

# POLITECNICO DI TORINO

---

## MASTER DEGREE IN MECHATRONIC ENGINEERING



## MASTER THESIS

### HiL for Power Systems and Smart Grids

A Resilient Information Architecture Platform for Smart  
Grids based application to control a SMA solar inverter

#### SUPERVISORS

Prof. Massimo Violante

Dr. Srdjan Lukic

#### CANDIDATE

Giangabriele Castorina

ID 254213

#### INTERNSHIP TUTOR AT FREEDM SYSTEM CENTER

Dr. Srdjan Lukic

ACADEMIC YEAR 2019/2020

# Abstract

Distributed energy sources are becoming a key component of many power systems. Therefore, it is necessary to analyze the best solutions to manage the energy with the dual intention of reducing waste and improving the quality of production. This study aims to demonstrate the integration of a smart energy management system into renewable energy resource systems in order to provide a list of benefits that is not normally obtained with standard platforms. These benefits include: resilience to faults, security access, plug and play features, a coordination system and distribute intelligence through the network. A resilient information architecture platform for smart grids, RIAPS, is implanted in order to control a solar inverter produced by SMA company, with the future intention being to manage data and send commands at different levels in a local sub-net.

To achieve this goal, a study is conducted which concludes the necessity of integrating the TCP/IP protocol into the platform to constitute the inverter data connection request.

The approach utilizes the V-shape model in order to accomplish tasks for implementation, test and validation of different steps related to the Model in the Loop, Software in the loop and Hardware in the loop simulations. Different tools are used during this process including hardware such as "OPAL-RT" and software simulators such as "Simulink", "QModMaster", concurrently with the "Sunny Explorer" platform, provided by SMA company, to validate the accuracy of the main test results related to Power limitation and Shut-down operations.

In conclusion, the results from the different tests prove the effective integration of the novel platform in commercial devices within a smart energy management environment. This is the first application in this field which the authors are aware of. Moreover, the project is designed to provide a solid foundation from which to develop large-scale applications, using multiple interconnected devices to cooperate on the same network and in general to provide benefits in the area of Smart Grids.

# Declaration

This work was funded in part by the Advanced Research Projects Agency-Energy (ARPA-E), U.S. Department of Energy, under Award Number DE-AR0000666. The views and opinions of authors expressed herein do not necessarily state or reflect those of the US Government or any agency thereof.

# Acknowledgements

I wish to express my sincere appreciation to my supervisor, Dr. Srdjan Lukic, who guided and helped me professionally suggesting me always the right path to be taken. I wish to express my deepest gratitude to PhD. Hao Tu, who assisted me during the technical part in developing the earlier steps of the application with suggestions and support in the most critical parts of the project.

I would like to recognize the invaluable assistance from the software developer team of Vanderbilt University, Nashville, TN that guided me during the learning phase of RIAPS platform. Above all, a distinguished thank addressed to the software developer Mary Metelko, supporting me continuously with detailed explanation of some standard applications described in RIAPS basic applications.

Moreover i wish to express my sincere gratitude to the Erasmus+ program association, including Politecnico di Torino, IT and the hosting North Carolina State University, Raleigh, NC institutions, both responsible for the program agreement, supporting me either in bureaucracy and economic sides during this time-frame abroad study.

I would like to pay my special regards to Dr. Iqbal Husain to welcoming me in FREEDM System Center, department of Electrical Engineering, that has all the tools necessary for the development of highly innovative research activities.

A special regard goes also to Danny Crescimone, colleague and friend of mine, which suggested me this amazing experience. I wish to thank all the people whose assistance was a milestone in the completion of this project, in particular to PhD. students Markus Mühlbauer, Alireza Dayerizadeh, Thomas Dotson, giving me brilliant suggestions for the drafting procedure of this thesis document and to Post-Doc Menhaz Khan, PhD. students Rahul Chakraborty, Valliappan Muthukaruppan, Siddarth Rath, Likhita Ravuri and Oscar Montès for the help availability given during this project development.

It is whole-heartedly appreciated the great support from the FREEDM system center staff member's, in particular lab manager Hulgize Kassa, secretaries Karen Autry, Rebecca McLennan, Ken Dulaney and Terry Kallal, giving me professional support accomplishing any possible request and contributing to make my experience everyday more comfortable in this new environment.

A huge and special reward goes to my whole family members especially to my parents



Camillo Castorina, Giuseppa Farina and my sister Giada Castorina that remotely assisted me basically in everything giving me moral support to overcome from the simplest to the toughest obstacles encountered in the whole research period.

I would like to pay my special regards to all my Italian friends and also to my new international friends, known in this abroad experience, that in general helped me to feel at ease even though thousands kilometers away from home.

I would like to conclude by saying that without the persistent help from the people mentioned above, the goal of this project would not have been partially or totally achieved correctly.

# Contents

<b>List of Figures</b>	VIII
<b>List of Tables</b>	XI
<b>1 Introduction</b>	1
1.1 Motivation and goal . . . . .	2
1.2 Method of working . . . . .	2
1.3 State of the Art of the actual system . . . . .	3
1.4 Prototype solution analysis . . . . .	4
1.4.1 The need for a new energy management system . . . . .	4
1.4.2 Requirements for the new system . . . . .	5
1.5 Thesis structure . . . . .	6
<b>2 Theoretical basics</b>	7
2.1 Distributed energy resources . . . . .	7
2.1.1 PV solar panels . . . . .	7
2.1.2 Solar inverter . . . . .	15
<b>3 A Resilient Information Architecture Platform for Smart Grid</b>	20
3.1 Challenges and expectation . . . . .	23
3.2 What is RIAPS . . . . .	23
3.3 System architecture layout . . . . .	25
3.3.1 Architecture run-time system . . . . .	25
3.3.2 Application model . . . . .	28
3.4 Component framework . . . . .	29
3.5 Components interaction . . . . .	31
3.5.1 Component execution engine . . . . .	31
3.5.2 Device interface service . . . . .	32
3.5.3 Messaging framework . . . . .	35
3.5.4 Resource and fault tolerance framework . . . . .	37
3.5.5 Security framework . . . . .	41
3.6 Run-time services . . . . .	42

3.6.1	Discovery service . . . . .	42
3.6.2	Deployment service . . . . .	45
3.6.3	Time synchronization service . . . . .	47
3.6.4	Distributed coordination service . . . . .	47
3.7	Design-time tools . . . . .	48
3.7.1	Modelling language for the architecture . . . . .	49
3.7.2	Software generators . . . . .	49
3.7.3	Debug tool support . . . . .	50
3.8	Application deployment and control tool . . . . .	51
3.9	Implementation . . . . .	53
3.10	Conclusion of the description of the platform . . . . .	55
<b>4</b>	<b>Background implementations of RIAPS</b>	<b>57</b>
4.1	GPIO Device Toggle . . . . .	58
4.1.1	Hardware configuration . . . . .	58
4.1.2	Software Configuration . . . . .	59
4.1.3	Application architecture . . . . .	59
4.1.4	Simulation and outcomes . . . . .	61
4.2	Distributed Estimator . . . . .	62
4.2.1	Hardware configuration . . . . .	62
4.2.2	Software configuration . . . . .	62
4.2.3	Application architecture . . . . .	63
4.2.4	Simulation and outcomes . . . . .	65
4.3	Distributed Estimator using GPIO . . . . .	66
4.3.1	Hardware configuration . . . . .	66
4.3.2	Software configuration . . . . .	67
4.3.3	Application architecture . . . . .	67
4.3.4	Simulation and outcomes . . . . .	70
4.4	UART A to UART B communication . . . . .	71
4.4.1	Hardware configuration . . . . .	71
4.4.2	Software configuration . . . . .	72
4.4.3	Application architecture . . . . .	73
4.4.4	Simulation and outcomes . . . . .	75
<b>5</b>	<b>Methods and Tools</b>	<b>78</b>
5.1	Algorithm requirements . . . . .	79
5.2	Protocols and hardware tools . . . . .	79
5.2.1	TCP/IP communication protocol . . . . .	80
5.2.2	Sunspec protocol . . . . .	82
5.2.3	BeagleBone Black . . . . .	84
5.2.4	Opal-RT 5600 simulator . . . . .	84
5.2.5	Inverter SMA STP 20000TL-US-10 . . . . .	85

5.3	Model in the loop simulation . . . . .	86
5.4	Software in the loop simulation . . . . .	90
5.5	Hardware in the loop testing and validation . . . . .	93
5.6	Code examination and software tool support . . . . .	94
5.6.1	Hardware configuration . . . . .	95
5.6.2	Software configuration . . . . .	96
5.6.3	Application architecture . . . . .	96
<b>6</b>	<b>Test-bench results</b>	<b>104</b>
6.1	Earlier steps . . . . .	107
6.2	Power limit simulation test . . . . .	110
6.3	Shut-down simulation test . . . . .	113
6.4	Critical review . . . . .	115
<b>7</b>	<b>Conclusion and Future work</b>	<b>118</b>
7.1	Conclusion . . . . .	118
7.2	Future work . . . . .	120
<b>A</b>	<b>Code for Software in the loop simulation</b>	<b>122</b>
<b>B</b>	<b>Code for Hardware in the loop simulation</b>	<b>128</b>
	<b>Bibliography</b>	<b>138</b>

# List of Figures

1.1	Original configuration of the system . . . . .	3
1.2	Prototype of the system as a possible solution . . . . .	4
2.1	Sunlight effect on a photosensitive material . . . . .	8
2.2	P-N junction[43] . . . . .	9
2.3	Solar cells arrangement (a)-(b) [8]-[9] . . . . .	10
2.4	Characteristic I-V of solar panel [30] . . . . .	11
2.5	Family of PV Power-Voltage characteristic[30] . . . . .	13
2.6	Flow chart constant voltage algorithm [30] . . . . .	13
2.7	Flow chart incremental conductance algorithm [30] . . . . .	14
2.8	Simple inverter circuits [11] . . . . .	16
2.9	Electrical circuit of a three phase inverter [10] . . . . .	18
2.10	Fundamental output six-step inverter 120° conduction mode [17] . . . .	18
3.1	Centralized structure [40] . . . . .	21
3.2	Distributed structure [40] . . . . .	22
3.3	Riaps as a distributed computing platform [24] . . . . .	24
3.4	Software architecture [16] . . . . .	26
3.5	Riaps nodes [33] . . . . .	30
3.6	Component Framework [33] . . . . .	30
3.7	Component execution [24] . . . . .	32
3.8	Device interface Service [33] . . . . .	33
3.9	External device component execution [24] . . . . .	35
3.10	Interactions between components [24] . . . . .	36
3.11	CPU utilization example [26] . . . . .	37
3.12	Memory utilization example [26] . . . . .	38
3.13	Space Utilization example [26] . . . . .	38
3.14	Network Utilization example [26] . . . . .	38
3.15	Resource Management application [26] . . . . .	39
3.16	Fault management [23] . . . . .	40
3.17	Security Network [23] . . . . .	41
3.18	Discovery Service [33] . . . . .	43

3.19	Deployment service [33]	45
3.20	File example ".depl" (a) [26]	46
3.21	File example ".depl" (b) [26]	47
3.22	Time synchronization service [33]	48
3.23	Riaps services [24]	49
3.24	Model-driven development [33]	50
3.25	Executable block scheme [22]	53
3.26	WLAN implementation [22]	54
3.27	LAN implementation [22]	55
4.1	Beaglebone Black pin-out data-sheet [2]	58
4.2	File gpioExample.riaps [20]	60
4.3	File ".dot" gpioExample	60
4.4	File gpioExample.depl [20]	61
4.5	GPIO example during execution	62
4.6	File "sample.riaps" [18]	63
4.7	Distributed estimator ".dot"	64
4.8	Distributed Estimator ".depl" file [18]	65
4.9	Distributed Estimator Simulation	66
4.10	File Distributed Estimator using GPIO ".riaps" [19]	68
4.11	File ".dot" Distributed Estimator using Gpio	69
4.12	File Distributed Estimator using GPIO ".depl" [19]	70
4.13	Distributed Estimator Using GPIO simulation	70
4.14	UART to UART connection	71
4.15	File ".riaps" UART Device Testing [27]	73
4.16	File ".dot" UART Device Testing"	74
4.17	File ".depl" UART Device Testing [27]	75
4.18	UART A to UART B communication simulation	76
5.1	V-shape model entire system development	79
5.2	OSI VS TCP [42]	80
5.3	Sunspec standard representation [38]	83
5.4	Connection model during MIL simulation	86
5.5	Opal-RT Simulator	87
5.6	Complete Simulink model	87
5.7	SC_Console model	88
5.8	OPAL RT model	89
5.9	Connection model during SIL simulation	91
5.10	Writing function in " <i>ComputationalComponent.py</i> "	91
5.11	Reading values from RIAPS	92
5.12	Reading values from Simulink	92
5.13	Connection model during HIL simulation	93

5.14	Solar panels (a), Inverter SMA STP-20000TL-US-10 (b), Router and BBBs connections (c) . . . . .	94
5.15	File "modbus_tcp_core.dot" . . . . .	97
6.1	Inconsistency on DC values from "QmodMaster" and "RIAPS" in different instances . . . . .	106
6.2	Step by step configuration . . . . .	107
6.3	Double-check operation of register values . . . . .	108
6.4	Debug operation during the saving operation phase . . . . .	108
6.5	Final implementation on RIAPS platform . . . . .	109
6.6	Sunny Day from "SunnyExplorer" platform . . . . .	109
6.7	Active power limit response from "Riaps" . . . . .	110
6.8	Active power limit response from "Sunny Explorer" . . . . .	111
6.9	Active power limit from "SunnyExplorer" events . . . . .	112
6.10	Graphical view from "SunnyExplorer" Limit 300[W]" . . . . .	112
6.11	Graphical view from "SunnyExplorer" limit "2000 [W]" . . . . .	113
6.12	Shut-down from Riaps platform . . . . .	114
6.13	Shut-down from "SunnyExplorer" events . . . . .	115
6.14	Graphical view from "SunnyExplorer" Shut-down . . . . .	115
A.1	File "modbus_tcp_core.riaps" . . . . .	122
A.2	ComputationalComponent.py . . . . .	123
A.3	ModbusTcpReqRepDevice.py . . . . .	124
A.4	ModbusTcpReqRepDevice.py . . . . .	125
A.5	tcpModbusComm.py . . . . .	126
A.6	tcpModbusComm.py . . . . .	127
A.7	File "modbus_tcp_core.depl" . . . . .	127
B.1	File "modbus_tcp_core.riaps" . . . . .	128
B.2	ComputationalComponent.py . . . . .	129
B.3	ComputationalComponent.py . . . . .	130
B.4	ComputationalComponent.py . . . . .	131
B.5	ComputationalComponent.py . . . . .	132
B.6	ComputationalComponent.py . . . . .	133
B.7	ModbusTcpReqRepDevice.py . . . . .	134
B.8	ModbusTcpReqRepDevice.py . . . . .	135
B.9	ModbusTCPLLogger.py . . . . .	135
B.10	tcpModbusComm.py . . . . .	136
B.11	tcpModbusComm.py . . . . .	137
B.12	File "modbus_tcp_core.riaps" . . . . .	137

# List of Tables

2.1	Comparison among different MPPT algorithms [30]	15
5.1	Functional values for Modbus TCP/IP block	90
5.2	Selected Modbus registers for SMA STP 20000TL-US-10	99





# Chapter 1

## Introduction

Nowadays real world applications related to the energy production and management are becoming more popular even in relatively small system.

Regardless, technology progress, is pushing towards progressively powerful systems in order to supply the increasing worldwide request of energy caused by multiple factors. In fact with the technology diffusion, many devices are supplied by energy and due to the raising of the population, this topic is becoming increasingly more relevant. Limited amount of non-renewable energy coming from fossil fuels and high harmful emissions of carbon dioxide, mostly related to the process of this type of resource, generate the necessity to recover energy differently.

Every year new strict normative regulating the amount of  $CO_2$  emissions are issued in order to reduce the pollution because it is the main reason of global warming that slowly affect our planet with catastrophic consequences for the entire ecosystem.

Renewable energy generation coming from high power wind generator and photovoltaic power systems are the most used in the every day common scenario [1], in order to supply to this problem. Renewable energy could also be generated by hydro-plants systems or by geothermal, or by bio-mass systems.

Nevertheless, most of renewable energy resource are affected by non-predictable factors because of the fluctuation of the weather condition they strictly depend on and for this purpose, many researches are faced in this context [4]. Renewable energy are also classified in programmable or non-programmable energy resource depending if the energy production could be provided right after a programmed request or not. The problem of spreading the use of renewable energy resource, is as important as the energy management, which is intended to reduce the excess of energy produced, and this will provide new solutions constituting a topic of continuously updated research.

## 1.1 Motivation and goal

The problem of emissions and all the issues related to it, give the necessity to think about a new energy management concept able to control and monitor the variables related to it in order to reduce the production of wasted energy. The increasingly demand of energy is indirectly related to the use of new management concept able to satisfy the standards and accommodate from the simpler to the complex requests. Moreover the concept of a smart energy management system is not only intended to be used as something able to push the actual system inside the generalized standards adopted by the different nations, but it is above all, a valid reason to create a connection between devices able to interact each others, exchanging information and data necessary for the self-management system.

In order to establish this connection, a sort platform solution, able to operate in this optimistic context and to generate some improvement on the basic production plants, is needed. The final goal is to successfully demonstrate the integration between a new energy management platform and the actual system, a simple solar inverter, able to produce green energy.

## 1.2 Method of working

In order to reach the final destination goal, several steps have to be encountered which are also important to subdivide the main big system into multiple smaller ones. For this purpose, an accurate study of the theoretical part, concerning the investigation of the system basics and limits it presents, including some insights about the working principle of the inverter and control strategy adopted, is necessary.

Afterwards, the comprehension of the theory of "Riaps", the revolutionary software platform used, is fundamental. This is followed by a training for the correct usage of this platform in different scenarios, through several basic applications directly provided by the developers.

Here it comes the point, where all the previous nodes are converging, this, require the necessity of the implementation of a communication protocol "TCP/IP" never implemented previously complementary with this revolutionary platform. A theory basic is analyzed to understand how the communication layers are organized. The utilization of new software able to interfere with the simulator and different simulation tools is necessary to detect errors during the debugging steps, but especially to generate the simulations performed accomplishing multiple requirements, mostly related to the V-shape model. A moderate knowledge of python coding and basics of C++ are necessary to face the lower level steps of the V-model and to be consistent with the constrained rules of the "Riaps" platform.

At the end, simulations on the real hardware, studying the behaviour during some

specific tests, are performed. Simulation results are pointed out leading to a conclusion that a new higher controller, able to operate above the existent one, with multiple benefits, inherited from the revolutionary software platform, is established.

### 1.3 State of the Art of the actual system

The state of the art of the system in the original configuration is simple and it is represented in Fig. 1.1. The inverter is able to exchange data through the router to the "Sunny Explorer" standard platform by means of internet. The purpose of "Sunny explorer" platform is to monitor and collect instantaneous and averaging values and it is provided by the same manufacturing company of the inverter. From this platform it is also possible to send commands on the basis of the user request, but these has to be sent manually and singularly for each connected device.

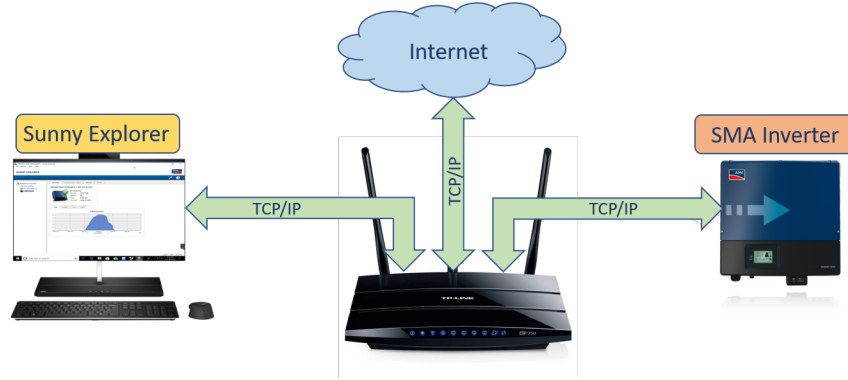


Figure 1.1. Original configuration of the system

#### Limits of the actual system

- Sunny explorer is a internet based system, so data have to circulate through the cloud before to reach the user. This could affect time constraints due to slow and long round trip.
- Sunny explorer platform is able to provide a max of 50 device connection within one master on the network.
- There is no possibility to manage data autonomously according to the interaction among devices.
- The computing platform is centralized and each device refer to a single point similarly to a central station of control.

## 1.4 Prototype solution analysis

The new model propose the use of a new software platform able to create a common environment in which all the nodes are interconnected each other. This provides a significant improvement on the management of the "DERs", distributed energy resources, because the new platform in general is able to constantly adapt to new scenarios due to its software based flexibility. This new concept includes several improvements over standard systems which lead it to become a "smart system". A graphical prototype solution representing the main idea of this new energy system is presented in Fig. 1.2.

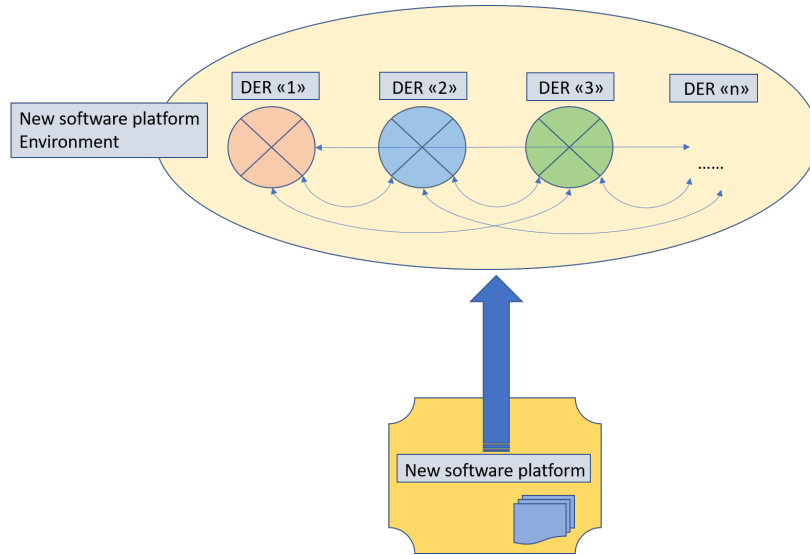


Figure 1.2. Prototype of the system as a possible solution

### 1.4.1 The need for a new energy management system

The main reason that lead to think about a new energy management system is to try to improve the actual system in better managing data and commands and even the faults and to fill gaps of an standard one.

For this reason it is important to remark that the electric energy, is a pillar of today's society, where electric and electronic devices constantly depends on. Nevertheless, nowadays the worldwide economy turns around the production, dispatching and selling of the electric energy, so that it is currently impossible to think about a world without it.

Once realized the importance of electricity today, the best ambition concern the following question.

What would happen if all the energy in the world will be managed in the best way avoiding harmful waste for our eco-system? In order to answer to this question it is necessary to improve the current systems by providing an upgrade in terms of management of important quantities.

### 1.4.2 Requirements for the new system

The requirements for the realization of the final goal are listed below as follows:

- The new system interaction should be based on a local network and not based on internet anymore, for data privacy and security reasons.
- The communication should be performed using TCP/IP protocol and not on Modbus RTU (remote terminal unit), because it is physically easier to install and to configure.
- The computing power of the data/command management, should be distributed among all the interconnected nodes inside the network.
- The platform used should be able to host multi-connection with plug and play accessibility.
- The system should be resilient in general to faults, to avoid loss of important data.
- The algorithm should own the predisposition to self manage devices on the basis of the information exchanged.
- Finally the system must support the use of a new software platform to realize the previous requirements.

## 1.5 Thesis structure

A summary of thesis presenting a short introduction for each chapter and its main contribution is presented as follows.

Chapter1: *Introduction* is providing an introduction to the part of the world studied, followed by the motivation and the final goal. Then, a state of the art including some limits of the actual system and the purpose of research are provided, concluding with a list of requirements for the new system.

Chapter 2: *Theoretical basics* is intended to introduce to the theory of the basic working principle of the systems considered related to the PV plant comprising PV panels and inverter explanation.

Chapter 3: *Resilient Information Architecture Platform for Smart Grid* is the theoretical focus of this work, intended to provide all the information that the platform is able to provide in a smart grid environment and how single components works inside.

Chapter 4: *Background implementation of RIAPS* is intended to give an idea of the working principle of the platform in different scenario, with basic applications useful to learn how does it work in a real context.

Chapter 5: *Methods and Tools* is providing all the information related to the methods used to reach the final goal. A detailed documentation of the steps during the simulations followed is given relating to the V-model, with the inclusion of some theory aspects considered. It include also the description of the file developed to run inside the platform.

Chapter 6: *Test-bench results* Simulations and outcomes are provided starting from step by step implementation and for the two major important tests related to "Active power limitation" and "Shut-down simulation" to let space for a final critical review where some aspects not covered or doubt solutions are pointed out.

Chapter 7: *Conclusion and Future work* is the last step providing a conclusion based on the certainties acquired from the study of the problem. This is followed by a section exploiting an introduction of possible topics to be investigated in order to further improve the already implemented system.

# Chapter 2

## Theoretical basics

This chapter is intended to provide basic explanation of the theory related to the topics covered in the next chapters. Starting with a description of Distributed energy resources regarding insights about PV plant to and some particular control strategy adopted, continuing with a basic explanation of the structure and the working principle of solar inverters and concluding with the differentiation of different implementation in the from the smaller to the bigger application.

### 2.1 Distributed energy resources

Distributed energy resources (DERs), are resource able to produce electricity that due to the technology development and environment protection are merged within a local distribution system. These DERs, are contributing significantly in the world change of electricity systems. Examples representing this category of DERs are: PV power plants, internal combustion (IC) engines, gas turbines, micro-turbines, fuel-cells and wind-power [31]. It is difficult to highlight which is the most powerful one in a global way, because this strictly depends on the major benefits the geographical area can exploit, where these DERs are installed.

A photovoltaic (PV) power plant is the composition of solar panels for the acquisition of the sunlight energy and the conversion to the electrical energy. A PV power plant is also including the conversion stage, performed by the inverter able to convert the electric energy on the basis of the end user for the requested application. Although the conversion is performed following the basics of electrochemistry, the content hereby is summarized in order to get an overview of the concept.

#### 2.1.1 PV solar panels

A PV panel is composed by a single or multiple solar cells, providing different characteristics depending on the configuration they are mounted. Different configuration



are present, these includes series or parallel connections that provide different voltages or powers. Since every standard single cells is able to produce less than 1V it is necessary to be combined together with the others to provide the required output.

### Working principle basics

The working principle is based on the electrochemistry concept. A beam of light coming from the sun, hit a photosensitive material which release electrons through the absorption of photons and ionization of crystal atoms as shown in Fig. 2.1.

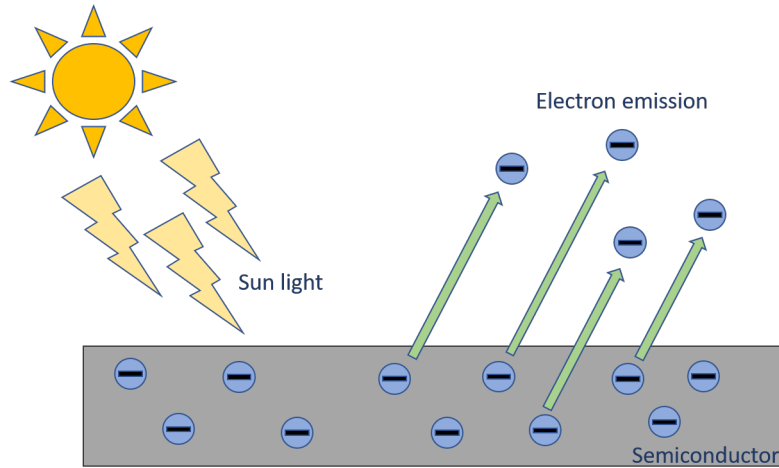


Figure 2.1. Sunlight effect on a photosensitive material

This create negatively charged electrons and positively charged ions, exclusively generated by the basic crystal atoms. Ions transfer its positive status to their neighbor atom, so that this generate "holes" that have the same property of an "electron" but with opposite charge. Every time a hole is generated, an electron is present in order to balance the total charge. In order to establish electric potential, a P-N junction is necessary using the P and the N doped material, presenting respectively an excess of electrons and an excess of holes. It should be emphasized that the material result to be globally neutral, because the doping is carried out with neutral atoms (not ions), on the one hand, and the holes of the same on the other and the only thing that changes is the excess of electrons in a covalent bonds. So, when a material of P type and a material of N type are joined together to create a P-N junction, a balance of charge is established by migration of electrons from N zone to P zone and a "built-in" electric field is created in the zone in the middle of the P-N junction called "depletion region" as highlighted in Fig. 2.2.

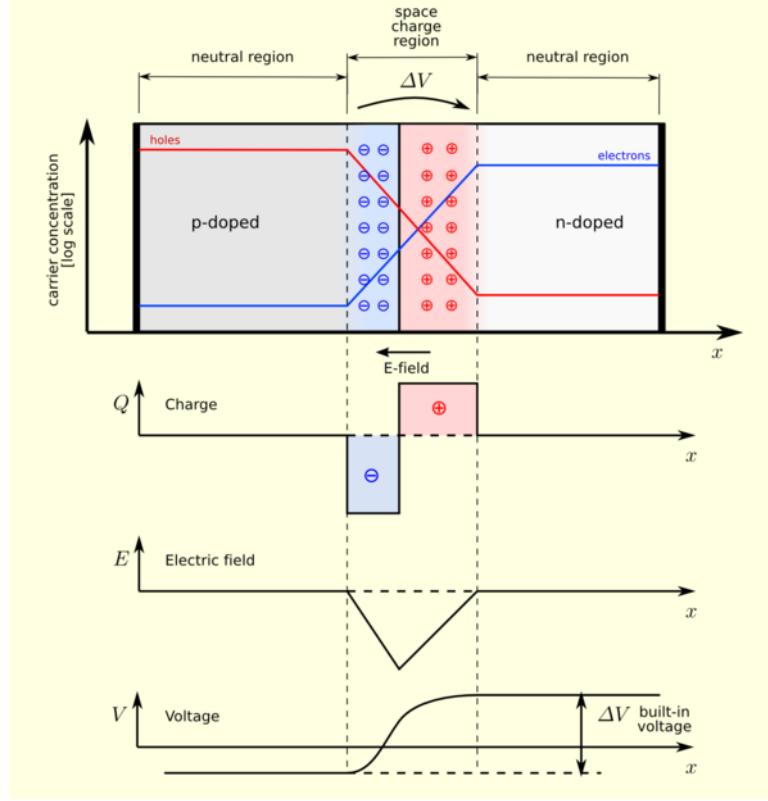


Figure 2.2. P-N junction[43]

Afterwards, once the beam of sunlight hit the photosensitive material, as a consequence of the electrons release, a current flow is established, through the action of the electrical field that pushes the just created electrons in excess in one side and the holes in the other side. The electric field is also able to avoid the charges going back to the previous configuration, and a voltage is generated from N to P material. These flow of electrons is generating a current that is captured by a wiring system inside the cell and connected to the other cells in order to create a panel. The entire process is repeated until the energy of the photon hitting the solar cell is at least equivalent to the energy of the band transition of the photosensitive material [7]. Different type of photosensitive material could be used to accomplish this task such as silicon, gallium arsenide, indium phosphide, cadmium telluride, copper indium diselenide, and cuprous sulfide [6], but the mostly used is the silicon present in: amorphous, multi-crystalline and crystalline stage.

The efficiency is depending on the purity of the material and for the previously mentioned stage and for a constant irradiance provided approximately it range from 10% to 25%. Nevertheless the efficiency of a solar cell is dictated also by the angles the sun incise on the panel. These angles are referring to the zenith, to the azimuth

and to the horizon. Many studies are carried out trying to find out the value of these angles that maximize the efficiency.

Moreover solar cells are arranged together to create a module configuration and this last one in turn is associated with other modules to create a panel in Fig. 2.3 (a). As consequence, multiple panels together form an array. Finally different arrays can be arranged together in order to generate a so called field as shown in Fig. 2.3 (b).



Figure 2.3. Solar cells arrangement (a)-(b) [8]-[9]

## Maximum power point of tracking

The "MPPT", maximum power point tracking, is one specific device running a particular power control algorithms followed by PV plants to maximize the power

extractable from the solar cells. It is responsible for providing also minimization of the overall system costs and maximize the array efficiency. The solar cells have a theory apart that describe the conversion of energy depending on multiple factors such as: solar irradiation, temperature and total resistance producing a non-linear I-V characteristic as shown in Fig. 2.4. The location of the MPPT is highly dependent on the previously mentioned factors and it has to be tracked continuously during the working operation. In fact, due to the mismatch between the load line and the operating characteristic of the solar cell, the power available from the solar cells is not always fully extracted [39].

The purpose of the MPPT control technique is to adjust the the terminal voltage of PV panels so that maximum possible power can be extracted. The MPPT control

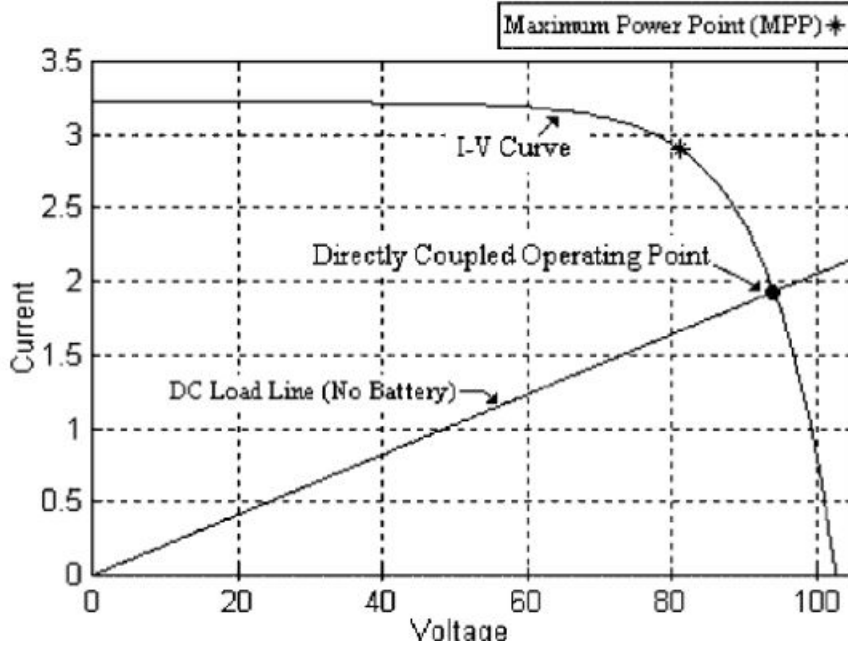


Figure 2.4. Characteristic I-V of solar panel [30]

strategy is responsible for determine the value of resistance identified as a load to their output in order to reach the maximum amount of power for any environment conditions. The concept of the MPPT is related to the fill factor (FF), which is a parameter determining the maximum power from a solar cell.

It is defined by 2.1:

$$FF = \frac{P_{max}}{(V_{oc}I_{sc})} \quad (2.1)$$

where the " $V_{oc}$ " and the " $I_{sc}$ " are respectively the open circuit voltage and the short circuit current values. Different kind of MPPT algorithms exists they are:

- *Perturb and observe "P&O"* that is the most widely used and easiest to implement control algorithm even if it is defined in literature the less efficient one.
- *Constant voltage* that is also easy to implement, but have also drawbacks in the evaluation of parameters.
- *Incremental conductance* which is one of best algorithm for evaluating the MPPT.

### Perturb and observe technique

In the "perturb and observe technique" [30] the voltage at the terminals of the solar cells is continuously perturbed, and it is calculated the correspondent output power difference. In Fig.2.5 is represented a family of PV array power curves as a function of voltage (P-V curves), at different irradiance "G" levels, for uniform irradiance and constant temperature. Considering the power array operating at point A, in order to reach the final MPPT point in the curve, the voltage is perturbed to change in a small amount, and after that the power difference is calculated. If the power difference is positive, in this case, the perturbation of the operating point of the PV array is moved towards the MPPT. Instead, if the difference of power measured is negative, in this case, the operating point is moved away from the MPPT[30]. The drawback on this algorithm is that, whenever the sunlight decrease, the P-V curve flatten, and the MPPT is difficult to be identified. Since this algorithm is not so accurate because it perturb the voltage every single power measurement, the tendency is always to oscillate around the MPPT especially in cloudy days when the MPTT is changing rapidly. So it is really common that the algorithm continue perturbing the voltage moving the characteristic to a certain direction trying to approach the MPPT while this last one is moving rapidly to a different point. This lead to a loss of power that potentially could have been transformed.

### Constant voltage

The constant voltage technique [30] regards the study of the ratio between the maximum amount of power voltage obtainable in the PV array " $V_{MPPT}$ " and the open circuit voltage " $V_{OC}$ ". This is defined as a constant:

$$\frac{V_{MPPT}}{V_{OC}} = K < 1 \quad (2.2)$$

The value of " $V_{OC}$ " is computed disconnecting temporarily the array from the "MPPT" controller. The correct operating point is evaluated through the equation (2.2), establishing the constant parameter "K" in order to reach the maximum power voltage

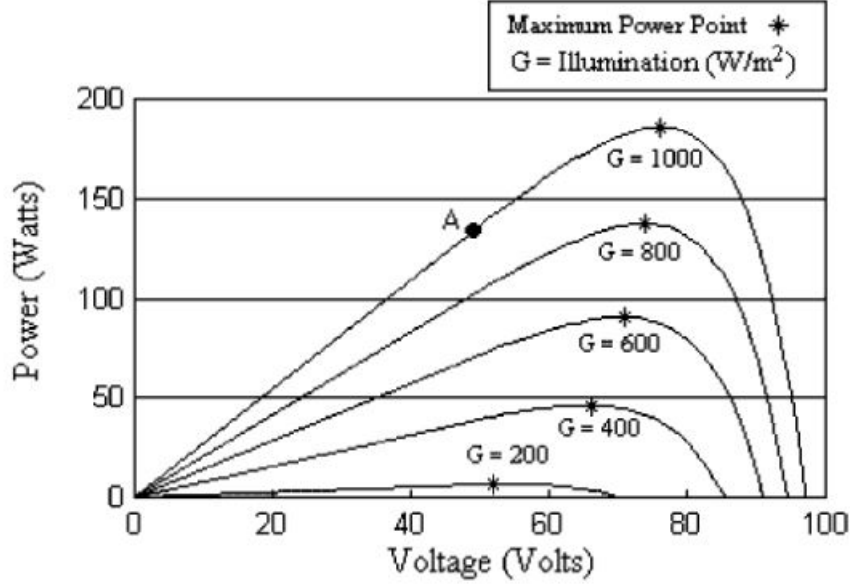


Figure 2.5. Family of PV Power-Voltage characteristic[30]

" $V_{MPPT}$ " adjusting the array voltage until this last value is reached. This operation is repeated continuously to evaluate the correct position of the MPPT. This algorithm can also be described using a flowchart represented in Fig. 2.6. Although

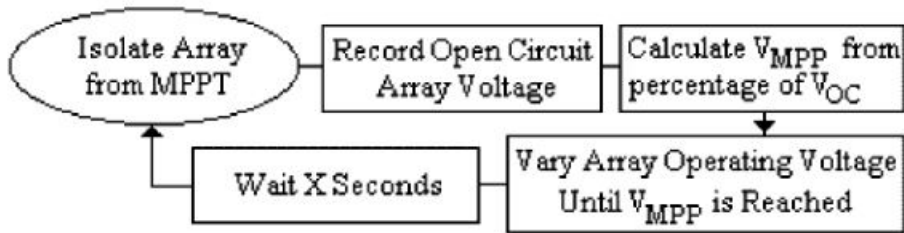


Figure 2.6. Flow chart constant voltage algorithm [30]

constant voltage algorithm is really easy to be implemented, using analogue hardware, this could be affected by errors in evaluating the K factor due to multiple approximations. Furthermore the value of "K" could also be dynamically adjusted, but this will lead to use a searching algorithm leading again to the P&O technique.

### Incremental conductance

The incremental conductance technique [30] is the more accurate with respect to the previously described technique but requires more computations. It is obtained through differentiating the PV array power with respect to voltage and setting the result equal to zero at MPPT as shown in (2.3).

$$\frac{dP}{dV} = \frac{d(VI)}{dV} = I + V \frac{dI}{dV} = 0 \quad (2.3)$$

The following equation becomes:

$$-\frac{I}{V} = \frac{dI}{dV} \quad (2.4)$$

The 2.4 represent: on the left side the negative conductance and on the right hand side the incremental conductance. This two quantities must be equal in magnitude but opposite in sign. If the MPPT moves towards another point, an unbalanced situation from the right hand side of the equation 2.4 happen. The magnitude and the sign of the variation is used by the algorithm to determine which direction to be undertaken for the perturbation in the voltage, which is repeated until a balanced situation is encountered. As shown in Fig. 2.5, when the irradiance increase, the MPPT moves towards the right side and to compensate this, the voltage must be increased. Since the irradiance is directly correlated to the current, a change on the amount of sunlight means a change on the "dI". A representation of the algorithm followed by this procedure is shown in Fig. 2.7.

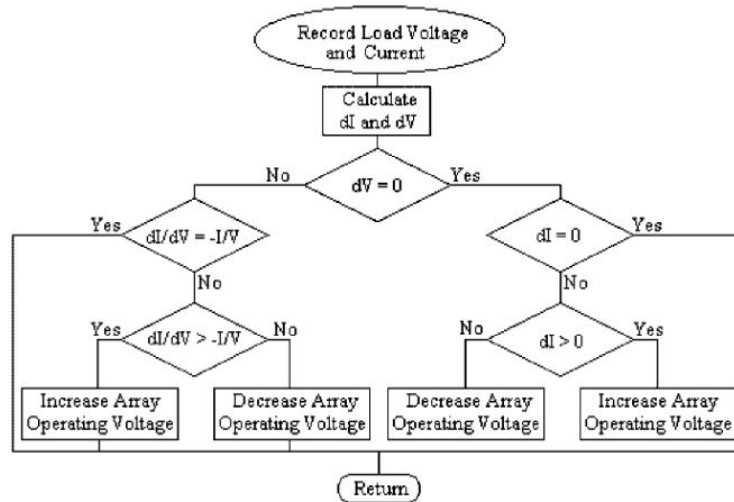


Figure 2.7. Flow chart incremental conductance algorithm [30]

The main advantage on this algorithm is that it can detect the direction of perturbation and also determine when it has reached the MPPT, waiting for any change before perturb again the voltage.

### Comparison MPPT algorithms

However the better performance of the MPPT algorithm is defined by the MPPT efficiency evaluated in these different cases. It is computed the efficiency with the equation (2.5), where " $P_{REAL}$ " is the actual power measured at the terminals of the PV array under the control of the MPPT algorithm and " $P_{MAX}$ " is the max power reachable. The table 2.1 provide comparative values for the algorithms analyzed.

$$\eta_{MPPT} = \frac{\int_0^t P_{REAL} dt}{\int_0^t P_{MAX} dt} \quad (2.5)$$

MPPT algorithm	Efficiency reported [%]
P&O	81,5-85
Constant Voltage	73-85
Incremental Conductance	88-89.9

Table 2.1. Comparison among different MPPT algorithms [30]

### 2.1.2 Solar inverter

A solar inverter is essentially an electronic power converter which is responsible for transforming the direct current "DC", coming from the photovoltaic panels "PV", into alternate current "AC" at a specific frequency (typically 50-60 Hz).

The AC current produced, is then connected to the electrical grid or directly used as a off-grid network [44].

#### Different topology of solar inverter

Depending on the configuration, there are different type of inverters [12]:

- *Stand alone* which are isolated systems, draining DC energy from the batteries, charged by photovoltaic arrays or other resources such as: engine generators, hydro turbines and wind turbines. They are not connected with any grid so that they are not required to have anti-islanding protections.
- *Grid-tie* that are connected back the the main utility grid and they require to match their phase with the phase already present in the electric grid with the same frequency rate. Moreover these last ones may also feeds electricity



directly to the electrical loads such as appliances, tools, HVAC, etc.. They are typically designed, for safety reason, to rapidly disconnect from the grid in case of utility outage following the concept of anti-islanding protection. In particular they provide an internal circuitry that detect the magnitude and the phase of the voltage and the current of the grid to be matched.

- *Battery backup* that mainly drain energy from a battery and they provide the energy during fallback situations where it is missing the AC power. The system include also an internal battery management system in order to restore the battery charge level when it drops down a certain amount of voltage.
- *Intelligent hybrid* which provide management of PV arrays, battery storage and utility grid and directly connected to the unit. This configuration include the all previously described typologies, so they are re-configurable depending on the situation.

### Structure of an inverter and working principle basics

The working basic working principle of an inverter is based on the use of high frequency switch able to let flow the DC current back and forth in a circuit connected with a primary winding. The earliest versions were provided with a electro-mechanical switch, now turned into electronic only based on the transistor behaviour as shown respectively on the left and on the right side of Fig. 2.8. As soon as electronic components are becoming more and more available to support also high power ratings systems, and due to their relatively small size, they became included into the inverter circuit design.

The alternation of the direction of the current in the primary winding is producing an alternate current AC on the secondary circuit which is present in a square wave form. In order to get the ideal sinusoidal shape, many electronic circuits are de-

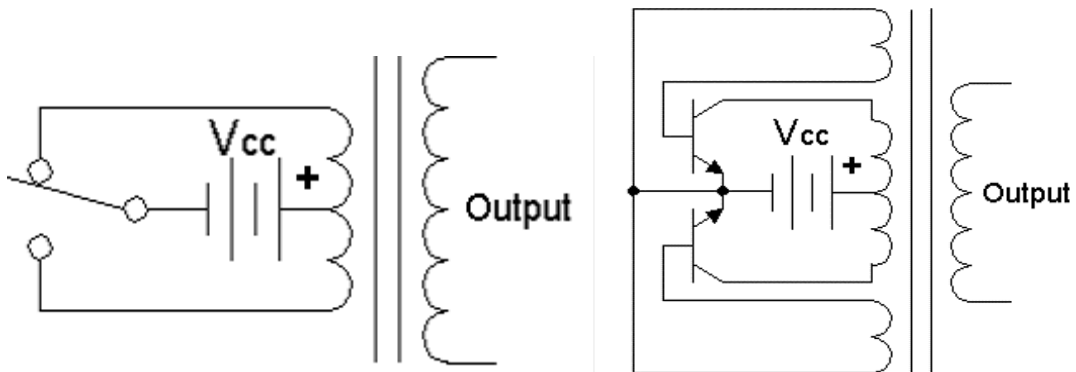


Figure 2.8. Simple inverter circuits [11]

signed to filter the waveform including capacitors and inductors depending on the real application. In order to be compared to the desired sinusoidal shape the Fourier analysis is encountered including the study of the harmonics and the coefficient of total harmonic distortion (THD) which is defined by the square root of the sum of the harmonic's voltage square divided by the fundamental one 2.6 [45].

$$THD = \frac{\sqrt{V_2^2 + V_3^2 + V_4^2 + \dots + V_n^2}}{V_1} \quad (2.6)$$

Basically the more the THD coefficient is low, the more the shape of the generated wave resemble the one of a sinusoidal one. Although the square waves presents harmonics that are required to be eliminated, different techniques are presented [34]. In a single phase inverter configurations such as half bridge and full wave bridge are involved as electronic circuit in order to provide the output waveform. One of the best solution to improve the quality of the waveform in output is using the pulse width modulated control (PWM). In order to eliminate as many harmonics as possible, the switching frequency of the PWM is required to be high.

### Three phase inverters

Basically three phase inverters consist of the grouping three single phase inverters each one connected to the load terminal. They are mostly used in motor drive applications or in high voltage transmission lines. The operation of the switches present in an inverter circuit, in Fig. 2.9, is designated in a way that they are switching at every  $60^\circ$  of the fundamental output waveform. Depending on the applications there are  $120^\circ$  and  $180^\circ$  degrees working mode operation. In  $120^\circ$  mode of conduction, only two switches are operating at the same time at each time step and each one of them remains active for  $120^\circ$  consecutively (2 steps) in a complete cycle. Conversely in  $180^\circ$  mode of conduction three switches remains active at the same time at the each time step and and this time each one of them remains active for  $180^\circ$  consecutively (3 steps) in a complete cycle. Finally the output waveform is provided in a total of  $(2^n - 2)$  steps, where n corresponds to the number of terminals (that for a three phase load corresponds to 3). In this way the output is generating a six-step output waveform in Fig. 2.10. Each line voltage ( $V_{AB}, V_{BC}, V_{CA}$ ), is recovered by performing the difference of voltage magnitude at the terminals ( $V_{AO}, V_{BO}, V_{CO}$ ) of the equivalent circuit for the combination of the active switches at each single step. Using two six-step circuit in series or in parallel increasing respectively the voltage and the current of the productive capacity of the inverter it is obtained also a correspondent 12 step output waveform, where phases are shifted of  $30^\circ$  each step. Furthermore, by increasing the number of steps in the output waveform, the output waveform is more defined and accurate to resemble the sinusoidal one providing a smaller THD. An inverter is able to work as voltage (VSI) or current source (CSI) to supply the load on the base of the configuration and on the request.

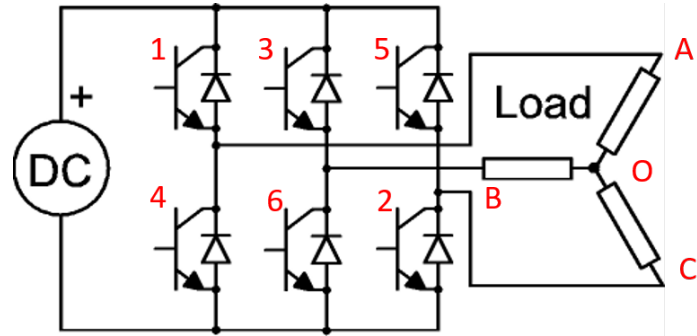


Figure 2.9. Electrical circuit of a three phase inverter [10]

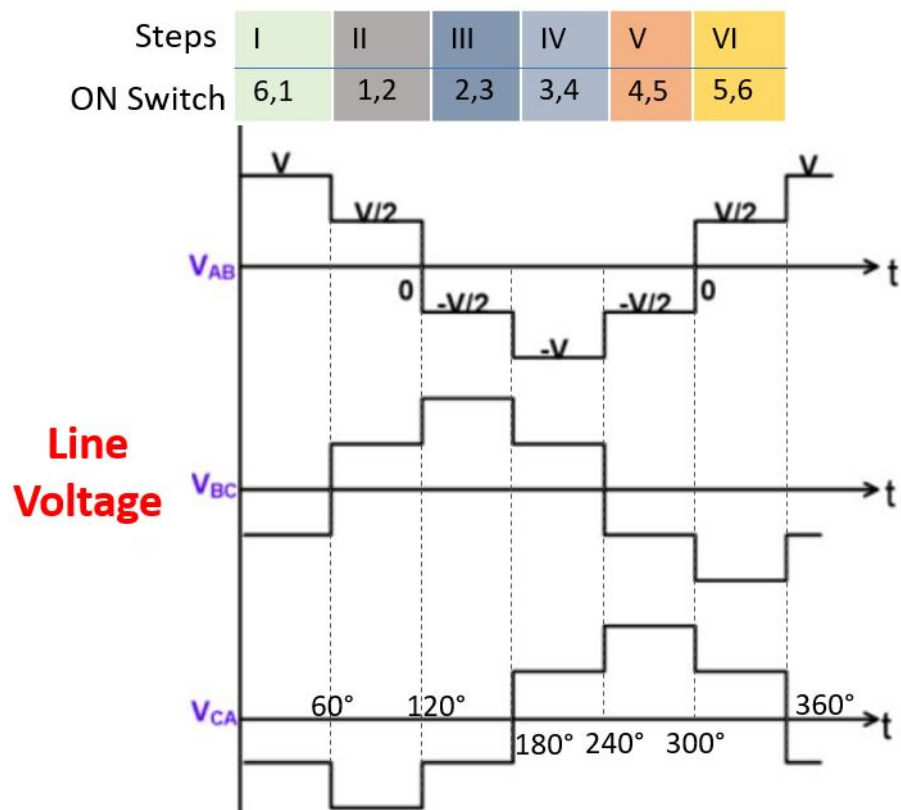


Figure 2.10. Fundamental output six-step inverter 120° conduction mode [17]



## Chapter 3

# A Resilient Information Architecture Platform for Smart Grid

The emerging trends on the field of computing platform shows the necessity of an additional computation layer based on distributed computation and communication resources able to monitor and control physical phenomena at the high levels where the commands and main data are managed. Example of these computation platforms are "Scale"[5] and "Paradrop"[46] used to collect fine-grained data and to filter data before sending to a cloud service.

In the Smart Grid domain, in a society made of increasing companies, communities and costumers, eventually "prosumers" able to provide themselves the energy consumed, the main idea is to distribute the computational effort in a way to create a net gap between centralized and decentralized structure. This assumption requires to monitor, control and manage software application at all levels.

### Centralized structure VS Distributed structure

Nowadays in most applications, the computing power is centered in one point and managed by a single control room which receive and sends data to the local loads. The computing effort is all concentrated in a single or multi devices that lead to a single component at the end shown in Fig 3.1.

The centralized structure is currently becoming old structure and it is not fitting anymore with modern applications because of :

- Sustainability since nowadays applications require its own dedicated control room.
- Scalability since a centralized structure cannot expand in significantly way.

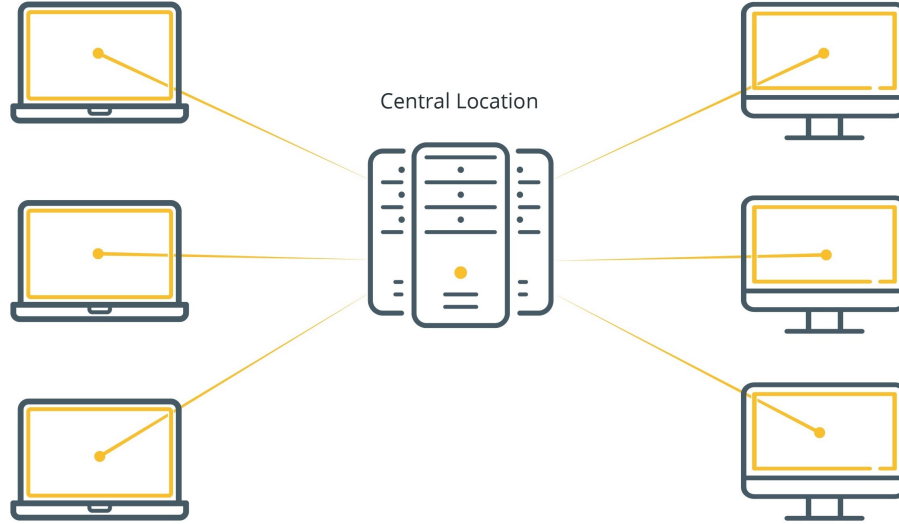


Figure 3.1. Centralized structure [40]

- Multiple nodes to be managed so the chaotic interconnection could slow down the speed of communication due to multiple link creating loss of efficiency.
- Slow time for sending and receiving data due to long round trips of data.
- Reliability directly linked to fault and accessibility.
- Resilience since if the fault is affecting the source, it is propagating to the peripherals in a non controlled situation.
- Accessibility since after the fault is issued any peripheral can be accessed during fault.
- Security since accessibility to all peripheral is established if access to central node.

Conversely a distributed paradigm structure is necessary in order to communicate with all devices across the network solving problems collaboratively and exchanging easily data among nodes facilitating the communication. It is responsible to spread the "intelligence" throughout all connected nodes and to rely on a multi platform implementation rather than a single one Fig. 3.2. The advantages of the distributed computing in the context of the micro-grid application include [13]:

- Improved cyber security by distributing the control point in multiple ones.
- Improved physical reliability by removing a single point susceptible to failures or damage.
- Faster decision making avoiding network penalties due to round trip to the cloud.
- Improved scalability to the new applications to operate with different number of load, sensors, actuators, relay and in general grids.
- Provide a better integration with hierarchical control systems.
- Modularity reducing the big system in a smaller one allowing the interaction among multiple small systems.
- Capability to distribute the center of command in a wider geographical area.

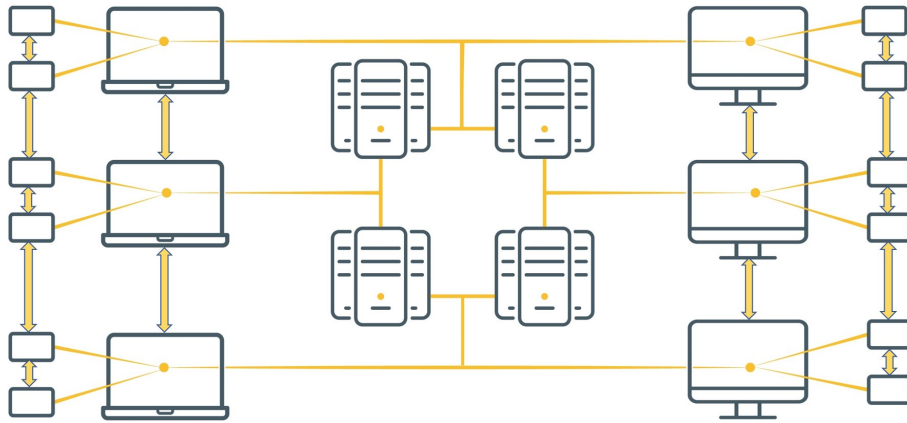


Figure 3.2. Distributed structure [40]

## 3.1 Challenges and expectation

The main role of these computing platforms is to provide a stable environment application development and deployment, that has resilience to the heterogeneity of different applications, to the dynamism of the occurrences that may happen and to potential failure of computing resources. All these knots lead to a solution, that expect a computing platform able to provide the primitives for the service core, necessary for the safely deployment, mitigating at maximum the risk of failure. On this purpose, a very powerful innovation algorithms has to be considered including time synchronization, distributed data management and service discovery and deployment mechanism for remotely manage distributed applications[16].

## 3.2 What is RIAPS

Resilient Information Architecture Platform for Smart Grid, better known as RIAPS, is a computing platform for prototyping real-time embedded applications using a component-oriented approach [25].

This software platform has been edited by Vanderbilt University and the applications developed by North Carolina State University and Washington State University.

Riaps is also defined as a "middleware" which is a multipurpose software that provides services and functionalities to applications outside of what is offered by the operating system itself. A middleware could be considered in general as a software layer that stays between the kernel and the user applications [29].

In this context, RIAPS is also responsible to provide a way for a bidirectional communication between the operating system and the developed application, that yields to a platform easily accessible and manageable from the user side.

### Usage of RIAPS

The main role of Riaps is to serve as a software platform, for the implementation of various functions. Typically it is used in the Smart Grids field for distributing the "intelligence" in different local points. In this occasion RIAPS is able to spread the computing power in multi command centers called "nodes" in a way to decentralize the original and unique center of command that was widely used before. Each node has its own independence and intelligence acting as well as the centralized one but with high efficiency Fig. 3.3.

### Characteristics and main features of RIAPS

Despite many software platform, Riaps provides a diverse number of services related to the domain specific logic, this include [13]:



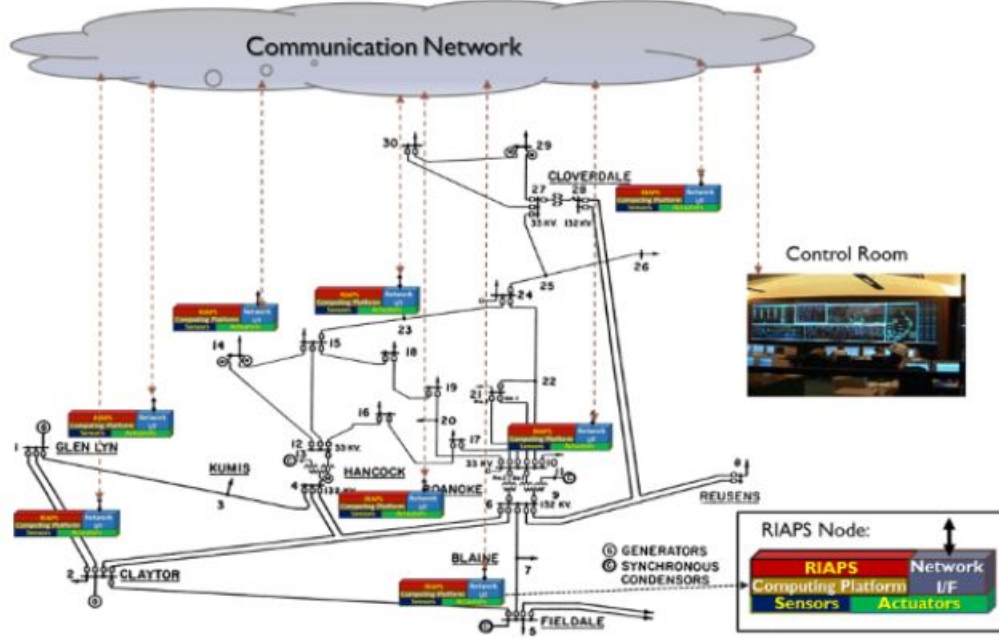


Figure 3.3. Riaps as a distributed computing platform [24]

- Time synchronization to improve control performances.
- Messaging middleware through which it exchange data to all user application.
- Coordination system and consensus to synchronize all the mechanisms inside.
- Discovery and deployment system mechanism able to deploy simultaneously multiple applications for multi-tasking operation.
- Fault-detection and recovery mechanism provided as prove of resilience by significantly reducing dependencies for each node.
- Distributed security mechanism in order to distribute the capability of reason to local end-points that prevent from cyber attacks and mitigate the risk of failure.
- Fast decision making, avoiding latency during data exchanging due to direct linking to the local end-points.
- Security by improving the access authentication through using secure keys files and codes, creating a reserved access.

- Progress, due to the updates of the platform which is able to provide always last functionalities in a world of continuous evolution.
- Interoperability between relatively small systems, able to communicate each other, exchange information and data, necessary for the success of the operations.
- Reliability of the system studied in minimal details to always fit with a number of different of application.
- Reusability due to multiple reuse of the platform in multi implementation, this include also small modification for slight different application.
- Easy implementation due to e-learning and the on-line data provided by the developers.
- Support from the developers for any kind of problem related to the software platform (settings, connections, clarifications and debugging)

### 3.3 System architecture layout

Riaps software platform has a complex structure inside which is able to coordinate everything together. The layout of its architecture is describing in details every single part which is fundamental for the correct execution of the platform and gives an important contribution to each component.

#### 3.3.1 Architecture run-time system

In order to produce a reliable system on which distributed applications can be built, Riaps relies on a open source operating system.

Riaps includes two main group that includes packages that works together to perform their work inside the platform. They are the *Component Framework* and *Platform managers*. The structure of the software architecture is shown in Fig. 3.4.

#### Component framework

The Component Framework includes a set of software libraries that are dynamically linked with the application components, it is close to the OS kernel interactions. The Component Framework layer is responsible for providing a higher level abstractions for building distributed applications on the platform that present the resilient and complex but reliable characteristics.

The middleware libraries includes the [16]:

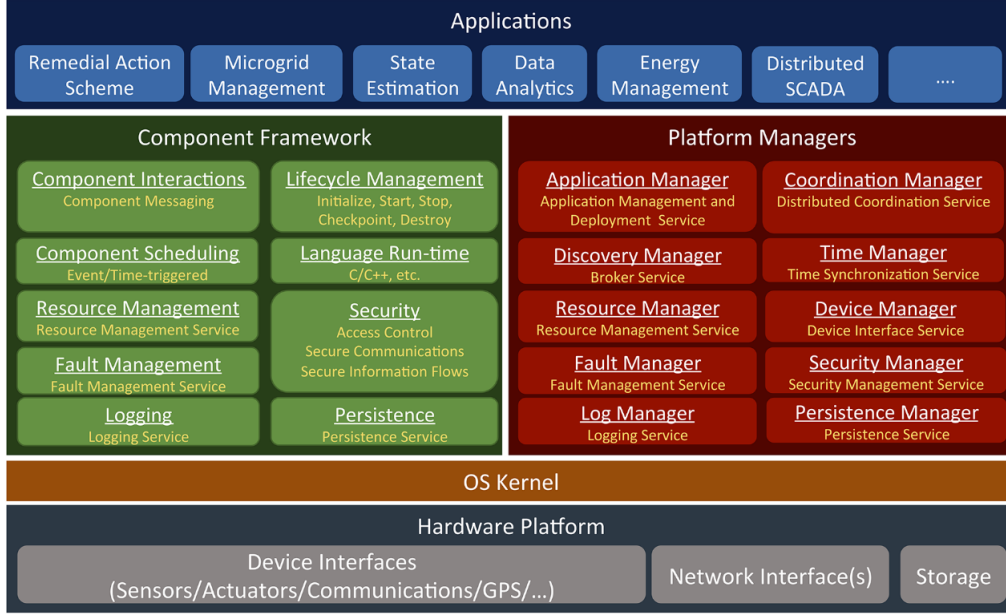


Figure 3.4. Software architecture [16]

- *Component Scheduler* that is responsible for managing the handler functions every certain period or once an event appear through implementing the component execution semantic.
- *Component Interaction* that enables the communication and the remote methods of invocation on the same node or in the network.
- *Resource Management* that provide support on the monitoring of the computing platform utilization and availability.
- *Fault Management* that provide support on detecting and mitigating some anomalies in the software components.
- *Logging* that provide support on recording components events such as messaging and errors track in a log-terminal or a log-file.
- *Life-cycle Management* that enable the remote managing of the software components.
- *Language Run-Time* that enable communications and interaction between the languages supported by the platform.
- *Security* that is able to guarantee private and selective access through key access.

- *Persistence* that provide functionality on containing and storing data.

### Platform managers

On the other side the *Platform managers* are specialized operating system processes, implemented as host in Linux systems responsible for implementing the system-level management capabilities. The platform managers incorporate the elements of the application framework, this include [16]:

- *Application Manager* that provide the ability to install and remotely manage the applications.
- *Discovery Manager* that provide support by proving the connection availability on the different nodes.
- *Resource Manager* that is able to provide functionality on monitoring the computing resources ensuring that Platform manager and components are able to run at the same time.
- *Fault Manager* that give support on detecting fault situations at node level.
- *Log Manager* that serves as a entry point to all activities on the node.
- *Coordination Manager* that provide fault-tolerant distributed services as well as coordination service.
- *Time Manager* that is able to give high precision time synchronization service.
- *Device Manager* that provide support on enabling input/output devices to have access to the messaging framework of the platform.
- *Security Manager* that handles the authentication through keys secured by digital signatures.
- *Persistence Manager* that enable to store data in a persistent manner.

### Applications

The *applications* are located in the top layer according to the Fig. 3.4 and obviously they rely on the services provided by both *Platform Managers* and *Component Framework* to perform their operation.

### 3.3.2 Application model

One of the most important thing in using this software platform is the study of the application. An application consist of a set of files. One of this file, with the extension ".riaps", includes all the information for the other files inside the application, such as message pattern, interactions and deadlines. It is written in C++ and it is giving the very first information of what concept is trying to exploit the system developed.

An application shortly called "app" in mainly composed by:

- Components
- Actors
- Message topics
- Ports

#### Components

As it was previously said, the application model is composed also by components. Components are objects and play an important role in the application development, because the are able to interact with other components trough sending and receiving messages.

The components could be single-threaded, that means they can afford just one operation at a time or multi-threaded that means the manage many operation at a time. A component run its own execution thread and the component execution engine release a thread, i.e when a message arrives that the component was expecting, Components have ports that are used as arguments to send and receive operations [24] as explained below. The responsible for the exchanging of information through the use of message topics is the message framework and it is explained better in a dedicate section.

#### Devices

Devices are special type of components that are able to interact with external threads and provide a correct operation during execution able to handle situations in which external sources or threads interact with the system.

#### Actors

Actors are "OS" processes and run on a host of a Riaps network [24]. They groups all components present in the application. There could be the possibility to find one or more actors in the same application, grouping different components inside.

Actors could be deployed in different target nodes, or simply replicate them in the same nodes. The reason why the components are grouped into actors is that the communication between components inside a single actor is way cheaper than the communication between actors running on the same node or in the worst case communication between actors running on different nodes. The actor is also responsible for loading a component, setting up its configuration and initializing its state.[16].

### Message Topics

The message topics are really important due to the capability to carry on informations from one side to the other and so they represent the means of transport for the communication among components. There are different message topics depending on the purpose they serve, they are differentiated and analyzed better in the messaging framework section.

### Ports

Ports are the elements through which components interface with other components. There are special kind of ports that are the "timer" ports, that acts as a programmable one-shot or periodic source of messages that provide the current timestamp for the component [24].

## 3.4 Component framework

As soon as the *Application model* is describing various components inside an application, this section is analyzing the way this pieces are grouped all together into a single process to be run.

Components are grouped into actors and run in applications. Once the link is established it is created a node, inside where components can communicate and interact using well defined patterns. In Fig. 3.5 is shown an example of nodes creation inside the Riaps platform. The advantage of having a component framework based on a software is that allows the platform to have reusable building blocks of applications due to the grouping of all the actors into a single process as previously said.

More advantages other than reusable components are that the concurrency among them is managed inside the framework and not in a logic condition as well as the triggering logic is encapsulated in a function that can handle complex decision, finally the timing analysis is performed inside the platform itself and not handled externally.

In Fig. 3.6 the basic functions and interactions of a single component are better explained.

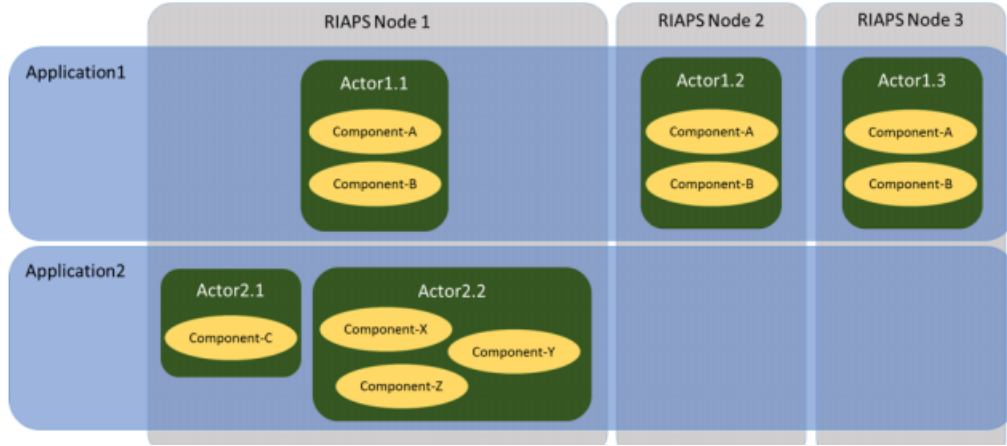


Figure 3.5. Riaps nodes [33]

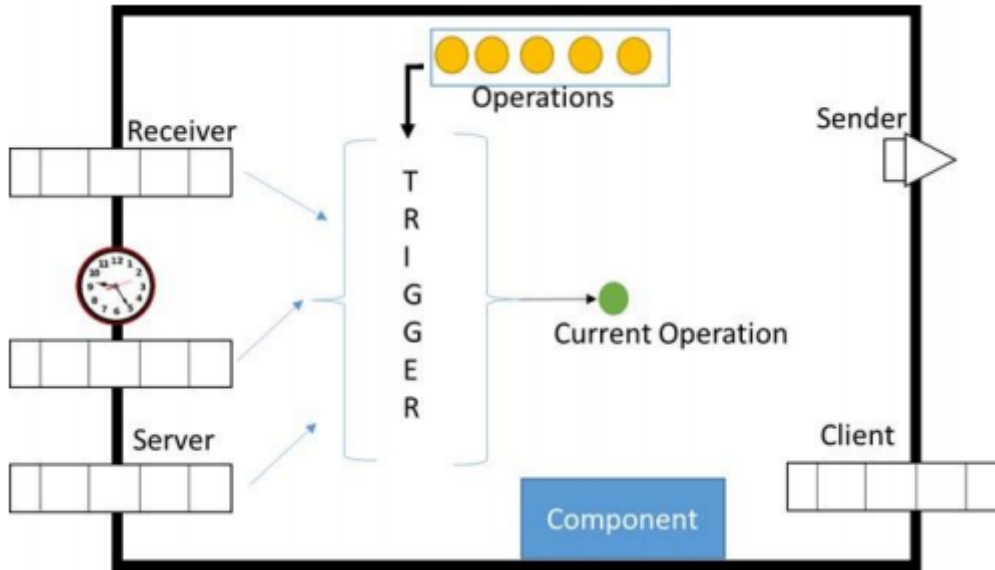


Figure 3.6. Component Framework [33]

Components have state and interact via ports that allows the components to send/receive messages and to be server/client in a broker network. The study of this platform demonstrates that the expected scale of node creations could push up to  $10^2$  nodes [33].

## 3.5 Components interaction

The components interaction could be understood better if the message type are clear. Different message type are present through which the components participate in a various interaction. All the components must implemented using these message patterns. The supported interaction pattern are described later in section "*Messaging Framework*".

### 3.5.1 Component execution engine

A component in its execution can be time or event-triggered. When a component is triggered, either because of a message arrived or because a timer expires or even when an operation is completed, there is an associated handler method which is responsible for calling a component object. Components in the same actor run in separate threads, so an actor which contains single or many components is multi-threaded [24]. This means that the handler that is called have to deal with many thread that can produce different triggering message to be read. The handler act like a normal function, so it is responsible to read this message, execute some operations and return the value to the output ports. The picture in Fig. 3.7, is depicting the basic operations deployed by a component during its execution.

In the configuration in Fig. 3.7, the component is running in a "undisturbed" situation, in which any other external source is not affecting the basic operations.

### External device management

Different situation happens whenever an external device is interfacing with the component. The basic operation can be interrupted and the handler function may loop in unbounded manner.

To handle this problem, Riaps platform makes use of the device component, which behave such as a normal component, but has also the capability to interact with threads coming from external sources i.e. I/O requests from external devices.

A device component can launch one or more threads that runs parallel to the main thread communicating through inside ports. Inside ports are special type of ports where internal threads can exchange data to the outer threads, improving the system of communication between external device and components avoiding also blocking processes that can end up with failure in the whole system.

A typical implementation that shows an I/O external device interaction in a normal operation of a device component is shown in Fig. 3.9.



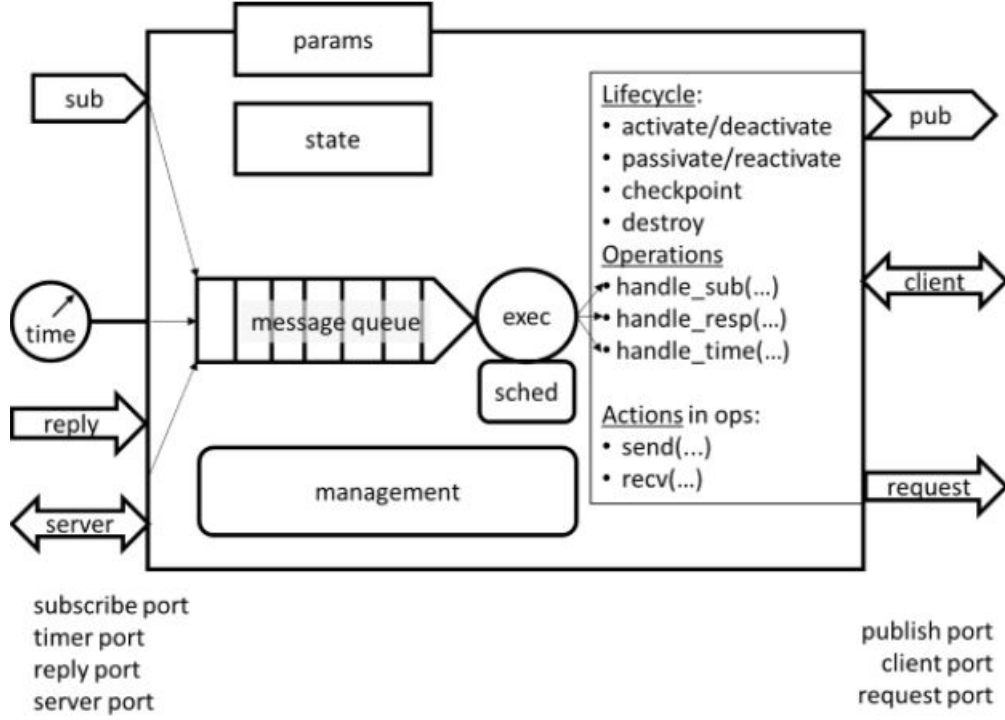


Figure 3.7. Component execution [24]

### 3.5.2 Device interface service

Since Riaps nodes are continuously varying on number and type, depending also on the hardware devices used, the configuration, to allow interactions basing on available utilization of the node, is also changing rapidly to meet certain necessary requirements. After all, depending on this configuration, the device interactions also change over time and may be added and removed depending on the necessity.

Naturally, Riaps platform is designated to run as a software, so it could never directly interact within power systems, but it only interact with them through the device interface service with small power signals.

There are specific ports and interfaces implemented by the "DIOC" (Device I/O components) related to the connected power system device, the lower-level protocol and the physical link.

The advantage of using this "DIOC" is that the I/O devices are included and for this purpose the application component is able to [14]:

1. Access them using a unified interface.
2. The timing interaction between the devices is highly accurate.

To reach these goals, the device components are provided within drivers, resource methods and real-time scheduler in order to externally interact within different applications.

In fact, every single match between I/O connection and software protocol, labeled as industrial standard such as RS-232, RS-485, TCP/IP on Ethernet, I2C, GPIO, is a device connection point. Other low level protocols are: Modbus, DNP3 and IEEE 61850.

Reference to the Device interface service describing the previous structure is illustrated in Fig. 3.8.

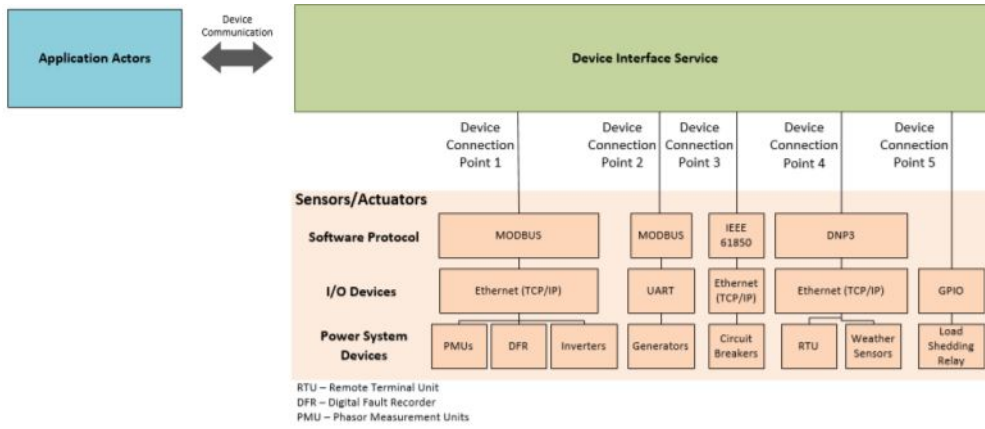


Figure 3.8. Device interface Service [33]

### Definitions and working principle

- The *Device Management Actor* called "DMA" is responsible to provide services to the device connection request from the actor inside the application and it is also called to start and stop monitoring the status of the connected devices.
- The *Device Communication Actors* called "DCA" instead is the means of communication between the devices and the Riaps node.
- The *Device Configuration Metadata* which is also labeled as "DCM" is to describe how specific node hardware are configured.
- The *Device Interface Metadata* called "DIM" which is the one configuring and identifying the specific power system devices that the application will use during its run-time.

The working principle is hereby shown: the "DCM" register the "DCA" to the "DMA" informing which power system device is available through the connection

point.

Each one "DCA" is initialized on demand, once this latter one is registered to the service.

The application actor is deployed together with the "DIM" since it is part of the configuration files of an application. Instead the implementation of the "DIOCs" describes above is not part of the application, because they only take part of the "DCA" and the application component is linked to the "DIOC" only during the execution.

The actors inside an application send a connection request whenever a power system device is registered to the service, in this way the "DMA" send information so that it can initialize its communication with the "DCA".

Moreover the communication between "DIOC" and "DCA" is allowed also through the use of the *Riaps Broker Service* that provide communication between the physical devices i.e. power system and the platform abstractions.

The "DMA" is also monitoring the status of connections in case a power system abruptly disconnect from the service, it immediately send notifications such as errors and the "DCA" remove it from the working node connections.

Once the connection has been established, the application actor and the "DMA" are in communication, and they can exchange informations in order to manage real-time access data.

Furthermore, there is also the presence of the "API" that are defined in a way to provide all the functionalities needed to access all the power system facilities.

The "DCA" is also able to intermediate between the physical and the software protocol providing in case of necessity also notifications based on the current time managed by the time-synch service.

The overall structure, can only support the interaction patterns intended as bounds between the physical power system device and the application actor listed below [14] [33]:

- *Sporadic input*: this is the interactions when the sensor is reading at an arbitrary time, that means the I/O sensory is generating a new timed sample and this is sent through publishing messages to the interested applications.
- *Periodic input*: the interaction related to the continuously reading from the sensor which is also called "stream" due to uninterrupted operations. This in operation could be associated as a pumping data to the *Device interface service* that then is distributed along the interested applications.
- *Sporadic output*: Is the interaction that could be described as the command emitted to an actuator at an arbitrary time. An application is responsible to generate the output as fast as possible, the request is done asynchronously due to the fact that the application does not explicitly wait for the actuator to receive the notification of successfully done operation.

- *Periodic output*: this interaction is when the application command the service to provide a periodic actuator output. The service as a result, launch a thread that read the value, then send the value to the actuator and the operation repeats in loop within a specific frequency.
- *Scheduled output*: is the interaction describing the situation when an application need the execution of actuation commands at a specified time or basically at a certain operation point. This "schedule" is obtaining through explicit request from the service and executed in the exact meeting point. The operation could also be rescheduled to be executed multiple times.

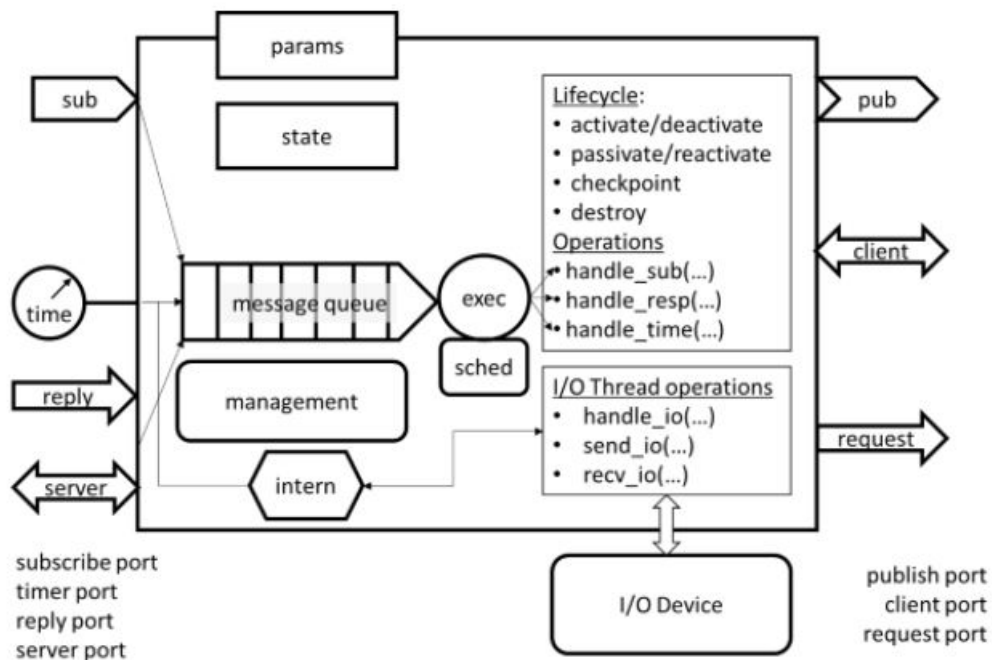


Figure 3.9. External device component execution [24]

In Fig. 3.10 is represented the communication between components including a legend exploiting the basic operations.

### 3.5.3 Messaging framework

The supported interactions [24] are the following:

- Publish/subscribe pattern: publishers produce messages of specific topics that are delivered by the framework to subscribers to that topic. Publishers and

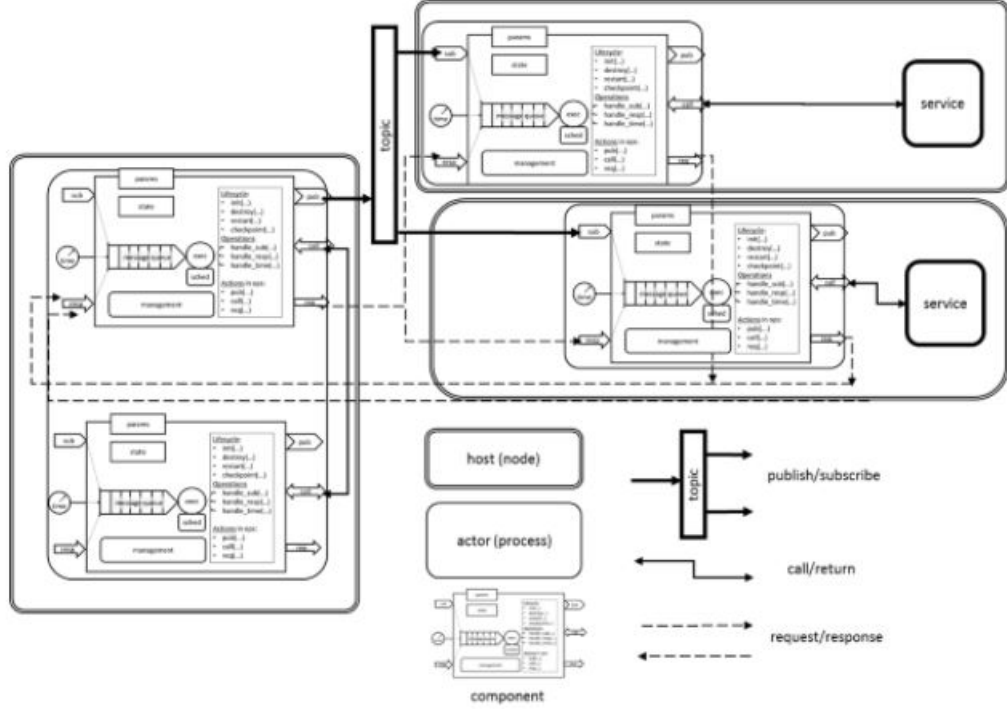


Figure 3.10. Interactions between components [24]

subscribers are anonymous, and the interconnections are many-to-many and interaction is asynchronous. Subscriber ports can specify a deadline for the completion of the message by adding a deadline period value to the port description[26].

- Request/reply pattern: a component send a request message and expect a reply message from another component. The interaction is asynchronous: the sender does not explicitly wait for a reply and when the reply arrive, the sender component is triggered again. The new request cannot be sent until the reply to a previous request has been received. In other words, request sends and reply receives must progress in lockstep.
- Client/server pattern: similar to the request/reply pattern, except the interaction is synchronous, so the sender component must explicitly wait for a reply message. The same lockstep rule applies.
- Query/answer pattern: similar to the request/reply pattern, except the lockstep rule is not enforced so arbitrary number of requests can be sent even if a reply is not received.

### 3.5.4 Resource and fault tolerance framework

The resource and fault tolerance framework is one of the most important services in the Riaps platform because include the ability to manage the system in the best conditions.

#### Resource Management Specifications

In order to avoid having completely busy the physical resources, it is used the resource management tool. During the code creation phase, the developer can specify a certain amount of system resource usage as a limit to be reached by the software platform. This lead to limit the computation effort or the memory usage and even more that could be useful for other processes running in background.

When the actor attempts to exceed the limits, it will receive a special message from the deployment service and the application component can react to this message [24]. The limitation is specified at the actor level through the "uses.." clauses for the following resources:

- *CPU utilization* where the usage value is an integer representing the percentage of the CPU that the actor is allowed to use added after "cpu" keyword. This could be either hard constraint if the keyword "max" is added or soft constraint if no keyword is present.

The limitation could be done for a specific amount of time if specified by the keyword "over" before the number indicating the time in "ms" (milliseconds) as default or "sec" (seconds) or "min" (minutes).

An example is shown in Fig 3.11



```
cpu max 10 % over 1;
```

Figure 3.11. CPU utilization example [26]

- *Memory utilization* where the memory usage limit is also specified by an integer value added after "mem" keyword representing an amount of memory available to the actor expressed as an integer value .The unit of measure supported are "kb" (kilobyte), "mb" (megabyte), "gb" (gigabyte).

An example is shown in Fig. 3.12.

```
mem 200 mb;
```

Figure 3.12. Memory utilization example [26]

- *File space utilization* where the space usage limit is also specified as an integer value added after "space" representing the amount of disk usage to be consumed by the specified actor. It is expressed in "mb" (megabyte) or "gb" (gigabyte). An example is shown in Fig. 3.13.

```
space 10 mb;
```

Figure 3.13. Space Utilization example [26]

- *Network bandwidth utilization* where the bandwidth limit is always specified by an integer number added after "net" defining the limit rate value that an actor is allowed to use. It could be expressed in either "kbps" (kilobytes per second) or "mbps" (megabyte per second). New features are added that are ceiling value as "ceil" that is the maximum amount of bandwidth that an actor can borrow by allowing an extra bandwidth available and the "burst" rate value that specify the amount of data to be sent at maximum speed before another actor use the very same network, only expressed in "k" (kilobytes). An example is shown in Fig. 3.14.

```
net rate 10 kbps ceil 12 kbps burst 1.2 k;
```

Figure 3.14. Network Utilization example [26]

A complete example [26] including all these feature in execution is shown in Fig. 3.15.

```
actor ActorName1 {  
  local aRequestMessage, aReplyMessage;           // Local message types  
  uses {  
    cpu max 10 % over 1;                          // CPU limit  
    mem 200 mb;                                    // Memory limit  
    space 10 mb;                                   // File space limit  
    net rate 10 kbps ceil 12 kbps burst 1.2 k; // Network bandwidth limits  
  }  
  {  
    componentInstanceName1 : RequestComponent;  
    componentInstanceName2 : ReplyComponent;  
  }  
}
```

Figure 3.15. Resource Management application [26]

### Fault Tolerance management

As a matter of fact, faults could happen at different levels. This includes faults in components, actors, system services or network. The software platform is responsible to detect such faults reacting to them either to [24]:

1. Automatically restarting the application interested in the actors.
2. Informing the application components through a special message in a way to allow this last one to react immediately.

The fault tolerance management in Fig. 3.16, is also located in the section *Run-time services* which incorporates it during the explanation of the "Fault management" applied to the "Discovery service". The developer should create applications in a respectively manner such that the Fault management tool, which include the fault tolerance feature, could run without any problem.

In Fig. 3.16, is represented faults at different levels and the boundary within which these could expand inside the platform. Some faulty situation are analyzed such as [23]:

1. Reported application process termination.
2. Unreported application process termination.
3. Application resource limit violation.
4. Application component operation deadline violation.
5. Unexpected service termination.
6. Operating system crash.



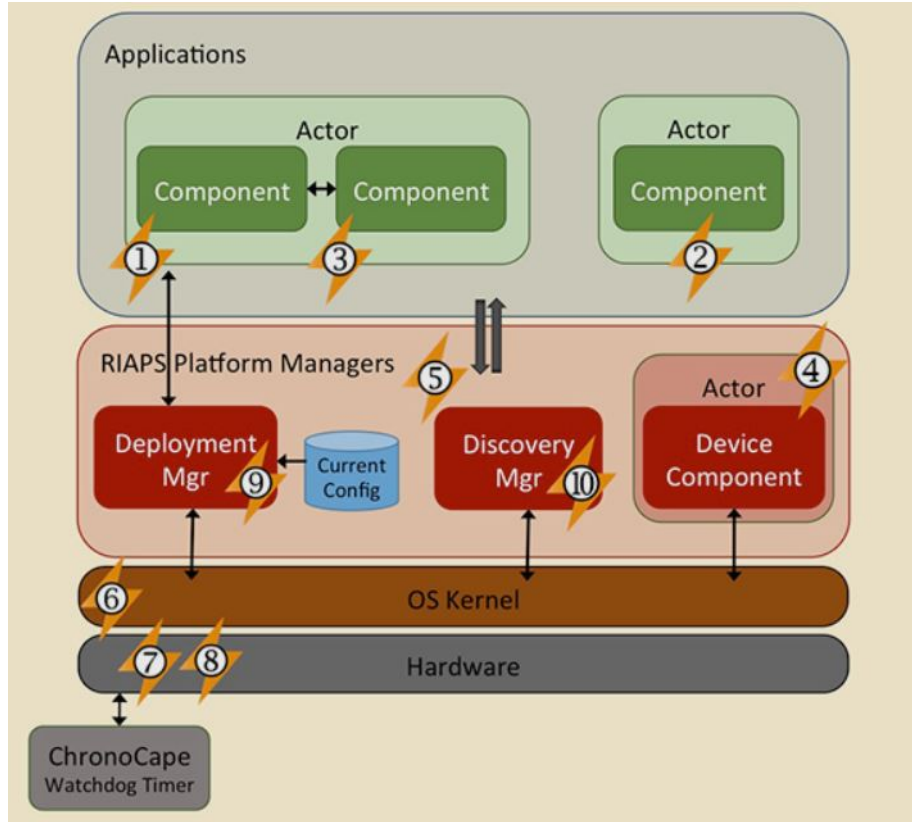


Figure 3.16. Fault management [23]

7. Network link failure.
8. Network node failure.
9. Application deployment failure.
10. Loss of connectivity to control station.

Countermeasures are taken for each level in order to restore the initial balance such as [23]:

- Detection which is the recognition of an anomalous situation.
- Isolation - regards the discovery of the root causing the problem.
- Mitigation - which is handled by application developers and describe the action taken to mitigate the consequences of the faults.

### 3.5.5 Security framework

The security is granted by the following steps [23]:

- Confidentiality of communication by encrypting all communications network and ensuring that the messages were not tampered with.
- Availability of resources by providing facilities for strict access control to resources and moderating processing activities to mitigate "DDoS" (distributed denial of service) attacks.
- Confidentiality of data by ensuring strict access control of data owned by an application to protect against malicious or faulty application code.
- Applications will be remotely deployed and controlled through the use of cryptographic signatures on the application binaries to be installed.

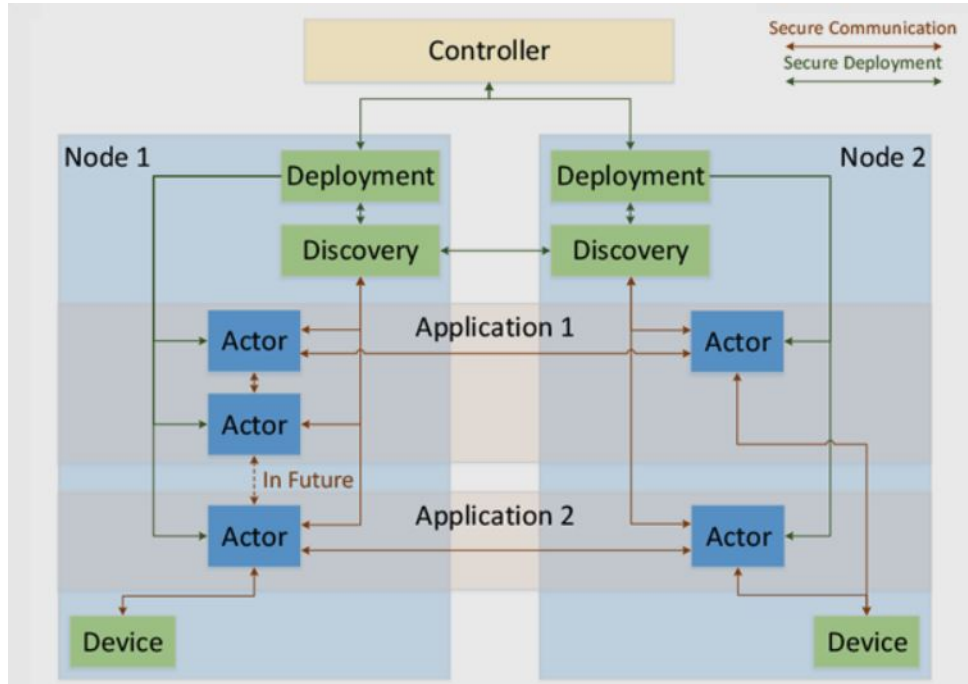


Figure 3.17. Security Network [23]

## 3.6 Run-time services

RIAPS as a resilient platform include many support services that manage different application functions to make them works in the best conditions.

### 3.6.1 Discovery service

The Discovery service is one of the most important services in Riaps platform because it plays an important role on the execution of this revolutionary software platform. Moreover, the discovery service is needed for a robust decentralized platform and is also envisioned as a critical service for the resilience of the platform itself.

The main role of the Discovery service is to keep track of all the links between the message producer ports such as publish, request, query, client and the consumer ports such as subscriber, reply , answer, server [24]. The matching between the producer ports and the consumer ports is established automatically based on the underneath framework hidden to the developer. The discovery service has two implementation[24]:

1. Distributed hash table "DHT" where a hash table contains the registration, and then this last one is distributes across the network. It is used "OpenDHT" which is a fast and lightweight platform able to store, query and disseminate the service detail through the network. The usage of DHT is not distinguishing the nodes as a client or server, because they are all peers and are applied to all the same rules[16].
2. Centralized database where a centralized database keep track of all registration nodes. It is used "Redis" (Remote dictionary service).

Obviously in a Riaps installation only one of this two methods presented before can be used for all nodes.

#### Working principle of the Discovery service

During the matching phase the message producer register with the discovery service and in the meanwhile, the message consumer look for the service that can produce messaging for itself. The linking operation is done asynchronously such that it may happen in any order at any time. In fact, if the matching is not succeed because of any reason at the first attempt, the component send an asynchronous notification and the matching is established afterwards.

The discovery service runs on each node as an independent process and listen for the message nodes either coming from the Riaps application or data coming through the network to system connected to the same service. In Fig. 3.18 it is possible

to analyze the main features of the Discovery service, which includes: receiving data from the application service such as message type, communication protocol, IP addresses and ports related. Moreover it is also responsible for storing the app service details and send new info to the local nodes[16]. The Discovery service

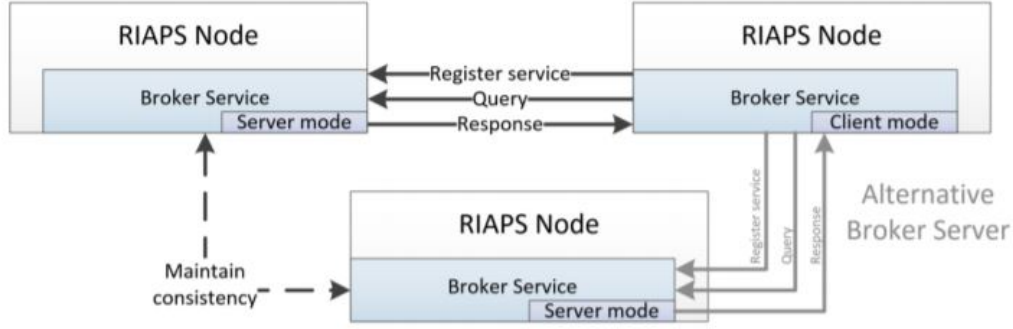


Figure 3.18. Discovery Service [33]

operate in two different instances:

- To find the nodes on the same sub-net of the network through the use of UDP-beacons which implements peer to peer communication. In fact each Discovery service periodically announce its network address and wait for incoming beacons. If no message is received from a node in two listening steps, the "silent" node is removed by the list of the known nodes. It is automatically added a new node which respond to the beaconing message storing its own address in OpenDHT.
- To find the nodes on different sub-net of the network, where the remote addresses are passed by the designated gateways running the Discovery service with different IP addresses on different sub-nets assumed reachable each others.

### Requirement to the Discovery Service

The discovery service is needed to keep track whenever an application is entering or exiting the network.

It useful to know in a network of communicating entities when :

- Two nodes join together in a group.
- A node leave a group (exit for any reason).

Another feature of the Discovery service is to check the status of the application service and the message types. The Discovery service has to be widespread in all the nodes such that it can access and share the information of everyone of them across the network.

When a Riaps application require an app service, it queries the local Discovery service and the result is asynchronously sent back to the Riaps application [16].

If a node join to a group the Discovery service is able to access and share data also to that node because of the data stored in the network, the same apply if a node leave a group so the Discovery service is able to communicate with the nodes still "alive".

### **Managing the ingress and egress scenarios**

OpenDHT is a service able to store the registered values and locally send them to a maximum of eight neighbours, forming a eight node cluster, in order to share them inside an application service registration data. Nevertheless, it is necessary to use this data stored across the network.

As already described, when a Riaps application start, it automatically register its services in the Discovery Service, then this last one store the data in the DHT that propagate the information through the group. During the registration phase of a Riaps node, it also subscribe to the needed service. If a compatible service is already in DHT, the discovery service send back a notification to the request from the Riaps application and so the application could connect to the service. If the desired service is not available, the Discovery service will call a function to request the application when this last one is available [16].

### **Fault tolerance of the Discovery service**

As previously said, the Discovery Service is also responsible for renewing the registration of application service inside the DHT. Before renewing, the service need to consolidate whether the application is still running and reachable or it left the group. The Discovery service alert the application services that abruptly leave the cluster without forewarning and have to manage also this situation.

Another important feature of the Discovery service in terms of fault tolerance, is that it is designated in a way to leave and restart the communication with components, in case this last one suddenly stop working. The components are not independent, and their communication to the Discovery Service is maintained through the actors. If the Discovery service fails, actors with components inside, continue to run silently without interfering with new services. Obviously new actors cannot be started until the whole Discovery service is not restarted. Moreover the Discovery service implement a secure coupling between each service data and the unique actor process identity (PID). Each time an actor register its data to the Discovery service, it looks

for the "PID" which identify the unique one to share data. Every time the discovery service checks if the couple "PID"/service established are valid by checking if the actor within its associated "PID" is still running. If the "PID" of the Discovery service is not on the list of the running process, the actor restart its registration possibly finding a new Discovery service. The fault tolerance is here explained: whenever a Discovery service stops working, it send notifications so that the actors can register to another one avoiding faulty situations. Briefly the Discovery service could be seen as a matchmaker of components/actor of an app find each other in a network.

### 3.6.2 Deployment service

The main role of the deployment service is to remotely install and manage the applications reachable in the network. The deployment services is also responsible for securing the setup, starting and stopping the application actors and it is a service associated to the control app. For the secure setup, the application actor taken as example run under a user-ID in a specific folder different from the other application actors. The Deployment service is shown in a sketch in Fig. 3.19. The deployment service as previously said is responsible for starting all Riaps processes that includes the application actors and the Discovery service analyzed in the previous section. In order to enhance the fact that the fault tolerance is present, it is possible to de-

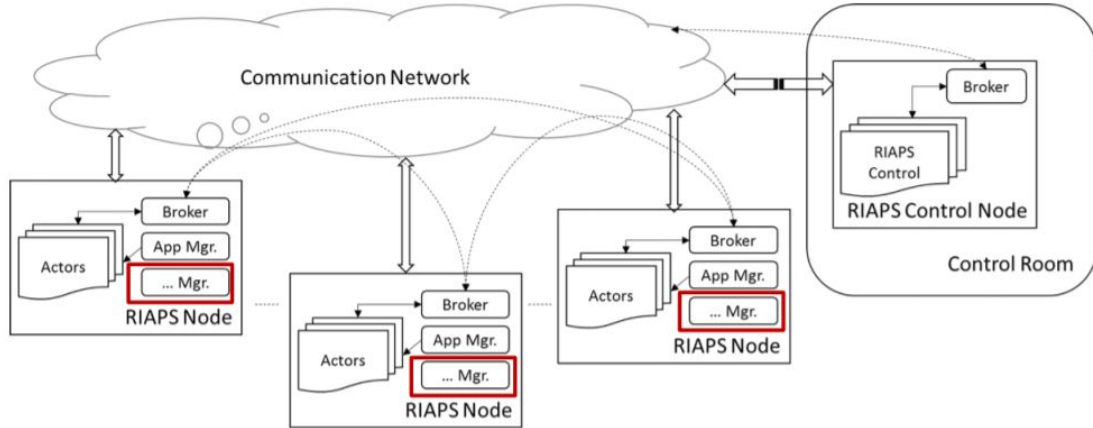


Figure 3.19. Deployment service [33]

scribe the failure situation in which a centralized implementation could be seriously dangerous for the whole system.

Fault tolerance is also supported by the discovery services which take into account the fact that if there is a single point of failure, the whole system cannot crash,

but only the single application running on the specific actor, that is able to be re-launched autonomously through the discovery service process.

### Working principle of the Deployment service

When the deployment service starts its process, the first step that is trying to do is to connect to the Riaps control App. Whenever the connection is established, it automatically launch the Discovery service and listen to the Control App for commands. After received the commands, the Deployment service is also responsible for downloading the application inside the actor on the network. Then it is also able to launch the actor's applications and execute two main operations listed below:

1. Setting a communication channel in order to send and receive data such as exceptions in component code execution or the availability of a new actor in the network.
2. Constantly monitor the status of the actor which, if suddenly terminated, send notifications and it will automatically be restarted.

The Deployment service is represented by the "Application Deployment File" which is nothing but the ".depl" file. In fact this file is responsible for allocating the actors to the physical devices [26].

The keyword "on" is used to inform which actor has to be installed in the respectively hardware device Fig.3.20.

```
app TestApp {  
  on (192.168.1.101) ActorName1;  
  on (bbb-1234.local) ActorName2;  
}
```

Figure 3.20. File example ".depl" (a) [26]

It can also be used the "all" keyword followed by the actor names as shown in the Fig.3.21

### Control app

Previously it has been announced the presence of the "control app" which is not a service but acts as it is, because provide important features such as:

- Prepare the download package of the Riaps application.

```
app TestApp {  
  on all ActorName1, ActorName2;
```

Figure 3.21. File example ".depl" (b) [26]

- Download the package to the target nodes thanks to the deployment service.
- Provide commands to the deployment service such as start/stop or deploy/remove the application through the control app "GUI" (graphical user interface) manageable directly by the end user.

The control app, represented by a "GUI" (graphical user interface), is started by the "riaps\_ctrl" executable better explained in the subsection "Application deployment and control tool".

### 3.6.3 Time synchronization service

The Time synchronization service is responsible for keeping the precision of the system clock on the Riaps target node synchronized on the network.

It is based on the IEEE-1588 "PTP" (precision time protocol) for clock distribution. Whatever is the source for the accurate clock, "PTP" ensures that other clocks in the network start to follow the main source of clock [24].

Furthermore the time-synchronization service allows to maintain a group synchronized on the same local time and provide high accuracy on the timing.

The application involved can [33]:

- Query the global time.
- Sleep until a specified point in time.
- Query the status of the service.

The accurate timing clock could come from a GPS signal which is attached to a target node through "NTP" (network time protocol).

It is shown a sketch of the working principle of the Time synchronization service on Fig. 3.22.

### 3.6.4 Distributed coordination service

The Distributed coordination service is responsible for coordinating the applications distributed on the network. Since each node on the network can join or leave the



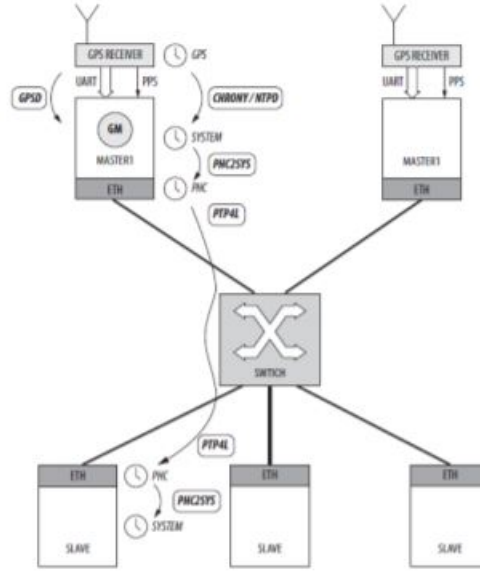


Figure 3.22. Time synchronization service [33]

group independently, the Distributed coordination service has to develop a leader election service over a group membership and a distributed consensus service with time coordinated action.

In this way a leader is elected for centralized functions and if it suddenly become unavailable, it automatically elect the next one in the pending list of the nodes. Furthermore the nodes on the group agree on a value on the base of consensus and finally it execute control action on multiple nodes simultaneously [33]. The main services till now mentioned could be also displayed in the Fig. 3.23.

### 3.7 Design-time tools

The design time tool section is a preview to better visualize the tools used to analyze what is described in the subsections below regarding the implementation details. However, Riaps platform is based on a model-driven development which means that the code is generated by the model itself. In this way the developer is more proficient in developing complex applications because is concentrates himself only on the code structure and not on the configuration of the model. The model is representing the component and the the composition of multiple components become an application where the developer provide the business logic in the ".riaps" file to be respected.

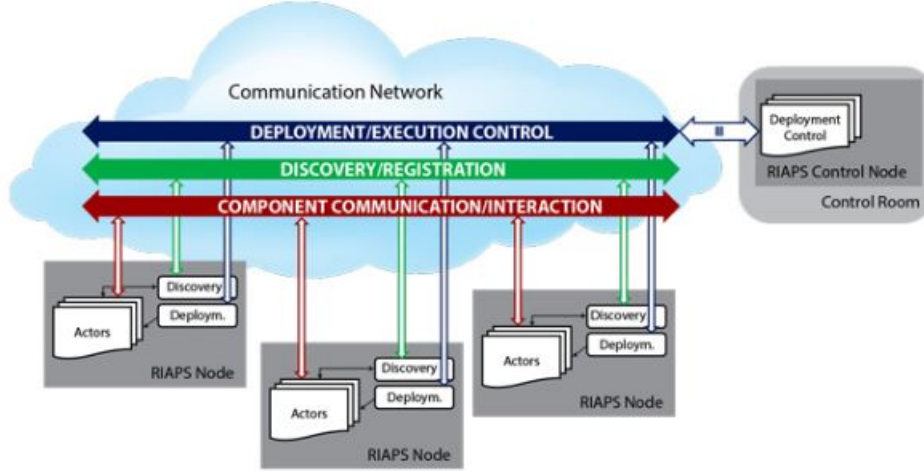


Figure 3.23. Riaps services [24]

### 3.7.1 Modelling language for the architecture

The modelling languages used for this platform are mainly Python and C++. Either programming languages are valid, but there is a difference in the applications developed. In particular the python language is mostly used for soft real-time constraints, i.e. applications that don't require any particular timing deadline or better the missing deadline could not degenerate in such as harmful situations.

On the other hand, C++ programming language it is more used for hard real-time constraints.

Riaps platform is using a combinations of this two programming languages to provide flexibility to the end user and easy application development for rapid prototyping due to higher level programming.

### 3.7.2 Software generators

As it has been previously described, Riaps platform supports python programming language. This could be compiled and tested by an eclipse based platform which is responsible to compile and also provide services such as debugging analyzed better in the *Debug tool support* section.

Moreover the Code based on IDE 'Eclipse' is responsible to : acquire the models, generate code, glue it with all necessary files and configure it and to leave it ready to run as shown in Fig. 3.24.

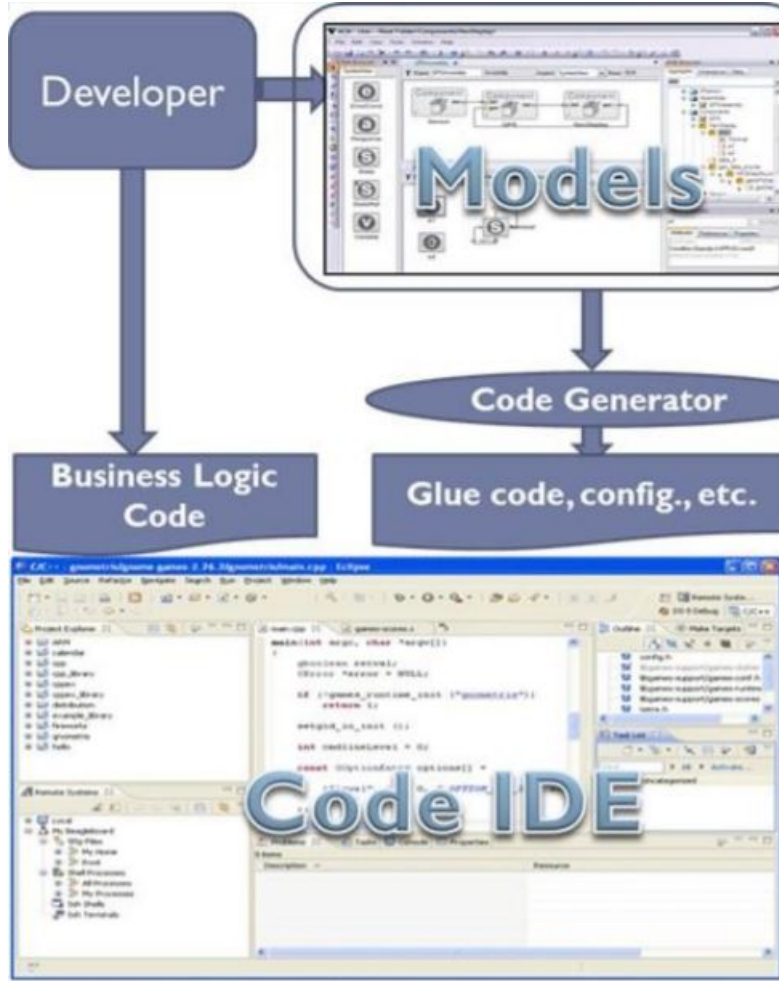


Figure 3.24. Model-driven development [33]

### 3.7.3 Debug tool support

Riaps components have to be debugged before being committed to the final application execution. This is done by the "PyDev toolkit" that runs on Eclipse under the name of "PyDev" and is used to debug components running on the actors of the Riaps target nodes while the graphical front end of the debugger is running on the development machine. The debugger uses the python source code of the component located on the development host [21]. The python debugger tools have to be configured in three main steps analyzed below:

- *Code preparation for Debugging* where "PyDev" library must be imported in the code and a tracepoint must be included to include the IP host node with the respective host port.

- *Riaps debugging configuration* done at file level precisely in a configuration file ".conf" which is responsible for acquiring also debug info modes.
- *Debugging process* which is exploiting the real debugging phase and it is done using breakpoints on the Eclipse environment, launching a deployment process by selecting the "Debug" perspective. Afterwards the application need to be launched from the development host to the target nodes and the respectively actor or device is automatically linked to the debugger.

### 3.8 Application deployment and control tool

This section analyze in detail the operation services offered by Riaps platform able to provide facilities on the implementation algorithm. In the field of application deployment it is analyzed in detail one of the most important services of the platform: the "*Deployment service*". Aside from this services marked as executable such as "riaps\_actor", "riaps\_device", "riaps\_deplo", "riaps\_disco", "timesyncd & timesyncctl" and "riaps\_ctrl" are used to exploit some performances.

In particular [22]:

- *riaps\_actor* is the executable responsible to load the components using the dynamic loader implemented either in python or C++.  
Each "riaps\_actor" process run a single application actor, so multiple "riaps\_actor" runs on multiple processes.
- *riaps\_device* is the executable responsible for dynamically loading and running the device components. It differs from the "riaps\_actor", because it holds only one device component at a time, moreover it activate the communication within components inside the same host so that they cannot be accessed by the network.
- *riaps\_deplo* is the executable responsible for implementing the *deployment manager service*, *resource management* and the *fault management services*.  
The main role of this executable is to receive instruction from the "riaps\_ctrl" executable and implement the functions such as: "deploy, start, stop, remove" for the applications.  
It maintains also the communications between the actors in fact it sends and receives messages and fault notifications.
- *riaps\_disco* is the executable responsible to host the discovery service, which is as previously described the fundamental service of the platform. The service is automatically started because of the "riaps\_deplo" process and exploit the very same functionalities of the Discovery service connecting the actors to the service and enabling the communications.

- *timesyncd* & *timesyncctl* where "timesyncd" is the service that tunes the configurations of the components for the time synchronization service based on the GPS reference clock and the control host reference time.  
The "timesyncctl" is the tool command responsible to configure the time-synchronizaion services according to predefined roles it is used.  
This service is not much related to the other so that it could be considered independent because it can provides functionalities based on external factors.
- *riaps\_ctrl* is the executable which runs on the control node responsible to download the code.  
This process is started manually with respect to the others and it is the first process that start a deployment of an applications as soon as it is the one that triggers the activation of all the others.

Each of this process analyzed above, it is not intended to work alone, they interact and communicate each others as well as the developer run the "riaps\_ctrl" on the host and the target nodes runs the "riaps\_deplo" that will launch the "riaps\_disco", the "riaps\_actor" and the "riaps\_device". An exception is the "time-synchronization" service as explained in its appropriate section.

An example showing the application and all the services linked each others could be the one depicted on Fig. 3.25.

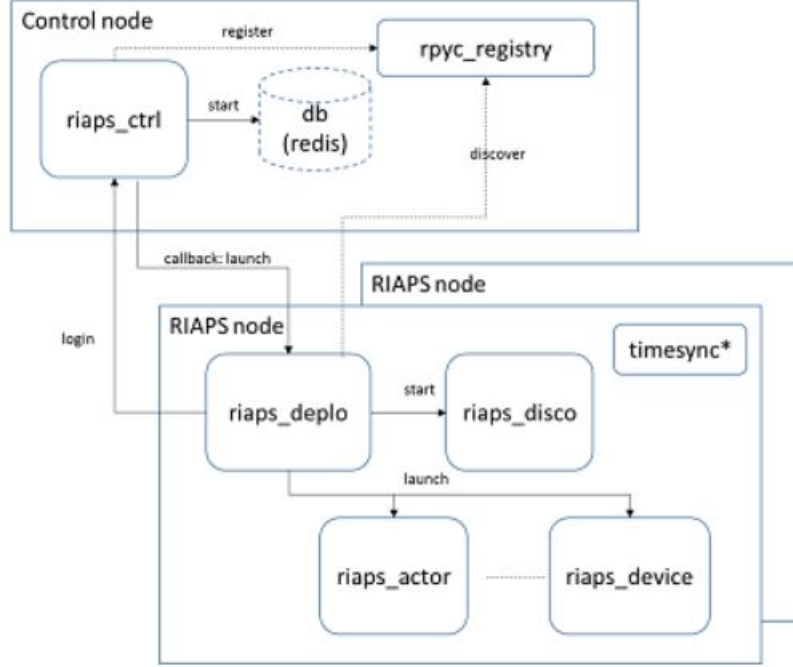


Figure 3.25. Executable block scheme [22]

### 3.9 Implementation

For the implementation of the Riaps platform is designated to be run on Linux machine and on the host development machine it is provided an already pre-configured disk with extension ".vmdk" with the main features including the software IDE and the platform itself created with "Oracle Virtualbox", even if this not excludes the use of other Virtual machines emulators.

On the other hand, for the development host nodes it is also provided a pre-configured lightweight SD-card image containing only the basic features and the tree folder architecture needed to implement the platform. In particular while disk ".vmdk" is provided for an architecture based on AMD class of microprocessors, the image to be stored in to the SD-card is configured to run in a ARM-cortex microprocessor for embedded system. In the implementation about the ARM-cortex micro-processor it is used the "BBB" (Beaglebone Black) hardware, retained a valid hardware able to support all the requirement of the platform itself.

Beside this, since the development host and the target nodes must be on the same sub-net [22], the hardware configuration can be represented in two different structures.

- The first one represented in Fig. 3.26 that depict the situation in which the

development machine has two different "NICs" (Network interface controller) where one "NIC (WLAN)" is used for the internet connectivity and the other one "NIC (B)" is used for connecting with the local target nodes. In this case that the host has multiple physical "NICs" the "riaps\_deplo" executable has to know which one is used to find the local nodes, that is specified in the /usr/local/riaps/etc/riaps.conf file [22].

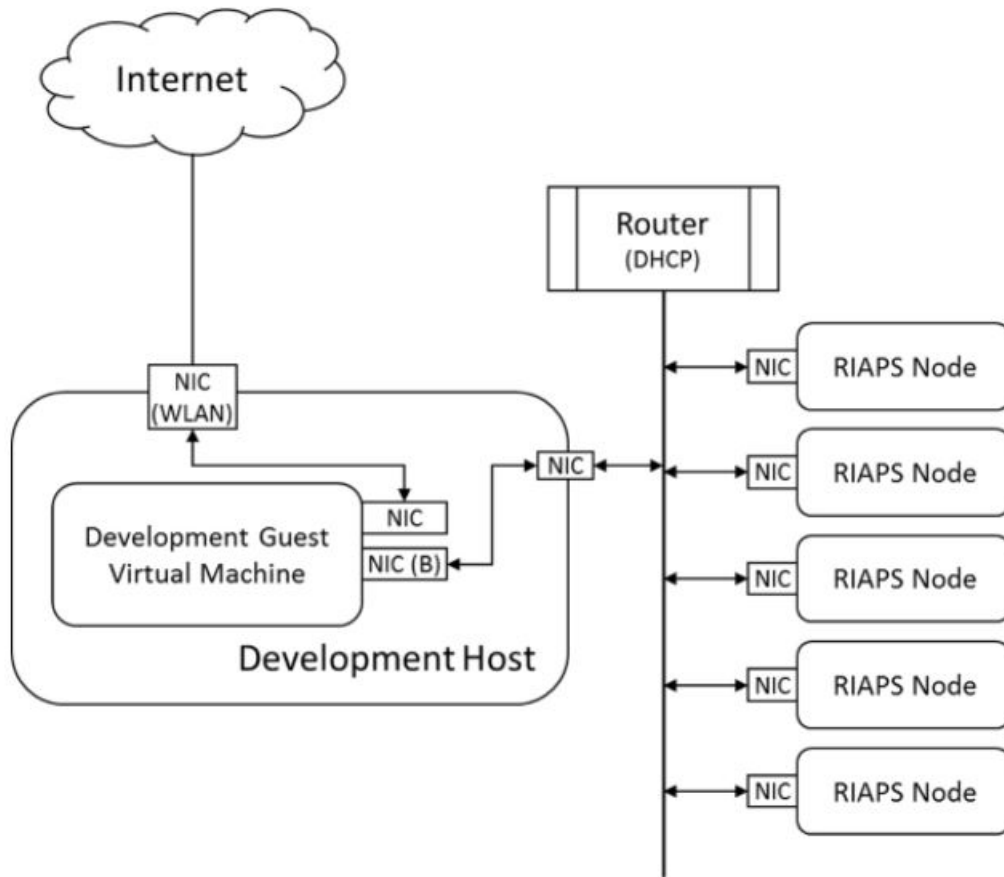


Figure 3.26. WLAN implementation [22]

- The second one represented in Fig. 3.27 provides that all nodes are connected to a router that runs the "DHCP" (Dynamic host configuration protocol) service where all nodes can have access to the internet network if needed.

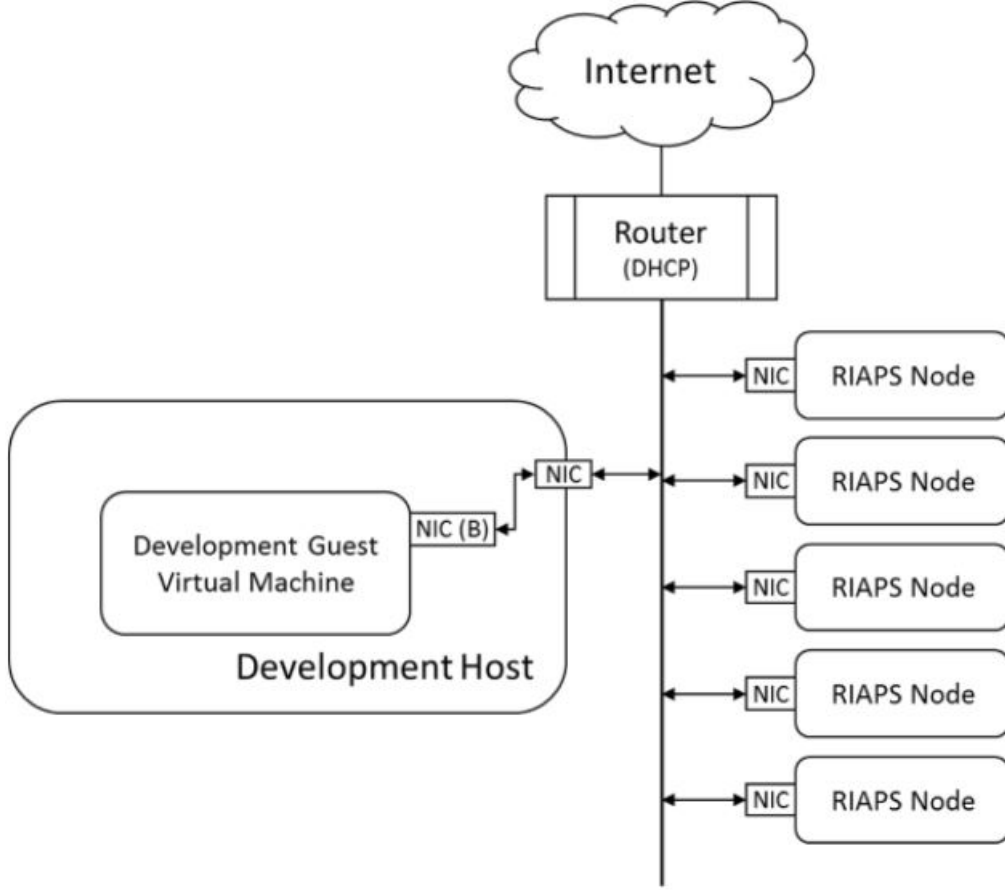


Figure 3.27. LAN implementation [22]

### 3.10 Conclusion of the description of the platform

The component frameworks are increasing drastically and there is a daily request to collapse the cyber and physical features of nowadays infrastructures, especially when this last one is distributed along a wide geographical area. The aim of this platform is to serve as a sort of a smart operating system distributing the "intelligence" to different attached and controllable location [14]. In this way the progress goes further and beyond the simple concept of a "SCADA" (Supervisory control and data acquisition), because it provide a big list of valid reasons that lead to become benefit in a future modern society.





## Chapter 4

# Background implementations of RIAPS

In this chapter as a proof of the platform's potential and its effectiveness in applications a set of examples are shown, that has been case of study in the past.

However the examples in questions are of considerable importance because they have the role of better explaining the correct usage and to understand the field of application where RIAPS platform works or could be involved in the next future.

Nevertheless, these examples take also the position of being excellent tutorials for the self learning and since Riaps is an open source platform, all of the sample project and basic implementations are loaded into the Github repositories.

As a matter of clearance the projects described are shown from the simpler to the complex one.

### Importance of Github for RIAPS

Github is an open source and online platform which is defined as the home to an interconnected community of developers from all over the world [28].

Github provides also a list of benefits which are reflecting the actual world and support on diversity for allowing people with different cultures and level of instructions to push the innovation through the boundaries of the software developing.

In Github the developers are able to upload, share and collaborate at the same projects to develop and finally test the applications collecting opinions by the all the connected users.

Github is easy to use, effective to the purpose of group developing and finally it is easy to access, due to the free accessibility from any device connected to the world wide web.

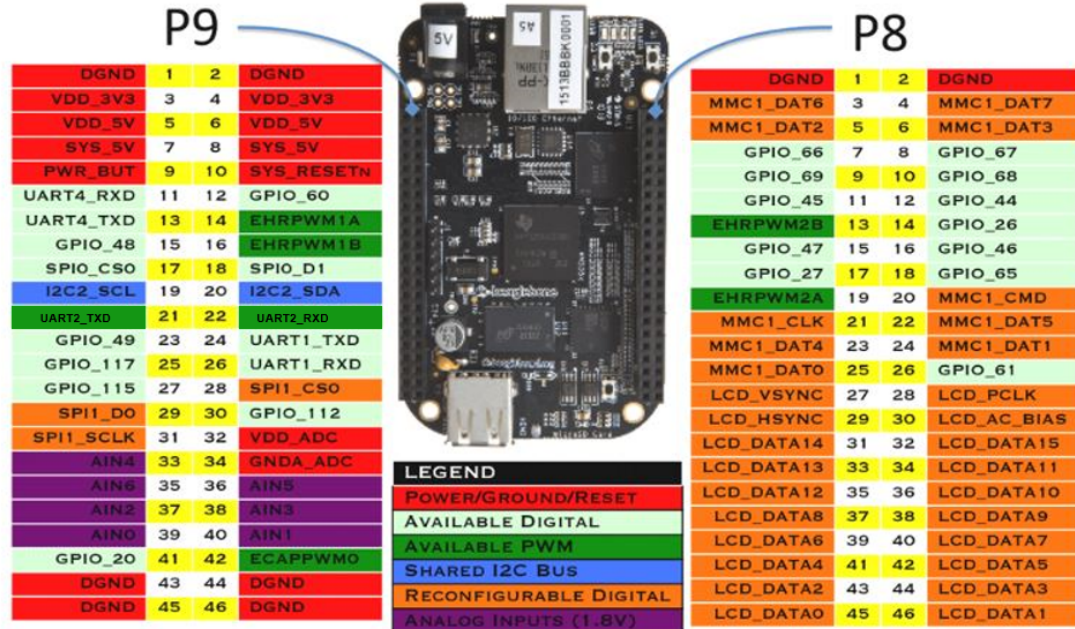
## 4.1 GPIO Device Toggle

As the first example of application [20], it is shown the behaviour of the platform integrating the functionality to toggle the GPIO of the device.

In particular the example propose to create a communication between the component and the device. The component is "Toggle Gpio Component" while the device is "GPIO Device" and they are both written in python and implemented in the same actor deployed in the same target node. The component in this example is the responsible to toggle the GPIO value, while the device is the one responsible to interface with the hardware in order to send its signal. In order to simplify the description, only relevant code snippets are shown.

### 4.1.1 Hardware configuration

The hardware configuration is really intuitive because it provides the installation of a LED on the GPIO pin of the "BBB" (Beaglebone Black). According to the Beaglebone black in equipment, the data-sheet of the hardware configuration suggest the position of the GPIO pins. The data-sheet that shows the pin series represented in Fig. 4.1. The pin used for the GPIO implementation and testing is the "GPIO\_68"



P9				P8			
DGND	1	2	DGND	DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3	MMC1_DAT6	3	4	MMC1_DAT7
VDD_5V	5	6	VDD_5V	MMC1_DAT2	5	6	MMC1_DAT3
SYS_5V	7	8	SYS_5V	GPIO_66	7	8	GPIO_67
PWR_BTN	9	10	SYS_RESETN	GPIO_69	9	10	GPIO_68
UART4_RXD	11	12	GPIO_60	GPIO_45	11	12	GPIO_44
UART4_TXD	13	14	EHRPWM1A	EHRPWM2B	13	14	GPIO_26
GPIO_48	15	16	EHRPWM1B	GPIO_47	15	16	GPIO_46
SPI0_CS0	17	18	SPI0_D1	GPIO_27	17	18	GPIO_65
I2C2_SCL	19	20	I2C2_SDA	EHRPWM2A	19	20	MMC1_CMD
UART2_TXD	21	22	UART2_RXD	MMC1_CLK	21	22	MMC1_DAT5
GPIO_49	23	24	UART1_TXD	MMC1_DAT4	23	24	MMC1_DAT1
GPIO_117	25	26	UART1_RXD	MMC1_DATA0	25	26	GPIO_61
GPIO_115	27	28	SPI1_CS0	LCD_VSYNC	27	28	LCD_PCLK
SPI1_D0	29	30	GPIO_112	LCD_HSYNC	29	30	LCD_AC_BIAS
SPI1_SCLK	31	32	VDD_ADC	LCD_DATA14	31	32	LCD_DATA15
AIN4	33	34	GNDA_ADC	LCD_DATA13	33	34	LCD_DATA11
AIN6	35	36	AIN5	LCD_DATA12	35	36	LCD_DATA10
AIN2	37	38	AIN3	LCD_DATA8	37	38	LCD_DATA9
AIN0	39	40	AIN1	LCD_DATA6	39	40	LCD_DATA7
GPIO_20	41	42	ECAPPWM0	LCD_DATA4	41	42	LCD_DATA5
DGND	43	44	DGND	LCD_DATA2	43	44	LCD_DATA3
DGND	45	46	DGND	LCD_DATA0	45	46	LCD_DATA1

LEGEND	
POWER/GROUND/RESET	
AVAILABLE DIGITAL	
AVAILABLE PWM	
SHARED I2C BUS	
RECONFIGURABLE DIGITAL	
ANALOG INPUTS (1.8V)	

Figure 4.1. Beaglebone Black pin-out data-sheet [2]

corresponding to the P8 series on the right side of the Fig.4.1. The ground has multiple attachments so that it is possible to choose the nearest one, which in this case correspond to the pin 1. The LED need to be directed in the right position to work correctly i.e. the longer wire goes to the GPIO pin and the shorter one to the ground.

### 4.1.2 Software Configuration

First of all it is necessary to install the python library for the GPIO interactions on both Virtual machine and BBB (after logged by "ssh xxx.xxx.xx.xx" command) with the command:

```
$ sudo pip3 install Adafruit_BBIO
```

configuration and verify that the "riaps" user is in the "gpio" group with the command:

```
$ groups riaps
```

### 4.1.3 Application architecture

#### **gpioExample.riaps**

Here it is shown the main application file Fig. 4.2, that describe the connection logic decided by the developer between the component and device through message topics. In particular there is a req/rep relation between the device "*GpioDevice*" and the component "*ToggleGpioComponent*" that on the base of this communication pattern are able to carry on information on both directions.



### ToggleGpioComponent.py

The *"ToggleGpioComponent"* has two timer according to the *".riaps"* file. The first one which name is *"toggle"* is responsible to switch the value from 0 to 1 and viceversa every 5 seconds and to send the value to the *"GpioDevice"* on the *"gpioReqPort"* (1a).

The second timer which name is *"read value"*, wakes up every 7 seconds a request to read that is always sent on *"gpioReqPort"* (1b).

The *"ToggleGpioComponent"* at the end receive the value on *"gpioRepPort"* and print it out (6).

### GpioDevice.py

The *"GpioDevice"* is responsible for acquiring both requests (1a) to write and (1b) to read and reply to the request on *"gpioRepPort"* (2). The request is formatted in *"msgType"* (read or write) and *"msgValue"* (0 or 1) and analyzed, then sent to the inside port *"trigger"* (3). On *"trigger"* inside port the message is sent to a plug port (4) which is the one used to interface with the hardware on the BBB. Then the value either just written or just read, is successively sent to the *"ToggleGpioComponent"* on *"gpioRepPort"* (5).

### gpioExample.depl

The actor *TestGpioToggleActor* that include both the *"GpioDevice"* and the *"ToggleGpioComponent"* is totally deployed on a single target node as shown in Fig. 4.4.

```
1 app GpioToggleExample {  
2   on (192.168.10.90) TestGpioToggleActor;  
3 }
```

Figure 4.4. File *gpioExample.depl* [20]

#### 4.1.4 Simulation and outcomes

The code is successfully running on the target node and it is tested and demonstrated as shown in Fig. 4.5. In order to see the *"log"* files it is used on the BBB the command *"sudo journalctl -u riaps-deplo.service -f"*. The LED is toggled by the *"GpioDevice"* every time there is a write request that comes from the *"ToggleGpioComponent"* every 5 seconds and the status is read every 7 seconds according to the previous outline on the two timers.

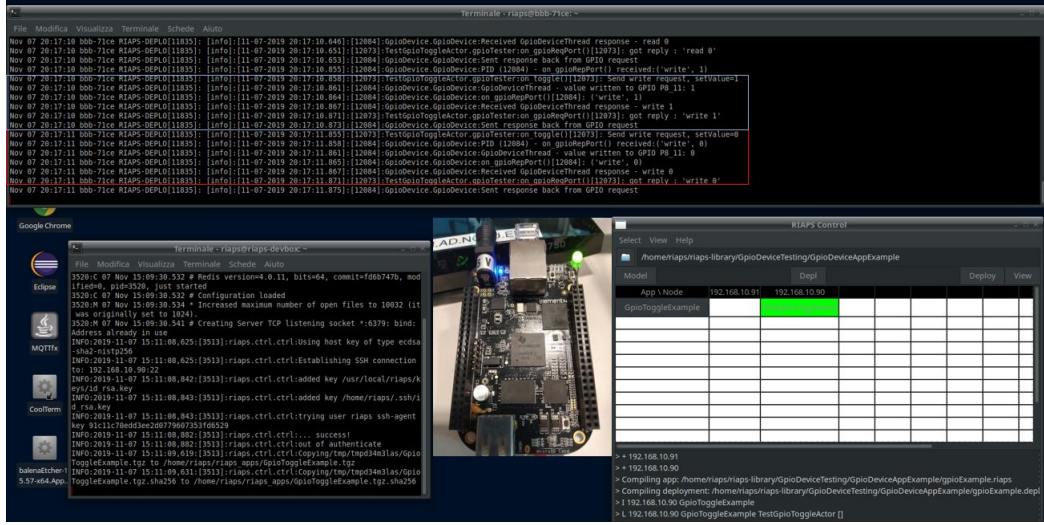


Figure 4.5. GPIO example during execution

## 4.2 Distributed Estimator

The *Distributed Estimator* example is often used as a starting point to the develop complex structure applications. The purpose of the "Distributed Estimator" application is to demonstrate the ability of RIAPS platform to handle two different message patterns: "pub/sub" and "req/rep" [18]. In this example, according with the timer, every second, data are exchanged between the Riaps nodes and the local "sensor". A "Global Estimator" is used to gather data from the "Local Estimator" and running an averaging algorithm.

### 4.2.1 Hardware configuration

There is no specific hardware configuration in this example because it is for demonstratively purpose only. The benefit of this application is that it could run in as many nodes as it is preferred, since it does not imply any complex hardware attachment.

### 4.2.2 Software configuration

For the currently implementation in python, the requirement of software are minimal, and they are all included by the platform itself. Differently for C++ implementation, this could involve the possibility to install libraries through commands on the command line, which are not discussed hereby.



### 4.2.3 Application architecture

#### sample.riaps

The ".riaps" file is describing the interactions between components. In particular there is one "req/rep" and also one "pub/sub" interaction between the "sensor" and the "Local Estimator" components. Moreover it is present one more interaction of type "pub/sub" between the "Local Estimator" and the "Global Estimator" component to send data necessary for the averaging step. The "Global Estimator" also receive the values to be written in the actor level coming from the ".depl" file as a sort of high hierarchy assignation of variables that later are going to overwrite the component variables as shown in Fig. 4.6. This strategy is used in more applications and an example is present in the section "Distributed Estimator GPIO". The

```

1 // RIAPS Sample
2
3 app DistributedEstimator {
4   // Message types used in the app
5   message SensorReady;
6   message SensorQuery;
7   message SensorValue;
8   message Estimate;
9
10  // Sensor component
11  component Sensor {
12    timer clock 1000; // Periodic timer trigger to trigger sensor every 1 sec
13    pub ready : SensorReady ; // Publish port for SensorReady messages
14    rep request : ( SensorQuery , SensorValue ) ; // Reply port to query the sensor and retrieve its value
15  }
16
17  // Local estimator component
18  component LocalEstimator (iArg,fArg,sArg,bArg) {
19    sub ready : SensorReady ; // Subscriber port to trigger component with SensorReady messages
20    req query : (SensorQuery , SensorValue ) ; // Request port to query the sensor and retrieve its value
21    pub estimate : Estimate ; // Publish port to publish estimated value messages
22  }
23
24  // Global estimator
25  component GlobalEstimator (iArg=123,fArg=4.56,sArg="string",bArg=true) {
26    sub estimate : Estimate ; // Subscriber port to receive the local estimates
27    timer wakeup 3000; // Periodic timer to wake up estimator every 3 sec
28  }
29
30  // Estimator actor
31  actor Estimator {
32    local SensorReady, SensorQuery, SensorValue ; // Local message types
33    { // Sensor component
34      sensor : Sensor;
35      // Local estimator, publishes global message 'Estimate'
36      filter : LocalEstimator(iArg=789,fArg=0.12,sArg="text",bArg=false);
37    }
38  }
39
40  actor Aggregator (posArg,optArg="optString") {
41    { // Global estimator, subscribes to 'Estimate' messages
42      agrgr : GlobalEstimator(iArg=posArg,sArg=optArg,bArg=true);
43    }
44  }

```

Figure 4.6. File "sample.riaps" [18]

".dot" file is shown in Fig. 4.7 where the main steps are enumerated for enhancing the comprehension.



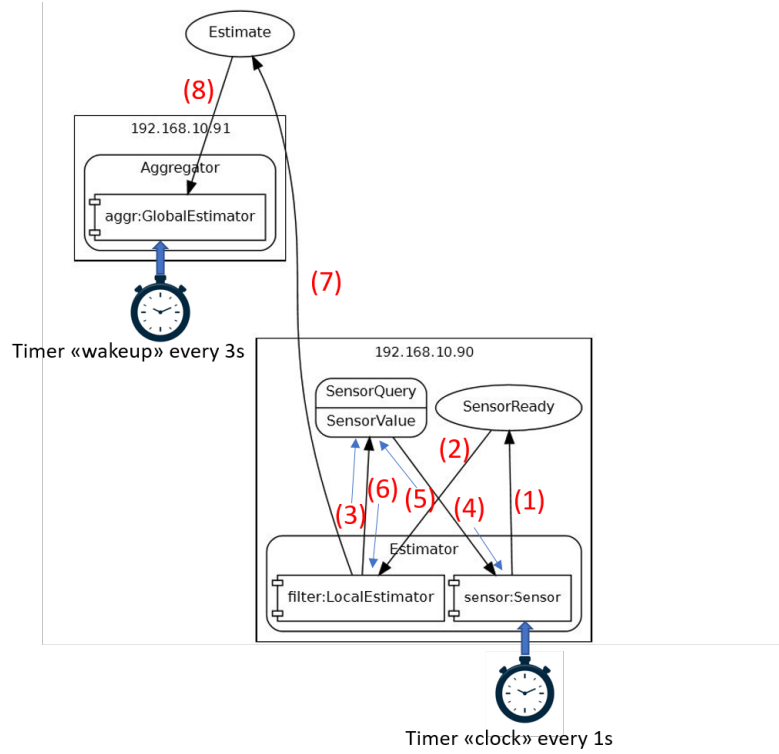


Figure 4.7. Distributed estimator ".dot"

### Sensor.py

The "sensor" component has a timer called "clock" which wakes up every second triggering on the "pub" port sending the message "data\_ready" (1). Then it waits for the operation on the "Local Estimator" receiving the message on "request" port (4) and sends back a response "sensor\_reply" again on "query" port (5).

### LocalEstimator.py

The "Local Estimator" receives the message "data\_ready" from the "sensor" (2) and subsequently triggers the "query" port to send the message "sensor\_query" as a request (3), increasing a pending request variable. Afterwards, it receives the message on "query port" decreasing the pending request variable (6) and, as a final step, the "Local Estimator" sends a message to the "pub" port "estimate" (7).

### GlobalEstimator.py

The "Global Estimator" receives the message from the "Local Estimator" on "estimate" port and prints it out as a very last step (8).

### sample.depl

In this section it is possible to analyze the file where the actors are deployed. As reference to Fig. 4.8 it is possible to see that the actors are deployed on two different nodes that are the ones connected and found by the "riaps\_ctrl" execution command through the discovery service. Some values inside the "Aggregator" actor, correspondent to the "*Global Estimator*", are defined. These values, overwrite the already existent ones in the correspondent actor and component of the ".riaps" file. The values shown in Fig. 4.8 and passed through the variables "posArg" and

```
1 // Eclipse-based execution (no deployment manager, just disco and direct start of actors)|
2 app DistributedEstimator {
3   on (192.168.10.90) Estimator;
4   on (192.168.10.91) Aggregator(posArg=12,optArg="fromDeployment");
5 }
```

Figure 4.8. Distributed Estimator ".depl" file [18]

"optArg" are not used in this application, but they are really useful as described before in the *Distributed Estimator Gpio* example.

#### 4.2.4 Simulation and outcomes

The simulation, shows that the application is able to manage correctly the data coming from the three previously mentioned components using the two different message patterns. In order to better visualize the updated lines, there are highlighted in a red box in Fig. 4.9.

```

Terminale - riaps@bbb-71ce: ~
File Modifica Visualizza Terminale Schede Aiuto
Nov 07 20:35:55 bbb-71ce RIAPS-DEPLO[11835]: [info]:20:35:55,704:[12816]:SENSOR: on_request():sensor_query
Nov 07 20:35:55 bbb-71ce RIAPS-DEPLO[11835]: [info]:20:35:55,709:[12816]:FILTER: on_query():sensor_rep
Nov 07 20:35:56 bbb-71ce RIAPS-DEPLO[11835]: [info]:20:35:56,693:[12816]:SENSOR: on_clock(): 1573158956.6924822
Nov 07 20:35:56 bbb-71ce RIAPS-DEPLO[11835]: [info]:20:35:56,698:[12816]:FILTER: on_ready():data_ready [12816]
Nov 07 20:35:56 bbb-71ce RIAPS-DEPLO[11835]: [info]:20:35:56,703:[12816]:SENSOR: on_request():sensor_query
Nov 07 20:35:56 bbb-71ce RIAPS-DEPLO[11835]: [info]:20:35:56,712:[12816]:FILTER: on_query():sensor_rep
Nov 07 20:35:57 bbb-71ce RIAPS-DEPLO[11835]: [info]:20:35:57,694:[12816]:SENSOR: on_clock(): 1573158957.6931949
Nov 07 20:35:57 bbb-71ce RIAPS-DEPLO[11835]: [info]:20:35:57,700:[12816]:FILTER: on_ready():data_ready [12816]
Nov 07 20:35:57 bbb-71ce RIAPS-DEPLO[11835]: [info]:20:35:57,704:[12816]:SENSOR: on_request():sensor_query
Nov 07 20:35:57 bbb-71ce RIAPS-DEPLO[11835]: [info]:20:35:57,708:[12816]:FILTER: on_query():sensor_rep
Nov 07 20:35:58 bbb-71ce RIAPS-DEPLO[11835]: [info]:20:35:58,695:[12816]:SENSOR: on_clock(): 1573158958.693996
Nov 07 20:35:58 bbb-71ce RIAPS-DEPLO[11835]: [info]:20:35:58,701:[12816]:FILTER: on_ready():data_ready [12816]
Nov 07 20:35:58 bbb-71ce RIAPS-DEPLO[11835]: [info]:20:35:58,705:[12816]:SENSOR: on_request():sensor_query
Nov 07 20:35:58 bbb-71ce RIAPS-DEPLO[11835]: [info]:20:35:58,709:[12816]:FILTER: on_query():sensor_rep
Nov 07 20:35:59 bbb-71ce RIAPS-DEPLO[11835]: [info]:20:35:59,695:[12816]:SENSOR: on_clock(): 1573158959.694797
Nov 07 20:35:59 bbb-71ce RIAPS-DEPLO[11835]: [info]:20:35:59,702:[12816]:FILTER: on_ready():data_ready [12816]
Nov 07 20:35:59 bbb-71ce RIAPS-DEPLO[11835]: [info]:20:35:59,708:[12816]:SENSOR: on_request():sensor_query
Nov 07 20:35:59 bbb-71ce RIAPS-DEPLO[11835]: [info]:20:35:59,712:[12816]:FILTER: on_query():sensor_rep
Nov 07 20:36:00 bbb-71ce RIAPS-DEPLO[11835]: [info]:20:36:00,696:[12816]:SENSOR: on_clock(): 1573158960.695478
Nov 07 20:36:00 bbb-71ce RIAPS-DEPLO[11835]: [info]:20:36:00,702:[12816]:FILTER: on_ready():data_ready [12816]
Nov 07 20:36:00 bbb-71ce RIAPS-DEPLO[11835]: [info]:20:36:00,706:[12816]:SENSOR: on_request():sensor_query
Nov 07 20:36:00 bbb-71ce RIAPS-DEPLO[11835]: [info]:20:36:00,710:[12816]:FILTER: on_query():sensor_rep
Nov 07 20:36:01 bbb-71ce RIAPS-DEPLO[11835]: [info]:20:36:01,697:[12816]:SENSOR: on_clock(): 1573158961.6961813
Nov 07 20:36:01 bbb-71ce RIAPS-DEPLO[11835]: [info]:20:36:01,704:[12816]:FILTER: on_ready():data_ready [12816]
Nov 07 20:36:01 bbb-71ce RIAPS-DEPLO[11835]: [info]:20:36:01,708:[12816]:SENSOR: on_request():sensor_query
Nov 07 20:36:01 bbb-71ce RIAPS-DEPLO[11835]: [info]:20:36:01,712:[12816]:FILTER: on_query():sensor_rep

Terminale - riaps@bbb-8cb1: ~
File Modifica Visualizza Terminale Schede Aiuto
Nov 07 20:35:45 bbb-8cb1 RIAPS-DEPLO[22351]: [info]:20:35:45,385:[23150]:AGGR: on_wakeup():1573158945.384727
Nov 07 20:35:45 bbb-8cb1 RIAPS-DEPLO[22351]: [info]:20:35:45,703:[23150]:AGGR: on_estimate():local_est(12816)
Nov 07 20:35:46 bbb-8cb1 RIAPS-DEPLO[22351]: [info]:20:35:46,704:[23150]:AGGR: on_estimate():local_est(12816)
Nov 07 20:35:47 bbb-8cb1 RIAPS-DEPLO[22351]: [info]:20:35:47,706:[23150]:AGGR: on_estimate():local_est(12816)
Nov 07 20:35:48 bbb-8cb1 RIAPS-DEPLO[22351]: [info]:20:35:48,386:[23150]:AGGR: on_wakeup():1573158948.385466
Nov 07 20:35:48 bbb-8cb1 RIAPS-DEPLO[22351]: [info]:20:35:48,706:[23150]:AGGR: on_estimate():local_est(12816)
Nov 07 20:35:49 bbb-8cb1 RIAPS-DEPLO[22351]: [info]:20:35:49,709:[23150]:AGGR: on_estimate():local_est(12816)
Nov 07 20:35:50 bbb-8cb1 RIAPS-DEPLO[22351]: [info]:20:35:50,706:[23150]:AGGR: on_estimate():local_est(12816)
Nov 07 20:35:51 bbb-8cb1 RIAPS-DEPLO[22351]: [info]:20:35:51,387:[23150]:AGGR: on_wakeup():1573158951.3861995
Nov 07 20:35:51 bbb-8cb1 RIAPS-DEPLO[22351]: [info]:20:35:51,708:[23150]:AGGR: on_estimate():local_est(12816)
Nov 07 20:35:52 bbb-8cb1 RIAPS-DEPLO[22351]: [info]:20:35:52,708:[23150]:AGGR: on_estimate():local_est(12816)
Nov 07 20:35:53 bbb-8cb1 RIAPS-DEPLO[22351]: [info]:20:35:53,709:[23150]:AGGR: on_estimate():local_est(12816)
Nov 07 20:35:54 bbb-8cb1 RIAPS-DEPLO[22351]: [info]:20:35:54,388:[23150]:AGGR: on_wakeup():1573158954.3870058
Nov 07 20:35:54 bbb-8cb1 RIAPS-DEPLO[22351]: [info]:20:35:54,714:[23150]:AGGR: on_estimate():local_est(12816)
Nov 07 20:35:55 bbb-8cb1 RIAPS-DEPLO[22351]: [info]:20:35:55,717:[23150]:AGGR: on_estimate():local_est(12816)
Nov 07 20:35:56 bbb-8cb1 RIAPS-DEPLO[22351]: [info]:20:35:56,717:[23150]:AGGR: on_estimate():local_est(12816)
Nov 07 20:35:57 bbb-8cb1 RIAPS-DEPLO[22351]: [info]:20:35:57,388:[23150]:AGGR: on_wakeup():1573158957.3880017
Nov 07 20:35:57 bbb-8cb1 RIAPS-DEPLO[22351]: [info]:20:35:57,712:[23150]:AGGR: on_estimate():local_est(12816)
Nov 07 20:35:58 bbb-8cb1 RIAPS-DEPLO[22351]: [info]:20:35:58,713:[23150]:AGGR: on_estimate():local_est(12816)
Nov 07 20:35:59 bbb-8cb1 RIAPS-DEPLO[22351]: [info]:20:35:59,717:[23150]:AGGR: on_estimate():local_est(12816)
Nov 07 20:36:00 bbb-8cb1 RIAPS-DEPLO[22351]: [info]:20:36:00,389:[23150]:AGGR: on_wakeup():1573158960.3887537
Nov 07 20:36:00 bbb-8cb1 RIAPS-DEPLO[22351]: [info]:20:36:00,714:[23150]:AGGR: on_estimate():local_est(12816)
Nov 07 20:36:01 bbb-8cb1 RIAPS-DEPLO[22351]: [info]:20:36:01,716:[23150]:AGGR: on_estimate():local_est(12816)

```

Figure 4.9. Distributed Estimator Simulation

### 4.3 Distributed Estimator using GPIO

The *"Distributed Estimator Gpio"* is the practical implementation of the general *"Distributed Estimator"* previously described. The importance of this example, is to demonstrate that from an high level structure, certain parameters are passed, in this way the blinking LED are executed at different frequency rate. The outline is on the base of the previous implementation, the Riaps nodes gather different local sensor data at different rates (0.5 Hz, 1Hz,2Hz) toggling an on-board LED when the estimate is published. Moreover one Riaps node provides a running average of the estimates at 4 Hz rate (0.25s) toggling also another on-board LED.[19]

#### 4.3.1 Hardware configuration

There is no specific Hardware configuration outside the simple connection between BBB and the local router as well as the toggling is happening on the on-board LED. The hardware configuration therefore reflects the one of the basic example

*"Distributed Estimator"* in which three nodes are used as local estimator and one is the global estimator of the system.

### 4.3.2 Software configuration

For the software configuration, it is used the Adafruit BBIO library installed through the command line:

```
$ sudo pip3 install 'Adafruit_BBIO==1.1.1'
```

### 4.3.3 Application architecture

#### DistributedEstimatorGpio.riaps

This section's duty is to describe the interaction between components and devices. As described in the *"Distributed Estimator"* example, the interactions pattern are of type "pub/sub" and "req/rep" between the *"Sensor component"* and the *"Local estimator component"* then a "pub/sub" relation between the *"Local estimator component"* and the *"Global estimator component"* is established. One more interaction is provided in this example, with respect to the basic implementation, which is the one between the *"Local estimator component"* and the *"GPIODevice"* in a pub/sub relation that is responsible for sending the message of toggling the LED. There are two actors, so that the run-time application could be deployed either in one or multiple nodes.

The whole interactions are shown in Fig. 4.10.

```

1 app DistributedEstimatorGpio {
2   // Message types used in the app
3   message SensorReady;
4   message SensorQuery;
5   message SensorValue;
6   message Estimate;
7   message Blink;
8
9   // GPIODevice
10  device GPIODevice() {
11    sub blink : Blink;
12  }
13
14  // Sensor component
15  component Sensor (value=1.0) {
16    timer clock 500; // Periodic timer trigger to trigger sensor every 2 Hz (0.5 sec)
17    pub ready : SensorReady; // Publish port for SensorReady messages
18    rep request : (SensorQuery, SensorValue); // Reply port to query the sensor and retrieve its value
19  }
20
21  // Local estimator component
22  component LocalEstimator (freqArg=2) { // fArg = frequency when estimate produced
23    sub ready : SensorReady; // Subscriber port to trigger component with SensorReady messages
24    req query : (SensorQuery, SensorValue); // Request port to query the sensor and retrieve its value
25    pub estimate : Estimate; // Publish port to publish estimated value messages
26    pub blink : Blink;
27  }
28
29  // Global estimator
30  component GlobalEstimator () {
31    sub estimate : Estimate; // Subscriber port to receive the local estimates
32    timer wakeup 250; // Periodic timer to wake up estimator every 4 Hz (.25 sec)
33  }
34
35  // Estimator actor
36  actor Estimator (freqArg,value=0.0) {
37    local SensorReady, SensorQuery, SensorValue, Blink; // Local message types
38    { // Sensor component
39      sensor : Sensor(value=value);
40      // Local estimator, publishes global message 'Estimate'
41      filter : LocalEstimator(freqArg=freqArg);
42      // GPIO component to blink
43      gpio : GPIODevice();
44    }
45  }
46
47  // Global Estimator actor
48  actor Aggregator () {
49    { // Global estimator, subscribes to 'Estimate' messages
50      aggr : GlobalEstimator();
51    }
52  }
53 }

```

Figure 4.10. File Distributed Estimator using GPIO ".riaps" [19]

### Sensor.py

The "*Sensor component*" is responsible for sending a message to the "*Local Estimator component*" containing "data\_ready" to the "ready" "pub/sub" port (1). Then it wait for the operation in the "*Local Estimator component*" receiving the message on "request" port (4). Afterwards always on the same port, it send the value number (5) that corresponds to the number passed on the ".depl" file.

### LocalEstimator.py

The "*Local Estimator component*" receive the "data\_ready" coming from the "*Sensor component*" (2) and send a request "sensor\_query" on "query" port (3) increasing a counting request variable. Then it receive the "value" on "query" port (6), decrease

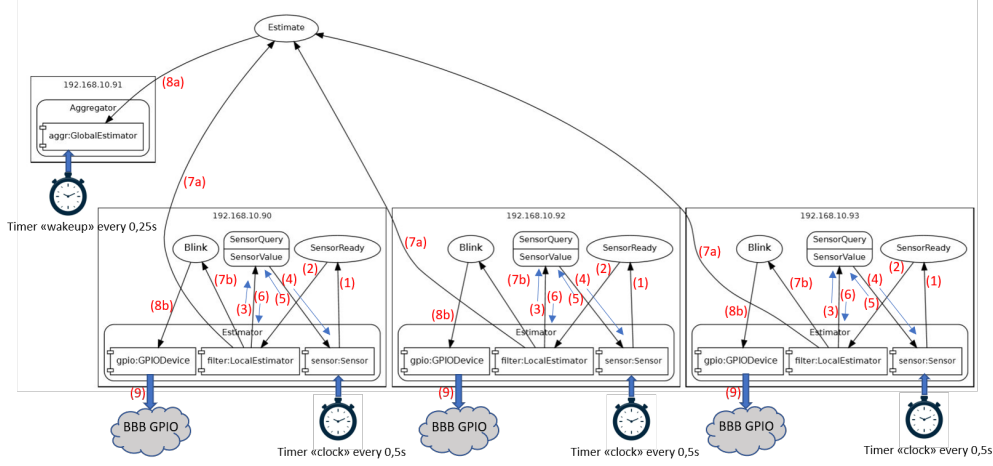


Figure 4.11. File ".dot" Distributed Estimator using Gpio

the counting request variable and classify the message. The *"Local Estimator component"* is also responsible to receive the frequency rate from an high hierarchy level assignation and use it for printing only operation. Afterwards it send back the value on estimate port to the *"Global Estimator component"* (7a) and also a "BLINK" text message to the pub/sub port "blink" to the *"GPIO device component"* (7b).

### GlobalEstimator.py

The *"Global Estimator component"* duty is to receive the message coming from the *"Local Estimator component"* on the "estimate" port (8a) and run its own averaging algorithm.

### GPIODevice.py

The task of the *"GPIO Device component"* is really simple, because it receive the "BLINK" message from the *Local Estimator component* on "blink" port and activate the procedure to establish a connection to the hardware to toggle the on-board LED.

### DistributedEstimatorGpio.depl

Hereby are shown the details of the nodes where each actor is deployed in Fig. 4.12. The "Estimator" actor is deployed in three different target nodes with different parameters "freqArg" and "value" defining the rate frequency and the number of the node taken into consideration. These parameters are passed through the *"Estimator"* actor in the "DistributedEstimatorGpio.riaps" respectively to the *"Local Estimator component"* and to the *"Sensor component"*, that are overwriting the



standard ones. In this way it is created a differentiation with the same structure of application for different nodes, where this last one is deployed. Nevertheless, the developers are able to reuse components that provide different configurability [26], so that, the functionalities they expect during the run-time meet the requirements of the plan studied.

```

1 // Application Deployment Configuration
2 app DistributedEstimatorGpio {
3   on (192.168.10.91) Estimator(freqArg=0.5,value=1.0); // 0.5 Hz update rate
4   on (192.168.10.92) Estimator(freqArg=1.0,value=2.0); // 1 Hz update rate
5   on (192.168.10.90) Estimator(freqArg=2.0,value=3.0); // 2 Hz update rate
6   on (192.168.10.93) Aggregator();
7 }

```

Figure 4.12. File Distributed Estimator using GPIO ".depl" [19]

### 4.3.4 Simulation and outcomes

As it is shown in Fig.4.13, the simulation is running as expected. For sake of simplicity the algorithm shown is running only in three target nodes demonstrating the values are correctly exchanged as well as the estimation is successfully performed.

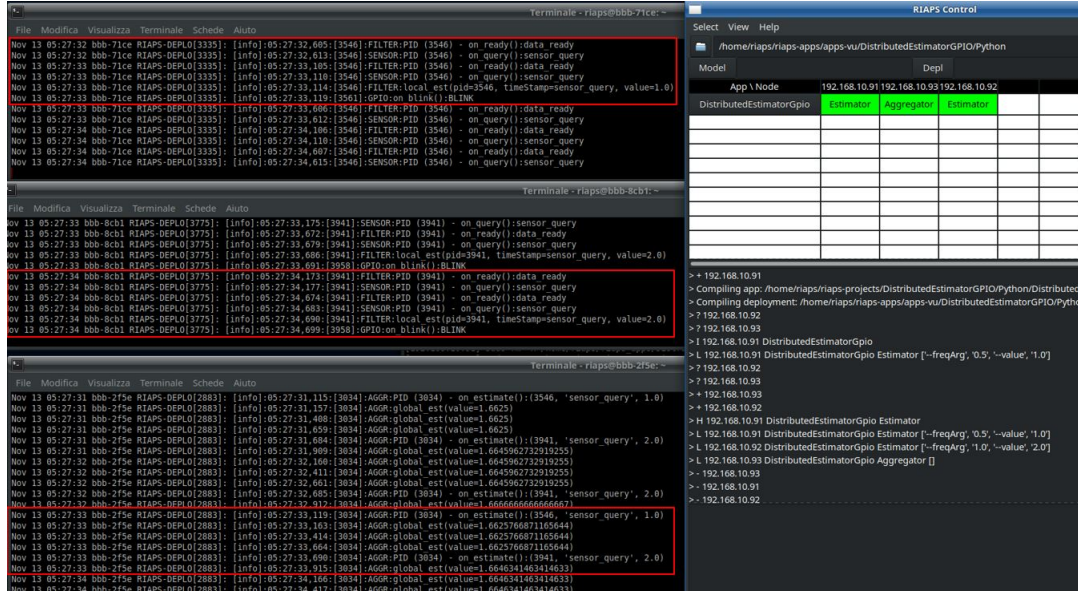


Figure 4.13. Distributed Estimator Using GPIO simulation

## 4.4 UART A to UART B communication

This example is used to demonstrate the effectiveness of the data transmission between "UART" ports of two different BBBs. In particular there are two components, "*TestUartComponentA*" and "*TestUartComponentB*", respectively inside two actors "*TestUartActorA*" and "*TestUartActorB*". The Actor "A" is responsible for writing in the UART port an incremental count, that the Actor "B" is reading and posting to screen to check the consistence [27].

### 4.4.1 Hardware configuration

In order to run this example is necessary to configure the physical connections between the chosen "UART" port of one BBB to the chosen "UART" port of the second BBB. In order to accomplish to this requirement, it is used the "UART2" port located between "Pin21" and "Pin22" of the P9 series expansion referring to the pin-out table in Fig.4.1 The connection need to be carried out in this way:

- BBB1 UART2 TX (P9,pin 21) ==> BBB2 UART2 RX (P9,pin 22)
- BBB1 UART2 RX (P9,pin 22) ==> BBB2 UART2 TX (P9,pin 21)
- BBB1 Ground (P9,pin 1) ==> BBB2 (P9,pin 1)

The wiring connection is also represented in Fig. 4.14

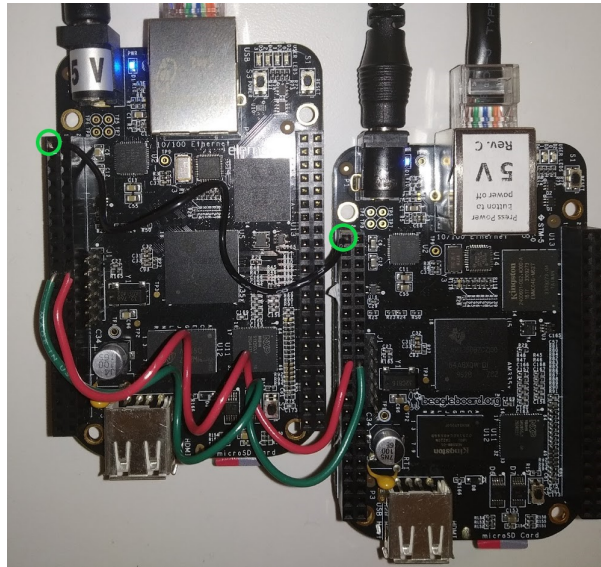


Figure 4.14. UART to UART connection



### 4.4.2 Software configuration

In order to enable the UART2 on the BBB, it is necessary to open and modify the "/boot/uEnv.txt" file with the command:

```
$ sudo nano /boot/uEnv.txt
```

adding the following lines:

```
###Master Enable
enable_uboot_overlays=1
...
###Additional custom capes
uboot_overlay_addr4=/lib/firmware/BB-UART2-00A
```

then reboot as follow:

```
$ sudo nano /boot/uEnv.txt
```

To verify the UART port is really activated check with the command :

```
$ ls -l/dev/ttyO*
```

It should appear that:

```
lrwxrwxrwx 1 root root (date) 6 (time) /dev/ttyO0 -> ttyS0
lrwxrwxrwx 1 root root (date) 6 (time) /dev/ttyO2 -> ttyS2
```

Then install the GPIO python Library on both VM and BBB with:

```
$ sudo pip3 install Adafruit_BBIO
```

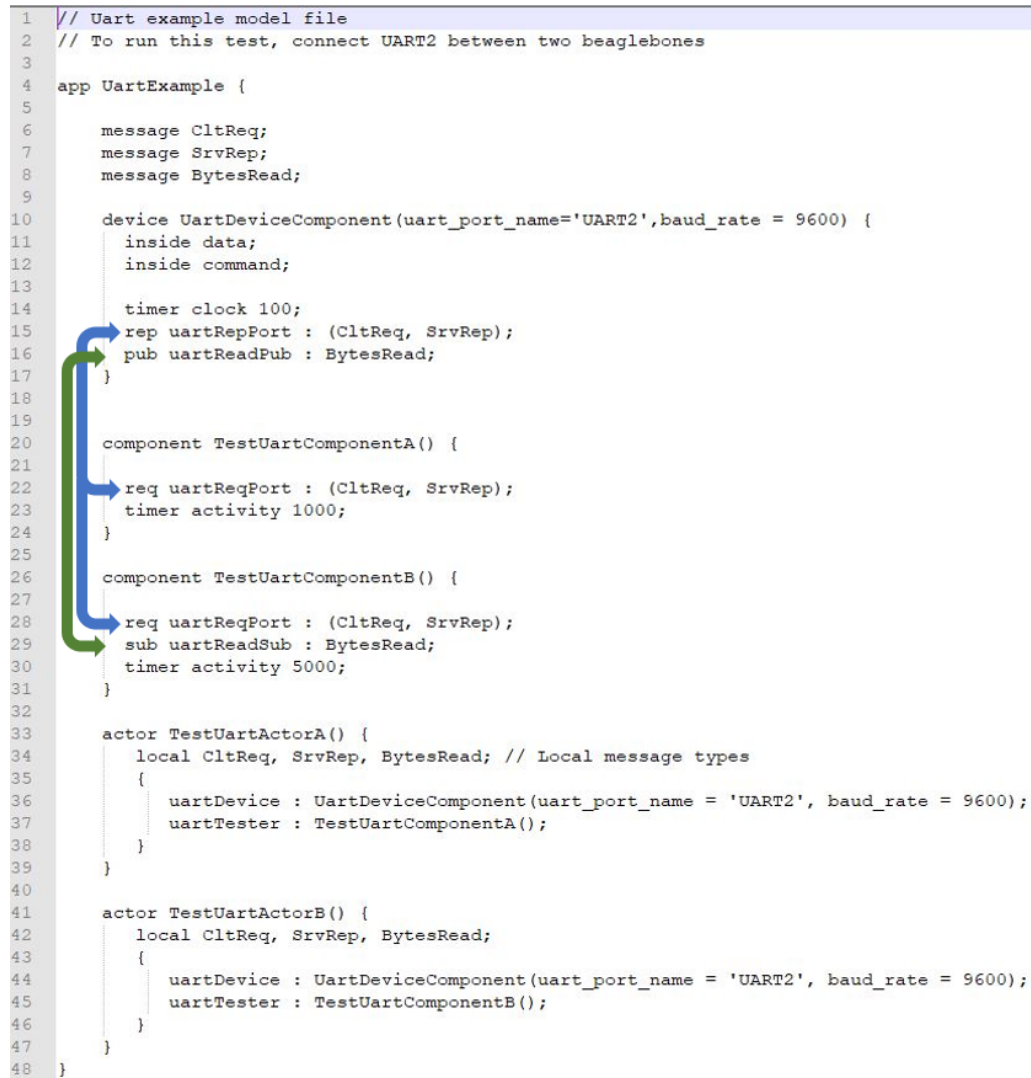
and verify that the user account in the BBB is in "dialout group" with:

```
$ groups riaps
```

### 4.4.3 Application architecture

#### uartExample.riaps

The ".riaps" file is always intended for giving the connection logic between the components and the devices. In this case, the requests coming from both "*TestUartComponentA*" and "*TestUartComponentB*" to "*UARTDeviceComponent*" on the UART port could be either to "write" or to "read" as represented in Fig. 4.15. A timer on both components is responsible for sending and receiving commands and data to the device which is interfere with the physical port.



```

1 // Uart example model file
2 // To run this test, connect UART2 between two beaglebones
3
4 app UartExample {
5
6     message CltReq;
7     message SrvRep;
8     message BytesRead;
9
10    device UartDeviceComponent(uart_port_name='UART2',baud_rate = 9600) {
11        inside data;
12        inside command;
13
14        timer clock 100;
15        rep uartReqPort : (CltReq, SrvRep);
16        pub uartReadPub : BytesRead;
17    }
18
19
20    component TestUartComponentA() {
21        req uartReqPort : (CltReq, SrvRep);
22        timer activity 1000;
23    }
24
25    component TestUartComponentB() {
26
27
28        req uartReqPort : (CltReq, SrvRep);
29        sub uartReadSub : BytesRead;
30        timer activity 5000;
31    }
32
33    actor TestUartActorA() {
34        local CltReq, SrvRep, BytesRead; // Local message types
35        {
36            uartDevice : UartDeviceComponent(uart_port_name = 'UART2', baud_rate = 9600);
37            uartTester : TestUartComponentA();
38        }
39    }
40
41    actor TestUartActorB() {
42        local CltReq, SrvRep, BytesRead;
43        {
44            uartDevice : UartDeviceComponent(uart_port_name = 'UART2', baud_rate = 9600);
45            uartTester : TestUartComponentB();
46        }
47    }
48 }

```

Figure 4.15. File ".riaps" UART Device Testing [27]

The steps are hereby enumerated in order to enhance the comprehension as shown in Fig. 4.16.

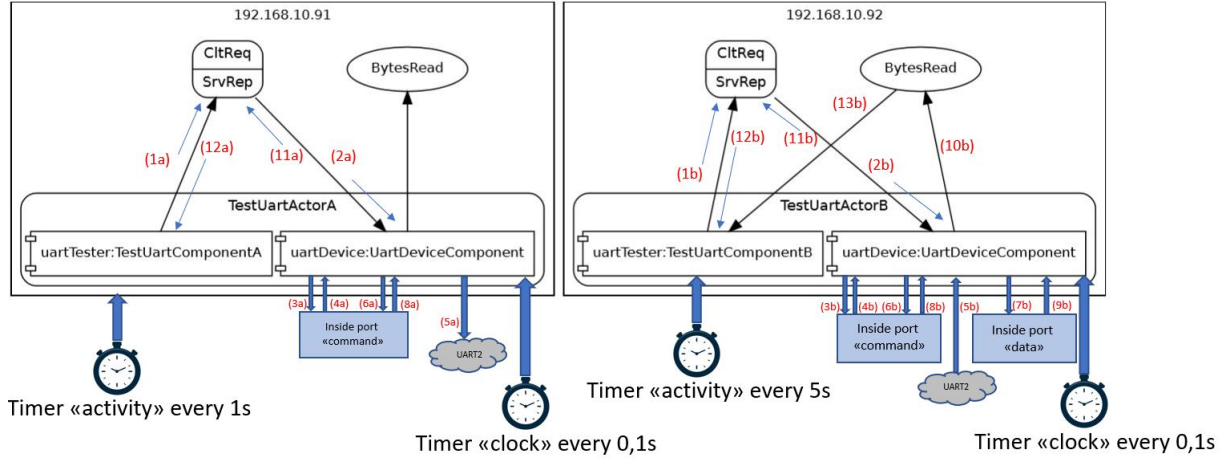


Figure 4.16. File ".dot" UART Device Testing"

### TestUartComponentA.py

The *TestUartComponentA* send the request to "write" and the *msgValue* (an incremental counter encoded in ASCII) in the *uartReqPort* (1a), then it wait for all the operation in the *UartDeviceComponent*, finally it receive always in the *uartReqPort* the value of successfully executed write operation printing out "Got reply" (12a).

### TestUartComponentB.py

The *TestUartComponentB* send the request to "read" in the *uartReqPort* together with the *msgValue* containing the size of reading (1b), then it wait for all the operation in the *UartDeviceComponent* receiving always in the *uartReqPort* the value of successfully executed read operation, printing out "Got reply" (12b). Finally, it subscribes to the *uartReadSub* to get and decode (from ASCII) the value read, previously sent by the *UartDeviceComponent* in the inside port "data" (13b).

### UartDeviceComponent.py

The *UartDeviceComponent* is responsible for acquiring both the two requests coming from *uartRepPort* (2a)-(2b), and classify it (if *Message.read* corresponds to number "2" if *Message.write* corresponds to number "3"). Then it send the

"Message.type" either "2" or "3" and "MsgValue" (encoded msg to be written for "write" or number of bits to be read for "read") to the inside port "command" (3a)-(3b). Afterwards, it receive from "plug" (command inside port) the "msg" (4a)-(4b) and operates in way to run a write function to write UART2 (5a) that return that number of bits written "10" in "returnValue" or to run a read function to read UART2 (5b). After that, it re-send to "plug" (inside command port) the "msg.Type,msgValue" containing either the "write=3" or "read=2" operation type and "returnValue" (number of bits written) for write or "msgValue" (conventionally set to "1") for read operation respectively (6a)-(6b). The read function, in the meanwhile, send the bits read to the "data\_plug" inside port (7b) and receive it successively "on\_data" inside port (9b). Afterwards, "on\_data" port, it send the "msg" of read with the encoded value to the "UartReadpub" port (10b). At the very end the "*UartDeviceComponent*" read the inside "command" port (8a)-(8b) and send the values of the previous operations (6a-6b) as reply in the "uartRepPort" (11a)-(11b) for both requests.

#### uartExample.depl

This file shows in which nodes the two actors "*TestUartActorA*" and "*TestUartActorB*" deployed, and so the communication is tested as shown in Fig. 4.17.

```
1 app UartExample {  
2   on (192.168.10.91) TestUartActorA;  
3   on (192.168.10.92) TestUartActorB;  
4 }
```

Figure 4.17. File ".depl" UART Device Testing [27]

#### 4.4.4 Simulation and outcomes

As shown in Fig. 4.18 the simulation is running and the effectiveness is demonstrated by checking the correct exchanging of values inside the variables and that the reading and the writing operation inside the "UART" ports is done thanks to the "*UartDeviceComponent*". The writing operation is performed according with the timer, every second, writing 10 bits ("Data:000"), while the reading operation is done every 5 seconds with a buffer size of 50 bits in order to recover the previous values, so that at second 5 it display: "Data:000 Data:001 Data:002 Data:003 Data:004".

Obviously the choice of reading length and update time has been thought before matching the exactly bits of the Data word, the space, the symbol and the three digits. The counter variable is updated uninterruptedly but the algorithm is studied

to visualize only a restart from "000" in case the counter exceed the maximum value of three digits (100).

```

Nov 15 18:24:46 bbb-8cb1 RIAPS-DEPLO[671]: [Info]:[11-15-2019 18:24:46.577]:[1250]:UartDeviceComponent.UartDeviceComponent:UartDeviceThread timeout
Nov 15 18:24:47 bbb-8cb1 RIAPS-DEPLO[671]: [Info]:[11-15-2019 18:24:47.577]:[1250]:UartDeviceComponent.UartDeviceComponent:UartDeviceThread: Attempting to read...
Nov 15 18:24:47 bbb-8cb1 RIAPS-DEPLO[671]: [Info]:[11-15-2019 18:24:47.582]:[1250]:UartDeviceComponent.UartDeviceComponent:UartDeviceThread: DONE READING
Nov 15 18:24:47 bbb-8cb1 RIAPS-DEPLO[671]: [Info]:[11-15-2019 18:24:47.583]:[1250]:UartDeviceComponent.UartDeviceComponent:UartDeviceComponent: Publishing Data
Nov 15 18:24:47 bbb-8cb1 RIAPS-DEPLO[671]: [Info]:[11-15-2019 18:24:47.588]:[1238]:TestUartActorB.uartTester:on uartReadSub():[1238]: got bytes : Data: 000 Data: 000 Data: 004
Nov 15 18:24:50 bbb-8cb1 RIAPS-DEPLO[671]: [Info]:[11-15-2019 18:24:50.583]:[1238]:TestUartActorB.uartTester:on activity():[1238]: requested to read: ('read', 50)
Nov 15 18:24:50 bbb-8cb1 RIAPS-DEPLO[671]: [Info]:[11-15-2019 18:24:50.589]:[1250]:UartDeviceComponent.UartDeviceComponent:on uartReqPort():[1250]: ('read', 50)
Nov 15 18:24:50 bbb-8cb1 RIAPS-DEPLO[671]: [Info]:[11-15-2019 18:24:50.596]:[1238]:TestUartActorB.uartTester:on uartReqPort():[1238]: got reply : (<Message.read: 2>, 1)
Nov 15 18:24:51 bbb-8cb1 RIAPS-DEPLO[671]: [Info]:[11-15-2019 18:24:51.593]:[1250]:UartDeviceComponent.UartDeviceComponent:UartDeviceThread: Attempting to read...
Nov 15 18:24:51 bbb-8cb1 RIAPS-DEPLO[671]: [Info]:[11-15-2019 18:24:51.598]:[1250]:UartDeviceComponent.UartDeviceComponent:UartDeviceThread timeout

```

App \ Node	192.168.10.91	192.168.10.92
UartExample	TestUartActorA	TestUartActorB

Figure 4.18. UART A to UART B communication simulation



# Chapter 5

## Methods and Tools

In this chapter, all methods and tools are analyzed in details in order to reach the correctness of the experiment results and to figure out how the work project has been carried out during the entire study time at FREEDM System Center, belonging to North Carolina State University, USA.

Starting with the analysis of the requirements for each step, moving to the description of the protocols and hardware tools used, proceeding to some insights about these steps development, concluding with an examination of the algorithm developed with the technical support from some software tools for the purpose application.

Moreover, the analyzed steps are somewhat referring to the V-shape algorithm in Fig. 5.1, with particular focus on the lower level steps. These includes, the algorithm requirements together with a basic explanation of communication protocols used, the model testing in a Model in the loop simulation "MiL", the software testing performing a Software in the loop simulation "SiL" and finally some tests performed on the real hardware generating a complete system testing in a Hardware in the loop simulation "HiL".

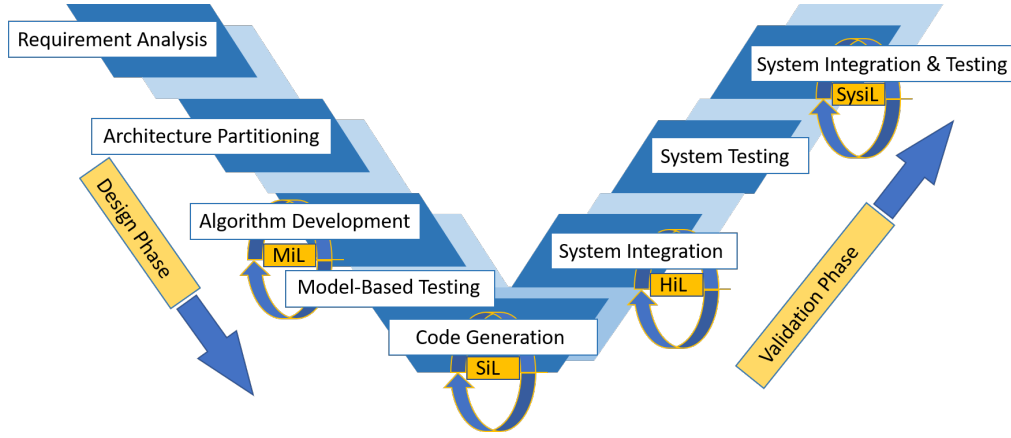


Figure 5.1. V-shape model entire system development

## 5.1 Algorithm requirements

The algorithm requirements depends on the tools and protocols used in order to reach the real and effective final configuration. In this section are analyzed the technical requirements to be accomplished by each simulation step and an advance of the main protocol an tools used.

For the "MiL" simulation, the TCP/IP protocol should be implemented in Simulink. Then a simulation should demonstrate the correctness of the running block connections. Moreover the model should run in interconnection with the OPAL-RT simulator to demonstrate the correct reading operation through modification of different parameters in the console of the model.

For the "SiL" simulation, the Riaps platform should be included in the loop running on a BBB and the model should be perform the read and write operations.

For the "HiL" simulation, the entire system should be tested in the real hardware, in this case an inverter, and this last one should interact with the platform to exchange data and commands.

## 5.2 Protocols and hardware tools

In this section are analyzed the main protocol used and the hardware tools necessary for running the software applications.

The main protocol used is the TCP/IP protocol, fundamental for the development of this project while the developed codes are launched in RIAPS environment based itself on Linux environment, which depicts the first fundamental tool on which all the others features are laid. The operating system itself, is able to host the RIAPS



software platform and due to the fact that is open source, it is possible to include libraries necessary for the code developing. The coding of the developed application is based on the use of Python 3.7 and C++ languages supported by RIAPS and by the GNU Compiler Collection 7 (GCC 7). All these feature are running on BBB, simulated with the Opal-RT 5600 simulator and validated in the real SMA STP 20000TL-US-10 inverter hardware.

### 5.2.1 TCP/IP communication protocol

TCP (Transmission control protocol) is a standard that defines the way the application program establishes a connection and exchanges packets of data on a network and Internet Engineering Task Force (IETF) defines TCP in the Request for Comment (RFC) standards document number 793. TCP is a connection-oriented protocol, which means a connection is established and maintained until the application programs have finished exchanging messages in appropriate sequence. It determines how to break application data into packets, that the networks can deliver, sends and accepts packets through the network layer, manages flow control and, since it is designed in order to provide error-free data transmission, it handles re-transmission of dropped packets as well as acknowledgement of all packets that arrive [42]. In the Open Systems Interconnection (OSI) communication model, TCP covers part of Layer 4, the transport layer, and part of Layer 5, the session layer, so in total it covers only four layers as highlighted in Fig. 5.2. OSI (Open Systems Interconnection), is

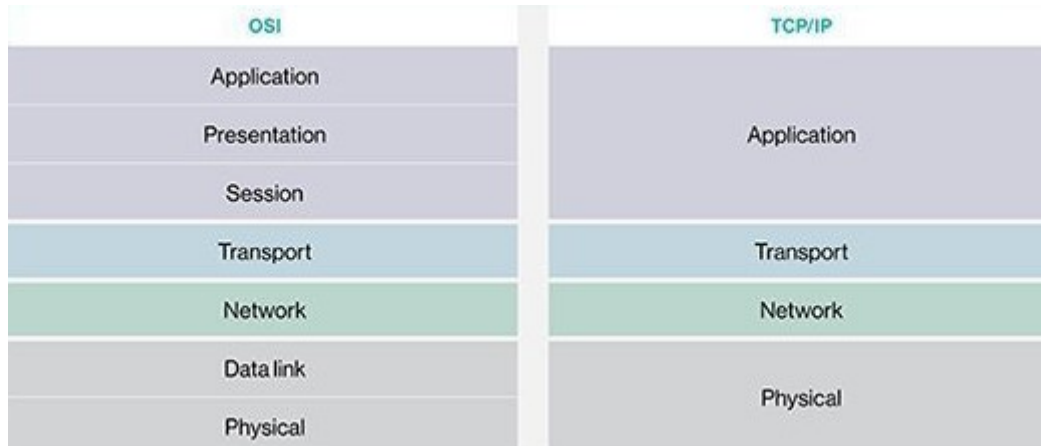


Figure 5.2. OSI VS TCP [42]

a reference model to explain how applications communicate over a network, and it is an abstraction of the functionality the protocol can provide, and provides a total of 7 layers [41]. The TCP protocol moreover can be seen as a black box that takes in input a flow of packets, de-multiplexes it and then sorts them in the order they

arrive, so that the application could see the right sequence of operations when those packets are sent to them [15]. Even if the data are sent asynchronously into the network, they are queued into the TCP stack and not removed until the receiver has acknowledged the reception of the data, and if no acknowledgment is received within a specific time frame, the data is re-transmitted back. The TCP protocol produce latency during the reorganization of the packets before re-transmission creating sometimes jitters. A class very similar to TCP protocol is the UDP (User datagram protocol) that does not provide the acknowledgments for dropping wasted packets during the transmission, but this is compensated by an higher speed transmission of the packets.

The TCP/IP protocol suite, is a very known protocol and it was widely used in the past and still used nowadays for the benefits it provide with respect to other communication protocols. The TCP/IP protocol was developed under the sponsorship of the Department of Defence for security reason, even if it is affected by some flaws on the security topic [3]. Nowadays, TCP/IP protocol is still used in the market mainly for the communication with commercial devices, because it provide the ability to connect different physical hardware basing on the Ethernet protocol standard cable connection. For this reason, many companies uses the TCP/IP protocol in order to interface with their devices due to the use of Ethernet cables that support the benefit to be plug and play and easy to install on their standard products.

In order to recall the previous means of communication, we refer to the serial communication that was used for example on diagnosis and testing a new system or configuring a firmware on a new motherboard, or in general to allow interaction between two computing systems. In order to accomplish to this task, serial ports were widely installed in the past on both local computing systems under testing and the remote computing systems.

The need of this communication arises when data are redirected by the remote computing system under test to the local computing system. A problem arises when doing this at the BIOS (basic input output system) or DOS (disk operating system) program level, because, when communicating with the firmware, the complexity and resulting implementation code size dictates the use of the UDP protocol or better the TCP/IP protocol [35].

Nevertheless, the TCP/IP protocol plays an important role for the development of this project work because it is the main communication protocol supported by the development hardware taken in consideration. Another communication protocol supported by the RIAPS platform is ZeroMQ, that is a pre-configured communication architecture in order to run inside the basic platform that allow the routing communication between all the nodes connected to it.

### 5.2.2 Sunspec protocol

Sunspec is a protocol which is provided by the Sunspec Alliance among over a hundred of companies producing distributed energy resources (DER) and solar and storage products. The purpose of this alliance is to provide a standardized information in order to enable the "plug and play interoperability among the resources. Sunspec is pointing on the application of this standard throughout the operational aspects of solar PV power and energy storage plants in the smart grids. Moreover this standard is approaching to residential, commercial and utility scale systems, promoting the industry innovation and growths and reducing costs [37]. Nevertheless, Sunspec information models are described and discussed in an open communication program, so that they could be accessed by multiple industries and in this way a common standard is defined that try to accommodate the requests from different fields converging into one channel. Sunspec models are therefore used by Sunspec modbus, a communication interface defined by IEEE 1547-2018, the U.S. national standard, and by a Sunspec Web Service interface.

The innovation that this protocol provides benefits regards different aspects such as [38]:

- TCP/IP based which is exploiting the Modbus TCP communication protocol.
- SunSpec information models are semantically identical to those incorporated in the IEEE 2030.5 and IEEE 1815 communication standards.
- Utilizes the HTTPS protocol which is used by most major internet applications and by the IEEE 2030.5.
- Supports the same security standard employed by IEEE 2030.5.
- Simple data transfer, very high speed, and low latency.
- Ideal for gateway applications incorporating IEEE 2030.5, IEEE 1815, or IEC 61850-7-420 standards.

The Fig. 5.3 exploits better the functionality offered by this protocol, among this, the ability to gather of all the information coming from different resources and to create a gateway to interconnect throughout internet to a Data center, where information are collected and analyzed and eventual commands are sent. This makes a huge step forward towards a futuristic agreement among different companies in the Electric and Electronic sector. Moreover the SunSpec Certified program releases certifications to the Sunspec member products by establishing objective criteria (regarding testing, security, communication result and quality controls) in verifying compliance to communication interface, in order to transparently interoperate with different "DER" system components.

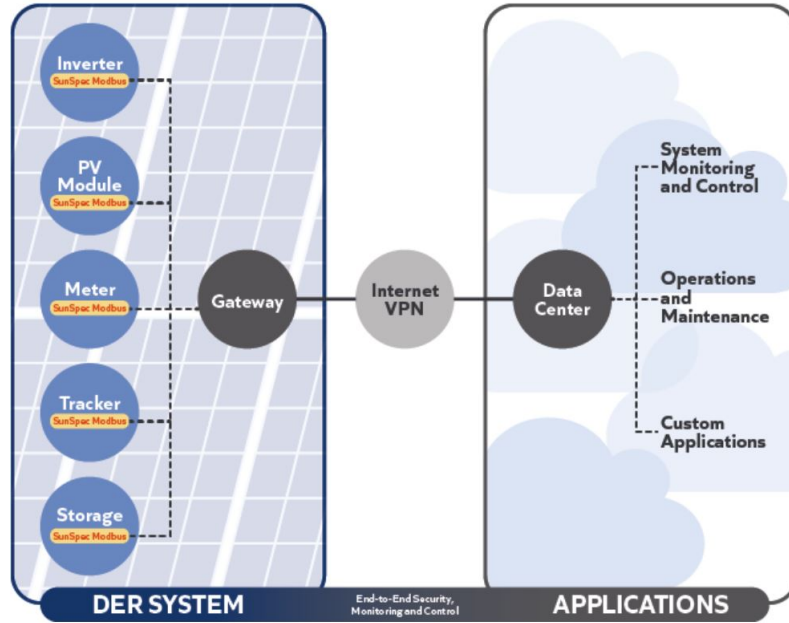


Figure 5.3. Sunspec standard representation [38]

### Discussion about the Sunspec protocol

Despite all the benefits that this great protocol and also standard can provide to the society of all over the world, it still present a point of continuous research. From the point of view of the real applications, some devices need to be continuously updated with new firmware until they meet the same requirements.

This protocol is mostly used when standard application or platforms are involved. Most of the time the tables and the information provided by this protocol are useful to make easy the computation and reduce the efforts, spending less time on analyzing modbus values.

In this real application Sunspec protocol is analyzed but not involved in deep. Since the "SMA STP-20000US-10" inverter supports the Sunspec specifications, it is also possible to use it in order to be conform with all the standards listed by the same. What is providing in the real application this standard, is a mapping of some previously selected registers, defined the most important, with a specified nomenclature and position organized in and clearly accessible common tables. Nevertheless in the application developed a "low-level" approach is used in order to deliver similar requirements. Thus, the selection and the conversion of values is manually done basing on the "Modbus" protocol profile. One of the reason this standard is not applied to the application developed, is that it does not properly fit with the platform requirements. In fact, some registers are accessible under the Slave ID=3,

classified in the "*SMA\_Modbus-TI-en-15*" manual, while the mapping from SunSpec parameters is provided with access through Slave ID=126 in "*SunSpec\_Modbus-TI-en-15*" manual. Regardless, in the custom application using RIAPS, the Slave ID, require to be taken as initialization input, so, this could not be modified during the simulation. The choice is therefore exclusive and since the SunSpec refers also to Slave ID=3 for the injection of the Grid Guard Code, necessary for enabling the writing capability in some registers, the choice to avoid it is made. This not comply any restriction in the future application as soon as Modbus registers numbers are modified with SunSpec ones and the injection of Grid Guard Code will be provided with Slave ID=126.

### 5.2.3 BeagleBone Black

The BeagleBone Black (BBB) Rev. C is an embedded computer system which run Linux as operating system. The BBB is fundamental for running the application because it provides the ability to run RIAPS platform as a layer over Linux. The use of this hardware has been defined by RIAPS developers, that designed the software to be fully integrated with the hardware functionalities. The hardware provide a list of connection ports such as: USB, Ethernet, UART, GPIO through which the BBB is able to interface with the external devices. Most of those connection require a minimum of knowledge of the hardware basic or at least the presence of a table classifying the ports, that might be activated through software, e.g. UART ports. Since the project's aim is to rely on plug and play capability and easy cabling, it is not used anyone of this ports, but the Ethernet one which is exploiting a double functionality. The first one regards the connection to router, that enable the communication through ZeroMQ protocol to the other nodes of the same type on the same sub-net. The other one is that commands and data are exchanged through TCP/IP also to other nodes of different types (external devices), and this provide the innovation from the RIAPS point of view.

### 5.2.4 Opal-RT 5600 simulator

OPAL-RT simulator is a product of OPAL-RT Technologies Incorporation, which is intended to develop and integrate a cost-effective rapid prototyping and real-time simulation systems. Opal-RT 5600 is optimized and fully integrated to execute the Mathworks products such as Simulink and Stateflow models.

OPAL-RT offers extremely scalable simulators, from single-processor rapid control prototyping systems to RT-LAB Distributed Real-Time hardware-in-the-loop Simulators containing up to 64 processors.

One of the benefits is the scalability because this simulator can be applied in many fields with the use of high-end multi-core processors that provides high performance results and real-time simulation data. It can also provide simulations at different

rates depending on the application, running inside power electronics components. It is also possible to run open-loop and closed-loop testing of embedded control units (ECUs) used in automotive, electrical systems, and aerospace applications [32]. The OPAL-RT used for this project lab for the model in the loop simulation, includes also the OPAL-RT OP5607 I/O expansion unit which provide additional input/output offering an high speed communication.

### 5.2.5 Inverter SMA STP 20000TL-US-10

The SMA STP-20000TL-US-10 is belonging to the class of solar inverters and provide, as well as all the inverters, the functionality to convert power from a DC source to AC which aim in this case is to supply the main grid attached to the commercial power line.

This specific inverter is designed in order to meet the American requirements, and is able to host a mid-large scale PV system. The DC input is rated between 600 and 1000 [V] offering large flexibility in the use of the PV power plants providing a maximum AC active power of 20[kW]. This solar inverter is equipped with two independent MPPT tracker and an arc fault detection system and an optimum shade management to provide a total efficiency around 98 %.

Other nominal parameter for the inverter in consideration are present in the manual of the vendor company [36].

The data connections and the real commissioned inverter is shown in Fig. 5.14 (b).

### 5.3 Model in the loop simulation

In this section are shown some logic aspects regarding the model creation finalized to establish a TCP/IP communication within the local computer and an OPAL-RT simulator. Since each step of the V-shape model need to be tested before proceeding forward to a next step, a model in the loop simulation is performed as first and testing results are pointed out. A connection model of the OPAL-RT device for the experiment is shown in Fig. 5.4.

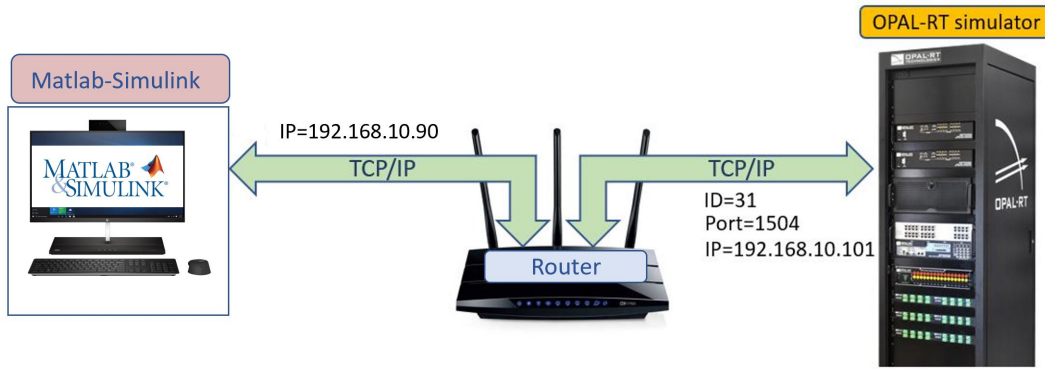


Figure 5.4. Connection model during MIL simulation

Opal-RT simulator played an important role on testing of the communication protocol that has been developed due to high performance exchanging rate data. A simulation including blocks provided by OPAL-RT technologies has been run on Simulink platform and connections to the OPAL-RT simulator is shown in Fig.5.5. To better understand, the simulation of the closed loop simulink model is shown in Fig.5.6. The "SM.MG.PBG" block in Fig. 5.8 is responsible for taking the output values from the Opal-RT and sending to the "SC.Console" 5.7 block. Successively, it is possible to modify the input values of the system and sending back to the OPAL-RT simulator. The parameter taken as input in the right hand side of the Fig. 5.7 by the multiplex are referring to the modbus block. The block variables are defined in a Excel file that gives the definition values represented in the table 5.1. The numbers in the console model are tunable during the simulation process, in this way it is possible to check the correctness though reading the just written ones. The sequence of the values type inserted is defined in the block definition, that take as input:

1. *Coil inputs* used to read either 0 or 1 values.
2. *Discrete inputs* used to either read or write 0 or 1 values.



Figure 5.5. Opal-RT Simulator

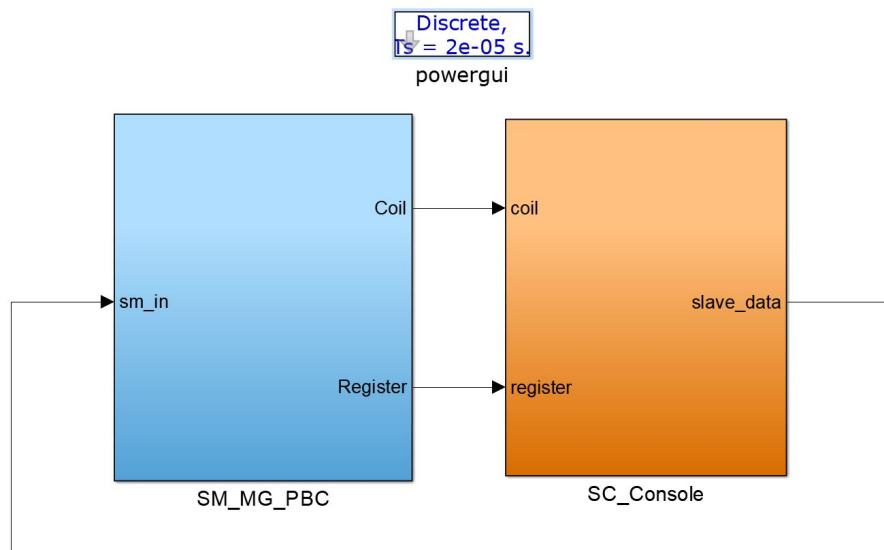


Figure 5.6. Complete Simulink model



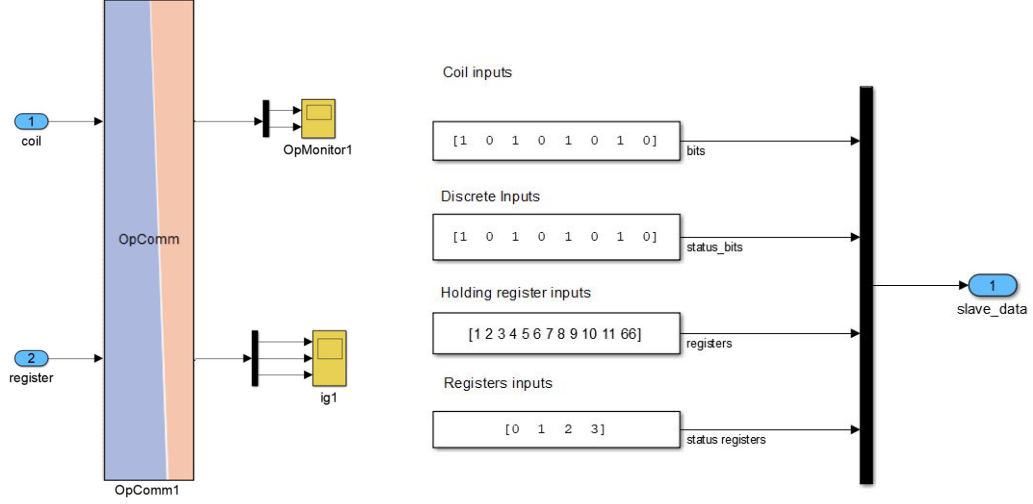


Figure 5.7. SC\_Console model

3. *Holding register input* used to read whatever values of holding type.
4. *Register input* used to read only whatever value.

However in this simulation it is only important to demonstrate the point number "3.", because it is the one used in the real application at the end. So referring again to the table 5.1, which is possible to modify, it is visible that there is a set of twelve input holding registers. For this purpose, twelve values are given as input to the block, as it is shown on the third row of the multiplexer in Fig. 5.7. After all, the four values type are packed and sent to the "slave\_data" variable which is connected to the "sm\_in" variable of the *SM\_MG\_PBC* block as shown in Fig.5.8.

Frequency rate change blocks are applied every time the values are crossing the Opal-RT simulator boundaries because of the frequency rate difference.

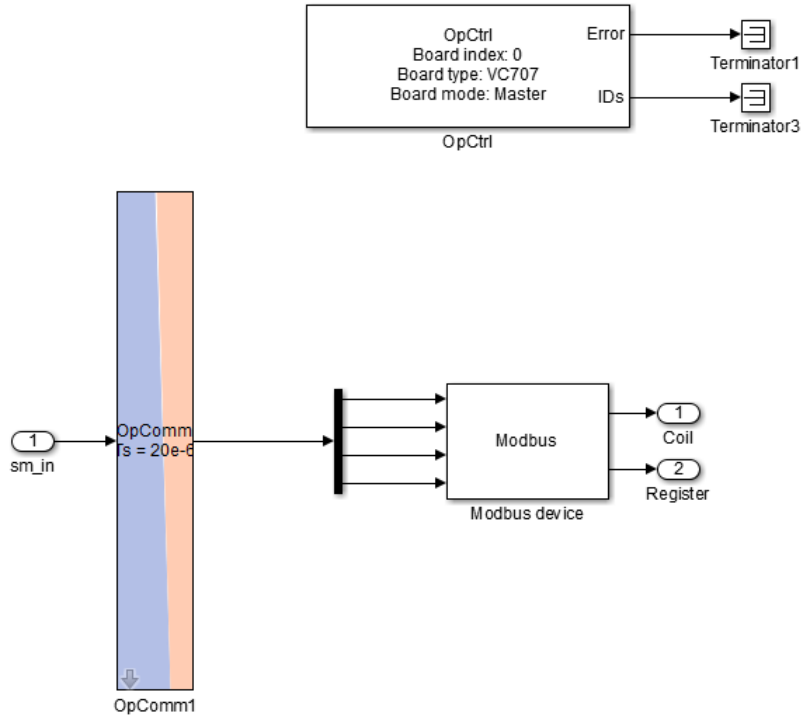


Figure 5.8. OPAL RT model

Regardless, in this simulation step it is demonstrated the effectiveness of running blocks correctly. Moreover, after the modification of the input holding register in the *SC\_Console* block it is accomplished the initial requirement of reading correctly through the modbus-TCP communication, while in the next section thanks to the external software it possible to see how to deploy also the writing of the holding registers.

Device name	modbus_slv_tcp
Device type	Slave
Protocol	TCP
Network interface	eth1
IP Address	192.168.10.101
Slave ID	31
Port	1504
CPU	0
Verbose	1
Addresses of coil inputs	0, 1, 2, 3, 4, 5, 6, 7
Addresses of coil outputs	16, 17
Addresses of discrete inputs	0 1 2 3 4 5 6 7
Addresses of holding register inputs	0; 1; 2; 3; 4; 5; 6; 7; 8; 9; 10; 11
Addresses of holding register outputs	20 - 21 - 22 - 23 - 24 - 25
Addresses of input register	0-1-2-3
Polling frequency(ms)	100

Table 5.1. Functional values for Modbus TCP/IP block

The functional table 5.1 is hosting predefined values taken from the modbus block as a configuration file. The Network interface indicate the Ethernet port which is interfaced the OPAL-RT, then the IP address assigned from the router to the OPAL-RT, the slave ID address and the port used for the TCP protocol, then it is shown a CPU value corresponding to the CPU number of the OPAL-RT used to run the simulation, finally quantity of all registers considered in the list and last but not the least the polling frequency the simulation run.

## 5.4 Software in the loop simulation

This section's aim is to provide an overview of the work developed during the simulation phase of the software. In particular the software responsible for running the reading and writing operation and the simulation inside multiple devices. The first part is related to the analysis of the BBB (Beaglebone black), then a part related to the preparation of the software developed and tested on both the hardware.

A connection model of the OPAL-RT device for the experiment is shown in Fig. 5.9. In order to demonstrate the capability to read and write under the TCP/IP protocol, a software in the loop simulation is performed. The algorithm presented in this section is really row and voluntarily left uncured, because it only points to demonstrate the possibility to communicate under the protocol constraints. In this occasion for this test, one BBB is connected to a router, which is able to see on the

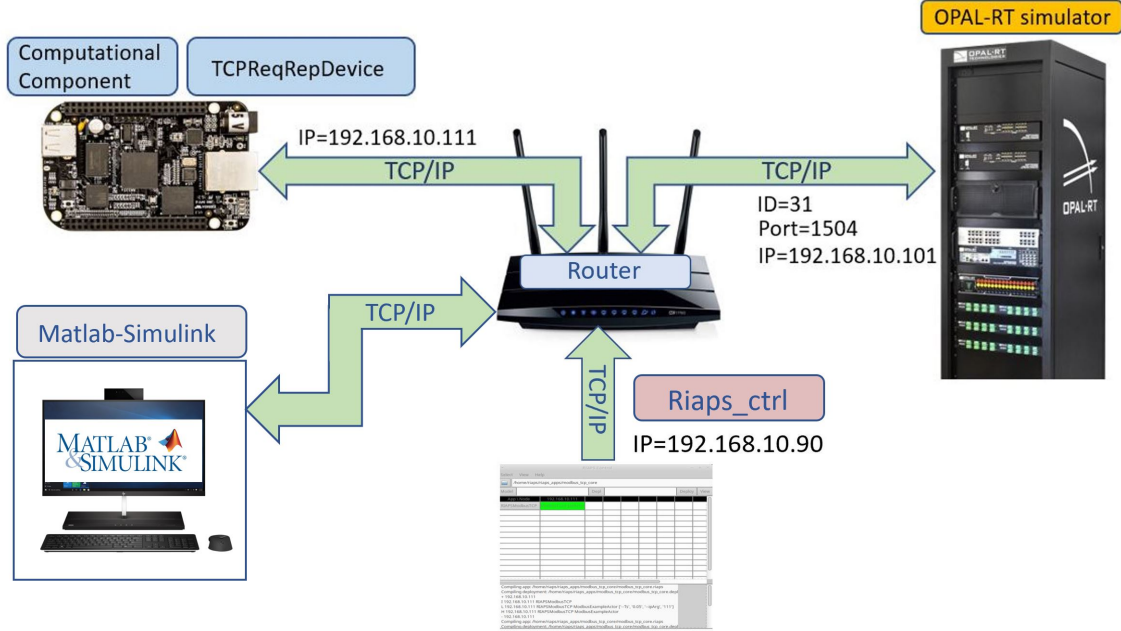


Figure 5.9. Connection model during SIL simulation

same sub-net the OPAL-RT simulator. The results exploits that the reading and writing operations under the TCP/IP protocol is performed correctly, but, since it is only a simulation, numbers read and written are meaningless.

Regardless the full code implementation is presented on the Appendix A, while a code snippets in Fig. 5.10 belonging to the *ComputationalComponent.py* file is presented to show where the values presented in Fig. 5.12 are coming from. With

```
'''Write multiple holding registers'''
self.values = [4000, 3000, 3276, 1000, 2000, 0]
self.command = CommandFormat(ModbusCommands.WRITEMULTI_HOLDINGREGS, self.defaultInitialReg, self.defaultNumOfRegs, self.values, self.signedDefault)
```

Figure 5.10. Writing function in *ComputationalComponent.py*

this simulation it is also possible to demonstrate the capability to write the holding registers and to visualize both the reading and the writing of the holding registers, that is preformed correctly.

In Fig. 5.11 it is possible to see the values that previously was injected, during the simulation, inside the third row of the console model in Fig. 5.7, read from the RIAPS platform. Instead in Fig. 5.12 it is possible to visualize the values injected from the RIAPS platform in the *ComputationalComponent.py* file and read by the simulink platform in the plot block named "Ig1" corresponding to the registers in Fig. 5.7.

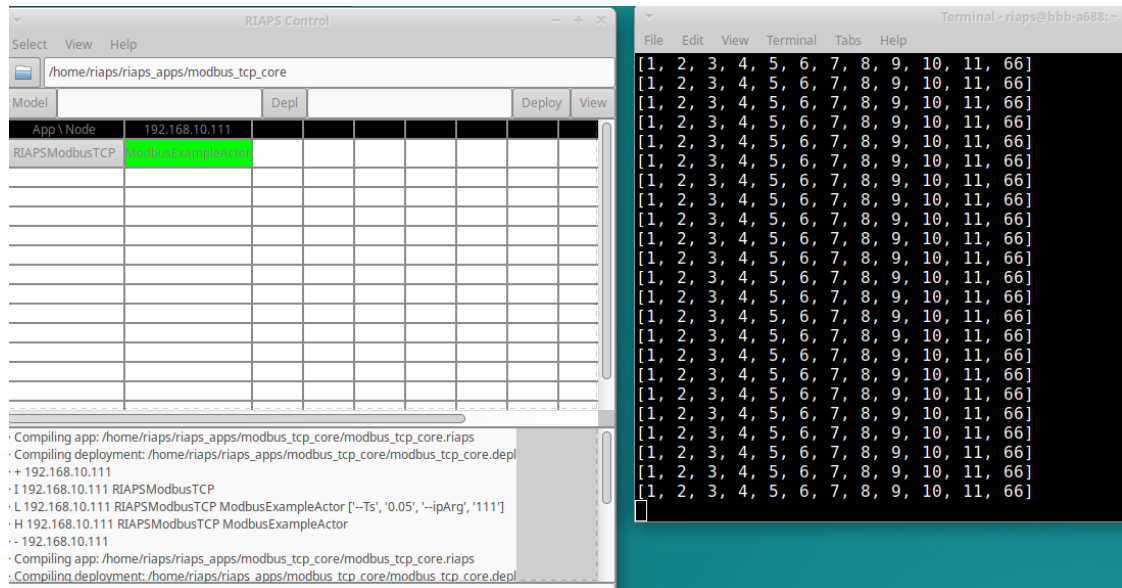


Figure 5.11. Reading values from RIAPS

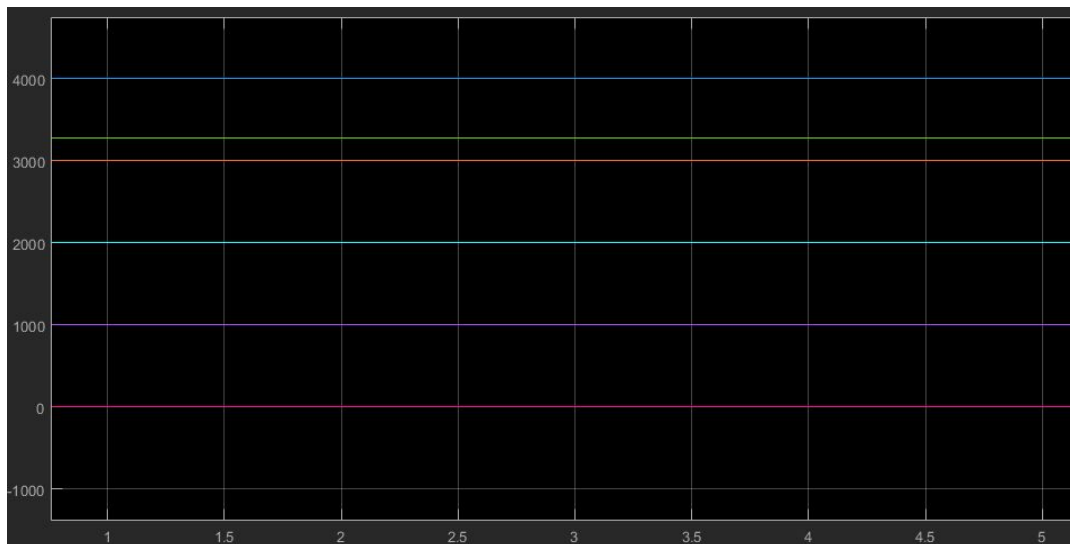


Figure 5.12. Reading values from Simulink

In order to introduce to the next step a real implementation is considered and running the simulation on a commercial device.

## 5.5 Hardware in the loop testing and validation

In this section, that reflects the integration and testing step of the V-shape model, it is demonstrated that the communication is performed correctly in a real hardware. The SMA company provide a lot of devices which communication protocol is based on a TCP/IP because it is the easiest and reliable way to communicate with. In particular the device analyzed is a solar inverter SMA STP-20000TL-US-10, where it is tested, for the first time, the ability to integrate the RIAPS platform. A connection model of the inverter for the experiment is shown in Fig. 5.13. The connections shows the router connected with two Beaglebone, representing the control and the logging operations. Moreover the router is connected to the commercial inverter and to the local computer from where Riaps platform is deployed. Real tests are performed within this structure with the final objective to control data and operations from the SMA device. The real connection, showing the harness within solar

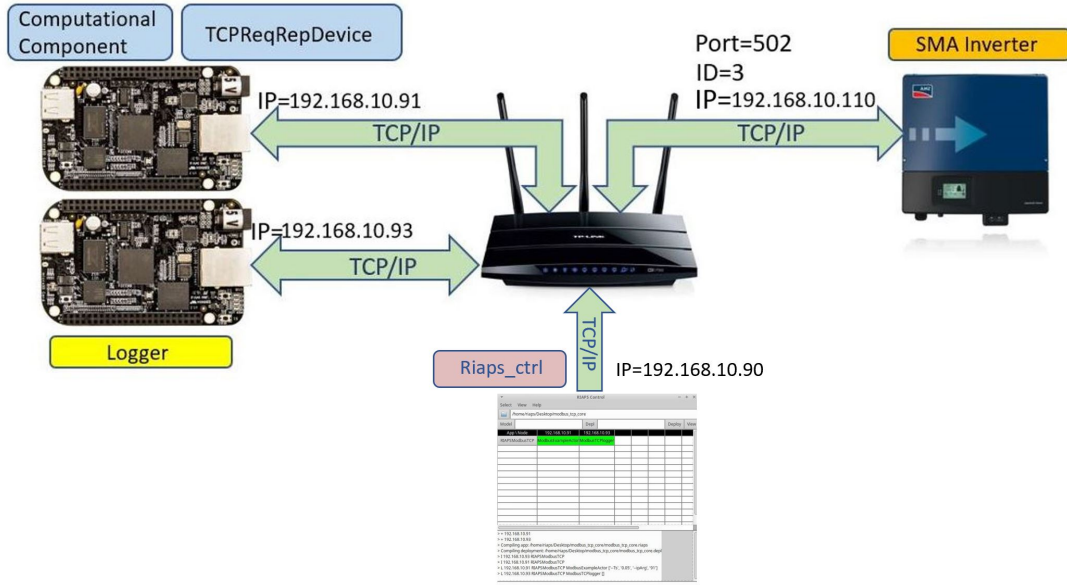


Figure 5.13. Connection model during HIL simulation

panels providing a 10kW max DC power is shown in Fig. 5.14 (a). The real inverter configuration is shown in Fig. 5.14 (b), instead the router connections in Fig. 5.14 (c).



Figure 5.14. Solar panels (a), Inverter SMA STP-20000TL-US-10 (b), Router and BBBs connections (c)

## 5.6 Code examination and software tool support

In this section it is presented the summary requirement, the software tool used and its related support given during the developing phase, concluding with specific description about the code structure.

The general requirement is to demonstrate the full integration of the RIAPS platform

through the development of an hardware specific application, which therefore is able to interface with the inverter in consideration to continuously monitor data and autonomously send commands, exploiting all the benefits that the platform is able to offer. One of the tools used for the development is "Sunny explorer" platform, that allow to monitor continuously the values during the simulation and verifying the correctness of the execution of the operations. Since Sunny explorer is the platform provided by the "SMA" company, it is fully integrated to host all data and send commands to the inverter. The software is presented with a GUI (graphical user interface) so that the user is able to read and write values in a easier way. Moreover Sunny explorer" platform plays an important role on the development of this project, because, since the platform has been calibrated during the commissioning phase of the inverter, it provide true values such as continuous monitoring through external measurements. Since the flow of data travel on the same Ethernet interface, through a special device, called "*speed-wire/web-connect*", the platform provide also charts of data. One of the most important graph, is the one used mainly for analyzing the daily energy acquired by the solar inverter, that in this project is useful during some tests on power limitation, explained better in the next chapter. Another tool that is used, this last one third party software, is "*QmodMaster*" which is providing information regarding the values associated with the register numbers in real time. All of them constitutes a real help to the development and calibration of the code proposed, as it possible to see in the simulation and outcomes during several tests.

### 5.6.1 Hardware configuration

This section' aim is to provide a basic knowledge of the hardware connection for the developing if this project. From the point of view of hardware connections, there are two class:

1. The first one is related to the power line connection divided in turn into:
  - DC side, where the inverter acquire power from the the solar PV array.
  - AC side, where the inverter release the converted AC power to the main grid called also "Public electricity" side.
2. The second one is related to the data line connection, which is the part analyzed in deep in this project.

As it is possible to see in Fig. 5.13, representing only the data line, the wiring connection is minimal, as accomplishment of the task requiring the use of only TCP/IP protocol. All the devices are hard-wired connected through only Ethernet cables to a router, which is the one, though which the software platform distribute the application, responsible for managing the flow of data and commands.



### 5.6.2 Software configuration

The multiple software parameters involved in this project are modified and tuned in order to accomplish the final task. First of all from the "*Sunny-Explorer*" platform, the procedure able to enable the external communication is performed. Moreover, following the guide from "Technical information", the following steps are:

- Start the sunny explorer platform previously installed on a computer, creating a Speed-wire system.
- Log-in as "installer" in the Speed-wire system inserting the related password.
- Select the inverter to be configured in the system tree, because the platform can host management of multiple ones.
- Select the tab "Settings" and go inside the parameter group "External communication".
- Select edit to modify parameters enabling the communication "TCP Server" under the category "Modbus" and save the modifications.

Some constraints about response time of the controller are modified, always from the platform, in order to provide a faster behaviour, especially during the shut-down phase.

Instead "QmodMaster" is configured to communicate in a TCP mode to the same address assigned from the router to the inverter.

Finally the RIAPS platform is configured in order to use some custom libraries able to interface directly to the inverter in a TCP mode. The libraries used are installed sending the following command from both the host development computer and from the BBBs after logging as "SSH":

```
$ sudo pip3 install umodbus
```

Then a new library is configured, the "*tcpModbusComm*", that is used to interfere directly with the "*umodbus*" previously installed. The "*tcpModbusComm*" is created on purpose for this project, but it constitutes a starting point to enable communication with all devices that supports TCP protocol, so that the Riaps platform can extend its applications on a wider field.

### 5.6.3 Application architecture

#### `modbus_tcp_core.riaps`

The "*modbus\_tcp\_core.riaps*" file is intended to resume all the interactions between components and the component-device in turn able to interact with the hardware and to establish a communication pattern by using predefined message topics. In

Appendix B is represented a full code implementation with particular focus highlighting bounds created by the publisher/subscriber and request/reply patterns on Fig. B.1.

In particular as previously described the interactions pattern are of type "req/rep" and "pub/sub" respectively between the "*ComputationalComponent*" and the "*ModbusTcpReqRepDevice*" and between the "*ComputationalComponent*" and the "*ModbusTCPLogger*" is established.

One more interaction is provided in this example with respect to the basic implementation used for "SIL" simulation that is the one between the "*ComputationalComponent*" and the "*ModbusTCPLogger*" in order to separate the data information coming from the inverter and logging of single operations performed.

There are two actors, in particular the "*ModbusExampleActor*" hosting the "*ComputationalComponent*" component and the "*ModbusTcpReqRepDevice*" component-device and the "*ModbusTCPLogger*" actor hosting the "*ModbusTCPLogger*" component. In this way is possible to deploy the run-time application in two different nodes deployed in two different BBBs.

A simplified version representing the whole interactions is represented in Fig. 5.15, where some steps are enumerated to enhance the comprehension and the facilitate the explanation on the following steps. Three timers are used in this application, their timing constraint and the necessity is different for each one of them.

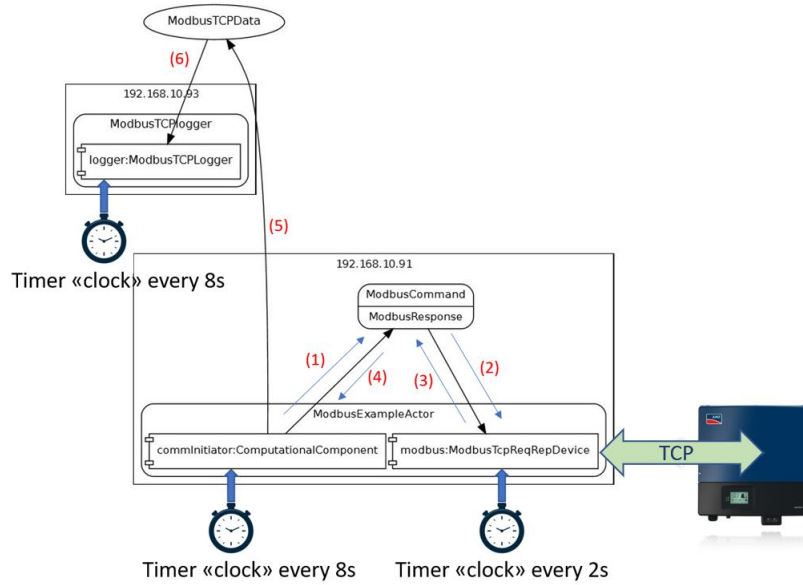


Figure 5.15. File "modbus\_tcp\_core.dot"

## ComputationalComponent.py

The "*ComputationalComponent*" is one of the most important file, because it is responsible to interface with the device-component "*ModbusTcpReqRepDevice*". In the "*ComputationalComponent*" are executed multiple tasks, such as opening requests and receiving data as reply, the classification of some registers, the conversion step for each one of them, finally also developing the logic of execution of some operations.

In particular for the choice of the registers to read, a certain important amount of them are selected and sent in chunks in order to reduce the number of request to be made. Since this application is developed in order to read from a starting register for a certain amount of them, and since the some particular modbus registers for the inverter are distant, a smart solution is adopted. The solution is to create an array containing inside the values of the same type for a certain length of read and in turn rotate to require the same values every cycle.

In this way two issues are solved, the first one is the classification of some values of the same type, the other is to override the distance between themselves, so that the limit of "125" read holding registers is never reached.

On the other hand, for the writing part, the same structure is adopted, but the function is different, and the returning value which is used by the algorithm to understand whether the writing is "successfully" or "failed".

Moreover the writing operation is performed in order to previously read from the register, then compare the value with the one desired to write, and if results from the comparison shows that they are not the same, the writing operation is performed, otherwise it is skipped raising an information that says "*value already written*". This functionality implemented is not trivial, because, cyclical writing operation of R/W (read and write) or WO (write only) registers could lead to destruction of the flash memory of the device, as soon as, the values are intended for long-term storage of device settings. Only few of them regarding the limitation power from PV system control are an exception and it could be cyclically modified.

The most important registers are selected and represented in a table 5.2:

Description	Address	Type	Format	Unit	Access
Total Energy yeld	30529	Wh	U32	FIX0	RO
Daily Energy yeld	30535	Wh	U32	FIX0	RO
Operating Time	30541	s	U32	FIX0	RO
Feed-in time	30543	s	U32	FIX0	RO
DC current input	30769	A	S32	FIX3	RO
DC voltage input	30771	V	S32	FIX2	RO
DC power input	30773	W	S32	FIX0	RO
AC power output	30775	W	S32	FIX0	RO
AC power phase L1	30777	W	S32	FIX0	RO
AC power phase L2	30779	W	S32	FIX0	RO
AC power phase L3	30781	W	S32	FIX0	RO
AC grid voltage phase L1	30783	V	U32	FIX2	RO
Grid voltage phase L2	30785	V	U32	FIX2	RO
Grid voltage phase L3	30787	V	U32	FIX2	RO
AC grid current	30795	A	U32	FIX3	RO
AC grid Frequency	30803	Hz	U32	FIX2	RO
AC Reactive power	30805	VA <sub>r</sub>	S32	FIX0	RO
AC Reactive power L3	30807	VA <sub>r</sub>	S32	FIX0	RO
AC Reactive power L2	30809	VA <sub>r</sub>	S32	FIX0	RO
AC Reactive power L3	30811	VA <sub>r</sub>	S32	FIX0	RO
AC Apparent power	30813	VA	S32	FIX0	RO
AC Apparent power L3	30815	VA	S32	FIX0	RO
AC Apparent power L2	30817	VA	S32	FIX0	RO
AC Apparent power L3	30819	VA	S32	FIX0	RO
Plant main connection	30881		U32	ENUM	RO
Language of UI	40013		U32	ENUM	RW
Mode of multifunc. relay	40575		U32	ENUM	RW
Active power limit	40915	W	U32	FIX0	RW
Fast shut-down	41253		U32	ENUM	RW
Grid Guard-Code	43090		U32	FIX0	RW

Table 5.2. Selected Modbus registers for SMA STP 20000TL-US-10

The chunks of reading registers are in this way organized:

- *PLANT MAIN CONNECTION* containing one type values starting from register "30881" and reading for a length of two 16 bits registers.
- *DC* containing three DC values starting from register "30769" and reading for a length of six 16 bits registers.

- *AC1* containing seventeen AC values starting from register "30775" and reading for a length of thirty four 16 bits registers.
- *AC2* containing three AC values starting from register "30977" and reading for a length of six 16 bits registers.
- *ENERGY* containing two energy values starting from register "30529" and reading for a length of four 16 bits registers.
- *TIME* containing also two time values starting from register "30541" and reading for a length of four 16 bits registers.

Instead the chunks of reading/writing register are in this way organized:

- *GRID GUARD-CODE* containing one type values starting from register "43090" and operating either to read and to write for a length of two 16 bits registers.
- *LANGUAGE* containing one type value starting from register "40013" and operating either to read and to write for a length of two 16 bits registers.
- *OPERATING MODE OF MULTI-FUNCTIONAL RELAY* containing one type values starting from register "40575" and operating either to read and to write for a length of two 16 bits registers.
- *POWER LIMIT* containing one type starting from register "40915" and operating either to read and to write for a length of two 16 bits registers.
- *ON-OFF* containing one type starting from register "41253" and operating either to read and to write for a length of two 16 bits registers.

Moreover either the read and write values are saved inside a combination of two "*nametuple*". In particular there is `RegSet(RegNum,value)` where "RegNum" is the number of the register and "value" is the value contained in the same register. Then there is "inputholdingRegisters" "*nametuple*" containing all the desired values to write and "holdingRegisters" "*nametuple*" containing all the desired read values. The combination is done in such a way that both "inputholdingRegisters" and "holdingRegisters" "*nametuples*" contains inside the "RegSet" "*nametuple*". This implementation allow to save variables replacing every cycle only values that changed through comparisons. This feature is designed for possible future applications where values could come from different devices and different values are required to be compared. During the reading function, since most of the register listed and taken in consideration are 32 bits long, it is necessary to identify the higher level bit and the lower level bit of the two 16 bits composing a single register value and doing the right conversion, taking into account also the type, format and unit of the value. The unit "FIXn" indicate with "n" the quantity of decimals to consider

while the format indicate whether the first bit of the value in question need to be interpreted as a signed or unsigned.

The application is designated to call the same function "sendModbusRequest" for either read and write and then inside it is performed a sorting depending on the real request. Basing on the reply, it is called another function "exitandcheckvalue" used just to exit from the request, check whether the value is consistent, and then send to a "registertable" function able to classify the value depending on the unit, type and format. Conversion is applied to every value read and consecutively to every one desired to be written. The reply is sent back to the upper level function "exitandcheckvalue" responsible for sending that to the logger.

In order to resume the operation done are basing on the 8 second timer, the "*ComputationalComponent*" send a request to the "*ModbusTcpReqRepDevice*" (1), then it wait for the operations, receive the value (4) and send it back to the "*ModbusTCPLogger*" (5) on "tx.ModbusTCPData".

### **ModbusTcpReqRepDevice.py**

The "*ModbusTcpReqRepDevice*" is also essential for the correct execution of the operations. It is responsible to interface with the "*ModbusTcpComm*" library which is using "umodbus" library installed externally. Moreover, the duty of the "*ModbusTcpReqRepDevice*" is to interface with the inverter, specifying parameters such as: slave ID, IP address and port number. The functionality are clocked by a timer, which is calling the loop every 2 seconds, in order to capture in a more accurate way any request coming from the "*ComputationalComponent*" that is operating slower for this purpose. The basic operation are resumed as follow: Receive the request from the "*ComputationalComponent*" on "ModbusRepPort" (2), call the "unpackCommand" function and sort the request to send over the inverter through the library. Then receive the response value from the inverter, and send it back to the "*ComputationalComponent*" on "ModbusRepPort" again (3).

### **ModbusTCPLogger.py**

The "*ModbusTCPLogger*" is a secondary file, but this does not mean that its role is useless during the run-time application. Its duty is to receive the values sent by the "*ComputationalComponent*" on the "rx.ModbusTCPData" port (6) and sort it listing correctly in order to facilitate the visualization. This data are the one not containing the information of the requests but the values of the register, so data cleaning, play an important role here. The timer, establishing a periodic call, in this file, is only used for printing a value separator between a block of old data and the new one, as soon as, the data logging happen through an event triggered port. Regardless, in order to match the two events, event and time triggered, the time frame of 8 seconds for the "*ComputationalComponent*" timer need to coincide with

the one of the *"ModbusTCPLogger"* timer.

Values on *"rx\_ModbusTCPData"* port are received with the same block structure they are sent from the *"ComputationalComponent"*, listed in a description array containing the name of the register and the value associated or the description in case of some particular registers. The *"ModbusTCPLogger"* component in this way act as a buffer acquiring the data, sorting, listing in columns associating to each register the correct value and finally printing these to the terminal for visualization.

### **modbus\_tcp\_core.depl**

The *"modbus\_tcp\_core.depl"* is the concluding file of the application, it constitutes together with the *"modbus\_tcp\_core.riaps"* the main files used for deploy the application to the designated target nodes. In particular the *"modbus\_tcp\_core.depl"* deploy the two actors *"ModbusExampleActor"* and *"ModbusTCPlogger"*, each one of them containing the components specified in the *"modbus\_tcp\_core.riaps"*, in two target nodes respectively *"192.168.10.91"* and *"192.168.10.93"*. In this way it is possible to separate the logging of operation accomplishments from real data value logging.





# Chapter 6

## Test-bench results

Until this moment, all the components, methods and tools used to accomplish the final objective of this thesis work have been analyzed stepping inside the whole project explanation, preparation and execution. In this section are taken in consideration some simulations about the developed code and a briefly introduction of the conclusion for each one of them. As remark, it is necessary point out that real measurements are taken by already implemented and calibrated sensors inside the inverter, that are able to self monitor and control itself during the normal operations. So far, the use of some software able to interfere with the hardware in consideration are used to read data and eventually also to write. This is useful during the development of the application, because it allow to establish confirms on the procedure of the interpretation of the single values. In particular "QmodMaster" is interesting to interface with the single registers, but it require a minimum of configuration even if for this procedure is necessary to operate manually through a simple GUI (graphical user interface). Several tests have been performed during the development phase, but, for simplicity they are briefly described to give space to the most important ones, that relates to the *Active power limitation* and to the *Shut-down test*, analyzed in the following sections. Furthermore to point out some imperfections that come out from the properly executed work some hints are provided in order to conclude with a critical review.

### Main obstacles to overcome

One of the most important obstacle to overcome is the interpretation of the read value. After a research on the "SMA" paper works, a solution is pulled out. The registers representing 32 bits are interpreted as follows:

- If it is a U32 (unsigned 32 bit) value, this is the case for most of the register taken in consideration, since it is a composition of two smaller 16 bits value, it is necessary to take into account the first data value as higher bit value (x[1])

and latter one as the low bit value ( $x[2]$ ). This lead to the use of the simple formula 6.1:

$$2^{16}x[1] + x[2] \quad (6.1)$$

- If it is S32 (signed 32 bits) value, only few registers taken in consideration with this feature, there is always to take in consideration the subdivision of 16 bits values, but then a different conversion must be applied. Since the sign is included in the data value, one solution is to consider the first bit from the left of the first 16 bits block, as the sign. In particular considering "0" positive value and "1" negative value and then considering the rest as a normal unsigned value as previously analyzed. The other possible solution, is to subtract the value to the highest value of representation as considered in the equation 6.2. This is the case analyzed in the values for the reactive power and subsequently also in the apparent power as effect on the reactive power. The sign of the data read is really important in the case of the reactive power, because it determine whether it is inductive reactive power (positive) or capacitive one (negative), so the sign cannot be neglected.

$$2^{16}(x[1] - 2^{16}) + x[2] - 2^{16} \quad (6.2)$$

Another issue to overcome is the validation of all the interesting registers proposed in the inverter data-sheet. Due to the short description in the data-sheet, the operation of some of them is ambiguous, because of repetition in different registers numbers declared to work with the same behaviour.

One of this example is related to the register of active power, several registers declare to operate the same functionality but only one is working fine.

Another example represent the one for the shutting-down test where a similar problem is presented. This problem is overcome by doing several tests of different registers, polling requests either to read and to write, obviously only if the accessibility allow this last one operation. In order to perform the shutting-down test, some parameters have been modified through the sunny explorer platform, that mainly are related to the timing constraints in order to make faster the response of the inverter during the simulations.

One more problem to overcome is the inconsistency of some data value, in fact when the inverter is turned off, the controller send initialized data values, because the sensors are not triggered and no updated value is assigned. In the particular case of the left side of Fig. 6.1 it is shown that the DC input voltage and power are initialized to the maximum value (view from "QmodMaster" simulator). The solution is based on experimental data, whenever some particular values are pointed out, in most of the registers, a "zeroing" operation of this values is performed by the new software platform, allowing to correctly read the data values also when the operating mode of the inverter is "Off". Another simulation inconsistency is shown in the right hand

side of Fig. 6.1 during the developing phase of reading only registers in a different instance of the previously seen on the left hand side. One more obstacle is repre-

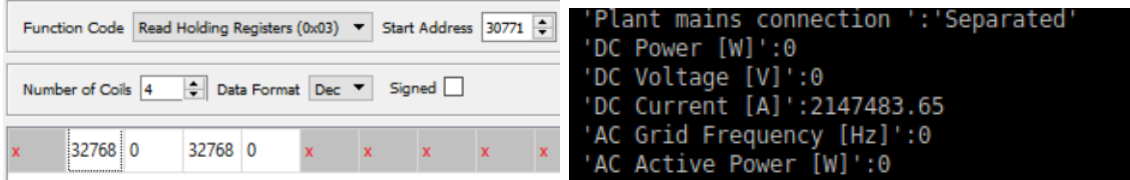


Figure 6.1. Inconsistency on DC values from "QmodMaster" and "RI-APS" in different instances

sented by the "Grid Guard-Code", which is a special code provided by the same vendor company of the inverter on request, that is responsible for enabling the possibility to access the writing operation of the R/W (read and write) registers. The particular register is hosting this special code, unique for each product associated with a serial number of the commercial product. The register analyzed is "43090" and it is U32 type, hosting in the first 16 bits the first five digits and in the latter 16 bits the last five digits of the code. Moreover this is the only register that is allowed to be written without any other code, and once written the right code, it should return the "1" value only if this is correct and the user is logged in or "0" if this last one is not correct and the user is logged out without any permissions.

Although the idea of the working principle is great, this continues to be "0" even if the "Grid Guard-Code" inserted is correct and the access is allowed. In order to supply to this problem, during the development of the code, several test have been made, using a generic non-functional register, where it is possible to change the language of the device interface. Following the experimental data, it is demonstrated that, if the language is correctly changed, after sending the correct code, the writing capability is unlocked allowing access to all registers type. Since this is a easy feature to be fixed, and since the company is continuously releasing update of the firmware, the problem could be solved soon, and consequently the RIAPS code developed could be adapted easily to the new situation.

One of the problem that luckily it is not presented is that the inverter data are accessible by multiple modbus master simulator, avoiding to point it out the problem of concurrency of operation. Since this is not a problem, data value inside the, read or write by RIAPS application during the developing phase, where double-checked by the other two platforms, previously mentioned polling the same registers at the very same time.

## 6.1 Earlier steps

During the development phase, some preliminary tests are performed in some specific registers, in order to demonstrate the basic working principle, such as "Language" and "Operating mode of multi-functional relay". They are previously tested because retained less important to be continuously accessed with respect to some others "critical" registers during the execution phase.

In fact, in one of the first steps of the development phase, only five group of registers were read and no writing operation is performed.

Those as shown in Fig.6.2 corresponds to:

- *PLANT MAINS CONNECTION*
- *DC*
- *AC1*
- *TIME*
- *ENERGY*

As it is possible to see in this phase the Logger is not sorting operations in any value, so that it only receive and print data in a different terminal and this is the reason why the printing stage appear such as a description.

```
[info]:[12-01-2019 23:37:22.152]:[6685]:ModbusExampleActor.commInitiator:Converted values :[1575243442,1517735]
[info]:[12-01-2019 23:37:22.270]:[6685]:ModbusExampleActor.commInitiator:Converted values :[1575243442,1517735]
[info]:[12-01-2019 23:37:22.421]:[6685]:ModbusExampleActor.commInitiator:Converted values :[1575243442,1517735]
[info]:[12-01-2019 23:37:22.565]:[6685]:ModbusExampleActor.commInitiator:Converted values :[1575243442,1517735]
[info]:[12-01-2019 23:37:22.746]:[6685]:ModbusExampleActor.commInitiator:Converted values :[1575243442,1517735]
[info]:[12-01-2019 23:37:22.886]:[6685]:ModbusExampleActor.commInitiator:Converted values :[1575243442,1517735]
[info]:[12-01-2019 23:37:24.153]:[6685]:ModbusExampleActor.commInitiator:Converted values :[1575243444,152508]
[info]:[12-01-2019 23:37:24.241]:[6685]:ModbusExampleActor.commInitiator:Converted values :[1575243444,152508]
[info]:[12-01-2019 23:37:24.290]:[6685]:ModbusExampleActor.commInitiator:Converted values :[1575243444,152508]
[info]:[12-01-2019 23:37:24.536]:[6685]:ModbusExampleActor.commInitiator:Converted values :[1575243444,152508]
[info]:[12-01-2019 23:37:24.696]:[6685]:ModbusExampleActor.commInitiator:Converted values :[1575243444,152508]

Dec 01 23:37:18 bbb-bcb1 RIAPS-DEPLO[15956]: [info]:[12-01-2019 23:37:18.812]:[15522]:ModbusTCPLogger.logger:RECEIVED {'Plant mains connection : Separated'}
Dec 01 23:37:20 bbb-bcb1 RIAPS-DEPLO[15956]: [info]:[12-01-2019 23:37:20.316]:[15522]:ModbusTCPLogger.logger:RECEIVED {'Total yield [Kwh]': 1288.058, 'Daily yield [Wh]': 4951}
Dec 01 23:37:20 bbb-bcb1 RIAPS-DEPLO[15956]: [info]:[12-01-2019 23:37:20.467]:[15522]:ModbusTCPLogger.logger:RECEIVED {'DC input Voltage [V]': 0, 'DC input Current [mA]': 0, 'DC input Active Power [W]': 0}
Dec 01 23:37:20 bbb-bcb1 RIAPS-DEPLO[15956]: [info]:[12-01-2019 23:37:20.646]:[15522]:ModbusTCPLogger.logger:RECEIVED {'Operating Time [h]': 582.54, 'Feed in time [h]': 542.39}
Dec 01 23:37:20 bbb-bcb1 RIAPS-DEPLO[15956]: [info]:[12-01-2019 23:37:20.802]:[15522]:ModbusTCPLogger.logger:RECEIVED {'AC Active Power [W]': 0, 'AC Active power phase L1 [W]': 0, 'AC Active power phase L2 [W]': 0, 'AC Active power phase L3 [W]': 0, 'AC Voltage phase L1 [V]': 0, 'AC Voltage phase L2 [V]': 0, 'AC Voltage phase L3 [V]': 0, 'AC Tot Grid Current [A]': 0, 'AC Grid Frequency [Hz]': 0, 'AC Reactive output Power [VAR]': 0, 'AC Reactive power phase L1 [VAR]': 0, 'AC Reactive power phase L2 [VAR]': 0, 'AC Reactive power phase L3 [VAR]': 0, 'AC Apparent power [VA]': 0, 'AC Apparent power phase L1 [VA]': 0, 'AC Apparent power phase L2 [VA]': 0, 'AC Apparent power phase L3 [VA]': 0}
Dec 01 23:37:20 bbb-bcb1 RIAPS-DEPLO[15956]: [info]:[12-01-2019 23:37:20.923]:[15522]:ModbusTCPLogger.logger:RECEIVED {'Plant mains connection : Separated'}
Dec 01 23:37:22 bbb-bcb1 RIAPS-DEPLO[15956]: [info]:[12-01-2019 23:37:22.276]:[15522]:ModbusTCPLogger.logger:RECEIVED {'Total yield [Kwh]': 1288.058, 'Daily yield [Wh]': 4951}
Dec 01 23:37:22 bbb-bcb1 RIAPS-DEPLO[15956]: [info]:[12-01-2019 23:37:22.426]:[15522]:ModbusTCPLogger.logger:RECEIVED {'DC input Voltage [V]': 0, 'DC input Current [mA]': 0, 'DC input Active Power [W]': 0}
Dec 01 23:37:22 bbb-bcb1 RIAPS-DEPLO[15956]: [info]:[12-01-2019 23:37:22.571]:[15522]:ModbusTCPLogger.logger:RECEIVED {'Operating Time [h]': 582.54, 'Feed in time [h]': 542.39}
Dec 01 23:37:22 bbb-bcb1 RIAPS-DEPLO[15956]: [info]:[12-01-2019 23:37:22.752]:[15522]:ModbusTCPLogger.logger:RECEIVED {'AC Active Power [W]': 0, 'AC Active power phase L1 [W]': 0, 'AC Active power phase L2 [W]': 0, 'AC Active power phase L3 [W]': 0, 'AC Voltage phase L1 [V]': 0, 'AC Voltage phase L2 [V]': 0, 'AC Voltage phase L3 [V]': 0, 'AC Tot Grid Current [A]': 0, 'AC Grid Frequency [Hz]': 0, 'AC Reactive output Power [VAR]': 0, 'AC Reactive power phase L1 [VAR]': 0, 'AC Reactive power phase L2 [VAR]': 0, 'AC Reactive power phase L3 [VAR]': 0, 'AC Apparent power [VA]': 0, 'AC Apparent power phase L1 [VA]': 0, 'AC Apparent power phase L2 [VA]': 0, 'AC Apparent power phase L3 [VA]': 0}
Dec 01 23:37:22 bbb-bcb1 RIAPS-DEPLO[15956]: [info]:[12-01-2019 23:37:22.802]:[15522]:ModbusTCPLogger.logger:RECEIVED {'Plant mains connection : Separated'}
Dec 01 23:37:24 bbb-bcb1 RIAPS-DEPLO[15956]: [info]:[12-01-2019 23:37:24.246]:[15522]:ModbusTCPLogger.logger:RECEIVED {'Total yield [Kwh]': 1288.058, 'Daily yield [Wh]': 4951}
Dec 01 23:37:24 bbb-bcb1 RIAPS-DEPLO[15956]: [info]:[12-01-2019 23:37:24.396]:[15522]:ModbusTCPLogger.logger:RECEIVED {'DC input Voltage [V]': 0, 'DC input Current [mA]': 0, 'DC input Active Power [W]': 0}
Dec 01 23:37:24 bbb-bcb1 RIAPS-DEPLO[15956]: [info]:[12-01-2019 23:37:24.541]:[15522]:ModbusTCPLogger.logger:RECEIVED {'Operating Time [h]': 582.54, 'Feed in time [h]': 542.39}
```

Figure 6.2. Step by step configuration

During this phase many registers were double-checked using "QmodMaster" simulator and "Sunny explorer" platform as shown in Fig. 6.3.





```

Terminal - riaps@bbb-71ce: ~
File Edit View Terminal Tabs Help

[Info]:20:36:20,167:[1157]:USERAPP:Request to Read type <Connection type>
[Info]:20:36:20,399:[1157]:USERAPP:Converted values :::::LOADING::::: to the logger
[Info]:20:36:20,393:[1157]:USERAPP:Request to Read type <DC>
[Info]:20:36:20,439:[1157]:USERAPP:Converted values :::::LOADING::::: to the logger
[Info]:20:36:20,443:[1157]:USERAPP:Request to Read type <AC1>
[Info]:20:36:20,605:[1157]:USERAPP:Converted values :::::LOADING::::: to the logger
[Info]:20:36:20,609:[1157]:USERAPP:Request to Read type <AC2>
[Info]:20:36:20,744:[1157]:USERAPP:Converted values :::::LOADING::::: to the logger
[Info]:20:36:20,748:[1157]:USERAPP:Request to Read type <TTime>
[Info]:20:36:20,894:[1157]:USERAPP:Converted values :::::LOADING::::: to the logger
[Info]:20:36:20,898:[1157]:USERAPP:Request to Read type <Energy>
[Info]:20:36:21,048:[1157]:USERAPP:Converted values :::::LOADING::::: to the logger
[Info]:20:36:21,052:[1157]:USERAPP:Request to Read type <GridGuardCode>
[Info]:20:36:21,199:[1157]:USERAPP:Converted values :::::LOADING::::: to the logger
[Info]:20:36:21,203:[1157]:USERAPP:Request to Write type <GridGuardCode> -Value already written-
[Info]:20:36:21,204:[1157]:USERAPP:Request to Read type <Language>
[Info]:20:36:21,349:[1157]:USERAPP:Converted values :::::LOADING::::: to the logger
[Info]:20:36:21,353:[1157]:USERAPP:Request to Write type <Language> -Value already written-
[Info]:20:36:21,355:[1157]:USERAPP:Request to Read type <Operating mode of multi-function relay>
[Info]:20:36:21,499:[1157]:USERAPP:Converted values :::::LOADING::::: to the logger
[Info]:20:36:21,503:[1157]:USERAPP:Request to Write type <Operating mode of multi-function relay> -Value already written-
[Info]:20:36:21,644:[1157]:USERAPP:Converted values :::::LOADING::::: to the logger
[Info]:20:36:21,647:[1157]:USERAPP:Request to Write type <Active power Limit> -Value already written-
[Info]:20:36:21,649:[1157]:USERAPP:Request to Read type <On/Off>
[Info]:20:36:21,795:[1157]:USERAPP:Converted values :::::LOADING::::: to the logger
[Info]:20:36:21,799:[1157]:USERAPP:Request to Write type <On/Off> -Value already written-

Dec 14 20:36:20 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:20,302:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'Plant mains connection ':'Public electricity mains'
Dec 14 20:36:20 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:20,442:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'DC Power [W]':727
Dec 14 20:36:20 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:20,444:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'DC Voltage [V]':246.02
Dec 14 20:36:20 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:20,446:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'DC Current [A]':2.11
Dec 14 20:36:20 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:20,608:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'AC Grid Frequency [Hz]':59.98
Dec 14 20:36:20 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:20,613:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'AC Active Power [W]':733
Dec 14 20:36:20 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:20,615:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'AC Active Power phase L1 [W]':246
Dec 14 20:36:20 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:20,618:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'AC Active Power phase L2 [W]':240
Dec 14 20:36:20 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:20,618:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'AC Active Power phase L3 [W]':247
Dec 14 20:36:20 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:20,618:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'AC Reactive Power [VAR]':135
Dec 14 20:36:20 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:20,618:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'AC Reactive Power phase L1 [VAR]':47
Dec 14 20:36:20 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:20,618:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'AC Reactive Power phase L2 [VAR]':38
Dec 14 20:36:20 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:20,618:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'AC Reactive Power phase L3 [VAR]':48
Dec 14 20:36:20 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:20,618:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'AC Apparent Power [VA]':745
Dec 14 20:36:20 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:20,618:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'AC Apparent Power phase L1 [VA]':250
Dec 14 20:36:20 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:20,618:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'AC Apparent Power phase L2 [VA]':243
Dec 14 20:36:20 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:20,618:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'AC Apparent Power phase L3 [VA]':252
Dec 14 20:36:20 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:20,619:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'AC Voltage phase L1 [V]':291.32
Dec 14 20:36:20 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:20,619:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'AC Voltage phase L2 [V]':292.07
Dec 14 20:36:20 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:20,619:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'AC Voltage phase L3 [V]':292.26
Dec 14 20:36:20 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:20,619:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'AC Grid Current [A]':2.56
Dec 14 20:36:20 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:20,747:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'AC Current phase L1 [A]':0
Dec 14 20:36:20 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:20,747:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'AC Current phase L2 [A]':0
Dec 14 20:36:20 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:20,747:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'AC Current phase L3 [A]':0
Dec 14 20:36:20 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:20,897:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'Operating Time [h]':702.34
Dec 14 20:36:20 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:20,897:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'Feed in time [h]':654.28
Dec 14 20:36:21 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:21,050:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'Daily yield [MWh]':5.04
Dec 14 20:36:21 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:21,051:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'Total yield [MWh]':1.504843
Dec 14 20:36:21 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:21,201:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'Grid Guard Code ':'Valid'
Dec 14 20:36:21 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:21,351:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'Language ':'English'
Dec 14 20:36:21 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:21,503:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'Multi-function relay mode ':'Control via communication'
Dec 14 20:36:21 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:21,647:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'Active power limitation in [W]':20000
Dec 14 20:36:21 bbb-8cb1 RIAPS-DEPLO176281: [Info]:20:36:21,797:[18034]:LOGGER:Received on rx_modbusTCPData() [18034]: 'Fast Start/Shut-down set to ':'Start'

```

Figure 6.5. Final implementation on RIAPS platform

It is also shown in Fig.6.6 the typical output from "SunnyExplorer" of a sunny day shape on a chart where experiments are performed.

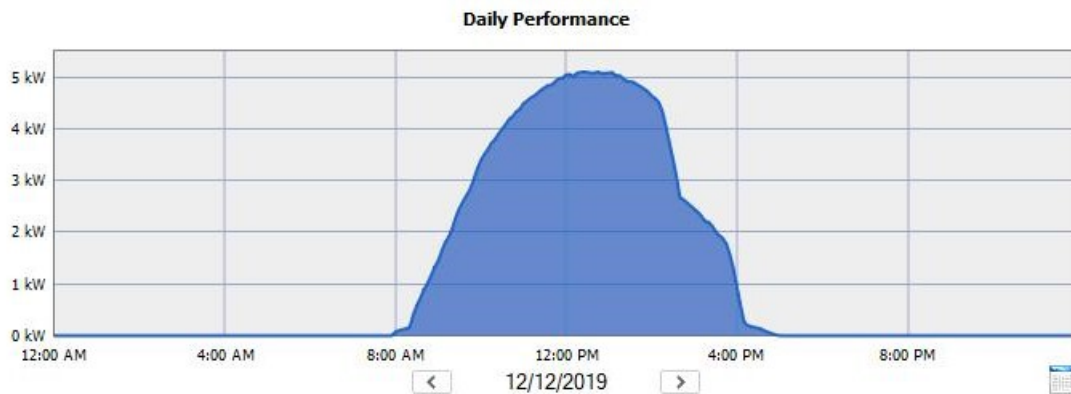


Figure 6.6. Sunny Day from "SunnyExplorer" platform

## 6.2 Power limit simulation test

The Power limitation test is considered together with the Shut-down test one of most the critical test manually performed. Several test have been performed in order to be sure the power limitation was performed correctly.

The first test shown is related to a power limitation of "300[W]". In Fig. 6.7 is possible to see that, from the RIAPS platform, the limit is set and the DC values and AC values are approaching that limit. The important values shown are highlighted with red arrows to enhance the comprehension. On the upper part it is shown the logging of the operation performed, while in the bottom part, the logger shows only the logging of the data value inside the registers. The same response, is shown in

```

Terminal - riaps@bbb-71c: ~
[info]:17:20:14,581:[9346]:USERAPP:On clock():[9346]: 1576344014.5806901 ..... UPDATING OPERATIONS .....
[info]:17:20:14,583:[9346]:USERAPP:Request to Read type <Connection type>
[info]:17:20:14,656:[9346]:USERAPP:Converted values .....LOADING..... to the logger
[info]:17:20:14,659:[9346]:USERAPP:Request to Read type <DC>
[info]:17:20:14,806:[9346]:USERAPP:Converted values .....LOADING..... to the logger
[info]:17:20:14,810:[9346]:USERAPP:Request to Read type <AC>
[info]:17:20:14,993:[9346]:USERAPP:Converted values .....LOADING..... to the logger
[info]:17:20:14,996:[9346]:USERAPP:Request to Read type <AC2>
[info]:17:20:15,131:[9346]:USERAPP:Converted values .....LOADING..... to the logger
[info]:17:20:15,134:[9346]:USERAPP:Request to Read type <Time>
[info]:17:20:15,281:[9346]:USERAPP:Converted values .....LOADING..... to the logger
[info]:17:20:15,284:[9346]:USERAPP:Request to Read type <Energy>
[info]:17:20:15,431:[9346]:USERAPP:Converted values .....LOADING..... to the logger
[info]:17:20:15,434:[9346]:USERAPP:Request to Write type <GridGuardCode>
[info]:17:20:15,436:[9346]:USERAPP:Requested to Write type <GridGuardCode> -Value already written-
[info]:17:20:15,437:[9346]:USERAPP:Request to Read type <GridGuardCode>
[info]:17:20:15,581:[9346]:USERAPP:Converted values .....LOADING..... to the logger
[info]:17:20:15,584:[9346]:USERAPP:Requested to Write type <Language> -Value already written-
[info]:17:20:15,586:[9346]:USERAPP:Request to Read type <Language>
[info]:17:20:15,731:[9346]:USERAPP:Converted values .....LOADING..... to the logger
[info]:17:20:15,735:[9346]:USERAPP:Requested to Write type <Operating mode of multi-function relay> -Value already written-
[info]:17:20:15,736:[9346]:USERAPP:Request to Read type <Operating mode of multi-function relay>
[info]:17:20:15,907:[9346]:USERAPP:Converted values .....LOADING..... to the logger
[info]:17:20:15,890:[9346]:USERAPP:Requested to Write type <Active power Limit> -Value already written-
[info]:17:20:15,891:[9346]:USERAPP:Request to Read type <Active power Limit>
[info]:17:20:16,031:[9346]:USERAPP:Converted values .....LOADING..... to the logger
[info]:17:20:16,035:[9346]:USERAPP:Request to Write type <On/Off>
[info]:17:20:16,036:[9346]:USERAPP:Request to Read type <On/Off>
[info]:17:20:16,182:[9346]:USERAPP:Converted values .....LOADING..... to the logger

Terminal - riaps@bbb-8cb1: ~
Dec 14 17:21:00 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:00,880:[16291]:LOGGER:On clock():[16291]: 1576344060.8792768 .....UPDATED VALUES INCOMING.....
Dec 14 17:21:02 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:02,676:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'Plant mains connection ':'Public electricity mains'
Dec 14 17:21:02 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:02,831:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'DC Power [W]':353
Dec 14 17:21:02 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:02,831:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'DC Voltage [V]':411.1
Dec 14 17:21:02 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:02,831:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'DC Current [A]':0.86
Dec 14 17:21:03 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:03,007:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'AC Grid Frequency [Hz]':59.99
Dec 14 17:21:03 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:03,007:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'AC Active Power [W]':332
Dec 14 17:21:03 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:03,007:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'AC Active Power phase L1 [W]':111
Dec 14 17:21:03 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:03,007:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'AC Active Power phase L2 [W]':109
Dec 14 17:21:03 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:03,008:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'AC Active Power phase L3 [W]':112
Dec 14 17:21:03 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:03,008:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'AC Reactive Power [Var]':-121
Dec 14 17:21:03 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:03,008:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'AC Reactive Power phase L1 [Var]':-57
Dec 14 17:21:03 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:03,008:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'AC Reactive Power phase L2 [Var]':-51
Dec 14 17:21:03 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:03,008:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'AC Reactive Power phase L3 [Var]':-61
Dec 14 17:21:03 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:03,008:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'AC Apparent Power [VA]':373
Dec 14 17:21:03 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:03,009:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'AC Apparent Power phase L1 [VA]':125
Dec 14 17:21:03 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:03,009:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'AC Apparent Power phase L2 [VA]':121
Dec 14 17:21:03 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:03,010:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'AC Apparent Power phase L3 [VA]':127
Dec 14 17:21:03 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:03,010:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'AC Voltage phase L1 [V]':291.87
Dec 14 17:21:03 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:03,010:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'AC Voltage phase L2 [V]':292.66
Dec 14 17:21:03 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:03,010:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'AC Voltage phase L3 [V]':292.69
Dec 14 17:21:03 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:03,010:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'AC Grid Current [A]':1.28
Dec 14 17:21:03 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:03,146:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'AC Current phase L1 [A]':0
Dec 14 17:21:03 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:03,149:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'AC Current phase L2 [A]':0
Dec 14 17:21:03 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:03,150:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'AC Current phase L3 [A]':0
Dec 14 17:21:03 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:03,291:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'Operating Time [h]':699.09
Dec 14 17:21:03 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:03,299:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'Feed in time [h]':651.03
Dec 14 17:21:03 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:03,447:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'Daily yield [Wh]':3923
Dec 14 17:21:03 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:03,449:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'Total yield [MWh]':1.503362
Dec 14 17:21:03 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:03,591:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'Grid Guard Code ':'Valid'
Dec 14 17:21:03 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:03,746:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'Language ':'English'
Dec 14 17:21:03 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:03,896:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'Multi-function relay mode ':'Control via communication'
Dec 14 17:21:04 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:04,041:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'Active power limitation in [W]':300
Dec 14 17:21:04 bbb-8cb1 RIAPS-DEPLO[16105]: [info]:17:21:04,191:[16291]:LOGGER:Received on rx_modbusTCPData():[16291]: 'Fast Start/Shut-down set to ':'Start'

```

Figure 6.7. Active power limit response from "Riaps"

Fig. 6.8 from "Sunny Explorer" platform. The values are not perfectly coinciding with the the one shown previously in Fig. 6.7 because the screenshots are taken in

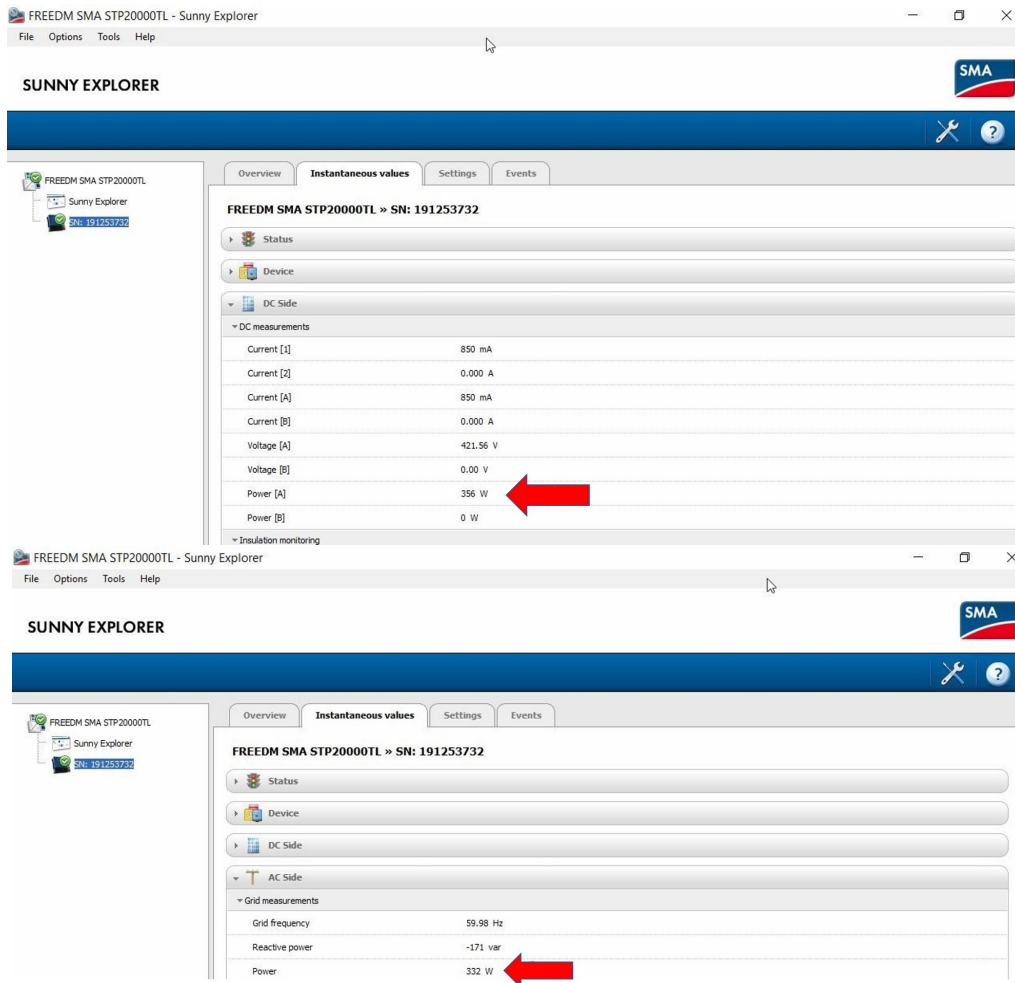


Figure 6.8. Active power limit response from "Sunny Explorer"

different instances and the values are updating fast. In Fig. 6.9, are shown some events regarding the power limitation test, the limit is set and reset at a certain time of the day in order to reach a picture as clear as possible explaining the effectiveness of the test. Actually the time values referring to the Fig. 6.9 are those reflecting the graph on Fig. 6.11 analyzed below.



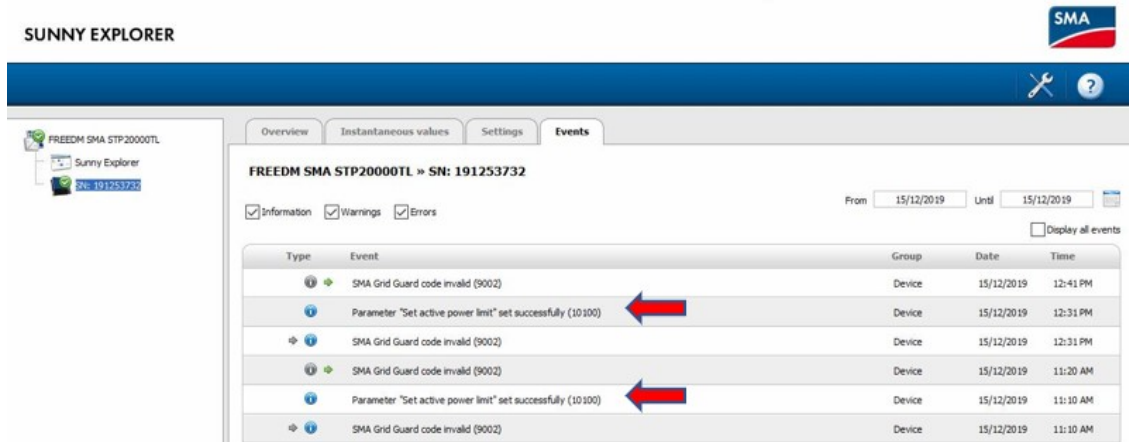


Figure 6.9. Active power limit from "SunnyExplorer" events

In Fig. 6.10 it is possible to see the graph representing the power limitation of "300[W]" from "SunnyExplorer" platform. This limitation test has been performed during a not perfectly sunny day so that the oscillation of the energy acquired is not perfect. The graph intention is only to show the functionality of the test simulation even in a cloudy day, where it is shown at least two hours of power limitation.

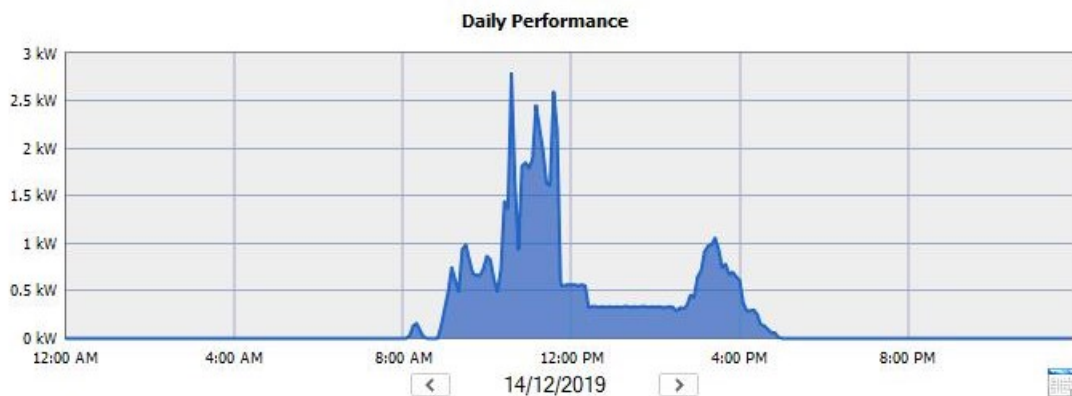


Figure 6.10. Graphical view from "SunnyExplorer" Limit 300[W]"

Several tests has been made, in particular the exact test type is repeated in order to limit to an higher value of "2000 [W]" shown in Fig. 6.11. This graph is a little different with respect to the previous one, because the experiment has been done in a different day, so the energy acquired is different, and consequently also the shape. In particular in this graph, the limitation of power is more clear because it is performed during a sunny day, where in the slot of time from 11:10 am to 12:30

am, the power has been limited to the previous mentioned value of "2000 [W]".

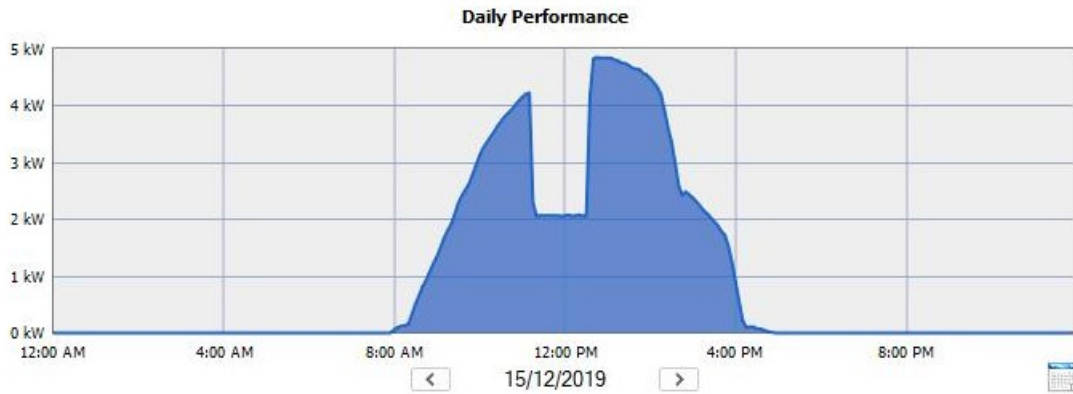


Figure 6.11. Graphical view from "SunnyExplorer" limit "2000 [W]"

### 6.3 Shut-down simulation test

The shut-down operation test is also performed and as previously defined, constitutes together with the "Active power limitation", one of the critical test. Nevertheless, this is considered the most critical, because usually this operation is not repeated cyclically during its normal operating conditions, and it is automatically handled by the main controller. Part of the simulation results, from RIAPS side, are shown in Fig. 6.12.

```

Terminal - riaps@bbb-8cb1:~
[info]:16:26:19,873:[4371]:LOGGER:On clock() [4371]: 1576513579.872044 .....UPDATED VALUES INCOMING:.....
[info]:16:26:20,895:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'Plant mains connection :Public electricity mains'
[info]:16:26:21,045:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'DC Power [W]:4319'
[info]:16:26:21,047:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'DC Voltage [V]:217.57'
[info]:16:26:21,049:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'DC Current [A]:13.61'
[info]:16:26:21,211:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Grid Frequency [Hz]:59.99'
[info]:16:26:21,216:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Active Power [W]:4288'
[info]:16:26:21,218:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Active Power phase L1 [W]:1403'
[info]:16:26:21,218:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Active Power phase L2 [W]:1404'
[info]:16:26:21,218:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Active Power phase L3 [W]:1401'
[info]:16:26:21,219:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Reactive Power [Var]:-4288'
[info]:16:26:21,219:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Reactive Power phase L1 [Var]:-5'
[info]:16:26:21,219:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Reactive Power phase L2 [Var]:-5'
[info]:16:26:21,219:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Reactive Power phase L3 [Var]:-4'
[info]:16:26:21,219:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Apparent Power phase L1 [VA]:1403'
[info]:16:26:21,219:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Apparent Power phase L2 [VA]:1404'
[info]:16:26:21,219:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Apparent Power phase L3 [VA]:1401'
[info]:16:26:21,219:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Voltage phase L1 [V]:289.73'
[info]:16:26:21,219:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Voltage phase L2 [V]:289.05'
[info]:16:26:21,219:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Voltage phase L3 [V]:290.03'
[info]:16:26:21,219:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Grid Current [A]:14.52'
[info]:16:26:21,355:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Current phase L1 [A]:0'
[info]:16:26:21,355:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Current phase L2 [A]:0'
[info]:16:26:21,355:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Current phase L3 [A]:0'
[info]:16:26:21,507:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'Operating Time [h]:717.28'
[info]:16:26:21,508:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'Feed in time [h]:668.27'
[info]:16:26:21,655:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'Daily yield [Wh]:8421'
[info]:16:26:21,657:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'Total yield [MWh]:1.536089'
[info]:16:26:21,805:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'Grid Guard Code :Valid'
[info]:16:26:21,955:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'Grid Guard Code just sent :':Successfully'
[info]:16:26:22,110:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'Language :English'
[info]:16:26:22,260:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'Multi-function relay mode :':Control via communication'
[info]:16:26:22,410:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'Active power limitation in [W] :20000'
[info]:16:26:22,560:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'Fast Start/Shut-down set to :':Start'
[info]:16:26:22,709:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'Change Start/Shut-down Command just sent :':Successfully'

Terminal - riaps@bbb-8cb1:~
[info]:16:26:35,875:[4371]:LOGGER:On clock() [4371]: 1576513595.8742895 .....UPDATED VALUES INCOMING:.....
[info]:16:26:37,020:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'Plant mains connection :Separated'
[info]:16:26:37,170:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'DC Power [W]:0'
[info]:16:26:37,173:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'DC Voltage [V]:401.62'
[info]:16:26:37,327:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'DC Current [A]:0.0'
[info]:16:26:37,342:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Grid Frequency [Hz]:0'
[info]:16:26:37,344:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Active Power [W]:0'
[info]:16:26:37,346:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Active Power phase L1 [W]:0'
[info]:16:26:37,346:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Active Power phase L2 [W]:0'
[info]:16:26:37,346:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Active Power phase L3 [W]:0'
[info]:16:26:37,346:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Reactive Power [Var]:0'
[info]:16:26:37,346:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Reactive Power phase L1 [Var]:0'
[info]:16:26:37,346:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Reactive Power phase L2 [Var]:0'
[info]:16:26:37,346:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Reactive Power phase L3 [Var]:0'
[info]:16:26:37,346:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Apparent Power [VA]:0'
[info]:16:26:37,346:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Apparent Power phase L1 [VA]:0'
[info]:16:26:37,346:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Apparent Power phase L2 [VA]:0'
[info]:16:26:37,346:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Apparent Power phase L3 [VA]:0'
[info]:16:26:37,347:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Voltage phase L1 [V]:0'
[info]:16:26:37,347:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Voltage phase L2 [V]:0'
[info]:16:26:37,347:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Voltage phase L3 [V]:0'
[info]:16:26:37,347:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Grid Current [A]:0'
[info]:16:26:37,475:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Current phase L1 [A]:0'
[info]:16:26:37,475:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Current phase L2 [A]:0'
[info]:16:26:37,475:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'AC Current phase L3 [A]:0'
[info]:16:26:37,631:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'Operating Time [h]:717.28'
[info]:16:26:37,642:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'Feed in time [h]:668.27'
[info]:16:26:37,775:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'Daily yield [Wh]:8427'
[info]:16:26:37,775:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'Total yield [MWh]:1.536089'
[info]:16:26:37,930:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'Grid Guard Code :Valid'
[info]:16:26:38,088:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'Language :English'
[info]:16:26:38,234:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'Multi-function relay mode :':Control via communication'
[info]:16:26:38,383:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'Active power limitation in [W] :20000'
[info]:16:26:38,526:[4371]:LOGGER:Received on rx.modbusTCPData() [4371]: 'Fast Start/Shut-down set to :':Full-stop AC+DC side'

```

Figure 6.12. Shut-down from Riaps platform

In Fig. 6.13 it is possible to see that the previous operation is correctly performed and the command is received and posted through the "SunnyExplorer platform" that shows a quick shut-down operation received first and then a quick Start intended as restart of the AC+DC side connections to the lines. Moreover from "SunnyExplorer" is possible also to see that the operation has taken really effect demonstrating with a graph in Fig. 6.14 the response from the inverter side during a normal day of sun. Nevertheless, several shut-down tests have been performed during the development phase, but in order to show the correct execution is taken a sample where in one day only one shut-down is performed, with the intention to demonstrate the command execution from a simple point of view by RIAPS platform.

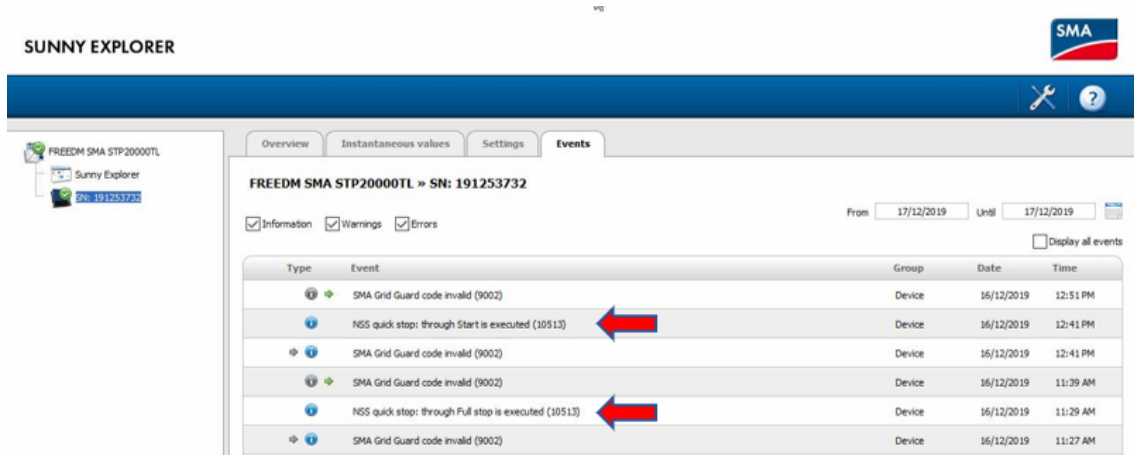


Figure 6.13. Shut-down from "SunnyExplorer" events

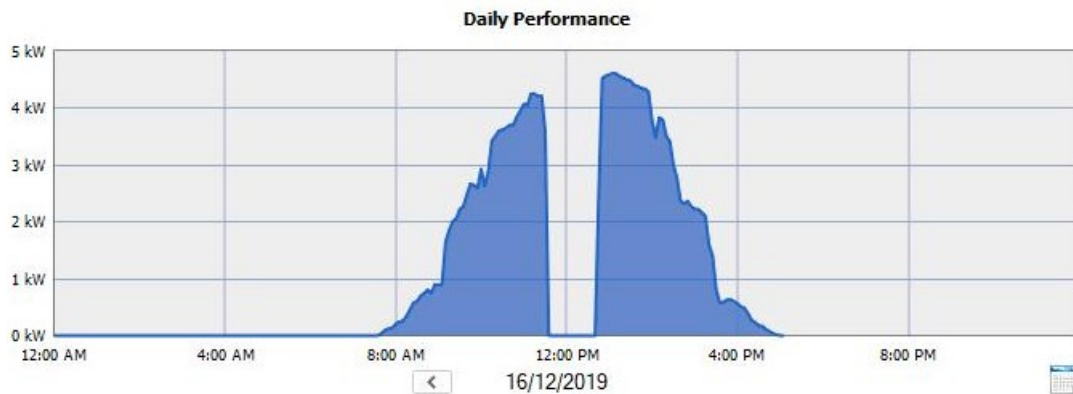


Figure 6.14. Graphical view from "SunnyExplorer" Shut-down

## 6.4 Critical review

Although, the register related to the "Operating mode of multi-functional relay" has been tested at the software level, there is no prove of effective operating. This is mainly because no real hardware has been connected physically to the inverter during this this test, since it was not the main topic of this research. Since it is already implemented and ready to be tested, further investigations could be done in order to understand the different hardware that is possible to interconnect and how do they interfere with the main controller of the inverter.

The power is not limited perfectly to the designated value. An example is provided in the power limitation test when it is set a limit of "300[W]". The actual limit

from "Sunny Explorer" platform shows that the AC and DC power limit are around the value of "330[W]". This is not clear, but maybe the limitation is performed in different parameters that influences the power at the end. For the future projects, this conclusion might be not sufficient for individual applications, and in order to reach better results, this require further investigations.

According with the code developed, data logging is performed at terminal level, this means that any data is available after the lines of the terminal scrolls down and overwriting operation is performed. This is also possible to be done through the use of some tools on the platform in order to produce a file able to retain the data collected during the simulation. This require possible study on this very specific topic that could be fundamental in some future applications, where for example data monitoring is performed in a second moment or in general to have a record of past values for a different purpose.

One more critical review should be done to the fact that the connected device are restricted to only one inverter. In this way, not all the benefits related to the use of the "Riaps" platform could be exploited. For this reason, the self-management on the basis of multiple data coming from different inverters are not performed. Furthermore possible problems related to the implementation of the developed code running on different devices and connecting each other were not faced.

In this context, although this project research led to important results from simulations and data analysis, this might be not the final configuration and at the end it will require an overall step further to be implemented basically in the real world applications.



# Chapter 7

## Conclusion and Future work

In this chapter, it is also important to evaluate the outcomes of the simulations consequences in order to conclude with a discussion of some acquired certainties obtained for the study of future world applications. Last step is related to the analysis of the possible future works and different implementations with some small modifications.

### 7.1 Conclusion

As a witness of the implementation in the real world application, could be said that this project represent the development of one of the first application developed using the revolutionary RIAPS software platform. This is almost ready to be used in a Smart Grid environment to provide a significantly step forward for the worldwide appliance, so that it constitutes a project not only related to the university field but also for the power institutions dealing with this products.

As previously announced, all the tests performed, are intended to demonstrate the effectiveness of the platform management with the intended code developed. All the tests and results accomplish their task and after decoding hardly some meticulous details, important data results demonstrate that every single component match with the other. The important registers selected are included in the code and the knowledge of the operations they perform has been acquired during the simulation phases. This project aim is also to demonstrate the effectiveness of the RIAPS platform to interface with external device, in particular with a commercial device, developing an application respecting the TCP/IP standards and accomplishing some tasks related to the communication itself. Moreover the custom application developed act as higher level controller that is operating in a way to manage operations, receiving data and sending commands from/to the embedded standard controller. In this occasion, RIAPS, provide not only a customized control structure but also offer all the benefit of the included in this revolutionary software platform such as: reliability,

security, simplicity on control and monitoring system, hosting multitasking operations, looking for time synchronization with simple communications and leading to the distribution of intelligence for "IoT" application. The importance of plug and play operation is also not trivial as soon as a device is connected and the application deployed works correctly, providing a scalable solution in the real world applications. Another important feature analyzed and here highlighted is the failure detection of the devices connected, that the platform is able to handle without letting crash the entire system.

The application developed, moreover, is intended to be used as a basis on which to develop other smart applications where some hints are shown in the next section. Only few modification are needed in order to reach a different configuration, because the application is already configured to be flexible in order to accommodate multiple tasks.

In conclusion the application is providing security and privacy of data, because it rely on local network area able to share information exclusively with all the nodes connected to the same access point. This connection is not relying anymore on the NCSU (North Carolina State University) network as it was previously implemented with the standard "SunnyExplorer" software platform solution.



## 7.2 Future work

A brief analysis of the future work that could lead to a bigger implementation of the application is discussed hereby. Regardless, multiple small modification could be done either on the hardware part and in the software part.

Starting from the hardware re-configuration, multiple inverter could be connected to the data transmission, deploying the same application gathering data from different nodes.

Moreover, a wireless implementation could be taken in consideration, using a smart WiFi modules attached to the BBBs. This could prevent from hard wiring harnesses and could be a solution to distribute the communication in a strategical geographical area, embedding the smart system inside the same device box and possibly powered by the same. A possibility is to use basic a step-down converter properly designed in order to convert to 5V - 500 mA to feed the controller.

One more step that is possible to implement is the one regarding the communication with other devices of different vendors type, this is important as soon as the inverter is able to supply different loads and send receive data with this last one even if they are not coming from the same production line. Some other small modification could be done in order to communicate with the "GEH" (Green Energy Hub) test-bed, present in the lab of FREEDM system center, that already host a RIAPS based implemented application where its working principle similarly reflect the one of a real Smart Grid System.

On the other hand, the software modification could lead to implement the functionality to host multiple devices as a consequence of the hardware modification. Another simple modification, could be to interact with a bigger quantity of register involved either to read and to write. As a consequence of the multiple devices implementation, it is possible to integrate it with the rest of the communication pattern already configured, in order to possibly host a big green hub smart energy system management.



# Appendix A

## Code for Software in the loop simulation

```
// RIAPS Modbus (TCP) Device Testing

app RIAPSModbusTCP {
  message ModbusCommand; // send Modbus action Request
  message ModbusResponse; // get response from Modbus action
  message NodeData;

  library tcpModbusLib;

  // Modbus (TCP) device interface
  // considered the server for the request/response interaction
  device ModbusTcpReqRepDevice(slaveaddress=31,ipaddress = '192.168.10.101',port=1504,serialtimeout=1.0) {
    rep modbusRepPort : (ModbusCommand,ModbusResponse);
    timer clock 1000; //life signal
  }

  // Example Component to show Modbus I/F usage
  component ComputationalComponent(Ts = 0.1, ip = 111) {
    timer clock 500;
    sub nodeReady : NodeData;
    pub thisReady : NodeData;
    req modbusReqPort : (ModbusCommand,ModbusResponse); // Port used to communicate with the ModbusUartDevice
  }

  // Modbus Communication Example actor
  actor ModbusExampleActor(TsArg, ipArg) {
    local ModbusCommand,ModbusResponse; // Local message types
    {
      modbus : ModbusTcpReqRepDevice(slaveaddress=31,ipaddress = '192.168.10.101',port=1504,serialtimeout=1.0) ; // Slave Address is in decimal
      commInitiator : ComputationalComponent( Ts = TsArg, ip = ipArg );
    }
  }
}
```

Figure A.1. File "modbus\_tcp\_core.riaps"

```

'''
Created on Nov 25, 2014

Now only three function codes are supported. They are,

    ModbusCommands.READMULTI_INPUTREGS
    ModbusCommands.READMULTI_HOLDINGREGS
    ModbusCommands.WRITEMULTI_HOLDINGREGS

'''

import zmq
from riaps.run.comp import Component
from riaps.run.exc import PortError
import uuid
import os
import time
from collections import namedtuple
from ModbusTcpRegRepDevice import CommandFormat, ModbusCommands
import pydevd
import logging

''' Enable debugging to gather timing information on the code execution'''
debugMode = False

RegSet = namedtuple('RegSet', ['idx', 'value'])
InputRegs = namedtuple('InputRegs', ['data1', 'data2', 'data3', 'data4'])

'''For Inverter control'''
HoldingRegs = namedtuple('HoldingRegs', ['unused', 'startStopCmd', 'power', 'reactive_power'])
'''For RMR future
1.start/stop, 2.power command, 3. frequency shift from secondary control, 4. voltage magnitude shift from secondary control.
HoldingRegs = namedtuple('HoldingRegs', ['startStopCmd', 'powerCmd', 'freqShift', 'voltMagShift'])
'''

class ComputationalComponent(Component):
    def __init__(self, Ts, ip):
        super().__init__(Ts, ip)
        pydevd.settrace(host='192.168.1.103', port=5678)
        self.uuid = uuid.uuid4().int
        self.pid = os.getpid()
        self.inputRegs = InputRegs(RegSet(0,0), RegSet(1,0), RegSet(2,0), RegSet(3,0))
        self.holdingRegs = HoldingRegs(RegSet(0,0), RegSet(1,0), RegSet(2,0), RegSet(3,0))
        self.ip = ip

        '''Setup Commands - for modbus'''
        self.readWrite = 0
        self.defaultInitialReg = 20
        self.defaultNumOfRegs = 5
        self.dummyValue = 0
        self.defaultNumOfDecimals = 0
        self.signedDefault = False

        self.logger.info("ComputationalComponent: %s - starting" % str(self.pid))

    def on_clock(self):
        on_clock_start = time.time()
        now = self.clock.recv_pyobj()

        if debugMode:
            self.logger.info("on_clock()[%s]: %s", str(self.pid), str(now))

        '''Request: Commands to send over Modbus - one command used at a time'''

        '''Read multiple holding registers'''
        #self.command = CommandFormat(ModbusCommands.READMULTI_INPUTREGS, 0, numRegsToRead, self.dummyValue)
        self.command = CommandFormat(ModbusCommands.READMULTI_HOLDINGREGS, 0, 12, self.dummyValue, self.signedDefault)
        msg = self.command

        repMsg = self.sendModbusRequest(self.command)
        # Check if the Modbus Response is valid
        #if repMsg != -1 and repMsg != 999 and repMsg != -9999:
        if repMsg != None:
            on_clock_modbus_measurement_end_and_algorithm_start = time.time()
            print(repMsg)
        else:
            self.logger.info("Modbus either not ready or no value read")

        '''Write multiple holding registers'''
        self.values = [4000, 3000, 3276, 1000, 2000]
        self.command = CommandFormat(ModbusCommands.WRITEMULTI_HOLDINGREGS, self.defaultInitialReg, self.defaultNumOfRegs, self.values, self.signedDefault)

        repMsg = self.sendModbusRequest(self.command)
        # Check if the Modbus Response is valid
        #if repMsg == -1 or repMsg == -9999:
        if repMsg == None:
            self.logger.info("Modbus either not ready")

    def sendModbusRequest(self, requestMsg):
        # Init to invalid response value
        rep = None
        try:
            self.modbusReqPort.send_pyobj(requestMsg)
            rep = self.modbusReqPort.recv_pyobj()
        except PortError as e:
            self.logger.info("on_clock: send exception = %d" % e.errno)
            if e.errno in (PortError.EAGAIN, PortError.EPROTO):
                self.logger.info("on_clock: port error received")
            #self.logger.info("[%d] rcv rep: %s" % (self.pid, rep))
        return rep

    def __destroy__(self):
        self.logger.info("[%d] destroyed" % self.pid)

```

Figure A.2. ComputationalComponent.py

```

'''
Created on Nov 11, 2014

This module utilizes the umodbus.
Both need to be installed in the development environment.
... $ sudo pip3 install umodbus
...

from riaps.run.comp import Component
import logging
import os
from collections import namedtuple
from enum import Enum
import time
from tcpModbusLib.tcpModbusComm import TcpModbusComm, PortConfig

''' Enable debugging to gather timing information on the code execution'''
debugMode = False

class ModbusCommands(Enum):
    READ_BIT = 1
    READ_INPUTREG = 2
    READ_HOLDINGREG = 3
    READMULTI_INPUTREGS = 4
    READMULTI_HOLDINGREGS = 5
    WRITE_BIT = 6
    WRITE_HOLDINGREG = 7
    WRITEMULTI_HOLDINGREGS = 8

CommandFormat = namedtuple('CommandFormat', ['commandType', 'registerAddress', 'numberOfRegs', 'values', 'signedValue'])

class ModbusTcpReqRepDevice(Component):
    def __init__(self, slaveAddress=31, ipAddress = '192.168.10.101', port = 1504, serialTimeout=0.05): # defaults for Modbus spec
        super().__init__()

        self.pid = os.getpid()
        self.port_config = PortConfig(ipAddress, port, serialTimeout)
        self.modbus = TcpModbusComm(self, slaveAddress, self.port_config)
        self.modbusInit = False
        if debugMode:
            self.logger.info("Modbus settings %d %s: %d %d %d", self.slaveAddress, self.port_config.ip, self.port_config.port, self.port_config.timeout, self.pid)

    def on_clock(self):
        now = self.clock.recv_pyobj() # Receive time (as float)
        self.logger.info("on_clock() [%s]: %s" % (str(self.pid), now))

        if debugMode:
            t0 = time.perf_counter()
            self.logger.debug("on_clock() [%s]: Request Modbus start at %f", str(self.pid), t0)

        if self.modbusInit == False:
            self.modbusInit = True
            self.modbus.startModbus()
            pydevd.settrace(host='192.168.1.102', port=5678)

            if debugMode:
                t1 = time.perf_counter()
                self.logger.debug("on_clock() [%s]: Modbus ready at %f, time to start Modbus is %f ms", str(self.pid), t1, (t1-t0)*1000)

        self.clock.halt()

    def __destroy__(self):
        self.logger.info("__destroy__")
        self.modbus.stopModbus()

    '''
    Receive a Modbus command request. Process command and send back response.
    '''
    def on_modbusRepPort(self):
        '''Request Received'''
        commandRequest = self.modbusRepPort.recv_pyobj()

        if debugMode:
            self.modbusReqRxTime = time.perf_counter()
            self.logger.debug("on_modbusRepPort() [%s]: Request=%s Received at %f", str(self.pid), commandRequest, self.modbusReqRxTime)

        self.unpackCommand(commandRequest)
        responseValue = -1 # invalid response
        if self.modbus.isModbusAvailable() == True:
            responseValue = self.sendModbusCommand()

        if debugMode:
            t1 = time.perf_counter()
            self.logger.debug("on_modbusRepPort() [%s]: Send Modbus response=%s back to requester at %f", str(self.pid), responseValue, t1)

```

Figure A.3. ModbusTcpReqRepDevice.py

```

#
pydevd.settrace(host='192.168.1.102',port=5678)
else:
    self.logger.debug("Modbus is not available")

'''Send Results'''
self.modbusRepPort.send_pyobj(responseValue)

def unpackCommand(self,rxCommand):
    self.commandRequested = rxCommand.commandType
    self.registerAddress = rxCommand.registerAddress
    self.numberOfRegs = rxCommand.numberOfRegs
    self.values = rxCommand.values
    self.signedValue = rxCommand.signedValue

def sendModbusCommand(self):
    value = 999 # large invalid value

    if debugMode:
        t0 = time.perf_counter()
        self.logger.debug("sendModbusCommand()[%s]: Sending command to Modbus library at %f",str(self.pid),t0)

    if self.commandRequested == ModbusCommands.READMULTI_INPUTREGS:
        value = self.modbus.readMultiInputRegValues(self.registerAddress, self.numberOfRegs, self.signedValue)
        #self.logger.info("ModbusUartDevice: sent command %s, register=%d, numofRegs=%d", ModbusCommands.READMULTI_INPUTREGS.name,self.registerAddress,self.numberOfRegs)
    elif self.commandRequested == ModbusCommands.READMULTI_HOLDINGREGS:
        value = self.modbus.readMultiHoldingRegValues(self.registerAddress, self.numberOfRegs, self.signedValue)
        #self.logger.info("ModbusUartDevice: sent command %s, register=%d, numofRegs=%d", ModbusCommands.READMULTI_HOLDINGREGS.name,self.registerAddress,self.numberOfRegs)
    elif self.commandRequested == ModbusCommands.WRITEMULTI_HOLDINGREGS:
        self.modbus.writeHoldingRegisters(self.registerAddress, self.values, self.signedValue)
        #self.logger.info("ModbusUartDevice: sent command %s, register=%d",ModbusCommands.WRITEMULTI_HOLDINGREGS.name,self.registerAddress)
        #self.logger.info("ModbusUartDevice: Values - %s", str(self.values).strip('[]'))

    if debugMode:
        t1 = time.perf_counter()
        self.logger.debug("sendModbusCommand()[%s]: Modbus library command complete at %f, time to interact with Modbus library is %f ms",str(self.pid),t1,(t1-t0)*1000)

    return value

```

Figure A.4. ModbusTcpReqRepDevice.py

```

'''
Created on Nov 19, 2019

This modbus device interface will read input registers and reads/writes holding registers of slave devices with modbus tcp.
At this time, it does not read/write coils.
'''

#!/usr/bin/env python
import socket

from umodbus import conf
from umodbus.client import tcp

from collections import namedtuple
import sys
from enum import Enum, unique

# import pydevd

'''serialTimeout is defined in seconds'''
PortConfig = namedtuple('PortConfig', ['ip', 'port', 'timeout'])

'''
Function Codes (per Modbus Spec)
'''
@unique
class FunctionCodes(Enum):
    READ_COIL = 1
    READ_BIT = 2
    READ_HOLDINGREG = 3
    READ_INPUTREG = 4
    WRITE_BIT = 5
    WRITE_HOLDINGREG = 6
    WRITEMULTI_COILS = 15
    WRITEMULTI_HOLDINGREGS = 16

class TcpModbusComm(object):
    '''
    This library will interface with umodbus with communications over a tcp/ip.
    def __init__(self, component, slaveAddress, portConfig):
    '''
    Constructor
    '''
    self.port_config = portConfig
    self.slaveAddress = slaveAddress
    self.portOpen = False

    '''
    Allow user to start initiation of the Modbus and opening of the UART port
    Defaults:
    mode='rtu' (versus 'ascii')
    CLOSE_PORT_AFTER_EACH_CALL=False
    precalculate_read_size=True - if False, serial port reads until timeout, instead of specific number of bytes
    handle_local_echo=False
    '''

    def isModbusAvailable(self):
        return self.portOpen

    def startModbus(self):
        try: #here 'sock' only works for one connection
            self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            self.sock.connect((self.port_config.ip, self.port_config.port))
            self.portOpen = True
            print("TcpModbusComm - open startModbus: " + self.port_config.ip + ", " + str(self.port_config.port))
            #self.modbusInstrument.debug = True
        except serial.SerialException: # change this to proper exception, about TCP
            print("TcpModbusComm - unable to startModbus: " + self.port_config.portname + ", " + str(self.port_config.port))
            self.sock.close()
            sys.exit(-1)

        '''
        Only port setting that is expected to be different from the default MODBUS settings is baudrate and timeout
        '''
        #self.modbusInstrument.serial.timeout = self.port_config.serialTimeout

    '''
    The user should stop the Modbus when their component ends (or wants to stop it). This will also close the TCP/IP port.
    '''
    def stopModbus(self):
        self.sock.close()
        self.portOpen = False

```

Figure A.5. tcpModbusComm.py

```

'''
Read multiple Slave Input Registers (16-bit per register)
Arguments:
    registerAddress (int): The starting slave register address (use decimal numbers, not hex).
    numberOfRegs (int): The number of registers to read
Returns: register dataset: list of int
'''

def readMultiInputRegValues(self, registerAddress, numberOfRegs, signed):
    value = -9999
    try:
        conf.SIGNED_VALUES = signed
        message = tcp.read_input_registers(slave_id=self.slaveAddress, starting_address=registerAddress, quantity=numberOfRegs)
        value = tcp.send_message(message, self.sock)
    except IOError: #change errors to TCP specific
        print("TcpModbusComm IOError: Failed to read input registers - address=" + str(registerAddress) + ", numberOfRegs=" + str(numberOfRegs))
    except TypeError:
        print("TcpModbusComm TypeError: Failed to read input registers - address=" + str(registerAddress) + ", numberOfRegs=" + str(numberOfRegs))
    return value

'''
Read multiple Slave Holding Registers (16-bit per register)
Arguments:
    registerAddress (int): The starting slave register address (use decimal numbers, not hex).
    numberOfRegs (int): The number of registers to read
Returns: register dataset: list of int
'''
def readMultiHoldingRegValues(self, registerAddress, numberOfRegs, signed):
    value = -9999
    try:
        conf.SIGNED_VALUES = signed
        message = tcp.read_holding_registers(slave_id=self.slaveAddress, starting_address=registerAddress, quantity=numberOfRegs)
        value = tcp.send_message(message, self.sock)
    except IOError:
        print("TcpModbusComm IOError: Failed to read holding registers - address=" + str(registerAddress) + ", numberOfRegs=" + str(numberOfRegs))
    except TypeError:
        print("TcpModbusComm TypeError: Failed to read holding registers - address=" + str(registerAddress) + ", numberOfRegs=" + str(numberOfRegs))
    return value

'''
Write multiple Slave holding register values (16 bits per register)
Arguments:
    registerAddress (int): The starting slave register address (use decimal numbers, not hex).
    values (list of int): The values to write - number of registers written is based on the length of the 'values' list
Returns: None
'''
Note: Command uses FunctionCode.writeHoldingRegs (16)
def writeHoldingRegisters(self, registerAddress, values, signed):
    try:
        conf.SIGNED_VALUES = signed
        message = tcp.write_multiple_registers(self.slaveAddress, registerAddress, values)
        value = tcp.send_message(message, self.sock)
    except IOError:
        print("TcpModbusComm IOError: Failed to write holding registers - address=" + str(registerAddress)) #MM TODO: add number of values
    except TypeError:
        print("TcpModbusComm TypeError: Failed to write holding registers - address=" + str(registerAddress)) #MM TODO: add number of values

```

Figure A.6. tcpModbusComm.py

```

hpp RIAPSMdbusTCP {
    on (192.168.10.111) ModbusExampleActor(Ts = 0.05, ipArg=111 );
}

```

Figure A.7. File "modbus\_tcp\_core.depl"



# Appendix B

## Code for Hardware in the loop simulation

```
// RIAPS Modbus (TCP) Device Testing

app RIAPSMoBusTCP {
  message ModbusCommand; // send Modbus action Request
  message ModbusResponse; // get response from Modbus action
  message ModbusTCPData;

  library tcpModbusLib;

  // Modbus (TCP) device interface
  // considered the server for the request/response interaction
  device ModbusTcpReqRepDevice(slaveaddress=3,ipaddress = '192.168.10.110',port=502,serialTimeout=1.0) {
    rep modbusRepPort : (ModbusCommand,ModbusResponse);
    timer clock 2000; //life signal
  }

  // Example Component to show Modbus I/F usage
  component ComputationalComponent(Ts = 0.1, ip = 91) {
    timer clock 8000;
    pub tx_modbusTCPData : ModbusTCPData;
    req modbusReqPort : (ModbusCommand,ModbusResponse); // Port used to communicate with the ModbusUartDevice
  }

  // Logging
  component ModbusTCPLogger() {
    timer clock 8000;
    sub rx_modbusTCPData : ModbusTCPData;
  }

  // Modbus Communication Example actor
  actor ModbusExampleActor(TsArg, ipArg) {
    local ModbusCommand,ModbusResponse; // Local message types
    {
      modbus : ModbusTcpReqRepDevice(slaveaddress=3,ipaddress = '192.168.10.110',port=502,serialTimeout=1.0); // Slave Address is in decimal
      commInitiator : ComputationalComponent( Ts = TsArg, ip = ipArg );
    }
  }

  actor ModbusTCPLogger() {
    {
      logger : ModbusTCPLogger();
    }
  }
}
```

Figure B.1. File "modbus\_tcp\_core.riaps"

```
'''
Created on Nov 25, 2019
@author: Giangabriele Castorini
'''

from riaps.run.comp import Component
from riaps.run.exc import PortError
import uuid
import os
from collections import namedtuple
from ModbusTcpRegRepDevice import CommandFormat, ModbusCommands

''' Enable debugging to gather timing information on the code execution'''
debugMode = False

'''Register number / value'''
RegSet = namedtuple('RegSet', ['RegNum', 'value'])
'''For Inverter Control'''
InputHoldingRegs = namedtuple('InputHoldingRegs', ['Grid_Guard_Code', 'Language', 'Operating_mode_relay', 'Active_Power_Limit', 'On_Off'])
'''For Inverter READ'''
HoldingRegs = namedtuple('HoldingRegs', ['Plant_mains_connection', 'DC_Power', 'DC_Voltage', 'DC_Current', 'AC_Grid_Frequency', 'AC_Active_Power',
'AC_Active_Power_phase_L1', 'AC_Active_Power_phase_L2', 'AC_Active_Power_phase_L3', 'AC_Reactive_Power', 'AC_Reactive_Power_phase_L1',
'AC_Reactive_Power_phase_L2', 'AC_Reactive_Power_phase_L3', 'AC_Apparent_Power', 'AC_Apparent_Power_phase_L1', 'AC_Apparent_Power_phase_L2',
'AC_Apparent_Power_phase_L3', 'AC_Voltage_phase_L1', 'AC_Voltage_phase_L2', 'AC_Voltage_phase_L3', 'AC_Grid_Current', 'AC_Grid_Current_phase_L1',
'AC_Grid_Current_phase_L2', 'AC_Grid_Current_phase_L3', 'Operating_Time', 'Feed_in_time', 'Daily_yield', 'Total_yield', 'Grid_Guard_Code', 'Language',
'Operating_mode_relay', 'Active_Power_Limit', 'On_Off'])

class ComputationalComponent(Component):
    def __init__(self, fr, ip):
        super().__init__()
        #pydevd.settrace(host='192.168.1.103',port=5678)
        self.uuid = uuid.uuid4().int
        self.pid = os.getpid()
        self.ip = ip
        self.inputHoldingRegs = InputHoldingRegs(RegSet(43090, [11830, 62961]), RegSet(40013, 0), RegSet(40575, 0), RegSet(40915, 0), RegSet(41253, 0))
        self.holdingRegs = HoldingRegs(RegSet(30881, 0), RegSet(30773, 0), RegSet(30771, 0), RegSet(30769, 0), RegSet(30803, 0), RegSet(30775, 0),
        RegSet(30777, 0), RegSet(30779, 0), RegSet(30781, 0), RegSet(30805, 0), RegSet(30807, 0), RegSet(30809, 0), RegSet(30811, 0), RegSet(30815, 0),
        RegSet(30815, 0), RegSet(30817, 0), RegSet(30819, 0), RegSet(30783, 0), RegSet(30785, 0), RegSet(30787, 0), RegSet(30795, 0), RegSet(30977, 0),
        RegSet(30979, 0), RegSet(30981, 0), RegSet(30541, 0), RegSet(30543, 0), RegSet(30535, 0), RegSet(30529, 0), RegSet(43090, 0), RegSet(40013, 0),
        RegSet(40575, 0), RegSet(40915, 0), RegSet(41253, 0))

        '''Setup Commands for modbusTCP'''
        #Initialization values
        self.readWrite = 1
        # 1 to will enable also write
        self.defaultInitialReg = 0
        self.defaultNumOfRegs = 0
        self.dummyValue = [0]
        self.defaultNumOfDecimals = 0
        self.signedDefault = False
        self.count = 1
        self.logger.info("ComputationalComponent: %s - starting" % str(self.pid))

    def on_clock(self):
        now = self.clock.recv_pyobj()
        self.logger.info("On_clock() [%s]: %s ::::::::::::::::::::::: UPDATING OPERATIONS ::::::::::::::::::::::: " % (str(self.pid), str(now)))

        '''Read multiple holding registers'''
        RegtypeR=['Connection type', 'DC', 'AC1', 'AC2', 'Time', 'Energy']

        for i in range (len(RegtypeR)):
            #Cycle for how many starting registers to read
            if(RegtypeR[i]=='Energy'):
                self.defaultInitialReg = self.holdingRegs.Total_yield.RegNum
                #Energy yield values
                self.defaultNumOfRegs = 8
                self.logger.info("Request to Read type <%s>" %str(RegtypeR[i]))
                self.command = CommandFormat(ModbusCommands.READMULTI_HOLDINGREGS, self.defaultInitialReg, self.defaultNumOfRegs,
                self.dummyValue, self.signedDefault)
                repMsg = self.sendModbusRequest(self.command)
                self.exitandcheckvalue(repMsg)

            elif(RegtypeR[i]=='DC'):
                #DC input values
                self.defaultInitialReg = self.holdingRegs.DC_Current.RegNum
                self.defaultNumOfRegs = 6
                self.logger.info("Request to Read type <%s>" %str(RegtypeR[i]))
                self.command = CommandFormat(ModbusCommands.READMULTI_HOLDINGREGS, self.defaultInitialReg, self.defaultNumOfRegs,
                self.dummyValue, self.signedDefault)
                repMsg = self.sendModbusRequest(self.command)
                self.exitandcheckvalue(repMsg)

            elif(RegtypeR[i]=='Time'):
                #Time values
                self.defaultInitialReg = self.holdingRegs.Operating_Time.RegNum
                self.defaultNumOfRegs = 4
                self.logger.info("Request to Read type <%s>" %str(RegtypeR[i]))
                self.command = CommandFormat(ModbusCommands.READMULTI_HOLDINGREGS, self.defaultInitialReg, self.defaultNumOfRegs,
                self.dummyValue, self.signedDefault)
                repMsg = self.sendModbusRequest(self.command)
                self.exitandcheckvalue(repMsg)

            elif(RegtypeR[i]=='AC1'):
                #AC1 output values
                self.defaultInitialReg = self.holdingRegs.AC_Active_Power.RegNum
                self.defaultNumOfRegs = 46
```

Figure B.2. ComputationalComponent.py

```

self.logger.info("Request to Read type <{s}>" %str(RegtypeR[i]))
self.command = CommandFormat(ModbusCommands.READMULTI_HOLDINGREGS, self.defaultInitialReg, self.defaultNumOfRegs,
self.dummyValue, self.signedDefault)
repMsg = self.sendModbusRequest(self.command)
self.exitandcheckvalue(repMsg)

elif(RegtypeR[i]=='AC2'):
    #AC2 output values
    self.defaultInitialReg = self.holdingRegs.AC_Grid_Current_phase_L1.RegNum
    self.defaultNumOfRegs = 6
    self.logger.info("Request to Read type <{s}>" %str(RegtypeR[i]))
    self.command = CommandFormat(ModbusCommands.READMULTI_HOLDINGREGS, self.defaultInitialReg, self.defaultNumOfRegs,
self.dummyValue, self.signedDefault)
    repMsg = self.sendModbusRequest(self.command)
    self.exitandcheckvalue(repMsg)

elif(RegtypeR[i]=='Connection type'):
    #Connection types values
    self.defaultInitialReg = self.holdingRegs.Plant_mains_connection.RegNum
    self.defaultNumOfRegs = 2
    self.logger.info("Request to Read type <{s}>" %str(RegtypeR[i]))
    self.command = CommandFormat(ModbusCommands.READMULTI_HOLDINGREGS, self.defaultInitialReg, self.defaultNumOfRegs,
self.dummyValue, self.signedDefault)
    repMsg = self.sendModbusRequest(self.command)
    self.exitandcheckvalue(repMsg)

'''Write multiple holding registers'''
if (self.readorwrite == 1):
    RegtypeW=['GridGuardCode','Language','Operating mode of multi-function relay','Active power Limit','On/Off']
    for i in range (len(RegtypeW)):
        #Cycle for how many starting registers to read
        if(RegtypeW[i]=='GridGuardCode'):
            self.defaultInitialReg = self.inputHoldingRegs.Grid_Guard_Code.RegNum #Register for Grid Guard Code
            self.defaultNumOfRegs = 2
            'Read before Write'
            self.logger.info("Request to Read type <{s}>" %str(RegtypeW[i]))
            self.command = CommandFormat(ModbusCommands.READMULTI_HOLDINGREGS, self.defaultInitialReg, self.defaultNumOfRegs,
self.dummyValue, self.signedDefault)
            repMsg = self.sendModbusRequest(self.command)
            self.exitandcheckvalue(repMsg)
            'Write'
            self.values = self.inputHoldingRegs.Grid_Guard_Code.value #Grid Guard Code (fixed)
            if (self.holdingRegs.Grid_Guard_Code.value==0 and self.count==1): #Write only if it is not already written
                self.count=0
                self.logger.info("Request to Write type <{s}>" %str(RegtypeW[i])) #Remove self.count if running for long time/SMA updt the firm.
                self.command = CommandFormat(ModbusCommands.WRITEMULTI_HOLDINGREGS, self.defaultInitialReg, self.defaultNumOfRegs,
self.values, self.signedDefault)
                repMsg = self.sendModbusRequest(self.command)
                self.exitandcheckvalue(repMsg)
            else:
                self.logger.info("Requested to Write type <{s}> -Value already written-" %str(RegtypeW[i]))

elif(RegtypeW[i]=='Language'):
    #Register for Changing Language
    self.defaultInitialReg = self.inputHoldingRegs.Language.RegNum
    self.defaultNumOfRegs = 2
    'Read before Write'
    self.logger.info("Request to Read type <{s}>" %str(RegtypeW[i]))
    self.command = CommandFormat(ModbusCommands.READMULTI_HOLDINGREGS, self.defaultInitialReg, self.defaultNumOfRegs,
self.dummyValue, self.signedDefault)
    repMsg = self.sendModbusRequest(self.command)
    self.exitandcheckvalue(repMsg)
    'Write'
    msgval = [0, 778] #Language: 777=German 778=English 779=Italian 780=Spanish
    msg0=msgval[0]*65536+msgval[1] #Converted to 1 value for comparison reason
    'Storing writing values for later usage'
    self.inputHoldingRegs = self.inputHoldingRegs.replace(Language=RegSet(RegNum=40013,value=msg0))
    if (self.inputHoldingRegs.Language.value != self.holdingRegs.Language.value): #Write only if it is not yet written
        self.values = msgval
        self.logger.info("Request to Write type <{s}>" %str(RegtypeW[i]))
        self.command = CommandFormat(ModbusCommands.WRITEMULTI_HOLDINGREGS, self.defaultInitialReg, self.defaultNumOfRegs,
self.values, self.signedDefault)
        repMsg = self.sendModbusRequest(self.command)
        self.exitandcheckvalue(repMsg)
    else:
        self.logger.info("Requested to Write type <{s}> -Value already written-" %str(RegtypeW[i]))

elif(RegtypeW[i]=='Operating mode of multi-function relay'):
    self.defaultInitialReg = self.inputHoldingRegs.Operating_mode_relay.RegNum #Reg. for Op. mode of multi-func relay
    self.defaultNumOfRegs = 2
    'Read before Write'
    self.logger.info("Request to Read type <{s}>" %str(RegtypeW[i]))
    self.command = CommandFormat(ModbusCommands.READMULTI_HOLDINGREGS, self.defaultInitialReg, self.defaultNumOfRegs,
self.dummyValue, self.signedDefault)
    repMsg = self.sendModbusRequest(self.command)
    self.exitandcheckvalue(repMsg)
    'Write'
    msgval = [0, 1349] #Operationlong mode : Switching status grid relay= 258 Fault indication=1341
    #Fan control=1342 Self-consumption=1343 Control via communication=1349 Battery bank=1359
    msg0=msgval[0]*65536+msgval[1] #Converted to 1 value for comparison reason
    'Storing writing values for later usage'
    self.inputHoldingRegs = self.inputHoldingRegs.replace(Operating mode relay=RegSet(RegNum=40575,value=msg0))
    if (self.inputHoldingRegs.Operating_mode_relay.value != self.holdingRegs.Operating_mode_relay.value):
        self.logger.info("Request to Write type <{s}>" %str(RegtypeW[i]))
        self.values = msgval
        self.command = CommandFormat(ModbusCommands.WRITEMULTI_HOLDINGREGS, self.defaultInitialReg, self.defaultNumOfRegs,
self.values, self.signedDefault)
        repMsg = self.sendModbusRequest(self.command)

```

Figure B.3. ComputationalComponent.py

```

        self.exitandcheckvalue(repMsg)
    else:
        self.logger.info("Requested to Write type <%s> -Value already written-" %str(RegtypeW[i]))

    elif(RegtypeW[i]=='Active power Limit'):
        #Register for Active power Limit
        self.defaultInitialReg = self.inputHoldingRegs.Active_Power_Limit.RegNum
        self.defaultNumOfRegs = 2
        'Read before Write'
        self.logger.info("Request to Read type <%s>" %str(RegtypeW[i]))
        self.command = CommandFormat(ModbusCommands.READMULTI_HOLDINGREGS, self.defaultInitialReg , self.defaultNumOfRegs ,
        self.dummyValue, self.signedDefault)
        repMsg = self.sendModbusRequest(self.command)
        self.exitandcheckvalue(repMsg)
        'Write'
        msgval = [0,20000]
        #Active power limit in [W]
        msg0=msgval[0]*65536+msgval[1]
        #Converted to 1 value for comparison reason
        'Storing writing values for later usage'
        self.inputHoldingRegs = self.inputHoldingRegs._replace(Active_Power_Limit=RegSet(RegNum=40915,value=msg0))
        if (self.inputHoldingRegs.Active_Power_Limit.value != self.holdingRegs.Active_Power_Limit.value) :
            self.logger.info("Request to Write type <%s>" %str(RegtypeW[i]))
            self.values=msgval
            self.command = CommandFormat(ModbusCommands.WRITEMULTI_HOLDINGREGS, self.defaultInitialReg, self.defaultNumOfRegs,
            self.values, self.signedDefault)
            repMsg = self.sendModbusRequest(self.command)
            self.exitandcheckvalue(repMsg)
        else:
            self.logger.info("Requested to Write type <%s> -Value already written-" %str(RegtypeW[i]))

    elif(RegtypeW[i]=='On/Off'):
        self.defaultInitialReg = self.inputHoldingRegs.On_Off.RegNum
        #Register for On/Off Fast shut-down
        self.defaultNumOfRegs = 2
        'Read before Write'
        self.logger.info("Request to Read type <%s>" %str(RegtypeW[i]))
        self.command = CommandFormat(ModbusCommands.READMULTI_HOLDINGREGS, self.defaultInitialReg , self.defaultNumOfRegs ,
        self.dummyValue, self.signedDefault)
        repMsg = self.sendModbusRequest(self.command)
        self.exitandcheckvalue(repMsg)
        'Write'
        if(==):
            Reason why start or stop the inverter (to be implemented)
            msgval = [0, 1467]
            #Start AC+DC side=1467 Stop DC side=391 Full Stop AC+DC side=1749
            msg0=msgval[0]*65536+msgval[1]
            'Storing writing values for later usage'
            self.inputHoldingRegs = self.inputHoldingRegs._replace(On_Off=RegSet(RegNum=41253,value=msg0))
            if (self.inputHoldingRegs.On_Off.value != self.holdingRegs.On_Off.value) :
                self.logger.info("Request to Write type <%s>" %str(RegtypeW[i]))
                self.values=msgval
                self.command = CommandFormat(ModbusCommands.WRITEMULTI_HOLDINGREGS, self.defaultInitialReg, self.defaultNumOfRegs,
                self.values, self.signedDefault)
                repMsg = self.sendModbusRequest(self.command)
                self.exitandcheckvalue(repMsg)
            else:
                self.logger.info("Requested to Write type <%s> -Value already written-" %str(RegtypeW[i]))

def sendModbusRequest(self, requestMsg):
    rep = None
    #Init to invalid response value
    try:
        self.modbusReqPort.send_pyobj(requestMsg)
        rep = self.modbusReqPort.recv_pyobj()
    except PortError as e:
        self.logger.info("on_clock:send exception = %d" % e.errno)
        if e.errno in (PortError.EAGAIN,PortError.EPROTO):
            self.logger.info("on_clock: port error received")
    return rep

def exitandcheckvalue(self, repmsg):
    if (repmsg != None):
        #Check in both READ/WRITE case
        Cmsg=self.registertable(repmsg)
        #Take input repMsg and receive d in Cmsg or return MsgVal again
        self.logger.info("Converted values ::::LOADING:::: to the logger" )
        self.tx_modbusTCPData.send_pyobj(Cmsg)
        #Send Log data values to the logger
        if (repmsg ==1):
            self.logger.info("Modbus not ready error level at on_RepPort(Device)" )
        if (repmsg ==999):
            self.logger.info("Modbus not ready to Read/Write error level at Function SendModbusCommand(Device)" )
        elif (repmsg ==-9999):
            self.logger.info("Modbus not ready error level at Function ReadMultiHoldingRegisters(Library)" )
        elif (repmsg ==-9990):
            self.logger.info("Modbus not ready error level at Function WritingHoldingRegisters(Library)" )
    else:
        self.logger.info("Modbus either not ready or no value read")

def registertable(self, Msgval):
    #Function recognize the register and correct the values
    if (self.defaultInitialReg==self.holdingRegs.Plant_mains_connection.RegNum and self.defaultNumOfRegs==2):
        #READ part Connection type
        msg0=(Msgval[0]*65535+Msgval[1])
        #30881 and 30882 Connection type
        'Storing values for later usage'
        self.holdingRegs = self.holdingRegs._replace(Plant_mains_connection=RegSet(RegNum=30881,value=msg0))
        if (msg0==1779):
            text='Separated'
        elif (msg0==1780):
            text='Public electricity mains'
        elif (msg0==1781):
            text='Inland mains'

```

Figure B.4. ComputationalComponent.py

```

else:
    text="UNKNOWN"
    d = {'Plant mains connection':text}

elif (self.defaultInitialReg==self.holdingRegs.DC_Current_RegNum and self.defaultNumOfRegs==6): #READ part DC values
    msg0=round((Msgval[0]*65536+Msgval[1])*.001,2) #30769 and 30770 DC input Current
    msg1=round((Msgval[2]*65536+Msgval[3])*.001,2) #30771 and 30772 DC input Voltage
    msg2=Msgval[4]*65536+Msgval[5] #30773 and 30774 DC input Power
    'Correction values when inverter is turned off'
    if (msg0==2147483648): #Correction values when inverter is turned off
        msg1=msg2=msg3=0
    'Storing values for later usage'
    self.holdingRegs = self.holdingRegs.replace(DC_Current=RegSet(RegNum=30769,value=msg0))
    self.holdingRegs = self.holdingRegs.replace(DC_Voltage=RegSet(RegNum=30771,value=msg1))
    self.holdingRegs = self.holdingRegs.replace(DC_Power=RegSet(RegNum=30773,value=msg2))
    d = {'DC Power [W]':msg2,
        'DC Voltage [V]':msg1,
        'DC Current [A]':msg0}

elif (self.defaultInitialReg==self.holdingRegs.AC_Active_Power_RegNum and self.defaultNumOfRegs==46): #READ part AC1 values
    msg0=Msgval[0]*6553+Msgval[1] #30775 and 30776 AC Active Power L1
    msg1=Msgval[2]*6553+Msgval[3] #30777 and 30778 Active Power L1
    msg2=Msgval[4]*6553+Msgval[5] #30779 and 30780 Active Power L2
    msg3=Msgval[6]*6553+Msgval[7] #30781 and 30782 Active Power L3
    msg4=round((Msgval[8]*65535+Msgval[9])*.001,2) #30783 and 30784 Grid Voltage L1
    msg5=round((Msgval[10]*65535+Msgval[11])*.001,2) #30785 and 30786 Grid Voltage L2
    msg6=round((Msgval[12]*65535+Msgval[13])*.001,2) #30787 and 30788 Grid Voltage L3
    msg7=round((Msgval[20]*65535+Msgval[21])*.001,2) #30795 and 30796 Tot Grid Current
    msg8=round((Msgval[28]*65535+Msgval[29])*.001,2) #30803 and 30804 Grid Frequency
    msg9=Msgval[30]+55935 #30805 and 30806 Tot Reactive pow
    msg10=(Msgval[32]*65535-(65535-Msgval[33])) #30807 and 30808 Reactive Power L1
    msg11=(Msgval[34]*65535-(65535-Msgval[35])) #30809 and 30810 Reactive Power L2
    msg12=(Msgval[36]*65535-(65535-Msgval[37])) #30811 and 30812 Reactive Power L3
    msg13=(Msgval[38]*65535-(65535-Msgval[39])) #30813 and 30814 Apparent Power L1
    msg14=(Msgval[40]*65536+Msgval[41]) #30815 and 30816 Apparent Power L2
    msg15=(Msgval[42]*65536+Msgval[43]) #30817 and 30818 Apparent Power L3
    msg16=(Msgval[44]*65536+Msgval[45]) #30819 and 30820 Apparent Power L3
    'Correction values when inverter is turned off'
    if (msg0==0 or msg0==2147483648 or msg0==2147450880 and msg1==2147450880 or msg10==4294901760):
        msg0=msg1=msg2=msg3=msg4=msg5=msg6=msg7=msg8=msg9=msg10=msg11=msg12=msg13=msg14=msg15=msg16=0
    'Storing values for later usage'
    self.holdingRegs = self.holdingRegs.replace(AC_Active_Power=RegSet(RegNum=30775,value=msg0))
    self.holdingRegs = self.holdingRegs.replace(AC_Active_Power_phase_L1=RegSet(RegNum=30777,value=msg1))
    self.holdingRegs = self.holdingRegs.replace(AC_Active_Power_phase_L2=RegSet(RegNum=30779,value=msg2))
    self.holdingRegs = self.holdingRegs.replace(AC_Active_Power_phase_L3=RegSet(RegNum=30781,value=msg3))
    self.holdingRegs = self.holdingRegs.replace(AC_Voltage_phase_L1=RegSet(RegNum=30783,value=msg4))
    self.holdingRegs = self.holdingRegs.replace(AC_Voltage_phase_L2=RegSet(RegNum=30785,value=msg5))
    self.holdingRegs = self.holdingRegs.replace(AC_Voltage_phase_L3=RegSet(RegNum=30787,value=msg6))
    self.holdingRegs = self.holdingRegs.replace(AC_Grid_Current=RegSet(RegNum=30795,value=msg7))
    self.holdingRegs = self.holdingRegs.replace(AC_Grid_Frequency=RegSet(RegNum=30803,value=msg8))
    self.holdingRegs = self.holdingRegs.replace(AC_Reactive_Power=RegSet(RegNum=30805,value=msg9))
    self.holdingRegs = self.holdingRegs.replace(AC_Reactive_Power_phase_L1=RegSet(RegNum=30807,value=msg10))
    self.holdingRegs = self.holdingRegs.replace(AC_Reactive_Power_phase_L2=RegSet(RegNum=30809,value=msg11))
    self.holdingRegs = self.holdingRegs.replace(AC_Reactive_Power_phase_L3=RegSet(RegNum=30811,value=msg12))
    self.holdingRegs = self.holdingRegs.replace(AC_Apparent_Power=RegSet(RegNum=30813,value=msg13))
    self.holdingRegs = self.holdingRegs.replace(AC_Apparent_Power_phase_L1=RegSet(RegNum=30815,value=msg14))
    self.holdingRegs = self.holdingRegs.replace(AC_Apparent_Power_phase_L2=RegSet(RegNum=30817,value=msg15))
    self.holdingRegs = self.holdingRegs.replace(AC_Apparent_Power_phase_L3=RegSet(RegNum=30819,value=msg16))
    d = {'AC Grid Frequency [Hz]':msg8,
        'AC Active Power [W]':msg0,
        'AC Active Power phase L1 [W]':msg1,
        'AC Active Power phase L2 [W]':msg2,
        'AC Active Power phase L3 [W]':msg3,
        'AC Reactive Power [Var]':msg9,
        'AC Reactive Power phase L1 [Var]':msg10,
        'AC Reactive Power phase L2 [Var]':msg11,
        'AC Reactive Power phase L3 [Var]':msg12,
        'AC Apparent Power [VA]':msg13,
        'AC Apparent Power phase L1 [VA]':msg14,
        'AC Apparent Power phase L2 [VA]':msg15,
        'AC Apparent Power phase L3 [VA]':msg16,
        'AC Voltage phase L1 [V]':msg4,
        'AC Voltage phase L2 [V]':msg5,
        'AC Voltage phase L3 [V]':msg6,
        'AC Grid Current [A]':msg7}

elif (self.defaultInitialReg==self.holdingRegs.AC_Grid_Current_phase_L1_RegNum and self.defaultNumOfRegs==6): #READ part AC2 values
    msg0=round((Msgval[0]*65535+Msgval[1])*.001,2) #30977 and 30978 Grid Current phase L1
    msg1=round((Msgval[2]*65535+Msgval[3])*.001,2) #30979 and 30980 Grid Current phase L2
    msg2=round((Msgval[4]*65535+Msgval[5])*.001,2) #30981 and 30982 Grid Current phase L3
    'Correction values when inverter is turned off'
    if (msg0==2147450,800): #Correction values when inverter is turned off
        msg1=msg2=msg3=0
    'Storing values for later usage'
    self.holdingRegs = self.holdingRegs.replace(AC_Grid_Current_phase_L1=RegSet(RegNum=30977,value=msg0))
    self.holdingRegs = self.holdingRegs.replace(AC_Grid_Current_phase_L2=RegSet(RegNum=30979,value=msg1))
    self.holdingRegs = self.holdingRegs.replace(AC_Grid_Current_phase_L3=RegSet(RegNum=30981,value=msg2))
    d = {'AC Current phase L1 [A]':msg0,
        'AC Current phase L2 [A]':msg1,
        'AC Current phase L3 [A]':msg2}

elif (self.defaultInitialReg==self.holdingRegs.Operating_Time_RegNum and self.defaultNumOfRegs==4): #READ part Time values
    msg0=round((Msgval[0]*65536+Msgval[1])/60,2) #30541 and 30542 Operating Time
    msg1=round((Msgval[2]*65536+Msgval[3])/360,2) #30543 and 30544 Feed in time

```

```

        'Storing values for later usage'
        self.holdingRegs = self.holdingRegs._replace(Operating_Time=RegSet(RegNum=30541,value=msg0))
        self.holdingRegs = self.holdingRegs._replace(Feed_in_Time=RegSet(RegNum=30543,value=msg1))
        d = {"Operating Time [h]":msg0,
            "Feed in time [h]":msg1}

    elif (self.defaultInitialReg==self.holdingRegs.Total_yield.RegNum and self.defaultNumOfRegs==3): #READ part Energy values
        msg0=round((Msgval[0]*65536+Msgval[1])*0.000001,6) #30529 and 30530 Total Yield
        msg1=round((Msgval[0]*65536+Msgval[1]),2) #30535 and 30536 Daily Yield
        'Storing values for later usage'
        self.holdingRegs = self.holdingRegs._replace(Total_yield=RegSet(RegNum=30529,value=msg0))
        self.holdingRegs = self.holdingRegs._replace(Daily_yield=RegSet(RegNum=30535,value=msg1))
        d = {"Daily yield [MWh]":msg1,
            "Total yield [MWh]":msg0}

    elif (self.defaultInitialReg==self.holdingRegs.Grid_Guard_Code.RegNum and self.defaultNumOfRegs==2): #WRITE/READ part GridGuardCode
        try:
            msg0=(Msgval[0]*Msgval[1])
            'Storing values for later usage'
            self.holdingRegs = self.holdingRegs._replace(Grid_Guard_Code=RegSet(RegNum=43090,value=msg0))
            if (msg0==0):
                text='Valid' #To be changed to 'INVALID' when the Firmware by SMA is updated
            else:
                text='Valid'
            d = {"Grid Guard Code ":text}
        except:
            msg0=Msgval #For Write format values
            if (msg0==2):
                text='Successfully'
            else:
                text='Failed'
            d = {"Grid Guard Code just sent ::: ":text}

    elif (self.defaultInitialReg==self.holdingRegs.Language.RegNum and self.defaultNumOfRegs==2): #WRITE/READ part Language
        try:
            msg0=(Msgval[0]*65536+Msgval[1]) #For Read format values
            'Storing values for later usage' #40013 and 40014 Language (777-German), (778-English), (779-Italian)
            self.holdingRegs = self.holdingRegs._replace(Language=RegSet(RegNum=40013,value=msg0))
            if (msg0==777):
                text='Deutsch'
            elif (msg0==778):
                text='English'
            elif (msg0==779):
                text='Italiano'
            else:
                text='UNKNOWN'
            d = {"Language ":text}
        except:
            msg0=Msgval #For Write format values
            if (msg0==2):
                text='Successfully'
            else:
                text='Failed'
            d = {"Change Language Command just sent ":text}

    elif (self.defaultInitialReg==self.holdingRegs.Operating_mode_relay.RegNum and self.defaultNumOfRegs==2): #WRITE/READ part Op. mode multi-func. relay
        try:
            msg0=(Msgval[0]*65536+Msgval[1]) #For Read format values
            'Storing values for later usage'
            self.holdingRegs = self.holdingRegs._replace(Operating_mode_relay=RegSet(RegNum=40575,value=msg0))
            if (msg0==200):
                text='Switching status grid relay'
            elif (msg0==1341):
                text='Fault indication'
            elif (msg0==1342):
                text='Fan control'
            elif (msg0==1343):
                text='Self-consumption'
            elif (msg0==1349):
                text='Control via communication'
            elif (msg0==1359):
                text='Battery bank'
            else:
                text='UNKNOWN'
            d = {"Multi-function relay mode ":text}
        except:
            msg0=Msgval #For Write format values
            if (msg0==2):
                text='Successfully'
            else:
                text='Failed'
            d = {"Change mode for relay Command just sent ":text}

    elif (self.defaultInitialReg==self.holdingRegs.Active_Power_Limit.RegNum and self.defaultNumOfRegs==2): #WRITE/READ part Active power Limit
        try:
            msg0=(Msgval[0]*65536+Msgval[1]) #For Read format values
            'Storing values for later usage' #Converted to 1 value for comparison reason
            self.holdingRegs = self.holdingRegs._replace(Active_Power_Limit=RegSet(RegNum=40915,value=msg0))
            d = {"Active power limitation in [W] ":msg0}
        except:
            msg0=Msgval #For Write format values
            if (msg0==2):
                text='Successfully'
            else:
                text='Failed'
            d = {"Change Active Power Limit Command just sent ":text}

    elif (self.defaultInitialReg==self.holdingRegs.On_Off.RegNum and self.defaultNumOfRegs==2): #WRITE/READ part On/Off
        try:
            msg0=(Msgval[0]*65536+Msgval[1]) #For Read format values
            'Storing values for later usage' #41253 (381-Stop), (1749=Full-stop) (1467-Start)
            self.holdingRegs = self.holdingRegs._replace(On_Off=RegSet(RegNum=41253,value=msg0))
            if (msg0==381):
                text='Stop DC side'
            elif (msg0==1749):
                text='Full-stop AC+DC side'
            elif (msg0==1467):
                text='Start'
            else:
                text='UNKNOWN'
            d = {"Fast Start/Shut-down set to ":text}
        except:
            msg0=Msgval #For Write format values
            if (msg0==2):
                text='Successfully'
            else:
                text='Failed'
            d = {"Change Start/Shut-down Command just sent ":text}

    else:
        text=self.defaultInitialReg
        d = ("...UNKNOWN REGISTER SELECTED...":text)
        #print (self.holdingRegs)
        #print (self.inputHoldingRegs)
        return d

def _destroy_(self):
    self.logger.info("[&d] destroyed" % self.pid)

```

Figure B.6. ComputationalComponent.py



```

'''
Created on Nov 11, 2014

This module utilizes the umodbus.
Both need to be installed in the development environment.
... $ sudo pip3 install umodbus
...

from riaps.run.comp import Component
import logging
import os
from collections import namedtuple
from enum import Enum
import time
from tcpModbusLib.tcpModbusComm import TcpModbusComm, PortConfig

''' Enable debugging to gather timing information on the code execution'''
debugMode = False

class ModbusCommands(Enum):
    READ_BIT = 1
    READ_INPUTREG = 2
    READ_HOLDINGREG = 3
    READMULTI_INPUTREGS = 4
    READMULTI_HOLDINGREGS = 5
    WRITE_BIT = 6
    WRITE_HOLDINGREG = 7
    WRITEMULTI_HOLDINGREGS = 8

CommandFormat = namedtuple('CommandFormat', ['commandType', 'registerAddress', 'numberOfRegs', 'values', 'signedValue'])

class ModbusTcpReqRepDevice(Component):
    def __init__(self, slaveaddress=31, ipaddress = '192.168.10.101', port = 1504, serialTimeout=0.05): # defaults for Modbus spec
        super().__init__()

        self.pid = os.getpid()
        self.port_config = PortConfig(ipaddress, port, serialTimeout)
        self.modbus = TcpModbusComm(self, slaveaddress, self.port_config)
        self.modbusInit = False
        if debugMode:
            self.logger.info("Modbus settings %d %s: %d %d [%d]", self.slaveaddress, self.port_config.ip, self.port_config.port, self.port_config.timeout, self.pid)

    def on_clock(self):
        now = self.clock.recv_pyobj() # Receive time (as float)
        self.logger.info("on_clock() [%s]: %s" % (str(self.pid), now))

        if debugMode:
            t0 = time.perf_counter()
            self.logger.debug("on_clock() [%s]: Request Modbus start at %f", str(self.pid), t0)

        if self.modbusInit == False:
            self.modbusInit = True:
            self.modbus.startModbus()
            pydevd.settrace(host='192.168.1.102', port=5678)

            if debugMode:
                t1 = time.perf_counter()
                self.logger.debug("on_clock() [%s]: Modbus ready at %f, time to start Modbus is %f ms", str(self.pid), t1, (t1-t0)*1000)

        self.clock.halt()

    def __destroy__(self):
        self.logger.info("__destroy__")
        self.modbus.stopModbus()

    ...
    Receive a Modbus command request. Process command and send back response.
    ...
    def on_modbusRepPort(self):
        '''Request Received'''
        commandRequest = self.modbusRepPort.recv_pyobj()

        if debugMode:
            self.modbusReqRxTime = time.perf_counter()
            self.logger.debug("on_modbusRepPort() [%s]: Request=%s Received at %f", str(self.pid), commandRequest, self.modbusReqRxTime)

        self.unpackCommand(commandRequest)
        responseValue = -1 # invalid response
        if self.modbus.isModbusAvailable() == True:
            responseValue = self.sendModbusCommand()

        if debugMode:
            t1 = time.perf_counter()
            self.logger.debug("on_modbusRepPort() [%s]: Send Modbus response=%s back to requester at %f", str(self.pid), responseValue, t1)

```

Figure B.7. ModbusTcpReqRepDevice.py

```

#
pydevd.settrace(host='192.168.1.102',port=5678)
else:
    self.logger.debug("Modbus is not available")
    '''Send Results'''
    self.modbusRepPort.send_pyobj(responseValue)

def unpackCommand(self,rxCommand):
    self.commandRequested = rxCommand.commandType
    self.registerAddress = rxCommand.registerAddress
    self.numberOfRegs = rxCommand.numberOfRegs
    self.values = rxCommand.values
    self.signedValue = rxCommand.signedValue

def sendModbusCommand(self):
    value = 999 # large invalid value

    if debugMode:
        t0 = time.perf_counter()
        self.logger.debug("sendModbusCommand() [%s]: Sending command to Modbus library at %f",str(self.pid),t0)

    if self.commandRequested == ModbusCommands.READMULTI_INPUTREGS:
        value = self.modbus.readMultiInputRegValues(self.registerAddress, self.numberOfRegs, self.signedValue)
        #self.logger.info("ModbusUartDevice: sent command %s, register=%d, numOfRegs=%d", ModbusCommands.READMULTI_INPUTREGS.name,self.registerAddress,self.numberOfRegs)
    elif self.commandRequested == ModbusCommands.READMULTI_HOLDINGREGS:
        value = self.modbus.readMultiHoldingRegValues(self.registerAddress, self.numberOfRegs, self.signedValue)
        #self.logger.info("ModbusUartDevice: sent command %s, register=%d, numOfRegs=%d", ModbusCommands.READMULTI_HOLDINGREGS.name,self.registerAddress,self.numberOfRegs)
    elif self.commandRequested == ModbusCommands.WRITEMULTI_HOLDINGREGS:
        self.modbus.writeHoldingRegisters(self.registerAddress, self.values, self.signedValue)
        #self.logger.info("ModbusUartDevice: sent command %s, register=%d",ModbusCommands.WRITEMULTI_HOLDINGREGS.name,self.registerAddress)
        #self.logger.info("ModbusUartDevice: Values - %s", str(self.values).strip('[]'))

    if debugMode:
        t1 = time.perf_counter()
        self.logger.debug("sendModbusCommand() [%s]: Modbus library command complete at %f, time to interact with Modbus library is %f ms",str(self.pid),t1,(t1-t0)*1000)

    return value

```

Figure B.8. ModbusTcpReqRepDevice.py

```

'''
Created on Nov 27, 2019
@author: Giangabriele Castorina
'''
from riaps.run.comp import Component
import os

class ModbusTCPLogger(Component):
    def __init__(self):
        super().__init__()
        self.pid = os.getpid()
        self.logger.info("%s - Starting modbusTCPlogger" % str(self.pid))

    def on_clock(self):
        now = self.clock.recv_pyobj()
        self.logger.info("On_clock() [%s]: %s :::::::::::::::UPDATED VALUES INCOMING::::::::::::::::::" % (str(self.pid), str(now)))

    def on_rx_modbusTCPData(self):
        msg = self.rx_modbusTCPData.recv_pyobj()
        #self.logger.info("on_rx_modbusTCPData() [%s]: %s" % (str(self.pid), repr(msg)))
        item=msg.items()
        #self.logger.info("::::::::: Last Updated values ::::::::::")
        for k in item:
            #print('VARIABLE :',k[0], 'VALUE :', k[1]) #Variable, Value
            self.logger.info("Received on_rx_modbusTCPData() [%s]: %s:%s" % (str(self.pid), repr(k[0]), repr(k[1])))

```

Figure B.9. ModbusTCPLLogger.py



```

'''
Created on Nov 19, 2019

This modbus device interface will read input registers and reads/writes holding registers of slave devices with modbus tcp.
At this time, it does not read/write coils.
'''

#!/usr/bin/env python
import socket

from umodbus import conf
from umodbus.client import tcp

from collections import namedtuple
import sys
from enum import Enum, unique

# import pydevd

'''serialTimeout is defined in seconds'''
PortConfig = namedtuple('PortConfig', ['ip', 'port', 'timeout'])

'''
Function Codes (per Modbus Spec)
'''
@unique
class FunctionCodes(Enum):
    READ_COIL = 1
    READ_BIT = 2
    READ_HOLDINGREG = 3
    READ_INPUTREG = 4
    WRITE_BIT = 5
    WRITE_HOLDINGREG = 6
    WRITEMULTI_COILS = 15
    WRITEMULTI_HOLDINGREGS = 16

class TcpModbusComm(object):
    '''
    This library will interface with umodbus with communications over a tcp/ip.
    def __init__(self, component, slaveAddress, portConfig):
    '''
    Constructor
    '''
    self.port_config = portConfig
    self.slaveAddress = slaveAddress
    self.portOpen = False

    '''
    Allow user to start initiation of the Modbus and opening of the UART port
    Defaults:
    mode='rtu' (versus 'ascii')
    CLOSE_PORT_AFTER_EACH_CALL=False
    precalculate_read_size=True - if False, serial port reads until timeout, instead of specific number of bytes
    handle_local_echo=False
    '''

    def isModbusAvailable(self):
        return self.portOpen

    def startModbus(self):
        try: #here 'sock' only works for one connection
            self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            self.sock.connect((self.port_config.ip, self.port_config.port))
            self.portOpen = True
            print("TcpModbusComm - open startModbus: " + self.port_config.ip + ", " + str(self.port_config.port))
            #self.modbusInstrument.debug = True
        except serial.SerialException: # change this to proper exception, about TCP
            print("TcpModbusComm - unable to startModbus: " + self.port_config.portname + ", " + str(self.port_config.port))
            self.sock.close()
            sys.exit(-1)

        '''
        Only port setting that is expected to be different from the default MODBUS settings is baudrate and timeout
        '''
        #self.modbusInstrument.serial.timeout = self.port_config.serialTimeout

    '''
    The user should stop the Modbus when their component ends (or wants to stop it). This will also close the TCP/IP port.
    '''
    def stopModbus(self):
        self.sock.close()
        self.portOpen = False

```

Figure B.10. tcpModbusComm.py

```

'''
Read multiple Slave Input Registers (16-bit per register)
Arguments:
    registerAddress (int): The starting slave register address (use decimal numbers, not hex).
    numberOfRegs (int): The number of registers to read
Returns: register dataset: list of int
'''

def readMultiInputRegValues(self, registerAddress, numberOfRegs, signed):
    value = -9999
    try:
        conf.SIGNED_VALUES = signed
        message = tcp.read_input_registers(slave_id=self.slaveAddress, starting_address=registerAddress, quantity=numberOfRegs)
        value = tcp.send_message(message, self.sock)
    except IOError: #change errors to TCP specific
        print("TcpModbusComm IOError: Failed to read input registers - address=" + str(registerAddress) + ", numberOfRegs=" + str(numberOfRegs))
    except TypeError:
        print("TcpModbusComm TypeError: Failed to read input registers - address=" + str(registerAddress) + ", numberOfRegs=" + str(numberOfRegs))
    return value

'''
Read multiple Slave Holding Registers (16-bit per register)
Arguments:
    registerAddress (int): The starting slave register address (use decimal numbers, not hex).
    numberOfRegs (int): The number of registers to read
Returns: register dataset: list of int
'''

def readMultiHoldingRegValues(self, registerAddress, numberOfRegs, signed):
    value = -9999
    try:
        conf.SIGNED_VALUES = signed
        message = tcp.read_holding_registers(slave_id=self.slaveAddress, starting_address=registerAddress, quantity=numberOfRegs)
        value = tcp.send_message(message, self.sock)
    except IOError:
        print("TcpModbusComm IOError: Failed to read holding registers - address=" + str(registerAddress) + ", numberOfRegs=" + str(numberOfRegs))
    except TypeError:
        print("TcpModbusComm TypeError: Failed to read holding registers - address=" + str(registerAddress) + ", numberOfRegs=" + str(numberOfRegs))
    return value

'''
Write multiple Slave holding register values (16 bits per register)
Arguments:
    registerAddress (int): The starting slave register address (use decimal numbers, not hex).
    values (list of int): The values to write - number of registers written is based on the length of the 'values' list
Returns: None

Note: Command uses FunctionCode.writeHoldingRegs (16)
'''

def writeHoldingRegisters(self, registerAddress, values, signed):
    try:
        conf.SIGNED_VALUES = signed
        message = tcp.write_multiple_registers(self.slaveAddress, registerAddress, values)
        value = tcp.send_message(message, self.sock)
    except IOError:
        print("TcpModbusComm IOError: Failed to write holding registers - address=" + str(registerAddress)) #MM TODO: add number of values
    except TypeError:
        print("TcpModbusComm TypeError: Failed to write holding registers - address=" + str(registerAddress)) #MM TODO: add number of values

```

Figure B.11. tcpModbusComm.py

```

#pp RIAPModbusTCP {
    on (192.168.10.91) ModbusExampleActor(Ts = 0.05, ipArg=91.);
    on (192.168.10.93) ModbusTCPLogger();
}

```

Figure B.12. File "modbus\_tcp\_core.riaps"

# Bibliography

- [1] S. Alepuz, S. Busquets-Monge, J. Bordonau, J. Gago, D. Gonzalez, and J. Balcells. Interfacing renewable energy sources to the utility grid using a three-level inverter. *IEEE Transactions on Industrial Electronics*, 53(5):1504–1511, Oct 2006.
- [2] "Beaglebone". "beagleboneblack-pin-out". <https://beagleboard.org/Support/bone101>. [Online; accessed 15-January-2020].
- [3] S. M. Bellovin. Security problems in the tcp/ip protocol suite. *SIGCOMM Comput. Commun. Rev.*, 19(2):32–48, April 1989.
- [4] H. Beltran, E. Bilbao, E. Belenguer, I. Etxeberria-Otadui, and P. Rodriguez. Evaluation of storage energy requirements for constant production in pv power plants. *IEEE Transactions on Industrial Electronics*, 60(3):1225–1234, March 2013.
- [5] Kyle Benson, Charles Fracchia, Guoxi Wang, Qiuxi Zhu, Serene Almomen, John Cohn, Luke D’arcy, Daniel Hoffman, Matthew Makai, Julien Stamatakis, et al. Scale: Safe community awareness and alerting leveraging the internet of things. *IEEE Communications Magazine*, 53(12):27–34, 2015.
- [6] R H Bube. Photovoltaic materials, Aug 1998. [Online; accessed 15-January-2020].
- [7] Karl W Böer. Survey of semiconductor physics. *Dordrecht : Springer*, v.2 : Barriers, junctions, surfaces, and devices, 1992. [Online; accessed 15-January-2020].
- [8] T. F. S. E. Center. The florida solar energy center (fsec). [https://www.fsec.ucf.edu/en/consumer/solar\\_electricity/basics/cells\\_modules\\_arrays.htm](https://www.fsec.ucf.edu/en/consumer/solar_electricity/basics/cells_modules_arrays.htm), 2018. [Online; accessed 15-January-2020].
- [9] Wikimedia Commons. File:photovoltaik dachanlage hannover - schwarze heide - 1 mw.jpg — wikimedia commons, the free media repository. [https://commons.wikimedia.org/w/index.php?title=File:Photovoltaik\\_Dachanlage\\_Hannover\\_-\\_Schwarze\\_Heide\\_-\\_1\\_MW.jpg&oldid=365803574](https://commons.wikimedia.org/w/index.php?title=File:Photovoltaik_Dachanlage_Hannover_-_Schwarze_Heide_-_1_MW.jpg&oldid=365803574), 2019. [Online; accessed 15-January-2020].
- [10] "C.J. Cowie". "3-phase inverter circuit with wye connected load, drawn by c j cowie using micrografx designer". <https://commons.wikimedia.org/wiki/>

- File:3-phase\_inverter\_cjc.png, Nov 2006. [Online; accessed 15-January-2020].
- [11] "C.J. Cowie". "simple inverter configuration". [https://commons.wikimedia.org/wiki/File:Inverter\\_ckt\\_01cjc.png](https://commons.wikimedia.org/wiki/File:Inverter_ckt_01cjc.png), March 2006. [Online; accessed 15-January-2020].
  - [12] DIY. Types of solar inverter explained — DIY. <https://www.doityourself.com/stry/3-types-of-solar-inverters-explained>, 2019. [Online; accessed 25-November-2019].
  - [13] Y. Du, H. Tu, S. Lukic, D. Lubkeman, A. Dubey, and G. Karsai. Implementation of a distributed microgrid controller on the resilient information architecture platform for smart systems (riaps). In *2017 North American Power Symposium (NAPS)*, pages 1–6, Sep. 2017.
  - [14] Abhishek Dubey, Gabor Karsai, Peter Volgyesi, Mary Metelko, Istvan Madari, Hao Tu, Yuhua Du, and Srdjan Lukic. Device access abstractions for resilient information architecture platform for smart grid. *IEEE Embedded Systems Letters*, 6 2018.
  - [15] Adam Dunkels. Full tcp/ip for 8-bit architectures. In *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services*, MobiSys '03, pages 85–98, New York, NY, USA, 2003. ACM.
  - [16] S. Eisele, I. Mardari, A. Dubey, and G. Karsai. Riaps: Resilient information architecture platform for decentralized smart systems. In *2017 IEEE 20th International Symposium on Real-Time Distributed Computing (ISORC)*, pages 125–132, May 2017.
  - [17] "Electrical4U". "power inverters: What are they & how do they work?". <https://www.electrical4u.com/power-inverter/>, Dec 2019. [Online; accessed 15-January-2020].
  - [18] "Github". "ditributedestimator". <https://github.com/RIAPS/riaps-apps/tree/master/apps-vu/DistributedEstimator/Python>. [Online; accessed 15-January-2020].
  - [19] "Github". "ditributedestimatorgpio". <https://github.com/RIAPS/riaps-apps/tree/master/apps-vu/DistributedEstimatorGPIO/Python>. [Online; accessed 15-January-2020].
  - [20] "Github". "gpiodeviceexample". <https://github.com/RIAPS/riaps-library/tree/master/GpioDeviceTesting/GpioDeviceAppExample>. [Online; accessed 15-January-2020].
  - [21] "Github". "python-tutorial-debug". <https://riaps.github.io/tutorials/debug.html>. [Online; accessed 15-January-2020].
  - [22] "Github". "riaps-implementation". <https://riaps.github.io/impl.html>. [Online; accessed 15-January-2020].
  - [23] "Github". "riaps-index". <https://riaps.isis.vanderbilt.edu/index.html>. [Online; accessed 15-January-2020].

- [24] "Github". "riaps resilient information architecture platform for smart grid-architecture". <https://riaps.github.io/arch.html>. [Online; accessed 15-January-2020].
- [25] "Github". "riaps resilient information architecture platform for smart grid-what is riaps". <https://riaps.github.io/>. [Online; accessed 15-January-2020].
- [26] "Github". "riaps-tutorial-model". <https://riaps.github.io/tutorials/models.html>. [Online; accessed 15-January-2020].
- [27] "Github". "uartdevicetesting". <https://github.com/RIAPS/riaps-library/tree/master/UARTDeviceTesting>. [Online; accessed 15-January-2020].
- [28] "Github". "what is github". <https://github.com/about/diversity>. [Online; accessed 15-January-2020].
- [29] "Red Hat". "what is a middleware". <https://www.redhat.com/en/topics/middleware/what-is-middleware/>. [Online; accessed 20-November-2019].
- [30] D. P. Hohm and M. E. Ropp. Comparative study of maximum power point tracking algorithms. *Progress in Photovoltaics: Research and Applications*, 11(1):47–62, 2003.
- [31] Huang Jiayi, Jiang Chuanwen, and Xu Rong. A review on distributed energy resources and microgrid. *Renewable and Sustainable Energy Reviews*, 12(9):2472 – 2483, 2008.
- [32] "OPAL-RT Technologies Inc System Integration Services Mathworks". "opal-rt simulator". [https://www.mathworks.com/products/connections/product\\_detail/opal-rt-system-integration.html](https://www.mathworks.com/products/connections/product_detail/opal-rt-system-integration.html). [Online; accessed 15-January-2020].
- [33] "Institute of software integrated system Vanderbilt university". "towards a resilient information architecture platform for the smart grids: Riaps". <https://riaps.isis.vanderbilt.edu/pdfs/SGC17-RIAPS-Overview.pdf>. [Online; accessed 15-January-2020].
- [34] "MIT OpenCourseWare". "6.334 power electronics". <https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-334-power-electronics-spring-2007/lecture-notes/ch9.pdf/>, Feb 2017. [Online; accessed 15-January-2020].
- [35] Eswar KonduruJerry Petree. Udp to tcp bridge. *United States Patent*, March 2007.
- [36] "SMA". "sunny tripower 12000tl-us / 15000tl-us / 20000tl-us / 24000tl-us / 30000tl-us". <https://www.sma-america.com/products/solarinverters/sunny-tripower-12000tl-us-15000tl-us-20000tl-us-24000tl-us-30000tl-us.html>. [Online; accessed 15-January-2020].
- [37] "Sunspec-Alliance". "sunspec-protocol". <https://sunspec.org/>. [Online; accessed 15-January-2020].

- [38] "Sunspec-Alliance-Model". "sunspec-model". <https://sunspec.org/sunspec-information-model-reference/>. [Online; accessed 15-January-2020].
- [39] W R Swiegers and Johan H. R. Enslin. An integrated maximum power point tracker for photovoltaic panels. *IEEE International Symposium on Industrial Electronics. Proceedings. ISIE'98 (Cat. No.98TH8357)*, 1:40–44 vol.1, 1998.
- [40] Andrew Tar. Database distribuiti e decentralizzati, cosa sono? <https://it.cointelegraph.com/explained/decentralized-and-distributed-databases-explained>, Jan 2018.
- [41] "Techtarget". "osi model (open systems interconnection)". <https://searchnetworking.techtarget.com/definition/OSI>. [Online; accessed 15-January-2020].
- [42] "Techtarget". "tcp (transmission control protocol)". <https://searchnetworking.techtarget.com/definition/TCP>. [Online; accessed 15-January-2020].
- [43] Wikipedia. Giunzione p-n — wikipedia, l'enciclopedia libera. <https://en.wikipedia.org/wiki/P%E2%80%93junction>, 2019. [Online; accessed 15-January-2020].
- [44] Wikipedia contributors. Solar inverter — Wikipedia, the free encyclopedia. [https://en.wikipedia.org/w/index.php?title=Solar\\_inverter&oldid=917932310](https://en.wikipedia.org/w/index.php?title=Solar_inverter&oldid=917932310), 2019. [Online; accessed 24-November-2019].
- [45] "David Williams". "understanding, calculating, and measuring total harmonic distortion (thd)". <https://www.allaboutcircuits.com/technical-articles/the-importance-of-total-harmonic-distortion/>, Feb 2017. [Online; accessed 15-January-2020].
- [46] Dale Willis, Arkodeb Dasgupta, and Suman Banerjee. Paradrop: a multi-tenant platform to dynamically install third party services on wireless gateways. In *Proceedings of the 9th ACM workshop on Mobility in the evolving internet architecture*, pages 43–48. ACM, 2014.