POLITECNICO DI MILANO

SCHOOL OF INDUSTRIAL AND INFORMATION ENGINEERING

MASTER OF SCIENCE IN

ELECTRONIC SYSTEMS ENGINEERING

DESIGN AND DEVELOPMENT OF A SOFTWARE TOOL FOR THE MANAGEMENT OF ABB'S EKIP CONTROLLERS

Prof. Francesco Castelli Dezza

Julio Wladimir Torres Tello

Mat. 780320

December, 2013

Acknowledgments

First of all, I would like to thank Prof. Enrico Ragaini for giving me the opportunity to develop this project, for all his support, advise, patience and dedication to this work; because without his guidance and knowledge I could have never gone through it. Also, I would like to thank Prof. Francesco Castelli Dezza for agreeing to be my advisor, for his time and advice. Likewise I am very thankful with ABB company for giving me the chance of elaborating this work, as well as facilitating me its equipments, information and installations; and to the two people in the company who got involved in this project and have collaborated to its realization with time, ideas and support: Luca Omati and Enrico Amistadi.

Furthermore, I would like to thank Politecnico di Milano and its staff for making it possible for me to complete this Masters degree. I am grateful for the knowledge and experience that I have gained this semesters studying here. And of course my sincere thanks to Italy, the country which opened its doors to me in the best way, providing me support and motivation.

Finally I would like to thank to all the ones who contributed in some way to the achievement of this degree and not only in the academic aspects. Thanks to my family for all the emotional and financial support and to my friends in Ecuador for encouraging me even if it has been from far away; and of course big thanks to my friends in this country, who have become my family and have made my time in here remarkable.

A mis padres Julio y Adita,

a mi abuela Dolores

y a mi hermano Juan Carlos.

Table of contents

Index of figures

Index of tables

Abstract

This work is motivated mainly by the the **worldwide growing energy demand** which is an issue that governments, industry and research institutions have to face constantly, making mandatory for them to adopt strategies that provide a more **efficient use of electric energy** instead of just increasing the generation capacity. In that sense, **ABB**, a world leading company in power systems solutions, has found an economic and effective answer for the management of electric loads, by using the **Ekip controller**, which makes use of a patented algorithm to decide whether a load in an electric system must be powered or not at a defined time instant, according to given power limits. Moreover, ABB has the capability of calculating recommended power limits for an electric installation, based on the generation schemes and the prices of electricity at a given moment.

However, these controllers require a lot of manual intervention and they have a limited number of power limits (four per day – that can remain unchanged for many days) that can be assigned to an electric installation. Therefore this project is focused in the management of these controllers using the software tool here developed. In fact, the goal of the project is to **develop an interface** between the information available at ABB about power limits and the Ekip controllers, specially the way in which they act over the electric installation; providing a power limit profile that would have now a higher level of granularity in time and an adaptive behavior, since the power limits would be now updated every fifteen minutes and based in the constantly updated input information.

This software tool, called **Ekip Management Software** (EMS) is the final product of this project, and it is a program developed in Python which reads the power limits information from XML files, processes the data and configures the controllers over an IP network; and also reads information about the status of the controllers in order to generate log files. This allows a user to manage a number of controllers at the same time by only keeping the software running having as a result a more accurate power limit profile that the one that could be achieved by manually configuring the controllers.

Riassunto

Questo lavoro è motivato principalmente dalla continua crescita del bisogno di energia a livello mondiale: un problema che le istituzioni dei governi, dell'industria e della ricerca devono affrontare costantemente rendendo obbligatoria l'adozione di strategie che prevedono un uso più efficiente dell'energia elettrica invece del semplice aumento della capacità di generazione. In questo senso ABB, leader mondiale nelle soluzioni per Sistemi di Potenza, ha trovato una risposta economica ed efficace per la gestione dei carichi elettrici utilizzando il controller Ekip. Esso utilizza un algoritmo brevettato per decidere se un carico di un certo sistema elettrico debba essere connesso o no alla rete in un istante di tempo definito seguendo determinati limiti di potenza. Inoltre ABB ha la capacità di calcolare i limiti di potenza consigliati per un impianto elettrico sulla base degli schemi di generazione e dei prezzi dell'energia elettrica in un momento dato.

Tuttavia questi controller richiedono di un certo grado di intervento manuale e hanno un numero ristretto di limiti di potenza assegnabili ad un impianto elettrico (quattro al giorno con possibilità di rimanere immutati per molti giorni). Il progetto è quindi focalizzato sulla gestione di questi controller utilizzando lo strumento software sviluppato. Lo scopo finale è infatti lo sviluppo di un'interfaccia tra le informazioni disponibili ad ABB sui limiti di potenza e i controller Ekip, specialmente il modo in cui agiscono sull'impianto elettrico fornendo un nuovo profilo limite di potenza con maggiore livello di granularità nel tempo e comportamento adattativo. I limiti di potenza saranno ora aggiornati ogni quindici minuti in funzione dell'informazione in entrata costantemente aggiornata.

Questo strumento software, denominato Ekip Management Software (EMS), é il prodotto finale del progetto. E' un programma sviluppato in Python che legge l'informazione sui limiti di potenza da un file XML, ordina i dati e configura i controllori su una rete IP; legge inoltre l'informazione sullo stato dei controller per generare dei log files. Questo consente all'utente di gestire un certo numero di controller allo stesso tempo solamente mantenendo il software in esecuzione: il risultato è un profilo di limite di potenza più accurato di quello che potrebbe essere realizzato configurando manualmente i controller.

1 The Energy Efficiency Problem

"We can't have an energy strategy for the last century that traps us in the past. We need an energy strategy for the future – an all-of-the-above strategy for the 21st century that develops every source of American-made energy." -

<Barack Obama> ►[The White House, March 15, 2012]

Summary. <The continuously-growing energy demand is a worldwide problem that governments, industry and research institutions have to face day by day, making mandatory to adopt strategies that provide a most efficient use of energy (electric in our specific case), instead of just increasing the generation capacity. These new strategies are possible thanks to the new technologies available nowadays.>

1.1 The Electric Power Systems

We can trace back the origins of the electric power systems to 1882 when Thomas Edison put into operation the steam-electric plant of "Pearl Street Station" in New York; station that managed to serve five hundred customers using more that ten thousand electric lamps. In the same year, the first DC transmission line was built in Germany over a distance of around sixty kilometers. **[Casazza, p.1, 2][i].** Initiating in this way the history of a new energy system so successful that started a second industrial revolution in the world; a revolution that has not yet stopped and has been supporting the economic growth in Europe and the US since then, and other new economies in more recent years. However, given that this technology has been so effective in satisfying all the industry needs for decades, it hasn't changed so much since its beginnings and most of the same basic principles are still applied now.

However, more recently, the word “efficiency” is taking a stronger meaning in the electric energy filed, given the new considerations that arise from many points of view such as environmental, economical, political, etc. Efficiency is often defined as the ratio of a system's “output” to its “input”, and so it determines the quantity of resources (input) that are being unused or “wasted” and that have the potential to be also transformed into the mentioned output, or electric energy.

In electric systems, since the “input” or let's say the fuel, has been not a limitation because it has always been available, inexpensive and environmental issues haven't been really considered; the systems' efficiency has not been a real concern for a while. But now, the time has come when the world needs to change this scheme, making it more efficient; now we have environmental and economic concerns that are urging the governments and industries to change their conceptions about the electric systems; in that sense many institutions are working in the development of new strategies, policies and technological solutions that can make it happen. Regarding these considerations, there are some key factors that are necessary to remember, because they can limit these new developments, or they can be seen as opportunities for making this change possible; these are:

- **Transformation.** Electric energy can be transformed from and to other energy forms, giving it multiple uses but also many alternatives for its production.

- **Storing energy.** Despite all the current efforts and new technologies, nowadays it is still not possible to store big amounts of electric energy. In general, it must be used at the instant that it's produced.

- **Transmission.** One of the main advantages of electric energy is that it can be produced in one location, and instantaneously transmitted to another one.

- **Interconnection.** There are economic and technical benefits when we interconnect electric systems to provide mutual assistance, because in this way we reduce the number of backup plants that we need and improve the overall response to failures.

- **New technology.** In the last decades, there has been a great development in electronics, automation, informatics, and telecommunications, providing a lot of new options for the management and control of electric power systems.

1.2 Environmental Concerns

In the past few decades, we have watched, read and herd more and more news about unusual and unexpected meteorological phenomena around all over the world, from stronger droughts and floods, to the biggest hurricanes in history; and also how pollution affects life on Earth, both human and wild life. In the last days I have read news about studies showing that human life span drops in areas where air pollution is higher, even for the same kind of population, in the same country **[NYTimes.com]**[ii]; and also how endangered wildlife is, when I read in another study, how evolution is thousands of times slower that it should in order to adjust to climate changes, predicting that we will see extinct many species by the end of the century, especially in tropical areas (the most biodiverse) **[Awescience.com]**[iii].

And although scientific community has been saying for many years that the cause of these already occurring phenomena, and the ones to come is the pollution caused by us, due to our consuming habits that demand every time more and more energy to pursue our daily activities, most people related to the energy sector have not accepted this issue until recently. Now we see that global warming has become one of the biggest concerns for the energy sector (among others), including government, industry, and the society in general. Examples of this change are for example when we saw in June this year that the US government, which has always been reluctant to reduce carbon emissions, released a plan to cut carbon pollution saying that they also have the responsibility of leaving a cleaner and less damaged planet for future generations. In addition, many institutions including universities, the private sector and government agencies (especially in Europe) have been working in wide-range solutions, in order to reduce the energy consumption, not only by limiting its usage, but also by using this energy in the most efficient way. We say that these are “wide-range” solutions because they imply not only the technical aspects, but also a strong regulatory support, an effective business strategy and a change in the consumer mentality towards the environment.

<u>Energy consumption:</u>

The main problem here arises from the fact that the world keeps increasing its energy consumption no matter what, as we can see in the International Energy Agency Statistics[1] graphics (figures

1 https://www.iea.org/

1.1 and 1.2) **[IEA, p.32, 47]**[iv] . For the last decades, the fuel (needed to produce this energy) consumption has been increasing constantly, especially in the developing countries in Asia, such as China or India, impulsed by an accelerated economic growth that demands more energy for the industries and for the increasing consuming habits of the society that enjoys this new economic capability. The problem with this new habits and growing energy demands is that also the CO2 emissions are growing fast, affecting every time more and more the already fragile environment.

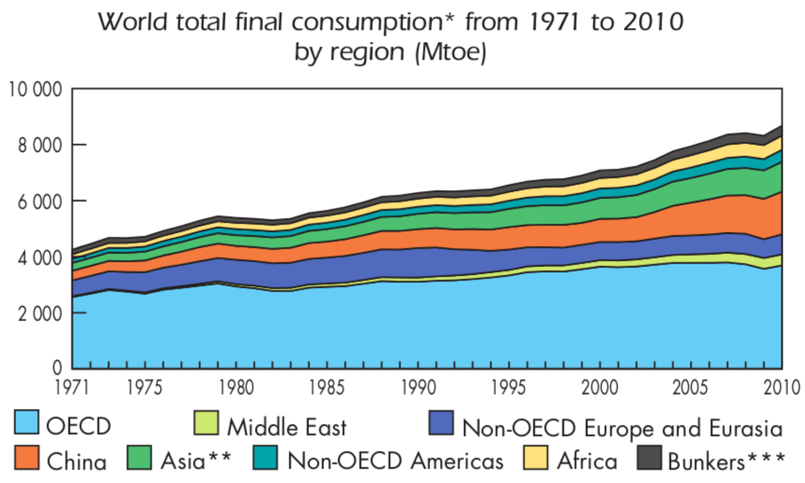


Figure 1.1: World total energy consumption

One of the challenges of this century is to reduce at maximum these carbon emissions, by reducing the consumption of energy that is produced by burning fuel, using environmentally friendly energy generation methods (most renewable energies), but also by using the available energy in the most efficient way.

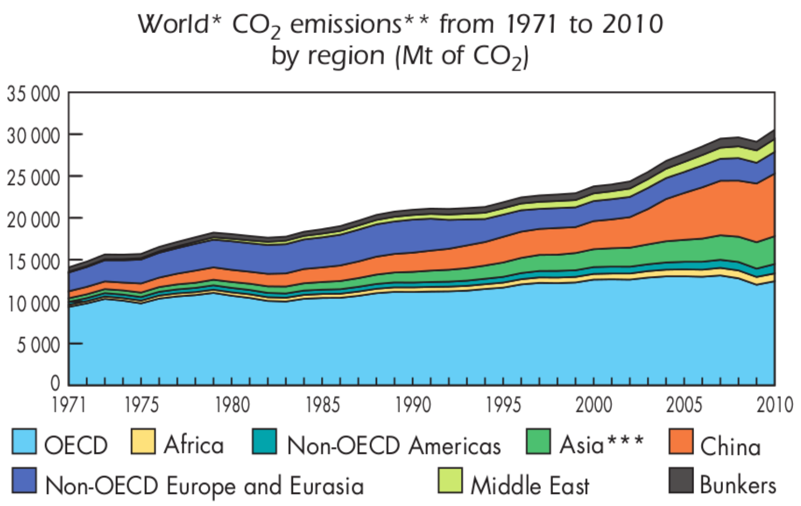


Figure 1.2: World CO2 emissions

1.3 Renewable Energies

With the current model of human development, reducing the consumption of energy is basically impossible, but what we can do is to reduce the kind of generation that affects the most to us and to the environment, which is the one produced by burning fossil fuels. But if we do it so, in order to satisfy what the world demands everyday we need to replace the source of energy generation by the usage of alternative methods that can be considered as "environmentally friendly"[2], which are most of the "renewable energies"[3]. Here we should note that renewable doesn't necessarily mean environmentally friendly, since for example we can say that wood is a renewable source given that we can always plant more trees and replenish our energy source, but it doesn't mean that this

2 Meant to reduce, or to cause minimum or no harm at all to the environment.

3 Also called "sustainable"; it means that it is generated using resources that can be constantly replenished.

is an environmentally friendly solution mainly because it will anyway discharge big amounts of CO2 into the atmosphere in the process, but also by means of using it there could be a devastation of a primary forest, affections to the local flora and fauna, to the human health and so on. Therefore, we should consider using some alternatives that have been proving effective results, such as:

- **Hydro-power.** This is one of the most popular options in terms of electric generation, and has been in massive use for many decades. This kind of generation works under the principle that potential energy is stored in the water, that is usually trapped by building dams in large rivers, where the water is released according to the generation needs, in order to move turbines that will produce electricity in the generators. The cost of producing hydroelectricity is relatively low, making it a very competitive alternative to fossil fuels.

- **Thermal Solar.** Solar radiation is another renewable, free source of energy that can be used in different ways and that has been used since ancient ages. The principle of modern thermal solar electric generation is that the radiation from the sun heats a fluid (water or oil) which flows inside pipes, to get and storage the energy. Among its advantages we have that in its simplest form[4], there is no need for hi-tech systems, and it can be used to heat water or for climatization in small, isolated and/or poor regions.

- **PV Solar.** Photovoltaic Solar generation is another, more sophisticated way to use solar radiation. In this case, radiation

4 In its simplest way, this source of energy can be used as thermal itself, without converting it in to electricity. Actually, it is being used in that way in developing countries such as Brazil. You can read an article about it in The Ecologist:"How to Make a Solar Water Heater from Plastic Bottles."

from the sun is transformed into electric energy by means of semiconductor PV panels that exploit the photovoltaic effect. This is a technology that hasn't been fully adopted so far because there are still some doubts about its efficiency, costs, and lack of knowledge about its duration, and how to deal with the disposals from the PV panels after their service life concludes. Most critics focus on the fact that the source (solar radiation) is not always available, so PV is characterized by an intermittent service, and hence it needs connection to the grid, or storage systems (expensive) and that it represents a high initial investment. On the other hand, we can say that it is available at daytime and stronger around noon and afternoon (industry pick hours) when electricity is usually more expensive, and that by means of storage systems (EV cars, thermal systems, batteries) this stored energy can be sold back to the utilities when it is the most expensive, reducing costs. Also it's a long term investment that will pay back after some time, considering that the "fuel" is free and they require little maintenance.

- **Wind Power.** This is another old source of energy, with a history that comes from the Dutch windmills. Nowadays the same principle is applied but not to mill grains, but to produce electricity; where the wind is used to drive a turbine that drives a generator. As a disadvantage we can mention that since the wind is not constant anywhere, the rotational speed is not constant either. However, being this a very mature technology, good solutions have been found using appropriate generators as well as advanced control systems. These automatic systems can calibrate the angle at which the generator faces the wind and the angle of each blade in the generator, increasing the efficiency. This is a very popular technology since it is economic and has proven good results, and in recent years it has been moving also to the sea, where new wind farms are being installed off shore.

- **Biomass.** In this case, just like petroleum or coal, we use another carbon-based fuel, but the main difference is that in this case the fuel is renewable. Biomass can come from fisheries, farms, as well as from industries and municipal waste. It can be of different types: Solid (nutshells, chips, etc), biofuel (animal fats, vegetable oils, etc), biogas (manure, sewage, etc.). Usually the costs of implementing this kind of plants is not high, and there can be small implementations, which makes it a good option in rural areas, and good for distributed generation[5] schemes. However, we should consider that this system of producing electric energy has some problems like a smaller but still present CO2 emissions, deforestation and food-energy competition[6].

- **Geothermal energy.** The first geothermal power plant to work and show the industrial value of this kind of energy was installed in Larderello, Tuscany in 1904 **[UGI[7]][v].** A power plant of this kind works just like a fossil fuel plant, in the sense that it uses steam to move a turbine; but in this case the steam is got directly from down the ground. This is a good alternative in regions where there is volcanic activity, and the best

5 Unlike most traditional solutions that use large centralized power plants along with enormous transmission systems; in distributed generation schemes, the power plants are widespread around the places where energy can be generated and probably used. The last are usually smaller and use renewable energies.

6 Food-energy competition arises from the fact that some of these biomass fuels are produced by harvesting potential food, like soy, sugar cane or corn; and if it is the case that fuel production is more profitable than food, it could threated the food provisioning of the population in that area.

7 UGI: Unione Geotermica Italiana.

example is Iceland, where geothermal energy generation has increased significantly in recent years and now it provides 25% of the country's total electricity **[NEA[8]][vi]**.

Although there are many options regarding renewable and/or environmentally friendly solutions, there are still a lot of questions about performance and costs; and the combined usage of these new solutions along with the traditional systems bring new problems and concerns, where coordination in both design and operation is fundamental, as well as the optimization of these systems, to mitigate congestion, minimize effects of periods of unavailability, and storage limitations.

1.4 Economic Concerns

The increase in the oil prices contributes to the economic crisis, especially in the developed countries that depend on this resource to keep their life standards; this because the profit margins for the companies are lower and lower and people invest more money in heating and mobility, making them spend less in other goods. All of this contributes (along with many other issues) to the major economies to grow slowly or enter into recessions, as we have seen in recent years. This situation doesn't find a relieve if we consider that electric energy prices increase constantly; in Italy for instance, the electricity prices have been increasing in the last years. If we consider the domestic context for example, they show and increasing tendency since 2004.**[Napolitano, p.20]**[vii]

This situation makes urgent the need for finding alternatives to the dependency on petroleum and coal, but also the need of making electric systems more efficient. If energy becomes so costly, we cannot afford wasting it at all levels. There are campaigns

8 NEA: National Energy Authority, Iceland.

focused on the users trying to push them to use electricity in a better way, but that is not the only solution, and considering all the technology available nowadays, it becomes obvious to have automatic systems that can be programmed to control the electric consumption in such a way that the resources are efficiently used, with the final goal of having a sustainable economy[9].

1.5 Smart Power

As mentioned before, one of the challenges of this century is to reduce at maximum the carbon emissions; and another way to do this besides the use of renewable, environmentally friendly energy sources, is by using the available energy in the most efficient or let's say “smart” way, what can be denominated “smart power” since its application can lead to very promising results in terms of saving both pure power, and money; and as an example here it's quoted the following paragraph from an on-line professional forum: *“On average power plants consume up to 7% of the electricity they generate, while industrial sites account for around 33% of global energy use and buildings account for nearly 40% of energy consumed. These figures could be cut by 10% to 30% by optimizing the various processes and systems that run the plants”* ***[Ferrera]***[viii].

This optimization in the electric system that is the key factor of what here is called “smart power”, is currently related to the concepts of “smart grids” and “virtual power plants”, and so it is useful to develop a little bit more these concepts:

- **Virtual Power Plants.** A Virtual Power Plant (VPP) is an arrangement of distributed generation installations, where the electric power can be generated by different means, such as

9 An economy in which the resources are not spent faster that they can be renewed.

wind, solar, traditional means, etc., that is managed by a centralized control entity. This arrangement usually shows economical and technical benefits. As an example, it's quoted a comment in a previous thesis work in this topic: *"In previous studies* (mentioned in that work) *we can see how using VPP's with the right amount of energy dealt, it can lead to profit margins that can justify the usage of this kind of solutions". **[Napolitano, p.55]***[ix]

- **Smart Grid.** As a general concept it can be said that a smart grid is defined by: *"Combing time-based prices with the technologies that can be set by users to automatically control their use and self-production, lowering their power costs and offering other benefits such as increased reliability to the system as a whole". **[Fox-Penner, p.34]***[x]

As stated in the definition, there are two main points in the development of smart grids, which are the time-based prices scheme and the new technologies. As for the first, prices are used to motivate the customers to keep their power use below certain margins preventing the whole system to exceed the installed capacity, especially during the most critical hours. On the other hand, when talking about technologies there are many key components that have been developed following different trends, and that have now allowed us to make a more efficient management of the electric networks, among which we can mention:

- **Smart meters.** These kind of meters are able to work with smart appliances that can be configured to make a more efficient use of the energy, according to price changes, given that the cost per kilowatt-hour varies at various times of the day.

- **Storage.** A key component that is still in early stages, very important since it would allow us to change the generation – usage ties, giving us more freedom in terms of when we can generate energy is not necessarily when we have to use it. Thermal, compressed air, flywheels are the current technologies in this area, and new developments like electric vehicles (EV) with incorporated energy storage units are increasing the electric network storage capacity.

- **Information, communications and control networks.** These are a fundamental component of smart grids, allowing real time calculations and management of the entire systems (now cheap and ubiquitous, most systems are already or in the process to become digital, so easier to integrate in these networks) . They introduce the possibility of having a duplex communication, in which the electric company should know not only the total amount of power used by all its customers, but a more detailed information of quantity and quality of this usage.

Besides this, there are other concepts that should be introduced here, to have a better understanding of the topic:

- **Demand response.** The new technologies, such as sensors and telecommunication networks bring the possibility of managing the customer power demands in terms of the available supply conditions, where the electric supply to some services that might not be so critical[10] can be reduced or eliminated for certain amounts of time, according to pre-planned schemes if the situation reaches a critical

10 Usually the ones related with temperature status (refrigeration, cooling, heating) since these conditions can remain practically unchanged for small amounts of time even if no power is applied, unlike the management of lights or motors that require a permanent supply.

condition.

- **Load aggregation.** Referred to either final users or actual electric loads which are joined together in order to make a more efficient use of the available power and/or to reach a better economical agreement with the upper level in the electric market scheme (see 1.6 The Market). Aggregation usually decreases the costs involved in dealing with energy suppliers, but it does not necessarily result in lower costs for the energy itself.

- **The Smart Integrator.** It is a utility that operates the power grid and its information and control systems but does not actually own or sell the power delivered by the grid. Its mission will be to deliver electricity with superb reliability from a wide variety of sources, from upstream plants to in-home solar cells, all at prices set by regulator-approved market mechanisms. **[Fox-Penner, p.175]**

In the end, the smart grid will give customers much more control over their own power use and make dynamic pricing a universal condition in electric systems.

1.6 The Market

In order to find the most viable solutions, we have to understand how the electric market works, and which part of it is the one we have to focus on.

In the Italian case, there was a process called "deregulation or liberalization"[11] that took place 10 to 15 years ago and was similar

11 Process that stimulates a free market concept when it refers to the purchasing

to other processes happening all around Europe. As a result, this deregulation divided the electric system into four separate actors:

- **Generation.** Involves the electric generation systems, no matter by which means, here there can be included fossil fuels burning plants, renewable energies, nuclear power, hydro power, etc.

- **Transmission.** The process of transmitting the electric energy generated from the power plants towards a closer point to the final customer. It comprises the transmission lines, DC or AC, for long or short distances.

- **Distribution.** This is the most "technical" part, and covers the process of receiving the energy from the transmission lines (HV), convert (transform) it into a lower voltage that is going to be sold to the final customers. The distribution system is controlled and operated by a SCADA (System Control And Distribution Automation) system at the dispatch center.

- **Retail.** Constitutes the process of selling the electric energy to the final user. It is basically an intermediary. They are also called utilities.

This concept is widely applied in Europe consenting small variations in each specific case; and especially for the last two aspects, there are some variations in the models in some countries such as Germany and Turkey.

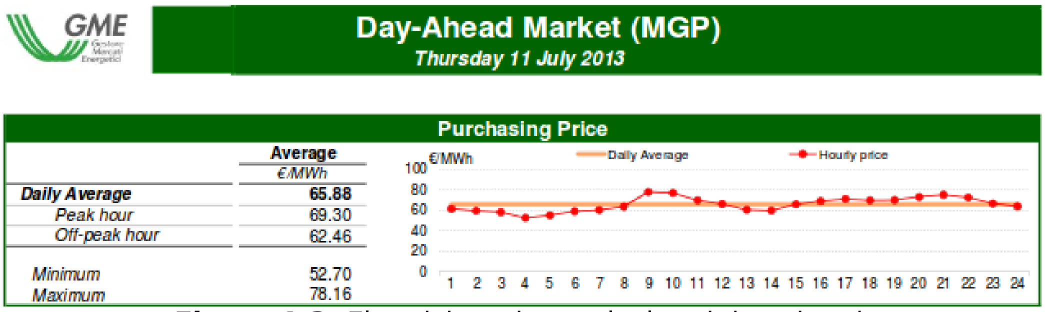
---

of electric energy among the four different actors. Officially started in 1999.

Consumption forecast:

Given this model, “consumption forecast” becomes the most important factor since the energy prices are not fixed, and they vary according to a wide range of factors, including the geopolitical situation, the weather, reliability of the systems, seasonality, and of course demand. In the case of this project, there is more interest in the especial case of this situation when it is happening in the last two parts of our market model (distribution and retail).

Going backwards, the retailer is the first one to enter into action. It has to forecast the consumption of energy or better, the future demand of electric energy from its users; and it has to be done for a variety of time periods, going from months (3, 6, 12) down to seconds. What is done here is that the retailer creates a “price band” with the expected maximum and minimum values that it will need during a certain period (as long as possible), in order to satisfy its customers demand, using this information to buy energy from the distribution company at a good price. Later, this forecast will be adjusted and the retailer will buy (or sell if possible) more energy if needed in week-ahead, day-ahead plans, or even in shorter times. The former has to be avoided as much as possible, since in those cases, the energy is usually much more expensive; and that is why a precise forecast is wanted.



	Average €/MWh
Daily Average	**65.88**
Peak hour	69.30
Off-peak hour	62.46
Minimum	52.70
Maximum	78.16

Figure 1.3: Electricity prices calculated day-ahead.
“GME - Gestore Dei Mercati Energetici SpA.”

Those last minute acquisitions are much more expensive, mainly because of the generators limitations: One cannot turn on and off easily a power plant in short times. In that case, the market has been divided according to the response times, in three:

- **Primary.** The most expensive, but it could respond in seconds.
- **Secondary.** Also expensive. These generators would be able to respond in a relatively short term (around 2 to 5 minutes).
- **Tertiary.** The cheapest one. It should take around half an hour to respond.

Now, if we consider the limitations in the side of the distribution, we can say that they are almost the same that for the retailer (the ability of the generators to respond fast), but also we have to consider here that the transmission company can establish some limitations in the sense that transmission lines are not able to support an infinite power, and they limit the maximum power that can be carried towards each sub-station (distribution).

Consumption peaks:

In both cases (retail and distribution), besides consumption forecast, there is another solution to avoid these unexpected, expensive, last-minute energy purchases, and this is saving the last-minute required energy. And one especial strategy is to **cut (or shave) the consumption peaks**.

Here, we also have to take into account the fact that the rates at which the customers consume the electric power is not usually measured instantaneously, but are calculated as an average level of usage during certain amounts of time; being 15, 30 and 60 minutes the most common used time intervals. **[Casazza, p.44]**[xi]

There are studies showing how this peak-shaving strategy can affect the final performance of the electric power system as a whole. For example, the top 6% of the electric capacity in France is used for only 1% of the time during one year, as a clear display of an overestimation of the system that could be avoided by means of a peak-shaving strategy. Also, studies show that customers could pay less if they can use this technology; for example, there is an average 23% reduction in pilots conducted in the US. **[Amistadi, p.41, 43, 45]**[xii]

2 Power Management

"As members of the European Community, we have the commitment to fulfill the overall 20/20/20 targets: they are referred to as the reduction of 20% in energy consumption thanks to energy efficiency techniques, to the reduction of 20% in carbon footprint emissions, to the increasing of 20% of the energy generated from renewable sources." -

< Flavio Ferrera> ►[ABB Conversations, July 13, 2013][xiii]

Summary. <ABB, a world leading company in power systems solutions, has found an economic and effective answer to the need for an efficient management of electric loads, by using an "absorbed-power" based controller that utilizes a patented algorithm to decide whether a load must be powered or not at a defined time instant. In the presented project, the goal is to manage some of these controllers using the software tool here developed, in order to make them work in such a way that the power limits profile would be closer to the information available at ABB, based on the always changing electricity prices.>

2.1 ABB Company[12]

ABB is a world leading company in power and automation technologies. Based in Switzerland, the company employed 145,000 people in 2012 and currently operates in approximately 100 countries. The company resulted form the merge of ASEA (Sweden) and BBC (Switzerland) in 1998, companies with a rich and successful

12 All the information about the company was taken from the official website: http://new.abb.com/about

history in the implementation of innovative solutions in power systems and automation, success driven particularly by a strong focus on research and development. Nowadays, the company maintains seven corporate research centers around the world and has continued to invest in R+D through all market conditions. All these characteristics make of ABB the largest supplier of industrial motors and drives, the largest provider of generators to the wind industry, and the largest supplier of power grids in the planet.

ABB’s business comprises five divisions that are focused on particular industries and product categories; these are: Power products, power systems, discrete automation and motion, low voltage products, and process automation.

In the specific case of the Power Systems division[13], the one that is interested in the realization of the project developed in this work, ABB offers solutions for traditional and renewable energy based power generation plants, transmission grids and distribution networks; solutions that play a very important role in the optimization of electric generation and the evolution of more flexible, reliable and smarter grids, which has been stated as one of the main goal of this project.

2.2 The Ekip Controller

Considering the current efforts of all the involved actors for the optimization of electric systems and the evolution of more flexible, reliable and smarter grids, and given the derived need for a more efficient management of the loads in an electric system, ABB has found an economic and effective solution to this

13 For more information: http://new.abb.com/about/our-businesses/power-systems-division

load-management problem by means of the already in place electronic trip unit used for protection against overcurrents, as an **“absorbed-power” based controller (so called Ekip)** that uses a patented algorithm to decide if a given load must be powered or not at a defined time instant. As one can infer from this short description of the Ekip controller, there are two main concepts involved in its operation, which are the physical connections and the used algorithm; concepts that are described in the following subsections.

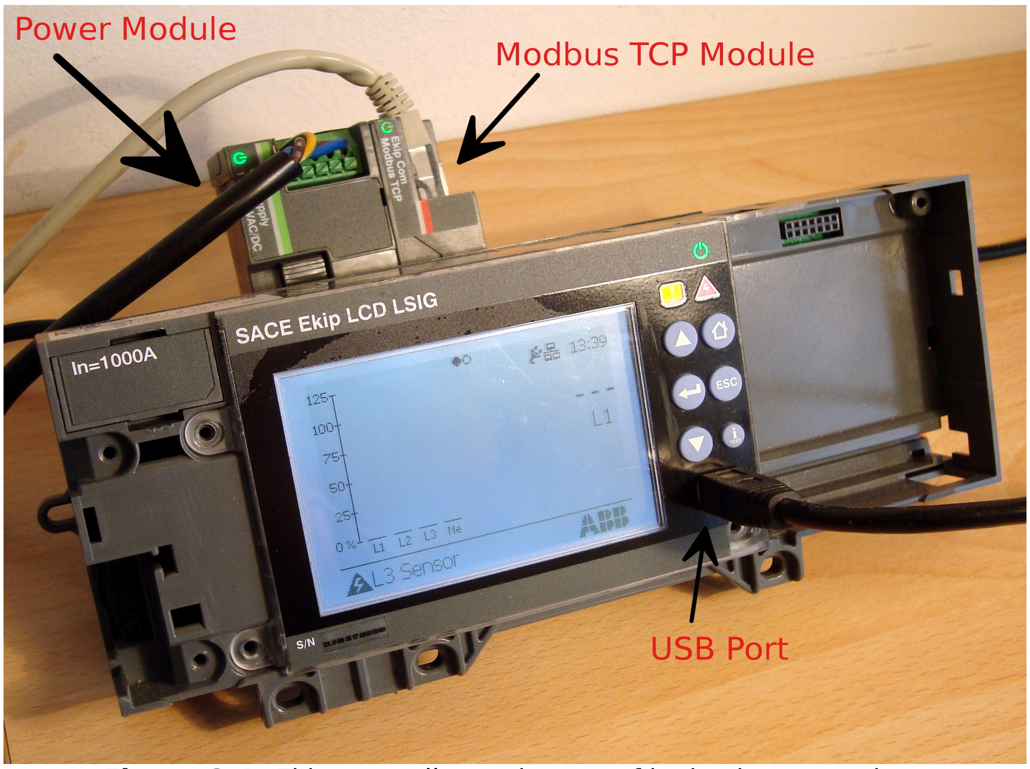


Figure 2.1: Ekip controller and some of its basic connections

<u>Physical connections:</u>

As it was mentioned before, the Ekip controller needs one circuit breaker (used as overcurrent protection) that has the capability of acting over a number of electric loads by **connecting or disconnecting one or many of them at a given time** (based on the patented algorithm that will be described later). This one has to be the main circuit breaker on the LV plant of the user.

In order to accomplish its tasks, the Ekip controller must constantly measure the energy absorbed by the whole installation, and with this information (energy / power) and the proper settings that the user has introduced in the system configuration, it will be able to remotely connect or disconnect a given load. At this point, it has to be said that the controller will not only manage the passive loads that are “consuming” energy, but **it can also connect the reserve generation device**, that will be understood as a “negative load”.

To obtain the information about the consumed energy, the controller uses the voltage and current measurements that the trip unit must keep in track (even to accomplish its primary function as a protection device). As for the other part of the required information, the content given by the user can be set directly on the device through the interface that it has and that can be different according to the specific model of the Ekip controller (i.e. touch screen, or keys and simple screen), otherwise it can be entered through a computer (using USB communication)[14] or by means of serial or Ethernet ports. In the last cases, the user should know the communication procedures for reading and/or writing the data, and the registers that have to be modified in the Ekip controller (all defined by the manufacturer). This situation (Ethernet communication)[15] is the one that has been used in order to develop the software tool that is the main product of the project here presented.

14 In this case, there is a software tool developed by ABB, that will help the user to configure the controller, but the user could also send Modbus commands to the device, in order to configure it. The situation will be similar to the one when using serial or Ethernet connections.

15 Defined by the Modbus TCP standard.

After the information has been collected, and the algorithm has made the decision (procedure that will be fully described in the next subsection) about the state of the loads, the Ekip controller has to remotely manage the loads, and to perform this task, there are two options that can be used:

- By means of a wired solution, in which the electric loads would be directly controlled by acting on the opening/closing shunts or on the motor operators of a given load.
- Through a dedicated communication system.

To summarize, in a LV installation where there's the need to apply this kind of control, there have to be the following elements:

- The Ekip controller, that will be acting as the main circuit breaker. It includes the devices to measure the power consumption.
- The human-device interface, that will allow the user to configure the controller.
- Up to 15 loads (maximum for this controller) that will have each one a connection/disconnection mechanism.
- The “communication” network, either intrinsic wired or dedicated.

The load control algorithm:

As the second main concept, we have the **patented** load control algorithm, the “brain” of this system; since it is the one that makes possible the data processing (both measured and configured by the user), providing a real-time, power-absorption based control scheme. This algorithm is based in the following four-step procedure:

1. **Measurement.** As previously mentioned, the Ekip controller needs to collect data as a first step, in which the circuit breaker measures the total power through it and integrates this power in one time period (which has been previously defined by the user and it is usually 15 to 30 minutes – as a part of the other component of the required data), obtaining the value of the energy used during that period. This measurement is reset to zero every time interval, so that one can obtain the energy for each one of this time slots.

2. **Synchronization.** Since the variable time plays a key role in the calculations made by the algorithm, there is the need for a synchronization mechanism that handles this variable. For this purpose the Ekip controller has an internal clock that will administrate the configuration and usage of the mentioned time periods (defined by the user). As an alternative, there's also the option of using an external signal that provides synchronization to the controller, usually provided by the utilities.

3. **Evaluation.** The most important “step” in this procedure is the evaluation of the scenario in which the controller operates at a given time instant and the possible actions that it could perform, providing to it the capability of an efficient

management of the electric loads. In this situation, the algorithm calculates the current scenario and predicts whether the total mean power will overcome the preset limits or not. This prediction is done by dividing the plane t, E into three sectors (scenarios) limited by the curves Emax and Emin, as shown in the graphic **[ABB SACE, p.8]**[xiv]:

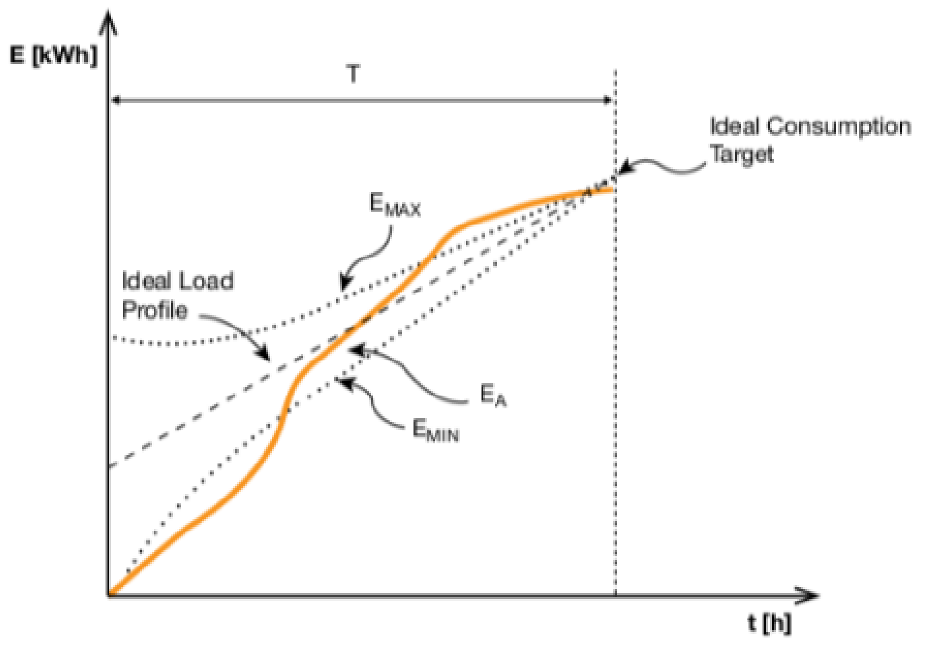


Figure 2.2: Energy consumption curve and limits.

The calculation is performed by **measuring the instant power through the circuit breaker** and obtaining the scenario (region in the plane) where the power (or energy) consumption is located, and depending on the location, there are three alternatives:

- If the consumed energy is **within the two lines** (limits) Emin and Emax, the decision will be to maintain the actual load configuration.

- If the consumed energy is **remarkably below the**

curve Emin, the decision will be to increase the number of loads that are connected.

- And finally, if the consumed energy is **over the curve Emax**, the decision will be to decrease the load configuration, it means to disconnect some of the loads. This is the mos critical situation, since it's the one that leads towards the savings of electric energy consumption.

4. **Load Management.** After evaluation and as a result of that step, we have some decisions that were made about the future scenarios and now those decisions have to be somehow applied to the electric system. This happens by means of the connection/disconnection mechanisms in the power lines of each load, and following some rules. This process involves another level of decision, in which the loads are connected or disconnected according to three criteria, which are:

   - **Priority.** After the next scenario has been defined, and when there's the need for connection or disconnection of electric loads, if there are two or more possible loads over which the controller can act; in order to decide which one it will connect or disconnect there's a priority table defined by the user that will help to the controller to choose the next load to be connected/disconnected. However, this decision must take into account also the other two criteria, which are useful in the case when there are different loads with the same priority.

   - **Respect times.** There are some loads, that given their nature, cannot be powered and unpowered (or vice versa) instantaneously[16] or on the other hand, the affectation of these loads when they are not working for some time is minimum[17], and they can be kept for certain margin of time in one state before they are taken to the other one. In this case, there's the possibility of adding a certain minimum “respect time” that will be followed by the controller when choosing the next load to be connected or disconnected.

- **Reordering.** Strictly related to the previous criterion. If a load becomes available again, after finishing its respect time, it takes its original position in the priority list.

All these characteristics make of the Ekip controller an ideal component in a LV installation if one wants to achieve energy efficiency.

2.3 Management of multiple Ekip Controllers

As it was already mentioned, one of the characteristics of the Ekip controller is that it can be remotely configured using Ethernet connections, which means that we can create communication networks in which some Ekip controllers can be connected to a unique central device that could manage some or all of the functions of the controller. In the precise case of this project, this “unique central device” is a computer that will run the software tool that has been developed to control a bunch of functions.

16 i.e. motors, generators.

17 i.e. Temperature control devices (heating systems, refrigeration, water temperature control in swimming pools, etc.)

Given the current development of technology and since one of the main goals of energy efficiency is also the automation of the processes; in this case, this automation comes from the fact that we can use the information about energy prices that the market analyses can provide to have power limits that are defined in a day-ahead scheme. This power limits could be automatically loaded to the controllers, as done in this project and that will be shown in the next chapter.

Taking into account the autonomous process of loading information into the controllers, and that this will contain basically power limits information, there was the necessity of defining which parameters would be configured in the controllers[18] by the software tool; since there are some other parameters that would be specific for each case and need to be set by its own user, it means that the last couldn't (or it would be useless to) be automatically got from the general information available to the software tool; for example the load priorities table. All the parameters under configuration by the software tool here implemented will be indicated in the next chapter.

18 This means, the definition of which registers will be read or written (modified).

3 The Ekip Management Software

"A billion saved is a billion earned" -

< Norman Ralph Augustine >

Summary. <In this chapter, it is described the process for creating the Ekip Management Software, which is the final goal of this project; as well as the used tools, concepts involved, and the working process of the EMS. Moreover, it is explained the little 'trick' used in order to get a more accurate power limit profile by treating the settings of the Ekip controller in a slightly different way. After the concepts given in the first two chapters, here it is basically shown how the full project has been developed.>

3.1 General Description

Provided the need for an improvement in the energy efficiency schemes, and given the characteristics of the Ekip controller, which is used as the main control device in this work, it came up the opportunity of developing a software tool that could use the previous concepts in order to try to provide a more accurate and precise method of saving electric energy and of course money in one or more low voltage installations. The Ekip Management Software (EMS) is the final product of this project; and it's a tool developed in Python that makes it possible to configure multiple Ekip controllers at a time remotely by means of an IP network. The figure 3.1 provides a general understanding of the working principles used by the EMS tool to accomplish its main tasks.

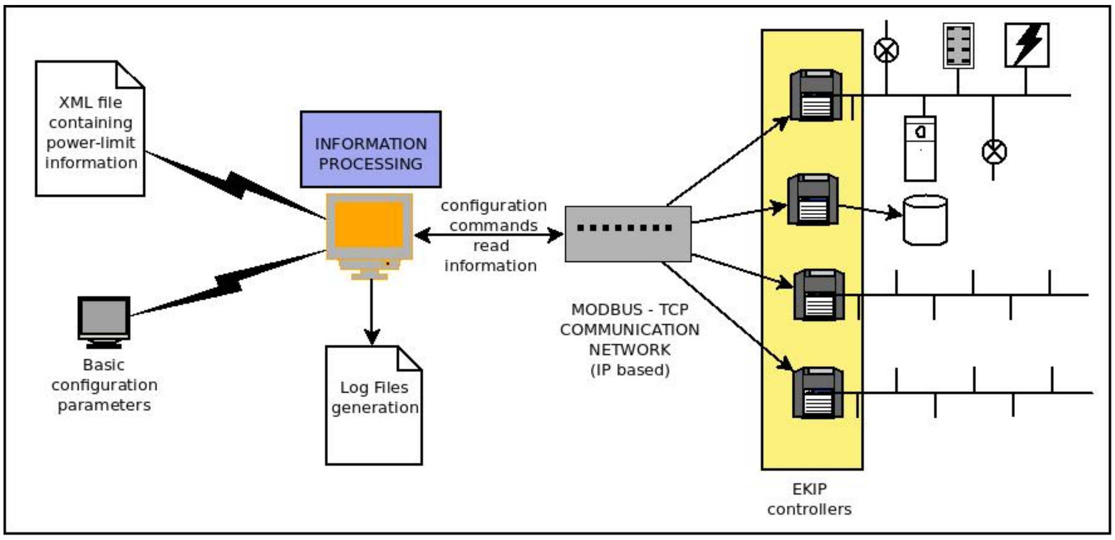

XML file containing power-limit information
INFORMATION PROCESSING
configuration commands read information
Basic configuration parameters
Log Files generation
MODBUS - TCP COMMUNICATION NETWORK (IP based)
EKIP controllers

Figure 3.1: General working process of the EMS tool

In the above figure, the first thing that becomes evident is that there are two kinds of inputs to the system; on one side there is the input information that the EMS will have from an **external system** (power limits, nodes identification, timestamps), and on the other side, the parameters that are **directly set** by the administrator of the EMS into the system's initial configuration settings. As for the first, it refers to the power limit information (along with other useful data that gives a meaning to these power limits) that is generated in another process, independent from this one, and it is contained in XML files.

It is also worth to note that there are some other parameters (unrelated to this specific work) that must be given directly by the user of each Ekip controller, since they depend on the internal electric arrangement of the LV installation. Here there is the information about the electric loads, priorities, protections and others. Anyhow, these parameters are the only ones that could be manually programmed into the Ekip controllers in any of the ways that were mentioned in the second chapter (serial, USB).

Finally, once all this information is available, the EMS tool can process it and generate the corresponding configuration commands that will remotely program the controllers through an IP network. For this process, it is important to have well defined the IP addresses for all the nodes (Ekip controllers); since these could be either static or dynamic (provided by a DHCP[19] server). These addresses are previously known by the system that generates the configuration information, and they will be included in the input XML file. Moreover, all the nodes have to be set in order to work connected to remote networks and with the appropriate addressing scheme.

Another important part of the EMS tool, besides writing the configuration information, is that it will constantly read some registers inside the controllers, in order to generate log files. This log file generation mechanism is periodic and continuous, and the output files can be created in a couple of different formats.

In general terms, this is how the EMS tool works and these are the basic tasks that it performs. In order to achieve these functionalities EMS uses some software tools (related to Python), and it's communication system is designed over the standard Modbus TCP. All of these components are going to be described in the following pages.

19 DHCP protocol allows client nodes to request an IP address form an IP pool that is defined in the DHCP server; taking out the need for a manual configuration of static IP addresses in each node. The addresses are constantly renewed, making an efficient use of this resource.

3.2 Software Tools

The EMS tool has been developed completely using the programming language Python 2.7[20], mainly because it runs on Windows, Linux/Unix, Mac OS X, and has been ported to the Java and .NET virtual machines; and it is free to use, even for commercial products, because of its OSI-approved open source license (PSF)[21], according to its official information. Besides that, Python has some external libraries that have been very helpful in order to implement the functions of the EMS tool. These libraries are:

- **Pymodbus** as described in the project webpage (http://code.google.com/p/pymodbus/) *"(it) is a full Modbus protocol[22] implementation using twisted[23] for its asynchronous communications core. It can also be used without any third party dependencies (aside from pyserial[24]) if a more lightweight project is needed. Furthermore, it should work fine under any python version > 2.3 with a python 3.0 branch currently being maintained as well"*. In the case of this project, the Pymodbus library provides the communication capability to the EMS software with the Ekip controllers, for both reading and writing the registers of interest. Here it is not used the asynchronous communication mode, since the EMS will only operate via IP networks.

20 www.python.org

21 Python Software Foundation license, which is compatible with GPL (GNU General Public License), which is a reference license for open software instances.

22 The Modbus protocol will be described in the next subsection.

23 Twisted is an event-driven networking engine written in Python and licensed under the open source. Please refer to: http://twistedmatrix.com/trac/

24 Pyserial provides access to the serial port. Please refer to: http://pyserial.sourceforge.net/

- **LXML**[25] is a Python library that provides an easy to use processing of XML and HTML files. In the case of this project, as depicted in the figure 3.1, the files containing the power limits information (which consist in the main input for the EMS tool) store their information in XML format. Therefore there is a need for a library to process the XML tags and obtain the stored information in order to be able to process the data that will be set into the Ekip controllers. For it's simplicity and abundance of examples and documentation, LXML has been chosen to perform this task. Also here it is interesting to mention that the parsing techniques for XML files can be done by two methods: line by line or the whole file at once. The second method is preferred for this project, because of the need for information about different nodes at the same time, as it will become evident later.

- **XLWT**[26] is a library used to create spreadsheet files complying with the formats of Microsoft Excel 97/2000/XP/2003 XLS files, on any platform, with Python 2.3 to 2.7. As for this project, it is useful in order to generate the log files (figure 3.1). These log files can be created in either XML or XLS formats, or both; as it will be shown later in this chapter. The XML files have a simple structure and can be easily generated from a Python script; however the spreadsheets have a more complicated structure that cannot be natively handled by the Python language, so there is the need for this additional library; and even with this, some considerations have to be taken into account when creating the XLS files.

25 http://lxml.de/index.html

26 https://pypi.python.org/pypi/xlwt

These libraries can be directly downloaded from the official websites of the respective projects, or they can be installed by using Python installation tools, such as **pip**[27] or **easy_install**[28].

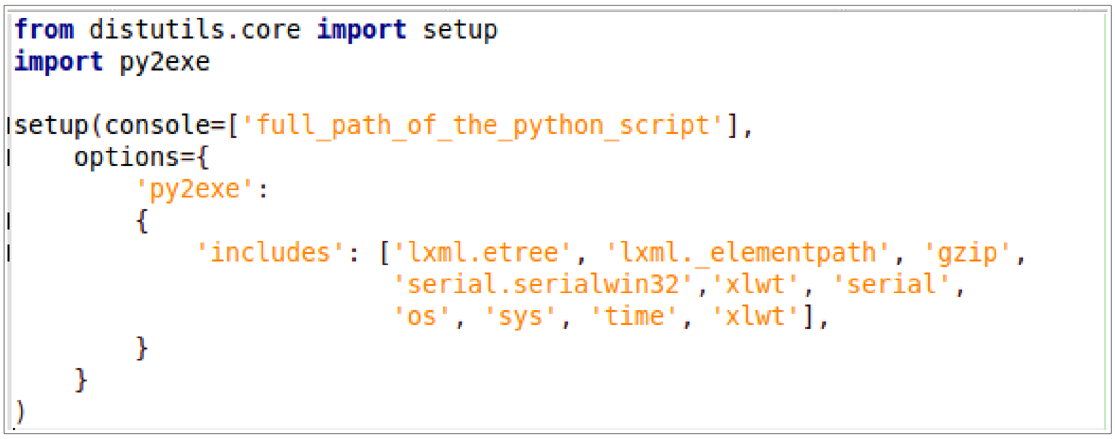
Here I have to mention that all the development of the software, including the installation of these libraries, the programing of the code for the EMS tool, and all the tests were done under GNU/Linux environment (Ubuntu 13.04[29]). However, the final tests in ABB laboratories were done in a Microsoft Windows environment, creating the need of porting the EMS tool to an executable file (EXE) that can be used under the mentioned environment. To accomplish this task, there was the need of installing the mentioned software (a full description of this process is given in the Appendix A) and generating the EXE file, by using the tool **py2exe**[30]. The generation of the executable file is quite simple when following the directions in the mentioned appendix, and the instructions in the web page of 'py2exe'. In this case, it is useful to read the tutorials about how to generate the EXE files, and to take care of the dependencies (external libraries used by the EMS software). Pyserial and XLWT were the ones causing some trouble, but in the tutorials it is mentioned how to include those libraries in the generation of the executable file; this is also shown in figure 3.2.

27 https://pypi.python.org/pypi/pip

28 http://pythonhosted.org/distribute/easy_install.html

29 http://www.ubuntu.com/

30 http://www.py2exe.org/



```
from distutils.core import setup
import py2exe

setup(console=['full_path_of_the_python_script'],
    options={
        'py2exe':
        {
            'includes': ['lxml.etree', 'lxml._elementpath', 'gzip',
                         'serial.serialwin32','xlwt', 'serial',
                         'os', 'sys', 'time', 'xlwt'],
        }
    }
)
```

Figure 3.2: Setup file used to generate an executable file from the Python script containing the EMS code

As it can be seen, the inclusion of these libraries in the final code is just a matter of declaration in the setup file. There are some tools that can be used to perform this task, but py2exe was chosen mainly due to its degree of maturity, for it's simplicity and abundance of examples and documentation.

3.3 The Modbus Protocol

The Modbus[31] protocol is the core of the **communication** process of the EMS tool, and here it has been implemented using the Pymodbus library already mentioned in the previous subsection.

As officially defined, *"MODBUS is an application layer messaging protocol, positioned at level 7 of the OSI model[32], which provides client/server communication between devices connected on different types of buses or networks"* **[Modbus, p.2][xv].**

31 http://www.modbus.org/

32 The Open Systems Interconnection (OSI) model is a conceptual, 7-layer model developed by a project at the International Organization for Standardization (ISO), that characterizes and standardizes the internal functions of a communication system.

Modbus is a protocol that has been used for serial communications in automation devices since 1979, and that today it still provides with communication to new devices, since it is continuously updating its functionalities according to the new available technologies, such is the case of the Modbus TCP implementation. The Modbus standard is based in a series of function codes that allow the devices to write and/or read registers in the connected devices, for both serial and TCP implementations. Also, by means of a connector device, called gateway, needed for compatibility, these two implementations can become inter operable. This gateway also allows to have different bus types in one unique Modbus network.

The general Modbus protocol defines a simple protocol data unit (PDU) independent of the layers below; and depending of the kind of implementation of the protocol it can include some additional fields in the application data unit (ADU) that would be needed for addressing and error checking. The PDU and ADU formats can be seen in the following graphic:

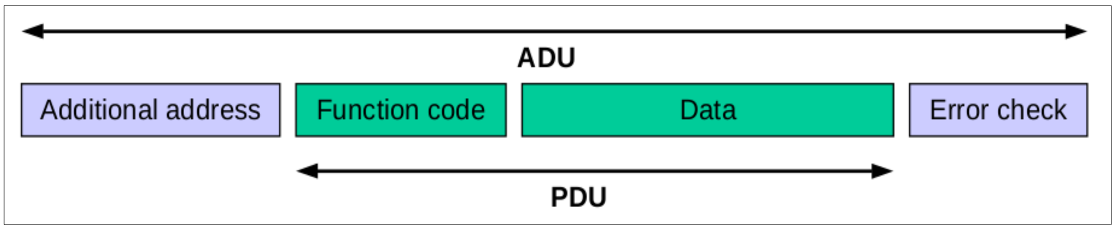


Figure 3.3: General MODBUS frame **[Modbus, p.3]The Modbus Organization,** “Modbus Application Protocol.”

As for the 'Additional address' and 'Error check' fields, they are used for serial implementations, which are outside the scope of this work. These functionalities are already implemented by the underlying protocols in the case of the TCP implementation.

The protocol data unit, as defined by the standard, consists of two fields, data and function code, both of interest for this project.

- **Data:** Is the payload itself, containing the information that will be read or written into the internal registers of the device in a big-endian[33] representation. The Modbus data model consists of four data types for which the protocol allows up to 65536 data items for an individual selection as described in the table below:

DATA TYPE	WORD LENGHT	READ - WRITE
Discretes Input	*Single bit*	*Read only*
Coils	*Single bit*	*Read & Write*
Input Registers	*16 bits*	*Read only*
Holding Registers	*16 bits*	*Read & Write*

Table 3.1: Modbus data types

These four groups of data types mean that depending on the needs of the manufacturer and the user, there are situations in which there is the need to access these registers at a bit level or byte level, where the information can only be read or also modified. These different types of data will cause that some **functions have to be defined** in order to manage them accordingly; and these functions will be specified by means of the so called function codes.

- **Function Codes (FC):** The second field of the PDU corresponds to the function codes which characterize the type of data that will be handled and the kind of operation that will

33 It means that the most significant byte is transmitted first.

take place upon them. Function Codes are characterized by a number called descriptor that will go from 1 to 127. The Modbus protocol defines three kinds of function codes, which are: public, user-defined and reserved. The **public Modbus function codes** are well defined, validated by the Modbus Organization and publicly documented. In the case of the **user-defined function codes**, the user can assign and implement them according to his/her own needs, but there's no warranty that the FC will be unique. At last, the **reserved function codes** are not in use anymore and are there just to support legacy products by some vendors. In the case of the current work, **only public FC's will be used**, and among them only the following:

- **04 (0x04) Read Input Registers** (implemented in Pymodbus as **`read_input_registers`**(*address*, *count*)).

Request			**Response**	
FC	*1 byte*	*04*	*1 byte*	*04*
Starting Address / Byte count	*2 bytes*	*0 to 65535*	*1 byte*	*2 x **N***
Registers	*2 bytes*	*1 to 125*	***N** x 2 bytes*	

Table 3.2: Request and Response details (FC 4)

Where **N** = Quantity of registers.

- **16 (0x10) Write Multiple Registers** (implemented in Pymodbus as **`write_registers`**(*address*, *values*))

Request			**Response**	
FC	*1 byte*	*16*	*1 byte*	*16*
Starting Address	*2 bytes*	*0 to 65535*	*2 bytes*	*0 to 65535*
Registers	*2 bytes*	*1 to 123*	*2 bytes*	*1 to 123*
Byte count	*1 byte*	*2 x **N***		
Registers value	***N** x 2 bytes*	*value*		

Table 3.3: Request and Response details (FC 16)

Where **N** = Quantity of registers.

The values in the tables 3.2 and 3.3 are useful as a reference[34], to take into account when using the Pymodbus commands that are the ones that will actually implement the reading and writing functions in the EMS tool. Also, it is useful to know that in case of an **exception response**, it will have the same PDU, but with its most significant bit set to 1. These exception responses can be of three classes:

- **Communication error**; in this case the response will be a “timeout”.

- **Error in the frame**, including parity, CRC, etc. Also in this case the response will be a “timeout”.

- **The request cannot be processed**, in which case the response will contain an exception code defined in the protocol.

34 This because in the case of this project, raw PDU's will never be used in the final implementation of the software. These were mainly useful for testing during the different stages of development of the project.

Modbus TCP:

In the beginning, the Modbus protocol was designed to operate over serial interfaces, but given the technological development of communication networks and the advance of TCP/IP networks in all environments (home, industry, etc.) it had to adapt to the new situation in order to survive in those conditions and that's how the TCP implementation of the Modbus protocol became a reality. Called Modbus TCP[35], it operates in a similar way as its predecessor for serial interfaces, only considering that the addressing and error check schemes are already implemented in lower layers of the OSI stack (Modbus operates at application level), making them no more necessary in this version; and that's why in this case the complete ADU shown in figure 3.3 is not needed, but **only the PDU** shown in the same figure, whose two components (function code and data) are the same that were already described in the previous subsection. Therefore, as it was already mentioned, there is the possibility of integration for this Modbus TCP networks with the serial version of Modbus, by using gateways.

Implementation in the EMS tool:

All the functions of the Modbus TCP protocol that were used in this work were implemented through the **Pymodbus library** in Python; which can be installed downloading it directly from the project website[36] (recommended option), or it can be installed by using the Python tools pip or easy_install. The library works well under Python 2.7 when reading and writing registers from or to generic Modbus devices; however when trying to make this library

35 For documentation and tools: http://www.modbus.org/toolkit.php

36 http://code.google.com/p/pymodbus/downloads/list

work with the Ekip controllers[37] there was a special situation. In order to be able to read registers, the default **unit identifier**[38] value implemented in the Pymodbus library was set to '0' (0x00), which usually means that all connected devices should respond to that request; but due to the implementation of the standard that was done by ABB in the Ekip controllers, it had to be changed to '1' (0x01). In order to do this, the 'constants.py' file included within the library had to be modified as stated in the Pymodbus tutorial **[Collins, p.126]**[xvi]. This had to be done **before** installation.

The Pymodbus commands that were used for the communication processes of the EMS tool were only two:

- **`read_input_registers`**(*address*, *count*); where**:**
  address is the starting address to read from,
  count is the number of registers to read.

- **`write_registers`**(*address*, *values*); where**:**
  address is the starting address to write to,
  values are the values to write to the specified address.

Both instructions are part of the class **client.sync [Collins, p.98]**[xvii] and they handle numbers only in decimal system; also they should give responses according to the mentioned request/response scheme (tables 3.2 and 3.3).

37 Previous tests with Modbus TCP simulators didn't cause any trouble. The used simulator was 'Modbus slave' (http://www.modbustools.com/modbus_slave.asp)

38 UnitId: Modbus slave address, used mainly for serial communications.

3.4 Main Workflow

The Ekip Management Software has been designed in order to accomplish two main tasks: an automatic configuration process and the generation of log files; tasks that have produced two independent modules that work together inside the primary program. The main workflow of the EMS tool can be seen in the figure 3.4, where it is shown how these two modules, the Ekip Controller Automatic Configuration Tool (ECACT) and the Ekip Controller Logs Generation Tool (ECLGT) interact; so far the mentioned modules are estated only as predefined processes, and they will be clearly described in the following sections of this chapter.

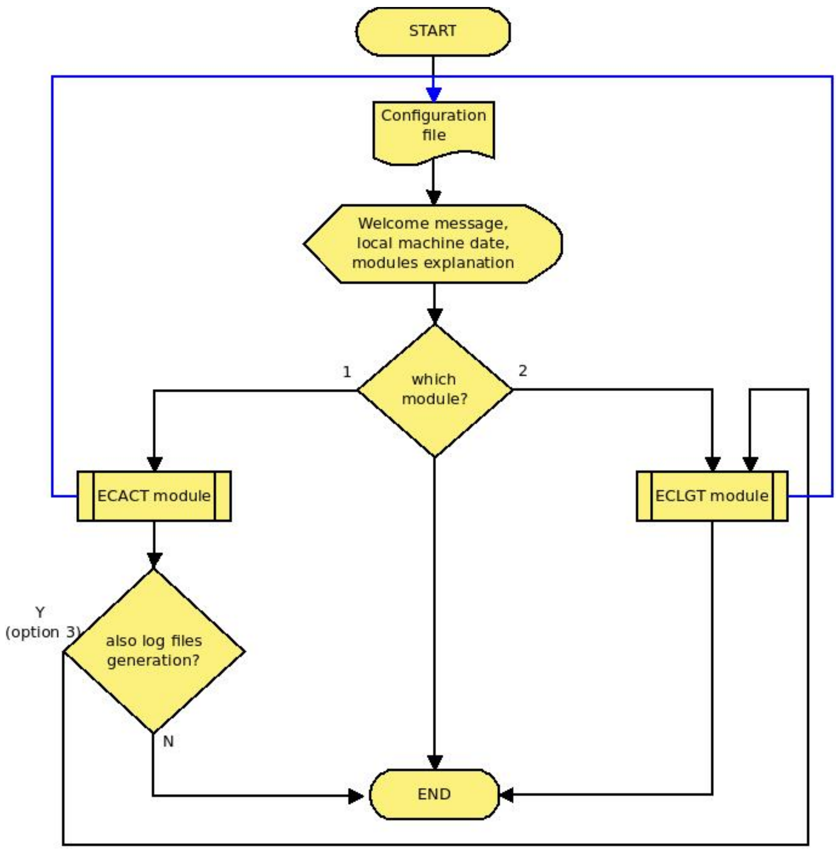
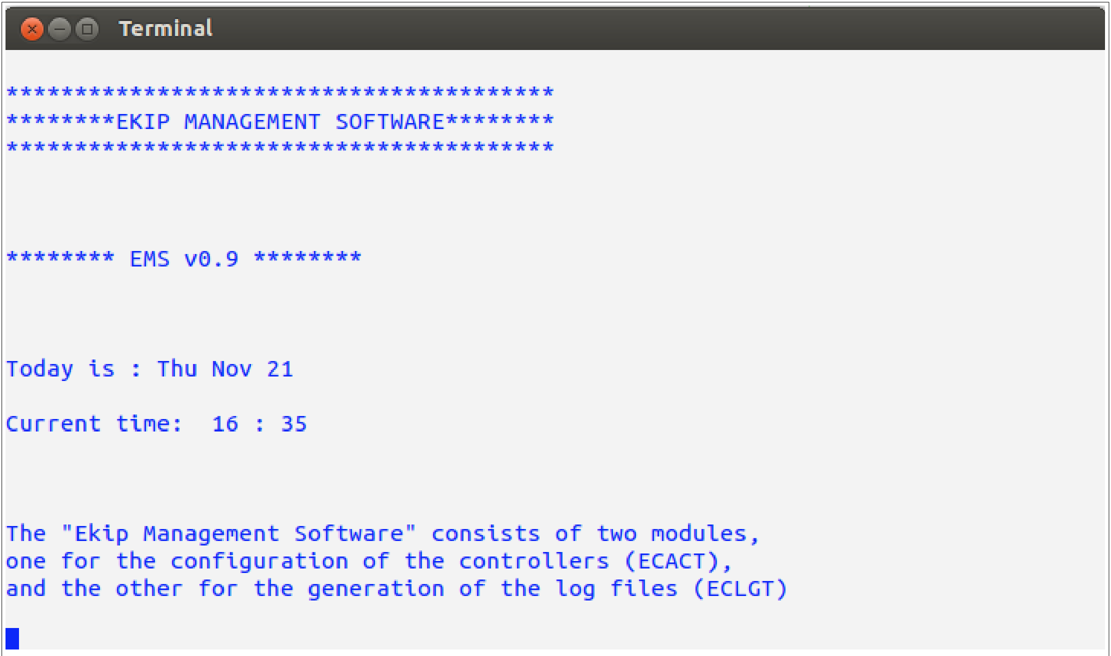

START
Configuration file
Welcome message, local machine date, modules explanation
which module?
1
2
ECACT module
ECLGT module
Y (option 3)
also log files generation?
N
END

Figure 3.4: General EMS workflow diagram

In the figures 3.4 and 3.5 it is possible to observe that when using EMS there's the chance to select which one of the modules is going to be used, being also possible to use both of them, if the user chooses to generate the log files of the already configured[39] nodes. This selection of the modules is not performed in the program itself, but they are already programmed in the configuration file.



```
****************************************
********EKIP MANAGEMENT SOFTWARE********
****************************************



******** EMS v0.9 ********



Today is : Thu Nov 21

Current time:  16 : 35



The "Ekip Management Software" consists of two modules,
one for the configuration of the controllers (ECACT),
and the other for the generation of the log files (ECLGT)
```

Figure 3.5: Initial window of the EMS tool, where it is possible to choose which of the two modules is going to be used

In the figure 3.4 there are also two blue lines that indicate that the program can go back to the beginning from any of the two modules. This situation happens when a new XML input file is found. EMS is constantly checking for new information; and when it arrives, it starts over the configuration/log generation process, according to the settings in the configuration file.

Once the EMS software has been started, it can keep running

39 By "configured" I mean the nodes that have been pointed as recipients of the configuration information that would be loaded by the ECACT module.

without the need for any kind of intervention, as long as it has new input files, at least once a day.

Configuration file:

There is a text file (config.txt) that contains the basic initial information that EMS will use in order to operate. This file has been created in order to make the software tool as independent as possible. There are five parameters specified in this file, which are:

- The directory where to look for new input XML files.
- The directory where to save the old input XML files.
- Which modules (ECACT, ECLGT or both) will be used.
- The directory where the log files are going to be stored.
- The format of those log files.

These parameters can be modified only before the program starts or before a new XML input file arrives. Otherwise, if there is a need for changing the parameters, the EMS tool has to be restarted. An example of this configuration file is shown in Appendix B.

3.5 The ECACT Module

The **Ekip Controller Automatic Configuration Tool**, here called ECACT module, has the main goal of retrieving data from XML files that contain information about the maximum powers for the different nodes (Ekip controllers) in a time based scheme; in order to automatically configure those nodes in an iterative process, assigning the maximum powers to the nodes, and updating the information in the controllers for every time slot (which has been defined in **fifteen minutes**, according to the information available in the XML files).

The first task that this module has to realize is the retrieval of the information contained in those XML files (it is possible to find a complete XML file as an example, in Appendix C) that are the input of the system. This information is organized according to some 'tags', being the most important, the following:

- **<sg:externalSystemIdent>:** It corresponds to the ID of system which has to receive the message section.

- **<sg:programIdent>:** It's the name of the virtual power plant.

- **<sg:participantCount>:** It says how many nodes will have to be considered in the message section. By default this value is set to '1', which means that all the nodes will be configured.

- **<sg:schedule>:** Shows that the time based power limits section for a specific node starts.

- **<sg:scheduleItem>:** Relates the two following tags (amount and date-time).

- **<sg:amount>:** It is the setpoint for the single node, which means the value of the power limit for the corresponding time slot. The value is expressed in MW.

- **<sg:dateTime>:** It corresponds to the timestamp for the above setpoint.

- **<sg:defaultValues>:** Marks the section where there are the default power limits for the four time periods in a day.

- **<sg:value_X>:** Where X can be 1 to 4; is the default power limit for a time period. It is also expressed in MW and it is used when there is no specific power information for a fifteen minute time slot.

- **<sg:periodStart_X>:** Where X can be 1 to 4; it indicates where a time period starts.

- **<sg:periodStop_X>:** Where X can be 1 to 4; it indicates where a time period finishes.

- **<sg:sectionIdent>:** It is the serial number (ID) of the site.

- **<sg:ipAddress>:** Corresponds to the IP address of a given node.

Once this information has been extracted from the input XML file and somehow processed (stored, standardized, organized and matched), the ECACT module will generate configuration commands that will be sent periodically to all the involved nodes, in order to keep their power limits set to the values stated in the input XML file. Although the management of the information will be better explained later, the general process can be understood in a better way by looking at the figure 3.6, which corresponds to the workflow diagram of the ECACT tool.

In the mentioned figure it is easy to see how the ECACT and ECLGT[40] modules are related and can coexist, but it also shows the parsing method of the XML input file, which consists in extracting the useful information into lists, storing the information that will be then used by the ECACT function in order to perform the required

40 Please refer to the next section (3.6).

tasks. This ECACT function has been created given the iterative nature of the process involving the configuration of the Ekip controllers. It is also clear how the program waits for a new XML file and in case of finding it, it will start over; otherwise it just waits for fifteen minutes.

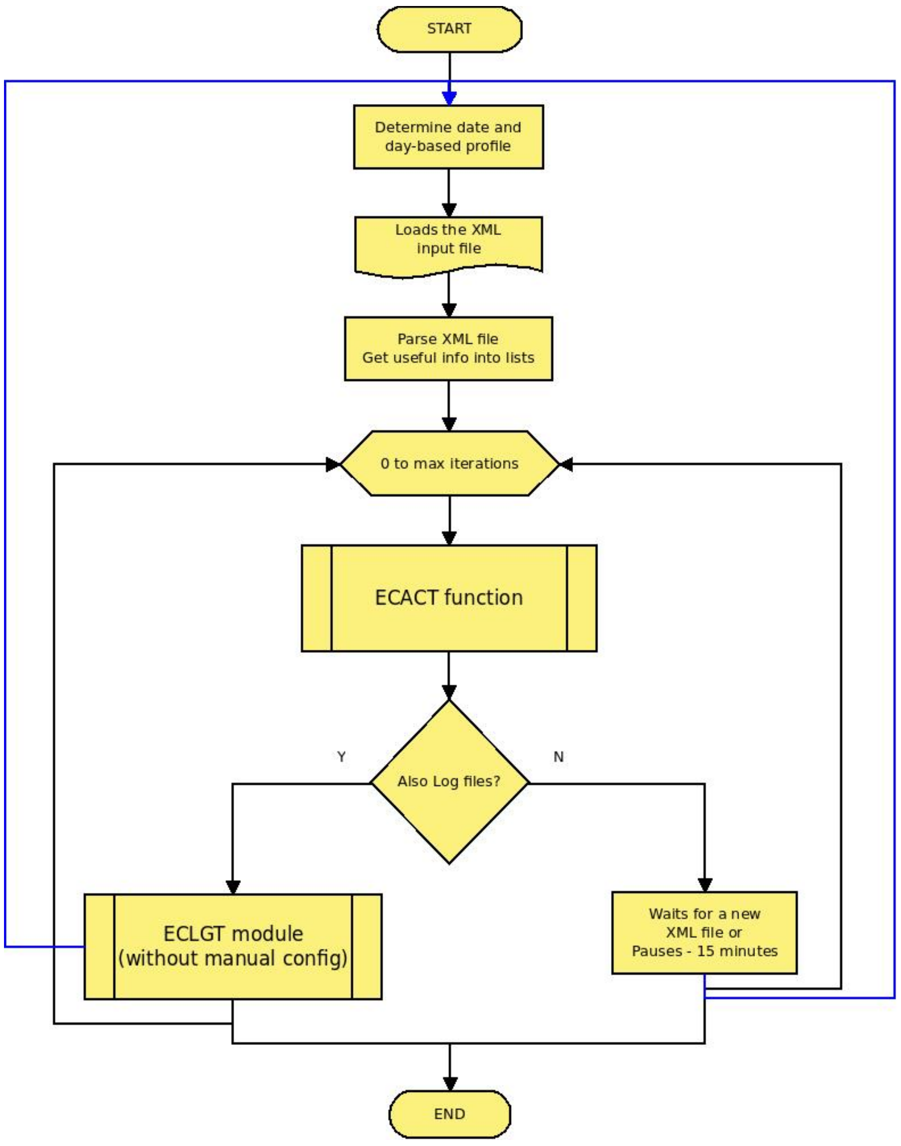
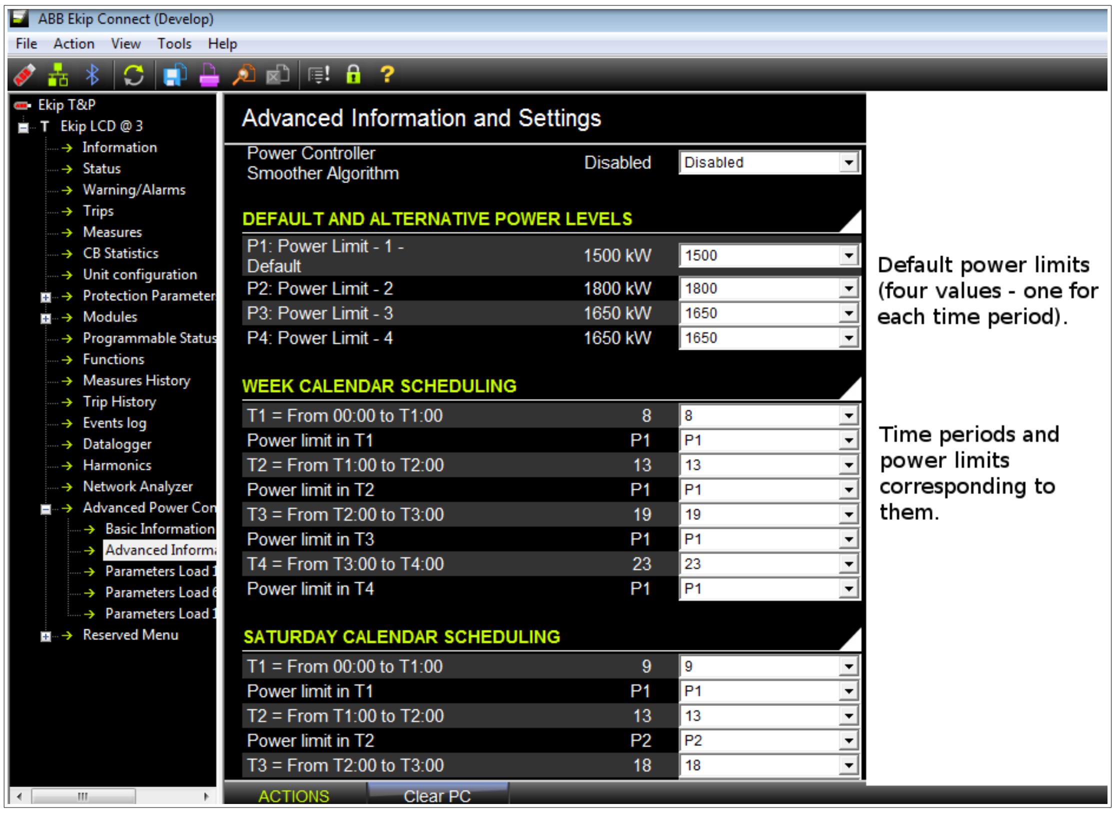

START
Determine date and day-based profile
Loads the XML input file
Parse XML file Get useful info into lists
0 to max iterations
ECACT function
Y
N
Also Log files?
ECLGT module (without manual config)
Waits for a new XML file or Pauses - 15 minutes
END

Figure 3.6: Workflow diagram of the ECACT module

What happens here is that for each node, the values for the time periods are first defined and loaded into the controllers. This is a simple process since the same information that is available in the XML files can be written in the corresponding Ekip controller. After that, the values for the power limits have to be programmed and constantly updated in the controllers. For this process, there is the need of **'tricking' the functions of the Ekip**. As it can be seen in figure 3.7, the Ekip controller has four power limits defined for the corresponding four time periods; and from the XML file there are four power limits available per hour. So the trick consists in updating these power limit values every hour, and choosing as default power limit value the one that corresponds to the current quarter hour (i.e. P1 for HH:00 to HH:15, P2 for HH:15 to HH:30, etc.). In this way, the power limits are not related to the time periods (as it is in the original operation of an Ekip controller), but to each quarter hour. This procedure allows the controller to have a more accurate control over the electric energy usage in the LV installation, with a higher level of granularity and fitting in a closer way the power limits profile that has been defined by the company, with more efficiency in order to reduce the expenses and make a better use of the generation processes and sources. However, there is no information available for each fifteen minute time slot along the whole day; and in this case what happens is that there are default power limits (included in the XML input file) that would be used to fill those spaces.



Advanced Information and Settings

Power Controller Smoother Algorithm	Disabled	Disabled
DEFAULT AND ALTERNATIVE POWER LEVELS		
P1: Power Limit - 1 - Default	1500 kW	1500
P2: Power Limit - 2	1800 kW	1800
P3: Power Limit - 3	1650 kW	1650
P4: Power Limit - 4	1650 kW	1650
WEEK CALENDAR SCHEDULING		
T1 = From 00:00 to T1:00	8	8
Power limit in T1	P1	P1
T2 = From T1:00 to T2:00	13	13
Power limit in T2	P1	P1
T3 = From T2:00 to T3:00	19	19
Power limit in T3	P1	P1
T4 = From T3:00 to T4:00	23	23
Power limit in T4	P1	P1
SATURDAY CALENDAR SCHEDULING		
T1 = From 00:00 to T1:00	9	9
Power limit in T1	P1	P1
T2 = From T1:00 to T2:00	13	13
Power limit in T2	P2	P2
T3 = From T2:00 to T3:00	18	18

Default power limits (four values - one for each time period).

Time periods and power limits corresponding to them.

Figure 3.7: Time periods, Power limits and their correspondence in the Ekip controller.

The function: (lists, defaults, hours, nodes, j):

Where **'lists'** is the main list of lists, containing all the useful information (power limits for the fifteen minute time slots) extracted from the XML input file, **'defaults'** corresponds to the default power limits for the four time periods defined for a day, **'hours'** is the list with the values of those time periods, **'nodes'** corresponds to the number of nodes from which there is available configuration information, and **'j'** is the counter of the number of iterations of the process. Returns **'ip_list'**, which is a list containing the IP addresses of the configured nodes.

This function performs the main task in the ECACT module, and makes use of the other functions that will be described in the section 3.7 of this chapter. It takes all the data from the input lists and after processing them, it will load the configuration values into the corresponding controllers. Here are included all the functions for writing the time periods, power limits and the relationship among them; but also some other auxiliary tasks, such as calculating date and time according to the information in the controllers, so the tool would know when to write the values into them. There are also functions for standardizing the information in order to fit the requirements of the controllers, because there are some defined values that the registers can take (e.g. the power limits have to be specified in steps of tens); and functions for "filling the blanks" with default power limits when there's no information available for a certain time period. The full code of the function can be found in Appendix D.

3.6 The ECLGT Module

The second module, called ECLGT, which stands for **Ekip Controller Logs Generation Tool**; as its name indicates is the one in charge of the generation of log files for the nodes (Ekip controllers) of interest. The nodes for which the log files will be generated are defined by their IP addresses, being these either directly gotten from the input XML file (in case that only this module is being used), or obtained from the ECACT module in case that it has been used first. The log files are updated with new information about the status of all the involved nodes every minute, and stored every 15 and/or 60 minutes, depending on the kind of log files that are being generated (these could be in either XML format or as a spreadsheet, or both). In the figure 3.8, there is a general scheme of the main procedure for this module.

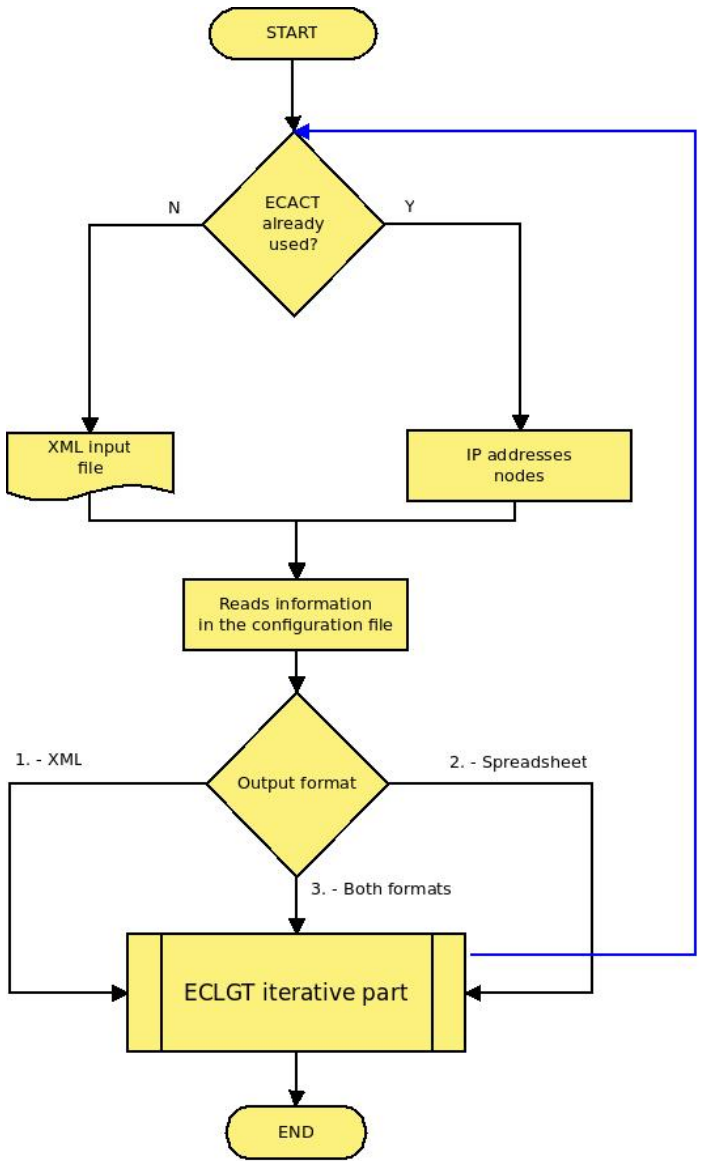


Figure 3.8: Workflow diagram of the ECLGT module - configuration process

In this diagram it is possible to see that there is a short configuration process that changes depending on whether the ECACT module has been executed before or not, since in the last case the relevant information for the ECLGT module wouldn't be read from the XML file but from the lists of information created by that module. This process, along with the information in the configuration file, stipulate the general rules that the ECLGT module will follow in the generation of the log files, being these the number of nodes involved in the process, the IP addresses of these nodes, and the directory and format in which the user wants the information to be saved. It is visible again the 'blue loop' indicating that the program could restart at any time, if a new input file is found.

Once the format of the log files is chosen, and the module enters the so called “iterative part”, there are three possible paths that the software could follow, being all of them different in their working procedure as well as in their results. However there are points that the three options have in common, such as the stored information and the time intervals in which it is updated. As for the collected data, these are:

- **Measurement time,** which is the timestamp taken from the controller itself, in order to have a time register accurate to the measurements of the other fields.

- **Active Power Total,** measured in watts [W]. Is one of the power measurements in the Ekip controller, and it indicates the total active power.

- **Active Energy Total,** measured in kilowatts-hour [kWh]. Corresponds to another measurement, the total active energy.

- **Default Power Limit,** measured in kilowatts [kW]. Corresponds to the actual value of the power limit that the controller has to keep.

- **Evaluation Window,** measured in minutes [min]. It corresponds to the “measurement time” defined for the Ekip controller, and it's the regulation window that was explained in the working principles of the controller (section 2.2). By default it is used the value of fifteen minutes.

- **Elapsed Time,** measured in minutes [min]. It refers to the amount of time that has passed since the evaluation window started.

- **Mean Power,** measured in kilowatts [kW]. It is the actual power consumption measured by the power controller.

- **Energy Log Index,** that indicates which one of the following indexes is being used at a certain time.

- **Energy logs (0 to 15),** measured in kilowatts [kW]. They correspond to the mean power that has been measured during a time window; and they takes place sequentially, according to the position indicated by the “energy log index” value.

All these data are updated in the log files every minute.

Now, the differences are both in procedure and results, depending on which type of output file the user chooses to generate, XML and/or spreadsheet.

XML file:

This is the most simple of the two generation processes, because the Python language makes possible the generation and edition of simple text files; characteristic that is used by the ECLGT module in order to create the XML log file by simply building text files to which it can add the appropriate tags, header and the .xml extension. The workflow for this procedure is shown in the figure 3.9.

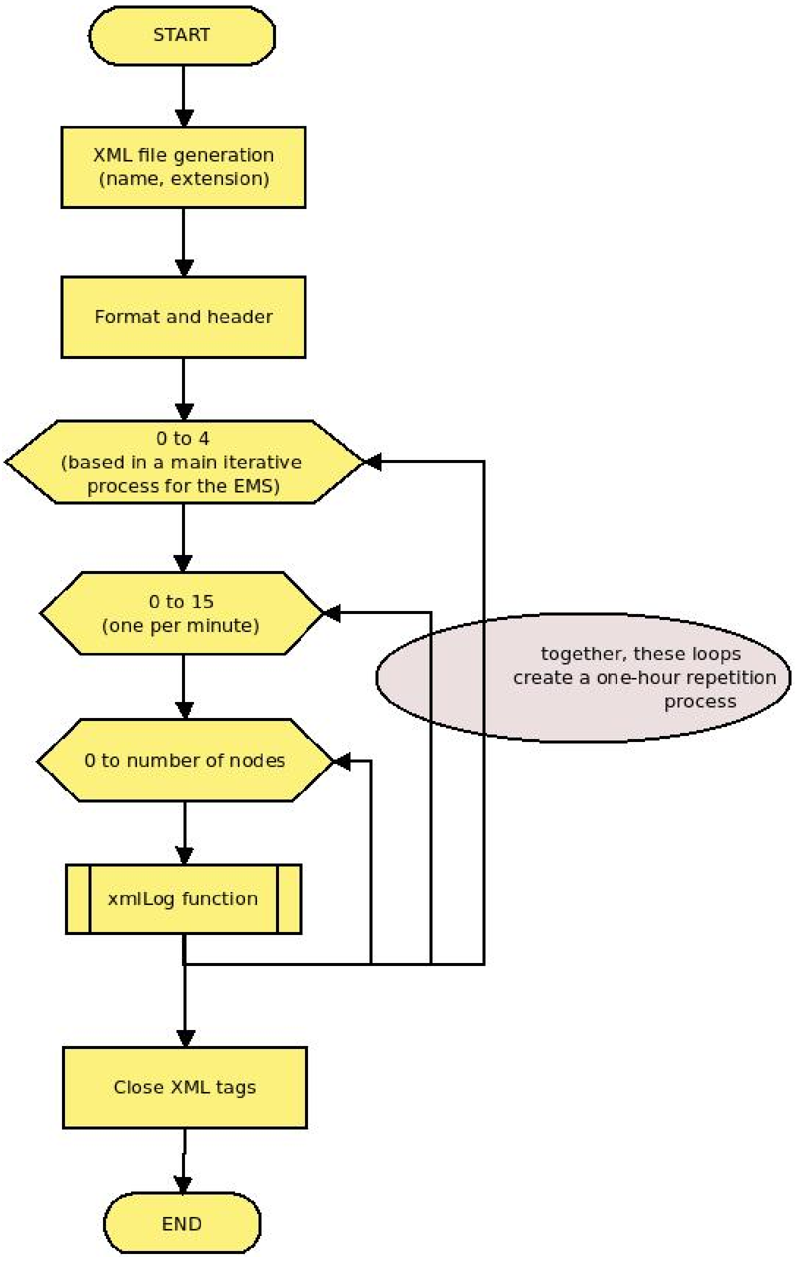
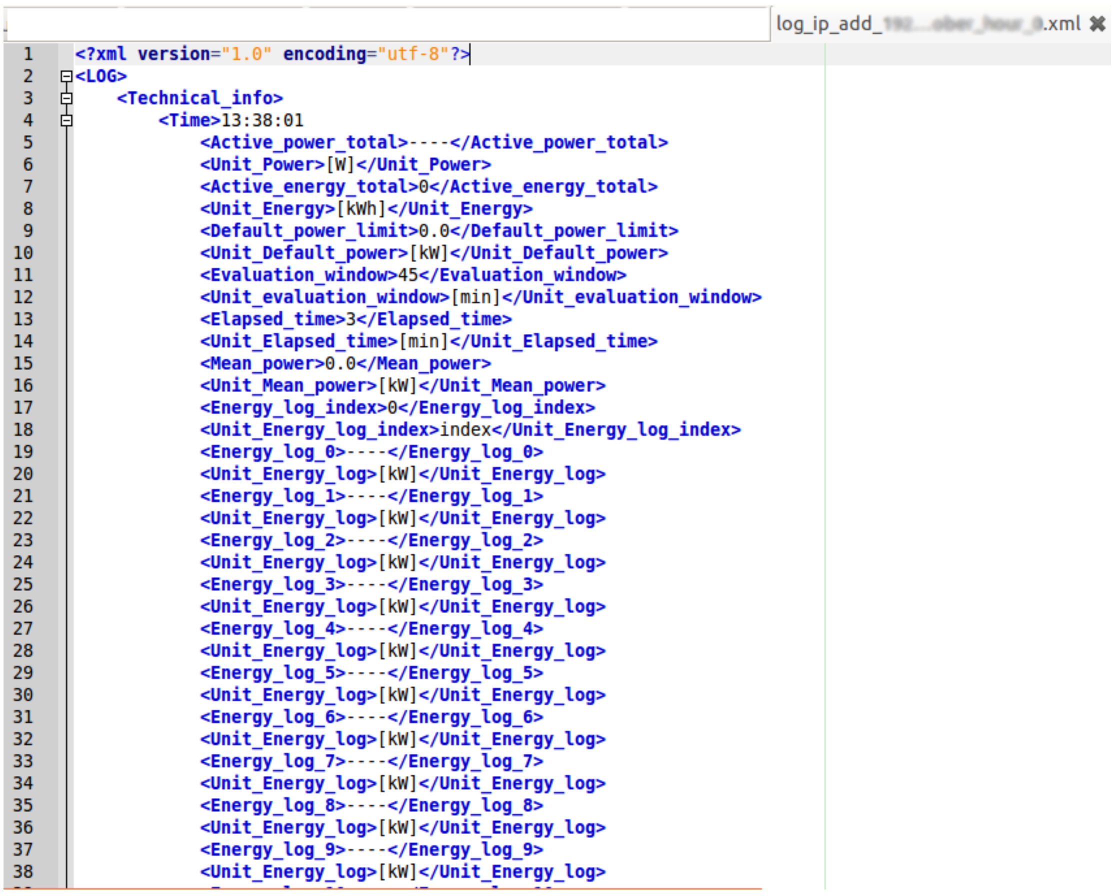

START
XML file generation (name, extension)
Format and header
0 to 4 (based in a main iterative process for the EMS)
0 to 15 (one per minute)
together, these loops create a one-hour repetition process
0 to number of nodes
xmlLog function
Close XML tags
END

Figure 3.9: XML files generation procedure

As it is visible in the above diagram, the generation of this kind of files is an iterative process that updates the information in the generated log files every minute for each one of the involved nodes; process that happens sixty times, which means that there will be **one log file per node per hour**. Due to the reason that at some point the ECLGT and ECACT modules might have to operate both at the same time, the loops have been divided into smaller loops of quarter hours. It was clear in the previous section that the ECACT module repeats the configuration process every fifteen minutes.

An important component of this module is the function **xmlLog(ip, i)**, that will be also mentioned in the next section. This function is the one in charge of the Modbus TCP communication procedure through which the workstation communicates with all the nodes in order to read their registers which contain the useful information, process these data and write them into the XML log files. As the final result the user will have a file with the structure shown in figure 3.10 (a full example can be found in the Appendix E), where it is visible: a small header, the read values (information in the controller) and their corresponding tags as it was previously indicated. It is also clear that the information is presented in a tree structure typical in XML formats, where the main tag corresponds to the 'time', and for each one of these timestamps there will be included all the information about that specific node. This kind of file will be generated for each one of the nodes of interest. As it was already said, one log file per node per hour.



```xml
<?xml version="1.0" encoding="utf-8"?>
<LOG>
    <Technical_info>
        <Time>13:38:01
            <Active_power_total>----</Active_power_total>
            <Unit_Power>[W]</Unit_Power>
            <Active_energy_total>0</Active_energy_total>
            <Unit_Energy>[kWh]</Unit_Energy>
            <Default_power_limit>0.0</Default_power_limit>
            <Unit_Default_power>[kW]</Unit_Default_power>
            <Evaluation_window>45</Evaluation_window>
            <Unit_evaluation_window>[min]</Unit_evaluation_window>
            <Elapsed_time>3</Elapsed_time>
            <Unit_Elapsed_time>[min]</Unit_Elapsed_time>
            <Mean_power>0.0</Mean_power>
            <Unit_Mean_power>[kW]</Unit_Mean_power>
            <Energy_log_index>0</Energy_log_index>
            <Unit_Energy_log_index>index</Unit_Energy_log_index>
            <Energy_log_0>----</Energy_log_0>
            <Unit_Energy_log>[kW]</Unit_Energy_log>
            <Energy_log_1>----</Energy_log_1>
            <Unit_Energy_log>[kW]</Unit_Energy_log>
            <Energy_log_2>----</Energy_log_2>
            <Unit_Energy_log>[kW]</Unit_Energy_log>
            <Energy_log_3>----</Energy_log_3>
            <Unit_Energy_log>[kW]</Unit_Energy_log>
            <Energy_log_4>----</Energy_log_4>
            <Unit_Energy_log>[kW]</Unit_Energy_log>
            <Energy_log_5>----</Energy_log_5>
            <Unit_Energy_log>[kW]</Unit_Energy_log>
            <Energy_log_6>----</Energy_log_6>
            <Unit_Energy_log>[kW]</Unit_Energy_log>
            <Energy_log_7>----</Energy_log_7>
            <Unit_Energy_log>[kW]</Unit_Energy_log>
            <Energy_log_8>----</Energy_log_8>
            <Unit_Energy_log>[kW]</Unit_Energy_log>
            <Energy_log_9>----</Energy_log_9>
            <Unit_Energy_log>[kW]</Unit_Energy_log>
```

Figure 3.10: Resulting XML log file example.

Spreadsheet (.xls) file:

The case of the spreadsheet file generation is not as straightforward as the previous situation since Python doesn't have an integrated function that could accomplish this task, and also the structure of this kind of files is much more complex than the one of a plain text file. That is why there is the need for an additional module and some considerations that had to be taken into account in order to be able to create and store the data in this format.

The additional module used was the “XLWT”, that was already described in the section 3.2 of this chapter; and one of the main issues is that once a spreadsheet file is created, there can only be added new sheets and data into the empty cells, but it is impossible to edit them. Once the information is added, the file has to be closed and saved. These things had a great impact in the procedure followed in the generation of this kind of files. This procedure can be seen in figure 3.11.

From that workflow diagram, it is possible to see that this is also an iterative process that depends on the number of nodes from which the information is being obtained, as well as the time variable. As it was previously mentioned, in this case the log files cannot be edited, making it mandatory to generate and save a new log file every fifteen minutes, but this is also convenient in the sense that the user will have a complete register of the system every quarter hour; and in this case these files will contain information about all the nodes connected to the system. The main component of this procedure is also its corresponding function, **xlsLog(sheet, i, ip)**, that will be described in the next section.

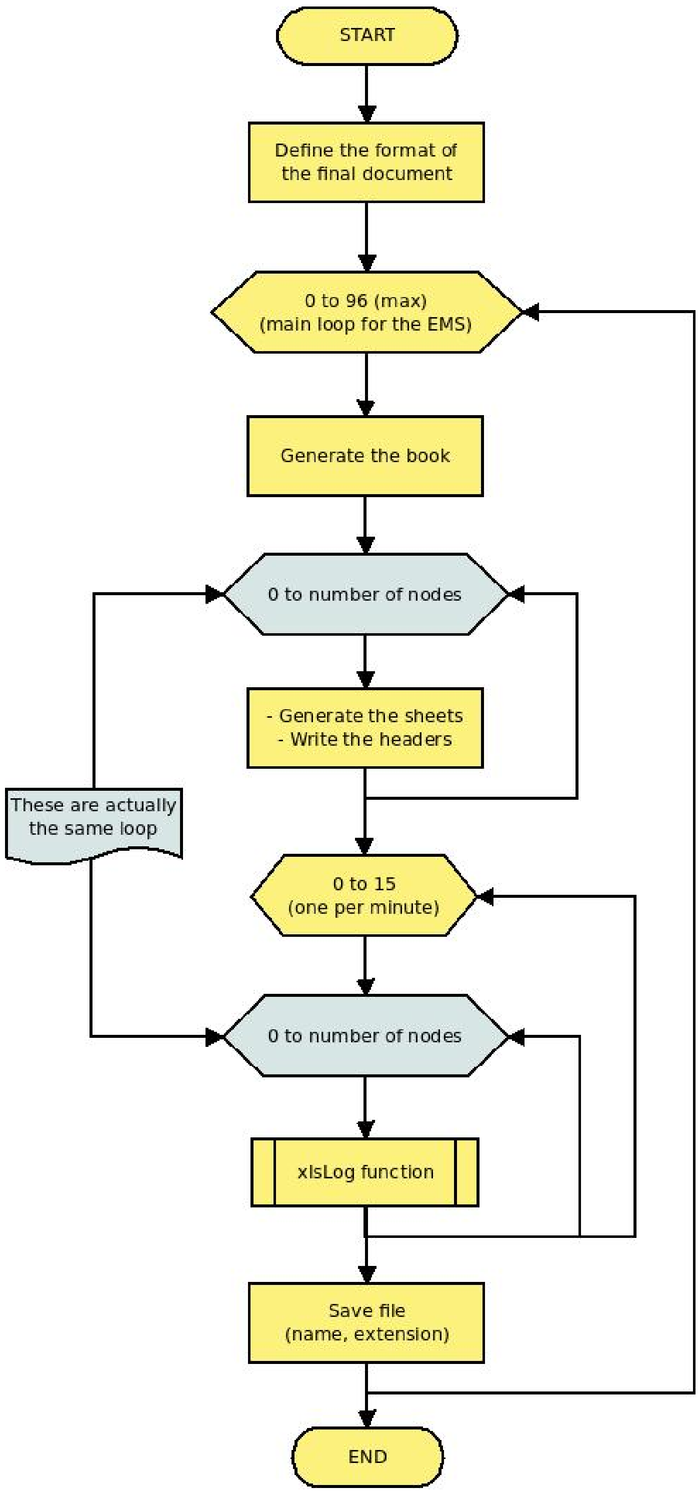
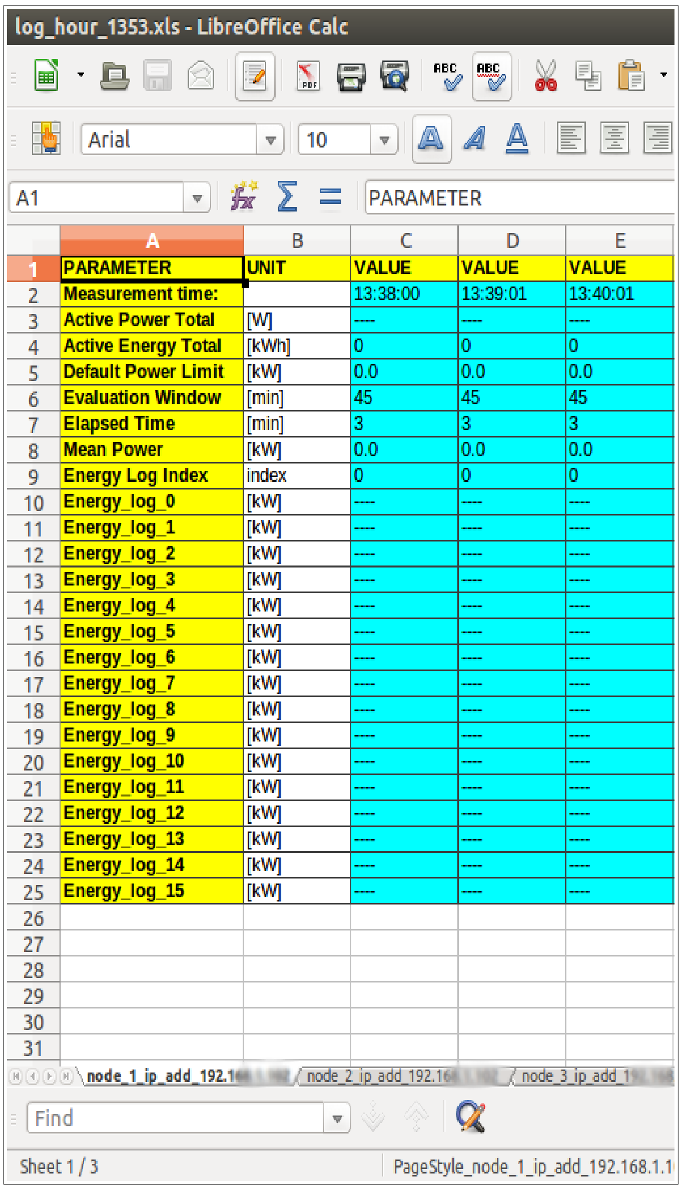


Figure 3.11: Spreadsheet file generation procedure

In the end, the user of the EMS tool would have the same information as in the previous case of XML log files, but organized in a different way and in a different format, that would be chosen by the user according to its own needs, by modifying the configuration file.


log_hour_1353.xls - LibreOffice Calc
Arial
10
A1
PARAMETER
Find
Sheet 1 / 3
PageStyle_node_1_ip_add_192.168.1.1

	A	B	C	D	E
1	PARAMETER	UNIT	VALUE	VALUE	VALUE
2	Measurement time:		13:38:00	13:39:01	13:40:01
3	Active Power Total	[W]	----	----	----
4	Active Energy Total	[kWh]	0	0	0
5	Default Power Limit	[kW]	0.0	0.0	0.0
6	Evaluation Window	[min]	45	45	45
7	Elapsed Time	[min]	3	3	3
8	Mean Power	[kW]	0.0	0.0	0.0
9	Energy Log Index	index	0	0	0
10	Energy_log_0	[kW]	----	----	----
11	Energy_log_1	[kW]	----	----	----
12	Energy_log_2	[kW]	----	----	----
13	Energy_log_3	[kW]	----	----	----
14	Energy_log_4	[kW]	----	----	----
15	Energy_log_5	[kW]	----	----	----
16	Energy_log_6	[kW]	----	----	----
17	Energy_log_7	[kW]	----	----	----
18	Energy_log_8	[kW]	----	----	----
19	Energy_log_9	[kW]	----	----	----
20	Energy_log_10	[kW]	----	----	----
21	Energy_log_11	[kW]	----	----	----
22	Energy_log_12	[kW]	----	----	----
23	Energy_log_13	[kW]	----	----	----
24	Energy_log_14	[kW]	----	----	----
25	Energy_log_15	[kW]	----	----	----
26					
27					
28					
29					
30					
31					

node_1_ip_add_192.1 / node_2_ip_add_192.16 / node_3_ip_add_1

Figure 3.12: Resulting spreadsheet log file example.

When talking about the results, the main difference with the previous case is that the log files are generated every fifteen minutes (instead of every hour); but in this case, each log file contains information regarding all the involved controllers (a different sheet per every controller), while in the XML case, there was one file per node. In the figure 3.12 it is shown an example of a resulting spreadsheet log file.

Both formats:

The final and obvious possibility is to have both formats at once; which results from the combination of the previous procedures, because the iterations will respect the same time intervals and they would be constrained to use the same nodes. The results obtained in this case are exactly the same as in the independent cases, but having both kind of files at the same time. It has been defined as the default and recommended option, since in this way the user would have the advantages of both methods, such as complete log files every fifteen and sixty minutes, information for the nodes in separate files and combined within one, and the benefits of storing and managing information in both formats.

For any of the three previous cases, if there is a new input XML file, the process would be interrupted and the current log files would be saved as they are (i.e. most likely they wouldn't contain the information for the mentioned fifteen or sixty minutes): However the information for the available time would be complete; and new log files would start to be created.

3.7 Description of the functions

Due to scalability and flexibility reasons, both of the modules previously described make use of functions, in fact the modules themselves are also defined as functions, as a part of the main EMS tool. Each of these functions has a specific task in the execution of the program, and thats why it is useful to have a brief description of them.

systemsCal(ip):

Where **'ip'** is the IP address of the node. Returns **'date_info'**, which is a list containing the values for: day of the week, actual date (day and month) and actual time (hour, minute and second) in the controller.

The function reads the two registers (32 bits) of the Ekip controller that contain the information of a counter with a value in milliseconds, where t=0 means 00:00:00.000 of December 31st, 1999; and by performing some calculations, it determines the values that are later saved in the resulting 'date_info' list. It is worth to note that in order to calculate the current month, given the issues of leap years and the variations in the number of days for every month, the calculations in that case are not so straightforward, and there was the need of using well known algorithms[41] for accomplishing this task.

writeTimePeriods(input_list, day_initial, day_final):

Where **'input_list'** is the list with the time periods (resulting from *timePeriods(nodeID)* but with the IP address of the node instead of the nodeID); **'day_initial'** and **'day_final'** are the registers' addresses of the time windows, according to the day of

41 There is useful information in the following websites:

- http://www.millersville.edu/~bikenaga/number-theory/calendar/calendar.html
- http://alcor.concordia.ca/~gpkatch/gdate-method.html
- http://alcor.concordia.ca/~gpkatch/gdate-algorithm.html

the week[42]. Returns nothing.

The aim of this function is to write the time periods that have already been obtained from the input XML file, into the controller. It has to handle the specific writing mechanism defined by the developers of the Ekip controller, and write into the assigned registers, according to the day of the week.

daylyPowers(a, lim, multi_lists, times):

Where **'a'** is the number of nodes to be configured (got from the XML file with the power limits information); **'lim'** is the length of 'multi_lists'; **'multi_lists'** is the list of lists, with all the configuration information for all the nodes; and **'times'** is the list with the time periods. Returns **'powers'**, which is a list containing two values, the hour and its corresponding power limit, regarding each timestamp in the XML configuration file.

This function simply relates each hour value in the mentioned XML file with its corresponding power limit, when also making the conversion of units, since the values given in the XML files are in MW, and the ones required for the Ekip controllers have to be in kW, and in steps of tens.

writeDefPowers(def_power, ip):

Where **'def_power'** is the list with the four values of the default powers; and **'ip'** is the IP address of the node to be configured. Returns nothing.

42 The time windows have different registers assigned according to the day of the week, classified as: week (monday to friday), saturday and sunday.

This function simply writes the four values of the power limits for each hour, as the default power values of the controller; always following the writing procedure of the Ekip controllers. Here there is the need for clarification: The “default powers” that are being written are the four values defined by EMS for each hour (interpreted as default power values for a whole day by the Ekip controller), and not the real default powers in the input XML file. The 'trick' applied to the controller has to be taken into account.

hourlyPowers(hour, pow_info, h_info, bias):

Where **'hour'** corresponds to the current time (only the hour value) in the controller; **'pow_info'** is the list with the power values of the day for that controller; **'h_info'** is the list with the hour data of the day for that controller and **'bias'** indicates from which quarter hour there is available information in that specific time of the day. Returns **'ekip_powers'**, which is a list containing the power limits for the current hour.

This function obtains into a list the power limit values that may exist for the current hour, taking as reference the controller's time. Considering that the power limits information in the XML files is given at most each 15 minutes, there might be up to four values in the 'ekip_powers' list.

fillBlanks(val_list, defa):

Where **'val_list'** is the list of values to be processed and **'defa'** is the default power limit for the hour at which the configuration process of the Ekip controller is taking place. Returns **'powers_list'**, which is a list containing the same number of elements as the original list, but where the zeros have been replaced for the default power limits, while keeping unchanged (value and position in the list) those non-zero values.

writeMaximumPowers(prof, ip, day_pow):

Where **'prof'** is the profile to be assigned, depending on which quarter hour is currently happening; **'ip'** is the IP address of the node under configuration; and **'day_pow'** is the register in which the power values are written, according to the day of the week[43]. Returns nothing.

What this function does is to relate the maximum power values to the 'profiles', by writing the right value of the profile into the corresponding registers, as defined in the scheduling scheme of the Ekip controllers. There are four values of the maximum power for each hour, and by means of this function, these values are correctly set by modifying the profile every fifteen minutes. This function is essential in the 'trick' that has been used for updating the power limits in the controller every quarter hour; since through it the EMS will periodically rotate the default power limit from P1 to P4; where P1 will correspond to the first quarter hour, P2 to the second and so on.

xmlLog(ip, i):

Where **'ip'** is the IP address of the node from which the EMS (more specifically the ECLGT module) is getting the information; and **'i'** is the number of the same node, considering only the nodes from where the information for the log files is being retrieved. Returns nothing.

43 Same concept as with the function *"writeTimePeriods".*

This function reads the registers that contain the information useful for the construction of the log files, gives them format, put the data in the XML files, and also it creates the repetitive tags and parts of them that are defined in the generation of the log files in XML format.

xlsLog(sheet, i, ip):

Where **'sheet'** is the sheet where the tool is currently storing the log information for the created spreadsheet; **'i'** is the number of the node from which the software tool is getting the information, considering only the nodes from where the information for the log files is being retrieved; and **'ip'** is the IP address of the same node. Returns nothing.

Just like with the 'xmlLog' function, this one reads the registers that contain the information useful for the construction of the log files, gives them format, put the data into the current sheet, and also it creates the repetitive headers that are defined in the generation of the log files in XLS format (Spreadsheet).

def restart():

This function is used in the case where a new XML input file is found. What it does is to restart the program from the beginning. Since with the new file comes new information, restarting the program deletes all previously saved data, making everything ready for the new incoming content and that's also why the modifications done to the configuration file will take place after this happens.

Appendix F shows the detailed code of the functions here described.

3.8 How to use

The EMS tool has been created as a python script that after installing the corresponding dependencies, could be directly executed in Linux-based systems, however it has also been "ported" to Windows systems by means of the generation of an executable file (.exe). After running or executing the corresponding file, the user doesn't have to interact with the program because it will keep running independently as far as it is fed with new XML input files at least once a day. However, the EMS tool requires a configuration file that has to be stored in a specific location, and that will follow a certain format, as seen in Appendix B.

To summarize, the procedure to make EMS work properly is the following:

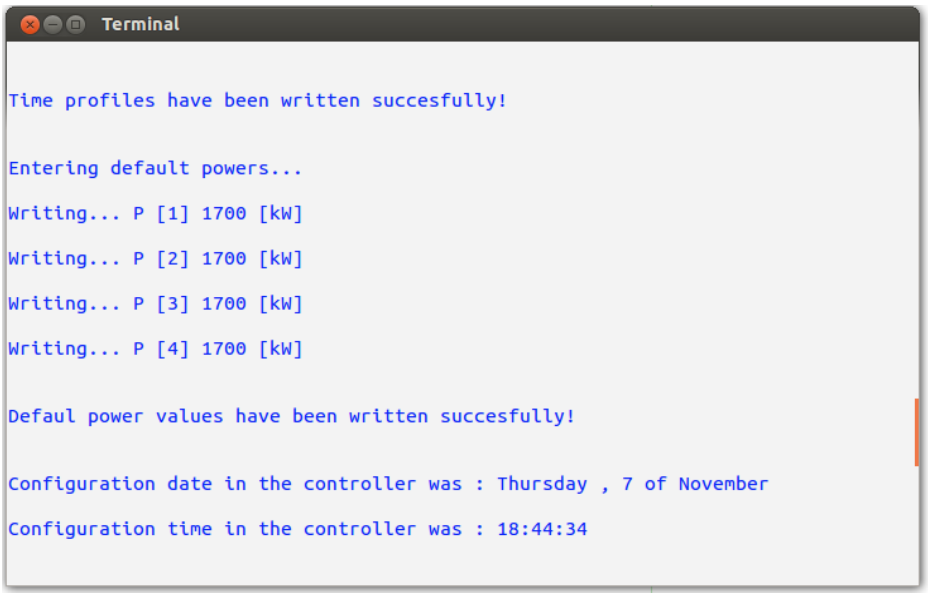
- Copy the configuration file (config.txt) in the directory: "/home/username/EMS/Config" for Linux systems or "C:\EMS\Config\" for Windows. In the first case, the directory could be easily modified before running the Python script.

- Modify the values in the configuration file, according the requirements of each case.

- Be sure that there is one and only one input XML file in the specified directory for this purpose. This file will be removed after being read by the software tool while waiting for a new one.

- Be sure that there is communication between the workstation and the Ekip controller(s).

After that, the EMS tool will show some results: either the values loaded into the controllers or the generation of log files, or both along with time stamps, as in the example of figure 3.13.



```
Time profiles have been written succesfully!

Entering default powers...
Writing... P [1] 1700 [kW]
Writing... P [2] 1700 [kW]
Writing... P [3] 1700 [kW]
Writing... P [4] 1700 [kW]

Defaul power values have been written succesfully!

Configuration date in the controller was : Thursday , 7 of November
Configuration time in the controller was : 18:44:34
```

Figure 3.13: Example of the result messages shown by the EMS tool

4 Tests & Results

“"... before you start planning a project, you have to understand why you need to carry out the project. Without understanding why you need the project, how will you be able to tell if you have succeeded?” -

< Kent Beck, Martin Fowler> ►[Planning Extreme Programming, 2001]

Summary. <In this chapter are presented the results of the project; how the EMS software works, and how it was tested during the design and development stages. Also there are recommendations for future work and conclusions about the project.>

4.1 Tests

The realization of tests has been an essential part of the design and development processes, from the first steps until the final release of the EMS tool. How the testing procedure has been evolving is just like a pyramid, where the wider approach happens in the first steps and the last ones are just for some adjustments. In this case every step is related to a specific activity and software tool, as depicted in figure 4.1.

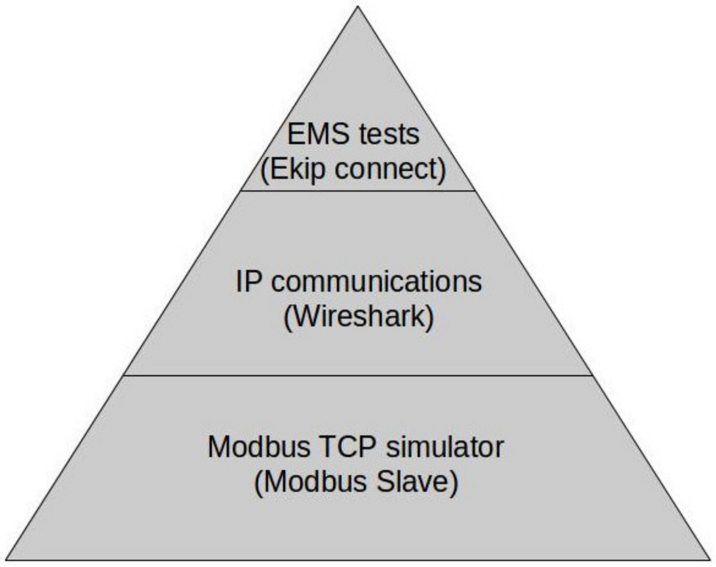


Figure 4.1: EMS software development - Testing process

In the base of this pyramid, the tests where initially done by using a Modbus TCP simulator[44] software, in order to understand the basic communication process with a Modbus capable device, how to read and write registers and modifying the preset values in a generic device. This part was also important in order to have a better understanding of how a Modbus device and the Pymodbus library work, and in order to create a strategy for the development of the software tool. At this point it was hard to conceive the full scheme that was going to be adopted and all the information gotten here was used to see the strengths and weaknesses of the available tools.

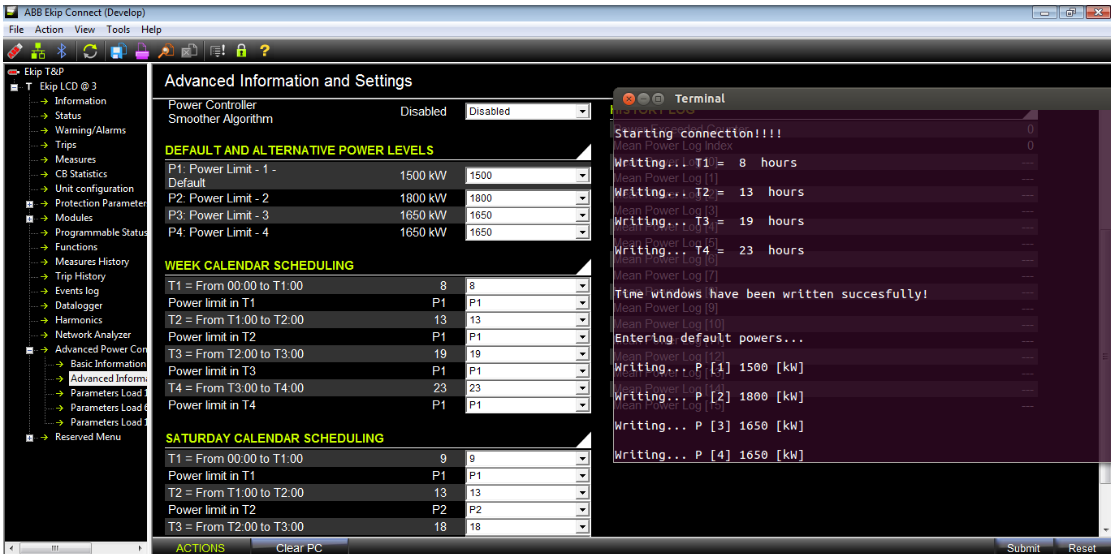
When the basics were set up correctly, there was the need to go to the upper step in the pyramid, i.e. perform the tests with a real Ekip controller. At that stage, a packet analyzer was another

44 The used simulator was 'Modbus Slave' (http://www.modbustools.com/modbus_slave.asp)

very useful tool[45], in order to have a complete understanding of the commands that were being both sent and received between the workstation and the controller. It was specially useful in order to 'tune' the communications scheme, providing the security that the commands can be sent and received with no trouble, leaving only the issues related to whether the controllers would respond in the way they were supposed to or not.

Finally, on top of the pyramid of the testing (and development) process it was needed to ensure that the configurations were being loaded and correctly used by the Ekip controller, and in other to do that it was required a tool that would communicate in parallel with the controller used for testing while the commands were being sent and received by the development workstation. To perform this task, there is a software called **Ekip Connect, developed by ABB** that can link to the Ekip controller via serial port or USB (the last one was used for tests) and display the configuration information. Ekip Connect can be used also to configure the controller, specially in the cases when the manual input of information is required, according to what has been mentioned in this work.

45 In the case of this project, 'Wireshark' (http://www.wireshark.org/) was the packet analyzer used for the tests.



Advanced Information and Settings

Power Controller Smoother Algorithm	Disabled	Disabled

DEFAULT AND ALTERNATIVE POWER LEVELS

P1: Power Limit - 1 - Default	1500 kW	1500
P2: Power Limit - 2	1800 kW	1800
P3: Power Limit - 3	1650 kW	1650
P4: Power Limit - 4	1650 kW	1650

WEEK CALENDAR SCHEDULING

T1 = From 00:00 to T1:00	8	8
Power limit in T1	P1	P1
T2 = From T1:00 to T2:00	13	13
Power limit in T2	P1	P1
T3 = From T2:00 to T3:00	19	19
Power limit in T3	P1	P1
T4 = From T3:00 to T4:00	23	23
Power limit in T4	P1	P1

SATURDAY CALENDAR SCHEDULING

T1 = From 00:00 to T1:00	9	9
Power limit in T1	P1	P1
T2 = From T1:00 to T2:00	13	13
Power limit in T2	P2	P2
T3 = From T2:00 to T3:00	18	18

Terminal

```
Starting connection!!!!

Writing... T1 =  8  hours

Writing... T2 =  13  hours

Writing... T3 =  19  hours

Writing... T4 =  23  hours

Time windows have been written succesfully!

Entering default powers...

Writing... P [1] 1500 [kW]

Writing... P [2] 1800 [kW]

Writing... P [3] 1650 [kW]

Writing... P [4] 1650 [kW]
```

Figure 4.2: Example of the final tests, using the Ekip Connect software

4.2 Results

The main result of this project is the management software for Ekip controllers that has been developed. The goal of the project was to **develop an interface** between the information about power limits and how they change during the day and the Ekip controllers that would have now a higher level of granularity (updated values every fifteen minutes) and an adaptive behavior (the power limits will constantly change). Originally the controllers define only four time ranges with their correspondent power limits and these values are kept fixed in time (with only three different schemes per week); which was changed by the adoption of the EMS tool, as it has been fully described in here.

As a result the EMS tool will make it possible to have a more precise management of the electric energy consumption, which will represent a more efficient usage of the resources and savings when paying the electric bills.

Another benefit that results from using the EMS tool is that the user doesn't have to worry about keeping a constant track of the information about the possible power limits for his/her installation, and also since it provides automatic configuration of some values, the user will have a smaller level of interaction with the controller or controllers.

Given that the communication with the controllers takes place via an IP network, it takes out the need of physically going to each node to perform the configuration processes, since this can be done remotely from one unique workstation towards a number of controllers from which the user only needs to know their IP addresses. So, it could performed even through the Internet.

The other important result of this work is that there will be log files stored in different ways and for all the nodes, which brings the possibility of generating statistic charts, tables, records, etc. that will be useful to improve the electric energy saving schemes and the functioning of the EMS tool.

4.3 Future work

Since it is the first work of this kind using the Ekip controller, there are many possibilities for future works. The first ones that come to mind are the determination of the precise amounts of electric energy and money that could be saved by using the EMS tool. In other words, the analysis of the economic and technical benefits of using this solution in a real life installation.

Also, there might be possibilities in the addition of new functions to the tool. Probably by having at disposition more information as input, the configuration of the controllers could reach a higher level of independence from the user. It would require additions and/or modifications to the source code of the EMS, and it is here that it would be obvious the benefits of having the main functionalities of the program defined as independent functions.

In the case of the logs generation, it would be useful to determine if the fields that have been included are actually useful for the user and whether there are some other fields that should be included in those files or the ones now included are enough. For this, it might be necessary also a statistical analysis of the results, which would provide a better idea of the behavior of the system and would justify or not the presence of those fields.

It would also be interesting to study the security issues that could come from the fact that all the configuration process happens over an IP network (which could even have public access). There might be vulnerabilities that could risk the stability of the whole electric installation, if they are found by hackers or computer criminals.

But of course the main field of study would be the contribution of this kind of tools in the area of energy efficiency, and in the development of better virtual power plant operation schemes.

4.4 Conclusions

From the personal point of view, this works has been very useful in order to increase my knowledge, mainly in three areas: networking, electric energy generation and programming. In the relative to **networking**, I had the need to review IP addressing schemes, the OSI model and perform tests where I had to analyze the structure of the IP packets that were being sent and received by the workstation, besides the need for understanding the way in which the Modbus standard operates. The **electric energy generation** was another main point, mainly because due to my background I had almost no knowledge about this topic, which has a great relevance nowadays given the need of the world to change the energy generation methods into more efficient and environmentally friendly ones. It happens to be a very interesting topic with many opportunities for research and development. Finally, another important knowledge that I've gained by means of this work is **programming** in Python language. So far I had limited opportunities for programming and they were mainly in C++, so this work might open a new door for me by giving me experience in a very useful and widely used programming language.

As for the work itself, it leaves clear the tendency of the world to move towards an scheme were the electricity is generated mainly by means of renewable sources, trying to adopt mechanisms that are environmentally friendly and also trying to implement solutions that make a more efficient use of this energy in order to satisfy the always growing needs for electricity in a cheaper way and wasting the fewer possible resources. This is also the reason why terms such as 'smart grids', 'energy efficiency', 'virtual power plants', etc. are becoming more and more common in publications, research projects, forums, etc.

The current technological development allows us to come up with interesting mechanisms for saving energy, produce it in a more environmentally friendly way and use it more efficiently; but also it causes a grow in the demand for that energy; and the challenge is to make the former move faster that the last.

This technology brings also a changing economic environment in which the pricing models in all the components of the electric market are always trying to adapt to the new situations, and the developers of new systems for energy efficiency have to adapt to them, too. This happens not only in Italy, but in most places.

From that situation it derives the fact that as well as many other companies, ABB is contributing to those kind of solutions that fit in the “energy efficiency” concept by developing new devices, creating statistical and mathematical models, software tools, etc. such as the ones used and referred in this project.

This is a fast changing environment in which there is the need for improvement in the systems, devices and all kind of solutions involved, bringing a lot of opportunities for studies, analysis, research and new implementations. And in that way, there are many future developments that could be derived from the situations analyzed in this work, and from its product and results.

Appendix A

Generation of the executable file. Here it is fully described the process followed in order to have a full executable file, that won't need the installation of any dependencies in order to work. This procedure is done in order to make the EMS tool work in a generic Windows machine.

Generation of an executable file in windows (running the tool in Windows):

1. Download Python 2.7 from here:

http://www.python.org/download/

2. Install easy_install. For that you need to install the setuptools that you will find here:

https://pypi.python.org/packages/2.7/s/setuptools/

After that, you will get the file easy_install.exe in the folder C:\Python27\Scripts.

Run the installer.

Otherwise, you can install pip, downloading it from here:

https://sites.google.com/site/pydatalog/python/pip-for-windows

3. Install Pymodbus.

There is a special situation occurring here. In order to be able to read registers, the default unit identifier value (0x00) has to be changed to '1' (0x01), because that is the way it is used in the Ekip controller. In order to do this, you have to modify the 'constants.py' file, as stated in the pymodbus tutorial. This has to be done BEFORE installation. So, the only way of installing pymodbus is by downloading it from the website:

http://code.google.com/p/pymodbus/downloads/list

During installation, it is possible that you find the following error:

Setup script exited with error: Unable to find vcvarsall.bat

In which case, you might need to install Microsoft Visual Studio.

4. Install LXML, from here:

http://lxml.de/index.html

Or you can try:

```
easy_install -U lxml
pip install  -U lxml
```

5. Install XLWT, from here:

https://pypi.python.org/pypi/xlwt

Or you can try:

```
easy_install -U xlwt
pip install  -U xlwt
```

At this point, you have finished installing all the dependencies, and the script containing the code of the EMS tool should work when run from a Python console.

Generation of the executable (standalone) file:

There are some software tools that can be used for this task, but in this case py2exe has been used; which can be downloaded from the site:

http://www.py2exe.org/

It is useful to follow the tutorials about how to generate the .exe files, and to take care of the dependencies. During the tests, pyserial and xlwt were the ones causing some trouble.

At this point, you will have the executable file that can be used in other machines, without doing all the process mentioned in this appendix. One has to take care of the files that will be needed in order to port the EMS software to other computers (it has to be kept in the same folder with those files).

Appendix B

Full example of the configuration file. This is the complete 'config.txt' file that the EMS tool uses for its operation.

```
######################################################################
#Configuration file for the EMS tool
######################################################################
#
#Please, respect the format and the order of the elements in this
#file.
#All the lines with comments or no information must begin with '#'.
#All the other lines include relevant parameters for the operation
#of
#this software, so they have to be modified carefully.
#
#####First parameter:
#This is the full path of the directory where EMS will look for the
#XML
#input file:
#
/home/julio/Desktop/EMS/Info
#You can specify any directory, according to your needs.
#
#
#####Second parameter:
#This is used to choose which modules are going to be used:
#
#1. Ekip Controller Automatic Configuration Tool (ECACT)
#2. Ekip Controller Logs Generation Tool (ECLGT)
#3. Both, one after the other (recommended option)
#
2
#Please, use only the numbers 1, 2 or 3.
#
#
#####Third parameter:
#This is the full path of the directory where the log files will be
#stored:
#
/home/julio/Desktop/EMS/Logs
#You can specify any directory, according to your needs.
#
#
#####Fourth parameter:
#This is used to choose the format of the log files to be generated:
#
#1. XML file
#2. Spreadsheet (.xls file)
#3. Both (recommended option)
#
3
```

```
#Please, use only the numbers 1, 2 or 3.
#
#
#####Fifth parameter:
#This is the full path of the directory where EMS will save (move)
#the
#old input XML files:
/home/julio/Desktop/EMS/Old
#You can specify any directory, according to your needs.
#
#
```

Appendix C

Full example of the input file. This is the complete XML file that the EMS tool (more precisely the ECACT module) will use as input file. It contains all the configuration information for all the involved nodes.

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"><SOAP-ENV
:Header/><SOAP-ENV:Body><sg:CreateEvent
xmlns:sg="http://services.gridpoint.com/">
<sg:schedule>
      <sg:eventIdent>140220121200</sg:eventIdent>
      <sg:externalSystemIdent>Smart Grid</sg:externalSystemIdent>
      <sg:identifiedBy/>
      <sg:programIdent>TWF_Pilot</sg:programIdent>
      <sg:sections>
            <sg:sectionSchedule>
                  <sg:participantCount>1</sg:participantCount>
                  <sg:schedule>
                        <sg:scheduleItem><sg:amount>.4812</sg:amount
><sg:dateTime>2012-02-14T01:15:00Z</sg:dateTime></sg:scheduleItem>
                        <sg:scheduleItem><sg:amount>.3888</sg:amount
><sg:dateTime>2012-02-14T01:30:00Z</sg:dateTime></sg:scheduleItem>
                        <sg:scheduleItem><sg:amount>1.8</sg:amount><
sg:dateTime>2012-02-14T11:45:00Z</sg:dateTime></sg:scheduleItem>
                        <sg:scheduleItem><sg:amount>1.2</sg:amount><
sg:dateTime>2012-02-14T13:00:00Z</sg:dateTime></sg:scheduleItem>
                        <sg:scheduleItem><sg:amount>1.5</sg:amount><
sg:dateTime>2012-02-14T13:15:00Z</sg:dateTime></sg:scheduleItem>
                        <sg:scheduleItem><sg:amount>1.8</sg:amount><
sg:dateTime>2012-02-14T14:00:00Z</sg:dateTime></sg:scheduleItem>
                        <sg:scheduleItem><sg:amount>1.5</sg:amount><
sg:dateTime>2012-02-14T14:45:00Z</sg:dateTime></sg:scheduleItem>
                        <sg:scheduleItem><sg:amount>1.8</sg:amount><
sg:dateTime>2012-02-14T19:30:00Z</sg:dateTime></sg:scheduleItem>
                        <sg:scheduleItem><sg:amount>1.8</sg:amount><
sg:dateTime>2012-02-14T20:45:00Z</sg:dateTime></sg:scheduleItem>
                  </sg:schedule>
                  <sg:defaultValues>
                        <sg:value_1>.4812</sg:value_1><sg:periodStar
t_1>00:00:00</sg:periodStart_1><sg:periodStop_1>08:00:00</sg:periodS
top_1>
                        <sg:value_2>1</sg:value_2><sg:periodStart_2>
08:00:00</sg:periodStart_2><sg:periodStop_2>13:00:00</sg:periodStop_
2>
                        <sg:value_3>1.753</sg:value_3><sg:periodStar
t_3>13:00:00</sg:periodStart_3><sg:periodStop_3>18:00:00</sg:periodS
top_3>
                        <sg:value_4>1.583</sg:value_4><sg:periodStar
t_4>18:00:00</sg:periodStart_4><sg:periodStop_4>24:00:00</sg:periodS
top_4>
                  </sg:defaultValues>
                  <sg:sectionIdent>FS1.1</sg:sectionIdent>
                  <sg:ipAddress>192.168.1.100</sg:ipAddress>
                  <sg:sectionLatitude>45.0685163</sg:sectionLatitude
>
                  <sg:sectionLongitude>7.6761308</sg:sectionLongitud
e>
            </sg:sectionSchedule>
            <sg:sectionSchedule>
                  <sg:participantCount />
                  <sg:schedule />
                  <sg:defaultValues />
                  <sg:sectionIdent>FS1.3</sg:sectionIdent>
                  <sg:ipAddress>192.168.1.100</sg:ipAddress>
                  <sg:sectionLatitude>45.073408</sg:sectionLatitude>
                  <sg:sectionLongitude>7.660646</sg:sectionLongitude
>
            </sg:sectionSchedule>
            <sg:sectionSchedule>
                  <sg:participantCount>1</sg:participantCount>
                  <sg:schedule>
                        <sg:scheduleItem><sg:amount>.26</sg:amount><
sg:dateTime>2012-02-14T18:45:00Z</sg:dateTime></sg:scheduleItem>
                        <sg:scheduleItem><sg:amount>.26</sg:amount><
sg:dateTime>2012-02-14T19:00:00Z</sg:dateTime></sg:scheduleItem>
                        <sg:scheduleItem><sg:amount>.26</sg:amount><
sg:dateTime>2012-02-14T19:30:00Z</sg:dateTime></sg:scheduleItem>
                  </sg:schedule>
                  <sg:defaultValues>
                        <sg:value_1>.9</sg:value_1><sg:periodStart_1
>00:00:00</sg:periodStart_1><sg:periodStop_1>09:00:00</sg:periodStop
_1>
                        <sg:value_2>1.8901</sg:value_2><sg:periodSta
rt_2>09:00:00</sg:periodStart_2><sg:periodStop_2>14:00:00</sg:period
Stop_2>
                        <sg:value_3>.3</sg:value_3><sg:periodStart_3
>14:00:00</sg:periodStart_3><sg:periodStop_3>19:00:00</sg:periodStop
_3>
                        <sg:value_4>.56</sg:value_4><sg:periodStart_
4>19:00:00</sg:periodStart_4><sg:periodStop_4>24:00:00</sg:periodSto
p_4>
                  </sg:defaultValues>
                  <sg:sectionIdent>FS1.5</sg:sectionIdent>
                  <sg:ipAddress>192.168.1.100</sg:ipAddress>
                  <sg:sectionLatitude>45.068295</sg:sectionLatitude>
                  <sg:sectionLongitude>7.683305</sg:sectionLongitude
>
            </sg:sectionSchedule>
            <sg:sectionSchedule>
                  <sg:participantCount>1</sg:participantCount>
                  <sg:schedule>
                        <sg:scheduleItem><sg:amount>1.0752</sg:amoun
t><sg:dateTime>2012-02-14T08:45:00Z</sg:dateTime></sg:scheduleItem>
                        <sg:scheduleItem><sg:amount>.744</sg:amount>
<sg:dateTime>2012-02-14T09:00:00Z</sg:dateTime></sg:scheduleItem>
                        <sg:scheduleItem><sg:amount>2.6</sg:amount><
sg:dateTime>2012-02-14T11:15:00Z</sg:dateTime></sg:scheduleItem>
                        <sg:scheduleItem><sg:amount>2.6</sg:amount><
sg:dateTime>2012-02-14T11:30:00Z</sg:dateTime></sg:scheduleItem>
```

```
                        <sg:scheduleItem><sg:amount>2.6</sg:amount><
sg:dateTime>2012-02-14T11:45:00Z</sg:dateTime></sg:scheduleItem>
                        <sg:scheduleItem><sg:amount>2.6</sg:amount><
sg:dateTime>2012-02-14T12:00:00Z</sg:dateTime></sg:scheduleItem>
                        <sg:scheduleItem><sg:amount>2.6</sg:amount><
sg:dateTime>2012-02-14T13:45:00Z</sg:dateTime></sg:scheduleItem>
                        <sg:scheduleItem><sg:amount>2.6</sg:amount><
sg:dateTime>2012-02-14T14:15:00Z</sg:dateTime></sg:scheduleItem>
                        <sg:scheduleItem><sg:amount>.4696</sg:amount
><sg:dateTime>2012-02-14T15:45:00Z</sg:dateTime></sg:scheduleItem>
                        <sg:scheduleItem><sg:amount>1.2408</sg:amoun
t><sg:dateTime>2012-02-14T16:00:00Z</sg:dateTime></sg:scheduleItem>
                        <sg:scheduleItem><sg:amount>1.6</sg:amount><
sg:dateTime>2012-02-14T18:15:00Z</sg:dateTime></sg:scheduleItem>
                        <sg:scheduleItem><sg:amount>2.6</sg:amount><
sg:dateTime>2012-02-14T18:30:00Z</sg:dateTime></sg:scheduleItem>
                        <sg:scheduleItem><sg:amount>3.6</sg:amount><
sg:dateTime>2012-02-14T23:00:00Z</sg:dateTime></sg:scheduleItem>
                        <sg:scheduleItem><sg:amount>1.8348</sg:amoun
t><sg:dateTime>2012-02-14T23:30:00Z</sg:dateTime></sg:scheduleItem>
                  </sg:schedule>
                  <sg:defaultValues>
                        <sg:value_1>.8</sg:value_1><sg:periodStart_1
>00:00:00</sg:periodStart_1><sg:periodStop_1>06:00:00</sg:periodStop
_1>
                        <sg:value_2>1.1</sg:value_2><sg:periodStart_
2>06:00:00</sg:periodStart_2><sg:periodStop_2>12:00:00</sg:periodSto
p_2>
                        <sg:value_3>2</sg:value_3><sg:periodStart_3>
12:00:00</sg:periodStart_3><sg:periodStop_3>20:00:00</sg:periodStop_
3>
                        <sg:value_4>.236</sg:value_4><sg:periodStart
_4>20:00:00</sg:periodStart_4><sg:periodStop_4>24:00:00</sg:periodSt
op_4>
                  </sg:defaultValues>
                  <sg:sectionIdent>FS1.8</sg:sectionIdent>
                  <sg:ipAddress>192.168.1.100</sg:ipAddress>
                  <sg:sectionLatitude>45.062225</sg:sectionLatitude>
                  <sg:sectionLongitude>7.688407</sg:sectionLongitude
>
            </sg:sectionSchedule>
      </sg:sections>
      <sg:startTime>2012-02-14T10:30:00Z</sg:startTime><sg:stopTime>
2012-02-14T19:00:00Z</sg:stopTime>
</sg:schedule>
</sg:CreateEvent></SOAP-ENV:Body></SOAP-ENV:Envelope>
```

Appendix D

ECACT function. In this appendix, it is possible to see the code written in Python for the ECACT function used in the homonym module, as part of the EMS code.

```
def ecact(lists, defaults, hours, nodes, j):
      #### (lists = list with the lists of info from the config
      file,       defaults = default power
      #### values for each time period, hours = time periods, nodes
      = number of nodes to configure, j = each of the up to 96
      iterations)

      len_lists = []
      all_info = [0]*96*2 #### The one with the power & hour values
      power_info = [0]*96 #### The one with the power values
      hour_info = [0]*96 #### The one with the hours
      hourly_powers = [0]*4
      i = 0 ### Initialize the node counter every iteration

      print ('\n***CONFIGURATION OF THE EKIP CONTROLLERS IN FUNCTION
      OF THE LOADED FILE***\n')

      for i in range (0, nodes):

            period_powers = [0]*50 ### Initialize for each node
            (powers is a time period)

            #### Information about the node
            print'\n\n\n\n\n****************************************
            ***********\nNode ID:',lists[i][0],'\nIP
            Address:',lists[i][1]

            if lists[i][0] != None: #### Checking valid nodes
                  print '\n\n\nTHE CONFIGURATION SECTION STARTS
                                    NOW!!!\n\n\n'

                  ################## GETS THE USEFUL INFO INTO LISTS
                              #############

                  ip_list[i] = lists[i][1]
                  ip_add = ip_list[i]

                  if len(lists[i]) >= 3:  #### Checking nodes that
                  have  information

                        #################### WRITING THE TIME
                        PERIODS ##################

                        time_period = hours[i]
                        writeTimePeriods (time_period, day_ini,
                        day_fin)
```

```
##################### DEFINITION OF PROFILES

profiles = [4369, ### bin 0001000100010001 -
P1 applied  for all the time periods
8738, ### bin 0010001000100010 - P2 applied
for all the time periods
17476,      ###  bin  0100010001000100  -  P3
applied for all the time periods
34952]      ###  bin  1000100010001000  -  P4
applied for all the time periods


system_time_ecact = systemsCal(ip_add)

### Iterations of 0,1,2,3
mins = system_time_ecact[4] #### mm
quarter = mins/15

profile_conf = profiles[quarter]
print '\nCurrent quarter:', quarter+1

##### Writing the registers
writeMaximumPowers  (profile_conf,  ip_add,
day_power)
time.sleep (3)  ### Pause - Time in seconds
                        (visualization)

############## AUTOMATIC UPDATE OF THE POWER
LIMITS

len_list = len(lists[i])
all_info = daylyPowers (i, len_list, lists,
                        time_periods)
power_info = all_info[1] #### Only the lists
of powers
hour_info = all_info[0] #### Only the lists
of hours
sys_time = system_time_ecact[3] ### HH
count = len (hour_info)
default_power = 0
bias = [0]*4

for x in range (0, count):
      ##### The case when we DO have info
      for the current hour
      if hour_info[x] == sys_time:  ###  Only
      the powers in the current hour
            bias_aux_1 = list (lists[i]
            [(x*2)+3])  ###  To  convert  the
            timestamps into lists, so I can
            extract the hour info easily
            bias_1 = (int (bias_aux_1[14] +
            bias_aux_1[15]))/15     ####  In
            which quarter there's the first
            info
            bias[0] = bias_1

            if len(lists[i]) >= ((x+1)*2)+3:
```

```
            bias_aux_2 = list
            (lists[i][((x+1)*2)+3])
            ### To convert the
            timestamps into lists, so
            I can extract the hour
            info easily
            bias_2 = (int
            (bias_aux_2[14] +
            bias_aux_2[15]))/15     ##
            ## In which quarter
            there's info
            bias[1] = bias_2

      if len(lists[i]) >= ((x+2)*2)+3:
            bias_aux_3 = list
            (lists[i][((x+2)*2)+3])
            ### To convert the
            timestamps into lists, so
            I can extract the hour
            info easily
            bias_3 = (int
            (bias_aux_3[14] +
            bias_aux_3[15]))/15     ##
            ## In which quarter
            there's info
            bias[2] = bias_3

      if len(lists[i]) >= ((x+3)*2)+3:
            bias_aux_4 = list
            (lists[i][((x+3)*2)+3])
            ### To convert the
            timestamps into lists, so
            I can extract the hour
            info easily
            bias_4 = (int
            (bias_aux_4[14] +
            bias_aux_4[15]))/15     ##
            ## In which quarter
            there's info
            bias[3] = bias_4

      hourly_powers = hourlyPowers
      (sys_time,  power_info,
      hour_info, bias)

      #### Assings the default value
      for m in range (1, 5):
            a = time_period[m]
            ### hour ini
            b = time_period[m+1]
            #### hour fin
            if sys_time >= a and
            sys_time <= b:
                  wildcard = float
                  (defaults[i][m])
      ### Extracting the needed value

      default_power = fillBlanks
```

```
                    (hourly_powers, wildcard)

                    ################ WRITING THE
                    DEFAULT POWER
                    VALUES ###############

                    writeDefPowers  (default_power,
                    ip_add)
                    break

        ##### The case when we DON'T have info for
        the current hour, but we do for the time
        period
        if default_power ==  0:
              for x in range (0, count):
                    if hour_info[x] != sys_time:
                          for m in range (1, 5):
                                a   =   time_period[m]
                                ### hour ini
                                b  =  time_period[m+1]
                                #### hour fin
                                if sys_time >= a and
                                sys_time <= b:
                                      wildcard =
                                      float
                                      (defaults[i]
                                      [m])  ###
                                Extracting the
                                needed value

                          default_power  =  fillBlanks
                          ([0]*4, wildcard)

                          ################  WRITING
                          THE DEFAULT POWER
                          VALUES ###############

                          writeDefPowers(default_pow
                          er, ip_add)
                          break

        #################### DATE / TIME CHECKING

        system_time_ecact = systemsCal(ip_add)
        print '\n\nConfiguration date in the
        controller was :',
        system_time_ecact[0],',',system_time_ecact[1
        ],'of',system_time_ecact[2]

        hh = '%02d' % system_time_ecact[3]  ####  To
        have a 2 digit format
        mm = '%02d' % system_time_ecact[4]
        ss = '%02d' % system_time_ecact[5]
        sys_timestamp = str (hh+':'+mm+':'+ss)

        print '\nConfiguration time in the
        controller was :', sys_timestamp
        time.sleep (2)  ### Pause - Time in seconds
```

```
                                                (visualization)

      ###############################################################
      #### No node messages
            else:
                  print '\n\n***NO INFORMATION FOR THIS NODE***\n\n'
                  time_periods.append(0)
                  time.sleep (2)  ### Pause - Time in seconds
                                          (visualization)

      return ip_list
```

Appendix E

Part of an example of the generated log files. This is (a part of) the text information contained inside any XML log file generated by the ECLGT module. Information regarding most minutes has been deleted, however it remains clear the format and information contained in the file.

```xml
<?xml version="1.0" encoding="utf-8"?>
<LOG>
      <Technical_info>
            <Time>13:38:01
                  <Active_power_total>----</Active_power_total>
                  <Unit_Power>[W]</Unit_Power>
                  <Active_energy_total>0</Active_energy_total>
                  <Unit_Energy>[kWh]</Unit_Energy>
                  <Default_power_limit>0.0</Default_power_limit>
                  <Unit_Default_power>[kW]</Unit_Default_power>
                  <Evaluation_window>45</Evaluation_window>
                  <Unit_evaluation_window>[min]</Unit_evaluation_window>
                  <Elapsed_time>3</Elapsed_time>
                  <Unit_Elapsed_time>[min]</Unit_Elapsed_time>
                  <Mean_power>0.0</Mean_power>
                  <Unit_Mean_power>[kW]</Unit_Mean_power>
                  <Energy_log_index>0</Energy_log_index>
                  <Unit_Energy_log_index>index</Unit_Energy_log_index>
                  <Energy_log_0>----</Energy_log_0>
                  <Unit_Energy_log>[kW]</Unit_Energy_log>
                  <Energy_log_1>----</Energy_log_1>
                  <Unit_Energy_log>[kW]</Unit_Energy_log>
                  <Energy_log_2>----</Energy_log_2>
                  <Unit_Energy_log>[kW]</Unit_Energy_log>
                  <Energy_log_3>----</Energy_log_3>
                  <Unit_Energy_log>[kW]</Unit_Energy_log>
                  <Energy_log_4>----</Energy_log_4>
                  <Unit_Energy_log>[kW]</Unit_Energy_log>
                  <Energy_log_5>----</Energy_log_5>
                  <Unit_Energy_log>[kW]</Unit_Energy_log>
                  <Energy_log_6>----</Energy_log_6>
                  <Unit_Energy_log>[kW]</Unit_Energy_log>
                  <Energy_log_7>----</Energy_log_7>
                  <Unit_Energy_log>[kW]</Unit_Energy_log>
                  <Energy_log_8>----</Energy_log_8>
                  <Unit_Energy_log>[kW]</Unit_Energy_log>
                  <Energy_log_9>----</Energy_log_9>
                  <Unit_Energy_log>[kW]</Unit_Energy_log>
                  <Energy_log_10>----</Energy_log_10>
                  <Unit_Energy_log>[kW]</Unit_Energy_log>
                  <Energy_log_11>----</Energy_log_11>
                  <Unit_Energy_log>[kW]</Unit_Energy_log>
                  <Energy_log_12>----</Energy_log_12>
                  <Unit_Energy_log>[kW]</Unit_Energy_log>
                  <Energy_log_13>----</Energy_log_13>
                  <Unit_Energy_log>[kW]</Unit_Energy_log>
                  <Energy_log_14>----</Energy_log_14>
                  <Unit_Energy_log>[kW]</Unit_Energy_log>
                  <Energy_log_15>----</Energy_log_15>
                  <Unit_Energy_log>[kW]</Unit_Energy_log>
```

```
        </Time>
        <Time>13:39:01
              <Active_power_total>----</Active_power_total>
              <Unit_Power>[W]</Unit_Power>
              <Active_energy_total>0</Active_energy_total>
              <Unit_Energy>[kWh]</Unit_Energy>
              <Default_power_limit>0.0</Default_power_limit>
              <Unit_Default_power>[kW]</Unit_Default_power>
              <Evaluation_window>45</Evaluation_window>
              <Unit_evaluation_window>[min]</Unit_evaluation_window>
              <Elapsed_time>3</Elapsed_time>
              <Unit_Elapsed_time>[min]</Unit_Elapsed_time>
              <Mean_power>0.0</Mean_power>
              <Unit_Mean_power>[kW]</Unit_Mean_power>
              <Energy_log_index>0</Energy_log_index>
              <Unit_Energy_log_index>index</Unit_Energy_log_index>
              <Energy_log_0>----</Energy_log_0>
              <Unit_Energy_log>[kW]</Unit_Energy_log>
              <Energy_log_1>----</Energy_log_1>
              <Unit_Energy_log>[kW]</Unit_Energy_log>
              <Energy_log_2>----</Energy_log_2>
              <Unit_Energy_log>[kW]</Unit_Energy_log>
              <Energy_log_3>----</Energy_log_3>
              <Unit_Energy_log>[kW]</Unit_Energy_log>
              <Energy_log_4>----</Energy_log_4>
              <Unit_Energy_log>[kW]</Unit_Energy_log>
              <Energy_log_5>----</Energy_log_5>
              <Unit_Energy_log>[kW]</Unit_Energy_log>
              <Energy_log_6>----</Energy_log_6>
              <Unit_Energy_log>[kW]</Unit_Energy_log>
              <Energy_log_7>----</Energy_log_7>
              <Unit_Energy_log>[kW]</Unit_Energy_log>
              <Energy_log_8>----</Energy_log_8>
              <Unit_Energy_log>[kW]</Unit_Energy_log>
              <Energy_log_9>----</Energy_log_9>
              <Unit_Energy_log>[kW]</Unit_Energy_log>
              <Energy_log_10>----</Energy_log_10>
              <Unit_Energy_log>[kW]</Unit_Energy_log>
              <Energy_log_11>----</Energy_log_11>
              <Unit_Energy_log>[kW]</Unit_Energy_log>
              <Energy_log_12>----</Energy_log_12>
              <Unit_Energy_log>[kW]</Unit_Energy_log>
              <Energy_log_13>----</Energy_log_13>
              <Unit_Energy_log>[kW]</Unit_Energy_log>
              <Energy_log_14>----</Energy_log_14>
              <Unit_Energy_log>[kW]</Unit_Energy_log>
              <Energy_log_15>----</Energy_log_15>
              <Unit_Energy_log>[kW]</Unit_Energy_log>
        </Time>
        <Time>13:40:01
              <Active_power_total>----</Active_power_total>
              <Unit_Power>[W]</Unit_Power>
              <Active_energy_total>0</Active_energy_total>
              <Unit_Energy>[kWh]</Unit_Energy>
              <Default_power_limit>0.0</Default_power_limit>
              <Unit_Default_power>[kW]</Unit_Default_power>
              <Evaluation_window>45</Evaluation_window>
              <Unit_evaluation_window>[min]</Unit_evaluation_window>
              <Elapsed_time>3</Elapsed_time>
              <Unit_Elapsed_time>[min]</Unit_Elapsed_time>
              <Mean_power>0.0</Mean_power>
              <Unit_Mean_power>[kW]</Unit_Mean_power>
              <Energy_log_index>0</Energy_log_index>
              <Unit_Energy_log_index>index</Unit_Energy_log_index>
              <Energy_log_0>----</Energy_log_0>
```

```
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_1>----</Energy_log_1>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_2>----</Energy_log_2>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_3>----</Energy_log_3>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_4>----</Energy_log_4>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_5>----</Energy_log_5>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_6>----</Energy_log_6>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_7>----</Energy_log_7>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_8>----</Energy_log_8>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_9>----</Energy_log_9>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_10>----</Energy_log_10>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_11>----</Energy_log_11>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_12>----</Energy_log_12>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_13>----</Energy_log_13>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_14>----</Energy_log_14>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_15>----</Energy_log_15>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
             </Time>
             <Time>13:41:01
                    <Active_power_total>----</Active_power_total>
                    <Unit_Power>[W]</Unit_Power>
                    <Active_energy_total>0</Active_energy_total>
                    <Unit_Energy>[kWh]</Unit_Energy>
                    <Default_power_limit>0.0</Default_power_limit>
                    <Unit_Default_power>[kW]</Unit_Default_power>
                    <Evaluation_window>45</Evaluation_window>
                    <Unit_evaluation_window>[min]</Unit_evaluation_window>
                    <Elapsed_time>3</Elapsed_time>
                    <Unit_Elapsed_time>[min]</Unit_Elapsed_time>
                    <Mean_power>0.0</Mean_power>
                    <Unit_Mean_power>[kW]</Unit_Mean_power>
                    <Energy_log_index>0</Energy_log_index>
                    <Unit_Energy_log_index>index</Unit_Energy_log_index>
                    <Energy_log_0>----</Energy_log_0>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_1>----</Energy_log_1>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_2>----</Energy_log_2>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_3>----</Energy_log_3>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_4>----</Energy_log_4>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_5>----</Energy_log_5>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_6>----</Energy_log_6>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_7>----</Energy_log_7>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_8>----</Energy_log_8>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_9>----</Energy_log_9>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_10>----</Energy_log_10>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_11>----</Energy_log_11>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_12>----</Energy_log_12>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_13>----</Energy_log_13>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_14>----</Energy_log_14>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_15>----</Energy_log_15>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
             </Time>
             <Time>13:42:01
                    <Active_power_total>----</Active_power_total>
                    <Unit_Power>[W]</Unit_Power>
                    <Active_energy_total>0</Active_energy_total>
                    <Unit_Energy>[kWh]</Unit_Energy>
                    <Default_power_limit>0.0</Default_power_limit>
                    <Unit_Default_power>[kW]</Unit_Default_power>
                    <Evaluation_window>45</Evaluation_window>
                    <Unit_evaluation_window>[min]</Unit_evaluation_window>
                    <Elapsed_time>3</Elapsed_time>
                    <Unit_Elapsed_time>[min]</Unit_Elapsed_time>
                    <Mean_power>0.0</Mean_power>
                    <Unit_Mean_power>[kW]</Unit_Mean_power>
                    <Energy_log_index>0</Energy_log_index>
                    <Unit_Energy_log_index>index</Unit_Energy_log_index>
                    <Energy_log_0>----</Energy_log_0>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_1>----</Energy_log_1>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_2>----</Energy_log_2>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_3>----</Energy_log_3>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_4>----</Energy_log_4>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_5>----</Energy_log_5>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_6>----</Energy_log_6>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_7>----</Energy_log_7>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_8>----</Energy_log_8>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_9>----</Energy_log_9>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_10>----</Energy_log_10>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_11>----</Energy_log_11>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_12>----</Energy_log_12>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_13>----</Energy_log_13>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_14>----</Energy_log_14>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
                    <Energy_log_15>----</Energy_log_15>
                    <Unit_Energy_log>[kW]</Unit_Energy_log>
             </Time>
      </Technical_info>
</LOG>
```

Appendix F

Functions used in the EMS tool. In this appendix, it is possible to see the code written in Python for all the functions used in the ECACT and ECLGT modules, as part of the EMS code.

systemsCal(ip):

```
def systemsCal(ip):
      #### (ip = ip addreess of the node)

      #### Starting connection
      client = ModbusTcpClient(ip)

      read_time  =  client.read_input_registers(3,2)  ##  Time  counter  in
      seconds
      ###(address, count)

      ###To manage the 2-bit, 2-register format used by the controller
      time1 = '{0:016b}'.format(read_time.registers [0])
      time2 = '{0:016b}'.format(read_time.registers [1])
      time_bin = time2 + time1   #### Total counter in binary

      #### Closing connection
      client.close()

      ######################################################################
      ###
      #### Process the counter info
      #### 86400 seconds in one day

      time_tot = int (time_bin, 2)
      time_2012  =  378691200  ####  2012,  00:00  ###  Min  time  for  the
      controller

      time_count = time_tot - time_2012 ####  Counter  in  the  interval  of
      interest

      days_count = time_count/86400 #### Counter converted into days since
      2012

      years_count = (days_count/365.2425)+2012 #### Starts in the min year
      (2012)
      actual_year = int (years_count)

      actual_day_count = (years_count - actual_year) * 365.2425 ### Days in
      a year

      ######################################################################
      ##### Values that I need (time)

      actual_hour = (time_count - (days_count * 86400)) / 3600

      actual_min  =  (time_count  -  ((days_count  *  86400)  +  actual_hour  *
      3600)) / 60

      actual_sec  =  time_count  -  ((days_count  *  86400)  +  (actual_hour  *
3600) +      (actual_min * 60))

#####################################################################
##### Values that I need (date)

day_aux = int (actual_day_count)

######################################################################
#### MONTH
#### Corrections for the year to start in march

if actual_year == 2012 or actual_year == 2016 or actual_year == 2020:
### leap year
       if day_aux <= 60: #### January and February sum 60 days in a
       leap   year
              day_in_year = day_aux + 366
       else:
              day_in_year = day_aux

       print day_in_year
       day_new = day_in_year - 61 #### compensates the correction

else:
       if day_aux <= 59:   #### January and February sum 60 days in a
       leap   year
              day_in_year = day_aux + 365
       else:
              day_in_year = day_aux

       day_new = day_in_year - 60 #### compensates the correction

month = ((100*day_new) + 52)/3060 ### As indicated in the web page

dic_month = {0 : 'March',
       1 : 'April',
       2 : 'May',
       3 : 'June',
       4 : 'July',
       5 : 'August',
       6 : 'September',
       7 : 'October',
       8 : 'November',
       9 : 'December',
       10: 'January',
       11: 'February'
       }

mm = dic_month[month]

######################################################################
#### DATE

actual_date = day_new - ((month*306 + 5)/10) + 1     ### As indicated
in the web page

######################################################################
#### DAY OF THE WEEK

weekday_aux = days_count % 7 #### Day of the week repeats every 7
days
#### Considering that the first possibe day (2012-01-01) was a Sunday

dic_day = {1 : 'Sunday',
       2 : 'Monday',
       3 : 'Tuesday',
        4 : 'Wednesday',
        5 : 'Thursday',
        6 : 'Friday',
        0 : 'Saturday'
        }

    dd = dic_day[weekday_aux]

    date_info   =   [dd,   actual_date,   mm,   actual_hour,   actual_min,
    actual_sec]  #### Generates a list with the system's date and time

    return date_info
```

writeTimePeriods (input_list, day_initial, day_final):

```
def writeTimePeriods (input_list, day_initial, day_final):
      ####  (input_list  =  time  periods  list;  day_initial,  day_final  =
      register
      #### addresses for the days of the week)

      ##############################Establish connection with the client
      client = ModbusTcpClient(input_list[0])
      print "\n\nStarting connection!!!!"

      ####################################### Write the desired registers

      ####### Writing the time periods -- It always starts at 00:00
      ###End of the first and second time windows

      ###To convert the hours into the 2-bit
      ###format managed by the controller
      window1 = '{0:08b}'.format(input_list[2])
      window2 = '{0:08b}'.format(input_list[3])
      w1and2 = window2 + window1
      write_1 = int(w1and2, 2)

      ############### WE START THE "WRITING COMMAND" #####################

      ### Start to write (4 in address 0)
      client.write_registers(0,[4,0])
      ####(address, [value1, value2]*number of registers) ## There are 2
      bits   in one register
      time.sleep (0.5)  ### Pause - Time in seconds

      ############## Here we write the desired values ####################

      print '\nWriting... T1 = ', input_list[2], ' hours'
      print '\nWriting... T2 = ', input_list[3], ' hours'
      ## day_ini is the number of the register for time windows
      client.write_registers(day_initial, [write_1])
      #### (address, [value1, value2]*number of registers)

      ###################################################################
      ###End of the third and fourth time windows

      ###To  convert  the  hours  into  the  2-bit  format  managed  by  the
      controller
      window3 = '{0:08b}'.format(input_list[4])
      window4 = '{0:08b}'.format(input_list[5])
      w3and4 = window4 + window3
      write_2 = int(w3and4, 2)
      print '\nWriting... T3 = ', input_list[4], ' hours'
      print '\nWriting... T4 = ', input_list[5], ' hours'
      ## day_fin is the number of the register for time windows
      client.write_registers(day_final, [write_2])
      ####(address, [value1, value2]*number of registers)

      ############### WE FINISH THE "WRITING COMMAND" ####################

      time.sleep (0.5)  ### Pause - Time in seconds
      ###### Close the writing session (6 in adress 0)
      client.write_registers(0,[6,0])
      ####(address, [value1, value2]*number of registers) ## There are 2
      bits   in one register
      time.sleep (0.5)  ### Pause - Time in seconds

      print '\n\nTime windows have been written succesfully!'

      ############# close the client
      client.close()

      return
```

daylyPowers(a, lim, multi_lists, times):

```
def daylyPowers(a, lim, multi_lists, times):
      ####  (a=node;  lim=lenght  of  'lists';  multi_lists  =  'lists';
      times=time   periods)

      config_hour = [0]*96
      config_power = [0]*96
      powers = [0]*2
      aux_1 = aux_2 = 0 #### Initialize

      for x in range (2, lim):
             if x % 2 != 0:   #### Odd values (hours)

      ######################################################################
      #### Analizing the time periods

                    hour_aux = list (multi_lists[a][x]) ### To convert the
                    timestamps into lists, so I can extract the hour info
                    easily
                    hour_str = hour_aux[11] + hour_aux[12]
                    config_hour[x] = int (hour_str)  ### I have the hour
                    info as integer

      ######################################################################
      #### Analizing the power information

                    power_aux = float (multi_lists[a][x-1]) ### To convert
                    the amounts
                    #### into floats, so I can extract the info easily

                    config_power[x] = ((int(power_aux*100))*100)/10  ### To
                    compensate MW to kW
                    #### To get the value in multiples of 10 (as used in the
                                  controller)

      aux_1 = filter(lambda a: a != 0, config_hour)
      aux_2 = filter(lambda a: a != 0, config_power) ### Deletes all the
```

```
        zeros

        powers = [aux_1, aux_2]

        return powers
```

writeDefPowers (def_power, ip):

```
def writeDefPowers (def_power, ip):
        #### (def_power = the 4 default powers, ip = ip address)

        #################################Establish connection with the client
        client = ModbusTcpClient(ip)
        print '\n\nEntering default powers... '

        for x in range (0,4):
                ############### WE START THE "WRITING COMMAND"

                ### Start to write (4 in address 0)
                client.write_registers(0,[4,0])
                ####(address, [value1, value2]*number of registers) ## There
                are 2 bits in one register
                time.sleep (0.5)  ### Pause - Time in seconds

                ############## Here we write the desired values
                reg = 4702 + x
                ### 4702 is the number of the first register for power limits
                write_power = def_power [x]  ### aux variable

                if write_power != 0:
                        client.write_registers(reg,[write_power])
                        #### (address, [value1, value2]*number of registers)
                        print '\nWriting... P',[x+1], write_power, '[kW]'

                else:
                        print '\nOld power values are being mantained for: P',
                        [x+1]
                        #### This should never happen

                ############### WE FINISH THE "WRITING COMMAND"

                time.sleep (0.5)  ### Pause - Time in seconds
                ###### Close the writing session (6 in adress 0)
                client.write_registers(0,[6,0])
                ####(address, [value1, value2]*number of registers) ## There
                are 2 bits in one register
                time.sleep (0.5)  ### Pause - Time in seconds

        print '\n\nDefaul power values have been written succesfully!'

        #### close the client
        client.close()

        return
```

hourlyPowers (hour, pow_info, h_info, bias):

```
def hourlyPowers (hour, pow_info, h_info, bias):
      #### (hour = time in the controller, power_info = lists with powers
      for    the day,
      #### h_info = lists with hours for the day; bias = from which quarter
      hour to start)

      a = 0
      ekip_powers = [0]*4

      count = len (h_info)
      for x in range (0, count):
            if h_info[x] == hour:      ### checks valid data for one hour
                  ind = bias[a]
                  ekip_powers[ind] = pow_info[x]  #### saving those values
                  into lists
                  a=a+1

      return ekip_powers
```

fillBlanks (val_list, defa):

```
def fillBlanks (val_list, defa):
      #### (val_list = the list with the values to ve averaged; defa =
      default power value)

      powers_list = [0]*4
      write_def = ((int(defa*100))*100)/10 ### To compensate MW to kW
                  #### To get the value in multiples of 10 (as used in the
                              controller)

      for x in range (0, 4):
            if val_list[x] != 0:       #### the final list keep the values
            that aren't 0
                  powers_list[x] = val_list[x]

            else:
                  powers_list[x] = write_def

      return powers_list
```

writeMaximumPowers (prof, ip, day_pow):

```
def writeMaximumPowers (prof, ip, day_pow):
      #### (prof = profile according to which quarter of hour, ip = ip
      address, day_pow =
      #### register for the powers according to the day)

      ################################Establish connection with the client
      client = ModbusTcpClient(ip)
      print '\nLoading information!!!!'

      ############### WE START THE "WRITING COMMAND" #####################
```

```
        ### Start to write (4 in address 0)
        client.write_registers(0,[4,0])
        ####(address, [value1, value2]*number of registers) ## There are 2
        bits   in one register
        time.sleep (0.5)  ### Pause - Time in seconds

        ############## Here we write the desired values ####################
        client.write_registers(day_pow,[prof])
        #### (address, [value1, value2]*number of registers)
        ### day_pow is the register for power limits according to the day

        ############### WE FINISH THE "WRITING COMMAND" ####################

        time.sleep (0.5)  ### Pause - Time in seconds
        ###### Close the writing session (6 in adress 0)
        client.write_registers(0,[6,0])
        ####(address, [value1, value2]*number of registers) ## There are 2
        bits   in one register
        time.sleep (0.5)  ### Pause - Time in seconds

        print '\n\nTime profiles have been written succesfully!'

        #### close the client
        client.close()

        return
```

xmlLog(ip, i):

```
def xmlLog(ip, i):
        #### (ip address, node counter)

        #### Time stamps
        ###Prints the date
        timestamp_aux = systemsCal(ip)

        hh = '%02d' % timestamp_aux[3]   #### To have a 2 digit format
        mm = '%02d' % timestamp_aux[4]
        ss = '%02d' % timestamp_aux[5]
        sys_timestamp = str (hh+':'+mm+':'+ss)

        print '\nCurrent time for node:', [i+1],'is:\n', sys_timestamp
        logfile[i].write('\n\t\t<Time>')
        logfile[i].write(sys_timestamp) ### Writes the info in the created
        file

        ####################################################################
        #### Start Connection
        client = ModbusTcpClient(ip)

        ####################################################################
        #### Active Power Total
        #### Reads the value for the total active power

        read_val = client.read_input_registers(206,2) ## Active Power total
        ### 206,2
        ###(address, count)

        ###To manage the 2-bit, 2-register format used by the controller
        val1 = '{0:016b}'.format(read_val.registers [0])
        val2 = '{0:016b}'.format(read_val.registers [1])
```

```
val = val2 + val1
power = int (val, 2)

if power != 2147483647:    #### 7FFFFFFF hex
       total_power = power * 0.1  ### As defined for the controller
       tot_pow = str (total_power)   ### The information has to be
writen as a string

else:
       tot_pow = '----'    #### No info available

logfile[i].write('\n\t\t\t<Active_power_total>')
logfile[i].write(tot_pow) ### Writes the info in the created file
logfile[i].write('</Active_power_total>')

logfile[i].write('\n\t\t\t<Unit_Power>')
logfile[i].write('[W]') ### Writes the info in the created file
logfile[i].write('</Unit_Power>')

######################################################################
#### Active Energy Total

#### Reads the value for the total active energy

read_ener = client.read_input_registers(304,2) ## Active Energy total
## 304,2
###(address, count)

###To manage the 2-bit, 2-register format used by the controller
ener1 = '{0:016b}'.format(read_ener.registers [0])
ener2 = '{0:016b}'.format(read_ener.registers [1])
ener = ener2 + ener1

energy = int (ener, 2)
tot_ener = str (energy)  ### The information has to be writen as a
string

logfile[i].write('\n\t\t\t<Active_energy_total>')
logfile[i].write(tot_ener) ### Writes the info in the created file
logfile[i].write('</Active_energy_total>')

logfile[i].write('\n\t\t\t<Unit_Energy>')
logfile[i].write('[kWh]') ### Writes the info in the created file
logfile[i].write('</Unit_Energy>')

######################################################################
#### Default Power Limit

#### Reads the value for the default power limit, according to the
current time period
read_actualpwr = client.read_input_registers(4751,1)
#### Actual Power Limit ## 4751,1

actual_power = str (read_actualpwr.registers[0] * 0.1 )   ### The
information has to be writen as a string

logfile[i].write('\n\t\t\t<Default_power_limit>')
logfile[i].write(actual_power) ### Writes the info in the created
file
logfile[i].write('</Default_power_limit>')

logfile[i].write('\n\t\t\t<Unit_Default_power>')
logfile[i].write('[kW]') ### Writes the info in the created file
logfile[i].write('</Unit_Default_power>')
```

```
######################################################################
#### Evaluation Window

#### Reads the value for the evaluation window
read_window = client.read_input_registers(4709,1)
#### Measurement time ## 4709,1

#### The info about measurement time is given as a table of possible
values
dic_eval = {0 : '5',
1 : '10',
2 : '15',
3 : '20',
4 : '30',
5 : '40',
6 : '45',
7 : '60',
8 : '90',
9 : '120',
10: '180',
11: '240'
}

eval_window = dic_eval[read_window.registers[0]]  ### The information
has to be writen in minutes

logfile[i].write('\n\t\t\t<Evaluation_window>')
logfile[i].write(eval_window) ### Writes the info in the created file
logfile[i].write('</Evaluation_window>')

logfile[i].write('\n\t\t\t<Unit_evaluation_window>')
logfile[i].write('[min]') ### Writes the info in the created file
logfile[i].write('</Unit_evaluation_window>')

######################################################################
#### Elapsed Time
#### Reads the value for the elapsed time

read_elapsed = client.read_input_registers(4753,1)
#### Time elapsed inside the measurement window ## 4753,1

elapsed_time = str (read_elapsed.registers[0])  ### The information
has to be writen as a string

logfile[i].write('\n\t\t\t<Elapsed_time>')
logfile[i].write(elapsed_time)  ###  Writes  the  info  in  the  created
file
logfile[i].write('</Elapsed_time>')

logfile[i].write('\n\t\t\t<Unit_Elapsed_time>')
logfile[i].write('[min]') ### Writes the info in the created file
logfile[i].write('</Unit_Elapsed_time>')

######################################################################
#### Log Registers

#####################################################################
#### Reads the value for all the remaining registers (they are in one
block)

log_registers = client.read_input_registers(4764,18)
####  Time  elapsed  inside  the  measurement  window  ##  4764,18;  18
registers

client.close()
```

```
######################################################################
#### Mean Power
mean_pow = log_registers.registers[0] * 0.1  ### As indicated in the
Modbus map
mean_power = str (mean_pow)  ### The information has to be writen as
a string

logfile[i].write('\n\t\t\t<Mean_power>')
logfile[i].write(mean_power) ### Writes the info in the created file
logfile[i].write('</Mean_power>')

logfile[i].write('\n\t\t\t<Unit_Mean_power>')
logfile[i].write('[kW]') ### Writes the info in the created file
logfile[i].write('</Unit_Mean_power>')

######################################################################
### Energy Log Index

energy_log  =  log_registers.registers[1]   ###  The  value  indicates
which  index is active
energy_log_index = str (energy_log)  ### The information has to be
writen as a string

logfile[i].write('\n\t\t\t<Energy_log_index>')
logfile[i].write(energy_log_index) ### Writes the info in the created
file
logfile[i].write('</Energy_log_index>')

logfile[i].write('\n\t\t\t<Unit_Energy_log_index>')
logfile[i].write('index') ### Doesn't have a physical unit
logfile[i].write('</Unit_Energy_log_index>')

######################################################################
##### Energy LogS

for x in range (0,16):  #### 16 log registers

       if log_registers.registers[2+x] != 65535: #### FFFF hex
              logs = log_registers.registers[2+x] * 0.1  ### The next
              register
              log_index = str (logs)  ### The information has to be
              writen as a string

       else:
              log_index = '----'  #### No info available

       #### Generates the tag names
       count = str (x)
       ini_tag = '\n\t\t\t<Energy_log_'+count+'>'
       log_tag1 = str (ini_tag)
       fin_tag = '</Energy_log_'+count+'>'
       log_tag2 = str (fin_tag)

       logfile[i].write(log_tag1)
       logfile[i].write(log_index) ### Writes the info in the created
       file
       logfile[i].write(log_tag2)

       logfile[i].write('\n\t\t\t<Unit_Energy_log>')
       logfile[i].write('[kW]')  ###  Writes  the  info  in  the  created
       file
       logfile[i].write('</Unit_Energy_log>')

return
```

xlsLog(sheet, i, ip):

```
def xlsLog(sheet, i, ip):
      #### (generated sheet, cell counter, ip address)

      #### Time stamps and headers
      ###Writes the header
      sheet.write(0, i+2, 'VALUE', style)

      ###Writes the date
      timestamp_aux = systemsCal(ip)

      hh = '%02d' % timestamp_aux[3]   #### To have a 2 digit format
      mm = '%02d' % timestamp_aux[4]
      ss = '%02d' % timestamp_aux[5]
      sys_timestamp = str (hh+':'+mm+':'+ss)

      sheet.write(1, i+2, sys_timestamp, style_data)

      ######################################################################
      #### Start Connection
      client = ModbusTcpClient(ip)

      ######################################################################
      #### Active Power Total
      #### Reads the value for the total active power

      read_val = client.read_input_registers(206,2) ## Active Power total
      ##     206,2

      ###To manage the 2-bit, 2-register format used by the controller
      val1 = '{0:016b}'.format(read_val.registers [0])
      val2 = '{0:016b}'.format(read_val.registers [1])
      val = val2 + val1
      power = int (val, 2)

      if power != 2147483647:    #### 7FFFFFFF hex
             total_power = power * 0.1  ### As defined for the controller
             tot_pow = str (total_power)   ### The information has to be
      writen as a string

      else:
             tot_pow = '----'    #### No info available

      ###Writes the info
      sheet.write(2, i+2, tot_pow, style_data)

      ######################################################################
      #### Active Energy Total

      #####################################################################
      #### Reads the value for the total active energy

      read_ener = client.read_input_registers(304,2) ## Active Energy total
      ##     304,2

      ###To manage the 2-bit, 2-register format used by the controller
      ener1 = '{0:016b}'.format(read_ener.registers [0])
      ener2 = '{0:016b}'.format(read_ener.registers [1])
      ener = ener2 + ener1

      energy = int (ener, 2)
```

```
tot_ener = str (energy)  ### The information has to be writen as a
string

###Writes the info
sheet.write(3, i+2, tot_ener, style_data)

######################################################################
#### Default Power Limit

#### Reads the value for the default power limit, according to the
current time period

read_actualpwr = client.read_input_registers(4751,1)
#### Actual Power Limit ## 4751,1

actual_power = str (read_actualpwr.registers[0] * 0.1 )  ### The
information has to be writen as a string

###Writes the info
sheet.write(4, i+2, actual_power, style_data)

######################################################################
#### Evaluation Window
#### Reads the value for the evaluation window

read_window = client.read_input_registers(4709,1)
#### Measurement time ## 4709,1

#### The info about measurement time is given as a table of possible
values
dic_eval = {0 : '5',
1 : '10',
2 : '15',
3 : '20',
4 : '30',
5 : '40',
6 : '45',
7 : '60',
8 : '90',
9 : '120',
10: '180',
11: '240'
}

eval_window = dic_eval[read_window.registers[0]]  ### The information
has to be writen in minutes

###Writes the info
sheet.write(5, i+2, eval_window, style_data)

######################################################################
#### Elapsed Time
#### Reads the value for the elapsed time

read_elapsed = client.read_input_registers(4753,1)
#### Time elapsed inside the measurement window ## 4753,1

elapsed_time = str (read_elapsed.registers[0])  ### The information
has to be writen as a string

###Writes the info
sheet.write(6, i+2, elapsed_time, style_data)

######################################################################
#### Log Registers
```

```
    #### Reads the value for all the remaining registers (they are in one
    block)

    log_registers = client.read_input_registers(4764,18)
    ####  Time  elapsed  inside  the  measurement  window  ##  4764,18;  18
    registers

    client.close()

    ######################################################################
    #### Mean Power

    mean_pow = log_registers.registers[0] * 0.1  ### As indicated in the
    Modbus map
    mean_power = str (mean_pow)  ### The information has to be writen as
    a string

    ###Writes the info
    sheet.write(7, i+2, mean_power, style_data)

    ######################################################################
    ### Energy Log Index

    energy_log  =  log_registers.registers[1]   ###  The  value  indicates
    which  index is active
    energy_log_index = str (energy_log)  ### The information has to be
    writen as a string

    ###Writes the info
    sheet.write(8, i+2, energy_log_index, style_data)

    ######################################################################
    ##### Energy LogS

    for x in range (0,16):  #### 16 log registers

          if log_registers.registers[2+x] != 65535: #### FFFF hex

                logs = log_registers.registers[2+x] * 0.1  ### The next
                register
                log_index = str (logs)  ### The information has to be
                writen as a  string

          else:

                log_index = '----'  #### No info available

          ###Writes the info
          sheet.write(9+x, i+2, log_index, style_data)

    return
```

restart():

```
def restart():

      ems = sys.executable
      ### ems = current programm being executed
      os.execl(ems, ems, * sys.argv)
```

Glossary of terms and abbreviations

AC: Alternating Current.

ADU: Application Data Unit.

DC: Direct Current.

DHCP: Dynamic Host Configuration Protocol.

EMS: Ekip Management Software.

EV: Electric vehicle.

FC: Function Code.

HV: High Voltage.

IP: Internet Protocol.

OSI: Open Systems Interconnection.

PDU: Protocol Data Unit.

PV: Photovoltaic.

SCADA: System Control And Distribution Automation.

TCP: Transmission Control Protocol.

USB: Universal Serial Bus.

VPP: Virtual Power Plant.

Bibliography

ABB SACE. “Load Management with Ekip Power Controller for SACE Emax 2,” 2012. [xiv]

Amistadi, Enrico. “An off-the-Shell Solution for Virtual Power Plants.” VENTYX, 2012. [xii]

Casazza, John. *Understanding Electric Power Systems: An Overview of the Technology and the Marketplace*. Vol. 13. Wiley. Com, 2003. [i] [xi]

Collins, Galen. “Pymodbus Documentation,” June 18, 2013. [xvi] [xvii]

“Energy Efficiency as a Key Enabler - ABB Conversations.” Accessed July 11, 2013. http://www.abb-conversations.com/2013/07/energy-efficiency-as-a-key-enabler/. [viii] [xiii]

Fox-Penner, Peter. *Smart Power: Climate Chage, the Smart Grid, and the Future of Electric Utilities*. Washington, DC: Island Press, 2010. [x]

“Geothermal | National Energy Authority of Iceland.” *NEA*. Accessed July 12, 2013. http://www.nea.is/geothermal/. [vi]

“GME - Gestore Dei Mercati Energetici SpA.” Accessed July 10, 2013. http://www.mercatoelettrico.org/EN/.

“How to Make a Solar Water Heater from Plastic Bottles.” *The Ecologist*. Accessed July 12, 2013. http://www.theecologist.org/how_to_make_a_difference/climate_change_and_energy/477574/how_to_make_a_solar_water_heater_from_plastic_bottles.html.

International Energy Agency (IEA). “Key World Energy Statistics—2012.” IEA Paris, France, 2012. [iv]

“Learning Sciences Research – Evolution Too Slow to Keep up with Climate Change, Study Finds.” *Awescience*. Accessed July 12, 2013. http://awescience.com/2013/07/10/evolution-too-slow-to-keep-up-with-climate-change-study-finds/. [iii]

Napolitano, Sergio. “A Market, Experimental and Business Analysis for the Commercialization of an Energy Demand Response Management System.” Università della Svizzera Italiana, 2012. [vii] [ix]

“Pollution Leads to Drop in Life Span in Northern China, Research Finds.” *NYTimes.com*. Accessed July 12, 2013. http://www.nytimes.com/2013/07/09/world/asia/pollution-leads-to-drop-in-life-span-in-northern-china-study-finds.html?_r=4&. [ii]

The Modbus Organization. “Modbus Application Protocol,” April 26, 2012. [xv]

“Unione Geotermica Italiana, UGI | Geotermia, Energia Geotermica.” *UGI*. Accessed July 12, 2013. http://www.unionegeotermica.it/. [v]