions for integrating different types of sensors with the structure, taking into consideration how information is retrieved. The last part of this chapter aims to relate the needs of the developed structure with vf-OS, resulting in the final architecture of the system.

3.1 Scenario

Modern agriculture uses sensors to grant high product quality that rewards farmers with higher profits. The usage of sensors can also be applied in several parts of the product chain increasing the product quality. This sensor technology that makes farms more intelligent, with real-time data gathering, processing and analysis is called 'smart farming' (Kamilaris et al., 2016). The chosen scenario deals with a modern fruit product chain that deals with the product from harvest to client delivery.

After fruit is harvested by the farmer, it is separated (manually or mechanically) and submitted to quality control to confirm that the desired standards have been achieved. The several strains of fruits are split according to the desired characteristics and made available to the client. From the moment that the product has been made available clients can decide which products to choose according to his own needs. To perform this, exists a buyer's application that allows not only to view the farmers registered in the platform but also to choose the products, based on features such as fruit strains and measurements. When client purchases a product, farmer receives a request and chooses weather to accept it or not, if he accepts it the product is shipped. The transportation is also monitored with sensors, informing the client of the conditions felt by the fruit. The figure 3-1 and table 3-1 describe this process:

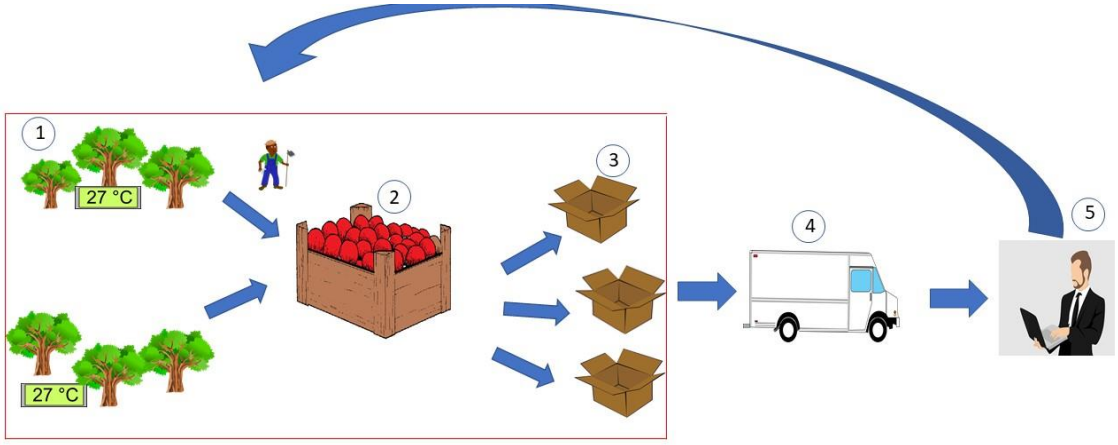


Figure 3-1: Global Scenario

Table 3-1 – Scenario Steps

Steps	Explanation
1	Fruit harvest, controlled by IoT Devices
2	Fruit selection, separation (manually or automatically)
3	Fruit boxing
4	Transportation control
5	Buying order

During the whole process can be used several technologies according to the necessary processes, such as harvest, transport, etc. Regardless the process used, due to the presence of sensors and software that use the recovered data, it is possible to be aware of the product's situation in real time. That information is recorded and delivered to the final client at the end of the product chain. The large amount of control during the process can be used not only to control the product´s quality but also his valor.

To fully achieve this scenario there must exist several functionalities to satisfy the different needs (such as quality monitoring, packing, monitoring during transportation, product request among others). In order to achieve a solution that can grant high product quality, based on information gathered by the sensors within the crop (red selection within figure 3-1) can be used a context based framework.

3.2 Framework

This work aims to provide a solution for the previous explained scenario, that involves the verification of contexts and rules, applied to certain sensor information. It also aims to provide a solution that can be easily modified to fit other applicational scenarios. The chosen solution should use the concepts from the three-level model, explained in 2.4.3. To create the architecture was first necessary to choose its processing model. From the existing models, the one that provides more benefits regarding a frameworks implementation is a layered architecture. This architecture type has as objective to reduce the design complexity, by organizing a series of levels, each one built upon the one below it. The main objective is to have the design divided in small pieces, in a way that the highest level can have access to the functionalities it needs, without the associated complexity. The benefits of the layered models are modularity and clear interfaces, i.e. open architecture and comparability between the different providers' components (Courses, 2014), which is compatible with a context framework. For this reason and based on the work of (Mostéfaoui & Hirsbrunner, 2003),(Bouvin et al., 2017) and (In & For, 2006) was decided to propose a layered architecture. The proposed architecture has four layers, which are Acquisition Layer, Sensor Data Harmonization Layer, Context Aggregation/Reasoning Layer and Application Layer. Considering the three level model, perception corresponds to Acquisition and Sensor Data Harmonization Layers. Comprehension and projection levels correspond to the Context Aggregation/Reasoning Layer. Application Layer does not correspond to any of those levels, however it allows the user to be informed about the projection level results.

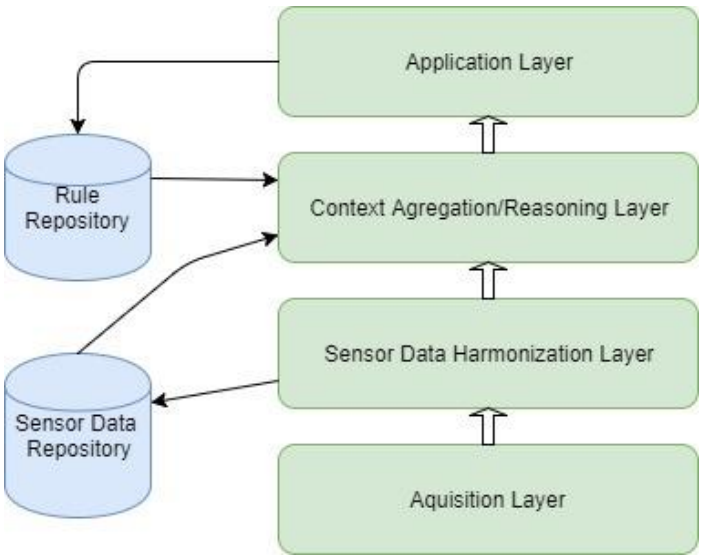


Figure 3-2: Framework High level architecture

The Acquisition Layer is responsible for obtaining sensor information. Its functions include knowing when new information has been inserted and removing information regarding the actual sensor data. The information that is processed by this layer comes from the sensors.

Sensor Data Harmonization Layer, is responsible for retrieving the sensor data from the previous layer and deploying it to the Sensor Repository. To achieve this, there must be a pre-processing where the different types of information within the sensor message are identified. This information is subsequently analyzed and subjected to harmonization. The harmonization process, refers to the different cases that will be presented in the data harmonization section. Once the information is harmonized and consistent with the structure data format, it is deployed in the repository and is available for higher layers.

The Aggregation / Reasoning Context layer finds the rules responsible for verifying whether the context that "judges" sensor data is verified or not. To achieve this, the layer identifies the rules that act on a particular sensor and, from them, obtains the context that encompasses the rules. This context and rules are those available in the rule repository. The rules are checked one by one, if they match, the context is checked successfully if the context is not failed. Information about whether a context is checked is available for the Application Layer.

The Application layer is the layer responsible for creating and deleting rules and context elements. This information is retrieved from a human user and stored in the corresponding structures (depending on whether the element is a rule or a context). Information about the registered elements is available to the user. This layer is also responsible for letting the user know when a context assessment is triggered and what the outcome of that evaluation is.

3.2.1 Acquisition Layer

Acquisition Layer retrieves the information that is received from the sensors, processes that information and sends it to Data Harmonization Layer. For that reason, it is necessary to define how the event of receiving data will be handled. In (Perera, Member, Zaslavsky, & Christen, 2013) events are considered to be occurrences that trigger conditions in a target area, they can be one of two types:

- Discrete Event: An event that occurs at a certain time (t) and at that certain time plus a time period (t+p). Because in these kinds of events exists a time period (p) between the first and second occurrences there are considered to be two different event instances.

- Continuous Event: An even that occurs at a certain time, where t and t+p cannot be considered a valid measured interval and consequently is only considered one event.

Because sensor information is delivered only when a farmer decides to measure the information from a certain sensor or group of sensors (i.e. using a button), Acquisition Layer has to treat that information as Discrete Events. To deal with this type of events, an approach based on an Event Driven architecture can be used. This means that deployment of sensor information will trigger actions. As seen in (Tan J.G., Zhang D., Wang X., 2005) this approach offers high performance in terms of flexibility, scalability and processing time. Being event driven, processing can be triggered according to user's demand, which result in the saving of resources. Based on this approach, the architecture of this layer was created, and its behavior can be seen in figure 3-7.

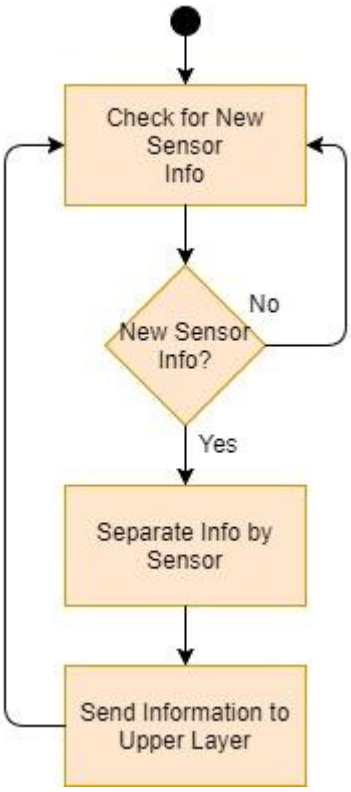


Figure 3-3: Acquisition layer behaviour

In this architecture, the system expects a new data deployment and checks, in (large) intervals, whether new sensor information has been implanted. When new information arrives, its processing begins. The incoming information can belong to several sensor types, such as measuring sensors and readings from laser counting sensors. This information can even be mixed and, for this reason, this layer must process the information, separating all information into individual sensor data. This information is then sent to the Sensor Data harmonization layer and the process of waiting for new sensor restarts.

3.2.2 Data harmonization Layer

The function of this layer is to treat each individual sensor reading, to harmonize it according to the structure of the sensor model and to store it in the sensor data repository. To achieve this, it is necessary to explain what a good set of sensor data is and how it can achieve interoperability. In (Symposium, Iot, Data, & Formats, 2015) are explained some common guidelines that can be used to obtain sensor interoperability. One such guideline is to use key pairs of values, expressed in readable form such as JSON. As seen in (Json, 2012), JSON can process a text format that is a stand-alone language, which means that it can be used regardless of the programming language. It is also a lightweight data exchange format, easy to read and write, and this makes it a good choice to analyze the sensor information. Other guidelines include the use of common naming keys such as name, ID, value, unit, and so on. Figure 3-4 describes a convenient way to retrieve and read the sensor key pair values:

```
<”sensor GUID”: “xx”>, <”reading”: “yy”>, <”units”:”uu”>
<”time”:”zz”>, <”location”: “g,e,o”>, <”meta1”: “abc”>,
<”meta2”:”def”>,…
```

Figure 3-4: Key par values example from (Symposium et al., 2015)

To define which set of items provide a minimalist set of interoperable features, it is necessary to first define what the application needs to read the sensors. In addition to the data readings collected by the sensor applications, there is also a need to know parameters such as the sensor identifier and the capture date. On the other hand, the applications that analyze the information collected need to know parameters such as type and unit.

When a request is made to a sensor, the information collected does not contain just the required pair of keys. When considering the wireless sensor based on XMPP should be considered a much greater response. XMPP (Extensible Messaging and Presence Protocol) is a protocol used for real-time information exchange on the internet. By applying this protocol to wireless sensors, it is possible to exchange recovered data between the user and a group of sensors as seen in (Hornsby, Belimpasakis, & Defee, 2009). An adapted example of (Waher, 2016) is shown in Figure 3-5.

```
<message from='device@example.org'
  to='client@example.org/amr'>
  <fields xmlns='urn:xmpp:iot:sensordata' seqnr='1' done='true'>
    <node nodeId='Device01'>
      <timestamp value='2013-03-07T16:24:30'>
        <numeric name='Temperature' momentary='false' automaticReadout='true'
value='30’ unit='°C'/>
      </timestamp>
    </node>
  </fields>
</message>
```

Figure 3-5: XMPP sensor response

In figure 3-9, it is possible to observe fields that represent sensor information. The from and to fields refer to the device that is sending the message and the client who is receiving it. The node field tells what is the sensors identification within its owning node. Depending on the network size is possible that exist two sensors with the same id, which is not necessary a problem since that on the message request is specified the required node. The timestamp value refers to the time when the readings were retrieved, and between its brackets rest the readings and some additional information. Although this fields are specially important to VHarvest it is necessary to reference them, since they exist on the data sent from the Acquisition Layer.

From the 3-9 it is easy to verify that besides the necessary information for our framework exists also several data that is not important, this fact is taken in account in the layer architecture depicted in figure 3-6.

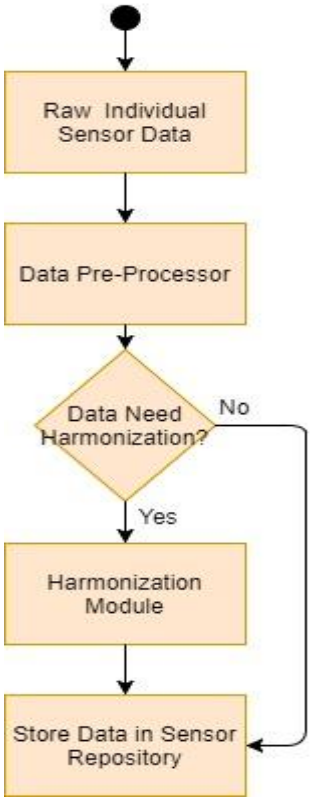


Figure 3-6: Data Harmonization layer behaviour

The Layer's work starts with the Raw Sensor Data delivered by the previous layer. This information passes through a preprocessor so that key information can be retrieved. As seen before, within the message of a sensor, there is much information that is not. The function of the Data Pre-Processor module is to eliminate the excess information. Considering Figure 3-9, its role is to obtain, in a first phase, the content between the timestamp supports. In a second phase, the values contained in the brackets are analyzed and stored in memory.

The retrieved information can then be stored in Data Repository, if it is already in a format compatible with the framework's data instance or else pass through an harmonization process. The frameworks representation of a sensor consists in a minimalist set of information, with the purpose of easing the integration process. It consists of a table called SensorReading that only contains an ID and value, this small amount of information for a representation solves by itself some of the possible information exchange mismatches. The harmonization consists in a series of functions, each one designed to solve a particular mismatch problem. According to the number of different mismatches, it can occur that a sensor passes through several harmonization functions.

3.2.3 Context Aggregation/Reasoning Layer

Context Aggregation/Reasoning Layer has two main functions, on a first phase it finds all the rules that evaluate a group of sensors, on a second phase executes the rules verification and reasons whether the context is verified. To understand how the rules

aggregation and context verification work it is first necessary to understand what is the framework's representation of context and rules and how they interact with sensor's values representations. The 3-7 depictures this relation:

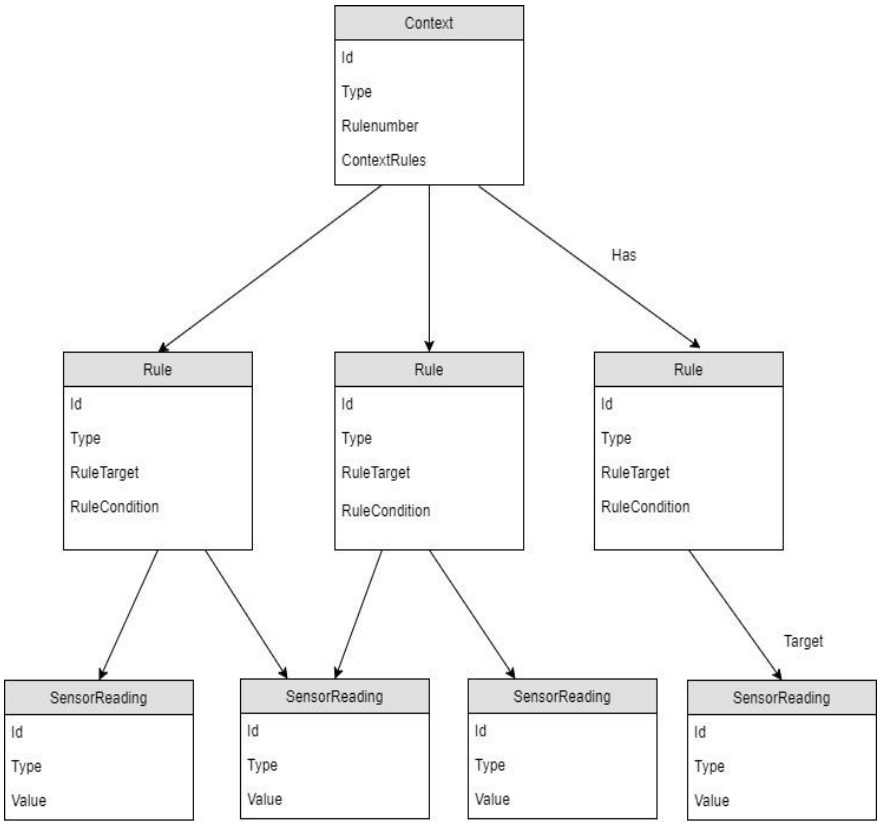


Figure 3-7: Context, Rules and Readings relation

The context's representation contains an identificatory field with its name and also Type, Rulenumber and ContextRules fields. The type field identifies the context element and, allows the system to understand if it is a context element or a rule element. It is necessary to have the type attribute because, it may ease further processing, when selecting the context and rules to reason. Rulenumber identifies the number of rules that a context needs to be verified. Imagine as an example, the measuring of a context for whose is necessary to have two different rules, its verification is far different from checking a context that only has one rule. ContextRules contains a list of all the rules existing in the system, under a defined Context.

Rules representation contains four different fields which, are ID, Type RuleTarget and RuleCondition. As it happens in the context representation the ID field contains the rules name. This field is unique, which means that there can not be two rules with the same name within the system. Type field works the same way as in the previous case, which means that all rules will have this field with "Rule" value. RuleTarget specifies the type of sensors under a specific rule. Each rule can only be applied to a single type of sensors, this implies that to evaluate two different kind of sensors will be needed two different rules. Although for a n number of types will be needed n number of rules, the opposite is not verified, different rules can evaluate the same Sensor type.

The Sensor's representation is called SensorReading and has already been introduced in the previous section, it is a minimal representation, containing only an ID, Type and Value. The ID value identifies the Reading and is composed of the sensor name plus the reading number. As an example, the ID of a sensor called TemperatureSensor would be "TemperatureSensorxxx", where xxx would be measure number (001,002,003, etc.). The type field contains the Sensor name, considering the previous example its value would be "TemperatureSensor". The Value field contains the valued read by the sensors at a determined time moment.

As it is possible to verify by the previous image information is compartmentalized, this means that a context element only knows what are his rules, rules know their corresponding context and SensorReadings to target and, SensorReadings don't know any of the other entities. The process of rules evaluation and context verification can be observed in figure 3-8.

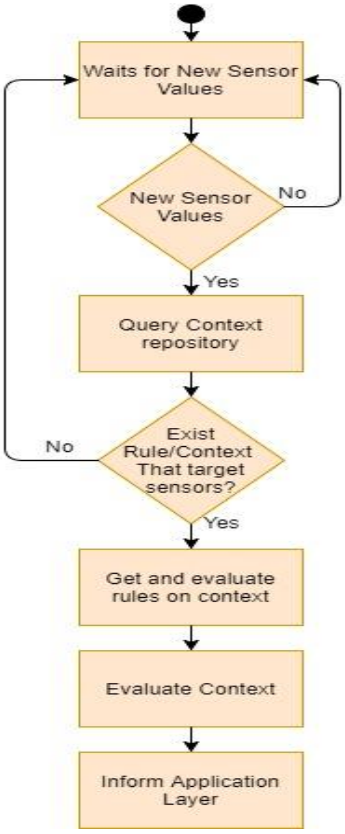


Figure 3-8: Context Reasoning layer behaviour

The process begins when Sensor values are sent by the previous layer, the system receives them and, searches in the context repository for a rule that targets the readings. If a rule exists, all contexts that depend on this rule are taken from the repository along with all other rules. Using query features, all SensorReadings whose type corresponds to the alignment of the rules are queried. From there, the values are checked according to the corresponding RuleCondition and the system records the number of positive cases, in the end the number of positive cases is compared with the total number of readings targeted by that rule. The result of the rule is then saved and, if all the directed cases are positive, the rule will be checked, if it does not fail.

This process repeats itself for as many rules as the needed for verifying a context. If the number of verified rules is less than the number of rules required by the context (can be verified by the field ContextRules) the context fails immediately and this situation is reported to the Application Layer. If that does not happen, system proceeds to a normal context evaluation based on the results from the previously evaluated rules. If any of the rules has failed the evaluation determines that the context fails, otherwise it succeeds. Despite if the context fails or succeeds the information is always sent to the Application Layer so that the user understands how the context behaved.

3.2.4 Application Layer

As introduced before, the Application Layer has three main functions which are, to insert rules and context into the repository, inform the user about the previous layer analysis and to communicate with other Vf APPS. The insertion of rules and context, starts with the processing of users that is submitted to several evaluations, ending with the element submission and information to the user This process is explained with more detail in figure 3-9.

After data is retrieved from user, it is submitted to a first evaluation to determine if it is suitable for the preceding processing. This first verification checks if all the necessary information is delivered to the system and prevents further wrong queries when retrieving data. Depending on which is the element to process (rule or context), the steps to take are slightly different.

When the element is a context, the first step to take, after validating the user input, it is to query the context repository and to check if already exists a context with the provided identification. If that context exists, because there cannot exist two context objects with the same ID, user is informed that he must insert new data.

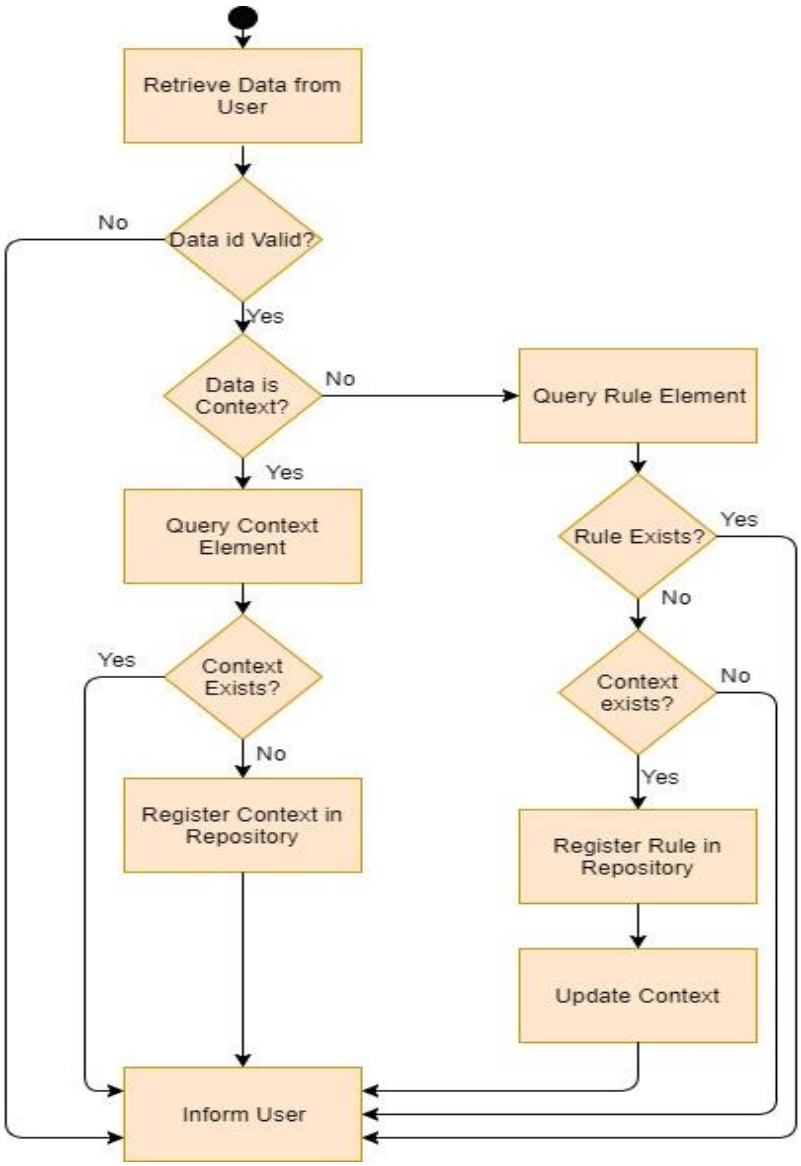


Figure 3-9: Application layer, insert elements behaviour

If the context does not exist, the system proceeds to his registration in the Repository, using the previously validated user input. After this whole process, user is informed that the input data has been registered as a context.

As it happens with context, the first step to take after verifying basic rule data is to verify whether it exists within repository. If the rule does not exist the user is notified, otherwise is verified if current context exists. This verification is necessary because no rule can exist without having a corresponding context. If the corresponding context exists, its ContextRules field is updated (the new rule is added to existing list). The new rule is then added to the repository and the user is notified.

The process to delete an existing element is very similar to the process of creating new elements, and that is why that is not necessary to show its diagram in this section. This process begins with a basic data verification to check if the required information is present. The rest of the process occurs like the element submission, with small changes. When deleting a rule, there is no need to verify if a context exists because that verification has already been made upon the element's submission. Context element is still updated, in this situation the corresponding rule is deleted from the list. Instead of inserting that element in the repository, it is deleted.

The process of telling the user if a context has or not been verified and, what is the result of the evaluation, uses the same logic as checking if a new sensor has been deployed. Communication with other software that allow to address other parts of the complete scenario is also made through this layer.

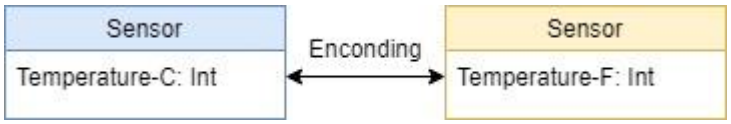
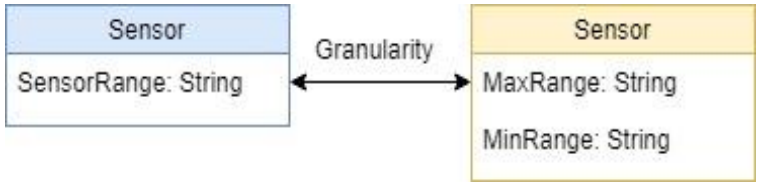
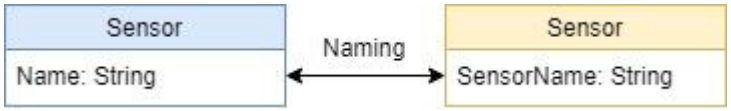
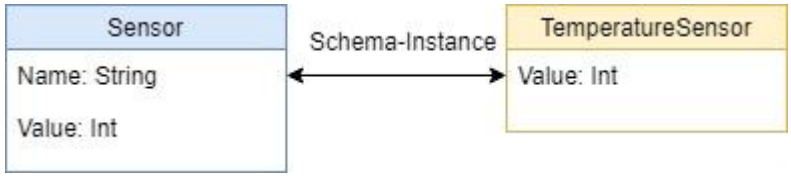
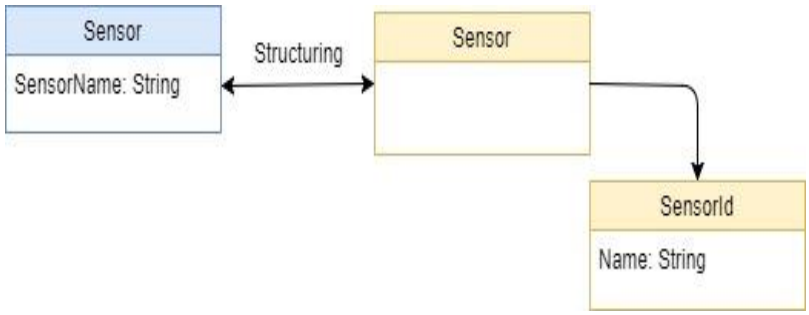
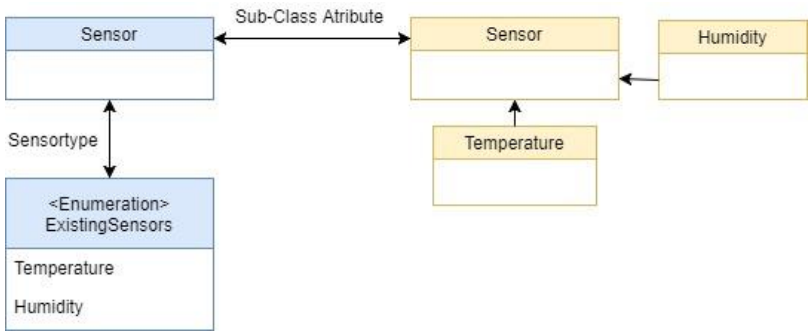
3.3 Data harmonization

To achieve an interoperable framework, it is necessary to prepare the harmonization module to deal with sensor data from different sources. It can occur that the current sensor, representation within the framework, is not compatible with the sensor data model that needs to be analyzed, this is called a mismatch. A framework can address heterogeneity problems, by following the generic steps introduced in 2.5.4. This means that is necessary to identify the possible mismatches between the origin and destiny data models, interpreter that information and propose a solution for each case. The sensor data mismatches deal with the data model incompatibilities between sensors and, harmonization guidelines provide solutions based on each case analysis.

3.3.1 Sensor data Mismatches

When receiving sensor information, it is necessary to store it within the framework's corresponding structure. Since sensors' information structure can be different from the defined for the structure on the framework, there can exist some mismatches when exchanging the information. In (Agostinho, 2011) are pointed and explained the most common mismatches that can occur when exchanging information from different data structures. According to the type of mismatch it can be considered as lossless, when the relating element can completely understand and capture data target's information, or lossy when that does not occur. The tables 3-2 and 3-3 were created, based on (Agostinho & Malo, 2007) and on the work of (Ferreira, 2012) and represent lossless and lossy information, considering sensor's exchange of information.

Table 3-2 – Lossless sensor mismatches.

Mismatch	Example
Encoding	
Granularity	
Naming	
Schema Instance	
Structuring	
Sub-Class Attribute	

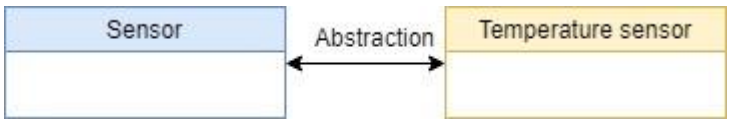
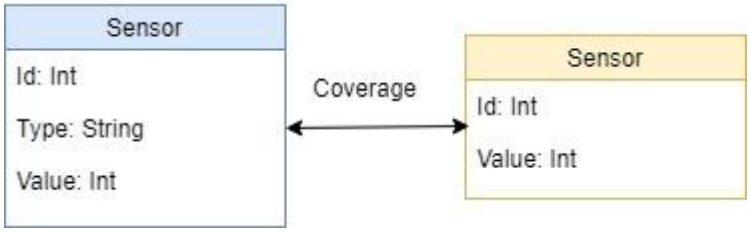
The table 3-2 describes lossless mismatch cases, meaning that even if information can not be passed through without harmonization, that information exists in data destiny and

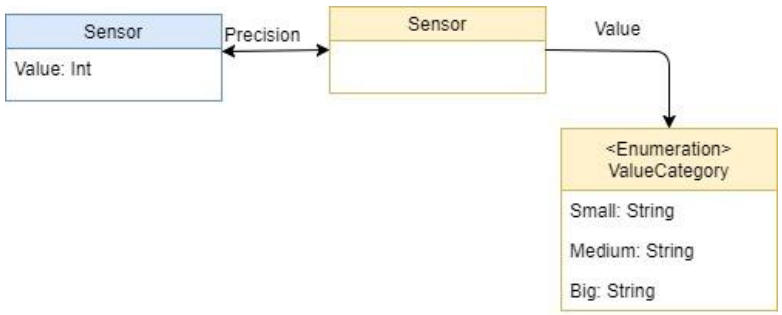
source tables. Encoding mismatch occurs when the units from data source are different from destiny data source. In the presented example two sensors measure temperature, storing that value in different units (one of them stores the information as Celsius and the other one as Fahrenheit).

Granularity is related to information that is present as a single field in a structure, being split in the target structure, or vice-versa. In this example, a sensor's range is present in one case as an interval (for an example a string containing "0-50") and in the other case, that interval is stored in two different fields (one containing "0" and the other "50". In Naming mismatch, different labels are attributed to a field on the structure that represents the same information. In this example, the field containing sensor's name has different names in each case.

Schema instance mismatch happens when a field in one model is part of another field's model. On this example, the ID of a sensor is defined in one table as its type and its ID (for example "TemperatureSensor1" and in the other table a TemperatureSensor is defined by its id. Structuring is a mismatch where exist different design structures for the same information. As an example, a sensor can have its name defined in a single structure or defined in a different attributes table. In Sub-Class attribute mismatch an attribute that has a predefined value set is defined by several subclass structures. On the provided example one sensor has its types in a SensorType structure and the other has each type as a different table.

Table 3-3 – Lossy sensor mismatches.

Mismatch	Example
Abstraction	
Content	
Coverage	

Precision	

The table 3-3 describes lossy mismatches. Abstraction mismatch occurs when exist different levels of specialization. The provided example represents exchanging information from a standard sensor to a specialized one. Although in some cases a specific sensor's information can be passed to a general one, the opposite is not true. Content mismatch is related to the existence of different context denoted by the same concept. In the example two tables with completely different information are identified by the same name. In coverage mismatch exists absence of information. In the example, the fields existing in the source table don't exist in the destiny or vice-versa. Precision mismatch happens when exist different levels of information accuracy. As seen in the example one of the entities comprehends values as integers and the other has a list of interval values.

3.3.2 Harmonization Guidelines

In this subchapter, are presented some guidelines to adapt this framework, to achieve interoperability. As it has already been explained, this framework aims to be used with several different sensors and sometimes the existing sensors may not be immediately suited to be used. In section 3.3.1, were identified the major mismatches that can occur when using different sensors with this framework, this guideline provides the steps to overcome that mismatches.

Encoding - this mismatch occurs when the units of a specific entity have different unit representations. Considering this framework, that can only happen between the target sensor representation and the Value field in SensorReading entity. To get data harmonized it is necessary to provide the framework with an expression that represents the conversion between units.

Granularity - The Granularity mismatch occurs when a source entity has an information field that is composed by several fields in the target entity. In this case, because the sensor reading structure has already been defined, the field composed by several fields can only be present in the sensors representation entity. To overcome this mismatch, it is necessary to identify the several necessary information fields and associate them to this framework's match.

Naming - In this mismatch the names of a specific entity have different corresponding designations. To harmonize this kind of mismatch is necessary to find the corresponding name within the entity and associate them.

Schema - Instance- Occurs when data within a field in one model is part of the entity information (such as name) in another. Considering the example in table 3-2, this would happen with a type of sensors that didn't present the name information within the expected brackets (considering figure 3-5 that corresponds to the "timestamp" brackets). To harmonize that kind of data, it is necessary to identify the place where that information is placed and match it to the variable that inserts that information in our readings field.

Precision - This mismatch is related to information exchange when exists a different level of abstraction between the two entities (as an example, when one displays a field as an integer value and the other as an interval of values). In this, case it is necessary to have a key to decodify the interval values, that key is applied to the sensor the same way it is done with Encoding mismatch.

There exist some cases in which mismatches won't be considered, one of that situations is the sub-class attribute that happens when a field within a table is part of another table (view table 3-2 for clarification). This case is not considered because the defined structure for the readings has a single table and, most of the sensors that exist in the market exchange information through IoT protocols, regardless of internal structure. This means that all the information is available in the request response such as the one shown in table 2.2. Mismatches like the Abstraction one are also not considered. These mismatches occur when there are different levels of specialization and mean no harm because since all the fields existing within the structure are in the request response, it does not matter if the sensor is specialized or not. Content mismatch occurs when the fields within the structures have different types (string and integer as an example). This mismatch is solved by the way the framework receives the information, since all the data is threated as strings and all numbers need to receive a cast. Sensors that present Coverage mismatch, in which a field is missing from one of the entities are not considered.

3.4 Vf-Os project

VF-Os has as main goal to become a reference regarding Virtual Factories Operating Systems. To achieve this goal the vf-Platform may count on vf-SK, vf-MW and vf-API. Vf-SK is a virtual Factory System Kernel, vf-MW represents the middleware and vf-API a Virtual Factory Application Programming Interface. These three elements can be used by Manufacturing and Logistic users that aim to explore already created solutions and adapt them to their business area. Manufacturing and Logistic Providers will be in charge of providing Information Technology and Communication(ICT) interfaces and manufacturing connections. A high-level

view of the different vf-OS layers and it's integration within the larger platform can be seen in figure 3-13.

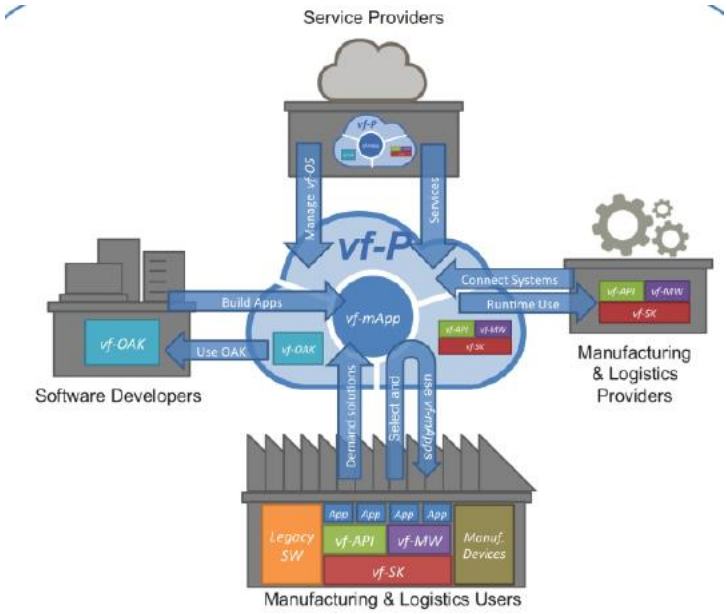


Figure 3-10: Vf-Os project overview (Lead, 2017)

Vf-OS uses concepts related to IoT, so that it can operate within a network of collaborative manufacturing and logistics environment. By enabling the connection of applications people and devices the vf-OS project will allow to implement FoF concepts in an IoT perspective and therefore creating faster and more efficient systems. As explained in the previous section, industry nowadays is based on centralized systems. By linking existing ICT technologies with the Vf platform it is possible to create a decentralized system that overcomes the actual industry limitations. This means that The VF operating system will be the intermediary of factory applications and devices, being also responsible for managing the virtualization and data exchange of physical elements(hardware), as it would be made in a regular operating system.

3.4.1 Vf-OS APPS

To solve current common industry issues/problems, were idealized several use case scenarios that integrate both industrial and user scenarios. They propose to produce advanced technical solutions to some of the existing industrial issues by developing suitable applications, the corresponding user scenarios are described in table 3-4 through well-defined objectives, processes, actuators and possible sets of data(Lead, 2017).

Table 3-4 – Vf-OS use case scenarios

Acronym	Description
vOrder	To handle customer orders that can be shared between order manager / production manager or ordering departments / financial department or directly within a supplier (it depends on the usage). This app can also be used in the returning process by clients. They can take a photo of the received faulty product and ask for a return.
vProductMon	For real-time monitoring on the status of a production, having the possibility to identify flaws and inform production managers that can immediately react.
vfSalesLead	To help a salesperson to identify sales leads in his region, and segment territories into employee count, competitor, and location.
vfColPlan	Provide high innovative tools to compute collaborative plans and optimize collaborative provisioning, manufacturing and supply processes.
vfNegDemand	Visualize and negotiate demand plans in real time. The application connects to production plans and data is shared with providers, in order to validate the demand plan, supporting on line negotiation
vfMan	Integration of CPS concepts to identify unexpected manufacturing events, the estimation of their impacts (in terms of quality, time, and quantity) and the decision of next operation
vfPhyt	Monitors consumption of phytosanitary products in agricultural productions to support demand management, taking into account quality requirements for the final product
vfHarvest	Optimizes the harvest process integrating data from crops, logistic and manufacturing process to optimize resource utilization and final product quality
vFail	IoT application for the automatic registration of spare-part failures in automation production equipment
vfPayment	Allow payments to be made after a negotiation process by integrating different payment gateways
vfMyCon	Interface with smart meters to provide energy consumptions and saving strategies
vfColPurchase	Provide option for collaborative purchase (reduction on logistic costs and better deals with suppliers)
vf3DViewer	To allow production employees find and view product parts via interactive 3D images
vfProducts	Provide tools to store all relevant documentation regarding the manufacturing of products
vfAdaptation	Provide a list of best practices and workflow processes to perform when failures and monitor alarms occur
vfNegotiation	Competencies and resources sharing. A negotiation support environment for the co-creation of products and business services

From table 3.1 were chosen the generic apps that could be used to solve the previously described scenario. The applications chosen to solve the previously presented scenario are vfNegotiation, vOrder, vfHarvest, vFail and vProductMon.

VfNegotiation is responsible for clients' buying orders. The buyers can choose, from a list, between all resellers the one that fits more efficiently his needs. The main point of this vf-APP is to make better business choices between all the agents in the negotiation. VOrder deals with the tracking during transportation and monitors success delivery. It is a platform where the farmers can control the products they dispatched , verify the transport conditions. It can also be used by the transport providers in order to create and check trucks within the system. VfHarvest deals with the production line, it deals with sensors and grants that they are available to the system. It is also responsible for making data available to the system and react when there is a sensors failure related to hardware. Finally the modules that are going to be developed within this this work are explained below:

- Vfail: Identifies if all the requirements regarding a specific objective have been achieved. This tool will act whenever new sensor values are retrieved and will find all the context that affects them. This provides an extra insurance regarding the product quality that aims to help a producer to have a higher quality level compared to other fruit producers.

- Vproductmon: Allows a user to submit context and its corresponding rules within the system, which means that he can have control above every quality checkpoint within the system. Because vProductMon communicates with vFail it will display any context evaluation in real-time.

3.5 Proposed Architecture

The framework that was explained in 3.2 will be built according to the scenario's needs and, to the necessary vf-APPS. The previously presented layers will be split among the two apps. As seen in 3.2 the framework's layers are Acquisition Layer, Sensor Data Harmonization Layer, Context Aggregation/Reasoning Layer and Application Layer. The three first layers will be provided by VFailure and Application Layer by VProductMon. Figure 3-11 shows the proposed architecture for the overall system.

Fiware Orion will be used to reason the received data, based on the defined rules. All stored information (context, rules and sensor values) will be access through Orion. This enabler will be run from inside a docker container (section 3.7), following the logic that will be explained in 3.6 .To interact with Orion will be used Rest requests, that will be explained in 3.6.1. This means that the operations usage must be defined within the APPS.

The sensor data evaluation will be made using Vfail APP and, following theories proposed by the three level model. The rules and context elements can be defined using the VproductMon APP and their current information can also be accessed from it.

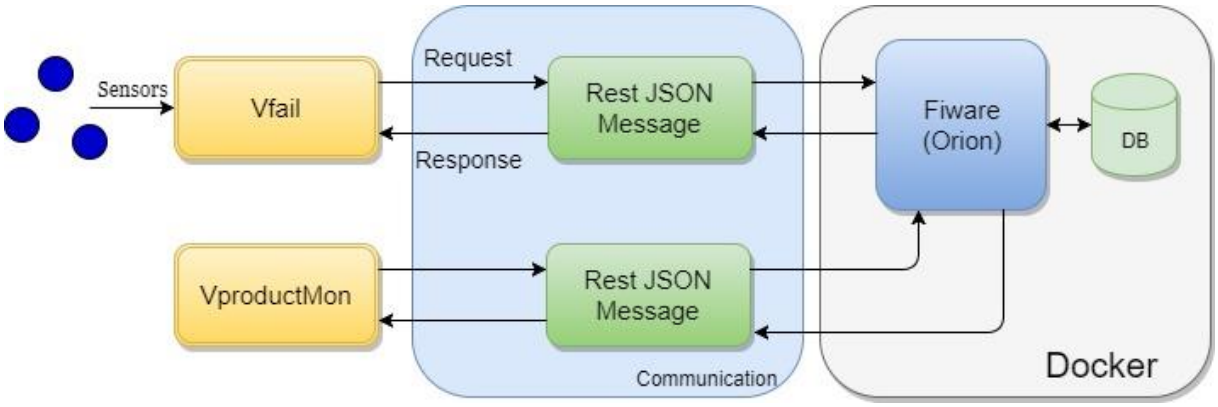


Figure 3-11: Proposed System Architecture

3.6 Orion

To be able to infer if a context is valid or not, based on rule interpretation it is necessary to use a platform that manages context. Researchers are already developing context reasoning engines that can solve specific scenarios issues. Ongoing projects include a body sensor network that monitors a patient's health based on the context inferred from medical sensors (Lo, Thiemjarus, & King, 2016) and a platform that can reason about social contexts for Socially-aware applications (Restfulapi, 2017). Although these platforms can successfully reason context within their own application resources, there is a lack of a platform that is truly adaptable to different scenarios without major modifications. To achieve an adaptable platform is first necessary to choose a reasoner that efficiently manages different information, a reasoner like fiware Orion.

Orion Context Broker is a Generic Enabler that allows to manage context and manipulate information, as seen in (Fiware Docs, 2017). Therefore, this GE can be used in a data/context scenario to perform information updates, queries, registrations and subscriptions using NGSI9 and NGSI10 interfaces. NGSI (Next Generation Services Interface) were developed in order to integrate physical devices using an IoT approach, these interfaces allow the making of a transition from device-level information into thing-level information. Both NGSI9 and NGSI10 can be accessed using http protocols however, their main purpose/usage is not the same . NGSI9 is used to verify the context availability and NGSI10 aims to perform exchange of information, this can be seen in (fiware.org, 2017).

In order to represent physical objects, Orion uses entities that help to successfully create a virtual representation of the real world, each entity has an identificatory and a type.

As an example, an entity virtualization of a book can have as ID his title and type “book”. Entities can have several attributes that describe its characteristics, each attribute has a name a type and a value. Following the previous example, a book chapter can be described as an attribute, the chapter title would be the ID, the type would be “chapter” and as value the matching pages. One of the main advantage of using attributes is that besides entity data, can also be incorporated metadata, which provides additional information regarding an attribute or even another attribute. One example of metadata is the time when an entity is updated, this is of major importance because can for example help a querying function to select information obtained between two periods of time. Attributes can be found using attribute domains that group them according to common sets. However, this function is being depreciated as the attributes type can also be used to query them. To store information about elements, exist Context Elements. They contain the names of attribute domains and a list of characteristics that are applied to all attribute elements(fiware.org, 2017)[i].

3.6.1 REST API

Orion can be accessed using REST API that is based on the concept of Software-Defined Networks(SDN), this kind of network decouples the central control elements that would be present in an hardware controller and implements them as software as seen in (Kirkpatrick, 2013). This type of network provides a higher programmability and access to network administrators, without the need of accessing physical hardware devices.

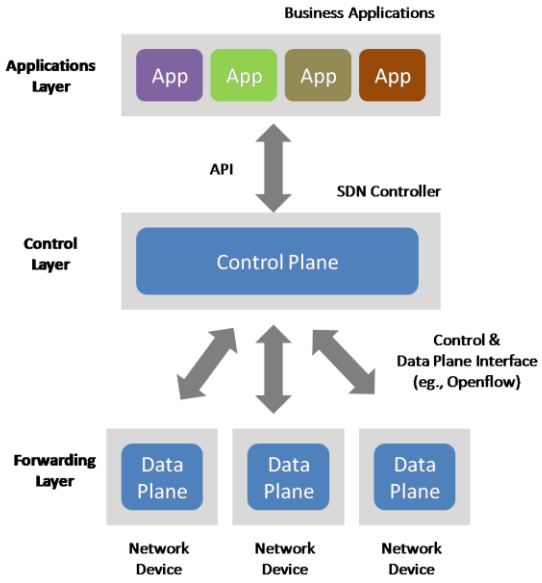


Figure 3-12: SDN architecture(Velrajan, 2017)

In SDN networks the control pane and data pane are enclosed by a control loop. During this loop, the control plane is able to receive and access network events from within the data pane. Based on that events, control is capable of processing computer network operations that are later executed and being able to change network states.

REST is an architecture style that can be used in the design of networked applications and that is increasingly gaining importance within the SDN architecture. According to (Zhou, Li, Luo, & Chou, 2014) it has several benefits:

1. The managing of dynamic resources is decentralized, REST relies on connections between existing resources to manage them as a whole. Its decentralization is achieved by allowing network elements (such as routers and switches) to be dynamically deployed and changed.

2. REST architecture allows the existence of several users, despite the platform that they are using. The resource representation, identification and interaction are separated and REST can adjust resource representations and network protocols based on SDN client capabilities and network conditions to optimize API performance.

3. Provides services independent of the programming language used. A developer creating a Java application can access REST services the same way that a developer using languages such as C or C++ would. This important feature helps to achieve interoperability.

4. Supports backward compatible service migration, this means that migrating a resource or adding a new one will only affect the resources that are immediately connected to it, all other resources remain unchanged. This feature combined with a uniform interface and hypertext service discovery helps to become easier the deployment of new information within the system without disturbing it as a whole.

5. Uses layered caches that allow to improve server stability. That feature is important in a scenario where an SDN controller has the need of supporting several concurrent host-based applications, contributing to the service stability.

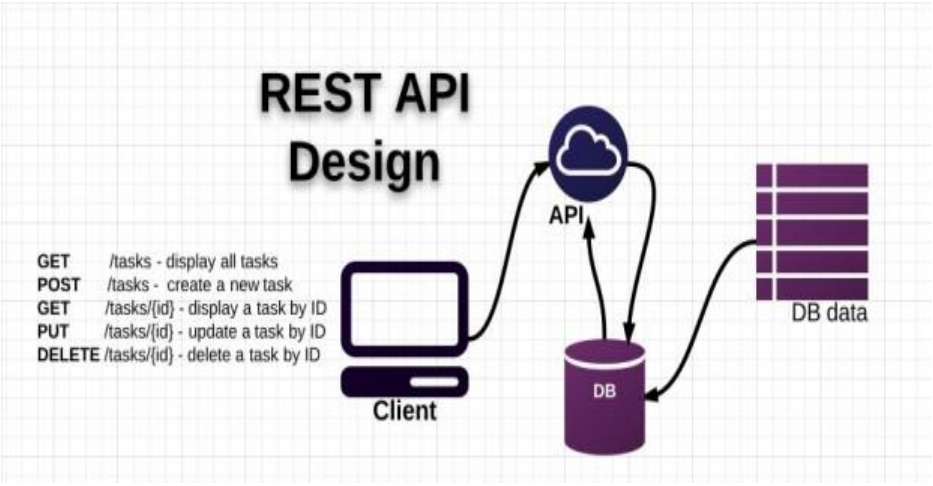


Figure 3-13: Rest API Design (DZONE, 2017)

To develop web applications, REST possesses several operations, each containing its own information such as description, request and response message, which are used to perform changes within a service. Those operations are GET, POST, DELETE and PATCH and its usage is depicted in the figure 3-11. GET is used to retrieve resource representation and information, it does not modify any of the retrieved data. GET APIs are called idempotent, that means all the information collected from multiple requests will always be the same until the server receives a POST or a PUT that changes it. POST creates new subordinate resources (for an example a row is subordinate to a database table) and are used to create a new resource into the collection of the existing ones. When performing a POST operation into a server the client receives a delivery code, informing if the information has or not been successfully posted. PUT APIs have as objective to update an existing resource, usually if that resource does not exist, the API can create it the same way that a POST would do.

When a new information is created from a PUT client, it receives a different status code so that he can know how the way information has been modified. The difference between the POST and PUT APIs can be observed in request URIs. POST requests are made of resource collections whereas PUT requests are made on individual resource (Restfulapi, 2017). PATCH requests make partial updates on resources and are to be used in situations where another type of request (PUT) would make too big changes in information. For this reason, PUT should be used when replacing a whole resource and PATCH should be used when changing a resource attribute. Finally, the DELETE request is used to remove a chosen resource. This kind of HTTP request is also idempotent, if it deletes a resource that resource is completely removed from the corresponding collection. If a DELETE request is called two times in a row for the same object, or if its target does not exist, the corresponding error message is delivered to the user.

Because ORION is based in SDN architecture, HTTP requests are used to get information from it. Depending on the type of request, ORION receives the corresponding message, decodes and processes it, leading to the performing of chosen action and response of the status code that indicates success or failure of the operation.

3.7 Docker

To run Orion on a personal computer or public server it is necessary to have access to its virtual instance, a possible solution for that is to use Docker, the same way it is done in (Stubbs, 2015) for microservices , presenting an alternative to actual Virtual Machines(VM).

Docker is a platform that lets user run an application in an isolated environment, which is called a container. Providing isolation means that it is possible to run multiple containers at the same time on the same host. The life cycle of any container created is ensured by the Docker platform. A client can develop applications using containers, which become the unit to test them, later this application can be placed in the chosen environment

as a service or as another container. To run Dockers, there is a client-server application called the Docker Engine, which configures the required components after startup. It defines a command-line interface, which can be used by the user to configure the desired option, this interface is mediated by a REST API that processes instructions. Finally, the Docker Engine also defines a server (also called a process daemon) in which the REST API, command line interface, and all containers and images are executed.

Docker has a client-server architecture, which means the client communicates with the server, which is responsible for processing all the actions necessary to configure images and containers. The client and the Docker server can run on the same server, or a client can be connected to a remote one. Figure 3-14 illustrates Docker's architecture:

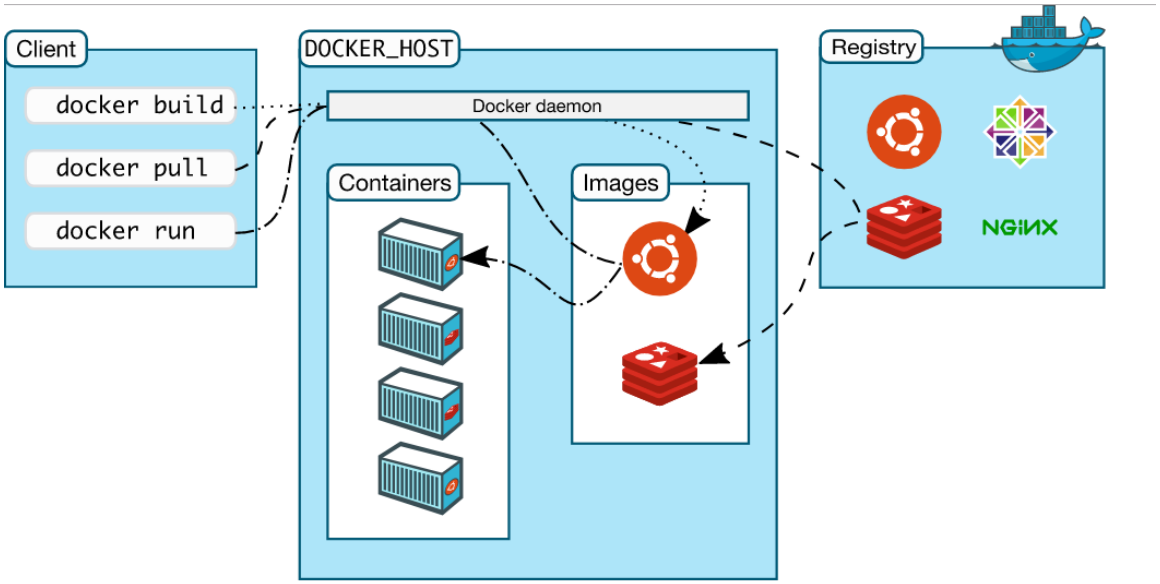


Figure 3-14: Docker Architecture(Inc, 2017)

As seen in Figure 3-14, Docker images are stored within registries that can be public or private. Public registries can be accessed from a Docker client, that is by default set to search them on Docker Hub, private ones are retrieved and pulled from a client's own private registry. The images that run on Docker server, are a read-only template that can be used for creating a container. Sometimes an image can be based on a previously existent one, that is customized according to the client´s needs. Containers memory and processor consume are very low, which occurs because they run directly within the host's machine kernel. This a very powerful feature and, means that even running a Docker container on another container will not make the system slow.

4 APPS Development

In this chapter, are explained the processes that allowed to create the prototype of a system based on the framework. It starts with the presentation of the application schema that explains each of the behaviors of the structures from an application point of view. In the first section the prototype overview is given, taking into account the architecture presented in the previous chapter. The second section presents the explanation of low-level functions / concepts that allow us to understand how the system processes REST requests and responses and how it manages incoming data into interpretable information. The other sub-chapters refer to the development of each of the vf-APPS functions. Images of the activity process refer to the logic used to create the prototype.

4.1 Applications overview

Given the required roles of the applications presented in the scenario from 3.1, and the applications from 3.5, was created the diagram from figure 4-1, that illustrates the applications relation. The applications marked in red are the ones to be developed in this work.

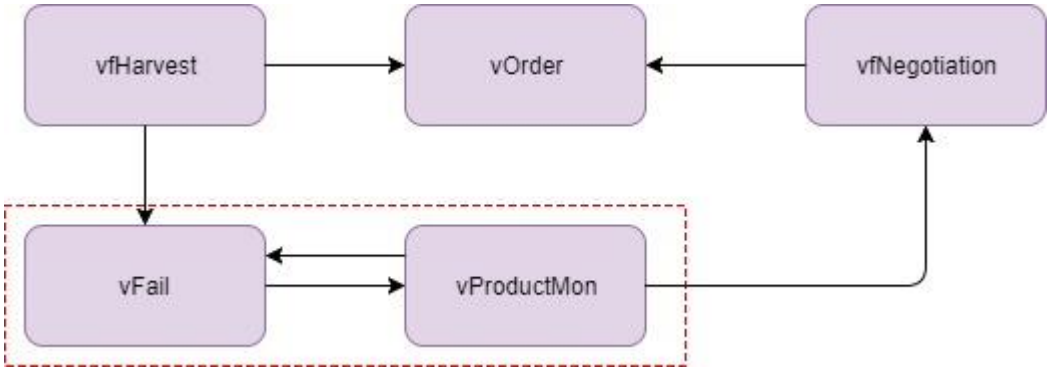


Figure 4-1: Necessary modules for the described scenario

Applications that are part of the framework system, need to receive raw sensor information from vfHarvest, analyze these data according to the context provided, and inform the user. When analyzing data related to a fruit size, the user will be informed about the actual amount of fruit available. In addition to informing the user, it is also necessary to inform vfNegotiation about this, so that a customer can know the current amount of fruit available.

Communications between vFail-vProductMon and vProductMon-vfNegotiation are made using the capabilities of Fiware Orion. Sensor resources and context rule repositories are also accessed using Fiware Orion.

The behaviors and functions of each layer, that make up this system, were explained in the previous chapter. In this sub-chapter is shown the process diagram of each APP (application view). Because the detailed working of each layer has already been explained, and the Scheme is based on them, this chapter will only explain the introduction of the rules in the repositories and how to use them to process the context assessment. The relationship and communication with Orion is also illustrated in Figures 4-2 and 4-3.

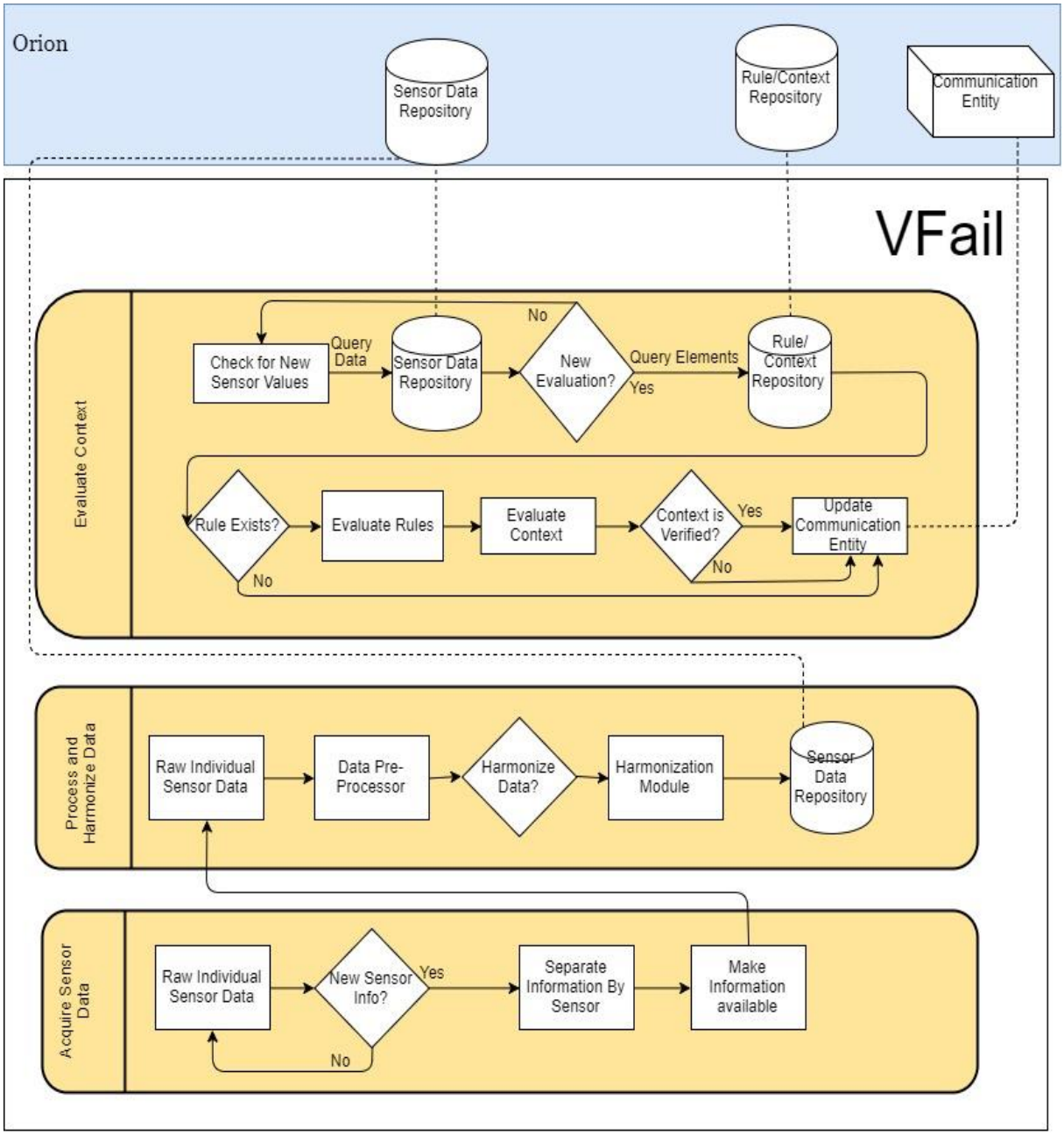


Figure 4-2: vFail application model

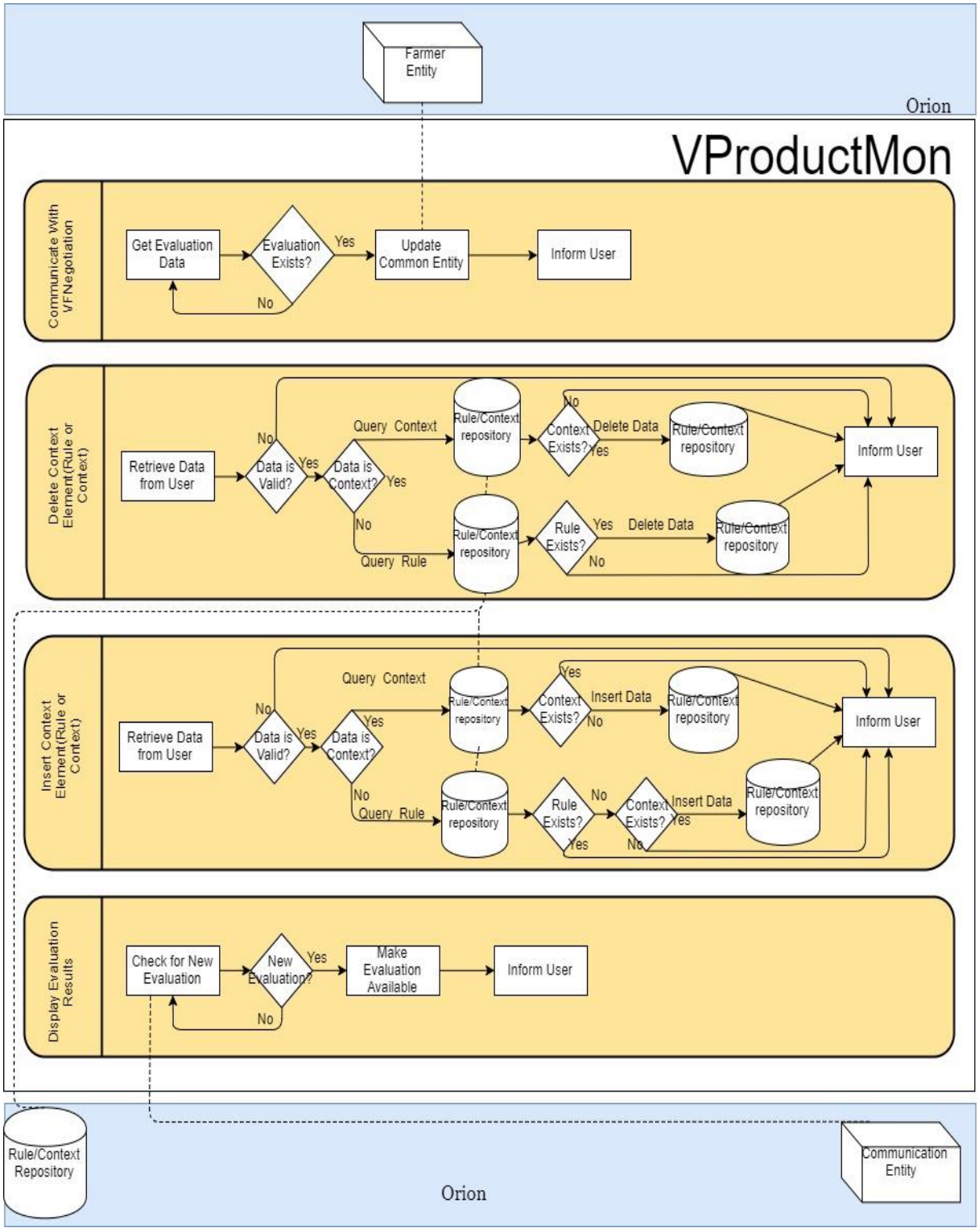


Figure 4-3: vProductMon application model

To evaluate a context, the user first needs to register it and at least one corresponding rule using VProductMon. Since no context still exists inside the repository, after the first data is checked, the context will be stored successfully. The next step is to introduce a rule in the previous context and update the context rules. As the inserted rule targets readings and specific sensor rules, are available in Orion for both APPS it is now possible to check the VFail context that has just been deployed in the repository. When the raw sensor readings are

implanted, they are separated by the sensor and the existing data is processed. If these data need harmonization, it goes through an articulation of functions contained in the harmonization module. The sensor information that can be read by the system is then stored inside the repository. When the data is stored in the repository, it is accessed and the rules / context that the target are selected. The system can then evaluate the sensor data based on the existing context, the result is then stored in the CommunicationEnty that can be accessed by the two APPS. This entity contains an ID, a type, and a content. ID is the basic identifier, the type contains the "communication" attribute and the content is the context evaluation. This entity is within Orion, which means that since each APP knows that its address information can be exchanged through it. VProductMon can display each of the evaluations, for this it uses the resources of Orion and verifies if the CommunicationEntity is updated. When this happens, it displays the result for the user and makes this information available for later use, the VProductMon application can then exchange this information with vFNegotiation. The exchange takes place through a farmer entity that contains and ID, type and various attributes of fruit. ID identifies the farmer, the type describes it within vFNegotiation and the attributes contain the amount of available fruit of a given strain and size. When data exchange is required, the result of the context is translated into numbers referring to the desired voltage and this information is written to the farmer.

4.2 Basic functions

As introduced in previous chapters, to interact with ORION the system uses REST requests that are parsed by the NGSI9 and NGSI10 interfaces. To interact with ORION, the author chose to use the JAVA code from (Amaxilat, 2017) because it already had the basic functions to interact with both NGSI interfaces. There are two different ways to handle ORION responses, one is to work on the serialized serial chain (JSON) and the other is to analyze information as JAVA objects. As the author considered working in String to be faster, choosed this method, however, because the framework is intended to be adaptive and interoperable functions to handle the responses as the objects have also been developed. When an ORION response is serialized, it looks like the one shown in Figure 4-4, which means that there must be functions that will retrieve the desired information.

Figure 4-4: Orion context element response (JASON response)

To use the values contained in the String, the first step is to delete all the quotation marks, from there it is possible to separate several responses from Context. Each response is an entry on the server (in this case a context), the String is processed row by row. The prototype has a class that deals with String processing and contains several functions, depending on the necessary field. Getting id and type it is easier than attributes, because the value is in the same line as “type”, to get attributes the program finds the desired name and then the corresponding value, by advancing two lines and separating the true value content.

4.3 Vfail

The following subchapters describe in detail the function of the Vfail prototype. Acquire sensor data details the process of receiving data from Vharvest and made it readable to the other processes. Process and Data Harmonization describe how the individual raw sensor data is processed to be stored within the repository. Evaluate context explain how the sensor data in the repository is related to the rules and how it leads to context Verification.

4.3.1 Acquire Sensor Data

The Acquire Sensor Data process receives the data from VHarvest and makes a small processing to make it readable for the other functions. To simulate the sensors from VHarvest It was used a text file containing raw sensor data as it was supposed to be received. To check when new information is received, a process verifies whether the file has been saved. When the file is saved, the rest of the processing is made, as shown in figure 4-5:

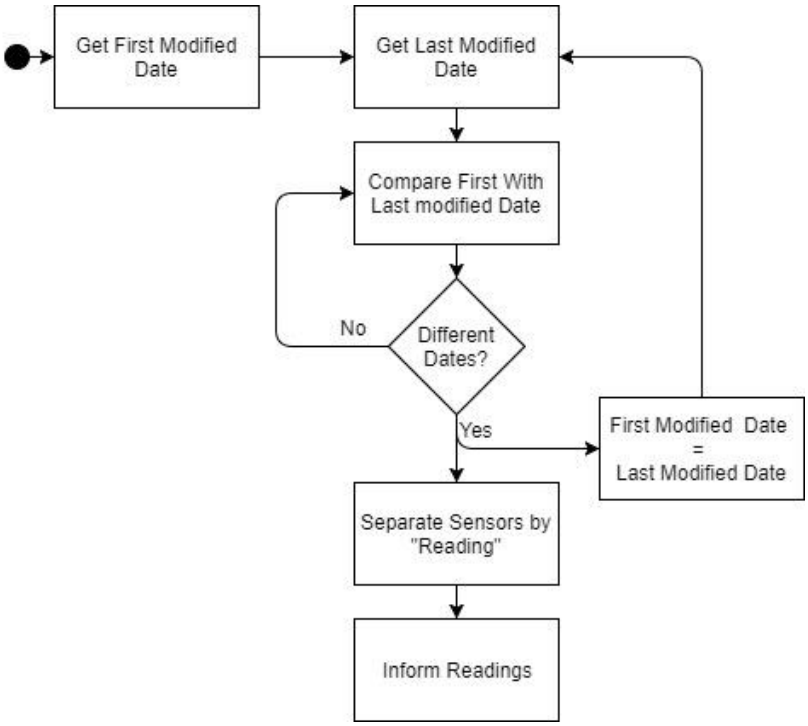


Figure 4-5: acquire sensor data activity process

The process begins when the application starts and obtains the current system date as the file was first modified. From there, it gets the last date when it was saved and compares it with the first date and, if they are the same, the process will be repeated until they are no longer. When the two dates are different, it means that the file has been saved, so the first modified date gets the value of the last one so that the process can be repeated. When the data comes from VHarvest, it contains many readings, however, each is separated by the word "read", so the system only needs to split the instances in that word. This was used as a generic case, if the sensor format is known, there is no need to "read" and the sequence may be the end part of the sensors' message. The separate information is then stored and sent to the next application.

4.3.2 Process and Harmonize Data

Process and Harmonize Data has the task of receiving the individual readings from the Acquire Sensor Data process and deal with them so that they are in a format that can be deployed into the repository. It is important to notice that not all the sensors are processed the same way, as an example readings from LX sensors (used for counting pieces or objects) as seen in (Bannerengineering, 2017) come with the value of "1", in that case the elements must be count before the insertion. This process is depicted bellow:

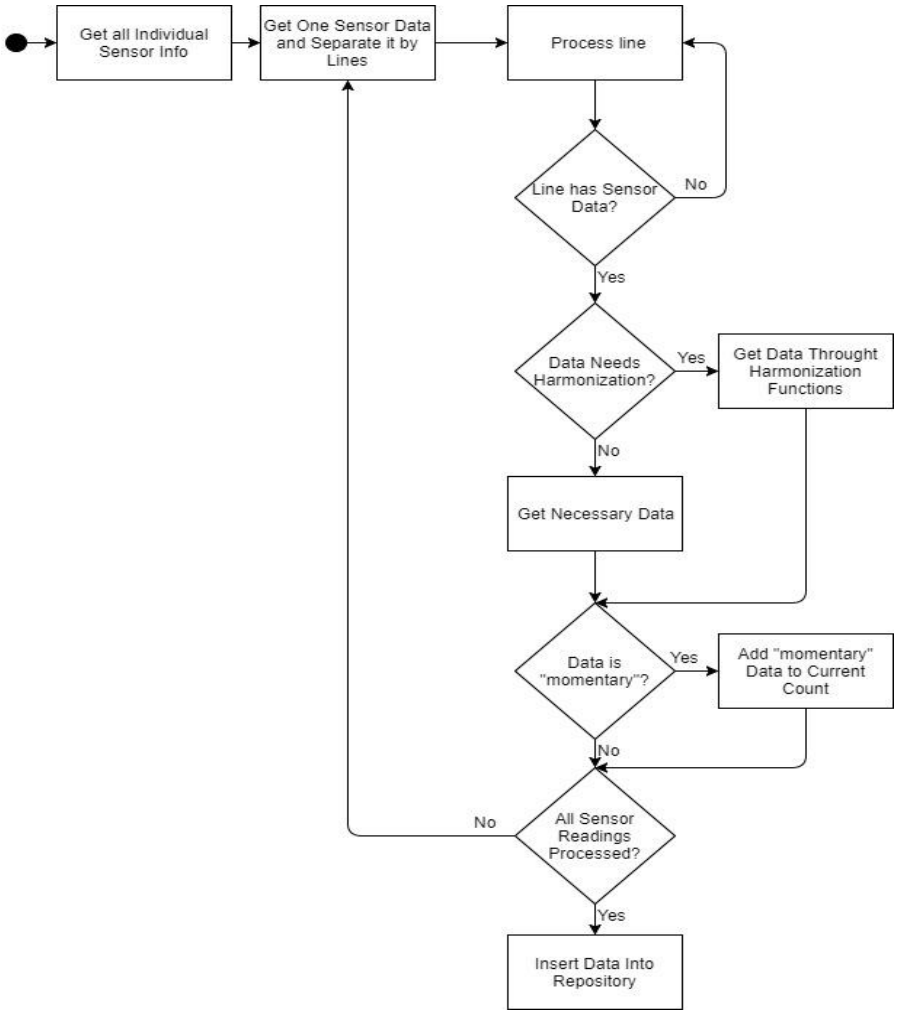


Figure 4-6: Process and harmonize data activity process

In the beginning of the process, it receives all the individual sensor information and processes it, one by one. As it was shown in chapter 3, it exists a lot of unnecessary information in the sensor response, and, for that reason, the response content is separated and parsed line by line. When important data is found, if it needs harmonization, passes through the necessary harmonization functions. If the data is already in the correct format, the values are stored.

Sensors that count elements have a field called "momentary" with the value "true", this allows to differentiate them from the other kind of sensors (such as temperature, measurement, etc.). When this value appears as "true" for a certain type of sensor, the

counting of that sensor is incremented. This assures that, at the end of the process, the system has an accurate number of elements. Sensors that don't count elements do not have values or readings stored. This process repeats until there are no more sensor readings and the stored information is deployed into the repository.

4.3.3 Evaluate Context

The context evaluation process looks for new sensor readings deployed in the repository, finds a rule / context that guides them and processes the context evaluation. This process has two different phases. In the first phase are the rule that the target and the corresponding context. In the second phase, the rules are evaluated, and the context is verified. Figure 4-7 illustrates the context information process of the sensor, for which there are targeting rules.

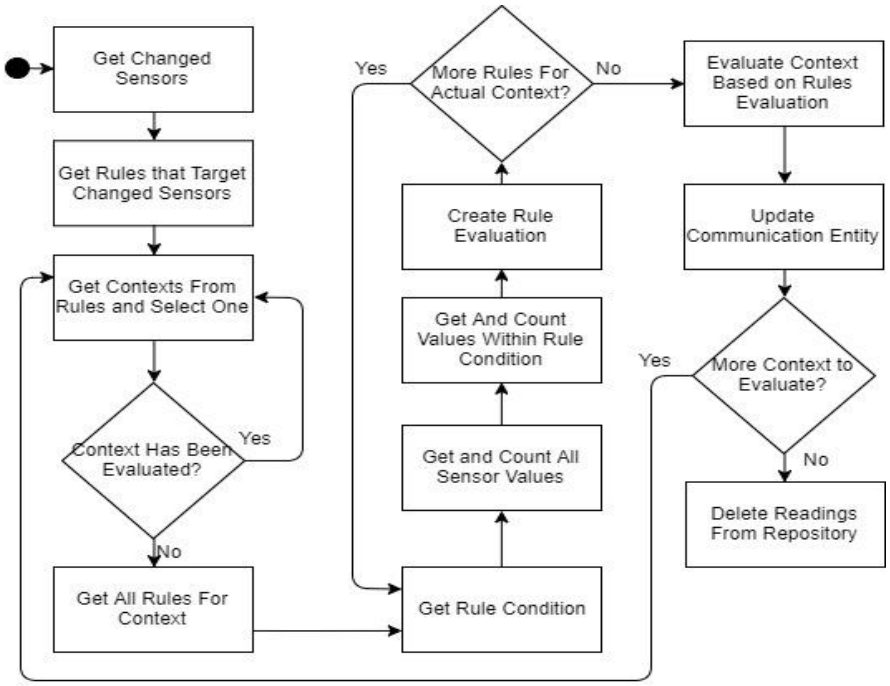


Figure 4-7: Evaluate context process

After the Changed Sensors are identified, the system can query ORION for the rules that target them. This is done through the basic functions introduced in 3.6 and used throughout the evaluation. Because the rules contain their corresponding context name within their attributes, system can access them. The various contexts identified are analyzed one by one until there is no more to evaluate. To evaluate them, it is necessary first to access each individual rule, if the number of registered rules is less than necessary, then their evaluation fails.

To evaluate the rules, it is first necessary to obtain the necessary condition of the rule's entity, on the basis that the matches that satisfy the rule are retrieved from ORION. Before querying the information, it is necessary to decode it in the RuleCondition field (the

reason behind this decoding will be explained upon presenting the process responsible for inserting rules). The number of readings that are within the rule condition is compared to the total number of sensor readings. The success rate of the rule is obtained by applying the expression

$$RuleSuccess[\%] = 100 * \frac{InsideRuleSensorValues}{TotalSensorValues}$$

The evaluation result (RuleSuccess) is saved and this process repeats until the moment when all rules within a context are evaluated. After all rules have been processed the context is evaluated and to do this every rule results are compared. if one of them is below one hundred percent the evaluation fails.

The result of the evaluation is sent to the Communication Entity so that vProductMon can report it to user. When there are no more results to be evaluated the used readings are deleted from the repository.

4.4 VProductMon

In the next subchapters are explained the processes within VproductMon. These processes include Display Evaluation Results, Insert Context, Insert Rules, Delete Element, Display Elements and CommunicatevFNegotiation. Display Evaluation informs the user about the context Evaluation, by Vfail. Insert Context and Insert Rules are responsible to deploy new elements into repository. Delete Elements remove entities from repository and Display Elements show what is within the repository. Finally, CommunicatevFNegotiation sends information from VProductMon to VfNegotiation.

4.4.1 Display Evaluated Results

The Display Evaluated Results is the bridge between the user and the information evaluated by Vfail. The function of this module is to display, in real-time the evaluated information. The interface of this process is composed by three buttons (Start,Stop and Clear) and a textbox where results are printed. The interface of this activity can be seen in figure 4-8.

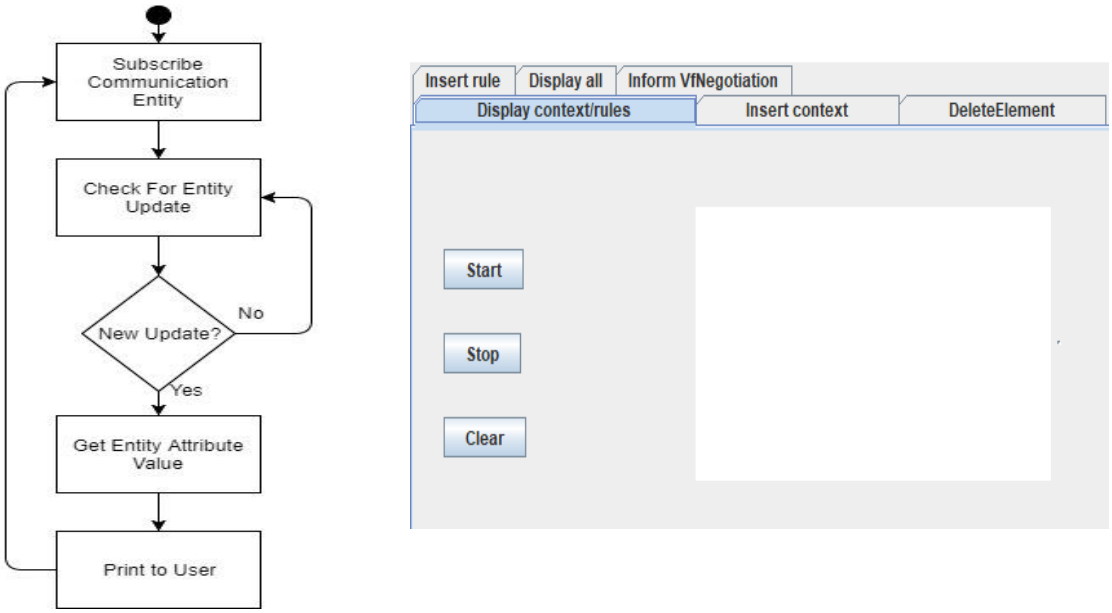


Figure 4-8: Display evaluated results activity process and interface

The evaluation begins when the Start button is pressed. The first time it happens, a parallel process begins, but if the Stop button is pressed, the process stops and can be resumed by pressing Start again. Orion allows the system to subscribe to an entity and check within time intervals whenever it changes. When the process is running, it is checked if the content value within CommunicateVproductMon has been updated. This process is repeated until the Entity is updated, at which point the value inside is retrieved and is displayed to the user. The process returns to the beginning, looking for new updates of new entities. The text box shown in Figure 4-7 is self-adaptive, meaning that all text displayed with it can be transmitted by the user, using scroll bars. The Clear button deletes the information displayed in the text area

4.4.2 Insert Context

Insert Context allows the user to define a context that can be verified by Vfail, Insert Rule creates a rule under a defined context. As it was explained before, because all context names are different it is possible for the farmer to create as many as he wants. To define a context, it is asked the user to insert the desired name and the number of necessary rules.

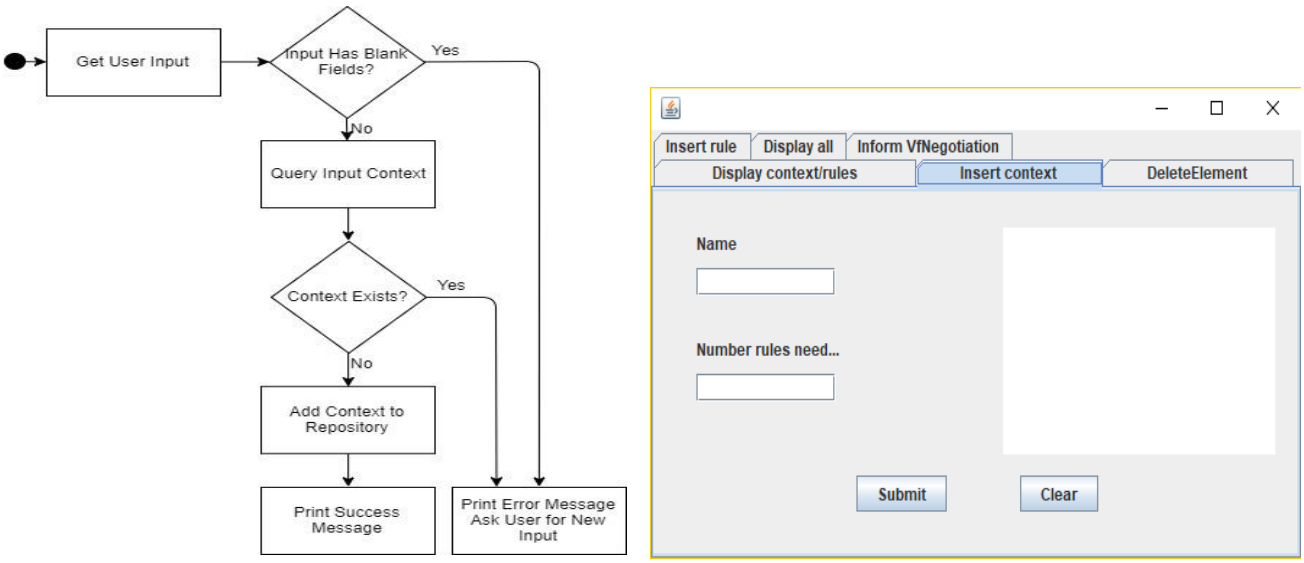


Figure 4-9: Insert context activity process and interface

After the user enters the requested data, it is submitted for a small check. This check verifies that all fields are filled, which is necessary to avoid other results from the ORION query, since it has already been referenced in 3.5.5. If the data is correct, the system checks the existing context in the repository, again this process is done by the basic functions that query information from the repository. If the context already exists, the user is warned and asked to enter new information, if this check did not exist, ORION would, by default, update an existing entity with the number of rules entered. When the context does not exist, it is inserted into the repository. The ContextRules field in the context entity is, by default, created without any value, it means that it needs to be updated when the rule is created. After creation, the information entered is displayed to the user whenever he wants to delete it. The Clear button does this.

4.4.3 Insert Rule

Insert Rule is responsible for creating a rule that fits a previously created context. A context can have as many rules as the needed verifications. To create a rule is demanded to insert its name, Condition, Sensor Target and corresponding Context.

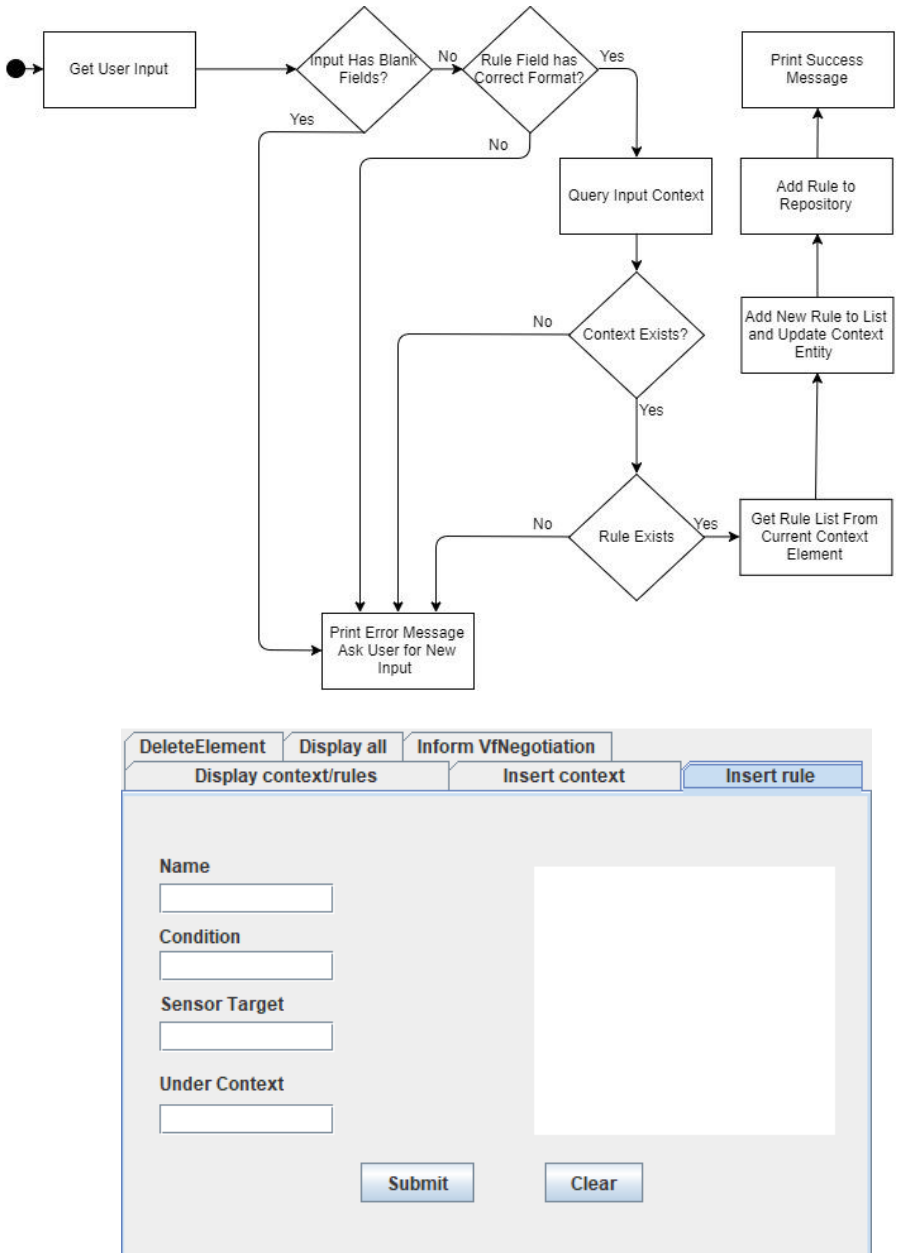


Figure 4-10: Insert rule activity process and interface

After submitting the requested information some verifications are made. It is checked if all fields were introduced and if the Condition was inserted properly. Condition field can have three different inputs which are:

1. > inserted value
2. < inserted value
3. Between value1 value2

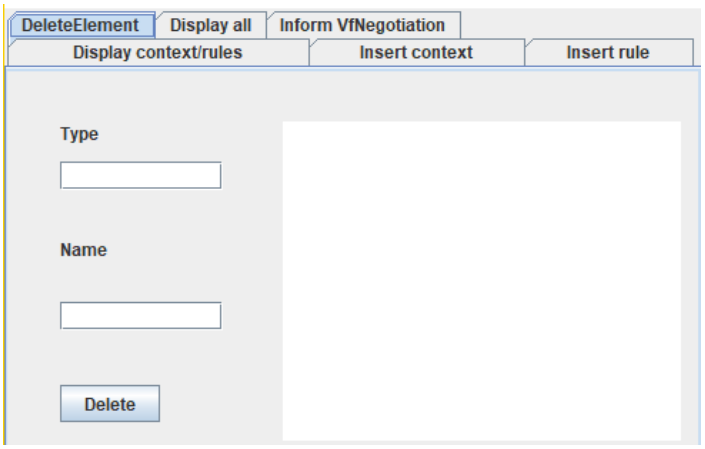
If the input is a “>” or “<”, the verification gets the inserted value and separates it on the space. Then it checks if the first argument is a valid one. It is also made sure that the second argument is a number. Some of the code in which the basic functions were built on has several protections. Some of those protections do not allow to create attributes with several symbols or to have > and < within an attribute. To overcome this problem, the

program modifies the inserted data and translates It into the words “Bigger” or “Smaller”. As an example, a condition input such as “> 20” will be stored as “Bigger 20”. This modification implies that when decoding the rule condition, Vfail will have to change the content to its original state, as introduced in 4.3.3. In the previous example the condition would turn again into “> 20”, a statement that ORION can easily use when querying values.

After verifying the inputs, it is checked whether the input context exists. If it does not exist, the user is warned and asked for a new entry, if it exists, it will be verified if the rule already exists inside the repository. If the rule already exists, the user will be notified and the rule list is not updated. To update the list of rules, system must proceed to three different steps. The first step is to get the current rule list for the target context, the second is to add the current rule to that list. The list of rules appears in the default format, "rule1 rule2 rule3 rule4 ..." and a rule name is added to the current list. The last step is to update the entity with the current list. After this process, the rule is added to the repository and the user is notified.

4.4.4 Delete Elements and Display elements

Delete Elements has as objective to delete a rule or context from repository. It can be used when user decides that an element is no longer necessary or to replace an existing one. To replace an existing element it is necessary to delete it first and insert after the new element. When deleting an element, the application asks for its type and name, based on that it starts the process of removal.



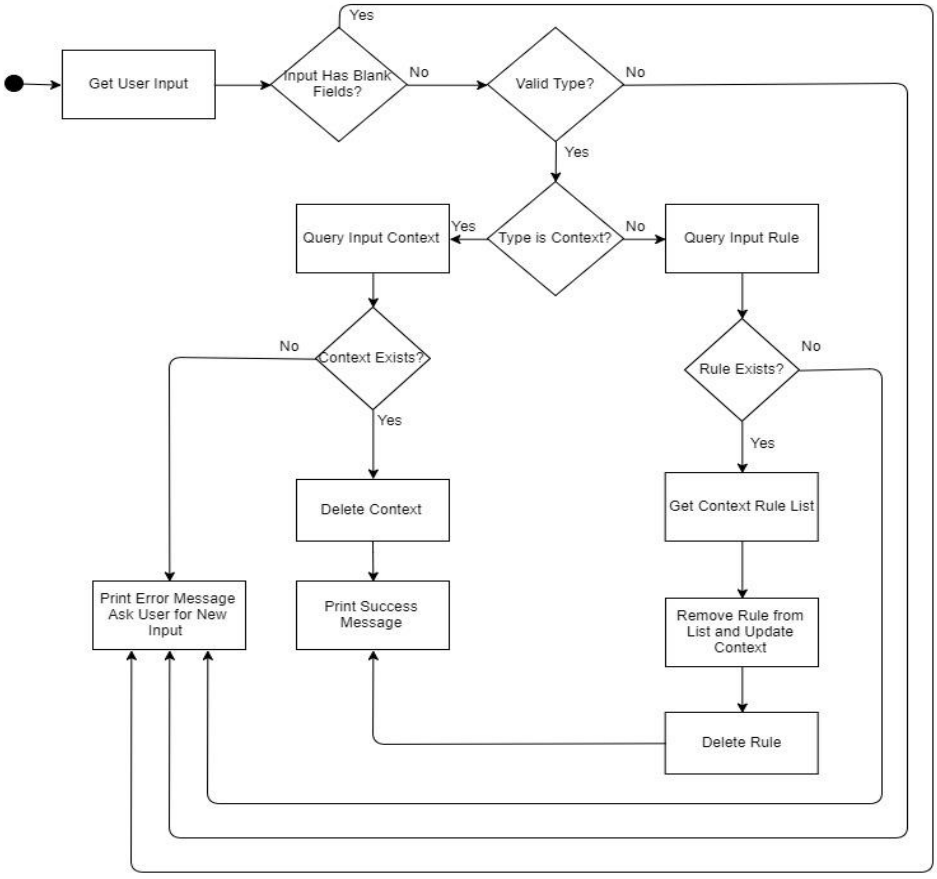


Figure 4-11: Delete element activity process and interface

As it happens with the other processes the first step is to verify if the information has blank spaces. After that is verified the system checks if the chosen option is a rule or a context. If the inserted option is different from “Context” or “Rule” the system stops the process and warns the user. If the information is correct, weather it is context or rule, the system proceeds in a different way.

When type is context, system checks if it exists in the repository, if it doesn't exist the user is warned, it is impossible to delete an entity that does not exist. If it exists is deleted and user receives a success message. If element is a rule system proceeds first to the existence verification and then proceeds to update the corresponding context entity, following a logic similar to the one used in 4-10. The first step is to get the actual rule information from the corresponding context entity, after this the list is updated. To update the list, it is necessary to find the rule name from within it and then remove the respective name and the following space from it. The new list is then updated with the context entity using the basic functions. The rule is then deleted from the repository and user is notified of the success. Display elements is used to notify the user about the information that is inside the repository. Because it is a simple process it is not necessary to show its activity diagram. After the user clicks a button, the process searches in the repository for all the context entities and displays them.

The interface of this process is depicted in figure 4-12.

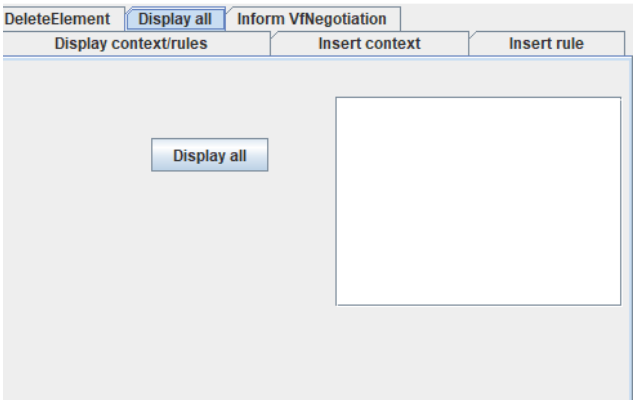


Figure 4-12: Display elements interface

4.4.5 Communicate vFNegotiation

Communicate vFNegotiation is responsible for the data exchange between vProductMon and the negotiation application. It allows a farmer to share the amount of its production maintaining a high-level of fruit quality. This process uses the information from the evaluation and associates it to an existent farmer. It is asked to the user the farmer's identification, and the breed of the evaluated fruit. It is necessary to ask for this information, because the farmer's id corresponds to the Farmer entity id and the breed corresponds to the attribute name.

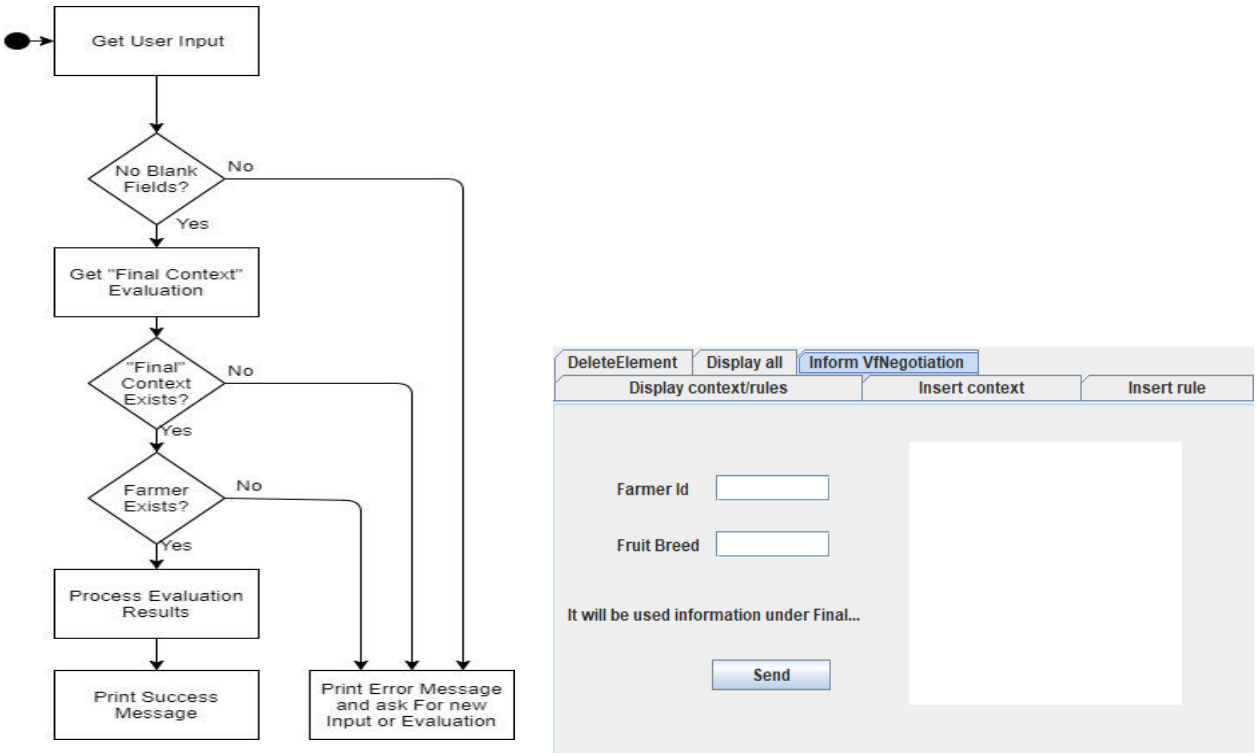


Figure 4-13: Communicate vFNegotiation activity process and interface

After verifying that the fields are not blank, the process searches within the repository for the chosen farmer. If this farmer does not exist, the user will be notified. The process obtains information about the evaluated fruit, this information is displayed within 4-13 figure text field. When defining a context type, exists a special format, defined by "Final_fruitname_fruitsize". This information is usually a calibration context and allows the system to calculate the exact number of fruits that are within a size. The reading is displayed with the total number of sensor readings evaluated and the percentage success. The information inside the text box is modulated and the percentage and the total value are retrieved and applied to each other, obtaining the total value of the fruits of a given voltage. If in the final context exists, the user is notified, if there is a farmer entity is updated.

5 Results and Validation

In this chapter, the results of the prototype developed in the previous chapter are demonstrated, taking in account all the work done in other chapters. It will be provided the demonstration of the functionalities of the system functionalities, illustrating how the user can insert rules and context. How the rules and context are deployed in the repositories and how the interface interacts with it, will also be demonstrated. It provides guidance on how to use the system based on the framework and, how to be aware of the outcome of any context assessment, since all necessary conditions are provided to the system. The communication demonstration shows how APPS interacts with other Vf-APPS to achieve the system introduced in the architecture section. The test evaluation sections provide a critical scientific evaluation of the results.

5.1 Testing methodology

Concerning testing methodologies, there exist several methodologies that allow to test software engineering. Some of them are abstract concepts, as for example black box testing or unit testing as seen in (White, 1987). Regarding the evaluation methodologies, those can be classified in Functional and Structural.

In structural testing all the existing code is analyzed, line by line, with the objective of getting all possible paths of execution covered. This implies that the selection of the test cases is based on the software entity implementation. Structural testing is also known as “white box” and empathizes the internal structure of an APP

Functional testing evaluates the target APP or program by observing it externally, which means that there are not considered its internal details or implementation. In this kind of testing the selection of the test cases is related to the requirements or software design specifications. This testing is also known as “black box” and emphasizes the external behavior of the software.

5.1.1 Test Control Notation

The Tree and Tabular Combined Notation is a notation standardised by the ISO/IEC 9646-1 for the specification of tests for communicating systems and has been developed within the framework of standardized conformance testing(Académicas & Académicas, 2015).

The notation is based on the black box testing model and the tests are defined using tables that are divided in general description, constrains, behavior and verdict.

Test behavior is a set of events that represent the test itself. These events appear in a tree format and contain branches of actions, which are based on the system's output after the executed event. There can exist two different types of events, actions or questions and they both must obey to an indentation level. When the event is an action, it is preceded by an exclamation mark and it refers to actions performed by the STU (System Under Test). If the type is a question it is proceeded by a question mark and refers to evaluations on the SUT output after actions are completed. There can exist multiple question on the same indentation level to cover all the possible outputs. After a TTCN test table is completed, its verdict can be either "Success", "Failure" or "Inconclusive" and is based on the sequence of evaluated events. The following example shows a TTCN table and its explanation, for a phone call establishment evaluation, from (Ferreira, 2012).

Table 5-1: TTCN example table

Test Case		
Test Case: Basic Connection **Group:** **Purpose:** Check if a phone call can be established **Comments:**		
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

Sequence explanation:

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 must dial the other phone's number.
4. Otherwise, if the dialling tone is absent, the verdict is a "Failure" of the possibility of establishing a phone call;
5. If there is a calling tone after dialling the number, the user may test if the line is in fact connected;
6. If the line is connected, the user may hung up the headphone and the verdict is set as "Success" on establishing a phone call, otherwise the verdict is a "Failure" of the possibility of establishing a phone call;
7. If the dialling tone is not heard, but a busy tone instead, then the user may hung up the headphone and the verdict is set as "Inconclusive" on establishing a phone call;
8. If none of the tones corresponds to calling or busy, then the verdict is set as "Failure" on establishing a phone call.

5.2 Testing Implementation

To show the full potential of the framework will be tested two different situations. The first will refer to the framework itself, by creating and evaluating rules. Regarding context insertion, the way how it is done is illustrated in figure 5-1 (top). A rule insertion can be seen in figure 5-2 (left) and a rule evaluation is depicted in figure 5-6. In the "framework implementation demonstration" section will be shown how to monitor the current rules /context inside the framework (figure 5-5).

The second situation will show how the framework communicates with vFNegotiation APP. In both cases it will be first showed the framework running from a farmer's point of view and then performed and evaluated based on the model from 5.1.

5.2.1 Framework implementation demonstration

To test the framework, will be considered that a farmer has an apple plantation and needs to verify three different situations, each one in a different checkpoint. Each checkpoint provides the necessary sensor readings to check if the desired situation is verified. The first thing that a farmer wants to check it is the expected fruit amount. To do this will be inserted context/rules that inform about the existent fruit amount and will infer if it is what the farmer was expecting or not.

The second checkpoint refers to the fruit calibration. Will be provided information to the system that allow it to evaluate if the analyzed sensor results are inside the expected

spectrum. This checkpoint occurs at the end of a calibration machine (regardless the model and fabrication date) and allows to infer about the processed fruit.

Finally, the third checkpoint is used to verify if the provided transport boxes are enough to the amount of collected fruit. It is important to that although only three checkpoints are currently being used (each one corresponding to a different context) the system supports much more, since the necessary information is provided to it. Throughout the demonstration are illustrated the inputs/system responses for successful situations and also some restrictions to avoid errors.

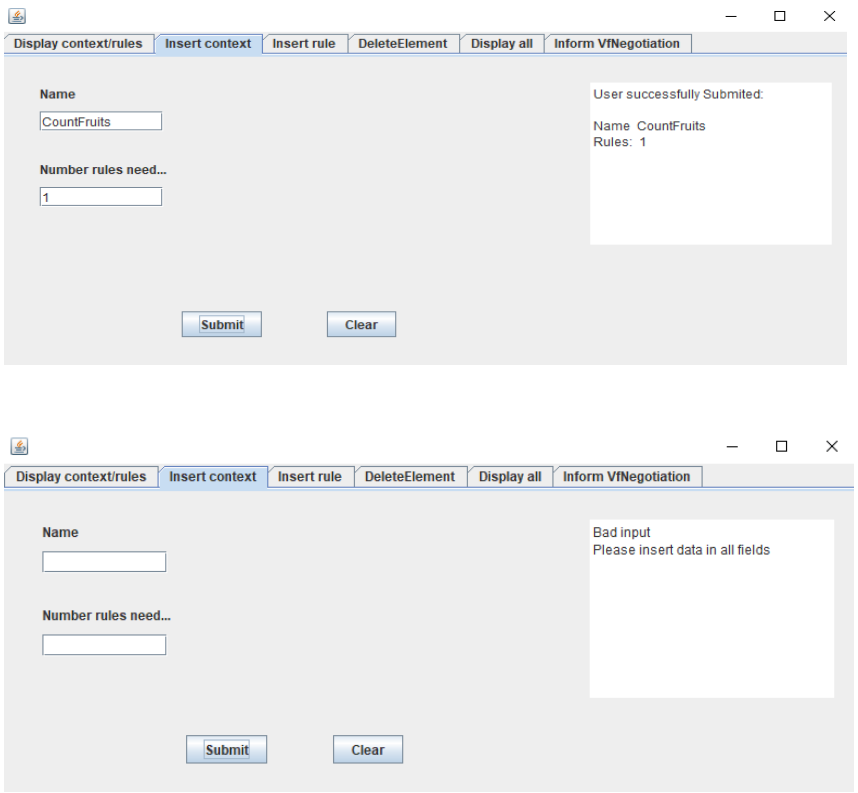


Figure 5-1: Context success submission(top) and blank space warning (bot)

To verify the if the quantity of fruits is the expected, will be necessary a context with a single rule. In Figure 5-1, on top, is shown how the vProductMon APP deals with the inserted information. After inserting the required input the application computes the process specified in chapter 4. The result is the one shown in 5-1 top, when the input is correct or in 5.1 bot if it is wrong. It is important to notice that, verifications like if the context already exist in de database, are not made during the pre-processing.

As it is possible to see from the images, this context requires one rule, that is shown in the image 5-2. Rules creation are made according to the logic explained in 4.4.3. Considering the rule that verifies the number of fruits, its formalization would be one of the following cases. If expected more than one thousand fruits (fruits>1000), it would be inserted in the application as “> 1000” (as it happens in the case from figure 5-2). If less than that value (fruits <1000), user would insert “< 1000”. Finally, if it is expected around five hundred or one thousand fruits (500<fruits<1000) user would type “Between 500 1000”.

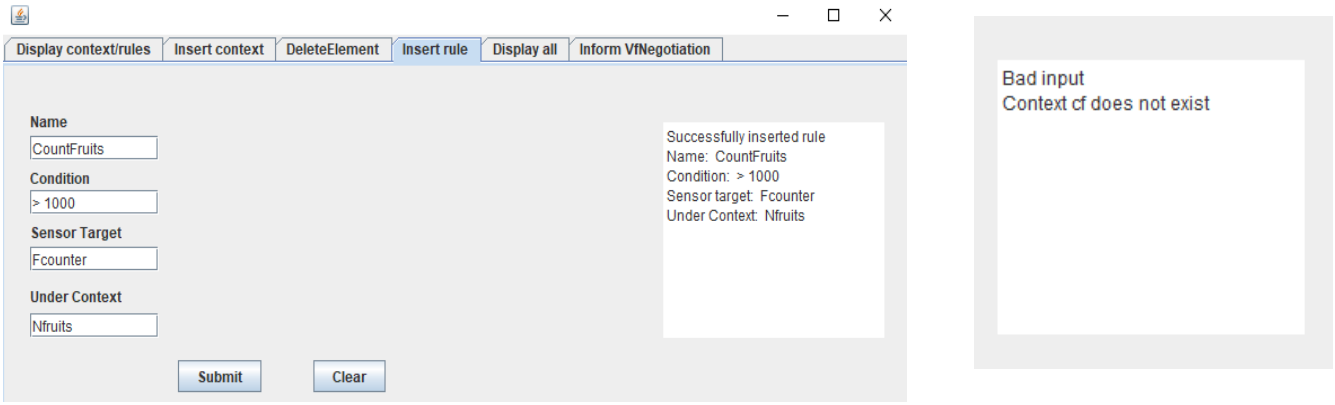


Figure 5-2: Successful inserted rule(left) and rule that fails for inexistent context(right)

Figure 5-2 (left) shows the result of a successful entered rule. “CountFruits” is the rule name that will be stored within the system. Condition “> 1000” refers to the value of the total account from the sensors reading. Although it is not depicted, besides the blank field verification that happens in 5-1 bot, it is also verified if this field is correct. As it was explained in chapter 4, to be in the correct format must exist two different arguments (in this case “>” and a number) and if the first element is a “>” or a “<”. If the first element were a “Between”, the verification would be different. “Fcounter” is the name of the sensor that this rule is targeting and besides the blank field, no more verifications are made to this field. “Nfruits” is the context from image 5-1 top. Image 5-2 (right) also shows what happens when the “Under Context” field is filled with a context that does not exist. In this case was introduced a context called “cf” and the system informs the user that the entered information it is wrong because the desired context name is invalid.

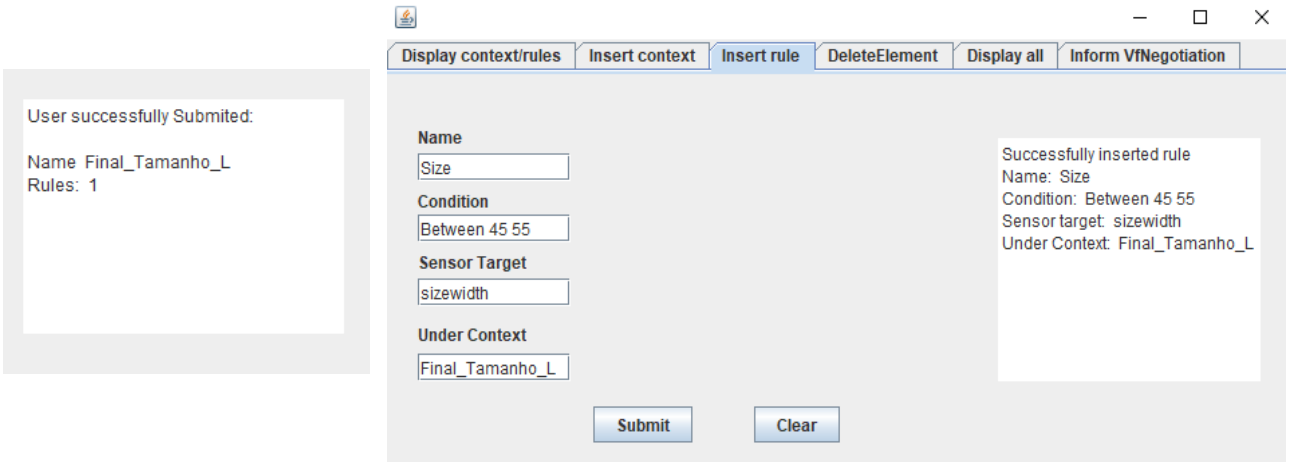


Figure 5-3: Successful context submission context (left) and rule (right)

To evaluate the fruit calibration of an apple will be necessary a context with one associated rule. The reason why is not necessary two rules for this case is because apples are usually round and, for that reason width will be slightly the same value as height. As it is possible to verify from Figure 5-3 the name of this context is “Final_Tamanho_L” and it has one associated rule. As it was explained in 4.4.5 the ” Final_name_size” is a special kind of context. The evaluation of this kind of context can be used to exchange information with

vFNegotiation. This type is only used at the end of the calibration process because it allows to infer the real number of available fruits. Figure 5-3 (right),shows the rule for this context, its name is “Size” and the condition is “Between 45 55”. In this type of condition, the system verifies if there exist two different elements after the “Between” and if these elements are numbers. The target sensor is “sizewidth” and the associated context is the one depicted in 5-4 (left).

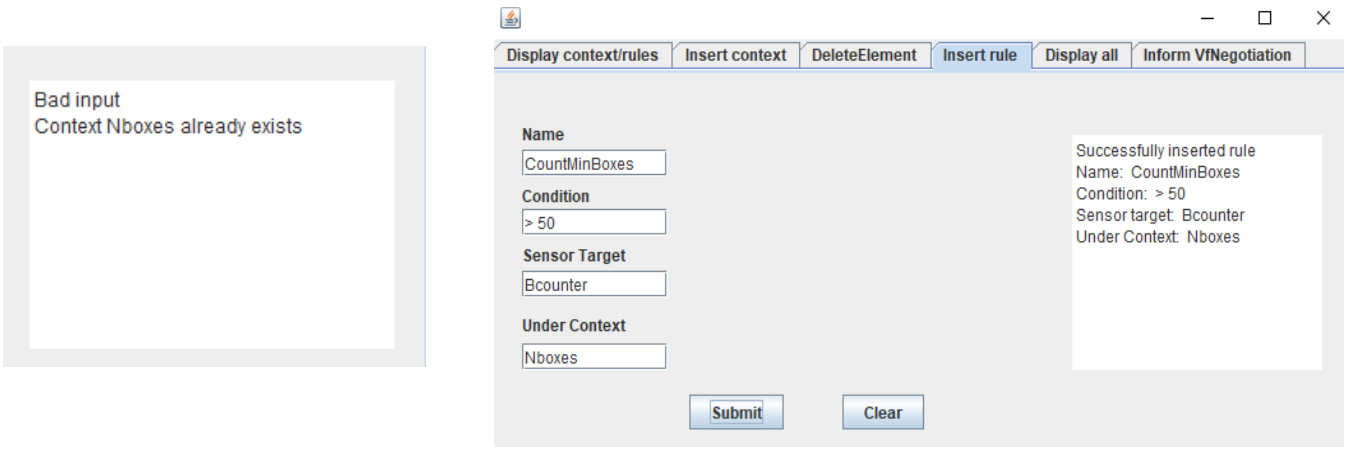


Figure 5-4: Response on an input of an existing context(left) and submission of one of that context rules(right)

The third verification evaluates weather the existing number of boxes is necessary for the expected amount of fruits. Imagine that for the expected number of uncalibrated fruits (expecting around 1000) are necessary fifty different boxes, that is the value that must be considered as condition. Figure 5-4 (right), shows the output for a good input and informs about the information stored in the repository. In this case the context name is “Nboxes” and it needs one rule. On left of image 5-4 is showed the output from submitting a new context right after submitting the first one (simplifying it means pressing two times in a row the submit button when registering a context) . In this case the system processes that the context “Nboxes” already exists and prevents the user from modifying it.

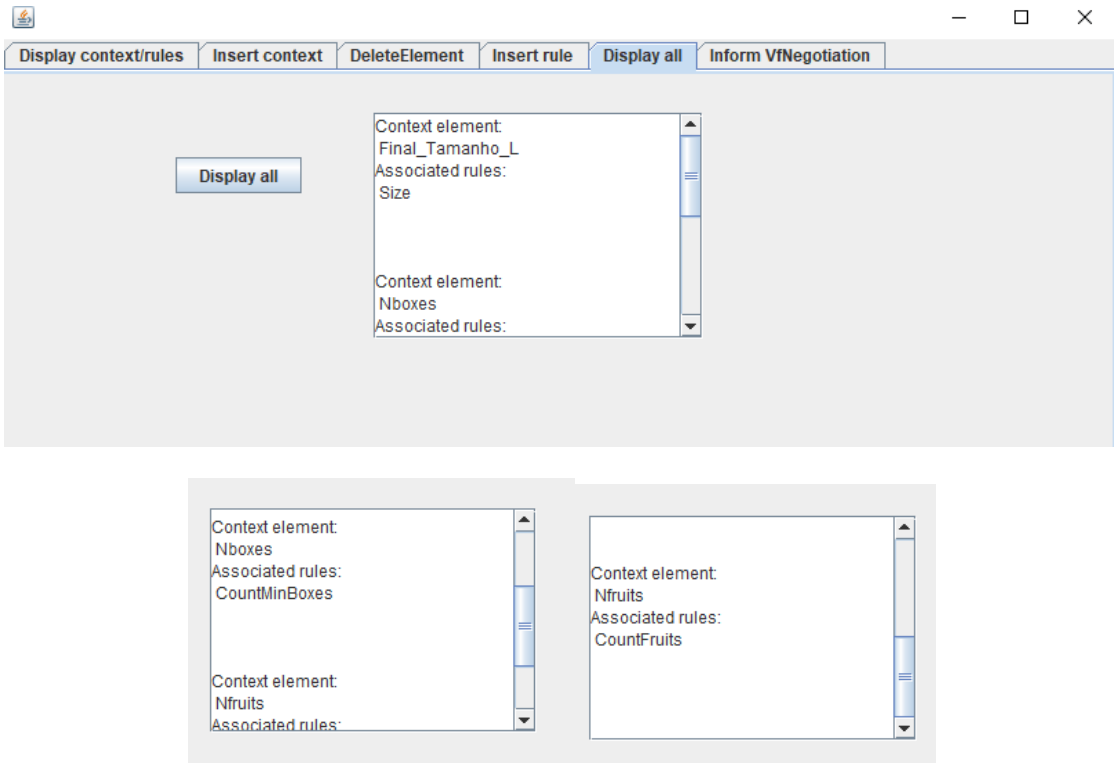


Figure 5-5: Context and rules registered in the system

All the inserted rules can be accessed by the user through the vProductMon tab "Display all". Figure 5-5 allows to understand which rules correspond to each contexts. As it was expected Size rule is associated to "Final_Tamanho_L", "CountMinBoxes" to "Nboxes" and "CountFruits" to "Nfruits". When a rule or a context are deleted, by pressing again the "Display all" button, the updated results are displayed. The same happens when new rules are inserted under an existing context.

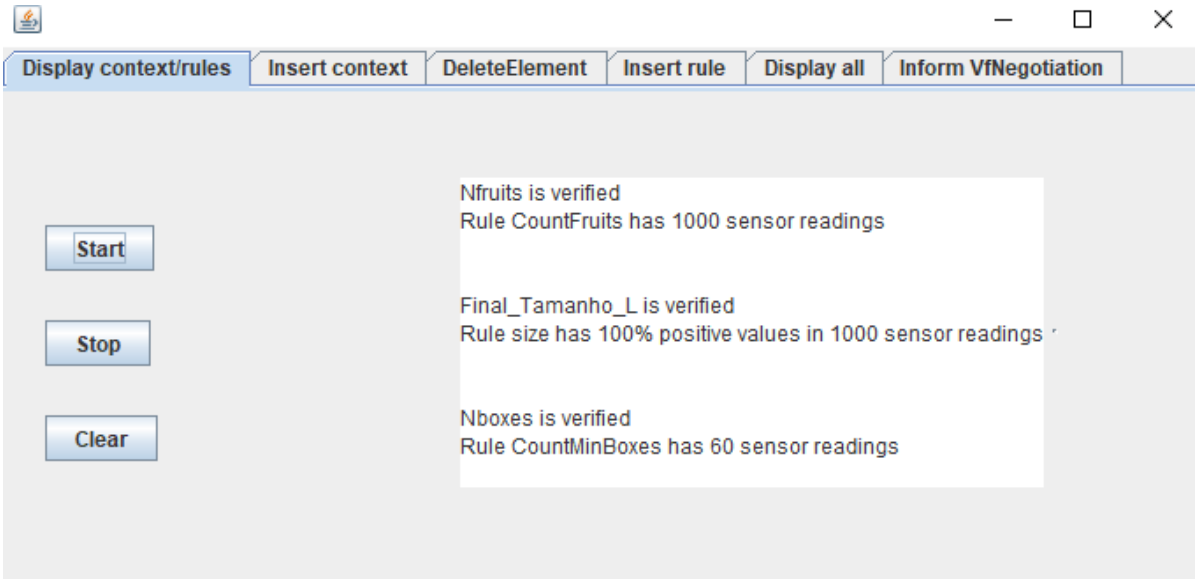


Figure 5-6: Evaluation results

To see evaluation results can be used the “Display context/rules” tab within vProductMon. From the moment when Start is pressed to the moment when user clicks on Stop the program looks for evaluations and displays them. The interface of Vfail is very simple, being composed only of a Start Button. When that button is pressed system can start analyzing whatever sensor information that it receives. The information analyzed by Vfail is received and displayed in real time to the user and that is shown in image 5-6. If the “stop” button is pressed, even when Vfail exchanges evaluation information, that information will not be displayed. Each evaluation is separated three paragraphs from the previous one, when information fills the textbox it will display the evaluation within a scrollbar, as it happens in figure 5-5. As it is possible to observe the evaluation is displayed after the sensor from each control point is processed. If the system receives information that is not targeted by any context/rule nothing happens and the user is not notified. It is also possible to understand that the evaluations have different outputs. The first difference is the message that tells if the context was or not verified, displaying “context is verified” or “context fails”. The second difference occurs with the rules outputs since the rules CountMinBoxes and CountFruits do not display the success rate. This happens due to the presence of “Count” within the name, that is processed in a different way by Vfail as explained in chapter 4.

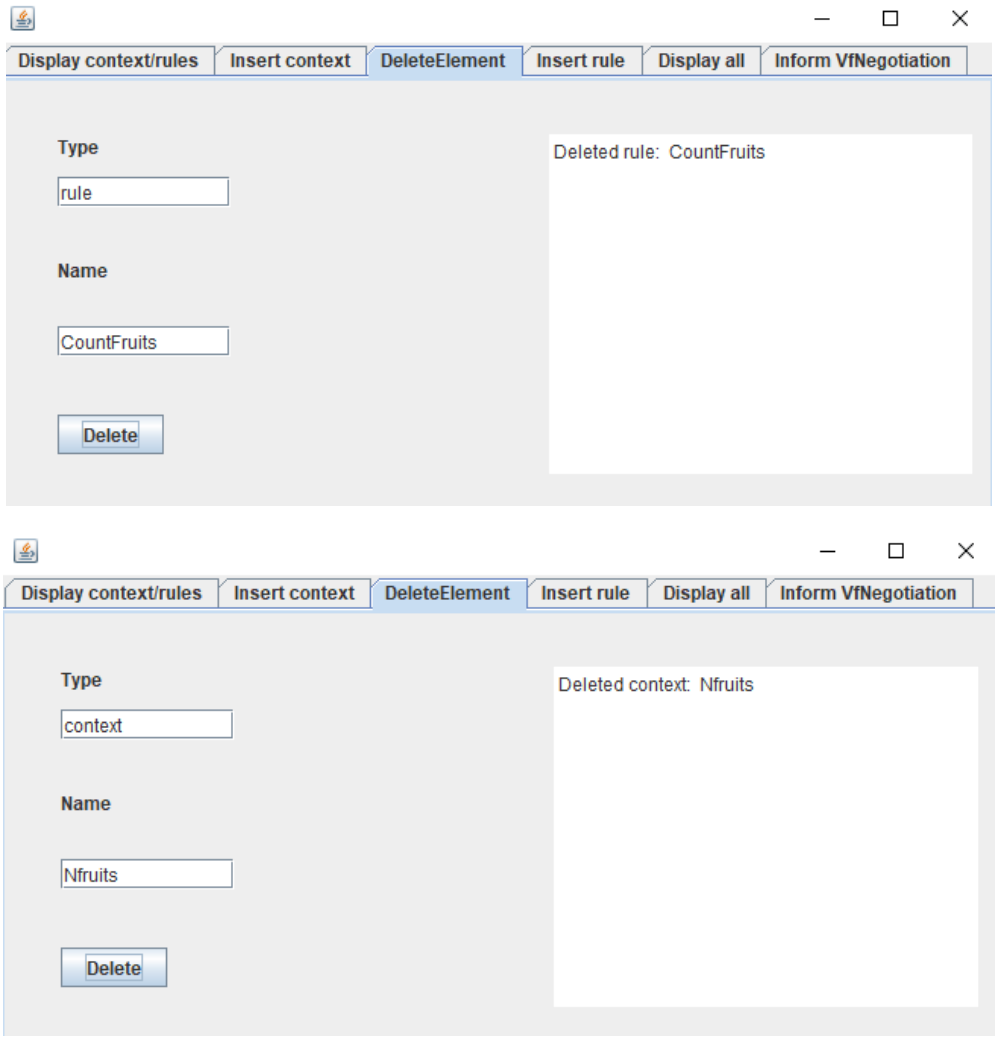


Figure 5-7: Delete a rule and a context

The rules can be deleted from the system, using the “DeleteElement” tab. The user chooses the element he wants to remove and specifies its type, if the input is accepted by the systems verifications the element is deleted. In image 5-7 it is showed the deleting of “CountFruits” rule. As it is possible to observe when the rule is deleted it ceases to be under its context available rules, meaning that it was successfully removed from the repository. Image 5-7 shows the removal of “Nfruits” from the system and by using the “Display all” tab it is possible to observe that the context no longer exists in the system.

In the previous examples it was demonstrated how the Systems creates rules/context to evaluate several sets of sensor data and display them to the user. The next example demonstrates the systems adaptability. Will be considered that a farmer wants to use a “Final_Tamanho_S” rule to be used with pears. Unlike apples, pears have to be measured in two different ways , weight and high. To do this is necessary to delete the existing rule and context and repeat the steps demonstrated in Figure 5-3 . At this point the context has already a rule associated to it that is called “Size” and that refers to width. After being added a new rule called “measurelenght” referring to a “sizelenght” rule, the framework receives information from “sizelenght” and “sizeweight” sensors. The result can be seen in figure 5-8, it is important to notice that now the context fails, due to the second rule evaluation not having a one hundred percent rate.

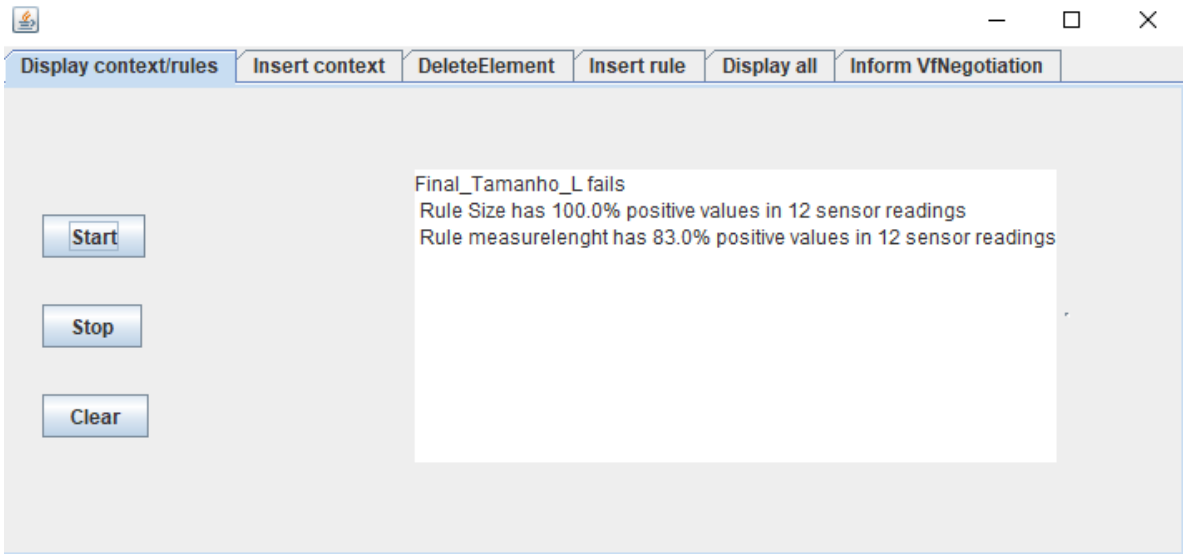


Figure 5-8: Evaluation of a context with two rules

5.2.2 Framework Test evaluation

To analyze the frameworks abilities, regarding the context and rules insertion, were deleted the ones that were already in the repository (from the previous example) and repeated the steps shoed in 5.2.1.

The table 5-1 represents the test case, to verify if it is possible to insert rules within the system. The demonstration in 5.2.1 corresponds to Figures 5-3, 5-4 and 5-9. From the table is

possible to understand that since all rules are correctly inserted by the user. They can be successfully inserted (registered) in the repository.

Although the table only covers the act of registering a rule the results to insert a context are slightly the same, as it is possible to verify from images 5-1 (top) and 5-3 (left) . It is then possible to conclude that since the input is correct (regarding the format and existence within the repository) it can be successfully inserted into the repository.

The test for deleting a rule or context is also similar to this one, the difference is that after getting the user input, it is evaluated for being fit to deletion, if it passes all the verifications is removed from the repository. This situation can be seen in 5-7.

Table 5-2: TTCN insert rule test

Test Case		
Test Case: Insert rule **Group:** **Purpose:** Check if a rule can be successfully inserted **Comments:**		
Behaviour	**Constraints**	**Verdict**
! Get user input ? User input is fit for registration ! Register rule OTHERWISE		Success Failure

The table 5-3 represents the testing of sensor data evaluation. It is possible to verify that since the sensor data has a context/rules that evaluate him the evaluation is performed. However, even if the data is evaluated, if the VProductMon is not listening to Vfail communication variable it cannot know what the evaluation is, as it was expected.

Table 5-3: TTCN evaluate sensor data test

Test Case		
Test Case: Evaluate Sensor Data **Group:** **Purpose:** Check if Sensor can be Evaluated based on Context **Comments:**		
Behaviour	**Constraints**	**Verdict**
! Get sensor data ? Exist context and rules that target sensor data? ! Evaluate Sensor Data ? User is listening to evaluations? ! Inform evaluation OTHERWISE OTHERWISE		Success Success Inconclusive Failure

5.2.3 Framework communication demonstration

In this subchapter it is shown how the presented framework interact with other vf-APPS. As it was already introduced in 3.1.1, the VfNegotiation is responsible for providing a virtual trading environment where farmers can sell their fruits and buyers can find the product that fits him better. This application has two different modes, the farmer mode and the client mode. The objective of the client mode is to filtrate the several existing product offers according to the desired need and it can be used by large or small trade companies that desire natural or high quality products. Farmer mode allows a producer to make his production available to all the buyers that use the platform. In this mode a client can register itself and in his “account” make the necessary steps to sell his product. The entry of the fruit sell is made manually, although the farmer has access to the evaluation displayed in VproductMon

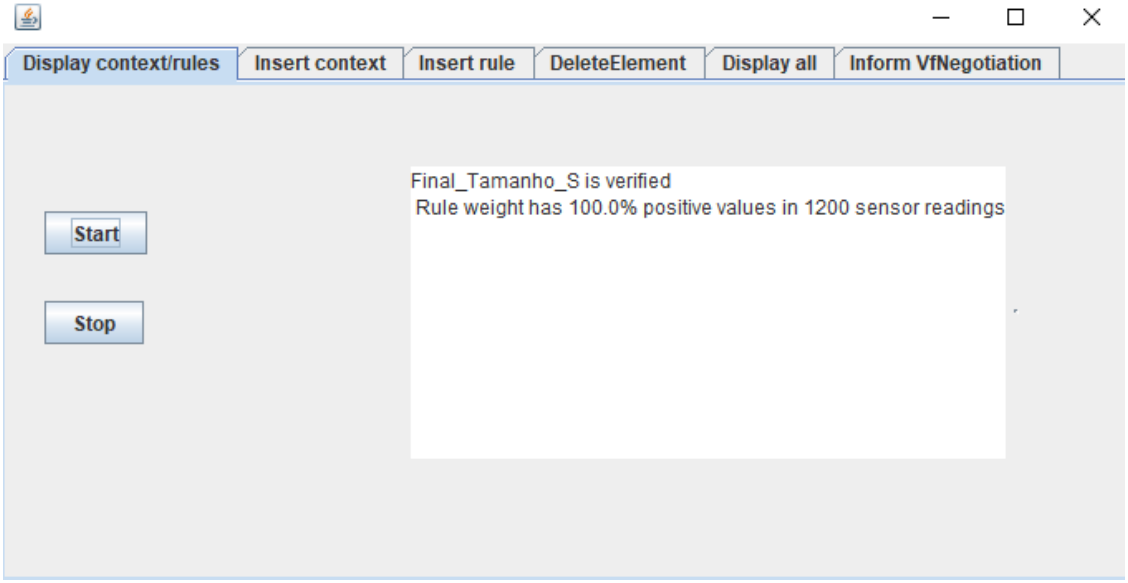


Figure 5-9: Evaluation of a calibration context

To demonstrate the interaction between the VProductMon and vFNegotiation a context, that evaluated the size of an apple strain, was first created. This context is called "Final_Tamanho_S" and because the fruit that is evaluating is an apple it only needs one rule. One thousand and two hundred sensor readings targeted by this contexts rule were evaluated and the result is shown in figure 5-9. From the moment when exists an evaluation, information can be exchanged. If no context were evaluated the system would display the error message from figure 5-10.

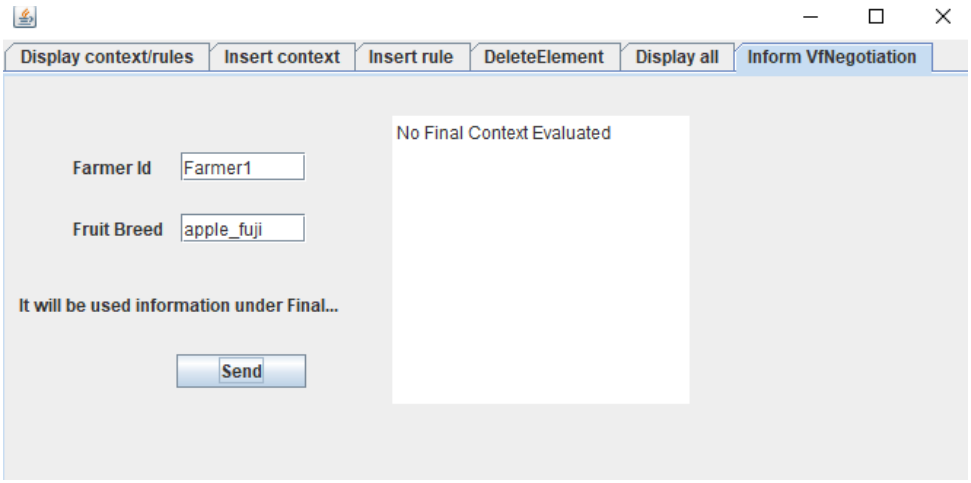


Figure 5-10: Error message resulting from non-existing evaluation

The Farmer which receives the information is created in vFNegotiation and to check if that the chosen farmer exists, a verification is made. When system processes that the chosen Farmer does not exist the error message from figure 5-11 is displayed. Upon choosing the Farmer it is also indicated the fruit breed that is being evaluated.

Figure 5-11: Error message resulting from a farmer not being registered in the system

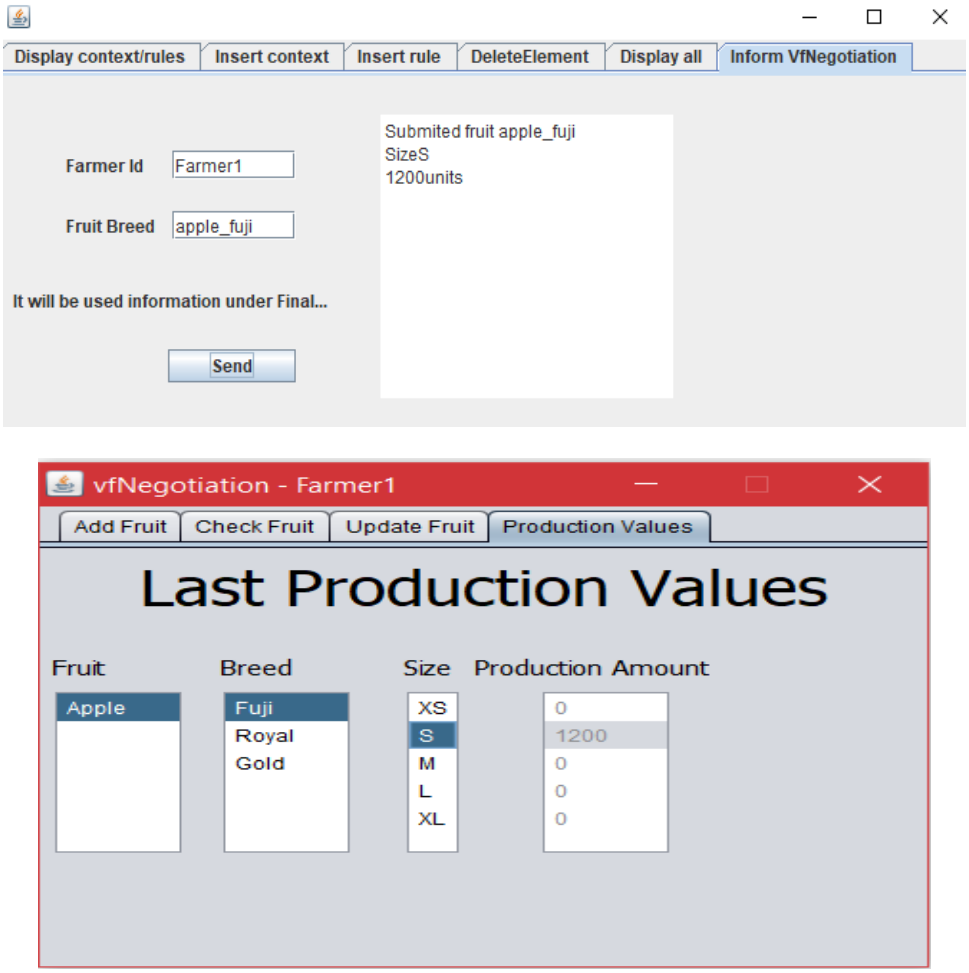


Figure 5-12: Successful communication from a verified context and view from vOrder side

When all the verifications are done the system calculates the effective number of fruits as explained in chapter 4. Figure 5-12 illustrates the system response for a successful evaluation. It is also possible to verify that vOrder application can access the submitted values. To test the response when the context fails were added ten sensors that failed the rule from figure 5-3 and the sensors were again submitted to the framework, the result of the

new results can be seen in figure 5-13i, as it was expected the system calculated the amount of apples within the success rate.

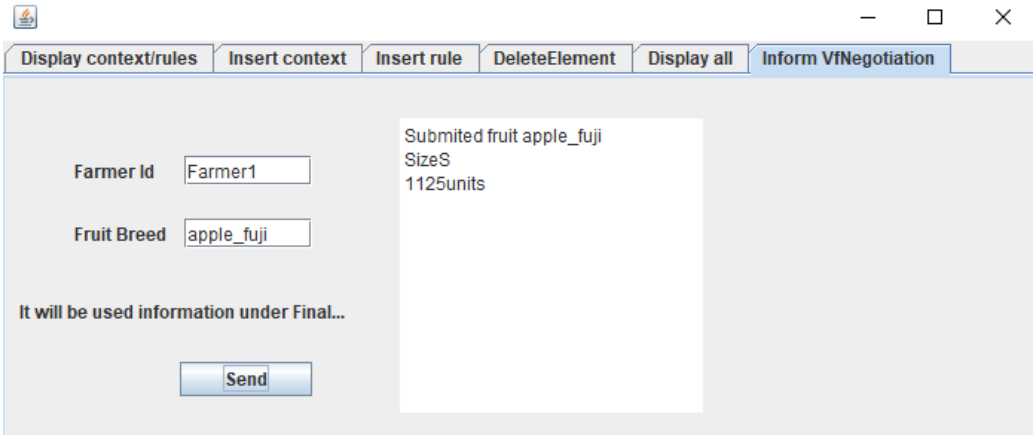


Figure 5-13: Successful communication from a failed context

5.2.4 Framework communication evaluation

To test if the communication process could be performed, were performed the actions from figures 5-11, 5-12and 5-13.

Table 5-4 allows to verify the success of communication between Vproducmon and Vfnegotiation APP. Since an evaluation is present and the user data is correct the communication will always be done. Otherwise the submission will fail and display an advise like the one from figure 5-11 will be showed.

Table 5-4: Communicate VProductMon test

Test Case		
Test Case: Communicate VProductMon **Group:** **Purpose:** Check if the VproductMon app can communicate with vFNegotiation **Comments:**		
Behaviour	**Constraints**	**Verdict**
! Get evaluation information ? Evaluation exists !Get user input (farmer id and apple strain) ?Input is fit for communication !Update communication entity OTHERWISE OTHERWISE		Success Failure Failure

5.3 Thesis validation

From the performed tests, it is possible to conclude that the framework is fully functional and, completely capable of performing sensor data evaluation based on previously defined context and rules. As it can be noticed in 5.2.1, the definition of new rules and context is fully protected, so that the farmer cannot undo a previous defined rule by mistake. The simulated sensors information can be accurately evaluated, and that evaluation displayed to the user in real time.

The hypothesis states that, if a system can be compliant with a mental modelling approach, then its processes of contexts formalization and consequent decisions/actions are enhanced. The rules and context formalization are made available to the system, in a way that it can understand them (as seen in figure 5-2 and corresponding explanation), which is in accordance with the hypothesis.

In (Ferro-beca & Jardim-goncalves, 2013) are described the Intelligence and Interoperability Levels, based on the capability of a system to interpret data. The lowest system intelligence level can only interpret data alone. As the intelligence level increases, systems can assure more complex processing, such as processing data on a semantic level. As

seen in figure 5-14, the third level can process information according to a defined context, which means that a system which performs it, is able to represent knowledge. The developed prototype, built upon the three-level mental model, can access sensor information and reason it, based on defined context rules. Therefore, the system has knowledge of certain situations that may occur, being in the knowledge level.

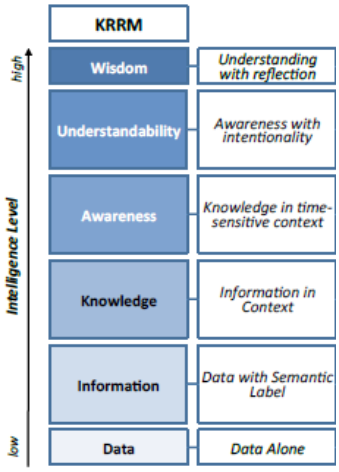


Figure 5-14: System intelligence levels, adapted from (Ferro-beca & Jardim-goncalves, 2013)

5.4 Scientific and Industrial validation

In this work was proposed to present a data processing pipeline in FoF perspective. Throughout the several chapters, were presented methods that allow to process sensor information, recurring to context rules, an approach that can be used to reason the information gathered by factory's sensors. The research results of this dissertation have been achieved in collaboration with GRIS UNINOVA. GRIS(Group of Research in Interoperability of Systems) is part of the center of technology and systems UNINOVA research institute, that acts in close partnership with the Faculdade de Ciências e Tecnologias (FCT) of Universidade Nova de Lisboa (UNL). Two scientific publications were made in order to validate this dissertations work, one was published in ICE/IEEE 2007 conference and the other submitted in the 9th international I-ESA conference 2018.

- Diogo Ferreira, Pedro Corista, João Gião, Sudeep Ghimire, J. S. and R. J.-G. (2017). "*Towards Smart Agriculture using FIWARE Enablers*". Accepted in: ICE/IEEE Conference 2017.

- Pedro Corista, Diogo Ferreira, João Gião, and R. J.-G. (2017). "An IoT Agriculture Platform using FIWARE". Submitted in: the 9th international I-ESA conference 2018.

Regarding industrial validation, this thesis was made in cooperation with the Vf-OS project and there were developed prototypes for two of the generic applications presented in

(Lead, 2017). Besides industrial application of the prototype, the architecture that was created in this work, can be used in the future to develop generic applications that have different applicational scenarios. Finally, the provided architecture and integration guidelines can be used by researchers that aim to develop a context driven, interoperable framework to be applied in specific scientific fields.

6 Conclusions and Future Work

6.1 Conclusions

As the concepts related to ICT and IoT are earning each time more importance, increases the use of these concepts to develop technologies that can be used in the paradigm of Factories of the Future. Modern factories rely on the use of concept and rules that allow to infer about several sets of sensor data. In this work was proposed a framework that could solve problems using these concepts. Besides being able to use concepts/rules to evaluate a situation or thematic, the framework should also be prepared to process several sets of data from different sensor models. This implied that even if the framework isn't prepared for all possible sensor data it should be able to be adapted for them, applying small modifications or using harmonization functions. This framework is integrated within the vf-OS that looks to produce several APPS which can be used within a platform that aims to be the reference concerning Factories of the Future.

The first step to accomplish this task, was to search and get informed about the main concepts and solutions that exist within the thematic. For this reason, it was explained the main thematic that surrounded Factories of the Future, highlighting its relationship with IoT and interoperability concepts. Since the Vf-OS modules were designed to be part of an operating system, became necessary to explain the logic behind an open OS, and address core components, such as the Kernel. Because this framework was aimed to be built using Fiware enablers, a section to provide an explanation of the Fiware project and, the main chapters that compose it, was created. To address the use of context and rules to evaluate data, where explained the main context/rules application theories in which modern industry systems are based on. Finally, the main rules behind data harmonization, that are used to exchange data from different systems, were explained.

To test the framework's concept, an agriculture scenario was created, and explained its relation to generic vf-OS APPS. The framework's architecture was introduced, considering the its tasks and the most suitable mental model. The relation between context rules and sensor readings was properly introduced and the several frameworks functions were defined. The specific technologies, chosen to build the framework and its harmonization needs, were clarified. The sensor mismatches and guidelines to overcome them, concerning the harmonization of sensor data exchange were also presented. The necessary APPS and technologies necessary, to implement a framework coherent with the chosen mental model, were clarified ending in the system architecture. It was then explained how the framework was implemented, concerning the two different vf-APPS. During the explanation all the needed steps, that were necessary to create the functionalities, were referenced, from the basic functions that allow to interact with ORION to the high-level processes.

After developing the framework was necessary to present the develop prototype. In a first phase was presented the implementation results and validated the proposed hypothesis. In a second phase was given an applicational presentation of the application, explaining how it can be used by farmers to control their fruit production. Was also explained how to exchange information about the context evaluation with vFNegotiation, a Vf APP that gives the farmer the opportunity of selling its production. In the end of the developed work, was obtained a versatile framework that can be used to control the several steps related to Fruit production and that can be easily adapted to different sensor messages.

6.2 Future work

Regarding the system itself, some improvements can be made to make the context/rule system more efficient. In the current form, the system treats the rules of a context in a static way. Every time there is a new situation or an unexpected event, it is necessary to delete the corresponding system rule and insert a new one. There is no automatic mechanism, in which, given a new situation, it is possible to change the rules without eliminating them. One way to work around this problem would be to implement changes that would allow you to process the rules dynamically. In this way, instead of being repeatedly creating and erasing rules, the system would have to choose the one most appropriate to the current situation.

To implement these changes, a major modification would have to be made to the vfNegotiation application. The application would have to let the user enter various rule options and, choose the rules configurations that best suited the current situation. Deleting elements procedure (rules or context), would have to be modified, in order to erase settings as well. On the other hand, few or no changes would have to be made to vFail. As long as the application was told which rule it was evaluating, the processing mode could remain the same. In this way, with some changes in the application layer, the capabilities of the framework could be visibly increased.

7 References

Académicas, H., & Académicas, H. (2015). Carlos Pedro Carvalho Raposo Framework for Detection of Harmonisation Breaking in Service Environments.

Agostinho, C. (2011). Tuple-Based Semantic and Structural Mapping for a Sustainable Interoperability. *Ilip International Federation For Information Processing*.

Agostinho, C., & Malo, P. (2007). Harmonising Technologies in Conceptual Models Representation.

Amaxilat. (2017). amaxilat github. Retrieved from https://github.com/amaxilat/orion-client

Atzori, L., Iera, A., & Morabito, G. (2010). The Internet of Things: A survey. *Computer Networks*, *54*(15), 2787–2805. https://doi.org/10.1016/j.comnet.2010.05.010

Bannerengineering. (2017). LX sensors. Retrieved from https://www.bannerengineering.com/us/en/solutions/error-proofing/counting-dropping-parts-with-an-lx-sensor.html

Benjamins, V. R. (2000). Overview of Knowledge Sharing and Reuse Components : Ontologies and Problem-Solving Methods, 1–15.

Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., & Riboni, D. (2010). A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing*, *6*(2), 161–180. https://doi.org/10.1016/j.pmcj.2009.06.002

Bordel Sánchez, B., Alcarria, R., Sánchez de Rivera, D., & Sánchez-Picot, A. (2016). Predictive algorithms for mobility and device lifecycle management in Cyber-Physical Systems. *EURASIP Journal on Wireless Communications and Networking*, *2016*(1), 228. https://doi.org/10.1186/s13638-016-0731-0

Bouvin, N. O., Christensen, B. G., Grønbæk, K., Hansen, F. A., Olof, N., Christensen, B. G., … Hansen, F. A. (2017). New Review of Hypermedia and Multimedia HyCon : A framework for context-aware mobile hypermedia HyCon : a framework for context- aware mobile hypermedia, *4568*(June). https://doi.org/10.1080/13614560410001725310

brokenthorn.com. (2009). Operating Systems Development Series. Retrieved October 21, 2016, from http://www.brokenthorn.com/Resources/OSDev12.html

Camarinha-matos, L. M. (2012). Unit 2 : SCIENTIFIC METHOD. *Scientific Research Methodologies and Techniques*, 2009–2012.

Chen, H., Finin, T., & Joshi, A. (2003). An ontology for context-aware pervasive computing environments. *The Knowledge Engineering Review*, *18*(3), 197–207. https://doi.org/10.1017/S0269888904000025

Chen, H. L. (2004). COBRA : An Intelligent Broker Architecture for Pervasive Rule-Aware Systems. *Interfaces*, *54*(November), 129. https://doi.org/10.1.1.4.4259

Colpaert, P., Compernolle, M. Van, Vocht, L. De, Dimou, A., Vander, M., Verborgh, R., &

Mechant, P. (2014). Quantifying the Interoperability, 50–56.

CORDIS Archive : European Commission : CORDIS : FP7 : ICT : Net Innovation : FI-PPP Call 3. (2013). Retrieved January 10, 2017, from http://cordis.europa.eu/fp7/ict/netinnovation/call3short_en.html

Courses, N. (2014). Layered Network Architecture.

Doctor, F., Hagras, H., & Callaghan, V. (2005). A Fuzzy Embedded Agent-Based Approach for Realizing Ambient Intelligence in Intelligent Inhabited Environments, *35*(1), 55–65.

DZONE. (2017). Introduction to restful apis. Retrieved from https://dzone.com/articles/an-introduction-to-restful-apis

Endsley, M. R. (2000). Theoretical Underpinnings of Situation Awareness: A Critical Review. *Situation Awareness Analysis and Measurement*, 3–32. https://doi.org/10.1016/j.jom.2007.01.015

Fazio, M., Celesti, A., Marquez, F. G., Glikson, A., & Villari, M. (2016). Exploiting the FIWARE cloud platform to develop a remote patient monitoring system. *Proceedings - IEEE Symposium on Computers and Communications*, *2016–Febru*, 264–270. https://doi.org/10.1109/ISCC.2015.7405526

Ferreira, J. (2012). Monitoring morphisms to support sustainable interoperability of enterprise systems. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *7046 LNCS*, 71–82. https://doi.org/10.1007/978-3-642-25126-9_15

Ferro-beca, J. S. M., & Jardim-goncalves, C. M. R. (2013). Knowledge Management support in Sensing Enterprises Establishment.

fiware.org. (2017). FIWARE NGSI-9 API Core (v1) · Apiary. Retrieved June 5, 2017, from http://docs.ngsi9.apiary.io/#introduction/specification/introduction

Fiware Docs. (2017). Readthedocs Fiware-Orion. Retrieved June 5, 2017, from http://fiware-orion.readthedocs.io/en/master/admin/install/

Fortineau, V., Paviot, T., & Lamouri, S. (2013). Computers in Industry Improving the interoperability of industrial information systems with description logic-based models — The state of the art. *Computers in Industry*, *64*(4), 363–375. https://doi.org/10.1016/j.compind.2013.01.001

Future Internet Public and Private Parceries. (2012). Retrieved January 10, 2017, from https://www.fi-ppp.eu/

Gerhard Friedrich &, T. B. B. K. (2005). *Information and management systems for product customization*.

Giacobe, N. (2013). Measuring the effectiveness of visual analytics and data fusion techniques on situation awareness in cyber-security, (May). Retrieved from https://etda.libraries.psu.edu/paper/17537/16009

Gräßler, I., Pöhler, A., & Pottebaum, J. (2016). Creation of a Learning Factory for Cyber Physical Production Systems. *Procedia {CIRP}*, *54*, 107–112.

https://doi.org/http://dx.doi.org/10.1016/j.procir.2016.05.063

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, *29*(7), 1645–1660. https://doi.org/10.1016/j.future.2013.01.010

Harvard College. (2011). A Guide to Writing a Senior Thesis in Sociology. *Writing*, 66.

Hermann, M.; Pentek, T. (2015). Desingn principles for industrie 4.0 scenarios: a literature review, (1), 15.

Hirama, J. (2015). The History and Advanced Technology of Plant Factories. *Environment Control in Biology*, *53*(2), 47–48. https://doi.org/10.2525/ecb.53.47

Hornsby, A., Belimpasakis, P., & Defee, I. (2009). XMPP-based wireless sensor network and its integration into the extended home environment, 794–797.

Ian Horrocks, P. P. S. and F. van H. (2006). From SHIQ and RDF to OWL: The Making of a Web Ontology Language, (0).

In, D., & For, E. (2006). N EXT -G ENERATION M OBILE S YSTEMS A Context Management Framework for Supporting Context-Aware Distributed Applications, (August), 67–74.

Inc, D. (2017). Docker Architecture. Retrieved from https://docs.docker.com/engine/docker-overview/

Indulska, J., Robinson, R., Rakotonirainy, A., & Henricksen, K. (2003). Experiences in using cc/pp in context-aware Systems. *Mobile Data Management.4th International Conference, MDM 2003 Melbourne, Australia, January 21–24, 2003 Proceedings*, 247–261. https://doi.org/10.1007/3-540-36389-0_17

Json. (2012). Introducing JSON. Retrieved from http://www.json.org/

Justin Garrison. (2010). What is the Linux Kernel and What Does It Do? Retrieved October 19, 2016, from http://www.howtogeek.com/howto/31632/what-is-the-linux-kernel-and-what-does-it-do/

Kamilaris, A., Gao, F., Prenafeta-Boldu, F. X. ., & Ali, M. I. (2016). Agri-IoT: A Semantic Framework for Internet of Things-enabled Smart Farming Applications. In *Internet of Things (WF-IoT), 2016 IEEE 3rd World Forum on*. Reston.

Karnouskos, S., Colombo, A. W., Lastra, J. L. M., & Popescu, C. (2010). Towards the Energy Efficient Future Factory, 367–371.

Keith Ridgway, Chris. W. Clegg, D. J. W. (2013). The factory of the future, 53. https://doi.org/10.1109/MCS.1987.1105295

Kirkpatrick, K. (2013). Software-Defined Networking. https://doi.org/10.1145/2500468.2500473

Kreimeier, D., Morlock, F., Prinz, C., Krückhans, B., & Bakir, D. C. (2014). Holistic learning factories - A concept to train lean management, resource efficiency as well as management and organization improvement skills. *Procedia CIRP*, *17*, 184–188. https://doi.org/10.1016/j.procir.2014.01.040

Lead, D. (2017). vf-OS : virtual factory Operating System WP1 : Vision , Scenarios , and Requirements D1 . 2 : User Scenarios Characterisation -.

Lo, B. P. L., Thiemjarus, S., & King, R. (2016). BODY SENSOR NETWORK – A WIRELESS SENSOR PLATFORM FOR PERVASIVE HEALTHCARE MONITORING, 77–80.

M. Tim Jones. (2008). Anatomy of Linux process management. Retrieved October 19, 2016, from http://www.ibm.com/developerworks/library/l-linux-process-management/

Mamo, B., & Ejigu, D. (2014). A generic layered architecture for context aware applications. *Procedia Computer Science*, *34*, 619–624. https://doi.org/10.1016/j.procs.2014.07.085

Mauerer, W. (2008). *Linux ® Kernel Architecture*. *Auditing*.

Miguel, F., & Miranda, F. (2014). Context Classification for Service Robots.

Morrison, K. J. H. and R. G. (2012). *The Oxford Handbook of Thinking and Reasoning*.

Mostéfaoui, S. K., & Hirsbrunner, B. (2003). Towards a Context-Based Service Composition Framework, (June), 42–45.

Nativi, S., Mazzetti, P., & Geller, G. N. (2013). Environmental Modelling & Software Environmental model access and interoperability : The GEO Model Web initiative q. *Environmental Modelling and Software*, *39*, 214–228. https://doi.org/10.1016/j.envsoft.2012.03.007

Pablo Fernández , José Miguel Santana , Sebastián Ortega, A. T., & José Pablo Suárez , Conrado Domínguez, J. S. and A. S. (2016). Smartport: A platform for sensor data monitoring in a seaport based on FIWARE. *Sensors (Switzerland)*, *16*(3), 1–25. https://doi.org/10.3390/s16030417

Panteli, M., & Kirschen, D. S. (2015). Situation awareness in power systems: Theory, challenges and applications. *Electric Power Systems Research*, *122*, 140–151. https://doi.org/10.1016/j.epsr.2015.01.008

Patnaik, D. P. M. & S. (2013). *Intelligent Computing, Networking, and Informatics*.

Perera, C., Member, S., Zaslavsky, A., & Christen, P. (2013). Context Aware Computing for The Internet of Things : A Survey. *IEEE Communications Surveys & Tutorials*, *X*(X), 1–41.

Plant, K. L., Stanton, N. A., & Harvey, C. (2013). The role of the Perceptual Cycle in teams, (May), 21–24.

Rebecca Guenther, S. M. (2003). New Metadata Standards for Digital Resources:MODS and METS.

Restfulapi. (2017). Restful api HTTP methods. Retrieved from http://restfulapi.net/http-methods/

Robert Love. (2005). Linux Kernel Process Management | Process Descriptor and the Task Structure | InformIT. Retrieved October 19, 2016, from http://www.informit.com/articles/article.aspx?p=370047

Rubini, A., & Corbet, J. (2005). *Linux Device Drivers*. *Classcloud.Org*.

https://doi.org/10.1017/CBO9781107415324.004

Salber, D., & Abowd, G. D. (1998). The Design and Use of a Generic Context Server. *Perceptual User Interfaces Workshop (PUI '1999)*, 5–6.

Schuhmacher, J., & Hummel, V. (2016). Decentralized Control of Logistic Processes in Cyber-physical Production Systems at the Example of {ESB} Logistics Learning Factory. *Procedia {CIRP}*, *54*, 19–24. https://doi.org/http://dx.doi.org/10.1016/j.procir.2016.04.095

Schwab, K. (2015). Welcome to The Fourth Industrial. *Rotman Magazine*.

Stanton, N. A., Chambers, P. R. G., & Piggott, J. (2001). Situational awareness and safety. *Safety Science*, *39*(3), 189–204. https://doi.org/10.1016/S0925-7535(01)00010-8

Stewart, D., & Narasimhan, N. (2007). Rethinking context frameworks.

Strang, T., & Linnhoff-Popien, C. (2004). A Context Modeling Survey. *Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing*, *Workshop o*(4), 1–8. https://doi.org/10.1.1.2.2060

Stubbs, J. (2015). Distributed Systems of Microservices Using Docker and Serfnode. https://doi.org/10.1109/IWSG.2015.16

Symposium, U., Iot, I., Data, S., & Formats, M. (2015). The Internet of Things The Internet of Things, (November), 1–7.

T. Higashino. (2005). Cyber Physical Systems (CPS) Research of Higashion Laboratory. Retrieved November 6, 2016, from http://www-higashi.ist.osaka-u.ac.jp/~higashino/eng/research/CPS.html

Tan J.G., Zhang D., Wang X., C. H. S. (2005). Enhancing Semantic Spaces with Event-Driven Context Interpretation.

The Fourth Industrial Revolution Gains Momentum. (2016). *Trends Magazine*, (June 2016).

The Linux Information Project. (2004). Kernel Definition. Retrieved October 20, 2016, from http://www.linfo.org/kernel.html

The Linux Information Project. (2005). PID is used to identify processes. Retrieved October 21, 2016, from http://www.linfo.org/pid.html

Tobergte, D. R., & Curtis, S. (2013). *Linux Kernel Development*. *Journal of Chemical Information and Modeling* (Vol. 53). https://doi.org/10.1017/CBO9781107415324.004

Trindade, C., Souza, A., Pereira, J., Oliveira, J., Cacho, N., Batista, T., … Loss, S. (2016). Usando o FIWARE para Desenvolvimento de Aplicações de Cidades Inteligentes.

Tuominen, L. (2013). Future Internet in the European Union - Case FI-WARE Future Internet in the European Union - Case FI-WARE.

United Nacions Economic Comission for Europe. (2017). Data harmonization. Retrieved January 31, 2017, from http://tfig.unece.org/contents/data-harmonization.htm

Unnikrishnan A. (2009). Udev: Introduction to Device Management In Modern Linux System | Linux.com | The source for Linux information. Retrieved October 22, 2016, from

https://www.linux.com/news/udev-introduction-device-management-modern-linux-system

Veeckman, C., Paeper, D. De, Kafka, Š., (UWB), K. J., Kozhukh, D., (HSRS), K. J. (UWB) D. K., & Colpaert, P. (2016). D7 . 3 GI INNOVATION WHITE PAPER I : DATA HARMONIZATION & INTEROPERABIILTY in OpenTransportNet, (620533).

Velrajan, S. (2017). SDN Architecture. Retrieved from http://www.thetech.in/2012/12/sdn-architecture.html

Waher, P. (2016). XEP-0323: Internet of Things - Sensor Data. Retrieved from https://xmpp.org/extensions/xep-0323.html

Wang, X. H., Gu, T., Zhang, D. Q., & Pung, H. K. (2004). Ontology based context modeling and reasoning using OWL. *IEEE Annual Conference on Pervasive Computing and Communications Workshops, 2004.*, 18–22. https://doi.org/10.1109/PERCOMW.2004.1276898

Wc, P. A., & Corve, A. S. (2015). D4 . 4 Data Harmonisation and, (620533).

Wen, S., & Guo, G. (2016). Control and resource allocation of cyber-physical systems. *IET Control Theory & Applications*, *10*(16), 2038–2048. https://doi.org/10.1049/iet-cta.2016.0050

White, L. (1987). Software Testing and Verification. *Advances in Computers, Academic Press*, *26*.

Zhao, W., & He, Q. (2009). Parallel K -Means Clustering Based on, 674–679.

Zhou, W., Li, L., Luo, M., & Chou, W. (2014). REST API Design Patterns for SDN Northbound API, 358–365. https://doi.org/10.1109/WAINA.2014.153

i