



Diogo Matos Ferreira

Licenciado em Ciências da Engenharia Eletrotécnica
e de Computadores

Framework for IoT Service Oriented Systems

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

Orientador: Doutor Ricardo Jardim Gonçalves, Professor Associado com
Agregação, FCT-UNL

Co-orientador: João Filipe dos Santos Sarraipa, Professor Auxiliar Convidado,
FCT-UNL

Júri:

Presidente: Doutor Luis Augusto Bica Gomes de Oliveira
– FCT/UNL

Arguente: Doutor Tiago Oliveira Machado de Figueiredo Cardoso
– FCT/UNL

Vogal: Doutor João Filipe dos Santos Sarraipa - FCT/UNL



FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2017

Framework for IoT Service Oriented Systems

Copyright © Diogo Matos Ferreira, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa.

A Faculdade de Ciências e Tecnologia e a Universidade Nova de Lisboa têm o direito, perpétuo e sem limites geográficos, de arquivar e publicar esta dissertação através de exemplares impressos reproduzidos em papel ou de forma digital, ou por qualquer outro meio conhecido ou que venha a ser inventado, e de a divulgar através de repositórios científicos e de admitir a sua cópia e distribuição com objetivos educacionais ou de investigação, não comerciais, desde que seja dado crédito ao autor e editor.

Ao futuro...

Abstract

The forth industrial revolution is here, and with it Industry 4.0, which translates in many changes to the industry. With the introduction of paradigms like Internet of Things, Cyber Physical Systems or Cloud Computing, the so called Smart Factories are becoming a main part of today's manufacturing systems. The vf-OS Project, where this thesis falls, intends to be an Open Operating System for Virtual Factories where the overall network of a collaborative manufacturing and logistics environment can be managed and thus enabling humans, applications and devices to communicate and interoperate in an interconnected operative environment.

This thesis intends to contribute to the vision that any kind of sensor or actuator plugged to the virtual factory network, becomes promptly accessible in the operative environment and the services that it provides can be accessed and used by any API composing the system. Finally, it also aims to prove that an IoT Service Oriented System constituted of open software components can be of great assistance and provide numerous contributions to the emerging Industry 4.0 and consequently to the Factories of the Future.

With that aim, this thesis will focus on the development of two out of five interconnected applications that answer not only to different use case scenarios presented in the vf-OS but also provide solutions to answer a practical agriculture scenario, which uses mainly IoT devices and other cutting-edge technologies like cloud computing and FIWARE.

Keywords: Internet of Things, Cyber Physical Systems, Cloud Computing, Smart Factories, Factories of the Future, vf-OS (virtual factory - Operating System), Framework, IoT Service Oriented Systems.

Resumo

A quarta revolução industrial chegou, e com ela a Indústria 4.0, o que se traduz em inúmeras alterações na indústria. Com a introdução de paradigmas como a Internet of Things, Cyber Physical Systems ou Cloud Computing, as assim denominadas Smart Factories (Fábricas Inteligentes), estão-se a tornar cada vez mais uma parte fundamental dos sistemas de manufatura atuais. O projeto vf-OS, no qual esta tese se insere, tem como objetivo o desenvolvimento de um Sistema Operativo aberto para Fábricas Virtuais, que pretende ser um quadro capaz de gerir a rede global de um ambiente colaborativo de produção e logística e assim permitir que humanos, aplicações e dispositivos comuniquem e operem num ambiente operacional interligado, e é exatamente sobre quadro que esta tese incide.

Esta tese pretende contribuir para a visão de que qualquer sensor ou atuador conectado à rede da fábrica virtual se torna prontamente acessível através do ambiente operativo da fábrica e todos os serviços por ele prestados podem ser acedidos por qualquer API que componha o sistemas. Finalmente esta tese pretende ainda provar que os sistemas orientados a serviços IoT, utilizando software livre, podem servir de grande ajuda e fornecer inúmeras contribuições para a emergente Indústria 4.0 e consequentemente para as Fábricas do Futuro.

Para isso esta tese consiste no desenvolvimento de duas de cinco aplicações interligadas que pretendem responder não só a diferentes “use case scenarios” apresentados no projeto vf-OS como ainda fornecer soluções para um cenário prático de agricultura, recorrendo principalmente ao uso de dispositivos IoT e a outras tecnologias de ponta como cloud computing e FIWARE.

Palavras-Chave: Internet of Things (IoT), Cyber Physical Systems (CPS), Cloud Computing, Fábricas Inteligentes, Fábricas do Futuro, vf-OS (virtual factory – Operating System), Framework, Sistemas Orientados para o Serviço IoT.

Contents

CONTENTS	XI
LIST OF FIGURES	XV
LIST OF TABLES	XIX
ACRONYMS	XXI
1 INTRODUCTION	1
1.1 MOTIVATION AND BACKGROUND.....	1
1.2 GOALS AND CONTRIBUTIONS.....	4
1.3 THESIS ORGANIZATION.....	5
2 STATE OF THE ART	7
2.1 KERNEL.....	7
2.1.1 <i>Kernel's components</i>	10
2.1.2 <i>Monolithic Kernel</i>	11
2.1.3 <i>Microkernels</i>	12
2.1.4 <i>Hybrid Kernels</i>	13
2.1.5 <i>Exokernels</i>	14
2.1.6 <i>Kernel Architectures Summary</i>	15
2.2 FACTORIES OF THE FUTURE.....	16
2.3 WEB SERVICES FOR INDUSTRIAL INTERNET OF THINGS	20
2.4 FIWARE.....	24
2.5 IOT AS A SERVICE.....	26
2.5.1 <i>Model</i>	26
2.5.2 <i>Framework</i>	27
2.5.3 <i>Mashup</i>	29
2.6 FROM THE STATE OF THE ART TO THIS THESIS CONCEPT	31

2.7	RESEARCH QUESTION AND HYPOTHESIS	32
3	PRACTICAL FRAMEWORK.....	33
3.1	VF-OS (VIRTUAL FACTORY - OPERATING SYSTEM).....	34
3.2	USE CASE SCENARIOS.....	36
3.3	PRACTICAL SCENARIO	39
3.4	FROM THE SCENARIOS TO THE APPLICATIONS	41
3.5	TECHNOLOGIES.....	44
3.5.1	<i>Docker</i>	45
3.5.2	<i>FIWARE Orion Context Broker</i>	46
3.5.3	<i>FIWARE Short Time Historic (STH) – Comet</i>	53
3.6	SUMMARY	57
4	DEVELOPED APPLICATIONS.....	59
4.1	APPLICATIONS ENTITIES.....	60
4.2	VORDER.....	62
4.3	VFNegotiation.....	66
4.4	APPLICATIONS DEMO	70
4.4.1	<i>vOrder – Choose User</i>	71
4.4.2	<i>vOrder – Farmer User Interface</i>	72
4.4.3	<i>vOrder – Main Interface</i>	75
4.4.4	<i>vOrder – Dispatch Order</i>	78
4.4.5	<i>vOrder – Truck Creation Tab</i>	79
4.4.6	<i>vOrder – Existent Trucks Tab</i>	86
4.4.7	<i>vOrder – Generate Values Tab</i>	89
4.4.8	<i>vOrder – Check Transport Tab</i>	95
4.4.9	<i>vOrder – Delete Truck Tab</i>	102
4.4.10	<i>vOrder – Manage Truck Tab</i>	103
4.4.11	<i>vfNegotiation – Choose User Interface</i>	108
4.4.12	<i>vfNegotiation – Farmer User Interface</i>	109
4.4.13	<i>vfNegotiation – Farmer Main Interface</i>	111
4.4.14	<i>vfNegotiation – Farmer Main Interface: Add Fruit Tab</i>	112
4.4.15	<i>vfNegotiation – Farmer Main Interface: Check Fruit Tab</i>	114
4.4.16	<i>vfNegotiation – Farmer Main Interface: Update Fruit Tab</i>	117
4.4.17	<i>vfNegotiation – Farmer Main Interface: Production Values Tab</i>	121
4.4.18	<i>vfNegotiation – Buyer Main Interface</i>	123
4.4.19	<i>vfNegotiation – Buyer Main Interface: Manual Search Tab</i>	124
4.4.20	<i>vfNegotiation – Buyer Main Interface: Automatic Search Tab</i>	127
4.4.21	<i>vfNegotiation – Buyer Main Interface: Order Draft</i>	131
5	CONCLUSIONS AND FUTURE WORK.....	135
5.1	CONCLUSIONS.....	135

5.2	FUTURE WORK.....	138
	BIBLIOGRAPHY.....	141
A.	ENTITIES TABLES.....	147

List of Figures

FIGURE 2.1 - KERNEL POSITION IN THE OPERATIVE SYSTEM (KERNEL LAYOUT, 2008).....	8
FIGURE 2.2 – MONOLITHIC KERNEL ARCHITECTURE (KERNEL MONOLITHIC, 2008).....	11
FIGURE 2.3 – MICROKERNEL ARCHITECTURE (KERNEL MICROKERNEL, 2008).....	12
FIGURE 2.4 – HYBRID KERNEL ARCHITECTURE (KERNEL HYBRID, 2008).....	13
FIGURE 2.5 – EXOKERNEL ARCHITECTURE (KERNEL EXOKERNEL, 2013).....	14
FIGURE 2.6 – MONOLITHIC KERNEL, MICROKERNEL AND HYBRID KERNEL COMPARISON (OS STRUCTURE, 2008)	15
FIGURE 2.7 – CHANGES IN MANUFACTURING PARADIGMS. ADAPTED FROM(S. J. HU ET AL., 2011).....	16
FIGURE 2.8 – FIFTEEN COMPONENTS OF A FACTORY OF THE FUTURE. ADAPTED FROM (LUETH, 2015).....	19
FIGURE 2.9 – IOT MODEL: KEY CONCEPTS AND INTERACTIONS. ADAPTED FROM (BAUER ET AL., 2011).....	27
FIGURE 2.10 – OPENIOT FRAMEWORK. ADAPTED FROM (KIM & LEE, 2014).....	28
FIGURE 2.11 – IOTMAAS (IOT MASHUP AS A SERVICE) CONCEPT. ADAPTED FROM (IM ET AL., 2013).....	30
FIGURE 3.1 – PRACTICAL SCENARIO ILLUSTRATION	40
FIGURE 3.2 – USE CASE SCENARIOS APPLICATIONS APPLIED TO THE PRACTICAL SCENARIO.....	43
FIGURE 3.3 – CONTAINER VS VIRTUAL MACHINE COMPARISON (DOCKER, 2017).....	45
FIGURE 3.4 – NGSII0 SCHEMA OF REST RESOURCES (NGSII0, 2014).....	47
FIGURE 3.5 – NGSII9 SCHEMA OF REST RESOURCES (NGSII9, 2014).....	49
FIGURE 3.6 – NGSII9/NGSII0 CONTEXT ELEMENT (ENTITY) SCHEMATIZED STRUCTURE.....	52
FIGURE 3.7 – ORION CONTEXT BROKER IN A NUTSHELL (FIWARE - ORION CONTEXT BROKER, 2014).....	53
FIGURE 3.8 – ORION CONTEXT BROKER + STH WORKING SCHEMATIC.....	56
FIGURE 3.9 – SCENARIO + VAPPS + GENERIC ENABLERS USE SUMMARY	57
FIGURE 4.1 – VF-OS INTERCONNECTED APPS DEVELOPED WITHIN THE MASTER THESIS WORKGROUP	59
FIGURE 4.2 – VORDER INTERFACE	63

FIGURE 4.3 – vORDER APPLICATION SCHEMATIC	65
FIGURE 4.4 – vfNEGOTIATION INTERFACES.....	66
FIGURE 4.5 – vfNEGOTIATION APPLICATION SCHEMATIC.....	69
FIGURE 4.6 – RUNNING FIWARE ORION CONTEXT BROKER THROUGH DOCKER	70
FIGURE 4.7 – RUNNING FIWARE SHORT TERM HISTORIC THROUGH DOCKER	71
FIGURE 4.8 – CHOOSE USER INTERFACE	72
FIGURE 4.9 – vORDER: FARMER USER INTERFACE	73
FIGURE 4.10 – FARMER USER INTERFACE: ERRORS INTERFACE.....	73
FIGURE 4.11 – FARMER USER INTERFACE: FARMER CREATION ERROR.....	74
FIGURE 4.12 – FARMER USER INTERFACE: INEXISTENT FARMER ERROR	74
FIGURE 4.13 – vORDER MAIN INTERFACE FOR TRANSPORT PROVIDERS	75
FIGURE 4.14 – vORDER MAIN INTERFACE FOR BUYERS	76
FIGURE 4.15 – vORDER MAIN INTERFACE FOR FARMERS	77
FIGURE 4.16 – vORDER FARMER: DISPATCH ORDER INTERFACE	78
FIGURE 4.17 – TRUCK CREATION: ONE SENSOR	80
FIGURE 4.18 – TRUCK CREATION: FOUR SENSORS.....	81
FIGURE 4.19 – TRUCK CREATION: EMPTY TRUCK ID MESSAGE.....	82
FIGURE 4.20 – TRUCK CREATION: SUCCESS MESSAGE	83
FIGURE 4.21 – TRUCK CREATION: DELETE TRUCK SUCCESS MESSAGE	84
FIGURE 4.22 – TRUCK CREATION: FAILURE MESSAGE.....	85
FIGURE 4.23 – TRUCK CREATION TAB: TRANSPORT PROVIDER USER.....	86
FIGURE 4.24 – EXISTENT TRUCKS TAB: TRUCKS LIST.....	87
FIGURE 4.25 – EXISTENT TRUCKS TAB: LAST READ SENSOR VALUES FOR A TRUCK	88
FIGURE 4.26 – EXISTENT TRUCKS TAB: SENSORS WITH NO VALUES EXAMPLE.....	89
FIGURE 4.27 – RANDOM GAUSSIAN DISTRIBUTION	91
FIGURE 4.28 – GENERATE VALUES TAB: GENERATING 30 VALUES OVER 1 MIN.....	93
FIGURE 4.29 – GENERATE VALUES TAB: VALUES GENERATION SUCCESS MESSAGE.....	94
FIGURE 4.30 – GENERATED VALUES TAB: EMPTY FIELD ERROR MESSAGE.....	94
FIGURE 4.31 – CHECK TRANSPORT TAB: BUYER USER.....	95
FIGURE 4.32 – CHECK TRANSPORT TAB: INEXISTENT ORDER ERROR MESSAGE	96
FIGURE 4.33 – CHECK TRANSPORT TAB: FARMER USER.....	97
FIGURE 4.34 – CHECK TRANSPORT TAB: NO TRANSPORTATION PERFORMED MESSAGE	97
FIGURE 4.35 – CHECK TRANSPORT TAB: NORMAL TRANSPORT CONDITIONS VIEW.....	98
FIGURE 4.36 – CHECK TRANSPORT TAB: TRANSPORT CONDITIONS RATED AS BAD	101
FIGURE 4.37 – DELETE TRUCK TAB: SUCCESS MESSAGE.....	102
FIGURE 4.38 – DELETE TRUCK TAB: TRAVELLING TRUCK ERROR	103
FIGURE 4.39 – MANAGE TRUCKS TAB.....	104
FIGURE 4.40 – MANAGE TRUCKS TAB: ARRIVED BUTTON	105
FIGURE 4.41 – MANAGE TRUCKS TAB: EDIT SENSORS BUTTON	105
FIGURE 4.42 – MANAGE TRUCKS TAB: EDIT TRUCK SENSORS.....	106
FIGURE 4.43 – MANAGE TRUCKS TAB: EDIT TRUCK SENSORS TYPE.....	107
FIGURE 4.44 – MANAGE TRUCKS TAB: EDIT TRUCK SENSORS ID	108
FIGURE 4.45 – vfNEGOTIATION: CHOOSE USER INTERFACE	109
FIGURE 4.46 – vfNEGOTIATION: FARMER USER INTERFACE.....	110

FIGURE 4.47 – vfNEGOTIATION: FARMER MAIN INTERFACE	111
FIGURE 4.48 – ADD FRUIT TAB: EXISTENT FRUIT ERROR MESSAGE.....	112
FIGURE 4.49 – ADD FRUIT TAB: FRUIT CREATION	113
FIGURE 4.50 – CHECK FRUIT TAB.....	114
FIGURE 4.51 – CHECK FRUIT TAB: FRUIT NAME LIST.....	115
FIGURE 4.52 – CHECK FRUIT TAB: FRUIT BREED LIST	115
FIGURE 4.53 – CHECK FRUIT TAB: FRUIT INFORMATION PROVIDED	116
FIGURE 4.54 – CHECK FRUIT TAB: 0’S IN THE INFORMATION.....	117
FIGURE 4.55 – UPDATE FRUIT TAB	117
FIGURE 4.56 – UPDATE FRUIT TAB: ERRORS INTERFACE.....	118
FIGURE 4.57 – UPDATE FRUIT TAB: SEARCH DONE	119
FIGURE 4.58 – UPDATE FRUIT TAB: FRUIT UPDATE FAILURE.....	119
FIGURE 4.59 – UPDATE FRUIT TAB: UPDATE FIELDS.....	120
FIGURE 4.60 – UPDATE FRUIT TAB: FRUIT UPDATE SUCCESS.....	121
FIGURE 4.61 – PRODUCTION VALUES TAB	122
FIGURE 4.62 – PRODUCTION VALUES TAB: FRUIT’S LAST PRODUCTION VALUES.....	123
FIGURE 4.63 – vfNEGOTIATION: BUYER MAIN INTERFACE	123
FIGURE 4.64 – MANUAL SEARCH TAB: AVAILABLE FILTERS.....	124
FIGURE 4.65 – MANUAL SEARCH TAB: FILTERS AND NO FILTERS RESULTS.....	126
FIGURE 4.66 – MANUAL SEARCH TAB: NO FARMER ERROR MESSAGE	127
FIGURE 4.67 – vfNEGOTIATION BUYER INTERFACE: AUTOMATIC SEARCH TAB.....	128
FIGURE 4.68 – AUTOMATIC SEARCH TAB: AVAILABLE FILTERS.....	129
FIGURE 4.69 – AUTOMATIC SEARCH TAB: FILTERS RESULTS COMPARISON	131
FIGURE 4.70 – vfNEGOTIATION: BUYER ORDER DRAFT	132
FIGURE 4.71 – vfNEGOTIATION: BUYER ORDER ID REMINDER MESSAGE	133

List of Tables

TABLE 3.1 – ANALOGY BETWEEN SOFTWARE OS AND VF-OS MANUFACTURING OS.....	35
TABLE 3.2 – VF-OS GENERIC USE CASE SCENARIOS.....	37
TABLE 3.3 – SCENARIO STEPS EXPLANATION.....	40
TABLE 3.4 – NGS10 SPECIFIC FUNCTIONS.....	48
TABLE 3.5 – NGS19 SPECIFIC FUNCTIONS.....	50
TABLE 3.6 – NGS19/NGSI10 CONTEXT ELEMENT (ENTITY) STRUCTURE.....	51
TABLE 3.7 – FIWARE STH HISTORICAL AGGREGATION METHODS.....	55
TABLE 4.1 – LIST OF ENTITIES.....	60
TABLE 4.2 – TRUCK CREATION EXAMPLE ONE.....	82
TABLE 4.3 – TRUCK CREATION EXAMPLE TWO.....	84
TABLE 4.4 – TRUCKS CREATED FOR EXEMPLIFICATION PURPOSES.....	87
TABLE 4.5 – MEAN AND DEVIATION VALUES USED FOR EACH SENSOR.....	91
TABLE 4.6 – BOUNDARIES FOR EACH SENSOR AND PERCENTAGE OF VALUES FALLING UNDER EACH ONE.....	92
TABLE 4.7 – BOUNDARIES OF THE TRANSPORT CONDITIONS EVALUATION.....	100
TABLE 4.8 – BOUNDARIES OF THE TRANSPORT CONDITIONS EVALUATION FOR CO2 (CONCENTRATION).....	101
TABLE A.1 – FLEET ENTITY.....	147
TABLE A.2 – TRUCK ENTITY.....	148
TABLE A.3 – SENSOR ENTITY.....	148
TABLE A.4 – SUBSCRIPTION (TRUCK STATE) ENTITY.....	150
TABLE A.5 – SUBSCRIPTION (SENSOR VALUES) ENTITY.....	150
TABLE A.6 – ORDER ENTITY.....	151
TABLE A.7 – FARMER ENTITY.....	152
TABLE A.8 – FRUIT PRODUCTION ENTITY.....	153

Acronyms

FoF Factories of the Future

IoT Internet of Things

CPS Cyber Physical Systems

CC Cloud Computing

SF Smart Factories

vf-OS virtual factory – Operating System

vf-SK virtual factory – System Kernel

vf-AP virtual factory - Application Programming Interface

vf-W virtual factory - Middleware

API Application Programming Interface

AI Artificial Intelligence

XaaS Anything as a Service

IT Information Technology

OS Operating System

I/O Input / Output

CPU Central Processing Unit

BIOS Basic Input / Output System

IPC Inter-Process Communication

IIoT Industrial Internet of Things

RFID Radio-Frequency Identification

SOA Service-Oriented Architecture

UDDI Universal Description, Discovery and Integration

WS Web Services

IaaS Infrastructure as a Service

PaaS Platform as a Service

SaaS Software as a Service

IoTaaS IoT as a Service

IoTMaaS IoT Mashup as a Service

ICT Information and Communication Technology

M2M Machine to Machine

GE Generic Enabler

VM Virtual Machine

NGSI Next Generation Services Interface

REST REpresentational State Transfer

HTTP HyperText Transfer Protocol

DB DataBase

STH Short Time Historic

1 Introduction

In this chapter, a brief introduction on the situation under which this thesis is developed is presented. This situation is the past and current development of the factories, manufacture and agriculture worlds. All of these big industrial worlds are constantly in development and modernization, and a new industrial revolution is arriving, getting together all these previously distinct worlds and turning them into one large interconnected industrial world. While the European vf-OS project aims to provide answers to the combination of all these big industrial worlds, this thesis' goal is to provide small tools, to help the vf-OS answer the requirements of the interconnected industry.

After the presentation of the background of the vf-OS project which includes this thesis and the enlightenment about the motivation for this thesis development, the primary goals of this thesis are enumerated and explained.

Finally, at the end of this chapter an overview of this document is given, explaining its organization and the information presented in each chapter of this master thesis document.

1.1 Motivation and Background

“The First Industrial Revolution used steam power to mechanize production. The Second used electric power to create mass production. The Third used electronics and information technology to automate production. Now a Fourth

Chapter 1. Introduction

Industrial Revolution is building on the Third. It is characterized by a fusion of technologies that is blurring the lines between the physical, digital, and biological spheres” – Professor Klaus Schwab, Founder and Executive Chairman of the World Economic Forum.

Since the First Industrial Revolution in 1784, with the transition from hand production to mechanize production, the industrial world has been experiencing constant technology evolution, further improving the way industry works. Lately, however, the technology used is evolving exponentially, and the changes it is bringing are dramatic to this world. The digitization is taking over the industrial domain, with many emerging technologies being used in factory production, such as artificial intelligence, advanced robots, complex sensors, Internet of Things, cloud computing, digital fabrication, XaaS (“anything as a service” – cloud computing term for services and applications accessed on demand over the Internet as opposed to being utilized on premises), autonomous vehicles or machines, and so on. There are also other technologies still under development that can also make a great difference in the industrial area like, nanotechnology, energy production and storage or even the much-anticipated quantum computer. Making use of all these technologies factories are reducing costs, improving efficiency, increasing speed and scale, developing smarter products and services, all this in a more sustainable way. With the combination of the digital and the physical worlds, intelligent products and machines can “talk” to one another across the value chain, storing and transmitting vital and real-time information, combining communications, IT (Information Technology), data analysis and decision-making ability, thus turning traditional factories into smart digital manufacturing environments using cyber-physical systems.

Another advantage brought by the incorporation of both physical and digital worlds into the industry ecosystem is the creation of virtually simulated factories. This virtual factory is a computer-generated copy of the real factory where operators can simulate the normal operation of the factory, predict, find and solve malfunctions, try to add new services to the virtual factory before applying them to the physical one, to see its response, keep track of the products whereabouts and state as well as constant knowledge about the factories numbers, amount of items produced, amount of machines working, etc. In order to achieve it, there is a need to develop a platform on which future manufacturing

1.1 Motivation and Background

applications can be built. Even though there are already some platforms able to do this, like for example AUTOSAR, ISOBUS or SmartAgriFood, they apply only to very specific industrial sectors (automotive, agricultural machinery and agri-food, respectively), which means that there is yet no open platform across all sectors, using open standards common to all of them.

The problem about creating an open platform able to be shared by all sectors is the diversity of technologies used by each one and therefore the existence of diverse and unconnected disjoint systems that are not interoperable. Yet, not only to interconnect different sectors, is this open platform useful, within the same sector many additions have been made to the industries over time, resulting in a highly heterogeneous manufacturing environment, to which more updates can be hard and expensive to apply. Therefore, an open general platform could greatly benefit both intra and inter sectors, big and small enterprises not only to make upgrades easier to implement but also cheaper. In order to fulfil this need, the European Union under the Horizon 2020 Programme commissioned the creation of a Virtual Factory Open Operating System.

The goal of the vf-OS Project is to develop an Open Operating System for Virtual Factories, which aims to become the reference system software for managing factory related computer hardware and software resources and providing common services for factory computational programs. This operating system will be the component of the system software in a real factory system where all factory application programs will run. There, a virtualization of the whole factory can be accessed and controlled and a range of services will be provided to integrate better manufacturing and logistics processes. This virtualization can always be further improved by developing and deploying smart applications in order to optimise communications and collaboration among supply networks of all manufacturing sectors in all the manufacturing stages and logistic processes. From the framework embedded in this vf-OS the overall network of a collaborative manufacturing and logistics environment can be managed as the operating system will serve as an intermediary between the application behaviour of the factory and the factory hardware itself. In short, the vf-OS will allow any type of factory to have its own virtual representation and virtualized services and functionalities, by simply developing and installing their machines applications

Chapter 1. Introduction

into the operative system. The platform will then add them to the factory virtualization and make them available through system calls to the OS.

1.2 Goals and Contributions

The purpose of this thesis, developed within the scope of the vf-OS project, described above, is the registration, discovery and provision of services or devices provided by installed sensors, machines or software components through the internet, more precisely through a cloud based database. During its development, the registered and discovered devices are mainly sensors, of all sorts, that allow the users of the developed applications, to keep track of values, vital to the production and the transport of goods. Besides the installed IoT devices, also critical information needs to be stored in the cloud database and easily reachable by the users.

Two framework applications are to be developed within a full functional real-life project composed of five interconnected applications. These interconnected applications aim to provide solutions to a small part of the vf-OS goals, using European technologies such as the FIWARE Program. Furthermore, these applications are applied to a real case scenario, and intend to completely fulfil all the problems found in that scenario. A smooth relation between all the applications is the primary goal of the project, since they will all need to work together to better provide the solutions to the presented scenario and ultimately to the vf-OS goals.

The developed application frameworks are expected to be accessible to all the users using the vf-OS project, at any given time, always providing all the necessary functionalities and information about the registered and discovered services and devices as well as any other essential information to the user. Besides that, the applications aim to be scalable, to incorporate large amounts of functions, users, and goods, interoperable, to be used alongside other functions and programs, user-friendly, so that it can be used by all types of people, from experts to lay people, and portable and easily deployable so that any person can use it from his personal computer.

1.3 Thesis Organization

This master's thesis is composed by 5 chapters organized as follows:

Chapter 1 - In chapter 1, "Introduction", the present chapter, is given a brief introduction about the background domain under which this master thesis falls (the past and current factory, manufacture and agriculture worlds), as well as the motivations behind the development of a master thesis in this field. The chapter ends with the main goals and objectives of this thesis, and its organization.

Chapter 2 - This "State of the Art" chapter provides several different information about topics related to this master thesis development. It starts with defining the central part of the Operative System which is where the vf-OS will be installed and from where it will run - the kernel. After that definition is introduced, a more detailed description about the background to where this thesis intends to contribute is given. With the background explained, alternative ways of incorporating the IoT in that world is presented starting by listing alternative, already used ways, going through the source of the technologies used in this thesis development, and ending with the value that these services can bring to the factories of the future. At the end of the chapter is presented the Research Question and the Hypothesis that guide all the development of the master thesis presented in this document.

Chapter 3 - In this chapter is presented the "Practical Framework" that shapes this thesis development. A proper presentation of the project that comprehends this master thesis, the European vf-OS Project, is given as well as the Use Case Scenarios found by the project developers. From the combination of some of these Use Case Scenarios a real-life scenario was created and is also presented in this chapter. After the scenario is set, the technologies used create the applications that will answer that scenario are presented and explained. Finally, this chapter ends with a summary establishing the relations between the use of the presented technologies, the applications created from the listed Use Case Scenarios and the real-life scenario.

Chapter 1. Introduction

Chapter 4 - This chapter explains the “Developed Applications” during the course of this master thesis progress. It starts with the presentation of a technology capable of making the created applications portable - the Docker. After that technology is briefly explained, all the entities (structures where the application’s vital information is stored) used by both the developed applications are listed and explained. Finally, this chapter goes through each of the application’s functionalities showing what it does and how it should be used. That explanation is given through the use of elucidative pictures of the applications during their normal operation. At the end of each application a summary is done explaining how a real user could make use of the application’s functionalities.

Chapter 5 - This chapter concludes this master thesis document and presents its “Conclusions and Future Work”. In this chapter is explained how this thesis answered to its main goals, which contributions it offers to both the answered scenario and the project where it falls (the vf-OS Project), and is also shown how this project proved that an IoT Service Oriented System would be a great asset, to be added to the industry of the Factories of the Future, which was the main question that this thesis wanted to answer. At the end of the Conclusions, parallel work done during the writing of this thesis is presented in the form of two papers, already accepted and published, which can be found included in the appendixes. After those conclusions are presented, some ideas of possible future works on how to change or improve the developed applications are identified.

2 State of the Art

In this chapter is presented fundamental knowledge necessary to the understanding of this thesis' related work. Since this thesis falls under the vf-OS project, whose aim is to create a standard Operative System capable of being used by all industries, this chapter starts by giving some information about where an essential part of an Operative System, the Kernel, on top of which the vf-OS will run. After this essential information is presented, an explanation about the evolution of the manufacture world is given, from its beginning to the present day and into one of its possible futures. In order to introduce this master thesis' contribution to that world, several different alternative ways of adding the IoT to it are presented like the Web Services, including the source of the technologies used during this thesis development, the European open source generic enablers, program FIWARE. Finally, it is explained how the IoT can be used as a service to benefit the factories of the future, and this chapter ends with the introduction of the Research Question and Hypothesis that guide this master thesis development.

2.1 Kernel

Dictionaries generally define kernel as “the central or most important part of anything”, “gist”, “core”. This applied to technology, and more precisely to computers shows that the kernel constitutes the central core of an operating

Chapter 2. State of the Art

system. One main aspect that shows how important the kernel is in a computer, is the fact that it is the first program of the operating system loaded into memory on system startup and it is the one that manages the rest of the startup.

Kernel however isn't exactly mandatory, i.e. it isn't strictly essential for an operative system to work. Programs can also be loaded and executed without the kernel to manage them. In fact, in the early days of computers when kernel wasn't used yet, computers could also be started and programs could also be loaded. An example of this type of startup could be found in are the early console video game systems, which had no kernel so, rebooting the system was required every time a new game designed for that console was to be executed (Mike, 2009).

In order to avoid having to restart the system every time a new program needs to run, each of these programs would need to have its own bootloader and direct hardware controlling. And this is one of the most important things that the kernel adds to an operating system. It provides the capability of executing multiple programs, at any given time while the other programs continue to run.

Besides managing the programs that run in the system, the kernel also provides basic services for all other parts of the operating system. These services include memory, processes and files management, as well as I/O (input/output) management which allows the computer to communicate with any kind of peripherals (keyboard, mouse, printers, speakers, etc.).

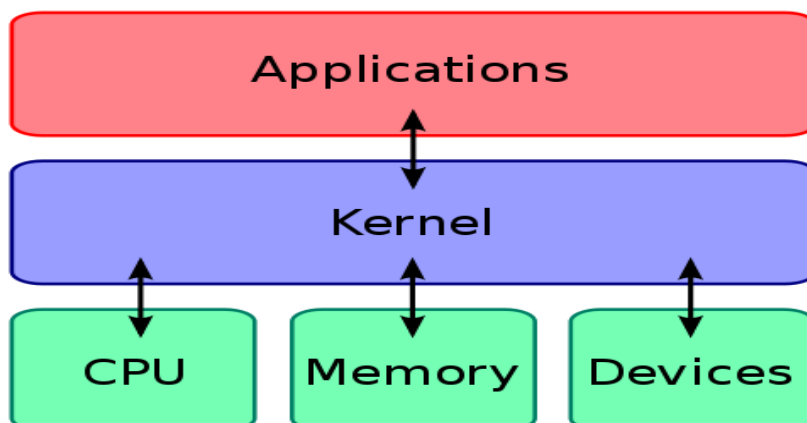


Figure 2.1 - Kernel Position in the Operative System (Kernel Layout, 2008)

Due to its outmost important role inside the operative system, the kernel code usually runs in a protected area of computer's memory. This fact intends to prevent it from being overwritten by other, less important parts of the operating system or by application programs. Also, in order to get an additional defence, the kernel has its own kernel space where it performs its tasks, detached from the user space even though they both coexist in the System Memory.

➤ **Kernel Space** is a slot in the System Memory where the kernel executes and from where it provides its services. This "space" can only be accessed by user processes through system calls, which occur when a process requests the kernel to perform something, like a process creation or an I/O communication establishment, etc.

➤ **User Space** is the "space" in memory where everything the user does its temporarily saved, from writing a document to running a program. When a process (instance of a program) is being executed, a copy of that program is transferred from the storage into the user space so that it can be accessed at a high speed by the CPU (central processing unit).

This division of the System Memory aims to prevent the user data to interfere with the kernel data in order to make the system more stable and secure.

Chapter 2. State of the Art

2.1.1 Kernel's components

Even though kernel's components can vary depending on the operating system, it can be said that almost every kernel include the following components:

1. **Scheduler**: manages the usage of the kernel time by the various system calls done by the processes and also the order by which they will use it;
2. **Supervisor**: oversees the computer usage by the processes when it is their time to use it (time given by the Scheduler);
3. **Interrupt Handler**: handles the many requests originated by the hardware devices that compete for the kernel's services;
4. **Memory Manager**: allocates the system's memory (i.e. tells the programs where they should be regarding the memory usage).

Beyond these components, different kernels can have more specific components and unlike the BIOS (Basic Input/Output System), which people tend to confuse with kernel, a computer's kernel can easy be replaced or upgraded with more and different components (The Linux Information Project, 2005).

A computer can use one of several different types of kernels. These kernels vary on their type of architecture and all of them have their pros and their cons as well as their advocates and their detractors. Based on their architectures they can therefore be split into four main classifications.

2.1.2 Monolithic Kernel

In Monolithic Kernels, all the processing, I/O communicating, devices, memory and hardware handling, etc. done by the kernel, happen in the Kernel Space and it also retains full privilege access over the various components under their control.

Therefore, the Kernel Space used by a Monolithic Kernel needs to be larger than the Kernel Space used by other Kernel Architectures, because it focuses all the necessary code in this space and it deals with all the computer processing there. This fact, besides needing greater space usage, makes the code heavier, slower to load and require its source code to be changed every time the kernel needs to be updated or fixed.

On the positive side, because of the fact that it condenses all code and processing in just one space, it decreases the number of context switches and messaging involved in its provided services making this architecture faster than other architectures more decentralized.

Linux is a very notorious example of this type of Kernel, where constant update and replacement is part of the way it has been conceived.

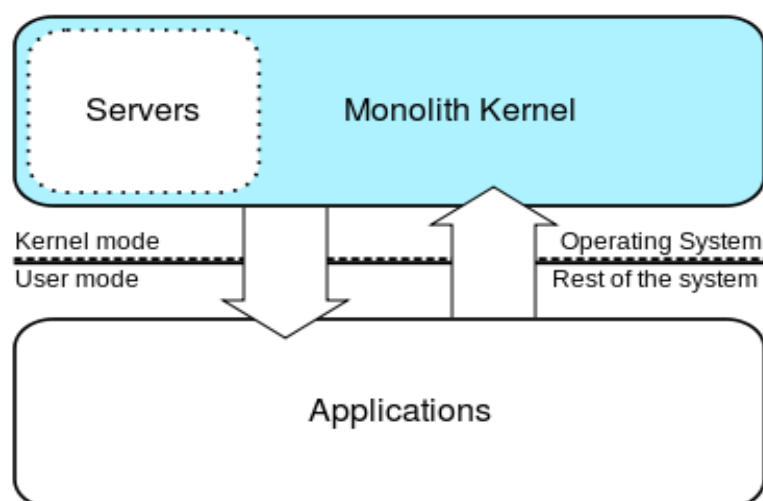


Figure 2.2 - Monolithic Kernel Architecture (Kernel Monolithic, 2008)

Chapter 2. State of the Art

2.1.3 Microkernels

In this architecture the Kernel, using the Kernel Space contains only minimal amount of functions, such as process management, inter-process communication (IPC) and memory allocation and transfers all the other services to be run in the User Space. This allows the kernel code to be split and some of its functions to be run as daemons (able to be turned on and off when needed, like normal programs).

Even though these Kernels need a greater amount of context switches and message trading, making them conceptually slower than Monolithic Kernels, they are much more responsive, more stable and easier to change and upgrade due to the lower amount of code they possess in Kernel Space.

One Operating System that makes use of this architecture is the *AmigaOS*, due to the fact that it has no memory protection it was able to trade messages very fast thus overcoming one of the downsides of this architecture.

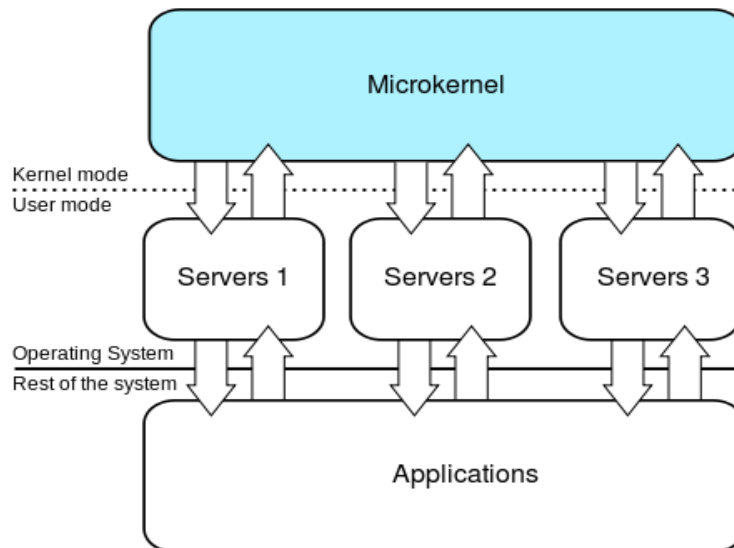


Figure 2.3 - Microkernel Architecture (Kernel Microkernel, 2008)

2.1.4 Hybrid Kernels

Hybrid Kernels, as the name suggests, are a combination of the architecture of both the Monolithic Kernels and the Microkernels. Unlike a Microkernel where almost every service is run in the User Space, or a Monolithic Kernel where everything runs in the Kernel Space, a Hybrid Kernel still run many of its functions in User Space but keep some of them in the Kernel Space.

This approach allows the Kernel to make use of the advantages of both the mentioned architectures. It combines the speed and simpler design of the Monolithic Kernel with the modularity and execution stability and safety of the Microkernel architecture.

The best-known example of this Kernel Architecture is the Microsoft NT Kernel that powers all Windows OS from Windows NT forward, since it makes use of its ability to use different modules that can communicate with the Kernel itself and with other modules.

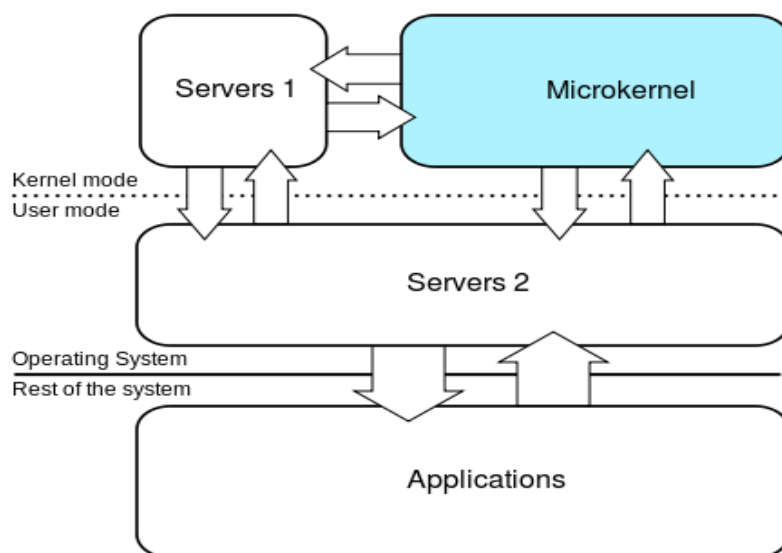


Figure 2.4 - Hybrid Kernel Architecture (Kernel Hybrid, 2008)

Chapter 2. State of the Art

2.1.5 Exokernels

This Kernel Architecture is an experimental approach developed by the Massachusetts Institute of Technology (MIT) that is much smaller in size due to its limited operability.

Exokernels were built with the intent to eliminate the notion that an operating system must provide abstractions upon which to build applications. In order to do this, they impose as few abstractions as possible, instead allowing application’s developers to efficiently choose to implement or not whatever abstractions are best suited to the applications’ task.

To accomplish this, the Kernel moves the hardware abstractions into untrusted libraries located in the User Space, called “Library Operating Systems” (libOSes). Using those libraries can grant the developers access to different Operating Systems applications, such as, for instance they can simultaneously run both Linux and Windows applications.

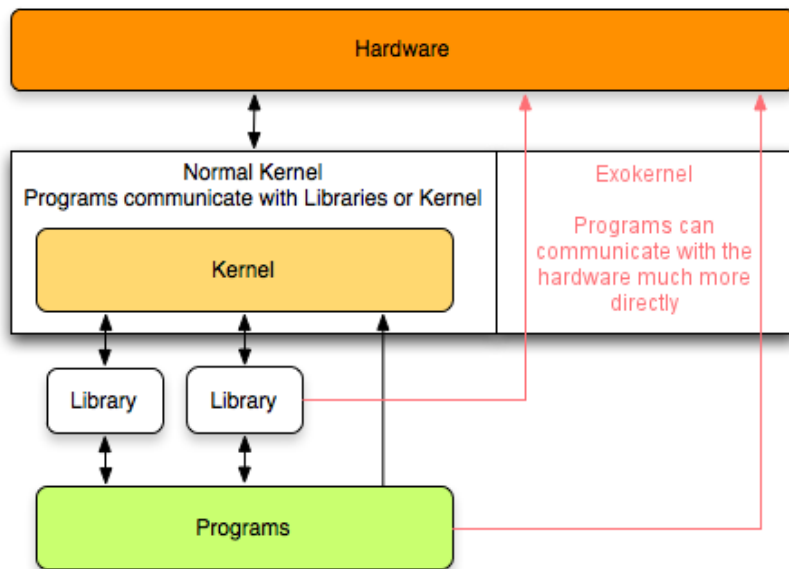


Figure 2.5 - Exokernel Architecture (Kernel Exokernel, 2013)

2.1.6 Kernel Architectures Summary

In addition to the listed architectures there are many others with relatively minor variations. Between the listed ones and all the others, the reason to choose either one is based not only on the mentioned pros and cons of each but also depend upon personal choice, reliability, speed and how easy specific goals can be reached using each Kernel.

On one side of the scale is the Monolithic Kernel that retains full privilege access over the various components under his control and manages all the system by himself from his own personal code and memory space (Kernel Space). At the other end of the scale, the Microkernel provides as few as possible Kernel services and negotiates the rest of his functions to user mode components located in the User Space. Between them two is the Hybrid Kernel that makes use of both the Monolithic Kernel and the Microkernel which is the most used by the modern Operating Systems.

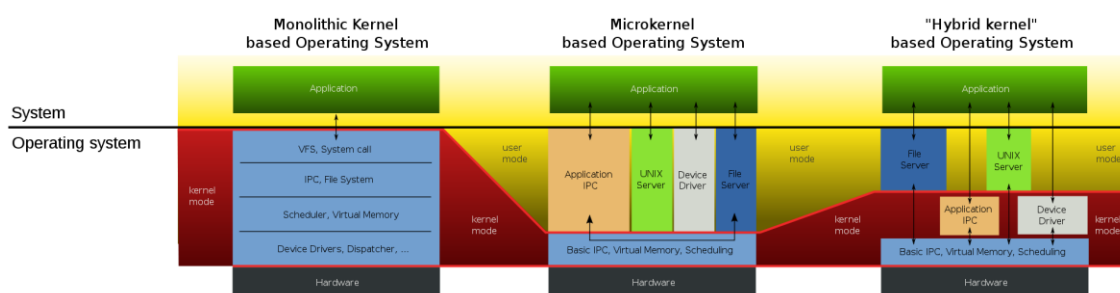


Figure 2.6 – Monolithic Kernel, Microkernel and Hybrid Kernel Comparison (OS Structure, 2008)

Chapter 2. State of the Art

2.2 Factories of the Future

Factories are commonly known as buildings or, sets of buildings where manufactured goods are made from raw materials on a large scale. Factories began to appear when the population needs became too large for the workshops or cottage industry production capacity. In order to keep up with the population demand of goods, machines were developed to help humans with the manufacturing of goods. In the factories, a large part of the work is done by machines while humans are there to operate these machines and to ensure they are producing the materials with the necessary quality.

With the advent of factories and their growing popularity and use, a new kind of industry was formed, the Manufacturing Industry. Manufacturing Industry however didn't remain idle, it kept evolving over the years through several paradigms. It started as **Craft Production**, but quickly evolved to **Mass Production** and more recently to **Mass Customization**.

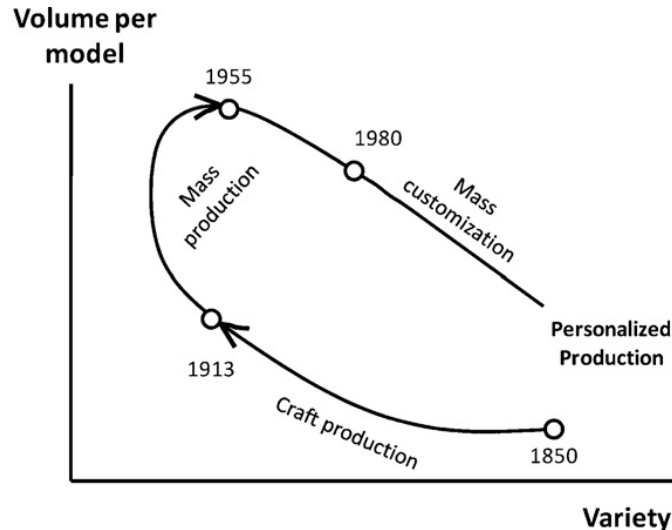


Figure 2.7 - Changes in Manufacturing Paradigms. Adapted from(S. J. Hu et al., 2011)

In Figure 2.7 we can see the evolution of the Manufacturing Industry paradigms over the years on a Variety vs Volume chart.

The first paradigm – **Craft Production** – each product was requested specifically by a customer and it was done according to his demands, which translated, as can be seen in the chart, in a high variety / low volume type of industry. During this paradigm each product was unique, exactly what the customer wanted, but it was very expensive. Also the product was made in a single, specific location and the production was not scalable (S. Jack Hu, 2013).

In order to increase the amount of products made and to reduce their cost, a second paradigm appeared – **Mass Production**. With the introduction of this paradigm the products did have a lower cost, however their variety decreased drastically as they were made in large volumes where every product looked exactly the same. This paradigm was described very clearly in Henry Ford's statement "Any customer can have a car painted any colour that he wants so long as it is black" (Ford, 1922). During this paradigm lots of changes happened in the industry, the division of labour allowed each worker to focus on a specific repetitive task while an assembly line led all the pieces previously made right to his hands (S. Jack Hu, 2013), which led to faster production lines and minimal errors due to distraction.

When people got tired of everyone having the exact same products a new paradigm emerged – **Mass Customization** – the main goal of this new paradigm was to provide the customers with a very customized product at the cost of a Mass Production product. In order to allow a large volume of products coming from a Mass Production to be customizable, this paradigm stated that certain modules of the product are equal among all the products coming from that production line, while other modules are provided with several variants (Herrmann, Schmidt, Kurle, Blume, & Thiede, 2014).

Chapter 2. State of the Art

Alongside the need to produce at a Mass scale with the variability of unique personalized product, the companies that produce those goods are under an ever-increasing pressure to lower the fossil resources usage and consequently reduce the pollution emissions. Therefore, the Factories of the Future must find a sustainable, environment friendly way to keep producing what the people are looking for.

In order to be sustainable a Factory of the Future has to address all three dimensions of sustainability - economy, ecology and society (Herrmann et al., 2014).

First, in the economy dimension - factories need to keep the production as high as the demand requests at the lowest cost possible;

Ecologically - factories need to not only meet the governments limits related to pollution level, fossil resources consumption, etc. but also try to positively influence its surroundings like recycling its wastes and look included in the landscape;

Last but not the least a factory must have a positive impact in the people live, not only make its workers feel needed but also stimulate learning, collaborative work and provide a good working environment.

The ultimate goal of the factory of the future is to interconnect every step of the manufacturing process. Factories nowadays tend to be more and more decentralized. They work across domains, geographic boundaries, value chains, life cycle phases, etc. (OECD, 2011). In order to integrate all their components, they must resort to a way of "approaching" their most distant parts, and the best way to do it is through the Internet of Things.

The main purpose of the Internet of Things is to link any type of objects in the physical world through a virtual representation in the internet. Being able to maintain connectivity and visibility across operations supply chains and business partners allows increasing the efficiency in many fields, such as reducing wastes, improving transport quality and swiftness, reducing manufacturing time and costs or even eliminating the need of high amounts of stock.

Future factories increasingly want to move away from a make to-stock manufacturing approach and mass production, and embrace more of a quality-

2.2 Factories of the Future

driven, make-to-order approach to meet defined and specific customer needs. In order to accomplish the mentioned goals, at the heart of the factory of the future will be data, visible, comprehensible and actionable (Zebra Technologies, n.d.).

Making use of the global, physical assets (goods and machines) awareness provided by the Internet of Things, allowing real time knowledge of their status and location, and the information stored in the factory data it is possible for the Factories of the Future to reduce the manufacturing time of goods while still controlling and reducing defects and eliminating over-production.

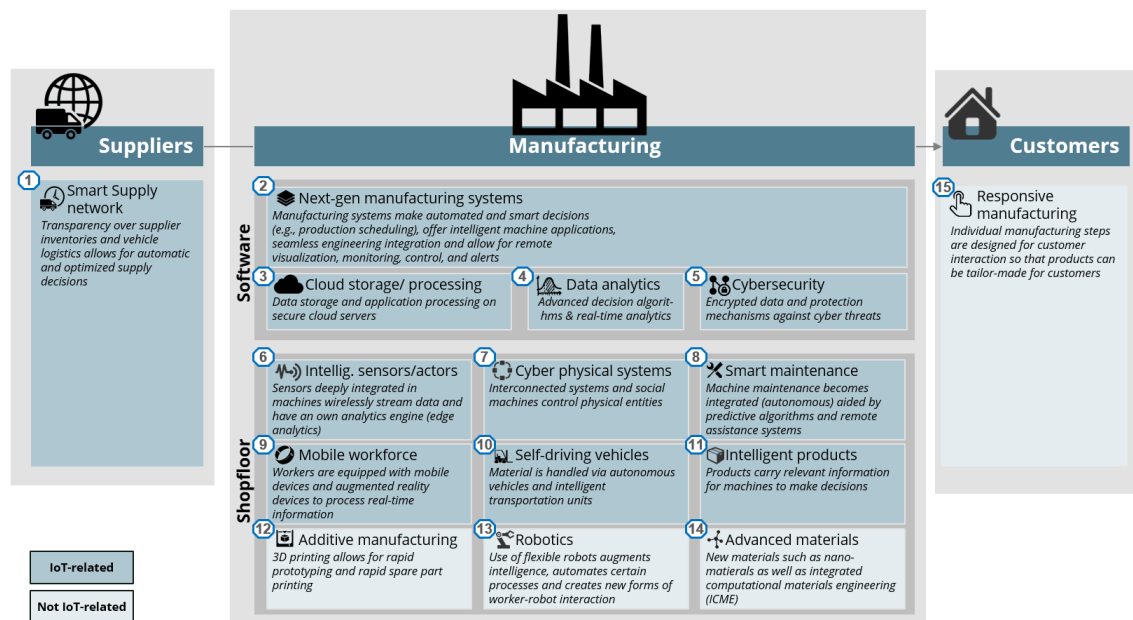


Figure 2.8 - Fifteen Components of a Factory of the Future. Adapted from (Lueth, 2015)

The Manufacturing Industry has come a long way since the creating of the first factories to the idealization and creation of the “Smart” Factories of the Future. Processes have been fastened, mass production and mass customization have been developed and new ways to increase the efficiency have been implemented. Fifteen of the main components that constitute a Factory of the Future can be seen in Figure 2.8, and many others are still in development, in order to further meet the pollution limits, the people’s demand and reduce cost and time of production.

Chapter 2. State of the Art

2.3 Web Services for Industrial Internet of Things

Nowadays, due to the increasing development of the technology used in factories and in industry in general which aim to further improve the way robots and machines in industry are being more and more interconnected and optimized, there is a branch of the Internet of Things (IoT) dedicated especially to the industry, the Industrial Internet of Things (IIoT).

There is no clear and fully accepted definition of the Internet of Things (IoT) paradigm, even searching the literature it might be difficult to understand what IoT really means, which basic ideas stand behind this concept, and which social, economic and technical implications the full deployment of the IoT will have. The first place where ideas differ is right on the approach taken by each stakeholders, business alliances, research and standardization bodies concerning the Internet of Things, each of these start approaching the issue from either an “Internet oriented” or a “Things oriented” perspective, depending on their specific interests, finalities and backgrounds (Atzori, Iera, Atzori, Iera, & Morabito, 2010).

According to (Atzori et al., 2010), the very first definition of IoT derives from a “Things oriented” perspective; the considered things were very simple items: Radio-Frequency IDentification (RFID) tags and the term “Internet of Things” is, in fact, attributed to The Auto-ID Labs (Auto-ID Labs) a world-wide network of academic research laboratories in the field of networked RFID and emerging sensing technologies. However, the semantic origin of the expression is composed by two different words bearing different concepts, while “Internet” can be defined as “The world-wide network of interconnected computer networks, based on a standard communication protocol”, “Things” is “an object not precisely identified. Thus, when putted together the words “Internet” and “Things” in “Internet of Things” assume a semantical meaning of “a world-wide network of interconnected objects uniquely addressable based on standard communication protocols” (On, Ystems, Ep, & Lange, 2008).

2.3 Web Services for Industrial Internet of Things

Simply put, such as (Jeschke, Brecher, Meisen, Özdemir, & Eschert, 2017) ascribes to (Atzori et al., 2010), “The Internet of Things (IoT) is an information network of physical objects (sensors, machines, cars, buildings, and other items) that allows interaction and cooperation of these objects to reach common goals”. The Internet of things has today many applications such as transportation, healthcare, smart homes and industrial environments (Whitmore, Agarwal, & Xu, 2016). When applied to the last, it can be called “Industrial Internet of Things” (IIoT). Its main purpose is to transform the way field assets (e.g., machines or robots) connect and communicate within a factory or between factories, resorting to the use of sensors, advanced analytics, and intelligent decision making (Wang, Wan, Li, & Zhang, 2016). Through it sensors, machines, and Information Technology (IT) systems will be able to interact with one another using industrial internet technology and send constant and real-time reports to whom they may concern.

According to a study commissioned by Forrester Consulting, 67% of the surveyed manufactures are concerned with lack of standard interfaces and interoperability challenges (Forrester Consulting 2015). Some efforts are being made in order to implement the IIoT in today’s industry, starting by addressing the standardization challenges, promoting open interoperability and the widespread usage of a common architecture. International Electrotechnical Commission (IEC), Standardization Management Board (SMB), for instance, has established in 2014 a Strategy Group, SG8, to deal with a number of tasks related to smart manufacturing, whose primary focus is to “leverage the adoption of current and next generation technologies to achieve safe and secure factory operations” (IEC.ch). The Industrial Internet Consortium (IIC) was founded with the purpose to, among other things, further improve the industry by accelerating the development, adoption, and widespread use of interconnected machines, devices, and intelligent analytics (IIC.org). Also the Institute of Electrical and Electronics Engineers (IEEE) Project P2413 and OneM2M, aim to focus on developing an architecture framework for IoT and defining how devices and services are used in the IoT communication, (IEE, 2016) and (onem2m.org).

Despite all this progress and improvement, there are still many ways to where IIoT can be further improved and optimized. According to (Jung, Wat-

Chapter 2. State of the Art

son, & Usländer, 2017) test-beds for smart production technologies (called experimental factories) have been created with the purpose of establishing interoperability guidelines and applying new IT technologies in existing automated systems, and thus demonstrate how technologies from different organizations can work together and support new innovations. However, still according to (Jung et al., 2017) has been no attempt to interconnect these experimental factories and allow them to flexibly adapt their production capabilities based on cross-site demands, so a project was created by Korea Evaluation Institute of Industrial Technology (KEIT) to build a way to easily interconnect different factories in order to meet all the demands (keit.re.kr).

In order to manage the usage of all these smart embedded devices in industry, applications become necessary to better integrate real-time state of the physical world, and hence, provide services that are highly dynamic, more diverse, and efficient. To incorporate these applications Service-Oriented Architecture (SOA) is in order, traditionally used to couple functionality of heavy-weight corporate Information Technology (IT) systems, and besides, in such infrastructures, composed of large numbers of networked, resource-limited devices, the discovery and usage of remote services is a significant challenge (Guinard, Member, Trifa, & Member, 2010). Web Services can, therefore, be used to allow each device to offer its functionalities and, at the same time, discover and invoke others functionalities offered by services of other devices dynamically and on-demand, as suggested by (Karnouskos, Baecker, & S, 2007).

To keep the available Web Services always accessible and organized the Universal Description, Discovery and Integration (UDDI) registry is widely used (for example the jUDDI by OASIS, <https://juddi.apache.org/>). The usage of UDDI allows the Web Services to be easily found and used, and furthermore, their registration and discovery turns transparent to the Web Services providers. According to (Qian, Baokang, Yunjian, Jinshu, & You, 2014), there are 3 roles in SOA: Service Provider, Service Registration Centre and Service Requestor, and is in the Service Registration Centre, that the Web Services are stored. There, the UDDI gives descriptive information related to the Web Services and, at the same time, also includes the standard specifications of Web services information registry (Liu, Liu, & Chao, 2007).

2.3 Web Services for Industrial Internet of Things

Web Services can be provided through Cloud Computing, especially as Software as a Service. In Cloud Computing environment, all the computing infrastructure resources are provided as services over the Internet, like Infrastructure as a Service(IaaS), or Platform as a Service (PaaS) or Software as a Service (SaaS) (like the Web Services), etc. Also, Cloud Computing is very used in Service Oriented Architectures (SOA) which are mainly implemented by Web Services (Duan, Yan, & Vasilakos, 2012), and because computer systems leased from a cloud service provider, are typically connected to internet, they can host web services. Such architectures have recently been adopted in factory automation, as they allow systems to reach high levels of decentralization. Those SOA-based Factories systems become able to combine physical production equipment with Web Services that belong to the information processing (cyber) domain, and that can be deployed on cloud resources (Puttonen, Lobov, Soto, & Lastra, 2016).

Chapter 2. State of the Art

2.4 FIWARE

FIWARE or FI-WARE is a community founded by the European Union whose goal is to provide a middleware platform where developers can create and deploy applications and services for the Future Internet. Once a service or application is created, they are filed in FIWARE Catalogue where anyone can open and use them at their own free will since they all are stored in a public and royalty-free platform.

This FIWARE community is an open one, which is formed not only by the developers of the technology but by all those who contribute to materialize the FIWARE mission: “to build an open sustainable ecosystem around public, royalty-free and implementation-driven software platform standards that will ease the development of new Smart Applications in multiple sectors” (Firmware.org, 2016).

In order to cover a large variety of purposes the FIWARE Community has some sub-programs besides the core FIWARE program where it provides a set of APIs (Application Programming Interfaces) like previously mentioned. Among these sub-programs, one can find:

FIWARE Lab - where the users can test the provided technology or their own, in some experimental infrastructures developed by the FIWARE community where entrepreneurs and domain stakeholders can meet;

FIWARE Accelerate - where interested developers are encouraged to develop new bold solutions to further improve the FIWARE platform. This program focuses mainly in SMEs (Small and Medium-sized Enterprises) and start-ups.

FIWARE Mundus - Program which ambitions are to turn the Europe-size community into a worldwide community, reaching foreign stakeholders and governments.

FIWARE iHubs – as the name implies, aims to create operational Hubs nodes in order to build communities of adopters and contributors at a local level.

Further information about the FIWARE platform and community can be found on their original website (www.fiware.org).

One of the many applications that the FIWARE platform can have is in the Smart Cities domain. Many of the developments achieved in the FIWARE community can be used to improve the concept of the Smart Cities creating smart cities applications which in time will attract enterprises and start-ups who will further innovate and provide a better city for citizens and businesses (Crouch, 2015).

In the FIWARE Catalogue is present a rich library of components, already programmed, implemented and ready to be used and/or changed by anyone since they are all public royalty-free and open source, called Generic Enablers. There, Enablers can be found which can be used in many different contexts such as:

- Internet of Things (IoT) Services Enablement;
- Advanced Web-based User Interface;
- Security;
- Interface to Networks and Devices (I2ND);
- Architecture of Applications/Services Ecosystem and Delivery Framework;
- Data/Context Management;
- Cloud Hosting.

In the last two are included not only the Enablers used during the development of this thesis “FIWARE Orion Context Broker”, described in chapter 3.5.2 and “FIWARE Short Time Historic (STH) – Comet”, described in chapter 3.5.3, but also the program used to run these enablers, the “Docker” described in chapter 3.5.1.

Chapter 2. State of the Art

2.5 IoT as a Service

The Internet of Things presupposes the interaction between innumerable, physical world objects and their virtual counterparts, allowing these objects to capture and send information to the Internet. Or as defined by (Lake, Rayes, & Morrow, 2012), “the Internet of Things (IoT) consists of networks of sensors attached to objects and communications devices, providing data that can be analysed and used to initiate automated actions”. In order to integrate all these physical objects with the digital world, there is necessary to find a way to make them accessible in the Internet, and therefore an Internet of Things as a Service approach, is necessary.

2.5.1 Model

The first step to making the “Things” from the Internet of Things, accessible is the creation of a model through which, the physical “things”, can be found and accessed. According to (Bauer, Martinbauerneclabeu, & Meissner, 2011), the research, so far developed, in this area has focused mainly on “sensor descriptions and observation data modelling” that offer sensor measurement data services on the web. The SENSEI Project (Herault & Presser, 2008), for instances (a project created with the intention of developing a framework of universal service interfaces for Wireless Sensor and Actuator Networks (WSANs)) made use of a resource model to capture resource functionalities and discover where and how they could be accessed. It would then publish that information in a repository, where it could be accessed by specific ontologies.

In the SENSEI Project, like mentioned before, the core modelling concept is the “resource”, which implies that all sensors, actuators, processors, etc. are there modelled as resources. Let us take as an example of a model, the IoT model used in this SENSEI Project context depicted in Figure 2.9.

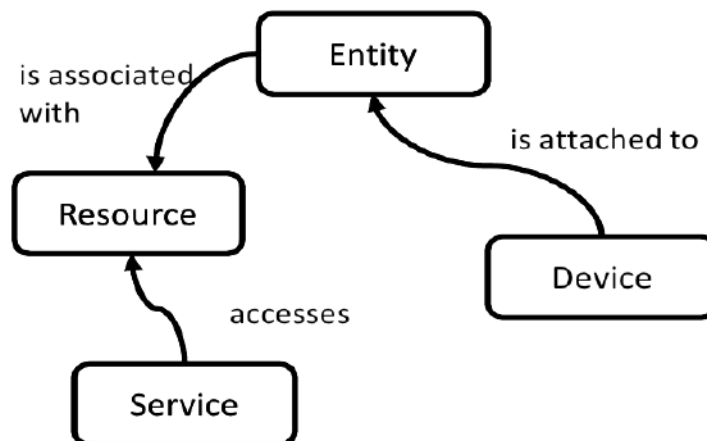


Figure 2.9 - IoT model: key concepts and interactions. Adapted from (Bauer et al., 2011)

In Figure 2.9, it is easy to see how an Entity, which has a Device attached, here associated with a Resource (a norm taken by the SENSEI Project), can be accessed through a service provided in the Internet.

2.5.2 Framework

Step two, in the Internet of Things as a Service paradigm is the creation and usage of a Framework. In order to make the paradigm accessible to everyone, no matter if they are a big industry or a small anonymous people, and to make sure all will benefit from it, an open service framework needs to be created. Once this open service framework is created and available to everyone it can bring many advantages to everyone's lives, service providers, device developers, software developers, consumers, etc.

According to (Kim & Lee, 2014), even though there have already been made some IoT Frameworks, those were mainly created by big enterprises such as governments or companies and are mostly based on B2B (Business to Business) and B2G (Business to Government) business models. However, there are also open service platforms for IoT, of which the best-known example is Cosm, former Pachube, and recently purchased by cloud computing service provider LogMeIn and called Xively (xively by LogMeIn). This IoT PaaS (Platform as a Service), allows developers and companies to connect IoT devices and Apps to

Chapter 2. State of the Art

securely store and exchange data. Through Web-based registration service it allowed users to control, monitor, and analyse data collected from IoT devices, besides giving them a way to search and find those devices (Kim & Lee, 2014). Another open service framework currently deployed is EVERYTHING that, as described by (Kim & Lee, 2014), grants every physical thing an Active Digital Identity (ADI), and provides to the device and software developers, all the necessary tools to create new services and applications to the everyday items and devices (EVERYTHING, n.d.).

In (Kim & Lee, 2014), an open service framework for IoT is presented where a developer-oriented structure is used to encourage developers to create and make available App/Web Software and services to the users. On the users side a quick search for IoT services and devices is offered, and when they intend to connect to the searched IoT device, the software related to that device is downloaded to their smartphone or tablet or etc. and its provided service is made available. The architecture used in this open service framework is depicted in Figure 2.10.

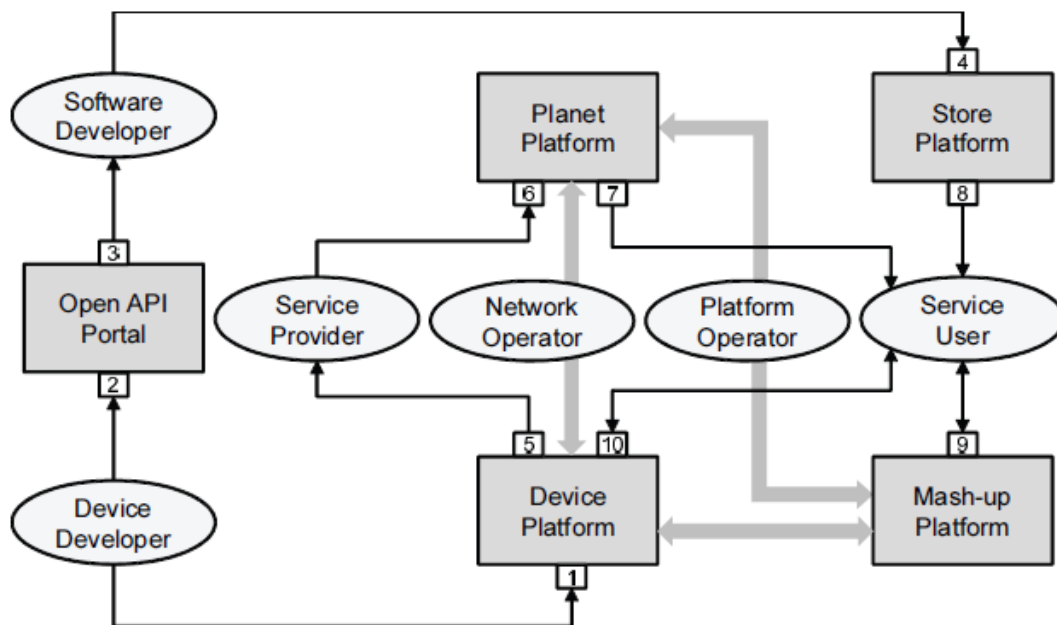


Figure 2.10 - OpenIoT framework. Adapted from (Kim & Lee, 2014)

At the framework presented in Figure 2.10, an open API (Application Programming Interface) is used, as well as a Planet Platform (described as “a server platform for IoT device registration, management, monitoring, and search in the global IoT environment”), a Device Platform (software platform “to help connecting and cooperating things to Open IoT platforms and application software”), a Store Platform (“App/Web store containing applications or links to Web address that provide user services through interaction between IoT devices or Mashup Platforms”) and finally a Mashup Platform (“service platform for providing new integrated services based on data sets collected from IoT devices and its mashup of information over the Internet”) (Kim & Lee, 2014).

2.5.3 Mashup

Also needed to allow further development of the IoT model or framework is a Mashup. The idea behind a “Mashup” is to create new content by reusing or recombining previous existing content from various sources thus allowing people who do not master all programming languages to easily build new Web Applications, or others, by providing some easy-to-use functionalities. A Mashup is, therefore, a way to compose a new service from existing services and, “when applied directly to the Web domain, a Mashup is a Web-based application that is created by combining and processing on-line third party resources that contribute with data, presentation or functionality” (Koschmider, Torres, & Pelechano, 2009).

Unlike regular Web Services that are provided in a specific domain and available at any time in the web, IoT devices, providing a service, are not always available and not always working in the same place. In addition, it is also important to realize that, when allied with the IoT, the number and variety of connected devices will be vast, and because of the fact that they will probably be producing real-time streaming data, the necessary computation power requested to the Mashup will be huge. This leads to the conclusion that, maybe, the “physical web mashups” (mixing real-world devices with virtual services on the Web) used nowadays, may not be enough to be used alongside the IoT (Guinard, 2010). An alternative solution is presented in (Im, Kim, & Kim, 2013),

Chapter 2. State of the Art

where an IoTaaS (IoT Mashup as a Service) is introduced.

According to (Im et al., 2013), IoTaaS is defined as “a mashup of things, software, and computation resource”, and is presented as a cloud-based IoT mashup service model. In IoTaaS, thing is described as “any identifiable object which can have sensing and actuation services; software is an “assembly description of software components”; and computation resource is “a current computer model consisting of CPU, memory, disk, persistent storage, network, etc. How these components interact with the IoT world is depicted in Figure 2.11.

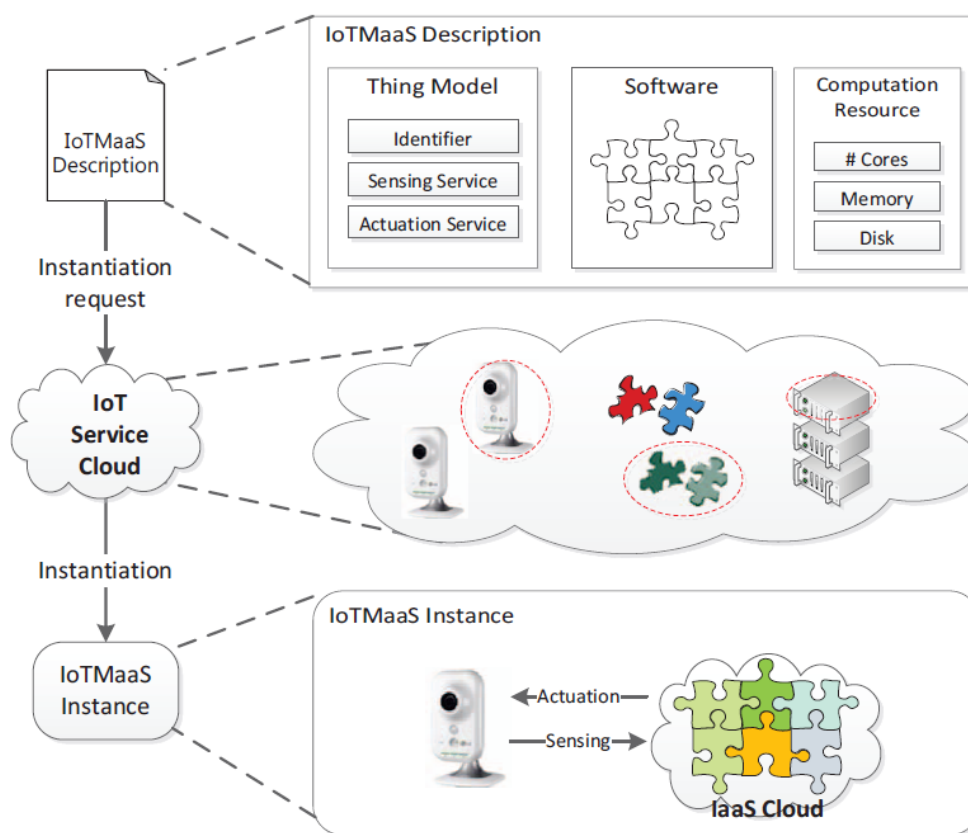


Figure 2.11 - IoTaaS (IoT Mashup as a Service) concept. Adapted from (Im et al., 2013)

When applied to the Internet of Things, the IoTaaS, whose concept is depicted in Figure 2.11, allows every IoT device to provide its service disregarding which platform it works on or which protocol it uses to communicate, because regardless of this vast heterogeneity of devices they are all treated equally by the IoTaaS. Therefore, users and or developers can make use of all their IoT devices and enhance their functions resorting to this Mashup.

2.6 From the State of the Art to this Thesis concept

As stated in this thesis' State of the Art, the kernel is the central part of a computer operating system and serves, among many other things, as a bridge between hardware devices and the software applications accessing those devices, whether they are I/O devices, plugged devices or IoT devices. The applications developed during this thesis' elaboration will make use of IoT devices and the services provided by them to offer contributions to the users and to the vf-OS project itself. Therefore it is important to understand the how the interaction between those devices and the vf-OS Manufacturing Operating System will happen, and that interaction happens through the system kernel.

Both the vf-OS project and this thesis' scenario in particular, purpose is to provide some contributions to the development of the factories of the future. Even though this thesis' applications were developed primarily to the agriculture scenario, with minor changes they can be used at any other strand included in the Factories of the Future paradigm.

In order to apply the perks of the IoT to this Factories of the Future world, there are many ways to interconnect the physical world objects and the software systems through a service oriented approach. Some ways are presented in this thesis' State of the Art, such as the usage of Web Services to offer online, the services provided by a certain IoT device. In addition to the Web Services, another approach to create an IoT service oriented system, is the usage of cloud computing to allow the IoT devices functions to be made available online. Some Generic Enablers provided by the European FIWARE make use of a cloud computing system to make available online the functions provided by IoT sensors.

After briefly going through all these essential aspects, the environment under which this thesis is developed is fully contextualized (the Kernel where the vf-OS that includes this thesis' project aims to provide solutions, the Factories of the Future where this project is to be applied, Web Services for Industrial Internet of Things as an alternative way to implement the IoT Service Oriented System, the FIWARE which provides the technology used during this project development and the IoT as a Service which is the main focus of this thesis' project).

Chapter 2. State of the Art

2.7 Research Question and Hypothesis

This master thesis project aims to show and provide a way to encompass physical objects working under an IoT approach to be used in a Service Oriented System. In other words, this thesis intends to show a way of how it is possible to make available online, the services and functions provided by IoT devices, or as its title states, present a Framework for IoT Service Oriented Systems.

In order to further clarify the goal of his master thesis project a Research Question and a Hypothesis were created with the help of the UNINOVA developers of the vf-OS Project.

After analysing what was already available in the market and which were the vf-OS needs, the Research Question found to guide the thesis development according to the market and the vf-OS needs was:

“How can a framework provide guidance to make IoT services discovered for effective use?”

To answer this question, and once again to guide the thesis development and answer the market and the vf-OS needs, a Hypothesis was generated. The produced Hypothesis intends to cover not only the thesis main objective, which is the implementation of an IoT service oriented system, but also the technology used for this purpose and its advantages for this goal. Thus, the produced Hypothesis was:

“If the FIWARE technology can provide modularity and discovery solutions then integrate IoT devices through generic enablers will facilitate IoT service oriented implementation and use on manufacturing systems.”

3 Practical Framework

In this chapter, is presented the practical framework under which this master thesis is developed. This master thesis is developed under the European vf-OS Project and is supervised by the UNINOVA institute. The main goal of this thesis is to provide small contributions to the vf-OS Project, explained in this chapter. In order to provide such contributions, the Use Case Scenarios presented in the projected were addressed and a real-life scenario was created. Combining the created scenario and the generic Use Case Scenarios, five interconnected applications were developed in order to provide answers to both the Use Case Scenarios and the practical scenario at the same time.

In addition to the project which holds this master thesis development, the Use Case Scenarios addressed and real-life scenario created, also the technologies used to develop the applications are presented in this chapter. Among the used technologies are the FIWARE Generic Enablers. The usage of these open source European technologies allows the vf-OS components, and therefore these thesis' applications, to become as generic and standard as possible allowing them to be used by anyone regardless of the FoF field on which they are using them, factories, manufacture, agriculture, etc.

Chapter 3. vf-OS (virtual factory - Operating System)

3.1 vf-OS (virtual factory - Operating System)

Like previously mentioned, the European vf-OS Project aims to be an Open Virtual Factories Operating System, including a Virtual Factory System Kernel (vf-SK), a Virtual Factory Application Programming Interface (vf-API) and a Virtual Factory Middleware (vf-MW) specifically designed for the factories of the future. This Open Framework will be able to manage the overall network of a collaborative manufacturing and logistics environment, and therefore enable humans, applications and devices to communicate and interoperate in the interconnected operative environment. Plus, the vf-OS will provide a set of Open Services, rooted in the cloud and instantiated at the vf-OS Platform, moving the industry from the device-centric to the user-centric paradigm. This Open Platform is to be linked by strong and advanced ICT (like CPS, IoT Cloud-Models, M2M, etc.) in order to fulfil the actual need on the market for open services interoperability based on data exchange. When it comes to hardware functions, the OS will act as an intermediary between the applications behaviours of the factory and the factory hardware itself. This will enable the application factory functionalities and services to be virtualized and executed, either directly by the hardware either through system calls to the OS.

The vf-OS, deployed in a cloud platform provide a range of services to the connected factory of the future to integrate better manufacturing and logistics processes. In order to do so, the vf-OS intends to not only create new technologies but also greatly re-use existing tools (especially Open Source ones) and technologies (especially standardised ones). Like a regular OS, the vf-OS comprehends core functionalities, but mainly focused in a manufacturing environment. An analogy between some components of a regular Software OS and the vf-OS Manufacturing OS is presented in Table 3.1 (adapted from the vf-OS technical sheet).

3.1 vf-OS (virtual factory - Operating System)

Table 3.1 - Analogy between Software OS and vf-OS Manufacturing OS.

Software Operating System	Virtual Factory - Operating System (vf-OS)
Kernel	Virtual Factory System Kernel (vf-SK)
Processor, Memory, Internal Bus	Framework, Generic Enablers, Manufacturing Enablers
I/O	Virtual Factory Application Programming Interface (vf-API)
Interfaces, Peripherals, Device Drivers, APIs	Devices Drivers, APIs Connectors, Security & Data Access
File and Data handling	Virtual Factory Middleware (vf-MW)
Interfaces	Data Infrastructure Middleware, Data Storage, Data Harmonisation, Data Analytics

Chapter 3. Use Case Scenarios

3.2 Use Case Scenarios

As described before, the vf-OS intends to be the reference system software for managing industry related computer hardware and software resources and providing common services for industry computational programs. This operating system will be the component of the system software in a real industry system where all industry application programs will run. To accomplish this, some generic case scenarios have been established which can be addressed with the vf-OS Platform and its Smart Applications.

The current industrial environment faces some problems related to specific industries and respective sectors. The Use Case Scenarios identified in the vf-OS project intend to create various applications that meet the users' needs in order to solve some of the identified problems. The existing scenarios and the solutions presented are described to a good extent by following a common methodology for all the industrial sectors.

The developed scenarios integrate both industrial and user scenarios as they propose to produce advanced technical solutions to some of the existing industrial scenarios by developing suitable applications, whereas the user scenarios are described following a standard methodology through well-defined objectives, processes, actuators and possible sets of data. The scenarios addressed by the vf-OS project capture the needs of different industrial sectors and process domains as well as providing the guidelines for applications to be developed in order to meet these needs.

Table 3.2, shows the generic use case scenarios developed by the vf-OS project as well as a brief description of the scenario and the relationship between the solutions' application and the market needs and expectations.

Table 3.2 – vf-OS Generic Use Case Scenarios.

Name	Description
vOrder	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 to returning process by clients.
vProductMon	Real-time monitoring on the status of a production, having the possibility to identify flaws and inform production managers that can immediately react.
vfSalesLead	Help salesperson to identify sales leads in his region, and segment territories into employee count, competitor, and location.
vfColPlan	Provide tools to compute collaborative plans.
vfNegDemand	Visualize and negotiate demand plans in real time. The application connects to production plan 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	Monitor consumption of phytosanitary products in agricultural productions to support demand management, taking into account quality requirements for the final product.
vfHarvest	Optimize the harvest process integrating data from crops, logistic and manufacturing process to optimize resource utilization and final product quality.
vfFail	IoT application for the automatic registration of spare-part failures in automation production equipment.
vfPayment	Allow making payments after a negotiation process by integrating different payment gateways.

Chapter 3. Use Case Scenarios

vfMyCon	Interface with smart meters and provide energy consumptions and saving strategies.
vfColPurchase	Provide option for collaborative purchase (reduction on logistic costs and better deals with suppliers).
vf3DViewer	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.

3.3 Practical Scenario

As said before the Factories of the Future comprise not only factories for themselves but also other strands, such as the manufacture, the agriculture, etc. In order to show the vf-OS wide range of applications and also to provide solutions to a gap found in the smart agriculture environment, an agriculture scenario was created. Through the usage of computers, IoT devices, cloud and smart applications in the agriculture world, an automatization of procedures can be achieved and that is what this thesis applied to this scenario proposes to do.

The scenario consists of an applications-assisted fruit production chain, covering every step, i.e. every step of the food chain is meant to be monitored and controlled by IoT devices accessible through applications.

The process begins with the harvesting of various types of fruits by several different farmers, in fields monitored by IoT devices (sensors, actuators, etc.). The collected fruit is then manually or automatically split according to different factors (type, size, weight, brand, quality, etc.), and stored in boxes grouped by the mentioned factors. When a buyer seeks to buy a specific type of fruit with specific features, makes use of a buying application where he can find the sellers of the fruit with those specifications and from where he can emit an order to acquire the fruits taking into account the price of the fruit, the seller, the quality etc.. Once the buying order reaches the producer, the latter dispatches the fruit using trucks equipped with IoT devices (sensors) in order to ensure the quality of the transported fruit. The producer, while dispatching his fruits is able to control, in real time, all the steps taken by the fruit since the its harvest to the time the fruit is delivered to the buyer. Besides, he is always aware of any faults or failures occurring during the process as well as keeping track of the amount of fruit that is being harvested, shipped, or removed due to not being in conformity with the quality standards.

In Figure 3.1 is presented an illustration of the described scenario.

Chapter 3. Practical Scenario

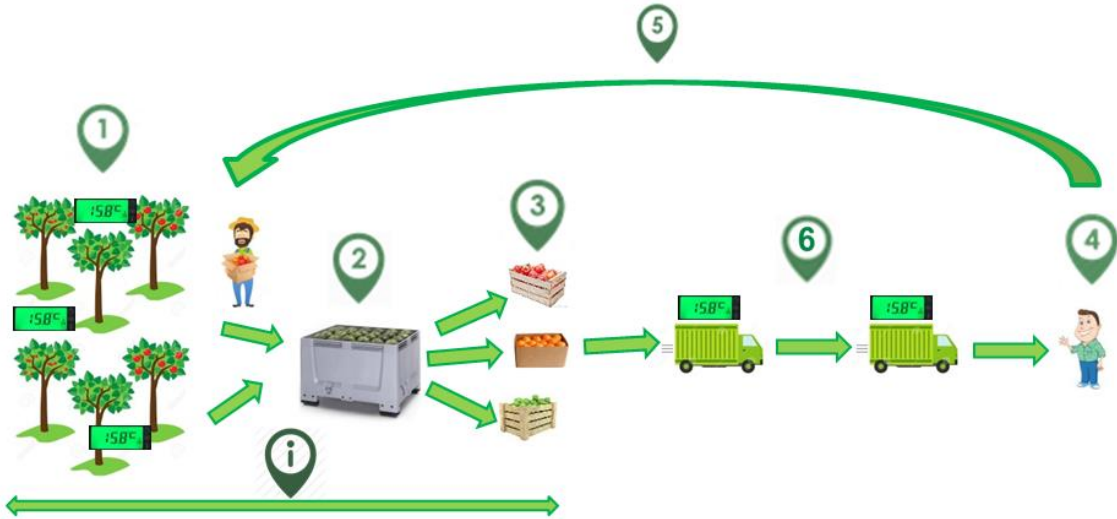


Figure 3.1 - Practical Scenario Illustration

All the steps enumerated in Figure 3.1 can be found listed and summarily explained in Table 3.3.

Table 3.3 - Scenario Steps Explanation.

Steps	Explanation
1	Fruit harvest, controlled by IoT Devices
2	Fruit selection, separation (manually or automatically by IoT devices)
3	Farmer automatic track of things through IoT devices (amount and quality of fruit, faults and failures, spare boxes, etc.)
4	Buyer selection of type of fruit, size and farmer
5	Buying order
6	Monitoring transport to ensure success delivery (through IoT sensors)
i	Monitoring production and checking for failures

3.4 From the Scenarios to the Applications

After aligning the given generic Use Case Scenarios with the created real-life practical scenario, five interconnected applications were found that making use of the described Use Case Scenarios functionalities could provide all the tools required to implement the scenario. Those five applications were divided between the work group of students doing their Master Thesis in the vf-OS Project, and to each of them a more concise summary was made, always having in mind the generic use of the source Use Case Scenario but now addressing the practical scenario in particular.

➤ **vfHarvest:**

To have a productive process, sensors can be used in all phases of the production to know if it is going according to the plan. The machines can also have sensors attached to them, to know if they are working correctly.

All productions want to have the most efficient production, without changing its quality, to have more profit on each product.

One application of vfHarvest is having sensors in different fields and on the production lines. If the sensors have different values, then something is not correct, and for this reason the production will not be efficient.

➤ **vOrder:**

A platform accessible to the farmers who shipped goods and the buyers who bought them, allowing both of them to control and verify the transport conditions and the safe arrival of the products.

Platform also accessible to the transport providers for them to be able to create and add trucks to a specific farmer's fleet.

➤ **vFail:**

The spare parts take some time to be delivered in the factory, and accordingly to the stages of the spare parts supply chain: demand forecast, planning, supply, manufacturing, distribution, storage, and replacement, it should exist one application to forecast the spare parts demand.

Chapter 3. From the Scenarios to the Applications

This problem is also affected by the non-visibility of equipment operation indicators and status.

When the production is stopped due to the lack of spare-parts, its production decreases and it can stop.

To packing the products, there are always present packing boxes. Some are used by real-time production, and others are stored, which will be used in a near-future. There must be an application that knows when to order new packing boxes.

➤ **vProductMon:**

Only check the current status and the expected status. If the result is different, then notify the stakeholders.

In agricultural scenario, it can be used in packing fruits, the shipment must contain the expected packages and correct weight and size. If neither of this is correct, the seller must be alerted and the buyer must know that don't have the expected merchandise.

➤ **vfNegotiation:**

The buyer must choose, between all its resellers, the most desirable one, according to different factors.

In most of the interactions, between one agent who produces the goods and the other agent who buys the goods, the purpose of the negotiation is to make the better negotiation choice, between several agents. There must be a list of agents to choose and one algorithm to choose one of them.

In Figure 3.2 are presented all the five interconnected applications built from the Use Case Scenarios and that together answer the real-life scenario, and the connections that exist between them

3.4 From the Scenarios to the Applications

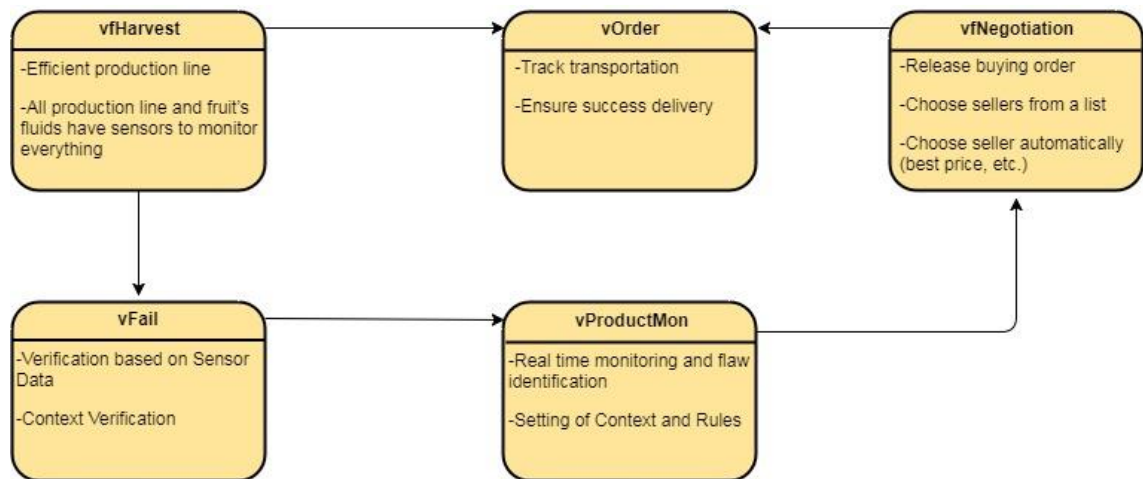


Figure 3.2 – Use Case Scenarios Applications applied to the Practical Scenario

This thesis in particular, addresses the solution of the **vOrder** and the **vfNegotiation** applications.

Chapter 3. Technologies

3.5 Technologies

As presented in the State of the Art of this Master Thesis there are several different ways to handle IoT devices and the services they provide. The most commonly used are the Web Services, that allow an IoT device to provide his services over the Internet. However, for the purpose of this thesis a different technology was used. The FIWARE program owned by the European Union has many open source Generic Enablers accessible in their website, and since this thesis falls under a project being developed for the European Union it makes all sense to use this FIWARE technology during the development of the vf-OS project.

For the development of the first application on which this thesis focuses, the **vOrder**, two FIWARE GEs were used:

1. FIWARE Publish/Subscribe Context Broker Generic Enabler - Orion Context Broker;
2. FIWARE Short Time Historic (STH) - Comet.

For the development of the second application, **vfNegotiation**, only one FIWARE GE was used:

1. FIWARE Publish/Subscribe Context Broker Generic Enabler - Orion Context Broker.

Before addressing these GEs however, it is important to mention another technology that made possible and easier the usage of both this FIWARE GEs, the **Docker**.

3.5.1 Docker

Docker is a container platform software created by the Docker, Inc. company, i.e. it is an open source project that allows the deployment and usage of applications from within software containers. Using these containers, it becomes much easier to create, deploy, and run applications from any computer using Docker without any need to install further applications and also insuring that the software will always run the same, regardless of where it is deployed or how many co-workers are working on it at the same time, from different platforms.

A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings (Docker, 2017). A container is therefore a place where a developer can package up an application with all of the parts it needs to run, and ship it all out as one package. Containers are in many ways similar to a Virtual Machine, with the fundamental difference being that unlike VMs, containers do not bundle a full operating system, only libraries and settings required to make the software work. Containers provide an additional layer of abstraction, which allows multiple containers to run on the same machine within the same OS kernel making use of isolated processes in the user space.

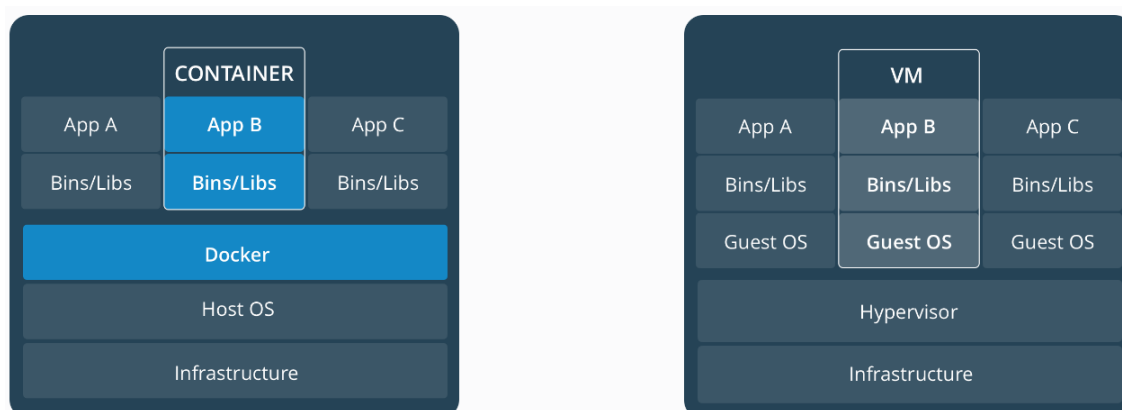


Figure 3.3 – Container vs Virtual Machine comparison (Docker, 2017)

Chapter 3. Technologies

By using Docker, it has been made possible to use either the FIWARE GEs without having to install or configure the enablers or any other programs they would need, such as the databases and others.

As previously mentioned all the FIWARE GEs are open source and can be acquired through the GitHub platform and have their own Docker container there, accessible to everyone.

3.5.2 FIWARE Orion Context Broker

The Orion Context Broker is an implementation of the Publish/Subscribe Context Broker GE used to develop a Data/Context Scenario through the NGSI9 and NGSI10 interfaces (Next Generation Services Interface).

In order to deal to physical devices through an Internet approach i.e. to create the so called IoT devices, FIWARE associated with OMA (Open Mobile Alliance) to create the NGSI concept which enables a transition from device-level information to Thing-level information and vice versa. In order to accomplish so, two interfaces were created NGSI9 and NGSI10. Both are RESTful APIs via HTTP but with slight differences. While NGSI10 purpose is to exchange context information, the NGSI9 is used to exchange information about the availability of context information.

NGSI10 technology has three main interaction types, which are (NGSI10, 2014):

1. one- time queries for context information;
2. subscriptions for context information updates (and the corresponding notifications);
3. unsolicited updates (invoked by context providers).

All the functionalities that the NGSI10 interface processes can be seen in the resource tree presented in Figure 3.4.

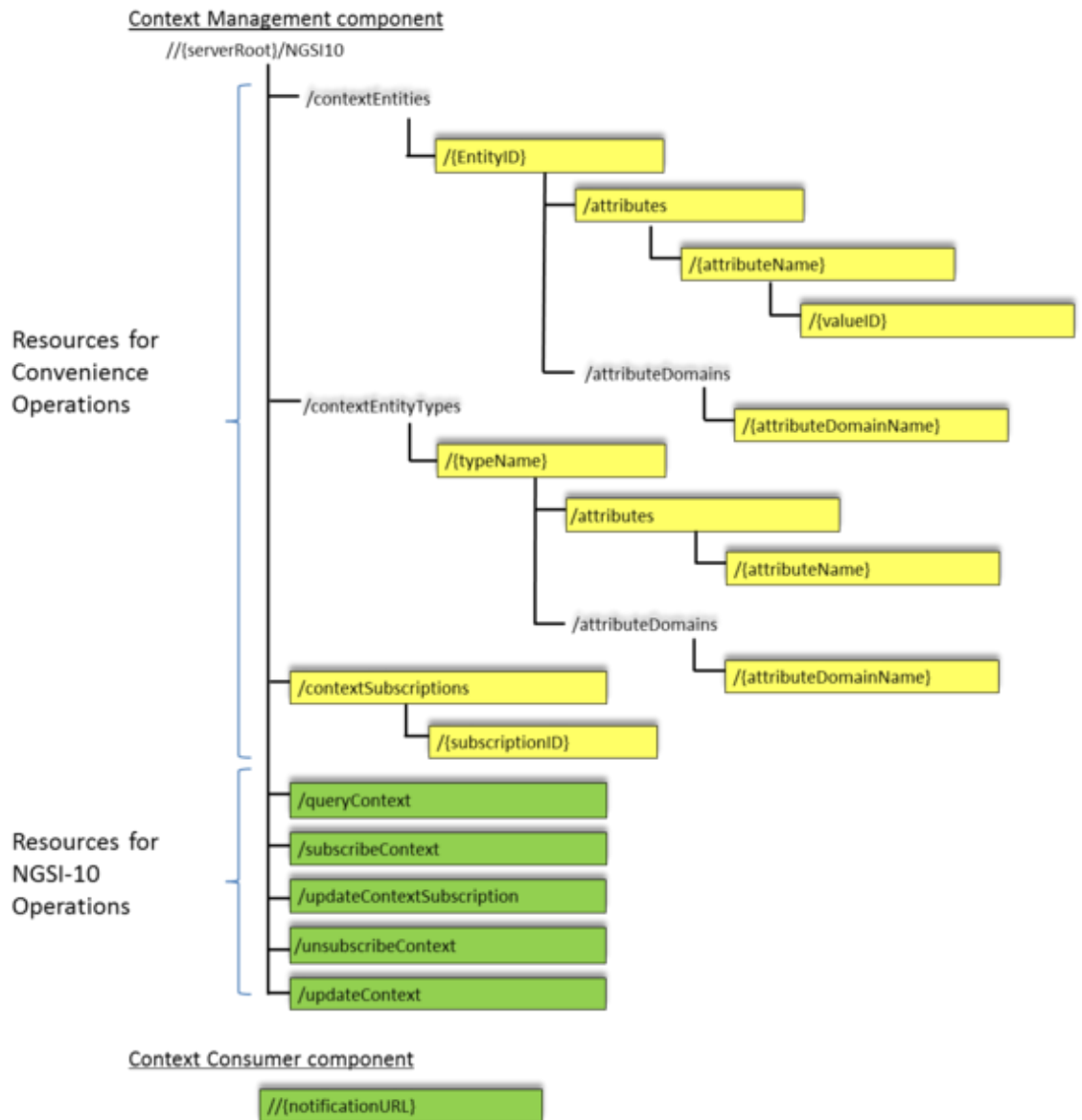


Figure 3.4 - NGSI10 Schema of REST resources (NGSI10, 2014)

Coloured yellow on both figures (Figure 3.4 and Figure 3.5) are listed the common functionalities, shared by the NGSI10 and NGSI9. Coloured green are the functionalities specific to the NGSI10 that permit the exchange of context information, allowing interaction via HTTP 'POST'. These NGSI10 specific functionalities can be found listed and explained in Table 3.4.

Chapter 3. Technologies

Table 3.4 - NGSI10 specific functions.

NGSI10 Operation	Explanation
/queryContext	Retrieve information (through a query) from an entity
/subscribeContext	Ask to be informed every time an attribute of the entity changes
/updateContextSubscription	Modify the change that needs to happen to an attribute in order to be notified
/unsubscribeContext	Delete the need to be notified on change (on that entity)
/updateContext	Update some information present in the entity

When it comes to the NGSI9 technology, it has its own three main interaction types, which are (NGSI9, 2014):

1. one- time queries for discovering hosts, where certain context information is available;
2. subscriptions for context availability information updates (and the corresponding notifications);
3. registration of context information, i.e. announcements that certain context information is available (invoked by context providers).

All the functionalities that the NGSI9 interface processes can be seen in the resource tree presented in Figure 3.5.

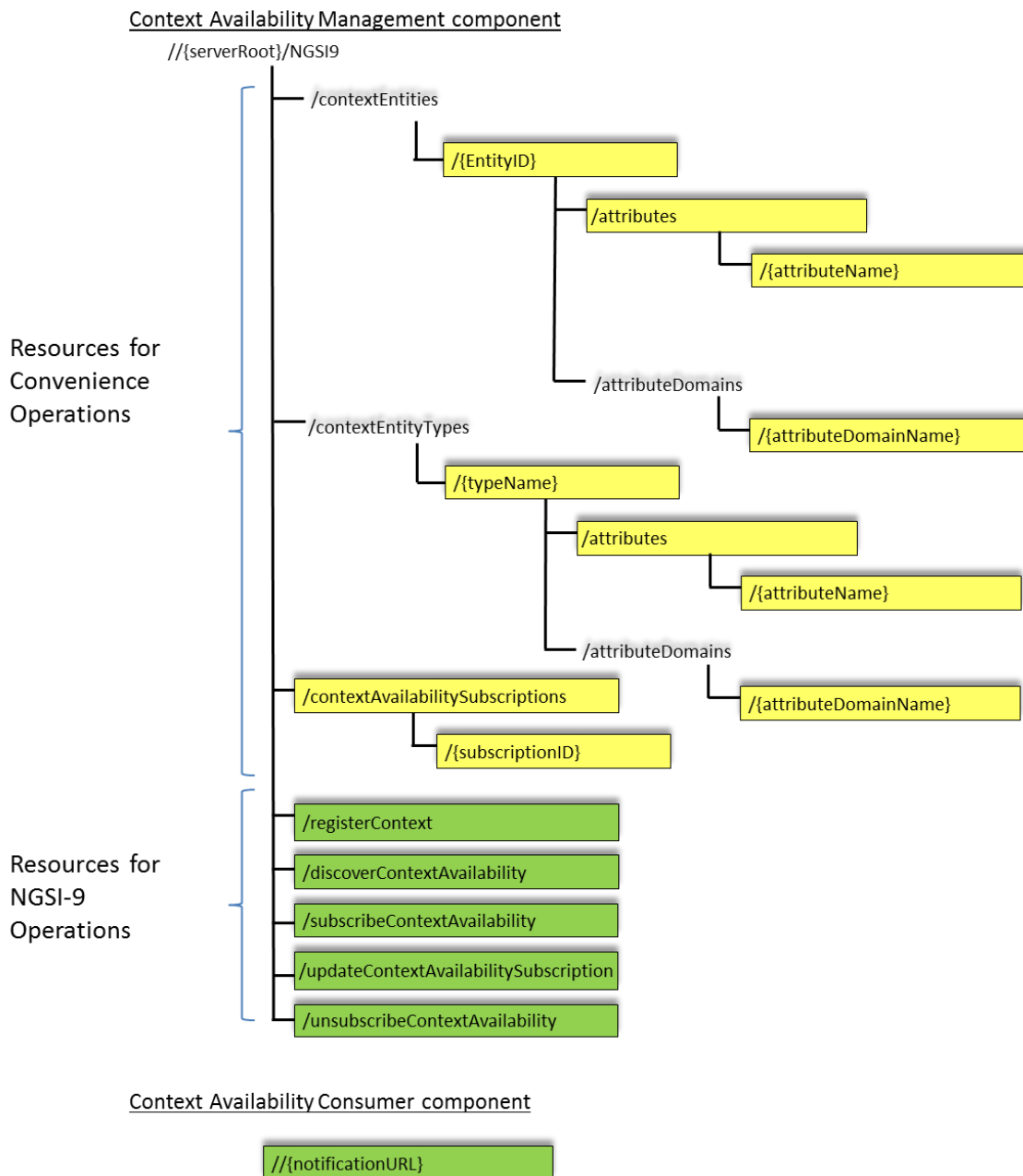


Figure 3.5 - NGSI9 Schema of REST resources (NGSI9, 2014)

Once again coloured yellow are listed the common functionalities, shared by the NGSI10 and NGSI9. Coloured green are the functionalities specific to the NGSI9 that permit the exchange of context information, allowing interaction via HTTP 'POST'. These NGSI9 specific functionalities can be found listed and explained in Table 3.5.

Chapter 3. Technologies

Table 3.5 - NGSI9 specific functions.

NGSI10 Operation	Explanation
/registerContext	Register an entity
/discoverContextAvailability	Discover if a given entity exists
/subscribeContextAvailability	Ask to be informed whenever a given entity begins or ceases to be available
/updateContextAvailabilitySubscription	Change the reason why the above-mentioned information should arrive
/unsubscribeContextAvailability	Delete the need to be notified on an entity availability

A NGSI context element is a representation of an entity through a data structure. This context element is where all the information about an entity will be stored. The information that makes this data structure entity is kept in specific fields within the structure, as shown in Table 3.6.

Table 3.6 - NGS19/NGSI10 Context Element (Entity) structure.

Name		Information
Type		Type of Entity (Room, Car, etc.)
ID		Entity Name (Room1, Car2, etc.)
Attribute	Name	Name of the attribute (Temperature, Humidity, etc.)
	Type	Type of the value of the attribute (Integer, Float, etc.)
	Value	Value of the attribute
	Metadata	Information specifically related to that entity's attribute.
Metadata	Name	Name given to the metadata field.
	Type	Type metadata field
	Value	Metadata information/value

As happens with the entity ID field, also the attribute Name field must be unique, i.e. each entity cannot have two attributes with the same Name. However, the FIWARE Orion allows the developer to, in case of wanting to have two attributes sharing the same name (for example a truck who has two temperature sensors, one in the front and one in the back), create two instances of the attribute, with the same name, as long as a metadata for each attribute is created with the Name (metadata name field) filled as "ID". Using this "trick" the developer can have an entity with two attributes with the same name where each of the attributes have a Metadata with the name "ID" and different values (metadata value field).

The information presented in Table 3.6, can be seen simply schematized in Figure 3.6.

Chapter 3. Technologies

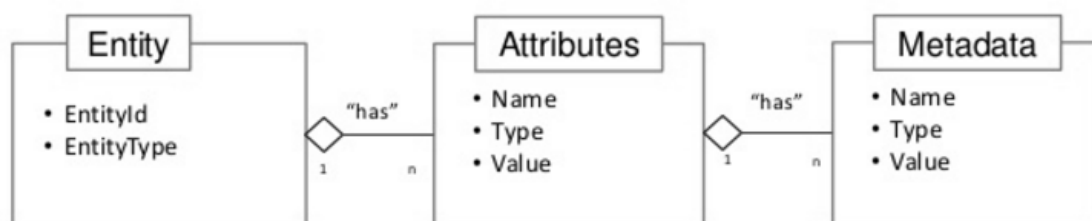


Figure 3.6 - NGSI9/NGSI10 Context Element (Entity) schematized structure

With the information presented in Table 3.6 and schematized in Figure 3.6, it is easier to better understand the main difference between NGSI10 and NGSI9. While NGSI10 is used to exchange information about the entities themselves (their attributes or metadata values), the NGSI9 is used for availability information about the entities and their attributes. Here, instead of exchanging attribute values, information about which entity can provide a certain attribute value is exchanged.

Using both the described NGSI interfaces the FIWARE Orion Context Broker makes possible the creation of entities with all the listed information. Making it also possible to update that information and to subscribe to that information (be informed when a change has occurred). The FIWARE Orion Context Broker serves than an intermediate crossing point between the Context Producers and the Context Consumers, storing all the information in a Database (MongoDB is the database used by Orion by default) which can be accessed by both the Producers (to update or generate information) and the Consumers (to query the entities or be notified through the subscriptions), as depicted in Figure 3.7.

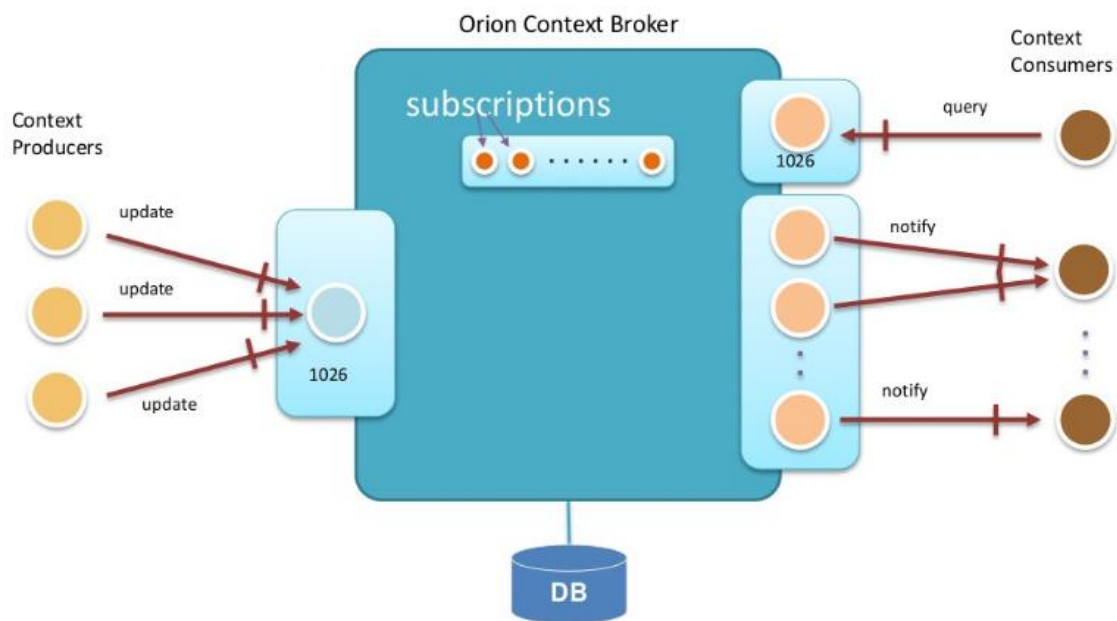


Figure 3.7 – Orion Context Broker in a nutshell (FIWARE - Orion Context Broker, 2014)

Even though the FIWARE Orion Context Broker allows to keep track of an entity's attribute's value through queries, or to be informed every time that attribute's value changes and to what it changed into, Orion does not come with a way of keeping history of the different changes a certain value has changed into. In order to do that the FIWARE Short Time Historic (STH) – Comet was used.

3.5.3 FIWARE Short Time Historic (STH) – Comet

The Short Time Historic is a FIWARE component capable of managing (storing and retrieving) historical raw and aggregated time series information evolution in time of context data (i.e., entity attribute values) registered in an Orion Context Broker instance. Like happened in the FIWARE Orion Context Broker, all the communications performed by the FIWARE STH make use of the NGSI9 and NGSI10 interfaces.

As stated, even though the FIWARE STH supports the storing and retrieval of raw context information, (the concrete entity attribute value which were registered in an Orion Context Broker instance over time), its main capability

Chapter 3. Technologies

and responsibility is the generation of aggregated time series context information about the evolution in time of those entity's attribute values.

In order to keep track of the different values a certain attribute takes over time, the STH makes use of the Orion Context Broker subscription function, so that every time that attribute's value changes, the STH can be informed that the change occurred and to which value the attribute has changed into. With this information arriving every time a change of values occurs, the STH keeps a history of the changes in a database (MongoDB by default, same DB that the Orion Context Broker uses).

Using an HTTP RESTful API, external clients can query the available historical raw context information maintained by the FIWARE STH. By doing so the consumers get a list of the different values that attribute has taken over time and at what time the changes occurred. These results can be filtered to be shown, for example, the changes that occurred in the last hour or day, or the 10 last changes that happened, or from a specific date to another specific date, etc.

On the other hand, if it is historical aggregated time series context information that the consumer is after, the FIWARE STH, also through an HTTP RESTful API, has many ways of interpreting the information and returning it through many different aggregations. The aggregation methods that the FIWARE STH provides can be found listed in Table 3.7.

Table 3.7 - FIWARE STH historical aggregation methods.

Type of attribute	Aggregation method	Explanation
Numeric	max	Maximum value
	min	Minimum value
	sum	Sum of all the samples
	sum2	Sum of the square value of all the samples
String	occur	Occurrences of each textual value that attribute have had over time

It is worth noticing that by combining the information provided by these aggregation methods with the number of samples, it is possible to calculate probabilistic values such as the average value, the variance as well as the standard deviation, etc.

Finally, the same way that in the raw context information history the values can be filtered, also here in the aggregated time series context information, the queried values can be filtered by time (last hour, last day, n last changes, specific time interval, etc.).

Using all these technologies combined it is possible to store the data produced by the Context Producers (the IoT devices, in this thesis), and it is possible for the applications or any other 3rd party program to access either the raw context information, and the aggregated time series context information by accessing the Orion Context Broker directly or the STH databases. A schematic showing all these relationships (data producing, data storing and data querying) is depicted in Figure 3.8.

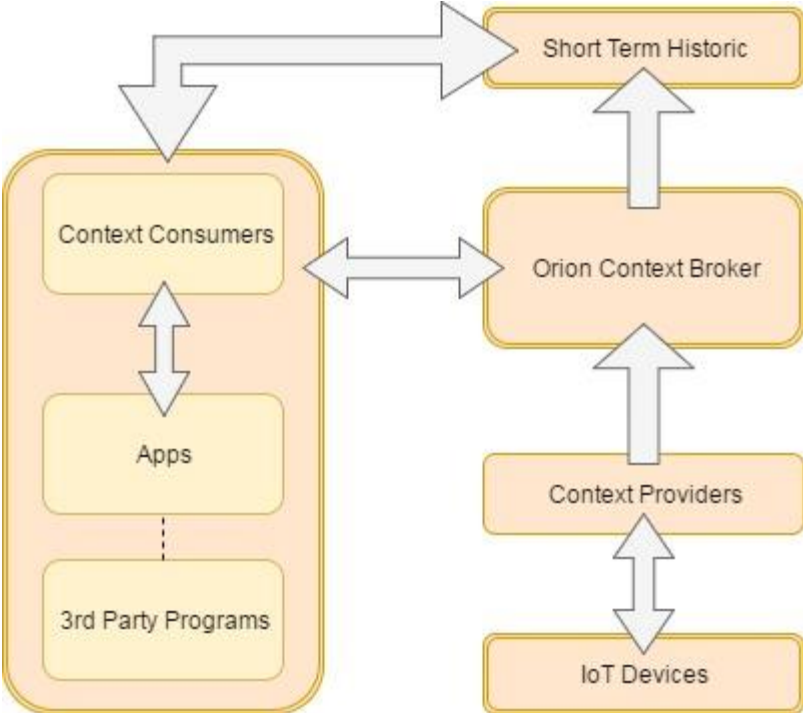


Figure 3.8 - Orion Context Broker + STH working schematic

3.6 Summary

Using the FIWARE Generic Enablers and other technologies it was possible to create the applications from the Use Case Scenarios presented in the vf-OS Project, always bearing in mind an answer to the described practical scenario.. Besides the Java interface, two European FIWARE GEs were used (FIWARE Orion Context Broker and FIWARE Short Time Historic - Comet), to develop both the applications addressed in this thesis - vOrder and vfNegotiation).

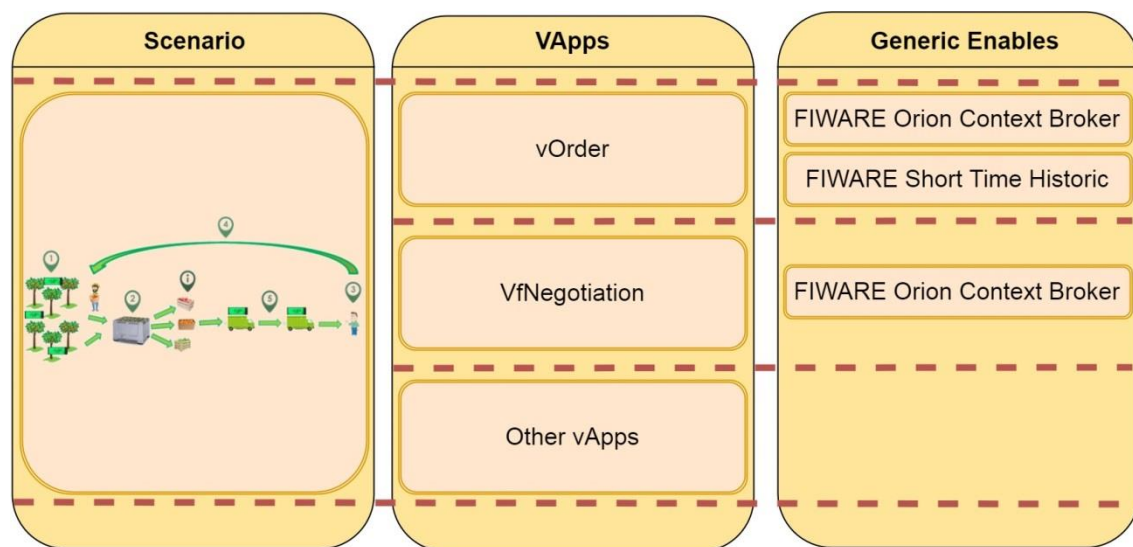


Figure 3.9 - Scenario + VApps + Generic Enablers use summary

In Figure 3.9 is depicted, in a summary way, which GEs were used in each of the developed applications and how the scenario involves all the applications. This thesis contributes, as showed and as previously mentioned, with the development of two of the five interconnected applications, the vOrder and vfNegotiation. These applications make use of some open source FIWARE Generic Enablers, always looking to provide solutions to the practical scenario, the fruit production chain, all the way from the harvest to the selling going through the distribution.

4 Developed Applications

In order to produce the automated food chain presented in the Scenario through the development of the generic Use Case Scenarios presented in the vf-OS Project, it was required the development of several different applications. Within the workgroup doing the master's thesis under the vf-OS project, it was established the development of five interconnected applications. Those applications and their relation can be seen in Figure 4.1.

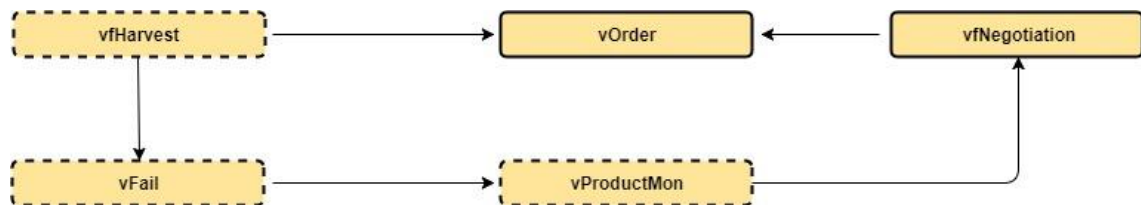


Figure 4.1 - vf-OS interconnected Apps developed within the Master Thesis workgroup

Throughout the development of this thesis in particular, two of the listed applications were developed, the **vOrder** and the **vfNegotiation**, both thoroughly described in this chapter. All the other applications were developed by other members of the workgroup.

Chapter 4. Applications Entities

4.1 Applications Entities

Before explaining the developed applications themselves, as stated in Chapter 3, the FIWARE Orion Context Broker makes use of Entities to represent IoT equipment's and other devices. In order to represent the IoT devices and other important information used in the applications development, several Entities were created during the development of the applications.

The entities created and used during this thesis development were:

Table 4.1 - List of Entities.

Entity Name	Entity Function
Fleet Entity	Contains the information about the fleet of each farmer.
Truck Entity	Represents every truck present in the applications.
Sensor Entity	Represents every sensor present in the applications.
Subscription (Truck State) Entity	Represents all the "Truck State Subscriptions" created during the applications' run.
Subscription (Sensor Values) Entity	Represents all the "Sensors Values Subscriptions" created during the applications' run.
Order Entity	Represents the different orders that will take place between a buyer and a farmer.
Farmer Entity	Represents each farmer using the applications and holds all the information about the goods they possess.
Fruit Production Entity	Serves as the link between the vfNegotiation and the vProductMon Applications.

4.1 Applications Entities

All the entities here briefly presented, can be found further explained in detail in Appendix A. There can also be found an enumeration of each entity's attributes.

With all these essential Entities thoroughly explained it's easier to understand each of the functionalities presented by the two different applications developed during the duration of this master thesis. These two applications (vOrder and vfNegotiation) will be described in the coming subchapters of this Chapter 4, Subchapter 4.2 and Subchapter 4.3 respectively. After the description of both the applications, their uses, and their relations to the other applications developed within the vf-OS Master Thesis workgroup, a demonstration of their use is presented in Subchapter 4.4.

4.2 vOrder

vOrder is an application able to be used either by a farmer wanting to expedite his products (fruits in this scenario), or a transport manager who provides the farmers with the necessary trucks to transport the sold products, or finally by a Buyer who wants to keep track of his purchased goods' transport conditions.

From the Transport Management point of view, he can add trucks to a specific farmer's fleet, by accessing the "Truck Creation" functionality of the vOrder app, and there providing the farmer to which fleet he will add a truck, and then specify the truck ID and the list of sensors that truck possesses. From the buyer point of view, he can check, at any time, the transport conditions of the goods he has purchased and that are travelling within a truck equipped with IoT sensors, by getting readings of those sensors' measurements. From the Farmer point of view he has the ability to, in addition to create and add trucks to his fleet, the same way that the Transport Provider can, and check the transport conditions of the goods being transported by a certain truck, the same way that a Buyer can, also check the available trucks to use as a transport mean, manage those trucks (change edit the truck's sensors types or IDs, as explained further ahead), and even analyse the application evaluation of the transport conditions by manually generate simulated values to the truck's sensors.

In Figure 4.2 is shown the vOrder Main Interface. It is composed by six tabs, some of them accessible only to some types of users, as explained in Subchapter 4.4 of this Chapter 4.

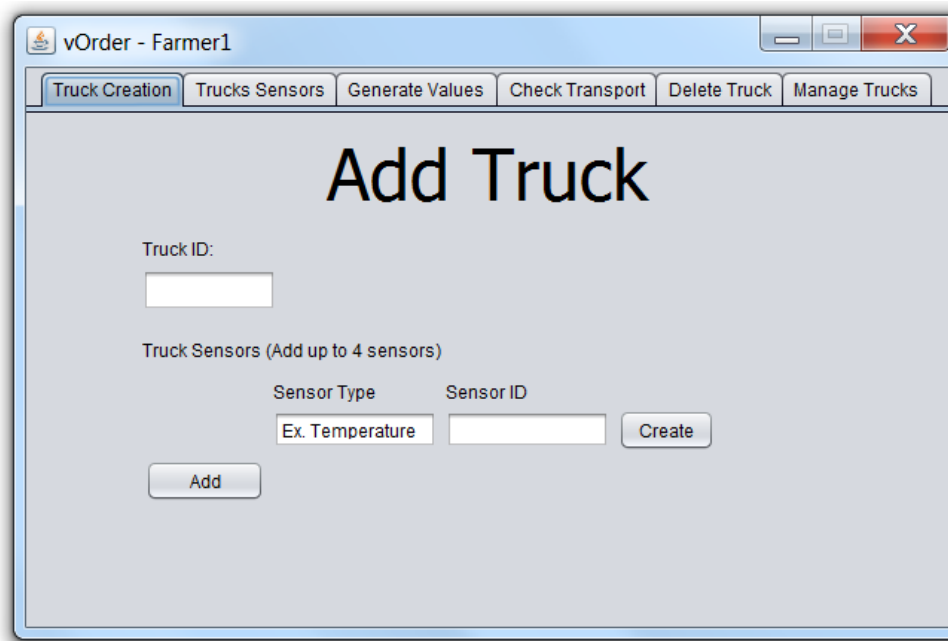


Figure 4.2 - vOrder Interface

A summary of all the vOrder application functionalities, are explained in the next paragraphs and can be found schematised in Figure 4.3.

When the vOrder Application is run the user is presented with an interface (Figure 4.8) where he can pick one out of three types of users (Transport Provider, Buyer and Farmer). Selecting either the Transport Provider or the Buyer, gives the user instant access to the vOrder Main Interface, however both of them only have privileges to access one of the application's tab each - Transport Provider reaches the "Truck Creation Tab" (Figure 4.13) and Buyer reaches the "Check Transport Tab" (Figure 4.14). A farmer user however, before reaching the vOrder Main Interface is faced with another interface where he must state which farmer he is (Figure 4.9), if a new one (in which case the application will create a new Farmer) or an existent one (in which case the application will verify if an existent Farmer ID was inserted and if so, "login" to that farmer).

When the farmer reaches the vOrder Application Main Interface, if that specific farmer has pendent Orders (i.e. orders commissioned by a buyer which still weren't expedited yet), a "Dispatch Order" Interface will appear (Figure 4.16), showing the farmer that order's intel (fruit, breed, size and ordered amount), as

Chapter 4. vOrder

well as the farmer's stock prior and after the order and the monetary value that transaction will give him. Also in that interface, the farmer has the ability to select an available truck from within his fleet to transport the goods to the buyer. Once the truck is sent, the application checks for others unintended orders and will present another Dispatch Order Interface if there are any.

The vOrder Main Interface is the core of the vOrder Application. There, it is possible the creation of trucks with sensors working from within them (Truck Creation tab - accessed either by a Farmer (Figure 4.17) or by a Transport Provider (Figure 4.23) type of user, the latter needing to state to which farmer's fleet he is adding the truck). After one or more trucks are created and added to the farmer's fleet they can be listed to the user (Existent Trucks tab (Figure 4.24) - only able to be accessed by a Farmer), and the application becomes able to receive the truck's sensors readings (which can be simulated in the Generate Values tab (Figure 4.28) - accessed only by a Farmer user). Once a truck has performed, or at least, started a transportation trip, the conditions of that trip and a qualitative evaluation of it can be checked (in the Check Transport tab - accessed by either the Farmer (Figure 4.33) or the Buyer (Figure 4.31) of the transported goods, while the first can check all of his trucks trip evaluations by choosing from a list containing all his trucks, the second can only check the transport conditions of the truck transporting his goods by providing his Order ID, which has a specific truck associated). Any truck present in the farmer's fleet can be removed from it by the farmer itself, if not performing at travel at that point, in the Delete Truck Tab (Figure 4.37). Finally, the Farmer can check all of his truck's states ("Stop" if not travelling or "Travel" if currently performing a trip) and set the trip's end, in case of a travelling truck, or edit the truck sensor's if that truck is stopped (by accessing the Manage Trucks tab (Figure 4.39)). By editing the truck's sensors, the farmer can not only change that truck's sensors' types but also completely switch the truck's sensors by new ones (by changing the sensor's ID).

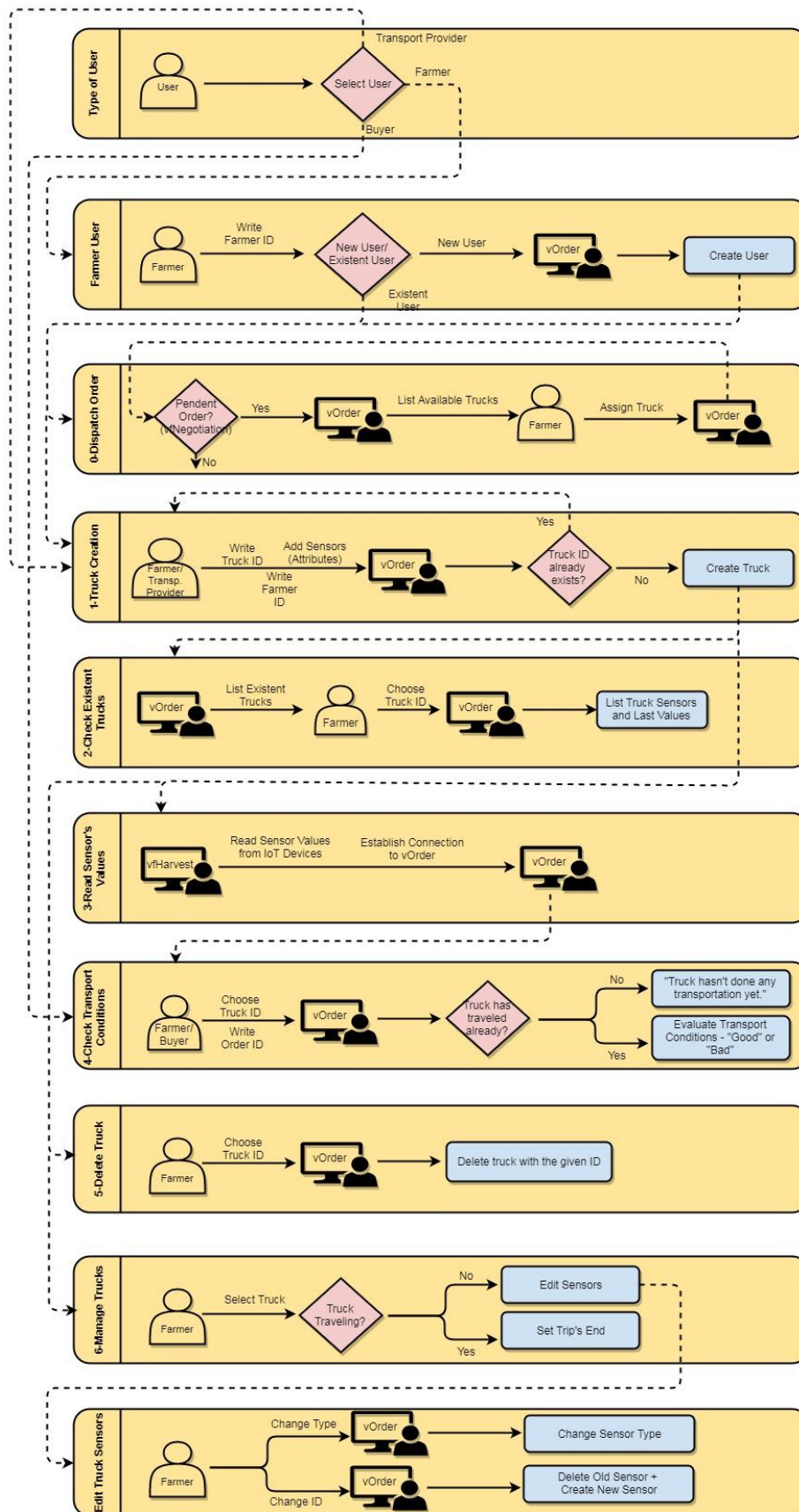


Figure 4.3 - vOrder Application Schematic

Chapter 4. vfNegotiation

4.3 vfNegotiation

vfNegotiation is an application created with the intent of being used both sellers (farmers in this scenario) wanting to get his products out for sale and by buyers looking for products to buy.

A farmer using this application can add new fruits to his stock (each defined by a fruit name, fruit breed and size), by setting a stock amount and a price for that fruit. Besides adding new fruit to his stock, the farmer also has the ability to check his current fruit stock and update that stock, by changing the amount of that fruit he has, and its price.

The buyer, from his side of the application, can search for the fruit provider based on several diverse criteria. The application provides the buyer with two different search methods, a manual search, where a list of farmers able to sell the searched fruit is presented and the buyer choose one of them, or an automatic search, where the user chooses the filters and only the best seller who meets the requirements is presented.

In Figure 4.4 is shown the vOrder Main Interface. It is composed two different interfaces, one for the Farmer and one for the Buyer, each one having different tabs, four and two respectively. Each tab allows the user to perform different actions, as shown in Subchapter 4.4 of this Chapter 4.

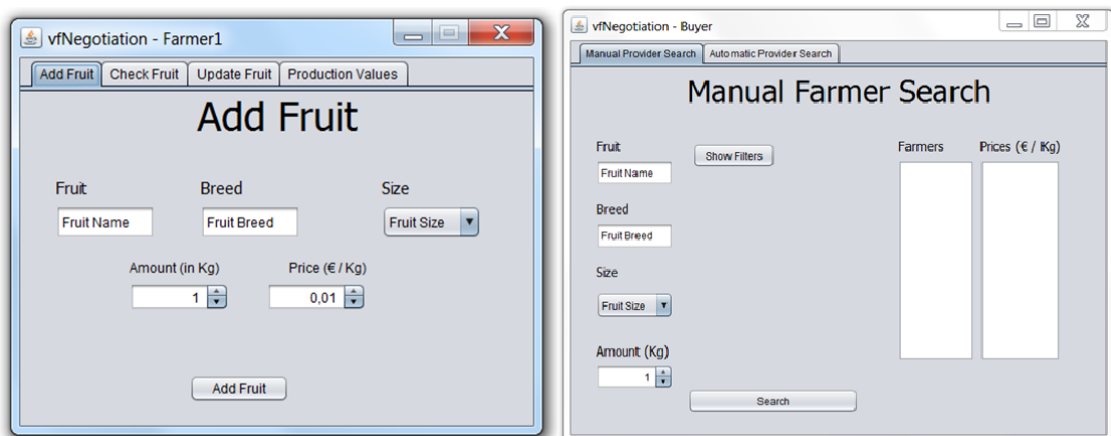


Figure 4.4 - vfNegotiation Interfaces

All the functionalities of the vfNegotiation Application, are explained in the next paragraphs, and can be found schematised in Figure 4.5.

Running the vfNegotiation Application will display an interface where the user can state which type of user he is (Figure 4.45). This application was developed to be used by two different types of users, farmers wanting to sell their produced goods, and buyers looking to buy different goods (the goods transacted in this application during its development are fruits, even though this app can easily be changed to the transaction of many other different goods).

To a farmer user, yet another interface is presented before he reaches the vfNegotiation Main Interface. This intermediate interface is the “Farmer User” Interface (Figure 4.46), where the farmer states whether he is a new user to the application or someone who has used this application before. If a new user presents himself he is added to the application system, if not the application will login into the presented user account informations, either way after this interface the user reaches the vfNegotiation Main Interface for farmers.

When a farmer reaches the vfNegotiation Main Interface he is presented with the “Add Fruit” tab (Figure 4.47), where he can choose which of his goods he wants to put for sale. By stating the Fruit Name, Breed, Size, Amount and Price/Kg that fruit automatically becomes available, under that farmer’s name, to be found and bought by any Buyer using this app. After one or more fruits are added to a farmer’s stock, he can check his entire stock in the “Check Fruit” tab (Figure 4.50). Here he is presented with lists containing all the fruits he currently has in stock, by selecting a specific fruit the application shows him the amount of that he still has in stock and for how much he is currently selling it. If the farmer wants to update a certain fruit, whether to change its current amount of its current price, he can do so in the “Update Fruit” tab (Figure 4.55). Here, after selecting which fruit he wants to update, the farmer can change that fruit’s values and update them in the selling system. In order to know which fruits he has produced and the amount of produced fruit that falls under each of that fruit’s size, the farmer can access the “Production Values” tab (Figure 4.61). This tab displays the information provided by the vProductMon application, that through the usage of IoT sensors at the production and calibration site, and

Chapter 4. vfNegotiation

rules, infers about the amount of fruit produced and transmits that information for the vfNegotiation application.

On the other hand, if a Buyer enters the vfNegotiation application, he will find functionalities enabling him to search for farmers selling the fruit he is looking to buy. When a Buyer first reaches the application, he is presented with the “Manual Provider Search” tab (Figure 4.63). Using this tab and after selecting the fruit he is looking for, the amount he intends to buy and optionally adding filters to the search (Figure 4.64), the buyer will be presented with a list of farmers that fulfil all the requirements, i.e. that currently have the targeted fruit in stock with an amount high enough to satisfy the order, and that meets the specs given in the filters. From the presented list, showing the farmer’s ID and the price at which he is selling the selected fruit, the buyer can choose the farmer and place the order. In addition to the Manual Search, the buyer also has access to an Automatic Search in the “Automatic Provider Search” tab (Figure 4.67). Similar to the Manual Search, here the buyer must also select the fruit he is looking for and the amount he intends to buy. Here however, the filter is a mandatory parameter, so the buyer must choose one of the existent filters (Figure 4.68), in order for the search to return the farmer that best meets the chosen filter. When the search is completed the automatic search will return a single farmer, displaying his ID and his price for the selected fruit. If the buyer is happy with the provided farmer he can order the selected goods.

After a search is made and a farmer is selected the buyer can order the goods in the “Order Confirmation” interface (Figure 4.70). Here he can have one last look at his order, where he can see the fruit he is ordering, its name, breed, size, amount and price/Kg, as well as total price of the order and the farmer ID of the selected farmer. If everything is correct he can place the order and will be given an Order ID (Figure 4.71), with which he can check the transport conditions of his goods in real time using the vOrder Application (Figure 4.35).

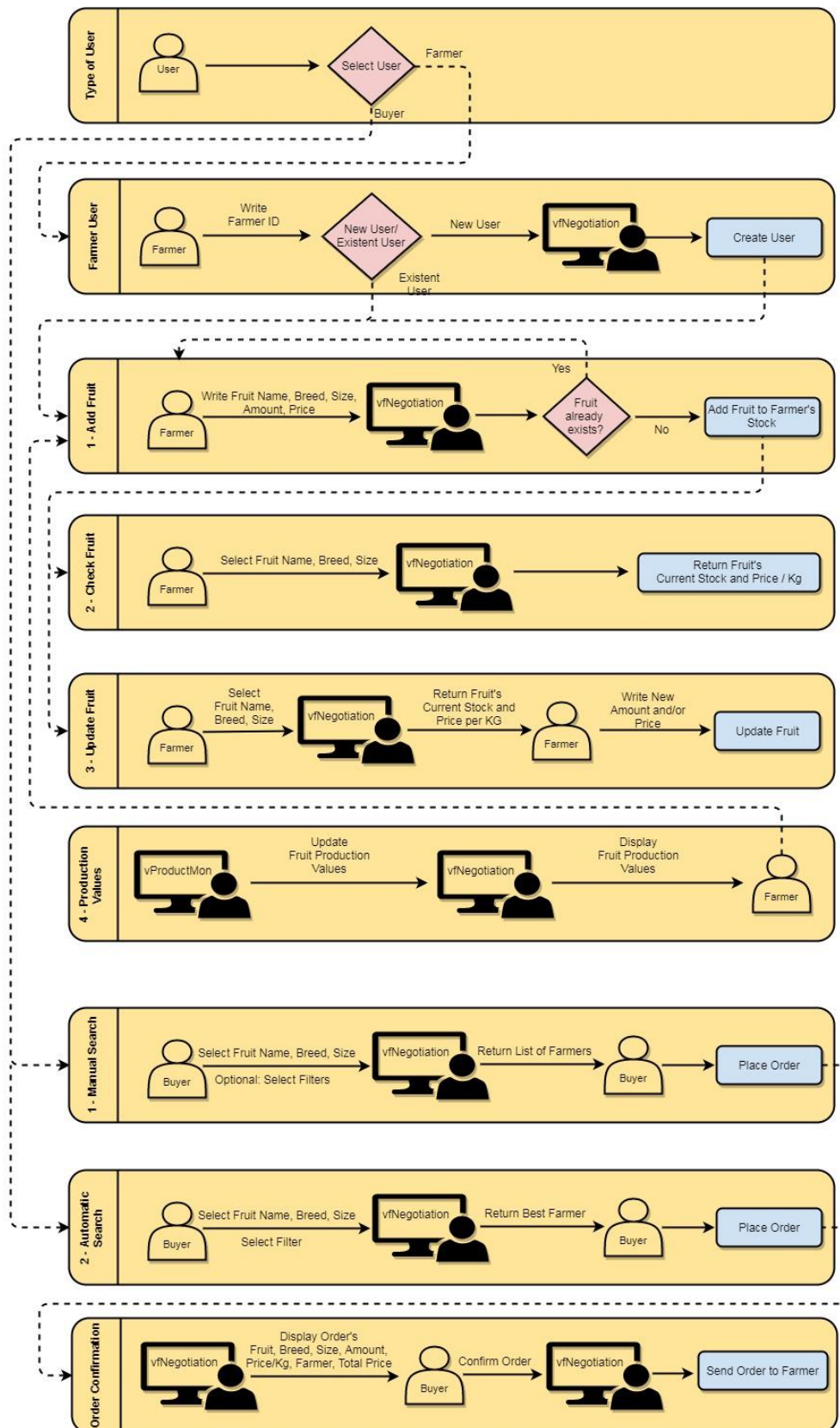


Figure 4.5 - vfNegotiation Application Schematic

Chapter 4. Applications Demo

4.4 Applications Demo

As stated before, to easily make use of all the Generic Enablers functions without needing to install and run them on the local machine, Docker was used. In order to launch the GEs using a Docker container all one has to do (after the developers encapsulate the GE in a Docker Image) is run the Docker container where the GE is installed.

To get the applications running, when they use a GE, the first thing to do is run the GE's Docker image, by accessing the Docker's Image directory and running it from there (through the Docker's Quickstart Terminal). Both the applications presented in this chapter use the FIWARE Orion Context Broker, so it needs to be launched, the way that is presented in Figure 4.6.

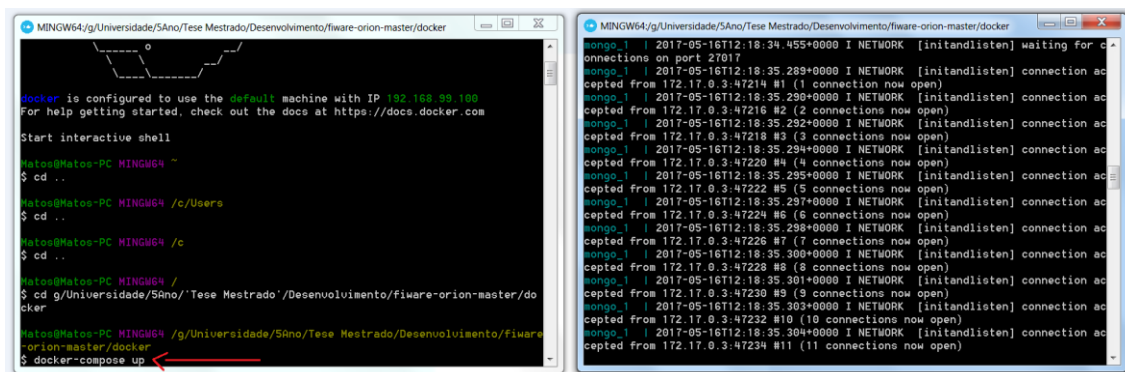


Figure 4.6 – Running FIWARE Orion Context Broker through Docker

The vOrder application though, also makes use of the FIWARE Short Term Historic (STH) – Comet, so similarly to running the FIWARE Orion Context Broker depicted in Figure 4.6, the FIWARE STH is put to run the same way, through a Docker container, as can be seen in Figure 4.7.

```

MINGW64/g/Universidade/5Ano/Tese Mestrado/Desenvolvimento/context.STH.Comet-master
docker is configured to use the default machine with IP 192.168.99.100
For help getting started, check out the docs at https://docs.docker.com

Start interactive shell
Matos@Matos-PC MINGW64 ~
$ cd ..
Matos@Matos-PC MINGW64 /c/Users
$ cd ..
Matos@Matos-PC MINGW64 /c
$ cd ..
Matos@Matos-PC MINGW64 /
$ cd g/Universidade/5Ano/Tese Mestrado/Desenvolvimento/context.STH.Comet-master
Matos@Matos-PC MINGW64 /g/Universidade/5Ano/Tese Mestrado/Desenvolvimento/context.STH.Comet-master
$ docker-compose up

```

```

disabled, invalid or not configured database name mapping, ignoring the mapping mechanism
fiware-sth-comet_1 | time=2017-05-16T12:31:34.321Z | lvl:INFO | corr:n/a | tran
s:n/a | op:OPER_STH_STARTUP | from:n/a | srv:n/a | subser:n/a | comp:STH | msg:D
atabase name encoding set to value: false
fiware-sth-comet_1 | time=2017-05-16T12:31:34.321Z | lvl:INFO | corr:n/a | tran
s:n/a | op:OPER_STH_STARTUP | from:n/a | srv:n/a | subser:n/a | comp:STH | msg:P
roof of life interval set to value: 60
fiware-sth-comet_1 | time=2017-05-16T12:31:34.322Z | lvl:INFO | corr:n/a | tran
s:n/a | op:OPER_STH_STARTUP | from:n/a | srv:n/a | subser:n/a | comp:STH | msg:S
TH component configuration successfully completed
fiware-sth-comet_1 | time=2017-05-16T12:31:37.012Z | lvl:INFO | corr:n/a | tran
s:n/a | op:OPER_STH_SERVER_START | from:n/a | srv:n/a | subser:n/a | comp:STH |
msg:Starting up the STH server version 2.2.0-next...
mongo_1 | 2017-05-16T12:31:33.639+0000 I NETWORK [initandlisten] wa
iting for connections on port 27017
fiware-sth-comet_1 | time=2017-05-16T12:31:37.015Z | lvl:INFO | corr:n/a | tran
s:n/a | op:OPER_STH_DB_CONN_OPEN | from:n/a | srv:n/a | subser:n/a | comp:STH |
msg:Establishing connection to the database at mongodb://@mongo:27017/sth_teste
ruice
fiware-sth-comet_1 | time=2017-05-16T12:31:37.111Z | lvl:INFO | corr:n/a | tran
s:n/a | op:OPER_STH_DB_CONN_OPEN | from:n/a | srv:n/a | subser:n/a | comp:STH |
msg:Connection successfully established to the database at mongodb://@mongo:2701
7/sth_testservice

```

Figure 4.7 – Running FIWARE Short Term Historic through Docker

The functionalities of both these GEs were explained in the previous chapter (Chapter 3) and their functionalities are essential to the development of both the presented applications. That fact will become clearer throughout the explanation of the applications always bearing in mind the GEs functionalities described before.

4.4.1 vOrder – Choose User

The first interface the user sees when he first opens the vOrder Application, is a simple three button’s interface allowing him to state which type of user is wanting to use the interface (a Transport Provider, a Buyer or a Farmer). According to the choice done here, different tabs in the main application interface will be enabled or disabled according to the user privileges and functions. An image of that “Choose User” interface can be seen in Figure 4.8.

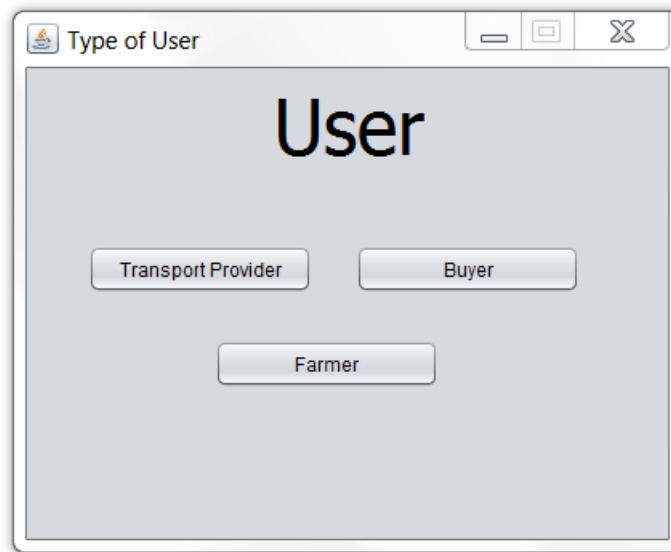


Figure 4.8 - Choose User Interface

The interface presented in Figure 4.8 allows the user to state whether he is a Transport Provider, a Buyer or a Farmer by pressing one of the three buttons presented. After this selection has been made, this interface will disappear, only to be set visible again once a new vOrder application is run. Once this interface disappears, the main vOrder interface will come to light (in most cases, read next paragraph for further information) with the active tab differing according to the option selected in the previous interface.

Even though both the Transport Provider and the Buyer types of user, reach the vOrder main interface as soon as they leave this first “Choose User Interface”, a Farmer user will first have to go through an intermediate checkpoint.

4.4.2 vOrder – Farmer User Interface

When the “Choose User” interface is closed due to the pressing of the Farmer button, a new interface is presented. The new presented interface “Farmer User Interface” can be observed in Figure 4.9.



Figure 4.9 - vOrder: Farmer User Interface

The interface presented in Figure 4.9 allows the farmer to state whether he is a new user or an existent user of the vOrder application. In case of the user trying to proceed with either not selecting any of the options (New User / Existent User) or not typing any Farmer ID an error message like the ones presented in Figure 4.10, is presented.

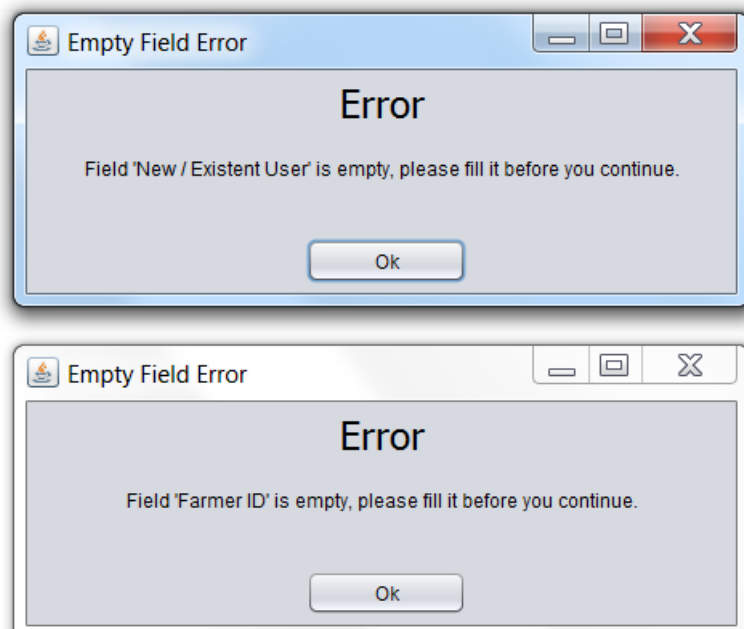


Figure 4.10 - Farmer User Interface: Errors Interface

Chapter 4. Applications Demo

On the other hand, if the New User option is selected but the provided Farmer ID, equals another already present in the application database, an error message like the one presented in Figure 4.11, is displayed.



Figure 4.11 - Farmer User Interface: Farmer Creation Error

In a third option, when the "Existent User" option is selected, but the provided Farmer ID is not present in the application database, an error message like the one presented in Figure 4.12, is displayed.

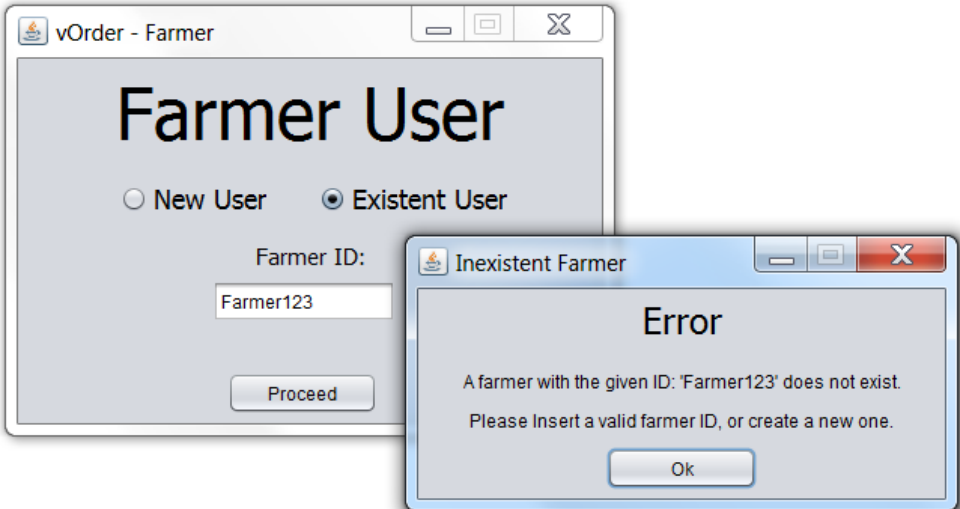


Figure 4.12 - Farmer User Interface: Inexistent Farmer Error

Finally, if either an inexistent Farmer ID is provided and the option New User is selected, or an existent Farmer ID is provided and the Existent User option is selected, by pressing the Proceed button, the user will reach the vOrder Main Interface.

4.4.3 vOrder – Main Interface

Once the opening interface “Choose User” is closed (after the user has selected which type of user he is – Transport Provider, Buyer or Farmer), or the “Farmer User” intermediate interface in the scenario where the user is a farmer, the main vOrder interface is presented. This interface is made of a Tabbed frame with the different tabs on top being enabled or disabled according to the user privileges and functions. A representation the different interfaces accessible to a Transport Provider, a Buyer or a Farmer are presented in Figure 4.13, Figure 4.14 and Figure 4.15 respectively (note that even though the presented tabs are the same, both the Transport Provider and the Buyer type of user have some tabs disabled, i.e. tabs that that kind of user cannot access due to reduced privileges).

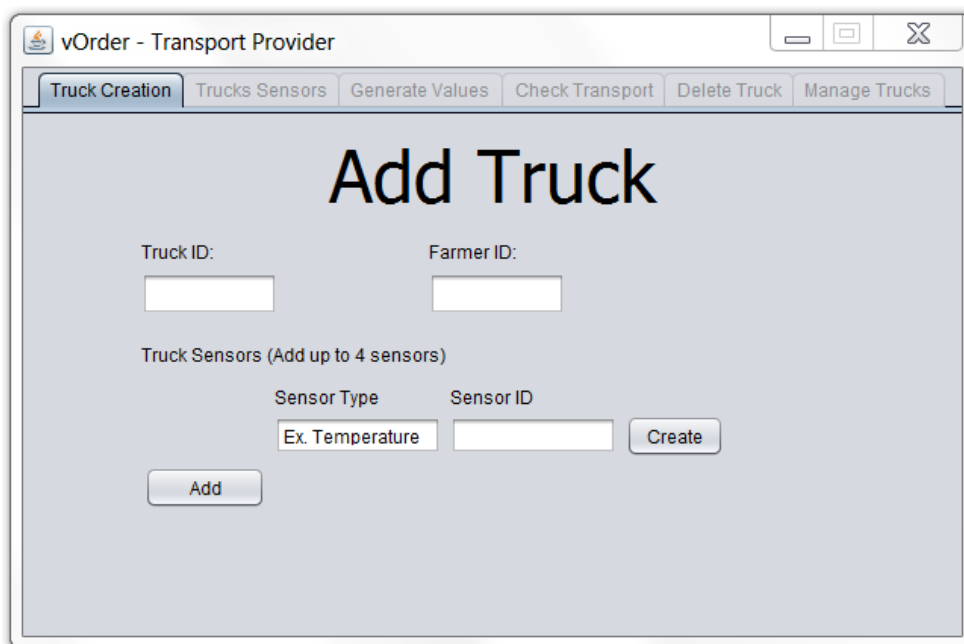


Figure 4.13 - vOrder main interface for Transport Providers

Chapter 4. Applications Demo

In Figure 4.13 is presented the vOrder main interface for a Transport Provider type of user. Due to reduced functions and privileges, this type of user can only access 1 out of the 6 tabs included in the vOrder app. This type of user can only create new trucks and assign them to a specific Farmer’s fleet (this function can be found explained later in this document, more precisely in Subchapter 4.4.5 of this Chapter 4).

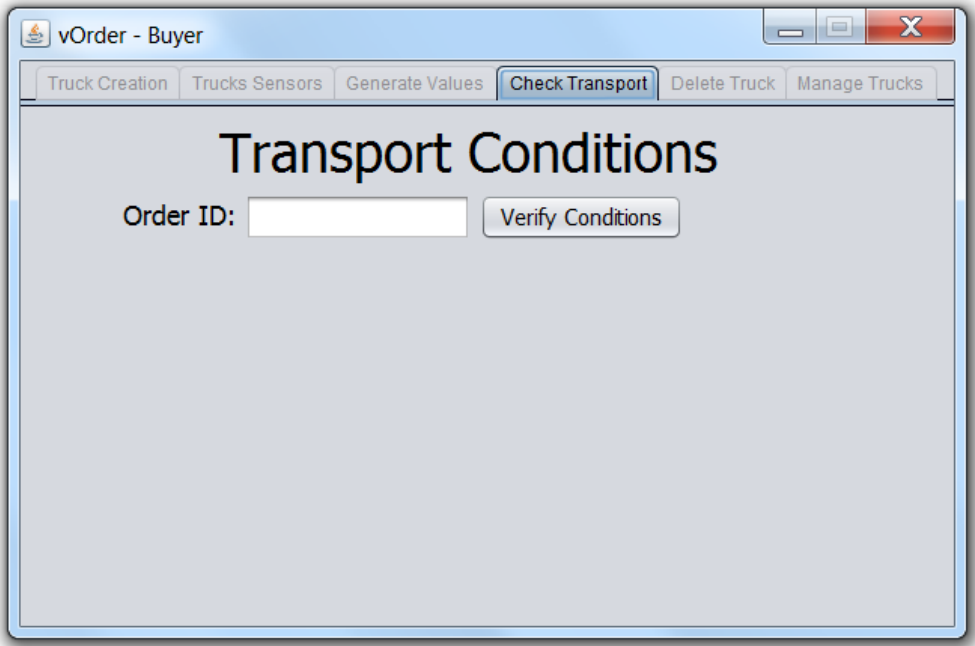


Figure 4.14 - vOrder main interface for Buyers

In Figure 4.14 is presented the vOrder main interface for a Buyer type of user. This user has the least privileges of all since he can only check the Transport Conditions of the products he has bought. Therefore, the only tab he can access is the “Check Transport” tab, and all the others are disabled to him. As all the others, the functions this tab allows are explained later in this document, more precisely in Subchapter 4.4.8 of this Chapter 4.

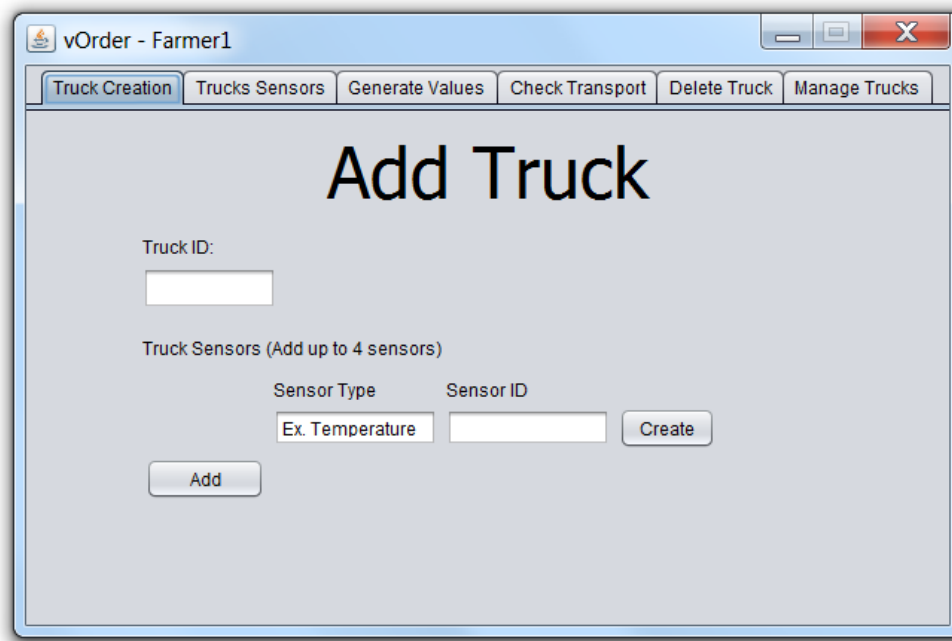


Figure 4.15 – vOrder main interface for Farmers

In Figure 4.15 is presented the vOrder main interface for a Farmer type of user. This user is the core user of this application, hence having full access to all of the application's tabs and functions. This user can therefore, as listed before, but now easier to understand by looking at the tabs presented in Figure 4.15:

1. create and add trucks to the fleet;
2. see a list of existent trucks, that truck's sensors and the last value for each of its sensors;
3. generate values for each of the truck's sensors (for test purposes only, since those sensors will be getting values from environment readings through an IoT device);
4. check the transport conditions of a given truck (to see if the goods being transported by that truck are traveling in good conditions);
5. remove a truck from the fleet;
6. manage his fleet's trucks (either to state that the truck's travel has ended, or to edit the truck's sensors – types or IDs).

Chapter 4. Applications Demo

4.4.4 vOrder – Dispatch Order

Another difference from the Farmer User to both the other users is the Dispatch Order interface appearance. Using the vfNegotiation application a buyer can order some goods from a specific farmer, creating an Order (as explained in Subchapter 4.4.21 of this Chapter 4). When that particular farmer “logs in” into the vOrder application a Dispatch Order interface is presented to him, like the one observable in Figure 4.16, as soon as he reaches the vOrder Main Interface. During his usage of the vOrder application a thread will keep running, constantly checking for pendent orders ten minutes (even though this value can be changed). If a buyer using the vfNegotiaion application orders some fruits from the logged farmer he will be immediately informed throw the appearance of a new Dispatch Order interface. If when the farmer logs in, he has multiple pendent orders, created during the time that he was not using the application, this interface will appear sequentially to the farmer, how many times as the number of pendent orders that farmer has, i.e. until a truck is dispatched to answer the Order, since as soon as a truck is sent to dispatch an order a new verification is made. After all the pendent Orders are dispatched, the 10 minutes thread will take place and will keep checking for new orders.

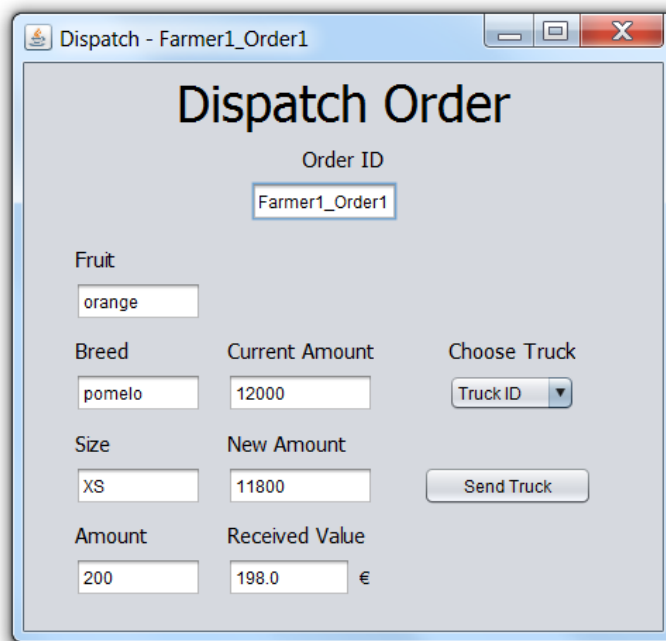


Figure 4.16 - vOrder Farmer: Dispatch Order Interface

For each pendent Order the logged farmer has, (any order addressed to him with no assigned truck) the farmer will face a Dispatch Order Interface (Figure 4.16). In that interface, the farmer will be able to see the Order ID, as well as the ordered fruit, breed and size and the ordered amount. The application will also show him his stock previous to the expedition of that order and his stock after that expedition, as well as the total value the farmer will receive from shipping that order. In order to expedite that order the farmer will have to choose one of his available trucks to carry the ordered goods (the application will list all the trucks contained in that farmer's fleet which aren't already performing any travel. Once a truck is selected this interface will disappear, and a new Dispatch Order Interface will automatically appear in case the farmer has any other pending orders. It is worth mentioning that if the farmer tries to expedite the order without assigning any truck to it the application will return an error message stating exactly so. If a truck is assigned to the order, that truck's state will change from "Stop" to "Travel", like it is better explained further ahead.

4.4.5 vOrder – Truck Creation Tab

The first tab present in the vOrder application, which is also the tab opened by default when the main interface is launched by either a Transport Provider or a Farmer user is the "Truck Creation" tab. This tab is accessible to both the Transport Provider and the Farmer users and, as the name of the tab implies, it is used to create/add a new truck to the fleet.

In order to create a new truck a valid ID needs to be provided (truck ID's are unique i.e. there can't be two trucks with the same ID in a single farmer's fleet, however two different farmers can have trucks with the same ID, this verification is presented ahead) as well as that truck's sensors (in the application denominated as attributes). The truck ID can be any name given by the user and goes in the "Truck ID" field. Additionally, up to four sensors can be added to the truck (i.e. a truck can contain one, two, three or four sensors reading values at the same time). The "Add" buttons allow the user to add new sensors to the truck and the "Create" button starts the truck creation and consequent addition to the fleet.

Chapter 4. Applications Demo

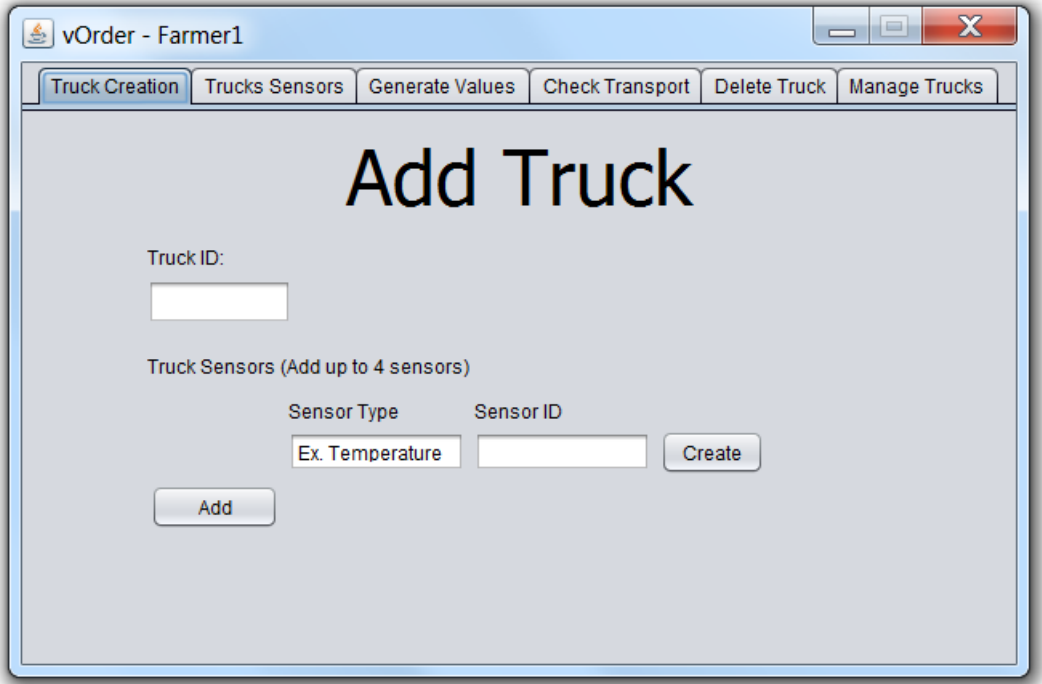


Figure 4.17 - Truck Creation: one Sensor

Once the “Add” button is pressed, two new fields appears, the first where the user can state what type of sensor he is adding (i.e. this is the field to write what type of sensor that truck will have – temperature sensor, humidity sensor, etc.), and the second where the user can state the ID of that sensor. From here the user has the ability to create the truck right away by pressing the “Create” button, or to add more sensors to the truck, by pressing the “Add” button. In Figure 4.18, are showed all the four sensors the user can add to a truck.

The screenshot shows a web application window titled "vOrder - Farmer1". The window has a navigation bar with tabs: "Truck Creation", "Trucks Sensors", "Generate Values", "Check Transport", "Delete Truck", and "Manage Trucks". The main content area is titled "Add Truck".

Truck ID:

Truck Sensors (Add up to 4 sensors)

Sensor Type	Sensor ID
Ex. Temperature	<input type="text"/>
Ex. Humidity	<input type="text"/>
Ex. Pressure	<input type="text"/>
Ex. CO2	<input type="text"/>

Figure 4.18 - Truck Creation: four Sensors

If the user wants to go back one step, i.e. not add as many sensors as he had previously anticipated, the remove button visible in Figure 4.18, will remove the fields related to the last added sensor.

The first verification the application does is to check if any Truck ID has been provided. If it has, the application continues to run, however if the user tries to create a truck without providing any Truck ID an error message similar to the one presented in Figure 4.19 is displayed.

Chapter 4. Applications Demo

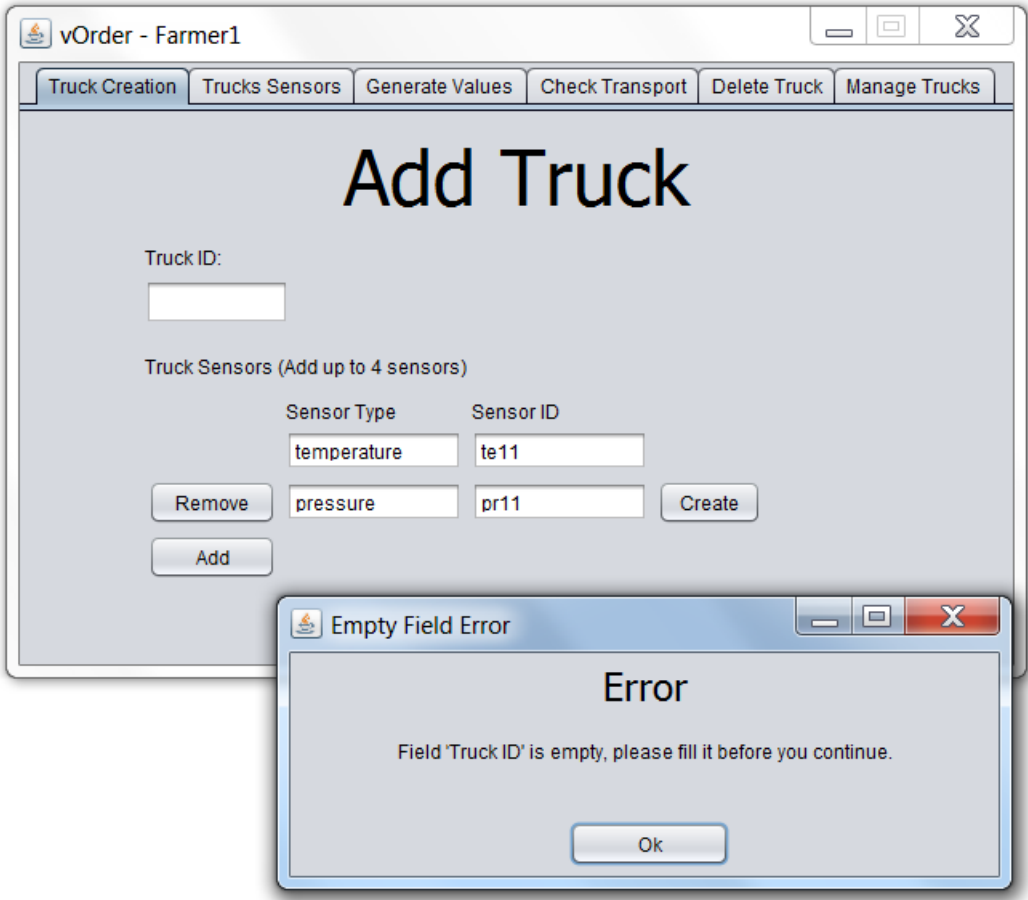


Figure 4.19 - Truck Creation: Empty Truck ID Message

Once a Truck ID is provided and all the desired sensors are added to the truck, the user can press the “Create” button, which will make the application check if all the input fields were valid, returning a message in conformity with the case it has found.

So, if for example the user creates a Truck with the information presented in Table 4.2:

Table 4.2 - Truck Creation Example One.

Truck ID	Truck Sensors
Truck1	temperature
	humidity

Assuming there is no truck in that user's fleet with the Truck1 ID, the application shows the success message presented in Figure 4.20. Alongside with the truck creation, a Subscription is made to each of the created sensors (as stated in Chapter 3, the Subscriptions is the way that enables the FIWARE Orion Context Broker and the FIWARE STH to read the sensor values, i.e. be alerted every time a sensor reads a new value).

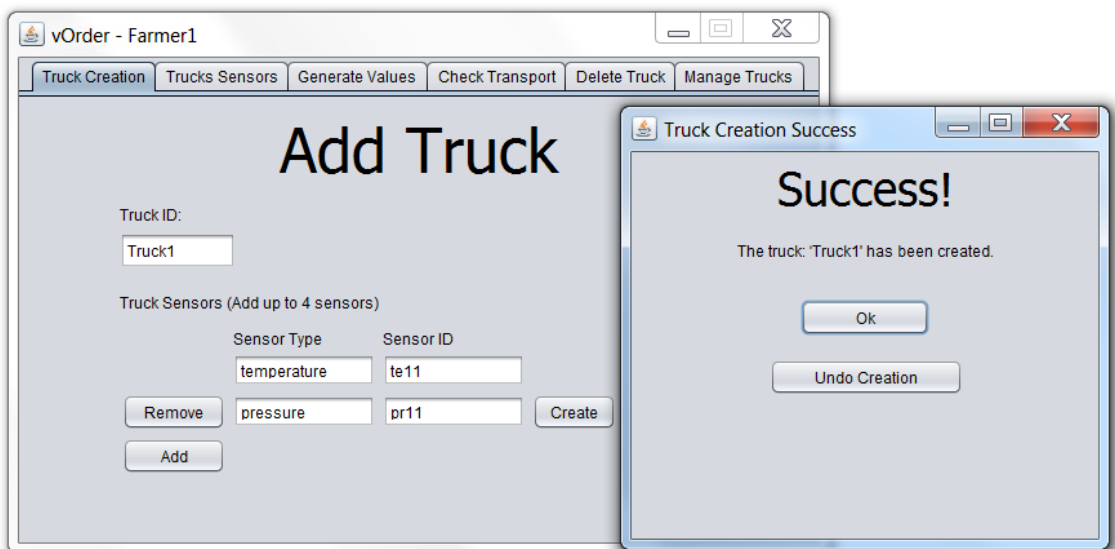


Figure 4.20 - Truck Creation: Success Message

The “Undo Creation” button present in the Truck Creation Success Message (here depicted in Figure 4.20), allows the user to, in case of realizing the occurrence of any mistake during the truck creation (wrong Truck ID or Sensors, for example), quickly and easily delete the truck just created, and create a new one after making the amends.

If that “Undo Creation” button is pressed, the Truck just created is deleted from the DB and a “Delete Truck Success” message appears, similar to the one presented in Figure 4.21.

Chapter 4. Applications Demo

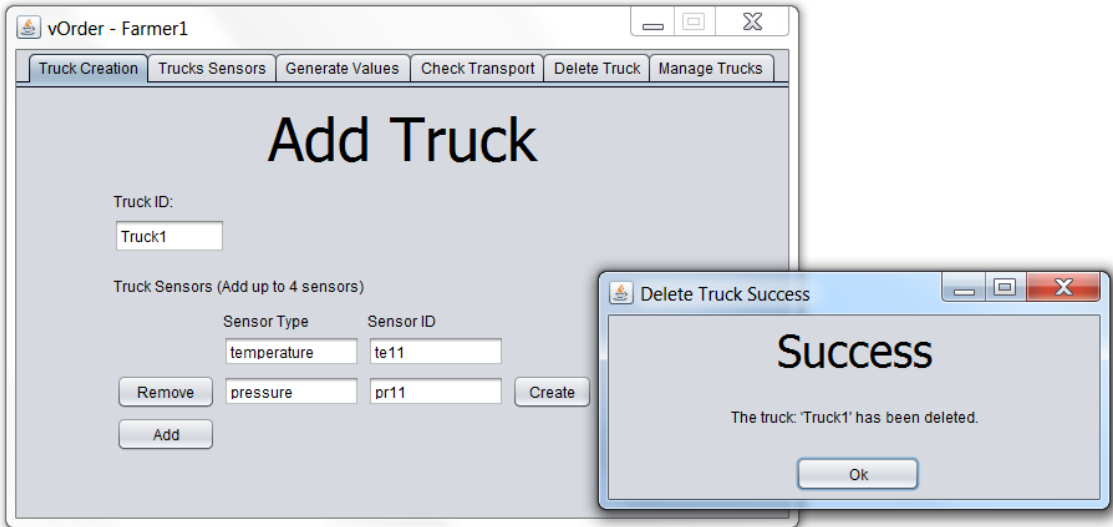


Figure 4.21 - Truck Creation: Delete Truck Success Message

If after the truck with the ID Truck1 is created (and not deleted or the have his creation undone) the user tries to create another truck with that same ID, even if all the sensors are different as represented in Table 4.3, the application will return the error message presented in Figure 4.22.

Table 4.3 - Truck Creation Example Two

Truck ID	Truck Sensors
Truck1	pressure
	CO2 (concentration)

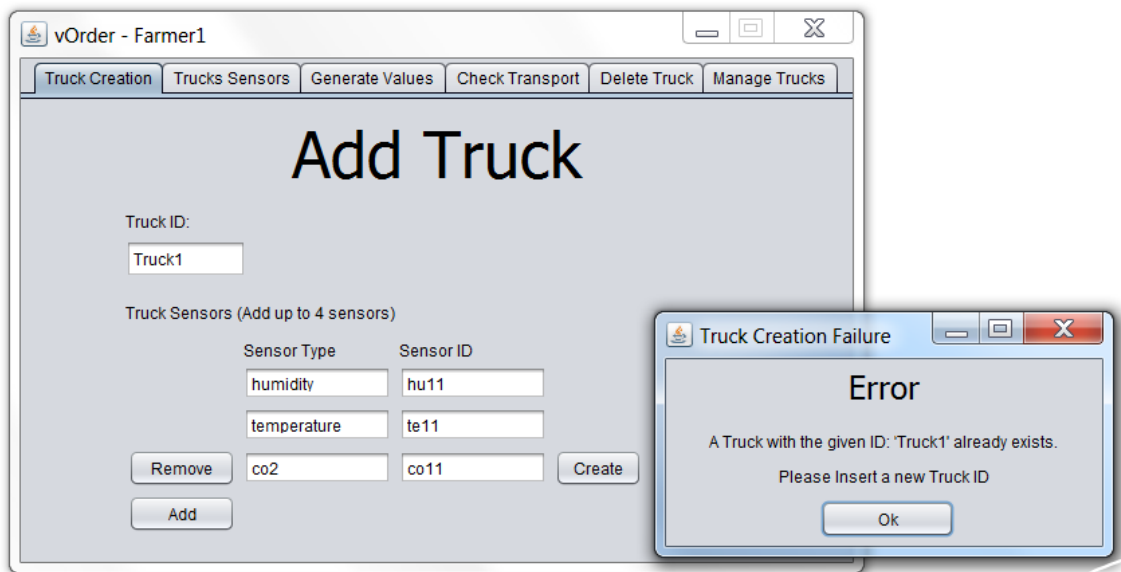


Figure 4.22 – Truck Creation: Failure Message

Right from the moment a new truck is created, it becomes instantly able to be consulted in the “Existent Trucks” tab, his sensors become ready to read and store values, and it becomes possible edit or eliminate the truck in the “Manage Trucks” and “Delete Truck” tabs respectively.

If this tab (Truck Creation) is accessed through a Transport Provider type of user, the only noticeable difference will be a field where the user will have to write to which farmer’s he is adding the new truck (by providing that Farmer’s ID). All the mentioned verifications remain active with an additional one making sure that the provided Farmer ID matches a farmer already signed in the application. That Truck Creation tab presented to a Transport Provider user can be observed in Figure 4.23.

Chapter 4. Applications Demo

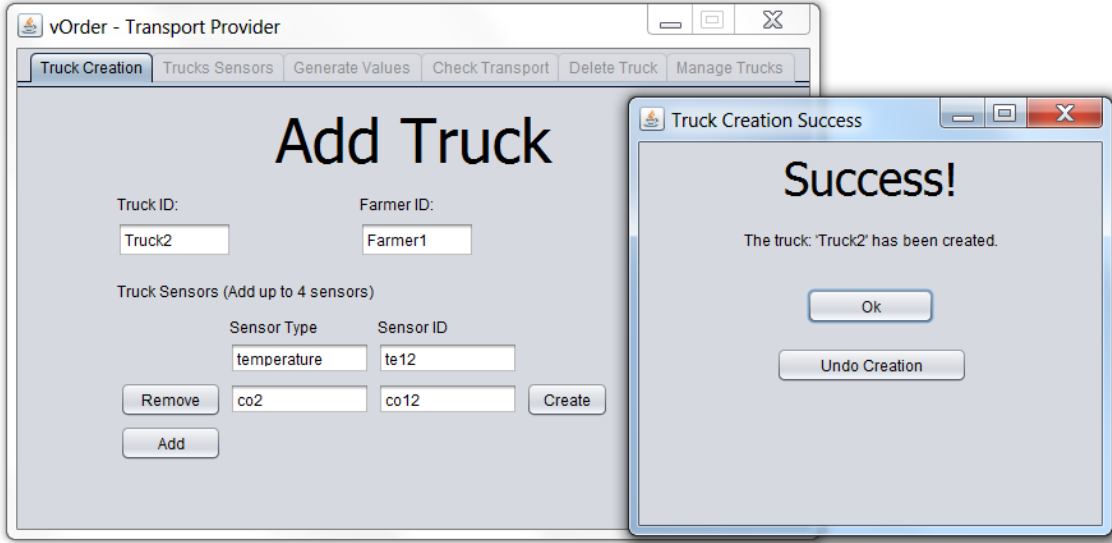


Figure 4.23 - Truck Creation Tab: Transport Provider User

4.4.6 vOrder – Existent Trucks Tab

The Existent Trucks Tab is a tab, like they will all be from now on (except for the Check Transport tab), accessible exclusively to the farmer type user. In this tab, the farmer can check his fleet of trucks, i.e. a list of trucks he has at his disposal to transport his goods to the buyer. In addition to getting a list of trucks, he can also check the last value every sensor from every truck has read. Both these functions are presented in Figure 4.24 and Figure 4.25 respectively.

In order to show the potential functionalities of this application, three more trucks were added to the Farmer1 fleet with different types and numbers of sensors. The trucks created and the sensors which equip each of them can be found listed in Table 4.4.

Table 4.4 - Trucks created for exemplification purposes.

Truck ID	Truck Sensors
Truck1	temperature
	humidity
Truck2	CO2 (concentration)
	pressure
Truck3	humidity
Truck4	CO2 (concentration)
	temperature
	humidity
	pressure

After created, these trucks will automatically be added to the fleet composed by available trucks, and will be present at any list showing the available trucks, like the one present in Figure 4.24.

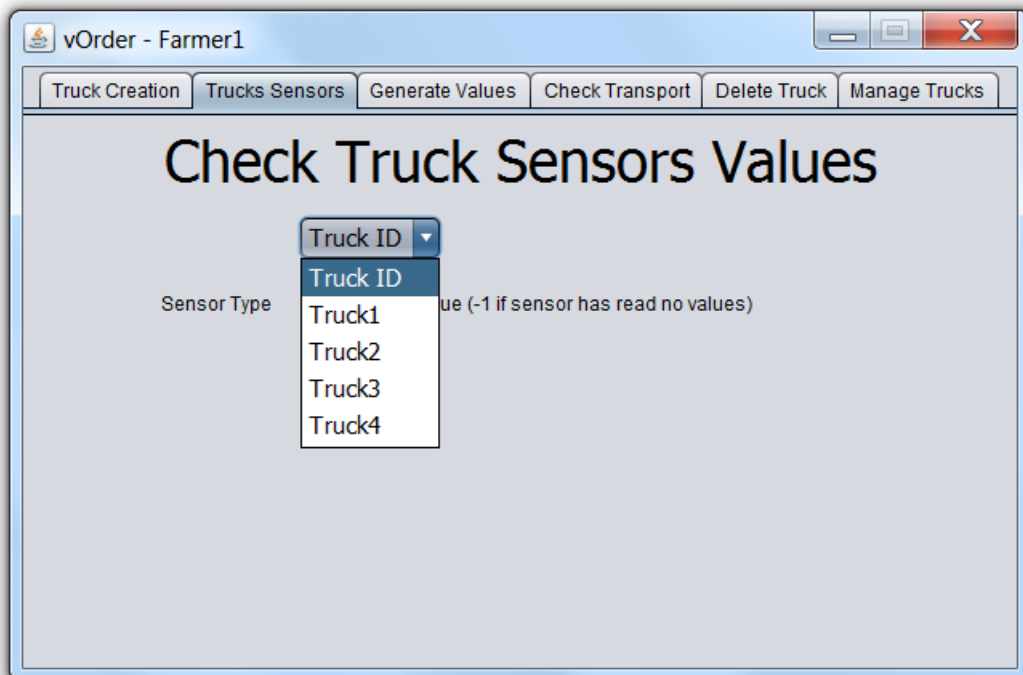


Figure 4.24 - Existent Trucks Tab: Trucks List

Chapter 4. Applications Demo

By selecting the "Combo box", as shown in Figure 4.24 the user is presented with a list containing all the Truck IDs of his available trucks. If a truck of that list is selected the application will reach for the database and show the last value read for each of that truck's sensors, as exemplified in Figure 4.25.

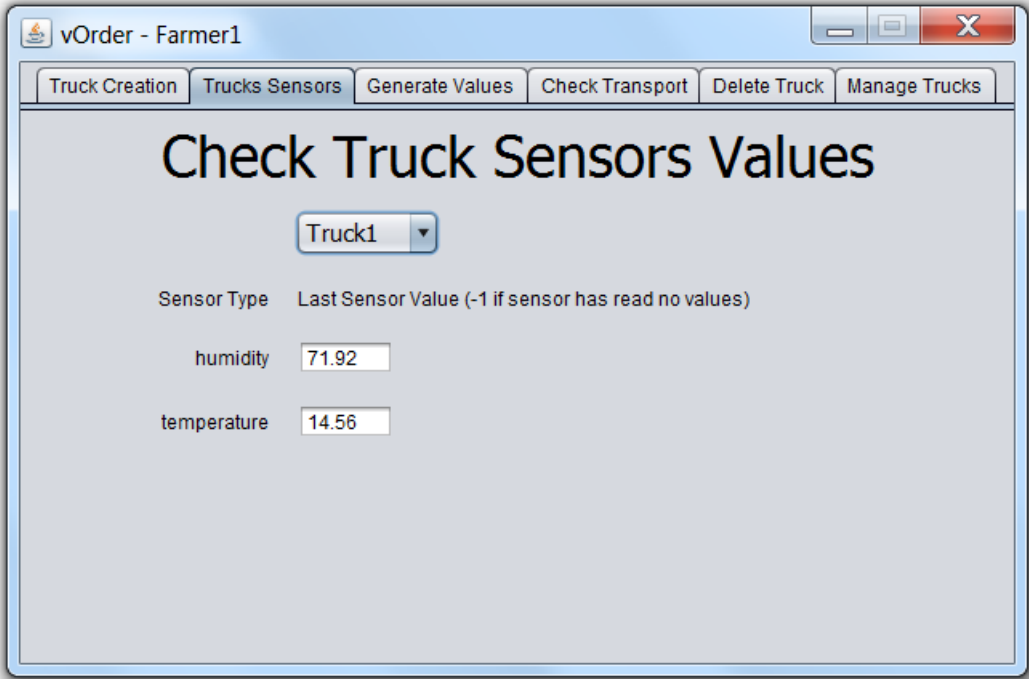


Figure 4.25 - Existent Trucks Tab: last read sensor values for a truck

If any of the sensors hasn't read any values yet, the "Last Attribute Value" section will present a value of "-1" by default, as depicted in Figure 4.26. This will be important to the "Check Transport" tab, which is explained further ahead in this document, in Subchapter 4.4.8 of this Chapter 4.

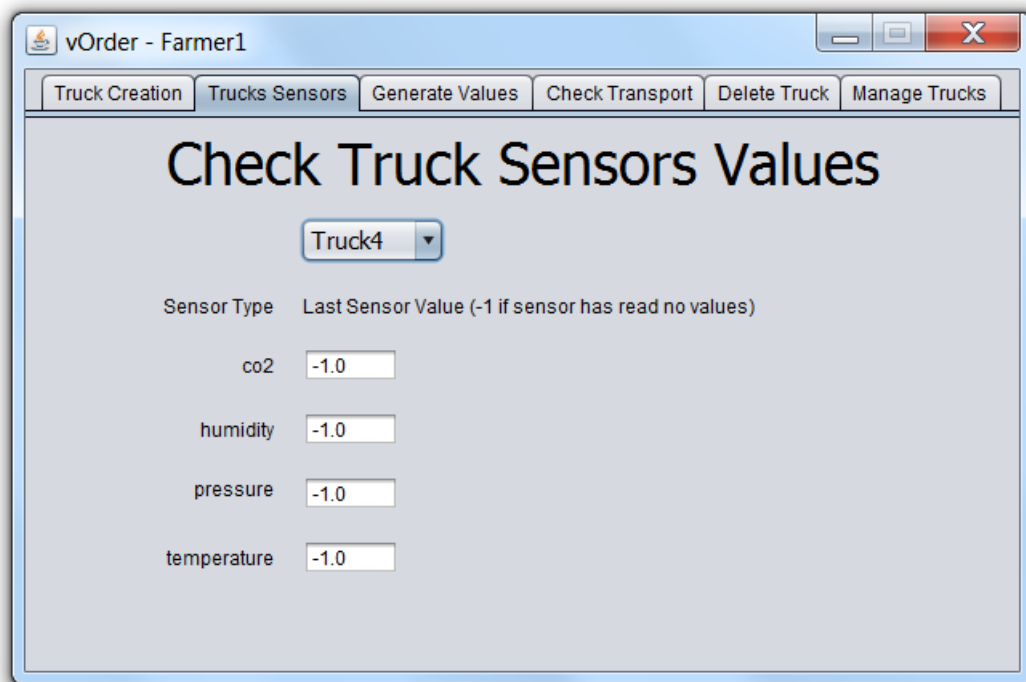


Figure 4.26 – Existent Trucks Tab: Sensors with no values example

4.4.7 vOrder – Generate Values Tab

This Generate Values Tab serves for testing purposes only, since its job is to simulate the reading of values by the sensors. During test phase while no sensor included in this application is directly connected to an IoT device, this tab allows the user to generate random values and assign those values to the truck's sensors in order to simulate the actual sensors readings.

After all the applications shown in Figure 4.1 are developed and put to use, this tab will no longer be required since all the values stored in the Orion's entities will come directly from real IoT sensors. These (sensors installed within every truck owned by farmers using this set of applications) and all the other sensors used throughout the deployment of this project are linked to the different applications in the vfHarvest Application. During the development of the vfHarvest application all the IoT sensors will be deployed and a way of communicating to the other applications, more precisely the vOrder and the vFail applications, will be developed. When the sensors are actively working on the field, and a way of communication is established, the values read by the sensors

Chapter 4. Applications Demo

will become available to the mentioned applications. The sharing of these values between the developed applications can be seen in Figure 4.1, represented by the arrows connecting the vHarvest application to both the vOrder and the vFail apps.

The random generated values for this tab's test purpose, even though they are random, they are always possible real values for each of the sensors used as test during this development section. In order to make sure only possible values were assigned to a specific sensor, a Random Gaussian Distribution was used.

A Random Gaussian Distribution generates random values clustered around an average value, i.e. given a mean value and a deviation value, all the random generated values will be within certain boundaries depending on the mean and deviation values. Regardless of the mean and the deviation values, all Random Gaussian generated values will obey these rules:

1. just under 70% of generated values will tend to have a value within one standard deviation to either side of the average;
2. just under 95% of generated values will tend to have a value within two standard deviations to either side of the average;
3. more than 99% of generated values will tend to have a value within three standard deviations to either side of the average.

Figure 4.27 helps to better understand these rules as it shows (using μ to represent the mean value, and σ to represent the deviation value) the percentage of values that will be generated within the mentioned boundaries.

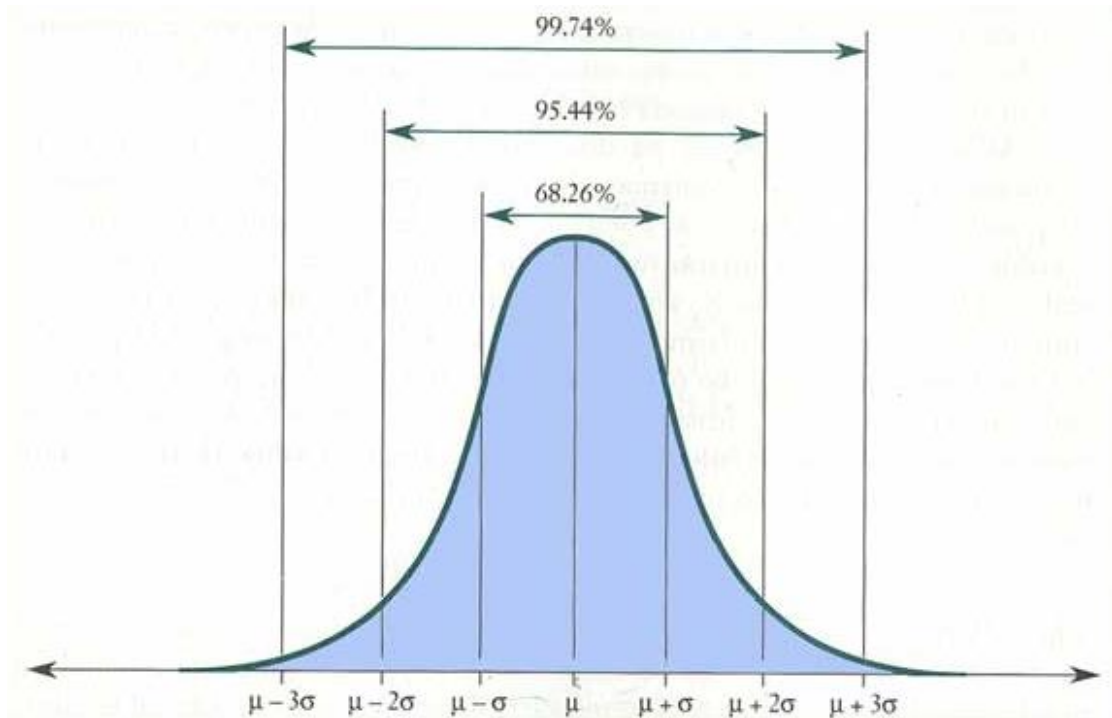


Figure 4.27 - Random Gaussian Distribution

For this thesis test purposes, the values for the mean value and for the deviation value chosen for each sensor are the ones showed in Table 4.5.

Table 4.5 - Mean and Deviation Values used for each sensor

Sensor	Mean Value	Deviation Value	Units
Temperature	18	4	°C
Humidity	50	15	%
Pressure	760	50	mmHg
CO2 (concentration)	4	1	%

According to the Mean and Deviation values presented in Table 4.5, and to the rules enumerated before it is expected that all generated values fall within the boundaries and percentages presented in Table 4.6.

Chapter 4. Applications Demo

Table 4.6 - Boundaries for each sensor and percentage of values falling under each one

Sensor	Boundaries			Units
	~70%	~95%	~99%	
Temperature	14 - 22	10 - 26	6 - 30	°C
Humidity	35 - 65	20 - 80	5 - 95	%
Pressure	710 - 810	660 - 860	610 - 910	mmHg
CO2 (concentration)	3 - 5	2 - 6	1 - 7	%

Table 4.6 shows that, for example, around 70% of the values generated for the sensor temperature will be comprehended within 14 to 22°C interval, around 95% of the generated values will fall within 10 to 26 °C interval and approximately 99% of the generated values will be between 6 and 30 °C. The same, with different boundaries will happen to each of the other sensor’s generated values.

With this working from behind the application the only data the user needs to send is the number of values the application should generate per minute (always bearing in mind that the sensors only read values with up to 1 second difference, i.e. in order to not overload the system the application only checks the sensors every second, even if the values change more than once a second, the application won’t see that), and for how many minutes the application should be generating values (this was done to keep the application working as close to reality as possible, since on the one hand the transportation probably take several minutes if not hours, and on the other hand and it is irrelevant to check a real sensor more than once a second because real values don’t change too much in a small amount of time). The interface that can be used to randomly generate these values is depicted in Figure 4.28.

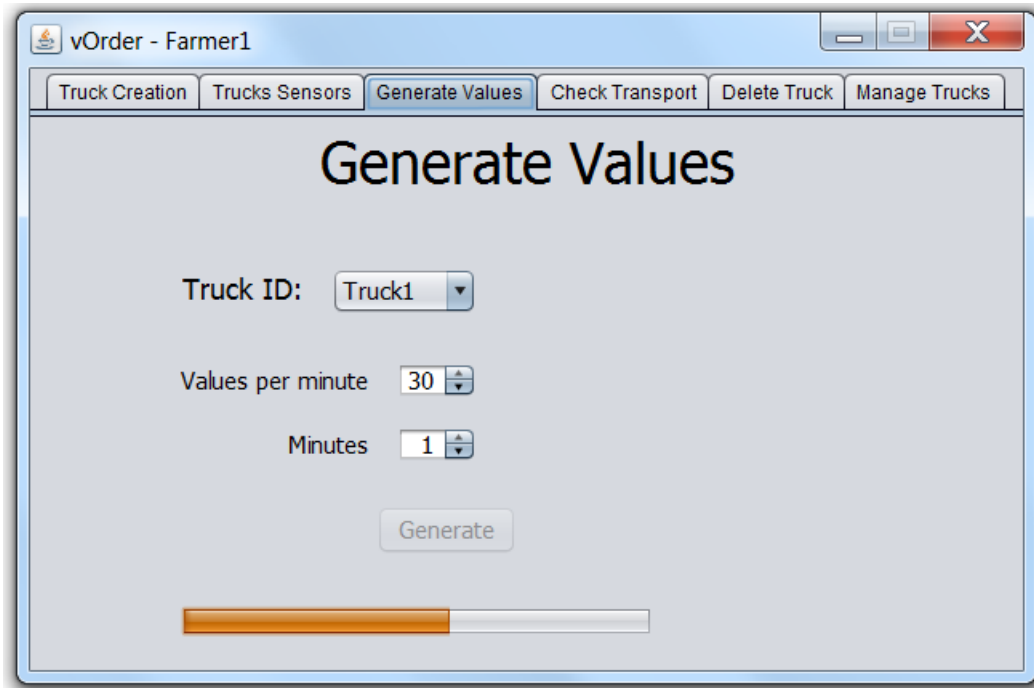


Figure 4.28 – Generate Values Tab: Generating 30 values over 1 min

Given the Truck ID to which sensors the user wants to assign the random generated values, the values to generate per minute and the minutes to run the generation of values the application will assign “values per minute \times minutes” values to each of the sensors that truck possesses. The user can accompany this procedure through the evolution of a progress bar (painted orange in Figure 4.28) or do any other action he wants (on a different tab) that the creation of values will continue to run in the background. When the creation is completed, a success message, like the one depicted in Figure 4.29, is presented.

Chapter 4. Applications Demo

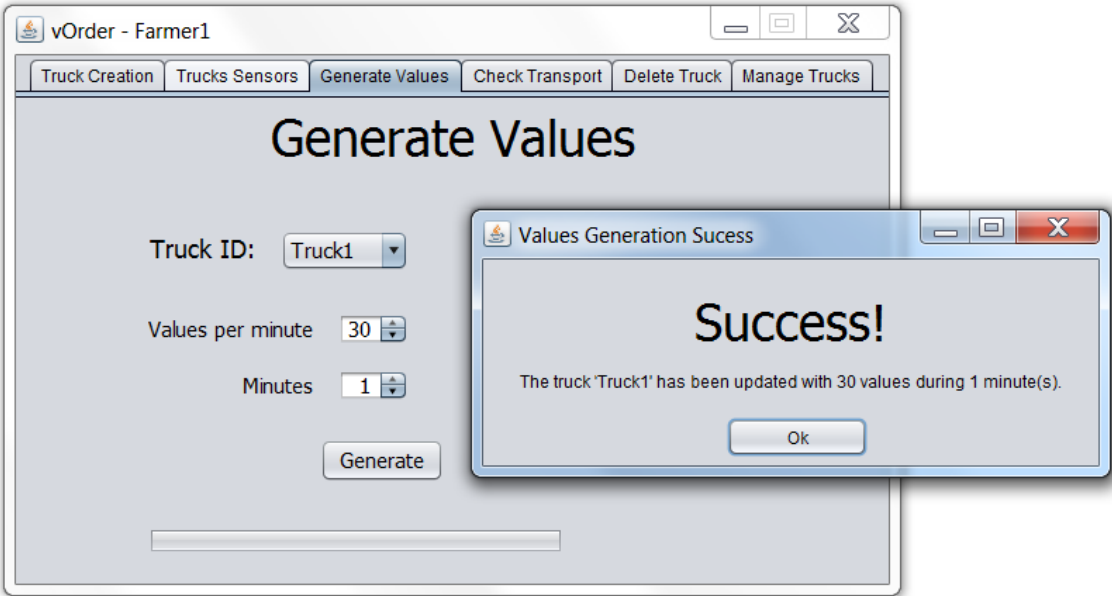


Figure 4.29 - Generate Values Tab: Values Generation Success Message

If, however, the user does provide the “values per minute” and the “minutes” but forgets to name the truck to which the values are to be created, a warning message (similar to the one in Figure 4.30) is displayed.

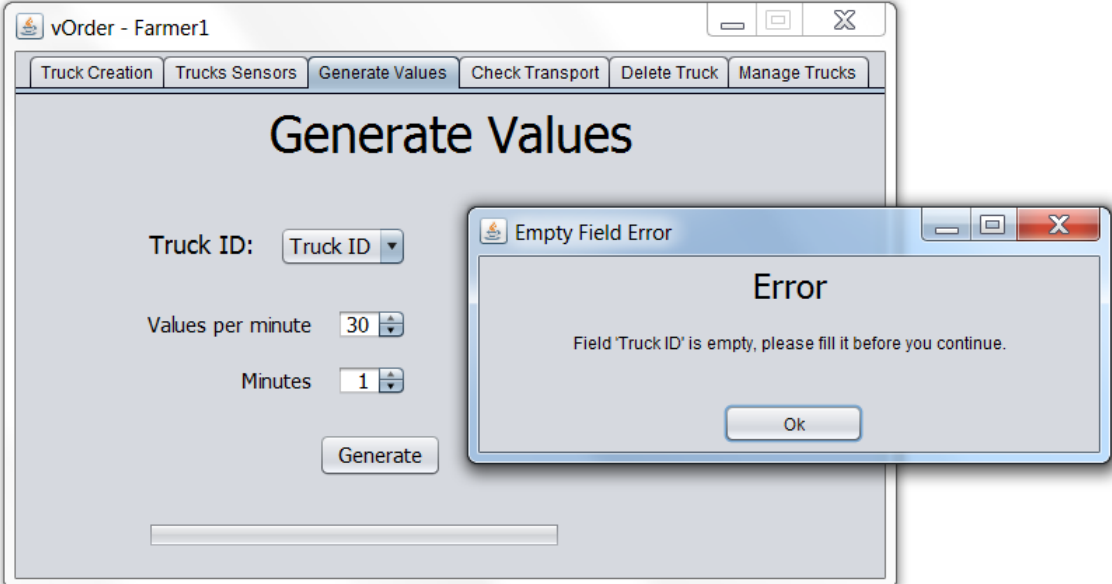


Figure 4.30 - Generated Values Tab: Empty Field Error Message

4.4.8 vOrder – Check Transport Tab

After a truck has completed transportation, or even at any point during that transportation, both the Farmer and the Buyer users can check the transportation conditions up until that point, through the Check Transport tab.

This tab however works very differently according to which type of user is accessing it. While for the Farmer a list of all his trucks, like the one presented in subchapter 4.4.6 – Figure 4.24 is presented, allowing him to check on the transport conditions of each and all of his trucks, for the Buyer only a text field is presented where he should type his Order ID.

Let's talk about each of these different interfaces in turns. As stated, on the one hand if this tab is accessed by a Buyer, the interface presented will be the one depicted in Figure 4.31.

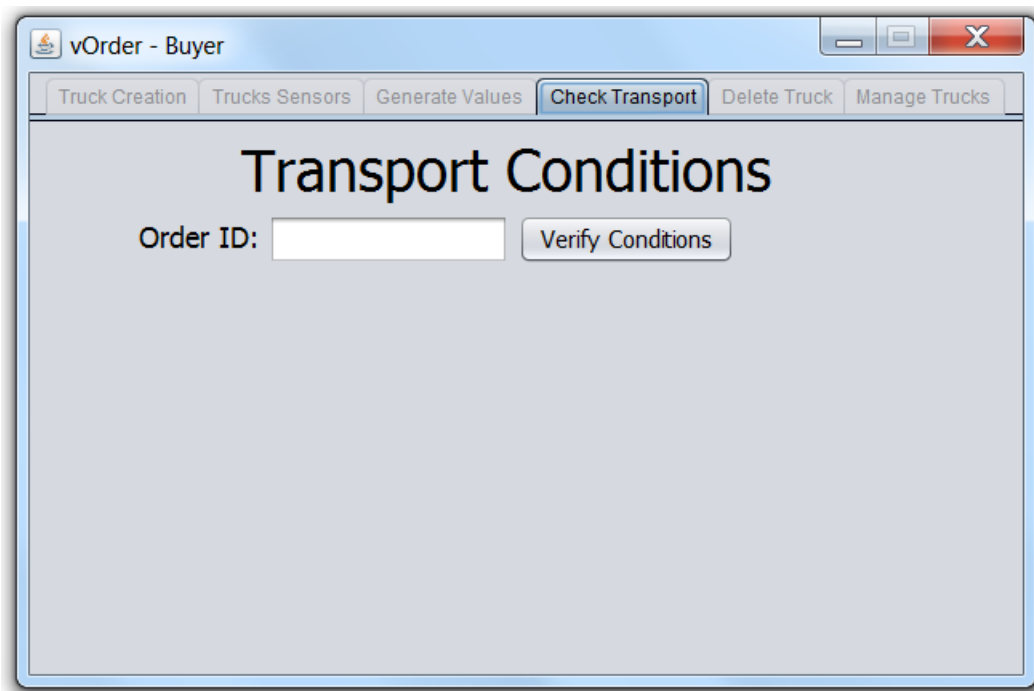


Figure 4.31 – Check Transport Tab: Buyer User

Since the Buyer is only supposed to be able to check the transport conditions of the truck transporting the goods he bought, in order to access that in-

Chapter 4. Applications Demo

formation, he needs to verify that he is the buyer of those goods, and he does that by typing the Order ID he got at the moment of the order creation (this order creation will be explained further ahead in the vfNegotiation application explanation – subchapter 4.4.21). Once a valid Order ID is provided the application will search which truck was assigned to that travel and will present the buyer with that truck’s transport conditions, i.e. that truck’s sensors readings. Besides the already presented verification of empty fields (Order ID being empty in this case), in case that the given Order ID does not exist in the Order Entity’s list the error presented in Figure 4.32 is displayed.

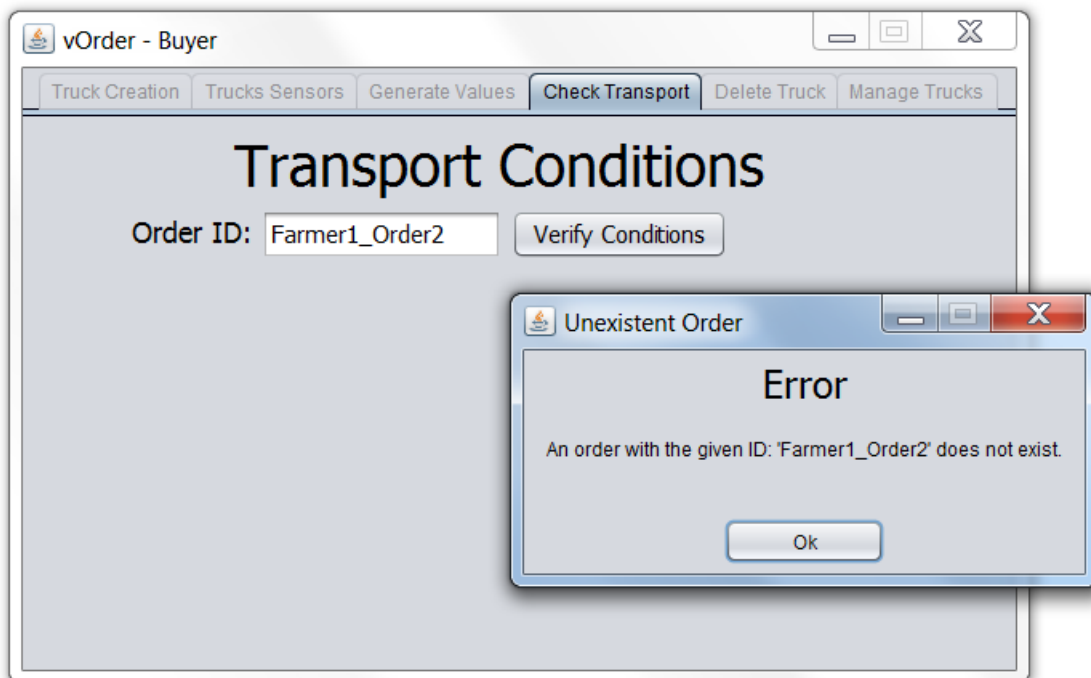


Figure 4.32 - Check Transport Tab: Inexistent Order Error Message

On the other hand, if this tab is accessed by a Farmer, a list of all of his trucks available for transportation is presented as depicted in Figure 4.33, and after one of those trucks is selected the tab will show that truck’s transport conditions up until that moment.

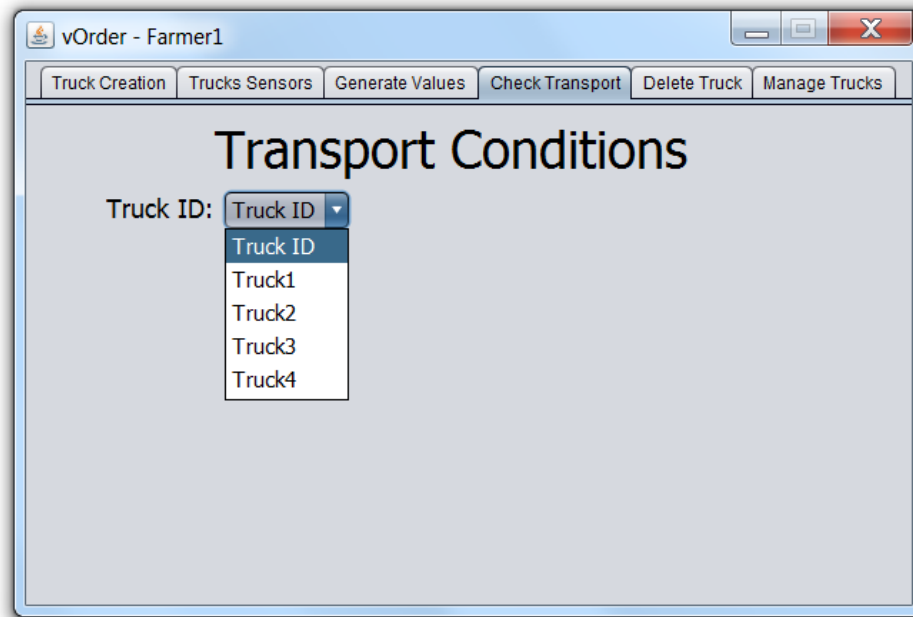


Figure 4.33 - Check Transport Tab: Farmer User

After a truck is selected, either by the farmer using the list, or by the buyer providing the Order ID, the tab will work the same regardless of the user.

In the eventuality that the selected truck hasn't done any transportation yet, i.e. his sensors haven't read any values so far, the application will show a message (like the one presented in Figure 4.34) stating exactly so.

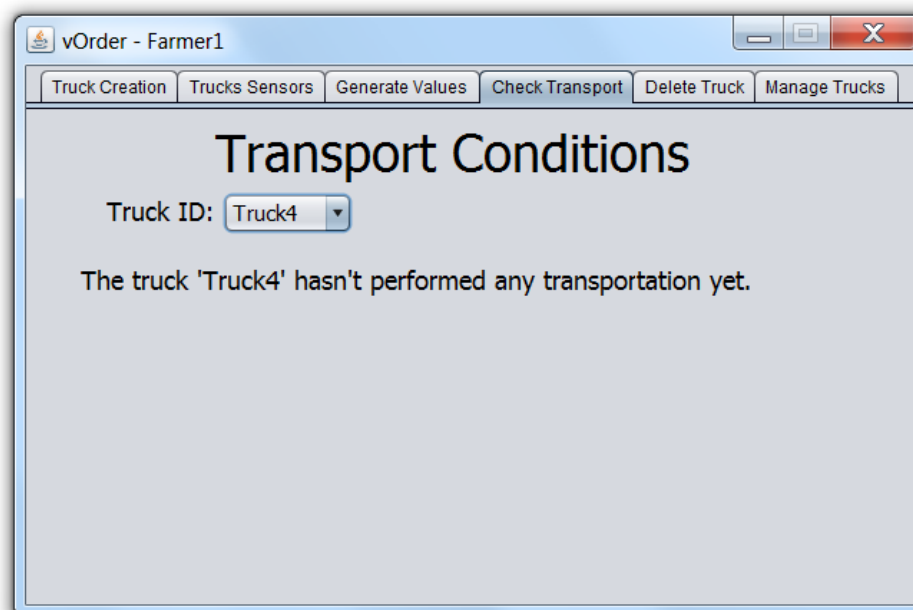


Figure 4.34 - Check Transport Tab: No Transportation Performed Message

Chapter 4. Applications Demo

If, on the contrary, if the selected truck has already completed a transportation or has already started one at the time the user reaches for his transport conditions, the application will display the key values of that truck's sensors that allowed it to infer about the transport conditions and, also, a qualitative evaluation of the transportation.

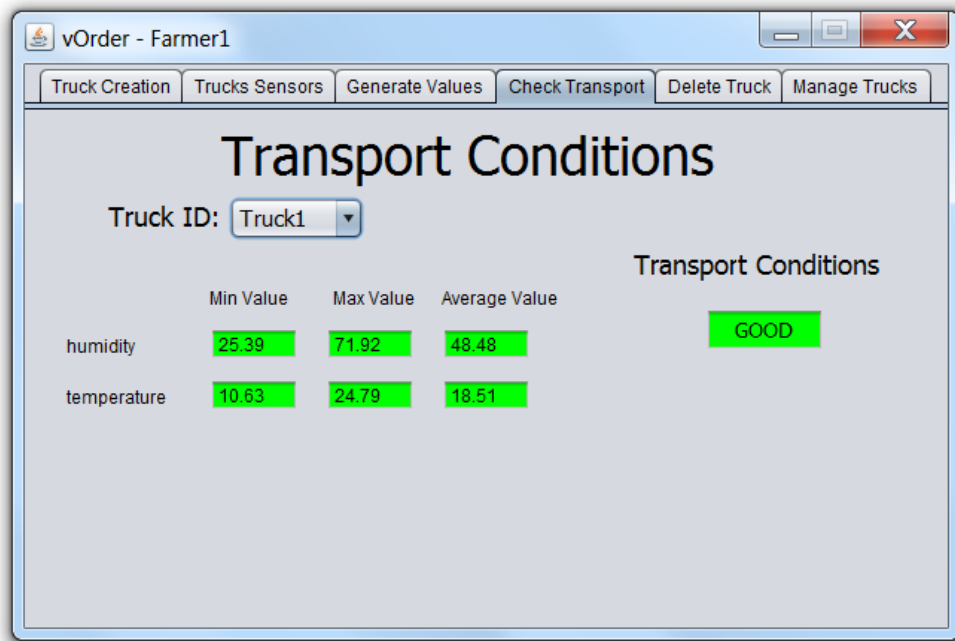


Figure 4.35 - Check Transport Tab: Normal Transport Conditions View

In Figure 4.35 is presented the information a user gets access to, once a truck that is performing or has already performed a transportation trip gets selected, either from the list of available trucks (Farmer user) or automatically through the verification of which truck is assigned to the given Order ID (Buyer user). As the image shows, the user is presented with:

1. a list of all the sensors that truck possesses;
2. the minimum value read by each of the sensors during the last transportation trip, up until the time the search was made;
3. the maximum value read by each of the sensors during the last transportation trip, up until the time the search was made;

4. the average value read by each of the sensors during the last transportation trip, up until the time the search was made;
5. a qualitative evaluation of the transport conditions during the last transportation trip, up until the time the search was made.

It is worth mentioning that in order for the STH to return the required values for the application to evaluate the trip transport conditions two dates need to be provided. The two dates provided create the limits within which the GE will search for the minimum and the maximum values, so a starting time/date for the trip needs to be provided, as well as an ending time/date. Both these dates are found using the subscription "Truck State" (explained before - subchapter 4.4.8). When a truck is assigned to an order, that truck state will change from "Stop" to "Travel", and the STH GE will be notified of that change will returning the time/date when it happened, hence creating the starting time/date of the trip. If the Transport Conditions check is done during the trip, the application will use that starting time/date and will search the required values until the present. If the trip ends (function done manually by the user, as explained further ahead in this document - changing the truck state from "Travel" to "Stop" and creating a new time/date) and the Transport Conditions check is done after it has ended, the starting time/date will remain the same and the ending time/date will have the time returned by the STH when the truck stated changed from "Travel" to "Stop". If that same truck is later assigned to a new trip (and to a new Order) the starting time will change, and so will the ending time when the truck ends his travel.

The qualitative evaluation mentioned before and observable in Figure 4.36 is created according to a mathematical analysis applied to the values read by the sensors. In order for the application to decide if the transport conditions were "GOOD" or "BAD" (the two alternative options to the Transport Conditions field), it calculates a series of comparisons using the values presented and the typical values of those weather conditions (at the moment, and for test purposes the application is working with the four enumerated sensors - temperature, humidity, pressure and CO2 concentration). The boundaries between

Chapter 4. Applications Demo

which each of the sensor's generated values are considered acceptable to the transport conditions and which are not, are presented in Table 4.7.

Table 4.7 -Boundaries of the Transport Conditions Evaluation

	Transport Conditions Evaluation			Units
	BAD	GOOD	BAD	
Boundaries	temperature<6	$6 \leq \text{temperature} \leq 30$	temperature>20	°C
	humidity<5	$5 \leq \text{humidity} \leq 95$	humidity>95	%
	pressure<610	$610 \leq \text{pressure} \leq 910$	pressure>910	mmHg
	CO2<1	$1 \leq \text{CO2} \leq 7$	CO2>7	%

As can be observed in Figure 4.35, the application automatically paints green all the values that meet the required conditions, presents in Table 4.7 (i.e. the application checks whether the minimum value of each sensor is above the lower bound, the maximum value is below the upper bound, and the average value is between both bounds). If all the conditions are met, the application rates the transport conditions of that truck's last trip, as "GOOD".

If on the other hand at least or more of the conditions are not met, the application will highlight with red the value that didn't fall within the delimited boundaries and the transport conditions is automatically rated as "BAD". An example where, even if just one of the values fails to respect the required the transport conditions are rated as bad can be found in Figure 4.36.

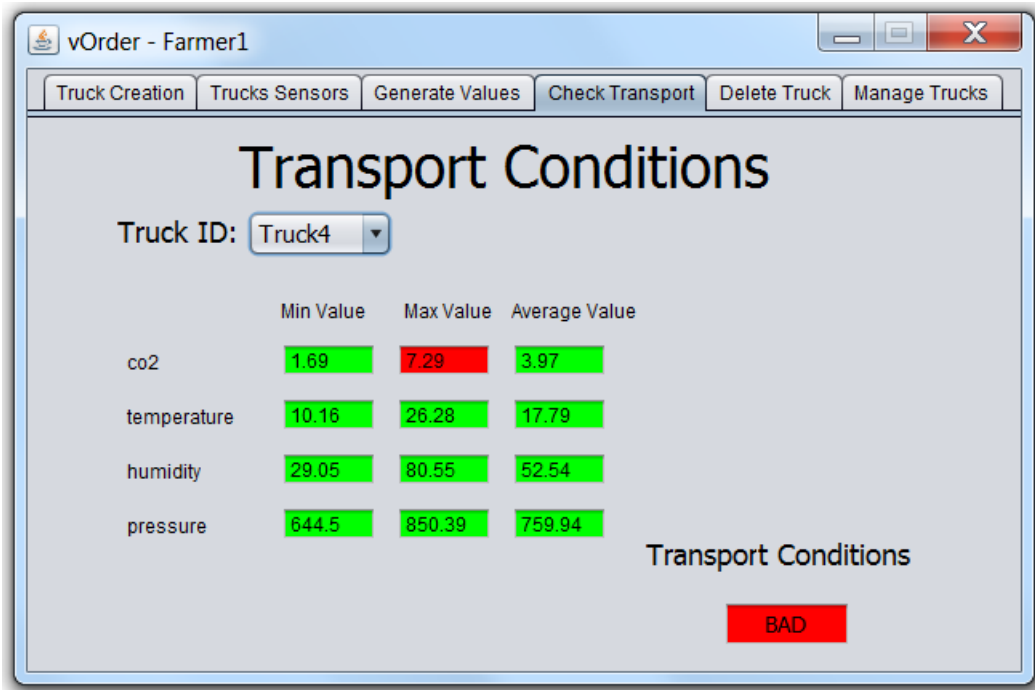


Figure 4.36 – Check Transport Tab: Transport Conditions rated as BAD

As can be observed in Figure 4.36, one of the values does not respect the required condition in order to allow the transport conditions to be evaluated as “GOOD” (the value painted in red in Figure 4.36), as the maximum value of one of the sensors is slightly above the upper bound for that sensor’s value (that bound can be observed in Table 4.8).

Table 4.8 – Boundaries of the Transport Conditions Evaluation for CO2 (concentration).

	Transport Conditions Evaluation			Units
	GOOD	GOOD	BAD	
Boundaries	CO2 < 1 <input checked="" type="checkbox"/>	1 ≤ CO2 ≤ 7 <input checked="" type="checkbox"/>	CO2 > 7 <input type="checkbox"/>	%

However, besides the minimum and maximum values, used for the application’s automatic evaluation of the transport conditions, to the user is also presented the average value. This was done with that intention that even though the application rates as “BAD” as soon as one of the values fall out of the

Chapter 4. Applications Demo

boundaries, the user can analyse all the provided information, including the average value and decide whether or not the transport conditions had a correct evaluation (for example, if some value slightly surpasses a boundary – and therefore the transport conditions is rated “BAD” by the application – but the average value for that sensor is far from that boundary, probably the goods are still in good conditions since for most of the time it was under good transport conditions and just during a small amount of time those conditions have deteriorated). The point behind the presentation to the user of this third value (the average value) was that even though the application rates the transportation based simply on raw values, the final decision lies with the user (both the Farmer and or the Buyer) to decide whether the goods are in good conditions or not.

4.4.9 vOrder – Delete Truck Tab

The Delete Truck Tab, once again accessible only to the Farmer user allows each farmer to remove an existent truck from his fleet. All his available Trucks are presented in a list, from which the farmer can choose which truck he wants to eliminate. By selecting the truck ID (from the truck to eliminate) and pressing the Delete button, the selected truck is eliminated from the DB and all the sensors subscriptions are removed. An example of a truck deletion is depicted in Figure 4.37.

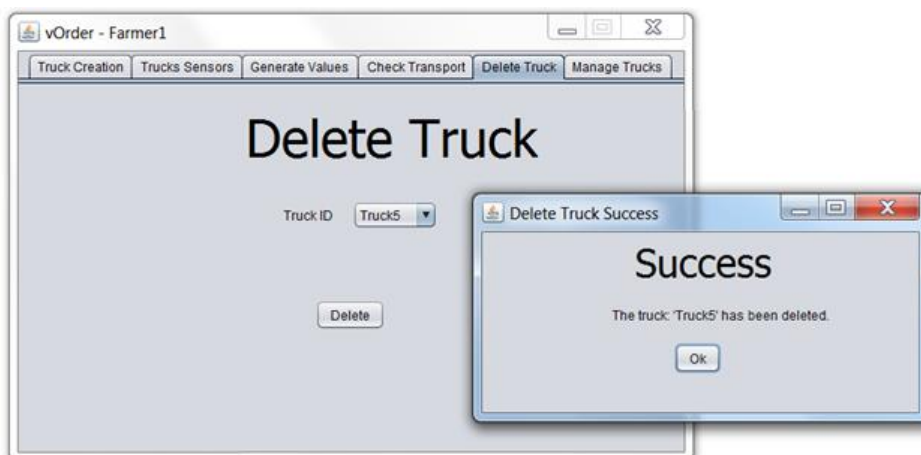


Figure 4.37 – Delete Truck Tab: Success Message

Removing the given truck from the fleet means having to delete several components and not just the truck itself. In order to correctly remove the truck all of the subscriptions bonded to that truck must be unsubscribed (all his sensor's subscriptions and the truck state subscription), all the sensors operating in that truck also need to be eliminated from the cloud database as well as the subscriptions entities. Only after deleting all these components can the truck be removed from the farmer's fleet and finally eliminated from the cloud database.

It is worth mentioning that a farmer can only delete trucks that are not currently performing any trips. If the farmer tries to delete a truck while it is travelling the application will show an error message like the one presented in Figure 4.38.

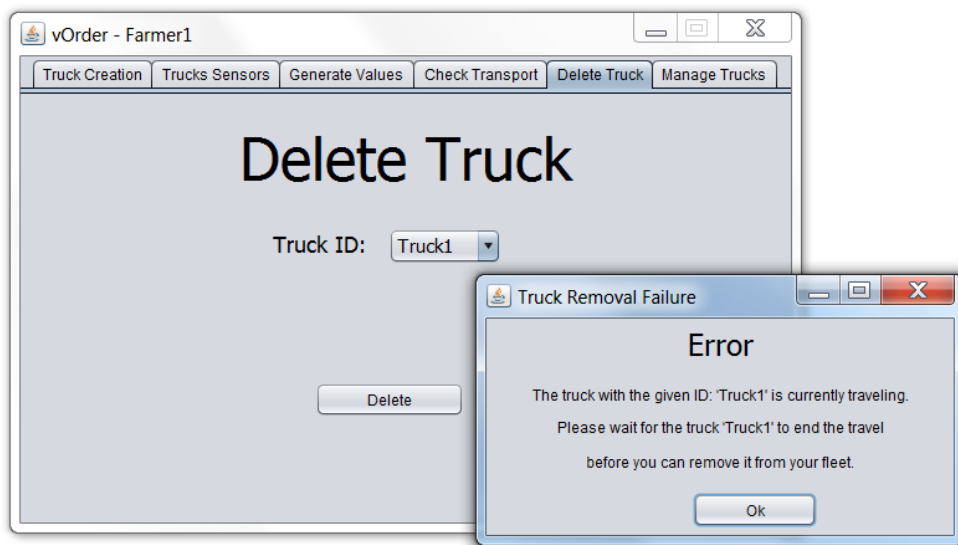


Figure 4.38 - Delete Truck Tab: Travelling Truck Error

4.4.10 vOrder – Manage Truck Tab

The last tab observable in the vOrder application interface is the “Manage Trucks” tab. Once again is a tab accessible only to the farmer, and is where he can see a list of all his trucks and their current state, as observable in Figure 4.39.

Chapter 4. Applications Demo

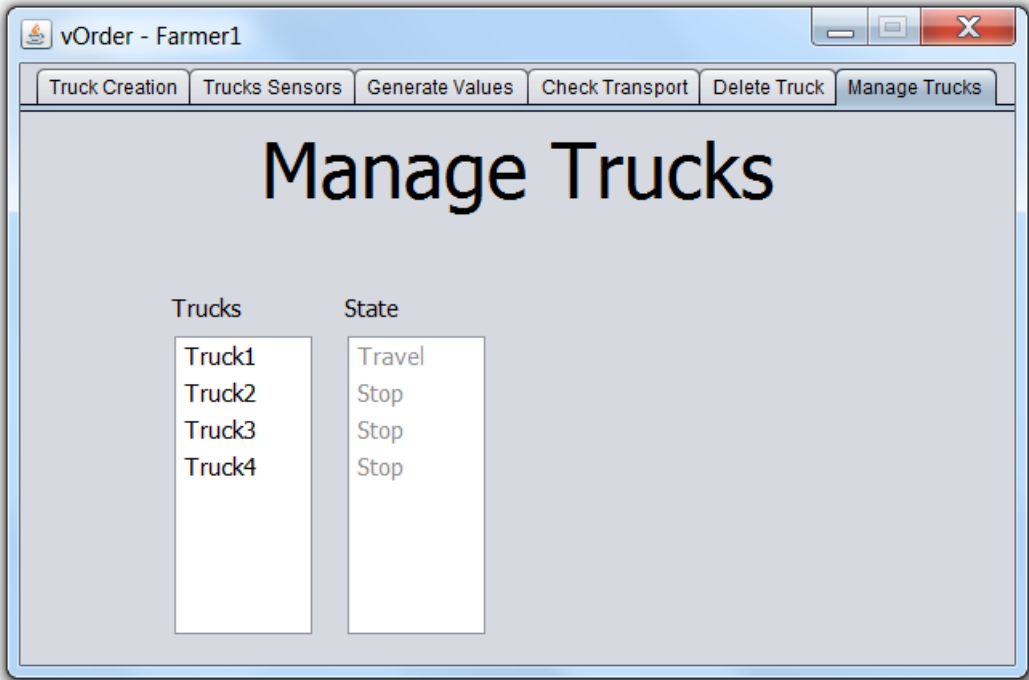


Figure 4.39 – Manage Trucks Tab

In this the tab, the farmer can check which of his trucks are currently traveling (State: Travel) and which are currently not assigned to any order (State: Stop). This tab has two distinct functions, if the selected truck current state is Stop a “Edit Sensors” button appears, allowing the user to change that truck’s sensors (depicted in Figure 4.41) if, however, the truck is currently traveling his sensors cannot be altered with and in this case this tab becomes the place where the farmer can perform the “truck arrived” function (pressing the “Arrived” button observable in Figure 4.40), setting the end of that truck’s trip.

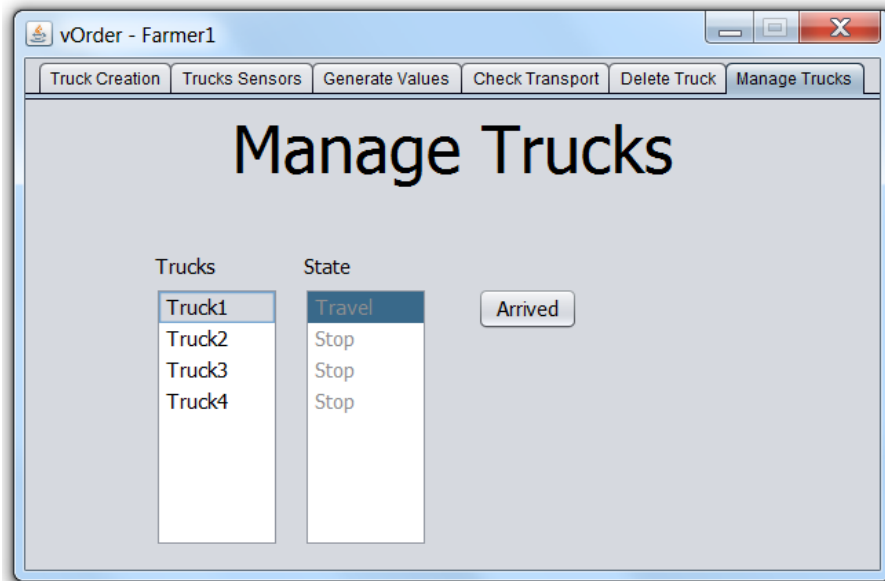


Figure 4.40 - Manage Trucks Tab: Arrived Button

As described in the previous chapter, the “Arrived” button present in Figure 4.40 allows the farmer to set the end of that truck’s trip. This will change that truck’s State to “Stop”, creating an end time/date for that trip, and making the truck’s sensors available to be altered (Figure 4.41) and the truck available to be used to deliver another Order.

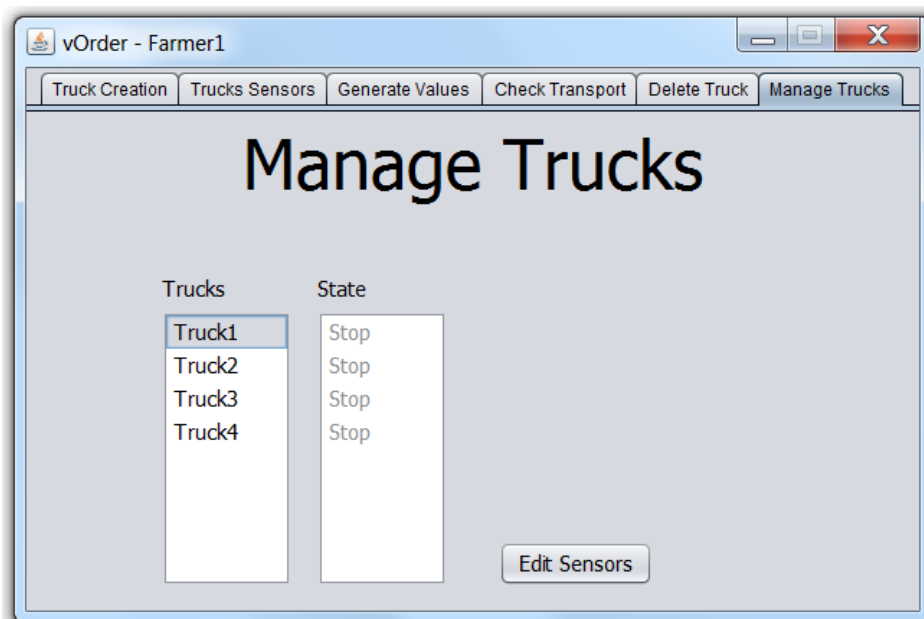


Figure 4.41 - Manage Trucks Tab: Edit Sensors Button

Chapter 4. Applications Demo

All the trucks listed whose state is “Stop” are allegeable for the user to edit their sensors. The truck’s sensors can be changed in one of two ways, first the user can change what type of sensor that sensor is (keeping the sensor - Sensor ID - change from a temperature sensor to a pressure sensor, etc.) or secondly, completely change the sensor that is present in that truck (by changing the sensor ID, always keeping in mind that every sensor in the market has a unique ID). That choice is presented to the user once a truck is selected and the “Edit Sensors” button is pressed, through the appearance of a new interface, like the one presented in Figure 4.42.

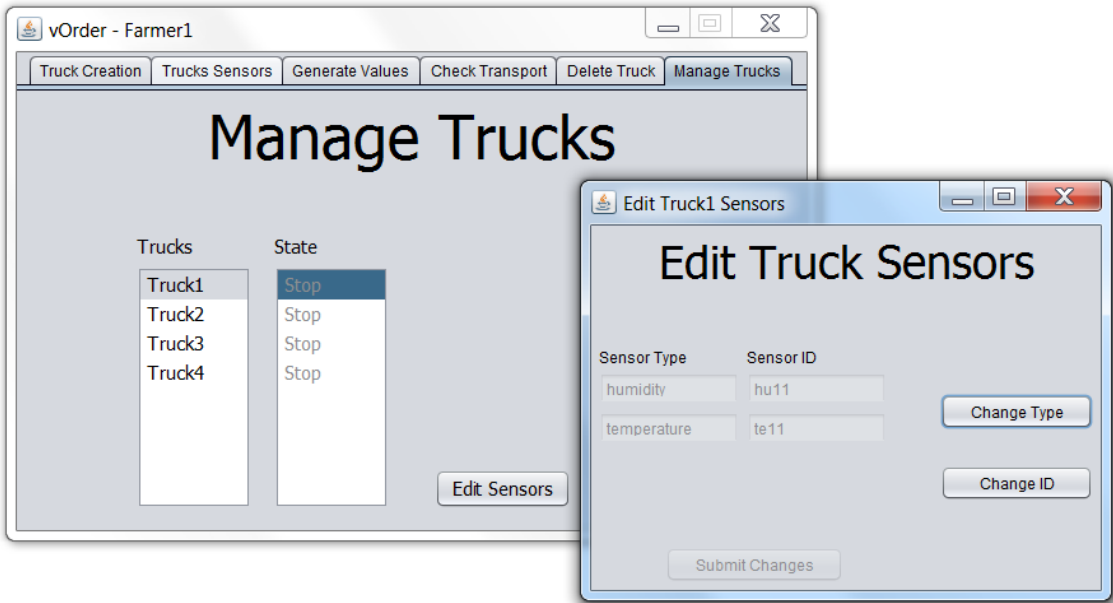


Figure 4.42 - Manage Trucks Tab: Edit Truck Sensors

In Figure 4.42 can be seen the two Edit Sensors choices presented to the user after the Edit Sensors button is pressed. All the “Sensor Type” and “Sensor ID” text fields are automatically filled with the current truck sensor’s types and IDs respectively. From here the user has two ways of proceeding, either by pressing the “Change Type” button or the “Change ID” button.

If the Change Type button is pressed all the Sensor Type fields turn enabled and so does the “Submit Changes” button, like presented in Figure 4.43.

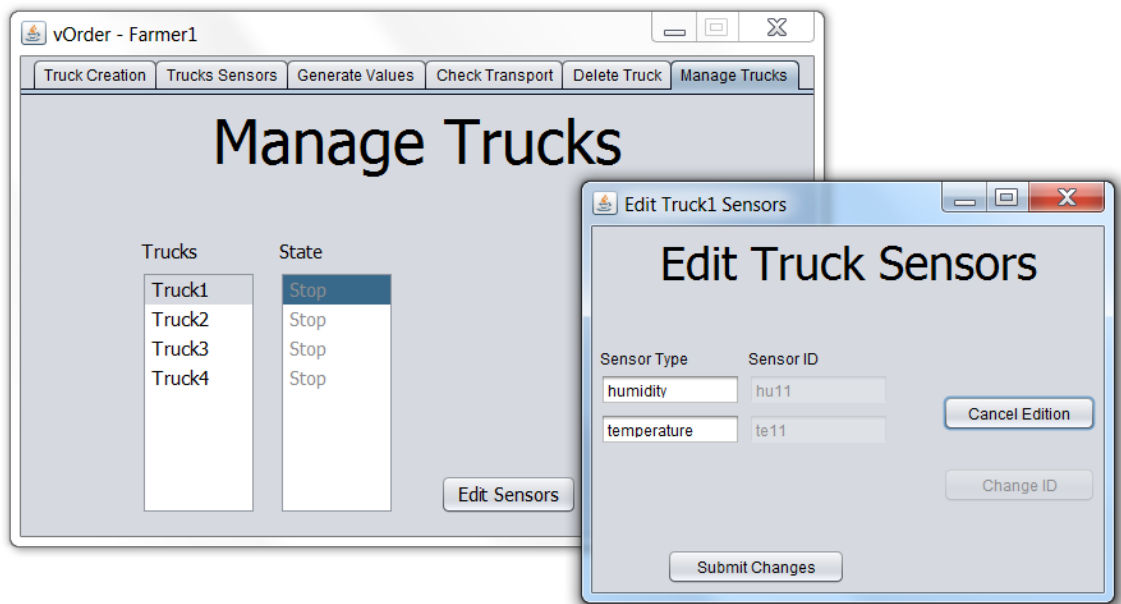


Figure 4.43 - Manage Trucks Tab: Edit Truck Sensors Type

Once the button (Change Type) is pressed the farmer becomes able to edit all the sensor types present in the selected truck, and after all the changes are made he just has to press the "Submit Changes" button and the application will automatically update that truck's sensors, and present the new types. If the farmer decides to undo the edition, hitting the "Cancel Edition" button reverses all the changes made, and the interface returns to the appearance it has before - Figure 4.42.

On the other hand, if the "Change ID" button is pressed, the Sensor ID fields will turn enable and the user will face an interface like the one depicted in Figure 4.44.

Chapter 4. Applications Demo

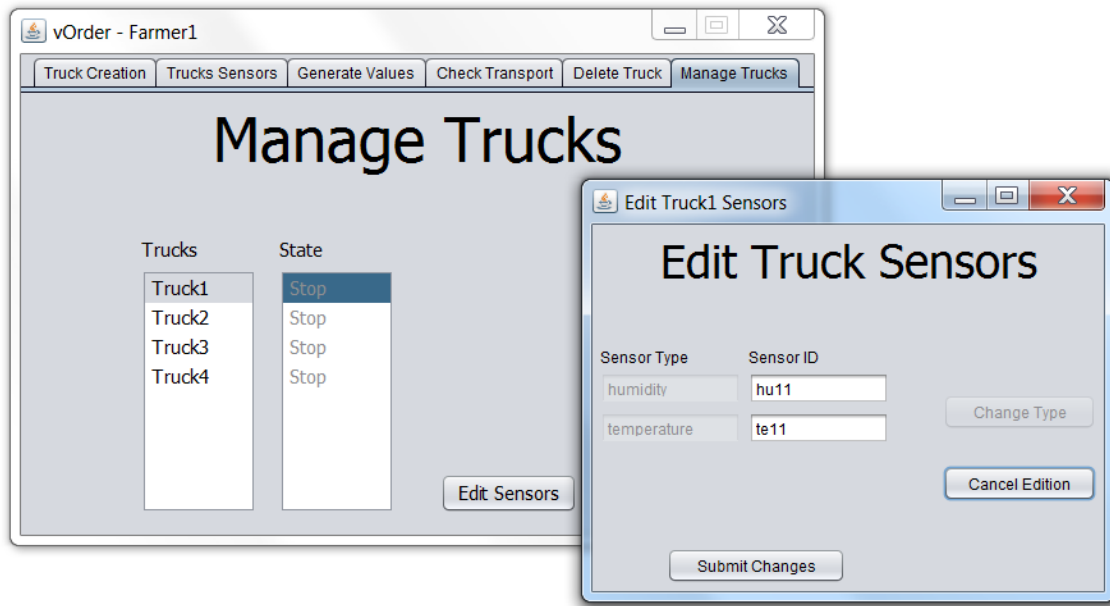


Figure 4.44 - Manage Trucks Tab: Edit Truck Sensors ID

Once the button (Change ID) is pressed the farmer becomes able to edit all the sensor IDs present in the selected truck, and after all the changes are made he just has to press the “Submit Changes” button. This time however the change isn’t as simple as before when only the names needed to be changed. Similarly to what happens when a truck is deleted, this “Change Sensor ID” comprehends the deletion of several components, however, this time, several others need to be created. In order to correctly change and truck sensor it is necessary for the application to delete that sensor subscription, and his entity, create a new entity and a new subscription and change the ID in the truck entity (where all that truck’s sensors information is stored). After these deletions and re-creations are made, the interface returns to its previous form, with the new sensors IDs displayed - Figure 4.42.

4.4.11 vfNegotiation – Choose User Interface

Similarly to what happens in the vOrder Application, the running of the vfNegotiation Application starts with the appearance of a “Choose User Interface” (Figure 4.45) where the user can choose to enter the application as a Farmer or as a Buyer.

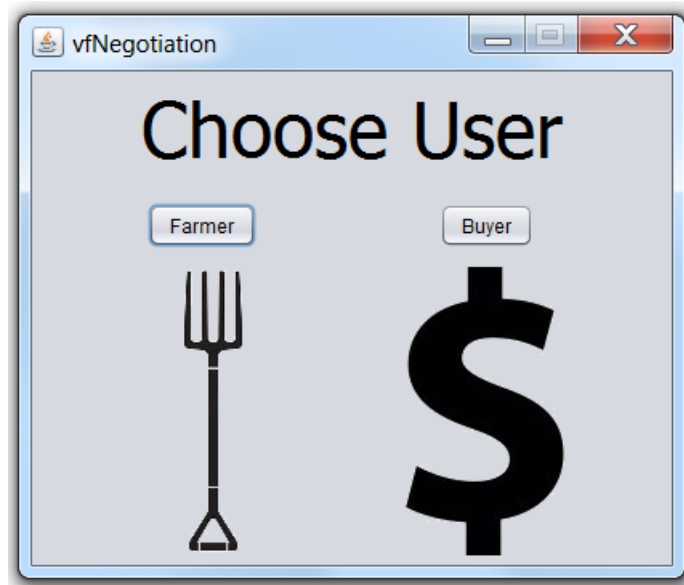


Figure 4.45 – vfNegotiation: Choose User Interface

By pressing either one of the buttons observable in Figure 1.42, the presented interface disappears going one of two different ways. If a buyer is accessing the application, the Buyer Interface is immediately presented (subchapter 4.4.18). If, on the other hand the application is being accessed by a Farmer, first he will have to state if he is a new farmer or an existent one in the “Farmer User” interface, similar to the one presented in the vOrder App, subchapter 4.4.12.

4.4.12 vfNegotiation – Farmer User Interface

Like mentioned in the previous subchapter, if a Farmer accesses the vfNegotiation, right after the “Choose User” interface he is presented with the “Farmer User” interface – Figure 4.46. Here, and just like it happened in the vOrder Application, the farmer user will need to provide his ID to “login” to his particular view of this App.

Chapter 4. Applications Demo



Figure 4.46 - vfNegotiation: Farmer User Interface

This interface allows the farmer to either login to an existent Farmer ID or to create a new Farmer ID and immediately login to that just created Farmer account.

Like the interface presented in subchapter 4.4.2, during the vOrder Application explanation, also this vfNegotiation interface presents some verification procedures. If either the New / Existent User options are left unchecked or the Farmer ID field is left empty when the “Proceed” button is pressed an error message like the one depicted in Figure 4.10 is shown. If the “New User” option is selected but the provided Farmer ID matches one that already exists in the cloud database, an error interface like the one present in Figure 4.11 is shown. Finally, if the “Existent User” option is selected but the provided Farmer ID is not yet present in the cloud database a fail message like the one observable in Figure 4.12 is shown.

By providing an inexistent Farmer ID and selecting the “New User” option, the application automatically creates a new Farmer Entity, adds it to the application database and presents the user with the Farmer Interface. On the other hand, if the “Existent User” option is selected and a Farmer ID that exists in the database is provided, the application will present the farmer with his particular view of the Farmer’s Interface.

4.4.13 vfNegotiation – Farmer Main Interface

After a farmer user successfully “logins” into an existent farmer ID or creates a new farmer account, he reaches the vfNegotiation Main Interface for Farmers, depicted in Figure 4.47.

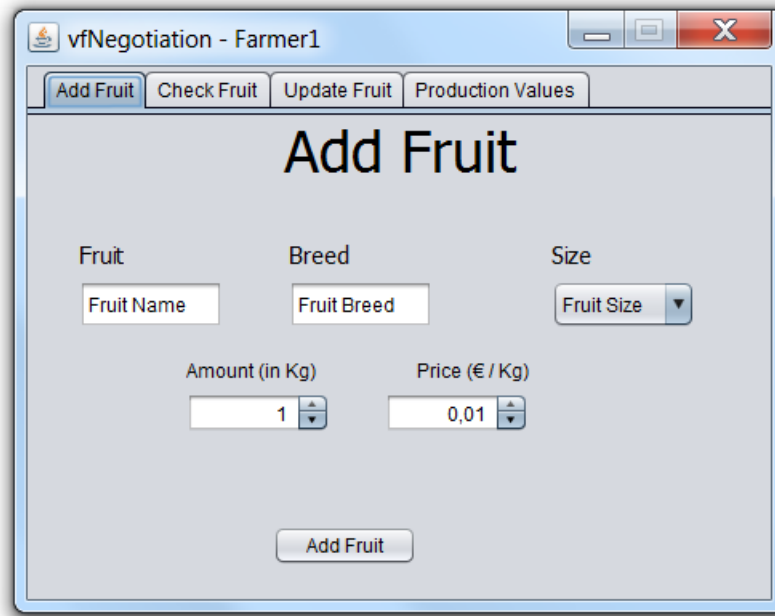


Figure 4.47 - vfNegotiation: Farmer Main Interface

This interface consists of four tabs each one providing the farmer with different functionalities. In the first tab, the one that is presented to the farmer as soon as he reaches this main interface, the farmer can add new fruits to his available fruit stock. In the second tab, the farmer can check all the fruit he currently has in his stock, and all the information regarding that fruit. Accessing his third available tab the farmer can update the set of information about any of the fruits he currently has in his stock. Finally, the fourth and last tab gives the farmer information about the amount of each fruit his last production provided. This tab uses the Fruit Production Entity populated by the vProductMon application, through the usage of IoT sensors that measure and count the produced fruit, providing the amount of produced fruit that falls in each size measure. This entity is the link between this application - vfNegotiation - and the vProductMon application, developed by another member of the workgroup doing the master thesis in the vf-OS project.

Chapter 4. Applications Demo

4.4.14 vfNegotiation – Farmer Main Interface: Add Fruit Tab

The first tab in the Farmer Main Interface, the one active when the interface first launches is the Add Fruit Tab. This tab is composed by several fields which the farmer needs to fill in order to add a new fruit to his available stock, Figure 4.47.

To add a new fruit to his stock the farmer will need to provide a combination of three values which does not yet exist in his stock, i.e. a combination of Fruit Name, Fruit Breed and Fruit Size. Besides the fruit information, for each combination of those three values, the farmer will have to set a price and an amount. It is worth mentioning that changing the size equals to the creation of a whole new different fruit, i.e. there can be different fruits with the same Name and same Breed with different Sizes.

Like every other interface in the applications, this too have some protections, like if for example, the farmer tries to add a fruit (Name plus Breed plus Size) that already exists in his stock, the application will display an error message stating so, like the one depicted in Figure 4.48.

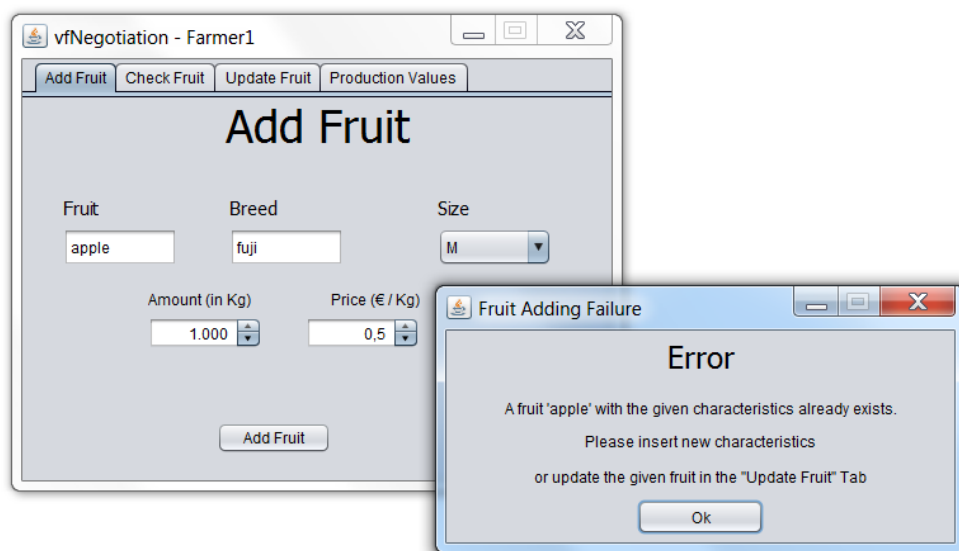


Figure 4.48 – Add Fruit Tab: Existent Fruit Error Message

However, if the farmer changes as much as the size, to a combination that does not yet exist in his stock, that fruit will be created and automatically added to his stock, with the provided price and amount, as can be observable in Figure 4.49.

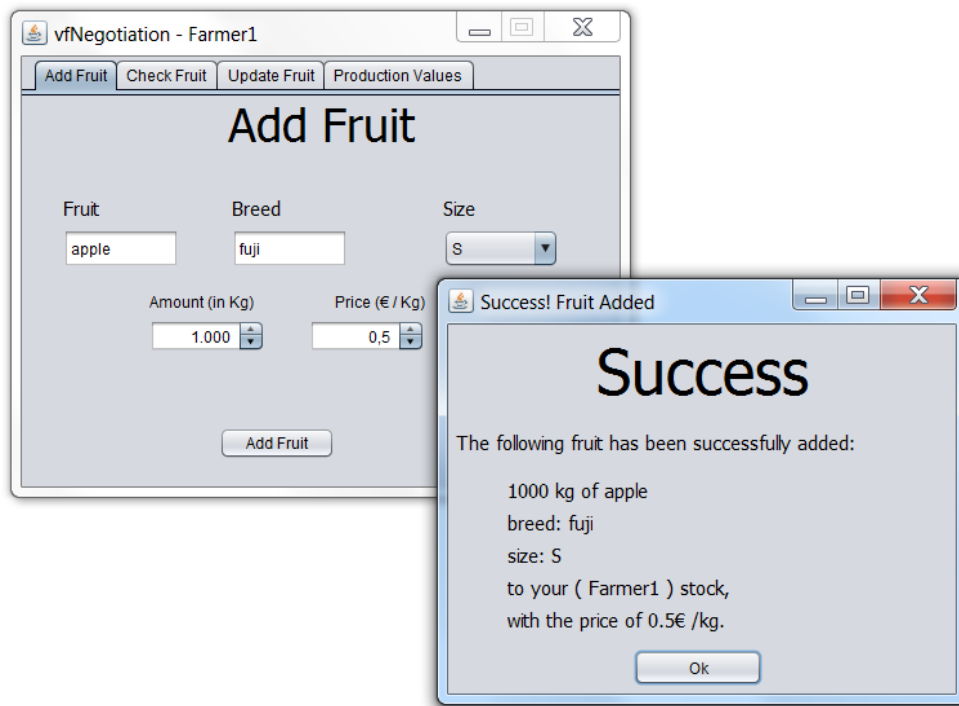


Figure 4.49 – Add Fruit Tab: Fruit Creation

Using this tab, the user can add as many different fruits as he want to his stock and they will immediately become available for any potential buyer to find when searching for a specific fruit.

Chapter 4. Applications Demo

4.4.15 vfNegotiation – Farmer Main Interface: Check Fruit Tab

The second tab accessible in the Farmer Main Interface is the Check Fruit Tab, Figure 4.50. Using this tab, the farmer can check at any moment all the fruit he has in stock, as well as its amount, its price and the total monetary value that that fruit represents.

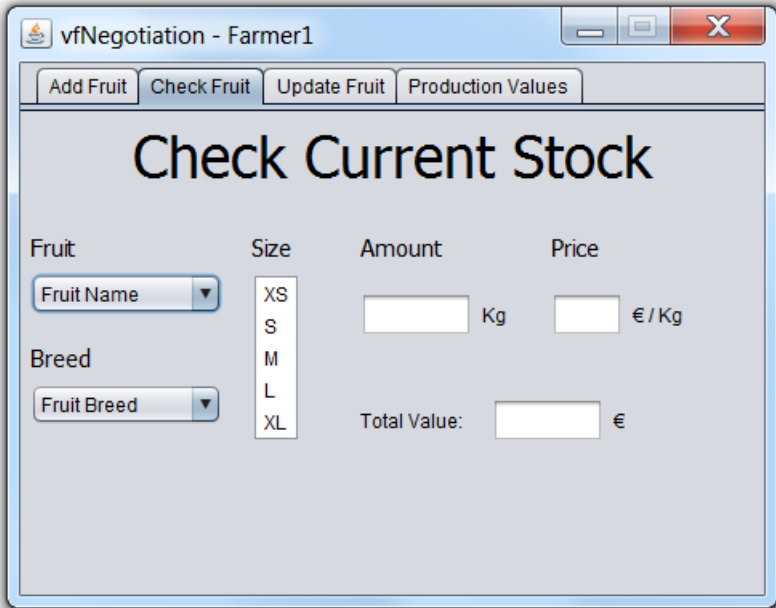


Figure 4.50 - Check Fruit Tab

When this tab is selected, the application automatically fills the “Fruit Name” list with all the fruit names of the fruits that are part of that farmer’s stock, as can be seen in Figure 4.51.

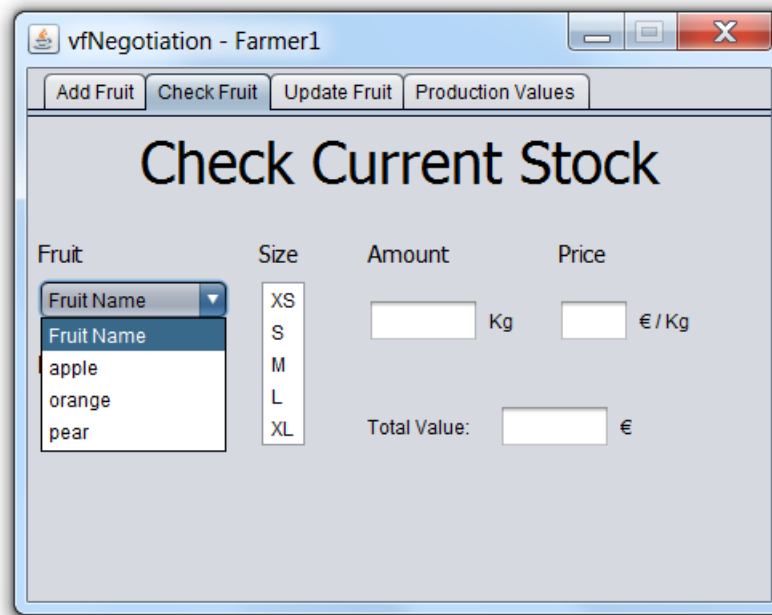


Figure 4.51 – Check Fruit Tab: Fruit Name List

When a Fruit Name is selected from the list, once again the application automatically fills the “Fruit Breed” list with all the different breeds of the given fruit that the farmer has in stock. For different fruit names, the fruit breed list differs, as shown in Figure 4.52.

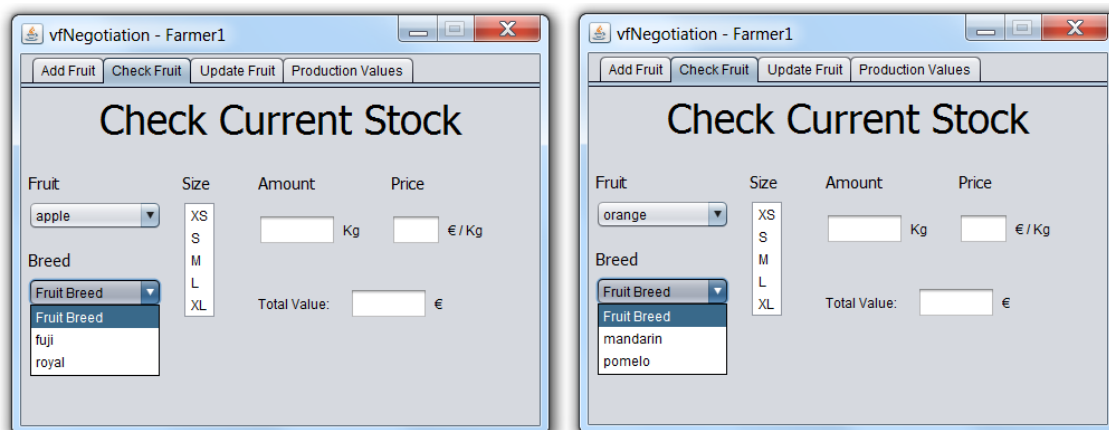


Figure 4.52 – Check Fruit Tab: Fruit Breed List

Chapter 4. Applications Demo

For the farmer to easily check all his fruits, he just has to select a fruit name and breed from the lists and then selecting the fruit size. As soon as a fruit size is selected the three fields (Amount, Price and Total Value) are automatically filled with the current fruit amount, fruit price and total monetary value of that fruit respectively, as depicted in Figure 4.53.

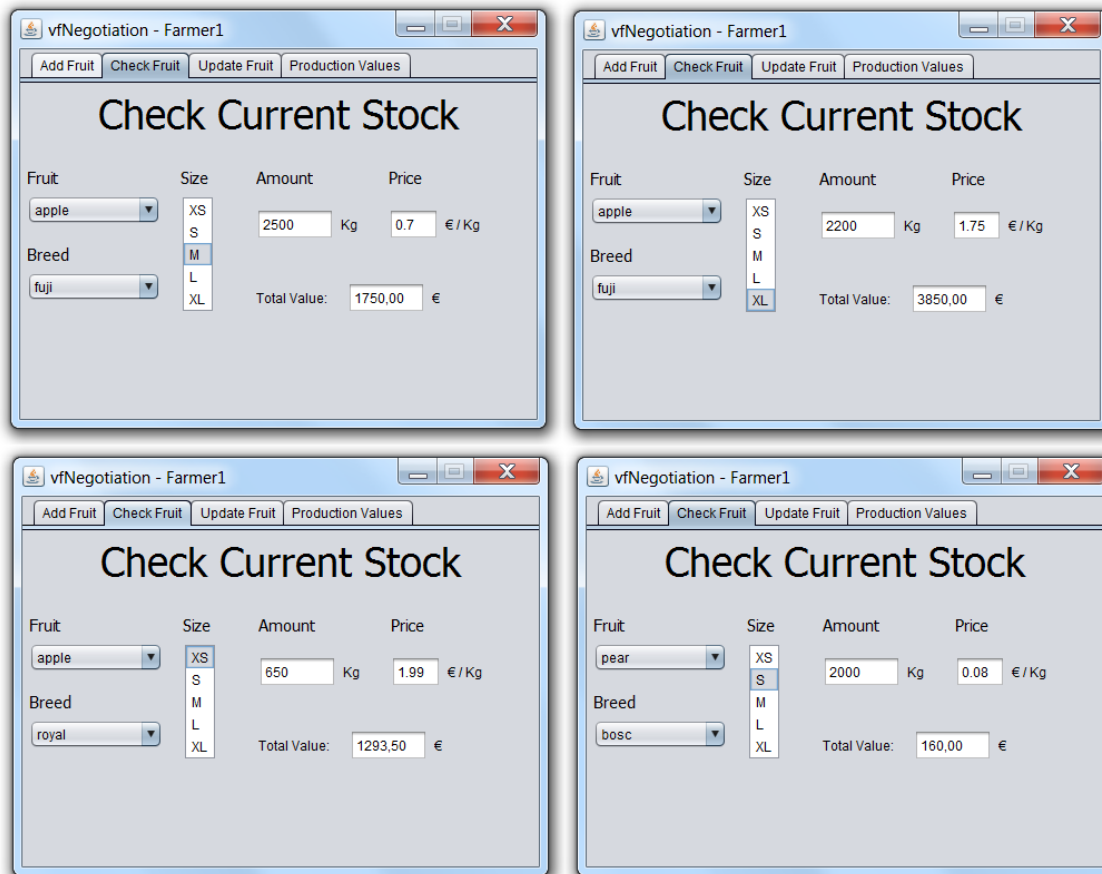


Figure 4.53 – Check Fruit Tab: Fruit Information Provided

In Figure 4.53, is observable an exhibition of different searches that can be done using this tab. The search can be done to find different fruits, different breeds within the same fruit or even different sizes within the same fruit breed.

If the search is done with a fruit name plus fruit breed plus fruit size combination that returns no fruit (i.e. that combination has never been added before), the application will still return the fields, but filled with 0. However, if the search focuses a fruit previously added but whose current amount is 0, the ap-

application will present 0 in the Amount and Total Value fields, but will still show that fruit's price in the Price field. Both these cases can be seen depicted in Figure 4.54.

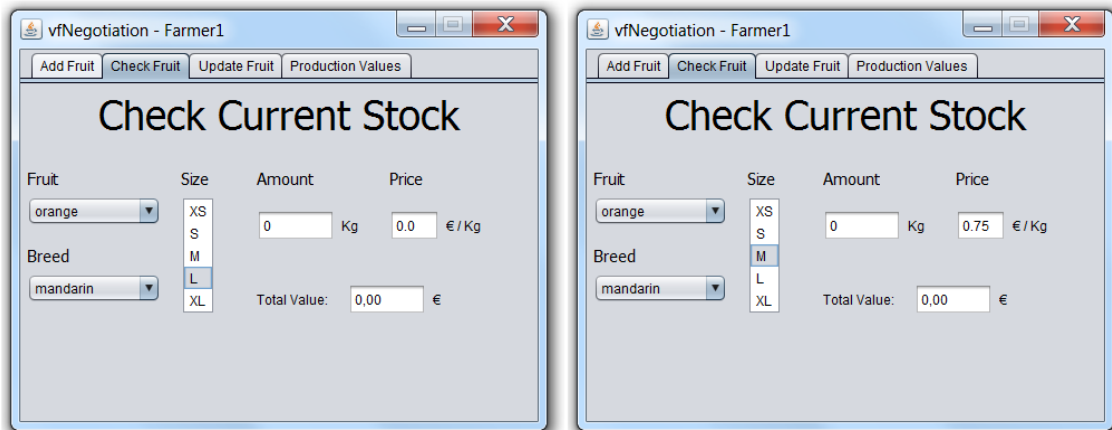


Figure 4.54 - Check Fruit Tab: 0's in the Information

4.4.16 vfNegotiation – Farmer Main Interface: Update Fruit Tab

The third accessible in the Farmer Main Interface is the Update Fruit Tab, Figure 4.55. This tab allows to farmer to update every bit of information regarding any of the fruits he has previously added to his stock.

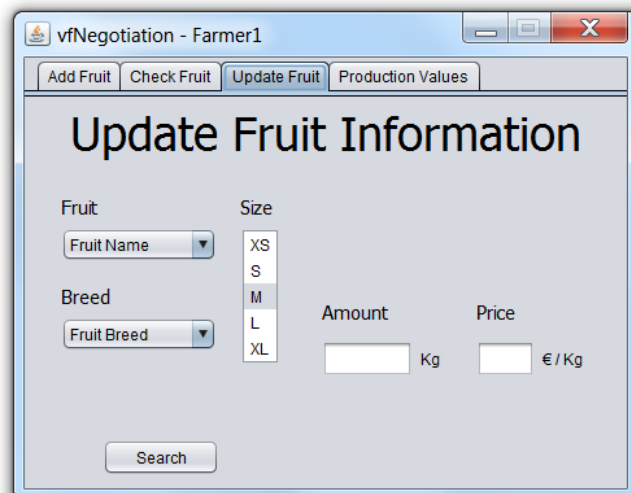


Figure 4.55 - Update Fruit Tab

Chapter 4. Applications Demo

In order to update the fruit he currently has in stock, he farmer will first need to search for that fruit, in a way very similarly to what needs to be done in the “Check Fruit” tab. The only difference from this tab to the previous one with regards to the search is the appearance of a “Search” button. Like the previous tab the Fruit Name list is automatically filled with all the fruits that farmer owns, and once a fruit is selected, the Fruit Breed list is also automatically filled. However, unlike the “Check Fruit” Tab where the user just needed to select an option from each field and the information about the selected fruit would automatically be presented in the white fields, in this tab after filling all the fields the farmer needs to press the “Search” button in order to see the returned information.

Since this tab makes use of the “Search” button, some verifications needed to be included in order to just return information when all the fields are correctly filled. This being said, if the Search button is pressed without selecting both the Fruit Name, Fruit Breed fields (since the Fruit Size field has a default selection automatically done), one of the two errors shown in Figure 4.56, will be presented to the user.



Figure 4.56 - Update Fruit Tab: Errors Interface

After all the fields are filled and the “Search” button is pressed both the Amount and Price fields will be filled with the selected fruit values (once again the Amount field will be 0 in case that that farmer currently has no stock of that fruit, and the Price field will be 0 if that fruit has never been added to that farmer’s stock before). An example of both a regular search and a search returning no values is shown in Figure 4.57.

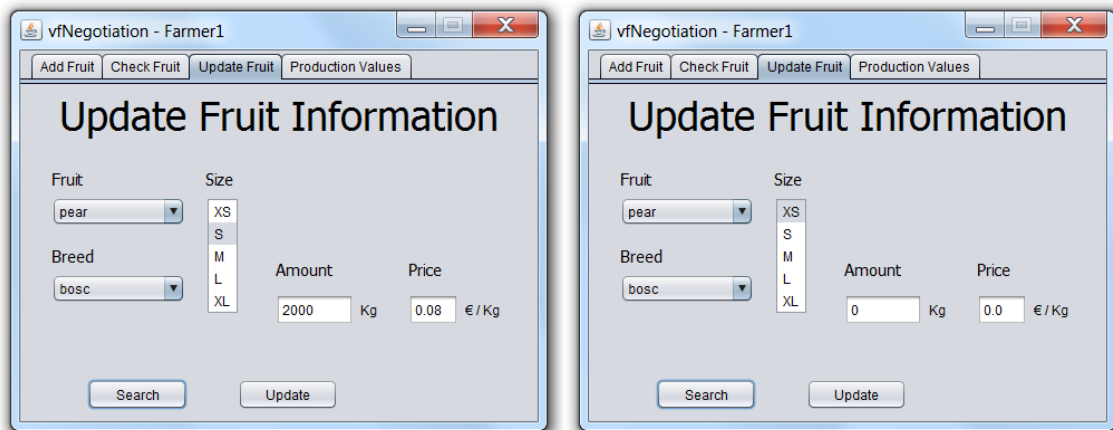


Figure 4.57 - Update Fruit Tab: Search Done

After the search has return values the farmer can update those values by pressing the “Update” button. Here another verification is made, the farmer is only supposed to be able to update the information of fruits he has previously added to the stock, so if in this tab he tries to “Update” a fruit never added before the application will return an error message like the one presented in Figure 4.58.

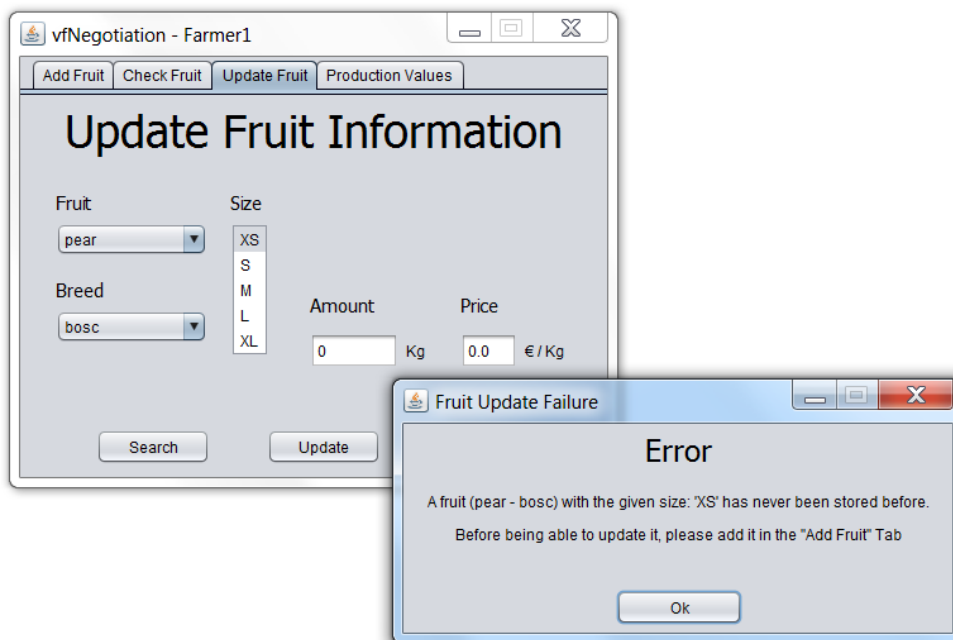


Figure 4.58 - Update Fruit Tab: Fruit Update Failure

Chapter 4. Applications Demo

After all the verifications are checked and the search returns a fruit previously added to the farmer's stock, the Update button slightly changes this tab's interface to allow the user to make the changes to that fruit information. In order for the changes to be performed the user is presented with the option of changing both that fruit amount and price, like depicted in Figure 4.59.

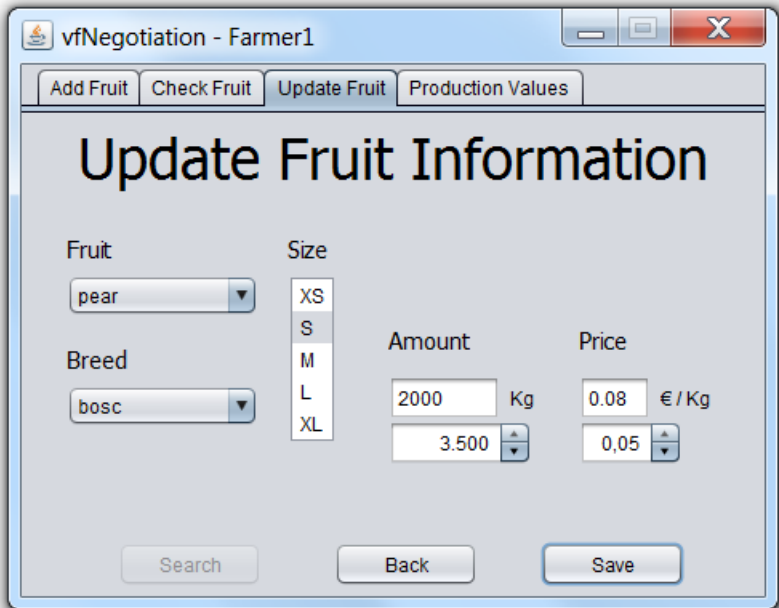


Figure 4.59 - Update Fruit Tab: Update Fields

During the fruit update, the current fruit information will remain static in the upper fields, while the lower fields can be freely changed by the user. At any point during the update, if the farmer decides to undo the changes he is making all he needs to do is press the "Back" button and nothing on that fruit's information will change. If, however, the farmer wants to proceed with the fruit update, by pressing the "Save" button that fruit's information will change, and to the farmer will be presented an interface showing the new information concerning that fruit. An example of that interface can be seen in Figure 4.60.

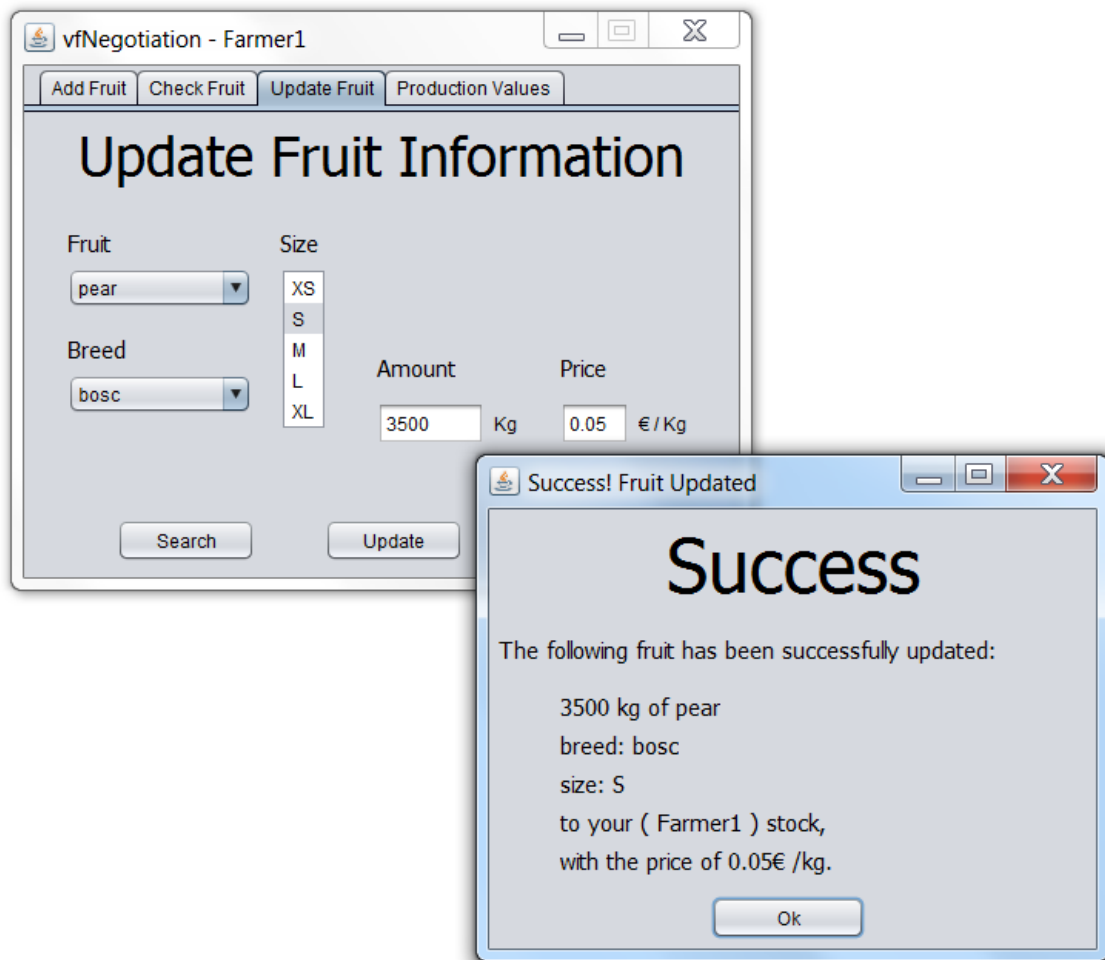


Figure 4.60 – Update Fruit Tab: Fruit Update Success

As can be observed in Figure 4.60, right after the Save button is pressed and the Success interface is shown, the “Update Fruit” tab will update the information fields to show the new values of that fruit.

4.4.17 vfNegotiation – Farmer Main Interface: Production Values Tab

The Production Values tab is the last tab accessible to the farmer in the Farmer Main Interface of the vfNegotiation app. This tab, observable in Figure 4.61, allows the farmer to examine the amount of fruit his last production gave him.

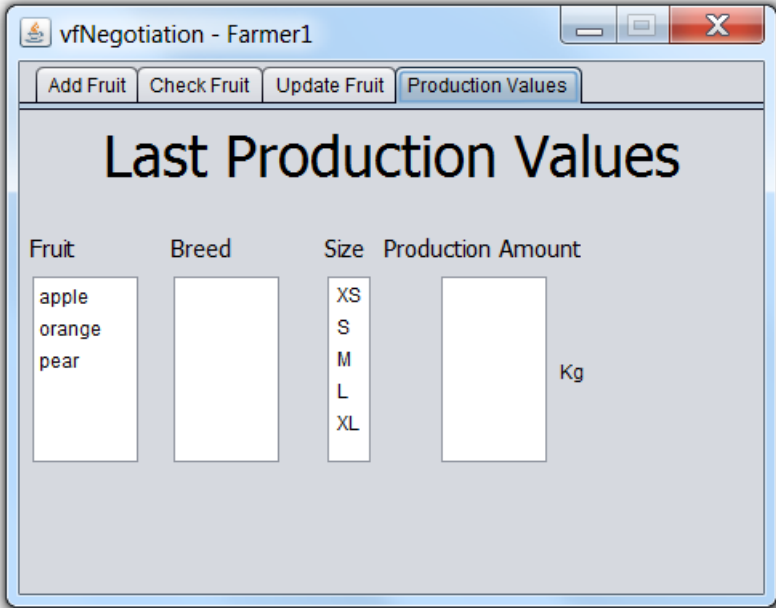


Figure 4.61 - Production Values Tab

As can be seen in Figure 4.61, this tab is composed by four lists, two of them where he can specify which food he wants to examine (“Fruit” and “Breed” lists), one where will he be presented the produced amounts (“Production Amount” list) and finally one list where he can select the final fruit’s information which will give the precise amount of the produced fruit.

As soon as the farmer enters this tab, the Fruit list will be presented automatically filled with all the fruits that farmer has produced (as can be seen in Figure 4.61, which is the image representing this tab as soon as it is reached).

Once the farmer selects any of the listed fruits, the “Breed” list will be filled with all the breeds of the selected fruit that that farmer has produced (as can be seen in the left image of Figure 4.62, which depicts the tab’s appearance once a Fruit is selected). After a Breed of the given fruit is selected, the “Production Amount” list is filled with the amount of fruit produced for each size (as can be seen in the right image of Figure 4.62). With the amount of produced fruit, listed to all sizes, selecting one size will select the according amount of the produced fruit, as exemplified in Figure 4.62.

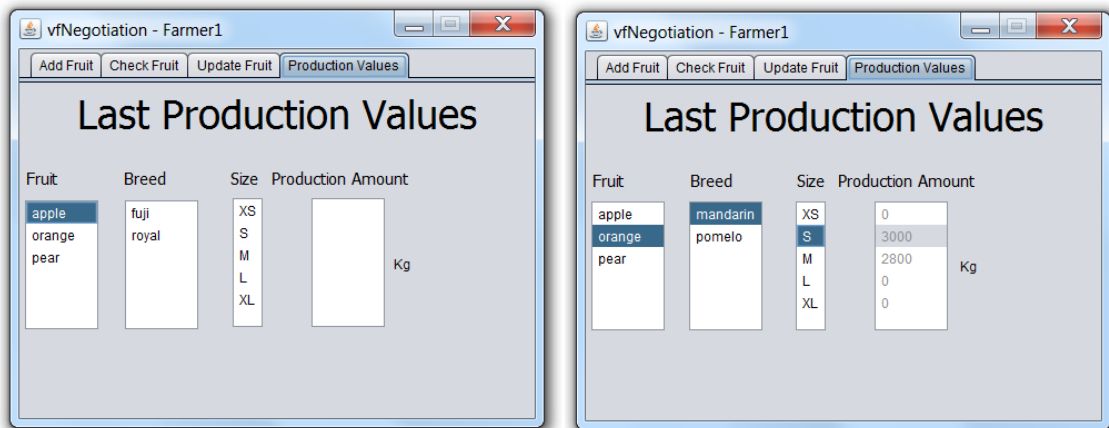


Figure 4.62 - Production Values Tab: Fruit's Last Production Values

4.4.18 vfNegotiation – Buyer Main Interface

The Buyer Main Interface is reached after the Choose User interface is left by clicking the “Buyer” button from Figure 4.45. Therefore, this is the interface presented to a Buyer user when using the vfNegotiation Application. This application allows the user (buyer) to search for farmers selling the type of fruit he intends to buy, and select them manually or automatically, using one of the two tabs observable in Figure 4.63. After a fruit and a farmer are selected the buyer can, still using this application, an Order to buy the wanted fruit from the selected farmer.

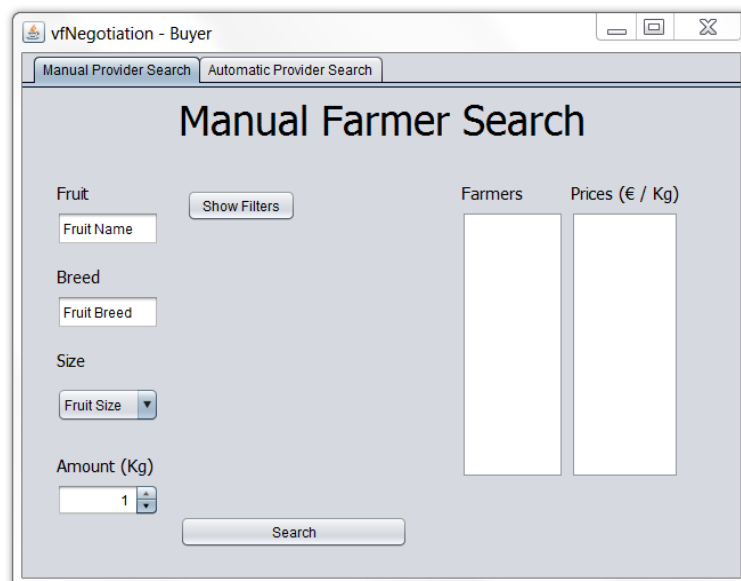


Figure 4.63 - vfNegotiation: Buyer Main Interface

Chapter 4. Applications Demo

4.4.19 vfNegotiation – Buyer Main Interface: Manual Search Tab

The Manual Search Tab is the tab the user reaches when accessing the vfNegotiation Application as a Buyer. Using this tab the buyer can manually search for all the farmers registered in this application, who currently have the searched food in stock.

The manual search can be done either by stating just the fruit the buyer is looking for (i.e. just by filling the Fruit Name, Fruit Breed, Fruit Size and Fruit Amount fields), or by adding some filters to the search. The available filters are presented to the user once he presses the “Show Filter” button. All the currently available filters can be seen in Figure 4.64.

Even though the search can be performed without adding any of the filters, if any of the mandatory fields is left empty, and like previously displayed in this document, the application will show an error interface alerting the user to which fields were left empty.



Figure 4.64 – Manual Search Tab: Available Filters

Each of the filters can be selected and deselected at any time during the search, changing the search's parameters. For each selected filter, a new field will appear in front of it where the user can set that filter's value. As can be seen in Figure 4.64, the vfNegotiation currently has four available filters for the Manual Search:

1. "Min Price" - which sets a minimum price for the farmer's selected fruit for him to appear in the search results, i.e. if a farmer's selected fruit price is above the minimum price set by the buyer he will not appear in the search results. This filter was implemented in order for the buyer to, theoretically, find better farmers regardless of the price;
2. "Max Price" - similar to the "Min Price" this filter focuses on the fruit's price, however using this filter the buyer can set the maximum fruit's price for the farmer to be showed in the search, i.e. the search will only return farmers whose selected fruit price is below the maximum price set. This filter was implemented for buyers wanting to find the farmers with the lowest prices;
3. "Min Produced Amount" - sets the minimum produced amount of the selected fruit a farmer must have, to appear in the search results, i.e. the farmer will only be returned in the search if his production stock of that fruit is above the minimum amount set. This filter was implemented to allow the buyer to search only for big producers, i.e. farmers which produce high amounts of that specific fruit;
4. "Max Produced Amount" - similar to the "Min Produced Amount" this filter affects the farmer's production stock of the selected fruit, however using this filter the buyer can set the maximum amount of fruit the farmer has produced. Unlike the previous one, this filter was implemented so that the farmer could search only for the small producers, i.e. he will be presented with a list of farmers which have produced small amounts of the selected fruit in stock.

It is worth mentioning that the last two presented filters were developed using the entity that establishes the connection between the vProductMon and the vfNefotiation apps. That entity holds the information automatically ob-

Chapter 4. Applications Demo

tained, through IoT sensors measurements, about the amount of produced fruit that falls into each category from a specific farmer. That information, can be queried by the farmer through the “Production Values” tab (subchapter 4.4.17), and is also made available to the buyer through the usage of these two filters, and others presented in subchapter 4.4.20.

Once a fruit is specified, and the wanted filters are set, by pressing the “Search” button the application will populate both the “Farmers” and the “Prices (€/Kg)” lists with the farmers who currently have the selected fruit in stock with an amount high enough to fulfil de order, and those farmers fruit’s prices respectively. A comparison of a search using and not using filters in the Manual Search can be seen in Figure 4.65.

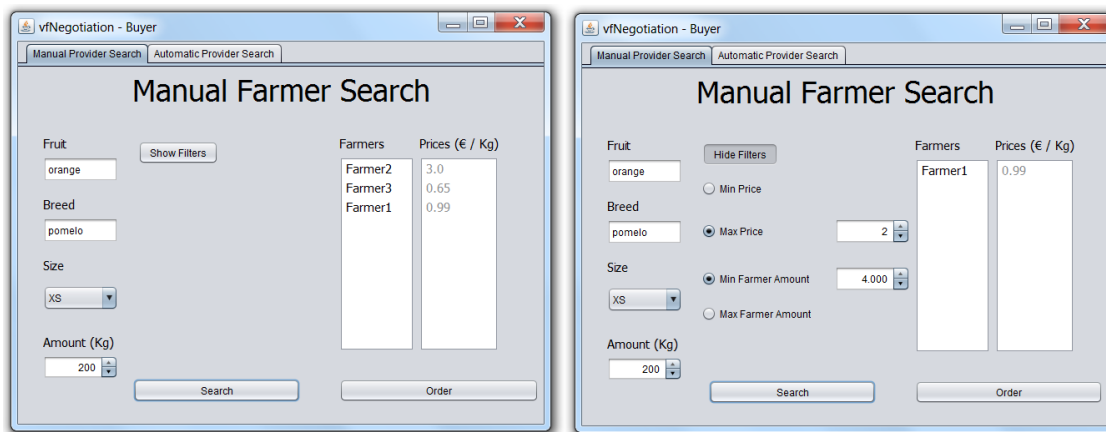


Figure 4.65 - Manual Search Tab: Filters and No Filters Results

As can be seen in Figure 4.65, the usage of filters will filter the search’s results, and more than one filter can be used simultaneously to further filter the results. If the filters are used in a way that makes the results return no farmers, i.e. any of the farmers registered in the application meet the necessary requirements to show up in the results, an error message like the one presented in Figure 4.66, will show up to the vfNegotiation user. The same message will appear even if no filters are used but no farmer currently has the searched fruit in stock.

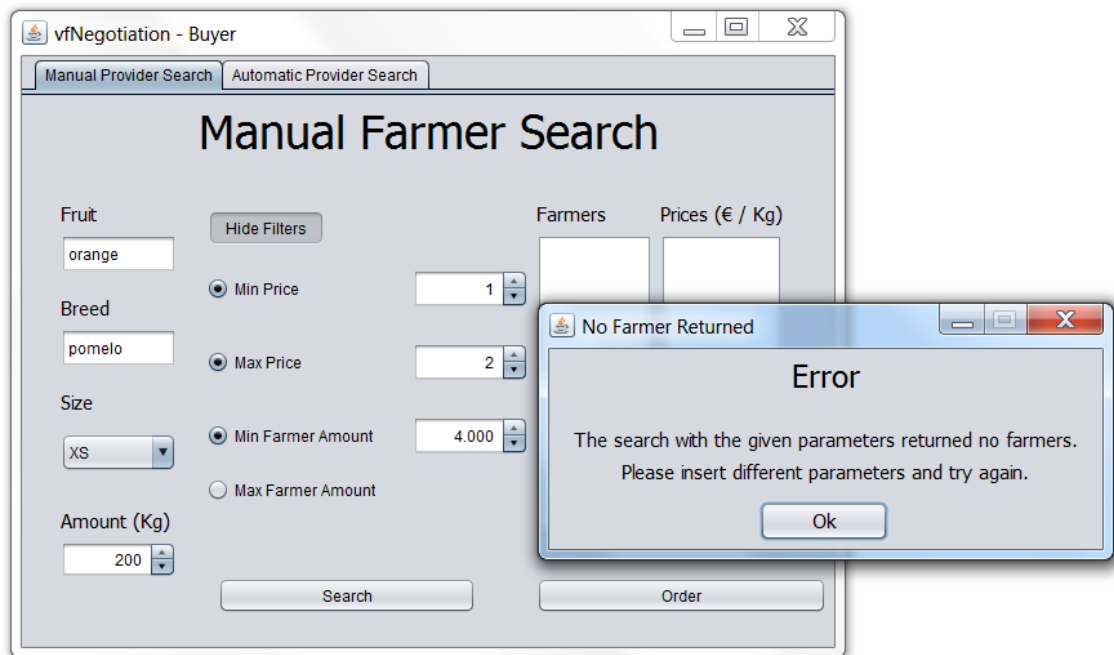


Figure 4.66 – Manual Search Tab: No Farmer Error Message

If the search returns at least one or more farmers, the buy can select one of them by clicking on his name in the “Farmers” list, and clicking on the “Order” button, an order draft will appear allowing the farmer to order the searched fruit from the selected buyer. This is order draft is approached on subchapter 4.4.21.

4.4.20 vfNegotiation – Buyer Main Interface: Automatic Search Tab

The second and last tab of this Buyer Main Interface is the Automatic Search Tab. Using this tab, the buyer can ask the application to automatically select the best farmer that provides a given fruit, according to the Buyer’s requirements. The interface presented to the Buyer when he first accesses the second tab of this interface is presented in Figure 4.67.

Chapter 4. Applications Demo



Figure 4.67 - vfNegotiation Buyer Interface: Automatic Search Tab

Once again, like happened with tab one of this interface, all the Fruit Name, Fruit Breed, Fruit Size and Fruit Amount fields must be filled before performing the search, or else the application will display an error message. However, unlike que previous tab, this tab's filter field is mandatory, i.e. the buyer must choose one of the possible filters to perform the search. The optional filters, shown in Figure 4.68, are used to inform the application's search function which parameter the buyer considers to be more important when buying a fruit from a producer.

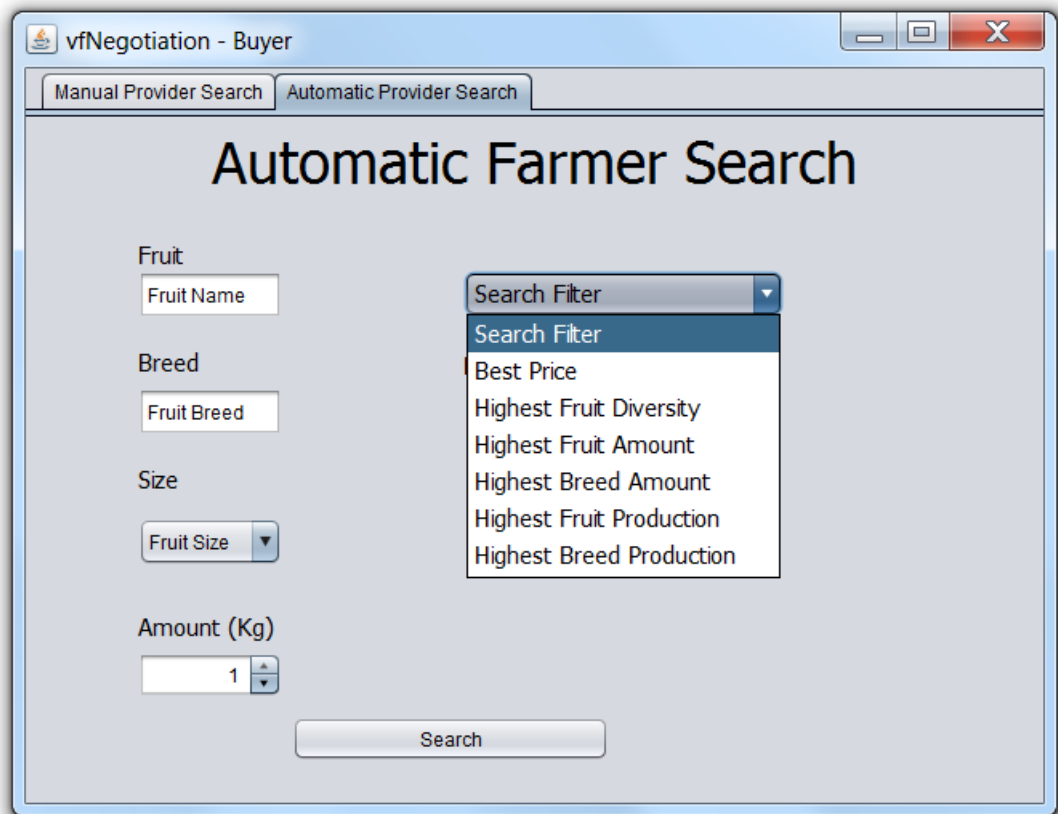


Figure 4.68 – Automatic Search Tab: Available Filters

At the moment, the Automatic Search has 6 optional filters:

1. “Best Price” – the application will search all the farmers who currently have enough of the searched food in stock to fulfil the buyer’s order, and return the one who has the lowest price / Kg of them. This filter was implemented for the buyer to get the lowest price possible for the selected fruit;
2. “Highest Fruit Diversity” – this filter will return the farmer which has enough of the searched fruit in stock to fulfil the order, but that also has the highest number of the fruit name, fruit breed and fruit size combination, i.e. the farmer that sells the highest variety of fruits. This filter was implemented for the buyer to find a farmer who sells many different fruits, in case for example, that the buyer wants to buy many different fruits always from the same farmer;

Chapter 4. Applications Demo

3. “Highest Fruit Amount” – this filter will force the search to return the farmer which has the highest current stock of the searched fruit. This filter was implemented for the buyer to know which farmer has the current highest amount of that fruit, in the case of wanting to make a big order of any breed and size within the selected fruit;
4. “Highest Breed Amount” – similar to the previous filter, this one makes the search return the farmer with the highest current stock of that the selected fruit’s breed. This filter was implemented for the buyer to know which farmer currently has the highest amount of that fruit’s breed, in the case of wanting to make a big order of any size within the selected fruit’s breed;
5. “Highest Fruit Production” – This filter looks for the farmer which has the highest production amount of the selected fruit, making use of the information present in the Fruit Production Entity (the entity establishing the connection between the vProductMon and the vfNegotiation applications). This filter was implemented for the buyer to know which farmer breeds the highest amount of that fruit, in the case of wanting to find the biggest producer of the selected fruit (regardless of breed and size);
6. “Highest Breed Production” – This filter focuses on the production amount of the selected breed. It makes the search return the farmer which has the highest production amount of that breed. This filter was implemented for the buyer to know which farmer breeds the highest amount of that fruit’s breed, in the case of wanting to find the biggest producer in the market, of the selected fruit’s breed.

Once a fruit, breed, size and amount are stated and the best fitting filter is selected the buyer can press the “Search” Button (remember that even any of the mentioned fields aren’t field at the time that the “Search” Button is pressed an error message will appear), and the application will return the farmer that meets all the requirements, as can be observed in Figure 4.69 (the same search as the one presented in Figure 4.65 was used in order to show the reliability of the application).

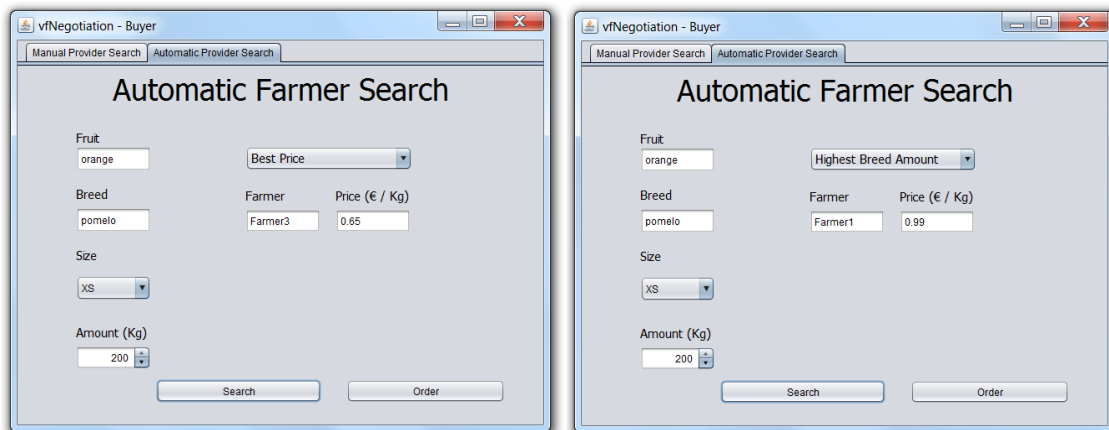


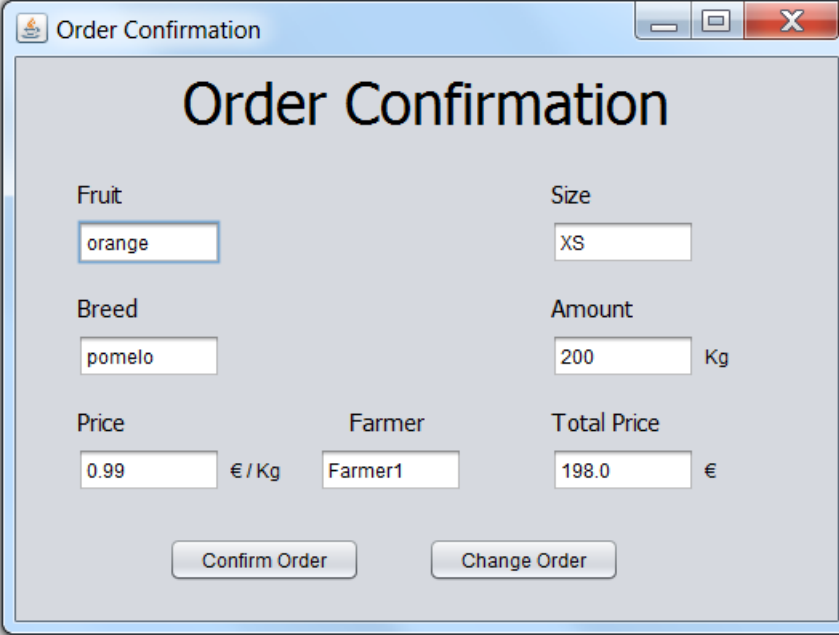
Figure 4.69 – Automatic Search Tab: Filters Results Comparison

As can be observed in Figure 4.69, unlike the Manual Search function used in tab one, this automatic search function returns only one farmer. This search is done based on the selected filter from the “Search Filter” field, and returns only the farmer that best meets the selected requirement. Once again and like in the “Manual Search” after the search returns a farmer, the buyer can order the goods from that farmer by pressing the “Order” button. This order is further explained in the next subchapter.

4.4.21 vfNegotiation – Buyer Main Interface: Order Draft

Once a fruit search is performed, by a buyer, in the vfNegotiation application, and returns one or more farmers, and one of those farmers is selected, the buyer has the option to create that Order, i.e. emit an order for the selected Farmer to dispatch the requested goods. In order to emit that Order the buyer needs to press the “Order” button present in both the Manual Search - Figure 4.65 – and the Automatic Search - Figure 4.69. Once that button is pressed, an interface similar to the one presented in Figure 4.70, will be presented to him.

Chapter 4. Applications Demo



The screenshot shows a window titled "Order Confirmation" with a light blue border and standard Windows window controls (minimize, maximize, close). The main content area has a light gray background and is titled "Order Confirmation" in large black font. Below the title, there are several input fields and labels arranged in a grid-like fashion. The fields contain the following text: "orange" (Fruit), "XS" (Size), "pomelo" (Breed), "200" (Amount), "0.99" (Price), "Farmer1" (Farmer), and "198.0" (Total Price). The units "€/Kg" and "Kg" are placed to the right of their respective fields. The "Total Price" field is followed by a "€" symbol. At the bottom of the dialog, there are two buttons: "Confirm Order" and "Change Order".

Figure 4.70 - vfNegotiation: Buyer Order Draft

In Figure 4.70 is depicted the Order draft presented to the buyer after he has pressed an Order button from either one of the searches available in the vfNegotiation application. In this draft, the buyer is presented with all the information about the order he is performing. Here he can see the fruit he is ordering as well as the breed, the size and the amount, and also the fruit's price, the farmer that will be providing the fruit and the total cost of the order. If the buyer agrees with the presented information, he just has to press the "Confirm Order" button and that order will be sent to the specified farmer (which will receive a "Dispatch Order" message similar to the one presented in the sub-chapter 4.4.4 of this document), if before sending out the order the buyer wants to make some changes, by pressing the "Change Order" button, the order send will be put on hold, this interface will be hidden and the buyer will return to the search tab that led him here.

It is worth mentioning that the farmer's stock present in the vfNegotiation App is immediately updated as soon as the order is emitted, i.e. to the current value is subtracted the value bought in this order. This was done to prevent new searches from taking into account an outdated stock value, because even though the bought goods have not yet been dispatched (only the farmer using

the vOrder application can really send them) they can no longer be available for sale since they are already destined to a buyer.

After hitting the “Confirm Order” button, the order will be automatically sent to the selected farmer and the buyer will be given an “Order ID”, depicted in Figure 4.71, which he will need to save in order to be able check the Transport Conditions of his purchased goods in the vOrder “Check Transport” Tab, like it was introduced in subchapter 4.4.8, Figure 4.31.



Figure 4.71 - vfNegotiation: Buyer Order ID Reminder Message

Chapter 4. Applications Demo

5 Conclusions and Future Work

After completed the development of two applications which provide solutions not only to the presented real case scenario but also to the vf-OS Project, this chapter holds the conclusions of such development as well as the future work that could still be developed in order to add further functionalities to both the scenario and the vf-OS Project.

5.1 Conclusions

The goal of this thesis was the development of two of the five interconnected applications which aimed to provide solutions primarily to a real-life scenario which, in turn also provided small contributions to the European vf-OS project being developed, among other institutions, by the UNINOVA institute. This thesis was therefore developed under the supervision and guidance provided by UNINOVA, in order to follow the vf-OS Project patterns and requirements.

The primary goal of this thesis was the usage of cutting-edge technologies, alongside European technologies to, more easily allow the creation of a standard application able to be used by all industrial and manufacturing sectors, which was the aim of the vf-OS project “the goal of the vf-OS Project is to develop an Open Operating System for Virtual Factories, which aims to become the reference system software for managing factory related computer hardware

Chapter 5. Conclusions

and software resources and providing common services for factory computational programs". In order to do so, the FIWARE Program was the main repository searched to find such technologies. As stated in this document, besides the java programming language, all the other technologies used where open source ones retrieved from the FIWARE Program, such as the Generic Enablers use, the Orion Context Broker and the Short Term Historic - Comet.

Even though the two developed applications were mainly oriented to the solution of the food chain scenario presented, they can easily be changed to answer any other scenario that requires the production, selling and transporting of goods. Also, several different user-friendly interfaces were produced, to allow any person to use the applications, and not just for people who are knowledgeable in the field. Any person wanting to sell his produced goods, who likes to keep track of every step of the process using IoT sensors, can use the presented applications, and all he needs to know is the ID of the sensors he is using and the goods he wants to sell, and the application will automatically take care of the rest. In addition to the user-friendly interfaces, several protections were added to the applications, as described during this thesis document, to make sure that even if the user fails to add certain information or adds it in a wrong way, the application will never be corrupted. The applications can also withstand not only a large number of users using the framework at the same time, but also large amount of registered sensors, goods and any other vital information, such as transport means, orders of goods or shared information between applications. All these aspects make the applications effortlessly scalable, interoperable, fault-tolerant and user-friendly.

For development and tests purposes, both the applications can work independently of the other applications developed by the master thesis work group, since arrangements were made to simulate the information that should be provided by the other applications. However, with all the applications fully deployed, there is a share of information, resources and functionalities between these two described applications and all the others developed within the vf-OS master thesis group project. The shared information goes from simply texts and values produced in one application and used in a different one, up to the subscription, registration and discovery of physical IoT sensors deployed in one application and accessed through a different one.

In addition to the applications developed during the writing of this master thesis and the creation of its prototypes, also two papers were also created. One of them has already been published in a conference, and the second one has already been submitted and is waiting to be published.

The first paper: Diogo Ferreira, Pedro Corista, João Gião, Sudeep Ghimire, João Sarraipa and Ricardo Jardim-Gonçalves (2017). "Towards Smart Agriculture using FIWARE Enablers", was already accepted and published in the ICE/IEEE Conference 2017, and depicts an intermediate state of this thesis development, after the preliminary search was made, and before the prototype of the applications was built.

The second paper: Pedro Corista, Diogo Ferreira, João Gião, João Sarraipa and Ricardo Jardim-Gonçalves (2018). "An IoT Agriculture System using FIWARE", was submitted in the 24th ICE/IEEE ITMC 2018 Conference, and is waiting for approval. This second paper includes a description of four of the five interconnected applications created within the Master Thesis workgroup working under the vf-OS projects and depicts the relations established between them.

With the development of this master thesis, it became clear that all the industries (whether they are factories, or manufacture industries or even agriculture industries) have only to gain, with the usage of IoT devices. This project showed many ways on how IoT sensors and other devices could greatly help all the steps, in this case, of a food supply chain, but that could easily be adapted to any other industry sector. That said, it is possible to say that an IoT Service Oriented System would be a great asset, to be added to the industry of the Factories of the Future, and this project proves that it can be done quite easily given the proper tools. Hence, by following the presented research question "How can a framework provide guidance to make IoT services discovered for effective use?", and making use of the consequent created Hypothesis "If the FIWARE technology can provide modularity and discovery solutions then integrate IoT devices through generic enablers will facilitate IoT service oriented implementation and use on manufacturing systems.", this thesis was able to prove that using the generic enablers provided by the FIWARE technology it was possible

Chapter 5. Future Work

to use IoT discovered services to provide contributions to the Factories of the Future service oriented manufacturing systems.

5.2 Future Work

As mentioned during the elaboration of this thesis' document, both the applications developed during this thesis, were created based on the presented scenario. Therefore, in a future work, they could both be easily adapted to the selling and transporting of any other products, with just minor changes to the interfaces. If any other industry wants to adapt these applications to its products, they will still be able to work over the same source code and only entity's and label's names need to be changed.

Besides, both the developed applications were made mainly from a seller/producer perspective, i.e. they were designed so that the core user of the applications was the seller/producer, both to put his goods for sale and to manage its production and expedition. The vf-OS project was designed to be an Operative System used by goods producers, and that was why these applications were developed according to that idea. If, however, using the same technologies, the applications were required to be done mainly from a buyer perspective, the vfNegotiation (that handles the selling and buying of goods), especially, could, and should be changed a bit. An idea on how to change the vfNegotiation to a buyer oriented perspective, easily achievable using the same technologies could be the following:

1. Instead of being the producers to sign up in the applications it could be the buyers;
2. Once all the buyers were signed in the system, a producer wanting to sell its goods, should inform all the signed buyers that he had a new product ready for sale, and the correspondent amount;
3. After receiving the information about the new product in their interface, each of the signed buyers could make an offer saying how much

of the product they would like to buy, and how much they were willing to pay for it;

4. After collecting all the propositions, the seller would choose the offer or offers that that best suited him and would then sell the produced goods to the respective buyer or buyers.

In the vOrder application, where allegedly is the producer that owns and controls the transport means, this could also be changed to make the buyers the owners of the transport carrying the goods, or even a third party being in charge of all the transportations.

Bibliography

- Atzori, L., Iera, A., Atzori, L., Iera, A., & Morabito, G. (2010). The Internet of Things: A Survey The Internet of Things: A survey. *COMPUTER NETWORKS*, (November 2014). <https://doi.org/10.1016/j.comnet.2010.05.010>
- Auto-ID Labs. (n.d.). Auto-ID Labs. Retrieved January 17, 2017, from http://autoidlabs.org/wordpress_website/
- Bauer, M., Martinbauerneclabeu, E., & Meissner, S. (2011). Service Modelling for the Internet of Things. In *Proceedings of the Federated Conference on Computer Science and Information Systems* (pp. 949–955).
- Consulting, F. (2015). Connect and Protect : The Importance Of Security And Identity Access Management For Connected Devices, (August).
- Crouch, C. (2015). TELEFONICA, ORANGE, ENGINEERING and ATOS join forces to push common standards for Smart Cities based on the FIWARE platform. Retrieved November 14, 2016, from https://atos.net/en/2015/press-release/general-press-releases_2015_03_03/pr-2015_03_03_01?utm_source=%2Fen-us%2Fhome%2Fwe-are%2Fnews%2Fpress-release%2F2015%2Fpr-2015_03_03_01.html&utm_medium=301
- Docker. (2017). Docker. Retrieved May 5, 2017, from <https://www.docker.com/>
- Duan, Q., Yan, Y., & Vasilakos, A. V. (2012). A Survey on Service-Oriented Network Virtualization Toward Convergence of Networking and Cloud Computing. *IEEE Transactions on Network and Service Management*, 9(4), 373–392. <https://doi.org/10.1109/TNSM.2012.113012.120310>
- EVERYTHING. (n.d.). EVERYTHING. Retrieved January 24, 2017, from

Bibliography

- <https://evrythng.com/>
- Fiware.org. (2016). FIWARE - About us. Retrieved November 14, 2016, from <https://www.fiware.org/about-us/>
- FIWARE - Orion Context Broker. (2014). FIWARE - Orion Context Broker. Retrieved May 5, 2017, from <https://pt.slideshare.net/fermingalan/introduction-to-fiware-cloud-context-broker?nomobile=true>
- Ford, H. (1922). My Life and Work. *Wikipedia*, 116.
- Guinard, D. (2010). Mashing Up Your Web-Enabled Home. In F. Daniel & F. M. Facca (Eds.), *Current Trends in Web Engineering: 10th International Conference on Web Engineering ICWE 2010 Workshops, Vienna, Austria, July 2010, Revised Selected Papers* (pp. 442–446). inbook, Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-16985-4_42
- Guinard, D., Member, S., Trifa, V., & Member, S. (2010). Interacting with the SOA-Based Internet of Things : Discovery , Query , Selection , and On-Demand Provisioning of Web Services, *3*(3), 223–235.
- Herauld, L., & Presser, M. (2008). *SENSEI - Integrating the Physical with the Digital World of the Network of the Future*.
- Herrmann, C., Schmidt, C., Kurle, D., Blume, S., & Thiede, S. (2014). Sustainability in manufacturing and factories of the future. *International Journal of Precision Engineering and Manufacturing - Green Technology*, *1*(4), 283–292. <https://doi.org/10.1007/s40684-014-0034-z>
- Hu, S. J. (2013). Evolving paradigms of manufacturing: From mass production to mass customization and personalization. *Procedia CIRP*, *7*, 3–8. <https://doi.org/10.1016/j.procir.2013.05.002>
- Hu, S. J., Ko, J., Weyand, L., Elmaraghy, H. A., Lien, T. K., Koren, Y., ... Shpitalni, M. (2011). Assembly system design and operations for product variety. *CIRP Annals - Manufacturing Technology*, *60*(2), 715–733. <https://doi.org/10.1016/j.cirp.2011.05.004>
- IEC.ch, I. E. C. (2017). Standardization Management Board SG 8. Retrieved January 17, 2017, from http://www.iec.ch/dyn/www/f?p=103:85:0:::FSP_ORG_ID,FSP_LANG_ID:11072,25
- IEE, S. A. (2016). Standard for an Architectural Framework for the Internet of Things (IoT) IEEE P2413, (September).
- IIC.org, I. I. C. (n.d.). Industrial Internet Consortium. Retrieved January 17, 2017, from <http://www.iiconsortium.org/index.htm>

- Im, J., Kim, S., & Kim, D. (2013). IoT Mashup as a Service: Cloud-based Mashup Service for the Internet of Things. In *2013 IEEE International Conference on Services Computing* (pp. 462–469). Washington, DC, USA: IEEE Computer Society. <https://doi.org/10.1109/SCC.2013.68>
- Jeschke, S., Brecher, C., Meisen, T., Özdemir, D., & Eschert, T. (2017). Industrial Internet of Things and Cyber Manufacturing Systems. In S. Jeschke, C. Brecher, H. Song, & D. B. Rawat (Eds.), *Industrial Internet of Things: Cybermanufacturing Systems* (pp. 3–19). inbook, Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-42559-7_1
- Jung, J., Watson, K., & Usländer, T. (2017). Design of Smart Factory Web Services Based on the Industrial Internet of Things, (Iic), 5941–5946.
- Karnouskos, S., Baecker, O., & S, L. M. (2007). Integration of SOA-ready Networked Embedded Devices in Enterprise Systems via a Cross-Layered Web Service Infrastructure.
- keit.re.kr. (n.d.). Korea Evaluation Institute of Industrial Technology. Retrieved from <http://www.keit.re.kr/eng/index.do>
- Kernel Exokernel. (2013). Kernel Exokernel. Retrieved January 23, 2017, from [https://commons.wikimedia.org/wiki/File:Exokernel_revised\(english\).png](https://commons.wikimedia.org/wiki/File:Exokernel_revised(english).png)
- Kernel Hybrid. (2008). Kernel Hybrid. Retrieved January 23, 2017, from <https://commons.wikimedia.org/wiki/File:Kernel-hybrid2.svg?uselang=pt>
- Kernel Layout. (2008). Kernel Layout. Retrieved January 23, 2017, from https://commons.wikimedia.org/wiki/File:Kernel_Layout.svg?uselang=pt
- Kernel Microkernel. (2008). Kernel Microkernel. Retrieved January 23, 2017, from <https://commons.wikimedia.org/wiki/File:Kernel-microkernel2.svg?uselang=pt>
- Kernel Monolithic. (2008). Kernel Monolithic. Retrieved January 23, 2017, from https://commons.wikimedia.org/wiki/File:Kernel-monolithic_v2.svg?uselang=pt
- Kim, J., & Lee, J. (2014). OpenIoT : An Open Service Framework for the Internet of Things. In *2014 IEEE World Forum on Internet of Things (WF-IoT)* (pp. 89–93). <https://doi.org/10.1109/WF-IoT.2014.6803126>
- Koschmider, A., Torres, V., & Pelechano, V. (2009). Elucidating the Mashup Hype: Definition , Challenges , Methodical Guide and Tools for Mashups. In *Proceedings of the 2nd Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009) held in conjunction with 18th International World Wide Web Conference (WWW 2009), April 20th, 2009, Madrid, Spain* (p. 8).

Bibliography

- Lake, D., Rayes, A., & Morrow, M. (2012). The internet of things. *The Internet Protocol Journal*, 15(3), 10–19. Retrieved from <http://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-57/153-internet.html>
- Liu, J., Liu, J., & Chao, L. (2007). Design and Implementation of an Extended UDDI Registration Center for Web Service Graph. In IEEE Computer Society Press (Ed.), *IEEE International Conference on Web Services (ICWS 2007)* (pp. 1174–1175). Salt Lake City, Utah, USA. <https://doi.org/10.1109/ICWS.2007.74>
- LogMeIn. (n.d.). xively. Retrieved January 24, 2017, from https://www.xively.com/?from_cosm=true
- Lueth, K. L. (2015). Will the industrial internet disrupt the smart factory of the future? Retrieved November 8, 2016, from <https://iot-analytics.com/industrial-internet-disrupt-smart-factory/>
- Mike. (2009). Operating Systems Development - Kernel: Basic Concepts. Retrieved October 17, 2016, from <http://www.brokenhorn.com/Resources/OSDev12.html>
- NGSI10. (2014). NGSII0. Retrieved May 5, 2017, from https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE_NGSI-10_Open_RESTful_API_Specification
- NGSI9. (2014). NGSII9. Retrieved May 5, 2017, from https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE_NGSI-9_Open_RESTful_API_Specification
- OECD. (2011). Future Factory. *OECD Economic Surveys: Ireland 2011*, 8–9. <https://doi.org/10.1002/yd.20002>
- On, P. L., Ystems, S. M. S., Ep, I. N., & Lange, S. (2008). *Internet of Things in.*
- onem2m.org. (n.d.). One M2M. Retrieved January 17, 2017, from <http://www.onem2m.org/>
- OS Structure. (2008). OS Structure. Retrieved from <https://commons.wikimedia.org/wiki/File:OS-structure2.svg?uselang=pt>
- Puttonen, J., Lobov, A., Soto, M. A. C., & Lastra, J. L. M. (2016). Cloud computing as a facilitator for web service composition in factory automation. *Journal of Intelligent Manufacturing*, 1–14. article. <https://doi.org/10.1007/s10845-016-1277-z>
- Qian, H., Baokang, Z., Yunjian, L., Jinshu, S., & You, I. (2014). A Structure P2P Based Web Services Registry with Access and Control. In S. Teufel, T. A. Min, I. You, & E. Weippl (Eds.), *Availability, Reliability, and Security in*

- Information Systems: IFIP WG 8.4, 8.9, TC 5 International Cross-Domain Conference, CD-ARES 2014 and 4th International Workshop on Security and Cognitive Informatics for Homeland Defense, SeCIHD 2014, Fribourg, Switzerland, September 8-12, 2014. Proceedings* (pp. 286–297). inbook, Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-10975-6_23
- The Linux Information Project. (2005). Kernel Definition. Retrieved October 17, 2016, from <http://www.linfo.org/kernel.html>
- Wang, S., Wan, J., Li, D., & Zhang, C. (2016). Implementing Smart Factory of Industrie 4 . 0 : An Outlook, 2016. <https://doi.org/10.1155/2016/3159805>
- Whitmore, A., Agarwal, A., & Xu, L. Da. (2016). The Internet of Things — A survey of topics and trends The Internet of Things — A survey of topics and trends, (April 2014). <https://doi.org/10.1007/s10796-014-9489-2>
- Zebra Technologies. (n.d.). The Factory of the Future new value in manufacturing . HOW MANUFACTURING SOURCING , SUPPLY CHAIN MANAGEMENT AND PRODUCTION.



A. Entities Tables

In the following tables are presented the different entities created along the development of the applications, which served as basis to represent the necessary information to ensure the optimal work of the applications. The fields filled with < text >, are assumed to be filled with a specific value during the normal functioning of the applications.

Table A.1 - Fleet Entity.

Field		Content
Type		Fleet
ID		Fleet_<Farmer>
Attribute	Name	<TruckID>
	Type	
	Value	Stop / Travel

In Table A.1 is presented the Fleet Entity. This entity contains the information about the fleet of each farmer, farmer whose name will appear in the Entity ID field. Each of this Entity's attributes will represent a Truck added to that farmer's fleet and the value will change according to whether the truck is currently traveling or stopped.

Appendix A. Entities Tables

Table A.2 - Truck Entity.

Field		Content
Type		Truck
ID		<TruckID> (<FarmerID>_<TruckID>)
Attribute	Name	<SensorID>
	Type	<SensorType>
	Value	<Value>

In Table A.2 is presented the Truck Entity. This entity represents every truck present in the applications. Each truck has and ID, that mandatorily contains his owner's ID (the farmer ID in this scenario), as well as a list of its sensors (present in each attribute). These sensors will be represented by its ID (unique for each sensor in the applications as explained below), his type (whether they are a temperature sensor, or a pressure sensor, etc.) and its last read value. Each of the sensors added to a truck represent a real IoT sensor providing values read on a daily basis. The development of these sensors and the sending of their read values to the Orion GE are part of the vfHarvest Application. The arrow connecting both the vfHarvest and the vOrder applications depicted in Figure 4.1 represents this sharing of information, i.e. the storage of the values read by the IoT sensors in the Orion entities used in the vOrder application.

Table A.3 - Sensor Entity.

Field		Content
Type		SensorID
ID		<SensorID>
Attribute	Name	<TruckID>
	Type	
	Value	

In Table A.3 is presented the Sensor Entity. This entity represents every sensor present in the applications. This Entity is used to facilitate IoT discovery as each sensor is here represented by his ID (ID that will be unique for each sensor bought and used in a real use scenario, for testing purposes during the development of the applications, the sensor ID is here represented with a suggestive name - for example "te31": temperature, farmer3, truck1). Within each Sensor entity the ID of which this sensor is associated to is present in the Attribute Name field.

This entity was created, as previously stated, for IoT discovery purposes. Since the regular sensors available in the market provide only its ID alongside the value they are currently reading, this entity can be used to, making use of that information provided by the sensor, discover to which truck that sensor is assigned to, and therefore, to which Truck Entity the value read by the sensor should be sent to (the "Attribute Value" field presented in Table A.2).

In order to keep track of some changes that happen with the Entity's attributes value fields, and once again as stated in Subchapter 3.5.2 of Chapter 3, the FIWARE Orion Context Broker provides a functionality called Subscriptions. This functionality allows certain enablers, like the FIWARE STH - Comet, presented in Subchapter 3.5.3 of Chapter 3, to be notified every time a certain attribute value suffers changes. These subscriptions are, during the normal functioning of the applications, created with a big life time (i.e. they are made to stay active for a very long time), so whenever an Entity which contains a subscription needs to be deleted, that subscription also needs to disappear regardless of how much time it still has left. Therefore, two entities were created in order to keep track of the different Subscriptions IDs that are generated during the functioning of the applications, so that they can be Unsubscribed when the corresponding Entity is deleted.

The two types of Subscriptions used in these applications are the "Sensors Value" subscription (a warning every time a sensor reads a new value) used to analyse the transport conditions (as showed further is this document) and the "Truck State" subscription, which notifies every time a truck state changes from "Stop" to "Travel" and vice-versa (used to control how much a time a certain truck voyage took).

Appendix A. Entities Tables

Table A.4 - Subscription (Truck State) Entity.

Field		Content
Type		Subscription
ID		Fleet_<FarmerID>
Attribute	Name	<Truck>
	Type	
	Value	<SubscriptionID>

In Table A.4 is presented the Subscription entity representing all the “Truck State Subscriptions” created during the applications run. This first type of subscription holds all the subscriptions IDs saved during the addition of a certain truck a farmer’s fleet (the subscription to notify every time a truck state changes from “Stop” to “Travel” or vice-versa). These subscriptions are represented by its ID (composed by the Farmer’s fleet ID, the monitored Truck’s ID and the respective subscription ID), so that it can be unsubscribed at any time using the subscription ID stored in this Entity.

Table A.5 - Subscription (Sensor Values) Entity.

Field		Content
Type		Subscription
ID		<FarmerID>_<TruckID>
Attribute	Name	<SensorID>
	Type	
	Value	<SubscriptionID>

In Table A.5 is presented the Subscription entity representing all the “Sensors Values Subscriptions” created during the applications run. This second type of subscription holds all the subscriptions IDs saved during the creation of the sensor’s subscriptions (the subscription to notify every time a sensor value changes). These subscriptions are represented by its ID (composed by the Farmer’s and the Truck’s ID, the Sensor’s ID and the respective subscription ID), so that it can be unsubscribed at any time using the subscription ID stored here.

All the entities presented so far operate mainly during the vOrder application run. All are used to store and retrieve information about the IoT sensors and others, used to make the vOrder functionalities accessible to the user. Between the vfNegotiation App and the vOrder App a new entity is used which represents the Order processed between the consumer (representing the buyer, in this scenario) and the producer (representing the farmer, in this scenario).

Table A.6 - Order Entity.

Field		Content	
Type		Order	
ID		<OrderID> (<FarmerID>_<OrderNumber>)	
Attribute	Name		<FarmerID>
	Type		
	Value		
	Metadata[0]	Name	OrderValue
		Type	
		Value	<Value>
	Metadata[1]	Name	Truck
		Type	
		Value	<TruckID>

Appendix A. Entities Tables

	Metadata[2]	Name	<FruitName>_<FruitBreed>
		Type	<FruitSize>
		Value	<FruitAmount>

In Table A.6 is presented the Order Entity. This entity represents the different orders that will take place between a buyer and a farmer. The Order Entity contains the OrderID (which for development and test purposes possesses the FarmerID to whom the order was emitted to, as well as an Order Number to keep a record of the orders previously emitted to that farmer - to keep two different orders to have the same identification number, as will be presented further ahead in this document). However, during the real usage of the applications every order can have a specific ID that doesn't need to be equal the one presented here. Furthermore, the Order Entity will also have in the Attribute Name field, the Farmer answering to that order, and a metadata split in three different parts. The first part saves the total monetary value involved in the order, i.e. the payment that will occur from the buyer to the farmer. The second part shows the Truck (chosen by the farmer) that is transporting the goods related to that order. Finally, the third part holds the transported goods information. In the Name field will be present the fruit name and breed, the type field will hold the ordered fruit size, and in the Value field the amount of fruit requested by the buyer.

Table A.7 - Farmer Entity.

Field		Content
Type		Farmer
ID		<FarmerID>
Attribute	Name	<FruitName>_<FruitBreed>
	Type	
	Value	<Amount>

	Metadata[0]	Name	ID
		Type	String
		Value	<Size>
	Metadata[1]	Name	Price
		Type	Float
		Value	<Price>

In Table A.7 is presented the Farmer Entity. This Entity represents each farmer using the applications and holds all the information about the goods they possess. The Farmer Entity is constituted by the Farmer's ID, and the attributes will represent each of the fruits that farmer produces. Every attribute (representing a different type of fruit) holds the fruit name and breed, as well as the amount of that fruit the farmer currently owns. In the metadata will reside the information about the fruit size (makes use of the previously explained "ID Name field" which allows the same entity to have different attributes with the same Name, to allow each farmer to have many different sized of a specific fruit breed). Besides all the fruit information reported the entity will also have another metadata field where the farmer can specify that fruit price (allowing the farmer to stipulate a different price for every combination of fruit name, fruit breed and fruit size).

Table A.8 - Fruit Production Entity.

Field		Content
Type		FruitInfo
ID		<FarmerID>
Attribute	Name	<FruitName>_<FruitBreed>
	Type	<Size>
	Value	<Amount>

Appendix A. Entities Tables

		Name	ID
	Metadata[0]	Type	String
		Value	<Size>

In Table A.8 is presented the Fruit Production Entity. This entity is shared by both the vfNegotiation Application developed by me and described in this thesis and the vProductMon Application developed by another member of the workgroup doing the master’s thesis under the vf-OS project. This Entity serves as the link between both these applications as depicted in Figure 4.1.

This entity is created whenever a new farmer is registered into any of the applications (vOrder or vfNegotiation) and is populated by the vProductMon application. Using IoT Sensors and created rules, the vProductMon App will evaluate how many of a farmer’s produced fruits fall within each fruit Size, i.e. every time a farmer produces a fruit (fruit being a set of fruit name and fruit breed), through measuring sensors and rules the vProductMon will evaluate within each size the produced fruit fits into, and will save the amount of fruit that falls within each size in this entity. Thus, this entity will represent the amount of fruit produced for each fruit type (fruit name, breed and size).

The fields that constitute this entity are the Farmer’s ID, which is the farmer producing the fruit, the fruit name and breed in the attribute Name field, and the fruit size and amount in the Type and Value fields respectively. In order for the entity to have more than one attribute (representing a fruit) with the same name (since each farmer can breed more than one size of the same fruit), the metadata field was introduced with the Name field being ID (in order for the Orion to allow the entity to have more than one attribute with the same name, as explained in Subchapter 3.5.2 of Chapter 3). With this being said, the last fields, the metadata fields, of this entity are the Metadata Name, Type and the most important one Value, which is the characteristic that distinguishes different attributes (fruits) with the same name.

After this entity is fully populated by the vProductMon each time a farmer has a new harvest, that farmer will be able to check his last production values in the vfNegotiation App, and choose which and how many of the produced goods he wants to put for sale.