



**Escola Politècnica Superior
de Castelldefels**

UNIVERSITAT POLITÈCNICA DE CATALUNYA

MASTER THESIS

TITLE: Study and implementation of Polisave Client for Linux

**MASTER DEGREE: Master in Science in Telecommunication Engineering
& Management**

AUTHOR: Joan Vila Canals

DIRECTOR: Marco Mellia

DATE: Monday, 24 May 2010

Title: Study and implementation of Polisave Client for Linux
Author: Joan Vila Canals
Director: Marco Mellia
Date: Monday, 24 May 2010

Overview

This project is written with the purpose to find a solution to the high levels of energy consumption that there are nowadays. The current proliferation of network devices that are continuously powered on produces an urgent need to think about a simple and effective way to reduce their power consumption. We find out that most people prefer to leave their PCs always on; this mainly dues to the little sensibility that people have toward the cost of keeping a PC on and the cost of both, in terms of time and technical skill, to properly and quickly switch on and off a PC. These somehow surprising facts suggested us to design a solution able to control the power state of PCs in the Campus, explicitly targeting the ease of use. The result is Polisave, a centralized web-based architecture which allows users to schedule power state of their PCs; the server remotely triggers power-up and power-down events by piloting a custom software which has to be installed in each PC.

This thesis is based on the realization of Polisave for Linux. This project examines how to perform actions to turn on, suspend, hibernate or shut down computers with Linux through the programming. There is also a great survey of two complex technologies such as HAL, which allows obtaining the hardware characteristics of a computer and DBUS, which allows multiple applications to communicate. It also analyses other simple but important technologies in the Linux software such as logs, daemons, GUI...

Therefore, the result of this thesis is not only a software to perform energy saving actions, but a small study with examples of technologies which can help anyone interested in learning these technologies.

Thanks to all the people whom I met in Italy, friends, family and especially to Neus, Joan, Xavier and Cristina for their patience and help.

“Quan coneixes dues coses teòricament iguals te n’adones del munt de diferències que tenen entre si.”

“Quando sai due cose teoricamente uguali ti rendi conto la pila di differenze tra di loro”

INDEX

Introduction.....	1
Motivations and objectives	2
1 General concepts	4
1.1 Why is Polisave necessary?.....	4
1.2 Polisave definition	4
1.3 Polisave structure.....	5
1.4 Basic functionalities.....	5
2 Polisave Server	7
2.1 Server information	7
2.2 Server-Client communication	8
3 Polisave Client	11
3.1 Technologies.....	11
3.1.1 Python	11
3.1.2 Graphical Interface	12
3.1.3 HAL - <i>Hardware Abstraction Layer</i>	13
3.1.4 DBUS.....	14
3.1.5 Pm-utils.....	16
3.1.6 Daemon-Autoboot.....	17
3.1.7 LOGs	18
3.2 Client files.....	19
3.2.1 General structure	19
3.2.2 DattiPc.py	20
3.2.3 Polling.py	22
3.2.4 Popup.py	27
3.2.5 My.polisave.service and my.polisave.conf	28
4 Client Installation.....	29
4.1 First step: Requirements.	29
4.1.1 Programming details.....	29
4.2 Second step: Sign up for Polisave Server.....	30
4.2.1 Programming details.....	33
4.3 Third step: Installation on the computer.	33
4.3.1 Programming details.....	34
4.4 Uninstall	35
5 Ubuntu, Fedora and Opensuse versions.....	36
5.1 Polisave daemon.....	36

5.2	Ubuntu.....	37
5.2.1	Functions of the daemon	37
5.2.2	Autoboot	38
5.3	Fedora.....	38
5.3.1	Functions of the daemon	38
5.3.2	Autoboot	39
5.4	OpenSUSE.....	39
5.4.1	Functions of the daemon	40
5.4.2	Autoboot	40
6	Using Polisave	42
6.1	Polisave distribution	42
6.2	Polisave installation.....	42
6.3	Polisave execution	43
6.4	Examples of the client-server communication	44
6.4.1	Example 1.....	44
6.4.2	Example 2.....	44
6.4.3	Example 3.....	45
7	Problems.....	46
7.1	Standardization	46
7.2	HAL-DBUS.....	46
7.3	Session or system DBUS? The graphical interface problem.....	46
8	Conclusions.....	48
9	Bibliography	50
	APPENDIX	51
A	DattiPc.py	51
B	Polling.py	53
C	PopupThread.py	58
D	My.polisave.service	60
E	My.polisave.conf.....	60
F	Installazione.py.....	61
G	Disinstallazione.py.....	69
H	Polisave daemon.....	73

INTRODUCTION

Nowadays, how to save energy is an important factor to consider. The Politecnico di Torino has designed a software (Polisave) to reduce the energy consumption caused by the large number of computers that there are in the campus.

The main objective of this project is to develop a software for the platforms of Linux capable of performing energy-saving features. Previously, this software has already been created for Windows platforms, so the software has been developed using the physical resources of the Windows version. Due to the diversity of Linux distributions, Polisave has been programmed as standard as possible, therefore, this software is capable of running on most Linux computers available on the campus.

In order to more easily explain the development and operation of Polisave, the thesis has been divided into 9 different sections according to the information they contain.

Initially there is a brief explanation about the objectives and motivations in developing this thesis. In other words, this section explains the reasons why it is important to make this software.

The first chapter is a description of general concepts, where it is found what is Polisave, its structure and how it works.

The second chapter is a detailed study of how is the Polisave server, that is, what functions it performs and how it is organized.

The next chapter is the most important, it explains how is the client Polisave. This chapter studies the technologies used and how software was created.

In chapter four we find the explanation of how to install Polisave, in other words, the whole procedure done by the user and the computer before the user can start using Polisave.

In the fifth chapter there is a brief explanation of how Polisave has been created for various Linux distributions, and the differences between different versions of Polisave.

The sixth chapter shows how the software works correctly and the explanation is supported by different images that show all the operations.

The seventh chapter lists the major problems we have found to make this thesis.

In the last chapter we find different conclusions from this thesis.

Finally there is also an appendix, where the full software codes created can be found.

MOTIVATIONS AND OBJECTIVES

Energy consumption has become a key challenge in the last few years. According to several studies (1) (2) the Information and Communication Technology (ICT) sector alone is responsible of a percentage which varies widely from 2% to 10% of the worldwide energy consumption. The largest majority of power consumption in the ICT is today due to the billions of terminals in both households and companies. Furthermore, with the current proliferation of networked devices that are continuously power on, it is becoming urgent to think about a simple and effective way to reduce their power consumption, not only by reducing the energy consumed while they are active, but by turning them off when left unused.

Politecnico di Torino has 1800 staff members, and 28.000 students. The energy consumption is more than 1000 MWh/month in 2008. The monthly bill is always more than 150keuros/month (3).

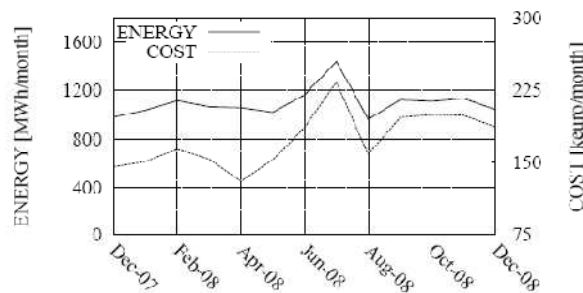


Fig.1 Power consumption 2008

Today a typical desktop PC consumes about 65 to 250 watts when active. The idle state power consumption is still in the order of 100W. Fig.2 reports the power consumed by two desktop PCs, equipped by a dual core CPU from Intel and AMD, which were running standard benchmarking software.

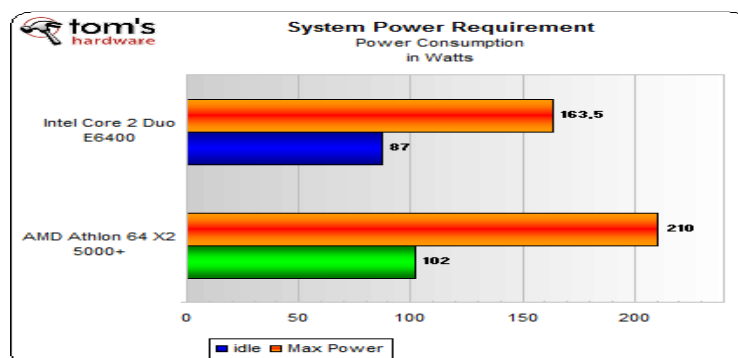


Fig.1 Benchmark of power consumption

Source: <http://www.tomshardware.com/reviews/truth-pc-power-consumption,1707-7.html>

Fig.3 reports the breakdown of the active devices, detailing different OS architectures. From the bottom, the plot reports: network, networked printers, VoIP phones and other small network boxes. All these devices are always powered on, with only printers that are seldom powered off at night. Most of Unix hosts are left up and running, possibly due to their “server” capabilities, even if a large fraction of the 350 hosts running Linux could be actually used as simple terminals. Finally, the largest fraction of devices is due to personal computers running Windows like OS.

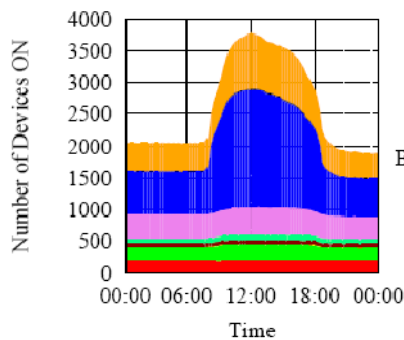


Fig.2 Variation of devices

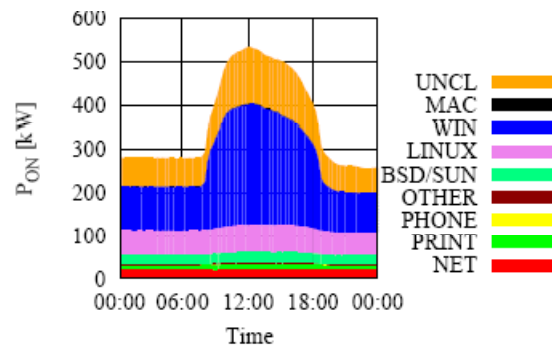


Fig.3 Estimation of power consumed

Fig 1.4 assumes that desktop computers consume 150W. It assumes that BSD/SUN systems are used as servers, for which the power footprint is higher than desktop PCs. Finally, the power consumption of network devices like routers is assumed to be 100W per interface. During the day the total energy required is more than 500 kWh, (around 26% of the total of the Campus). During the night, about 300kWh are consumed (35-40% of the total).

These results confirm the intuition that people usually leave their PCs up and running for most of the time, causing a considerable energy waste. What are the reasons that refrain users to turn off their PC. First, the economic incentive is absent in the Campus, since energy costs are not split among users. Second, the frustration of long power down and bootstrap times. Third, the loss of state a reboot causes has also been found to be annoying (if not upsetting), since users are used to leave the office with applications and documents still opened on their desktops. Fourth, some users want to access the applications and data on their office PCs even when they are at home. While technical solutions to the previous issues are already available (e.g., the “hibernate”, Wake of Lan).

These results suggested us to design a solution that controls the power state of PCs. The result is PoliSave, a centralized web architecture which allows users to schedule power state of their PCs; the server remotely triggers power-up/down events by piloting software which has to be installed in each PC.

This thesis is the creation of Polisave for Linux, because the Windows version is already created and Linux is the other OS most popular in the Campus.

1 GENERAL CONCEPTS

In this chapter, the reason why is necessary to create Polisave is pointed out. For this purpose, its description, its structure and its operation are analyzed.

1.1 Why is Polisave necessary?

Nowadays, computers are used in all kinds of works. Any operation, even the simplest one, needs the use of a computer. This means that the increase of computers is growing up every day at all levels: domestic, business or education.

To save natural resources should be a priority of all companies. If we analyze the quantity of computers that are in use nowadays and the consumption of energy that these need, the results are unsustainable. The principal problem is that many users leave the computer turned on during all day and this causes a considerable increasing of the energetic consumption. The following graph (Fig. 1.1) shows the accumulative function of computers switched on from one department of the Politecnico di Torino.

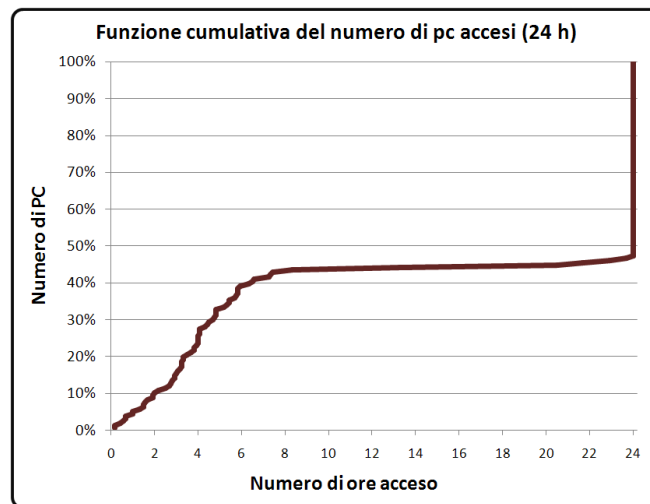


Fig. 1.1 Cumulative function of computers switched on
Source: http://www.polisave.polito.it/hosts_delen_domain.shtml

As can be seen, more or less half of the PCs are switched on all day long. For this reason, is necessary to create an application able to control when a computer should remain on, off, standby or hibernate.

1.2 Polisave definition

Polisave is software that allows the control and the programming functions of on, off, suspend, and hibernation of one or more computers. Polisave can run

these functions from a distance, in other words, you can control the status of your computer without need to be physically in front of the computer. Polisave also allows cancelling the planned energy savings if a user is using the computer at the same time.

Polisave currently is only designed to work within the Politecnico di Torino network.

1.3 Polisave structure

As mentioned in previous sections, after years working on this project, the result has been a Polisave version for Windows. Polisave for Windows is based on a client-server model. One of the requirements of the project is to use the same infrastructure used in the Windows version, therefore, the Linux version of Polisave is also based on a client server. The following graph (Fig.1.2) shows a basic example of the structure of Polisave.

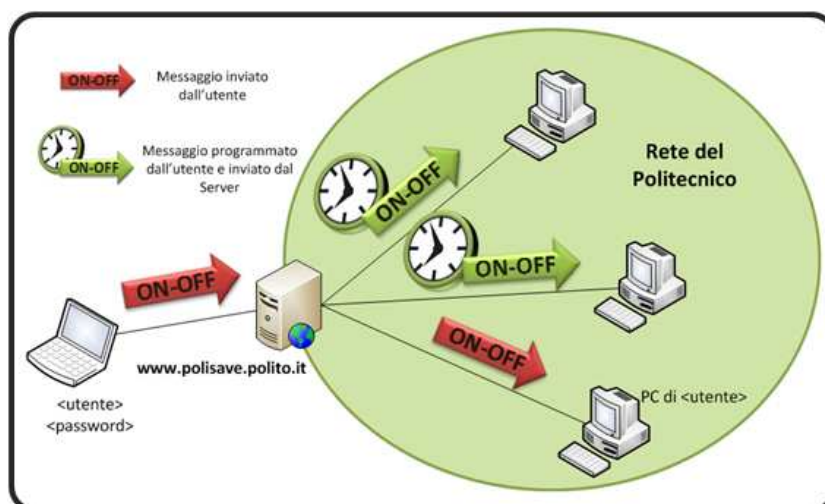


Fig. 1.2 Polisave structure.

Source: <http://www.polisave.polito.it/>

In summary, it is necessary to create an entirely new client, capable of interacting with a server (<http://www.polisave.polito.it/>) already created.

In the following chapters the different elements of the structure of Polisave are given in detail.

1.4 Basic functionalities

Before looking in detail the different parts of this project, is essential to know its basic operation.

This explanation considers a computer with Polisave already installed. When the computer is powered on, Polisave starts automatically; since it is a daemon, it is properly configured to start whenever the computer turns on.

On the server web (4), actions such as: hibernation, standby, shutdown or switch on can be programmed for saving energy. These actions can be programmed to be executables one or more times.

The client objective is to ask regularly if there is any unresolved action. However, the server aim is to answer to the client the questions indicating if there is there any programming action in the next 20 minutes. If there is an action, the server responds to the client indicating the action and how long does it take to run. If there is not any action, the server answers indicating to the client to ask it again in 20 minutes.

If the client receives a response from the server announcing that there is a hanging action in an alpha time , after this time, again the server is asked and this one responses indicating the action and time = 0. Then it is time to execute the action, but before, Polisave asks if the user agrees with the execution of that action. If he agrees, it is displayed a window with the action that is going to be executed and a countdown which begins in 60 seconds. The user can accept or cancel the action by clicking the appropriate button. If the user is not using the computer and allows these 60 seconds elapse, Polisave automatically performs the action. If the user cancels the action, the client will ask the server if there is any other pending action.



Fig. 1.3 Polisave pop-up

2 POLISAVE SERVER

First of all, should be remembered that the Polisave server was already created before starting this project, however it is absolutely necessary to make a detailed study of how it works.

Polisave server contains a database that stores information about clients using this software and information of scheduled action. Furthermore storing all this information, the server has to inform the client when an action is performed.

2.1 Server information

The server stores information related with customers such as: computer name, IP address, MAC address, user name, manager, description of user and operating system. These data are often static and are included in the Polisave client installation process, it means that there is a user registration dialog between the client and the server on which the client sends his information. This process will be detailed in Chapter 4. In addition to static information also saves the client's current state, in other words if it is off or if it is on. Since the computer is turning on, it stores the number of WAITs that have been generated since the client was on, and the time of the last WAIT. The following screenshot (Fig. 2.1) shows the data that is stored on the server about a client:

Stato	Nome	IP	MAC Address	MAC Iscrizione	Utilizzatore	Gestore
Spento	joanvila	130.192.86.27	00030d3702cc	00030d3702cc	VILA CANALS JOAN	MELLIA MARCO

Accendi	Spegni	Programmazione	Descrizione	Sistema Operativo	Ultimo Contatto	Count
			macchina per borsista	GNU/Linux	19/01/2010 17.59.19	2

Fig. 2.1 Client data stored by the server.

There is also stored energy saving actions were scheduled. These actions can be specific (only run once), daily or once at week. Possible actions that can be implemented are: on, off, hibernate and standby. Actions can be programmed via any machine by means of the Polisave server web interface: <https://www.swas.polito.it/intra/polisave/>. The following screenshot (Fig. 2.2) shows scheduled actions for a client.













Stato	Tipo	A Partire Dal	Ora Schedulata	Fino Al	Azione	Modifica	Cancella
●		17-3-2010	11.15	17-3-2010	Sospensione		
●	Tutti i giorni	16-3-2010	08.30		Accensione		
●	Tutti i giorni	16-3-2010	19.50		Spegnimento		
●	Tutti i Lunedì	22-3-2010	13.00		Standby		
●	Tutti i Martedì	16-3-2010	13.00		Standby		
●	Tutti i Giovedì	18-3-2010	13.00		Standby		
N. Righe: 6							

Fig. 2.2 Scheduled actions for a client.

Finally, the server also stores information about the messages sent to the users, the process is called LOG system. The following picture (Fig 2.3) shows this LOG:

POLISAVE - LOG

Data	IP	Host	Azione	Messaggio
16/03/2010 13.05.57	130.192.164.142	rnt	[polling]	Esecuzione Polling macchina rnt IP=130.192.164.142 next action=[WAIT] next time=20 minuti
16/03/2010 13.02.13	130.192.85.195	ismb-perfect	[polling]	Esecuzione Polling macchina ismb-perfect IP=130.192.85.195 next action=[WAIT] next time=20 minuti

Fig. 2.3 Polisave server LOG.

2.2 Server-Client communication

This section explains the process of communication between the server and the client. All communications are performed through HTTP requests and responses over TCP.

The server has always to perform the function of responding to the requests of the client. There is only one situation in which the server sends a message to the client without any response. This occurs in the case of action on, in which the server sends a Magic Packet WOL to turn on the computer.

On the other hand, there is also a communication between the client-server and the client in the installation process. This process is described in section 4.

The following explanation shows the basic communication that is taking place when a client is installed and it is working. At the beginning: the client sends a "POLLING" message to the server. Then the server looks for any scheduled

action in the next 20 minutes of this client. If there is any scheduled action, the server responds to the client announcing this action and how much time it needs to be completed. If the server does not have any pending action, it sends a message of "WAIT" with a definite time of 1200s. After this time, the client sends another "POLLING" asking again if there is any pending action and so on.

The following graph (Fig 2.4) shows POLLING dialogue between a client and a server.

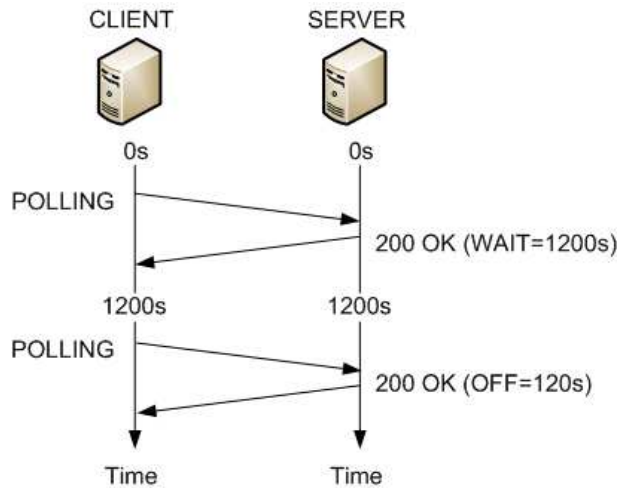


Fig. 2.4 POLLING communication

The response packets are "200 OK" type which indicate that everything is correct. The responses contain the following information in TEXT/HTML format:

ANS=[200]\r\n	→ HTTP RESPONSE (5)
IP=[XX]\r\n	→ IP Client
MAC=[XX]\r\n	→ MAC Client
ACTION=[XX]\r\n	→ Action, can be: WAIT, POWER ON, OFF, STBY and HIB
TIME=[XX]\r\n	→ Action Time

The following pictures show two examples of server response (in agreement with Fig. 2.4). Figure 2.5 A represents the "WAIT=1200s" message and Figure 2.5 B represents "OFF=120s" message.

<pre> ⊖ Hypertext Transfer Protocol ⊕ HTTP/1.1 200 OK\r\n ⊖ Line-based text data: text/html ANS=[200]\r\n IP=[130.192.86.27]\r\n MAC=[00:03:0D:37:02:CC]\r\n ACTION=[WAIT]\r\n TIME=[1200]\r\n </pre>	<pre> ⊖ Hypertext Transfer Protocol ⊕ HTTP/1.1 200 OK\r\n ⊖ Line-based text data: text/html ANS=[200]\r\n IP=[130.192.86.27]\r\n MAC=[00:03:0D:37:02:CC]\r\n ACTION=OFF\r\n TIME=[120]\r\n </pre>
(a)	(b)

Fig. 2.5 Wait and OFF response

It is important to know that in any request of the client, the server is checking if the IP is defined in its database. If the IP is not in the database, the server sends a "WAIT", but now with a "TIME" of 43200 seconds, thus the client will send a "POLLING" message within 12 hours. This situation usually occurs when Polisave users are connected outside of the Politecnico network.

```

⊖ Hypertext Transfer Protocol
  ⊕ HTTP/1.1 200 OK\r\n
⊖ Line-based text data: text/html
  ANS=[404]\r\n
  MSG=[IP non corrispondente]\r\n
  TIME=[43200]\r\n

```

Fig. 2.5 WAIT when user is out of Politecnico network.

"POLLING" message description is detailed in section 3.2.3.

3 POLISAVE CLIENT

The previous chapters described the general concepts and basic features of the server. As mentioned before, the server was created before this thesis.

Now it is the moment to concentrate the thesis in the Polisave client. The client is the main block of this thesis because it was created entirely new, so this first section examines what technological needs of the client and later its implementation.

3.1 Technologies

3.1.1 Python

All software is created using a programming language. There are infinite programming languages, each with specific characteristics. Polisave has been created using the Python language.

3.1.1.1 Why Python?

One of the requirements is that Polisave has to support different Linux distributions, so it is interesting the code standard. If using a compiled language (6) Polisave would be compiled for each type of computer. Simply, compiled languages create the resulting program on the machine from the developer, if the developer's machine is different from the user's machine, the program does not work. For example, the 32-bits Polisave version computer does not work in a 64-bits computer. Since there are many Linux distributions, a compiled language is not interesting. Python is an interpreted language (7), in other words, users install an interpreter on their computers and then the resulting program is generated when the user runs, that is, when the computer interprets. Thus, the final program is created according to the characteristics of the user's computer. Furthermore Python interpreter is usually available on most platforms.

Finally, Python is one of the most commonly used interpreted languages: fast learning, the large number of libraries available, easy to find and debug errors, dynamism and it is free technology. Python is the best solution to implement Polisave.

3.1.1.2 Programming details

On the Python homepage there is a very complete API, a tutorial and the interpreter that can be downloaded download for free.

<http://www.python.org/>

3.1.2 Graphical Interface

Polisave operation requires a graphical interface, that is, when the server sends an order to the client (Off, Hibernation, Standby ...) Polisave should show a popup to the user announcing that in 60 seconds, if the user does not oppose, Polisave will execute the order. GTK is the technology chosen to display the popup window.



Fig. 3.1 Polisave popup.

3.1.2.1 Why GTK?

GTK+ is a highly usable, feature rich toolkit for creating graphical user interfaces which boasts cross platform compatibility and an easy to use API (8). As you can see in the previous definition, GTK can generate simple GUI, is free and has a version for Python, the PyGTK. In addition, GTK has a lot of documentation and support on the network, since it is the most common technology in this field. GTK is also standard for all Linux systems.

3.1.2.2 Programming details

Developing graphical interfaces is similar in all the languages. First, all the graphic elements of the application (button, label ...) are defined. Then Polisave has to define the actions of the elements, for example, what should happen when you press a button. Finally a window must be assigned to each element in order to be showed.

For our purposes, we use GTK+ with the PyGTK wrapper (9). To use the features and elements of PyGTK and GTK, the libraries have to be imported: `import pygtk` and `import gtk`. PyGTK version must be at least 2.0. Later it is possible to see a small example:

```
import pygtk
```

```
pygtk.require('2.0')
import gtk

window=gtk.Window(gtk.WINDOW_TOPLEVEL)
buttonEnd = gtk.Button("Button 1")
buttonEnd.connect("clicked",function_after_click)
buttonEnd.show()
window.add(buttonEnd)
```

3.1.3 HAL - *Hardware Abstraction Layer*

Sometimes Polisave should consult the hardware of the machine, for example to find the MAC address of a network interface, or even to check if the WoL is enabled. There is a technology capable of classifying information of the hardware (low level) to other layers (high level) to perform queries and modifications to the hardware. This technology is known as HAL.

3.1.3.1 *What is HAL?*

According to (10) HAL is an abstraction layer, implemented in software, between the physical hardware of a computer and the software that runs on that computer. Its function is to hide differences in hardware from most of the operating system kernel, so that most of the kernel-mode code does not need to be changed to run on systems with different hardware. On a PC, HAL can basically be considered to be the driver for the motherboard and allows instructions from higher level computer languages to communicate with lower level components, such as directly with hardware.

3.1.3.2 *Why HAL?*

Today, all products support the HAL technology, these are defined so that HAL can get their data and classifies them. Thus, you can know all the features of hardware that a computer has, whether it's Linux, Windows or Mac. HAL's election is for the simple reason that today there is no other technology capable of performing these functions as standard for all systems and free.

3.1.3.3 *Programming details*

HAL simply orders and classifies the characteristics of the hardware, therefore, there is no direct interaction between Python and HAL. Polisave needs a connection between Python and HAL able to collect data from HAL to Python. This connection is created with DBUS. To sum up, HAL technology is not in programming terms.

3.1.4 DBUS

Polisave needs to connect data obtained by HAL and the program in Python. This connection is provided DBUS, a technology of *freedesktop*.

3.1.4.1 What is DBUS?

According to (11) D-Bus is a message bus system, a simple way for applications to talk to one another. In addition to interprocess communication, D-Bus helps coordinate process lifecycle; it makes it simple and reliable to code a "single instance" application or daemon, and to launch applications and daemons on demand when their services are needed.

D-Bus supplies both a system daemon (for events such as "new hardware device added" or "printer queue changed") and a per-user-login-session daemon (for general IPC needs among user applications). Also, the message bus is built on top of a general one-to-one message passing framework, which can be used by any two apps to communicate directly (without going through the message bus daemon).

3.1.4.2 Why DBUS?

DBUS connects two applications, through the system or the session. Apart from connecting Python and data HAL, Polisave also needs to connect the graphics application to the program in Python. Therefore, DBUS is the most appropriate technology because it is the most developed in this field. Moreover, DBUS is free and is standard for all the Linux distributions.

3.1.4.3 How does DBUS work?

Working with DBUS is not simple. There are several ways to work with DBUS.

Polisave can work depending on the bus type: session bus and system bus. It is very important to differentiate when to work through a session DBUS or system DBUS. If data are collected by applications run by the user, it must use DBUS session. Otherwise, if data from the system (HAL) or applications are run by the operating system (or other users), it uses DBUS system.

It also can be differentiated according to the form of exchanging data:

- Requests for static data: the receiver receives a request and responds with the existing static data. For example, when Polisave wants to obtain data from the hardware (via HAL), these data are static and HAL only responds to requests. It is a request-response system. It is typically used when an application does not depend on the programmer.

- Sending dynamic data: an application sends a message to the other without waiting for an answer. For example, the exchange of messages between the popup and the application in Python. The application is continuously waiting for a message from the Popup. When the popup sends a message, the application analyzes it and does not answer to the Popup. It is a messages exchange system. It is typically used when two applications depend on the programmer.

DBUS is based on objects and interfaces, request or data are sent as objects. To read an object, it is necessary knowing what kind of object is. For this, it is necessary knowing which interface can read data that contains the object. If the interface is not appropriate, data cannot be read correctly.

3.1.4.4 Programming details

Polisave uses DBUS to collect data from HAL and to send actions between the Polisave brain and the graphical interface. In both cases the information is sent through a **system** DBUS. The following is an example of how to get data from HAL, in particular to discover if the computer is capable of hibernation:

```
A1: bus = dbus.SystemBus()
A2: device = bus.get_object("org.freedesktop.Hal",
    "/org/freedesktop/Hal/devices/computer")
A3: prod = device.GetPropertyString(
    'power_management.can_hibernate',
    dbus_interface='org.freedesktop.Hal.Device')
```

A1 creates the bus to receive the information and indicates bus type (system). A2 asks an object of "org.freedesktop.Hal" with the following name: "/org/freedesktop/Hal/devices/computer". At this point, "device" has a computer object with several properties inside. A3 wants to analyze the properties of the computer object, it specifically requires the value (as string) of the variable 'power_management.can_hibernate' and shows what kind of interface is needed to consult the variable, in this case should be looked up through the interface 'org.freedesktop.Hal.Device'. The result indicates if this computer is able to hibernate or not.

The following example shows the exchanged data between applications. In particular the data from Polisave Popup to the Polisave brain are detailed:

```
MY_BUS_NAME = 'my.polisave'
MY_OBJECT_PATH = '/my/polisave'
```

PopUp:

```
B1: bus = dbus.SystemBus(mainloop=DBusGMainLoop())
B2: dbus.service.Object.__init__(self, bus, MY_OBJECT_PATH)
B3: @dbus.service.signal(dbus_interface=MY_BUS_NAME, signature='s')
```

```

B4: def Off(self,numAction):
B5:     ...

B6: self.Off("3")

```

Brain:

```

C1: dbus.mainloop.glib.DBusGMainLoop(set_as_default=True)
C2: bus = dbus.SystemBus()

C3: bus.add_signal_receiver(self.ExitAction,
    dbus_interface=MY_BUS_NAME, signal_name="Off")

C4: mainloop = GObject.MainLoop()
C5: mainloop = mainloop.run()

C6: #@dbus.service.method(MY_BUS_NAME,in_signature='s',
    out_signature='as')
C7: def ExitAction(self, numAction):
C8:     ...

```

B1, C1 and C2 define the type of DBUS, DBUS System here. Popup contains:

- B2 creates an object with path MY_OBJECT_PATH.
- B3 sends a MY_BUS_NAME object when someone runs "Off" (B4).
- B6 called an "Off" and thus sends the object. The object is formed by the "Off" function variables (B4), in this case, numAction.

The Brain contains:

- C3 defines a receiver of MY_BUS_NAME objects, this receiver only listens to the objects that have been sent through Off function. In addition, C3 indicates when an object is received, the program must execute the method ExitAction (C7).
- C6 defines that ExitAction (C7) is a DBUS method.
- Finally, the brain must always be listening for messages arrive, for this reason there is a listening loop (C4 and C5).

3.1.5 Pm-utils

Polisave need some kind of technology that allows suspending and hibernating. A few years ago, finding a standard technology to do this was very difficult, because each system and each distribution were based on different management systems. It is important to know that suspend and hibernation are actions that store all running processes in memory and execute a partial (suspend) or total (hibernation) shutdown of the computer, specifically:

- **Suspend:** Processes are stored in RAM because this memory is not interrupted. (12)

- **Hibernation:** It is a complete shutdown of the computer. In this case, the running processes are stored in the swap system. For this reason, it is often advisable to have a swap larger than RAM, because otherwise the computer cannot store all the RAM information in the swap and therefore hibernation is not successful. (13)

There are many ways to perform these functions, including hardware, software, or even kernel modification. In our case we adopt Pm-utils to perform these tasks. Pm-utils is a framework to set suspend and power state mode (14) (15)

We chose this framework since it is the simplest in application terms and it is the standard in all systems. Pm-utils contains a package of applications that can execute suspend and hibernation actions. These applications use HAL and are transparent to the user. So we can say that Pm-utils works at software level but using Hardware terms. Currently Pm-utils is already included in most Linux systems.

3.1.5.1 Programming details

In programming terms, Polisave has only to run the appropriate script when it wants to execute the action. To run the scripts Polisave uses the *commands* library of Python, this allows to execute Shell commands, specifically *getstatusoutput()* method. Therefore to call the scripts pm-suspend and pm-hibernate from python, the following code must be run:

```
import commands
commands.getstatusoutput('pm-suspend')
commands.getstatusoutput('pm-hibernate')
```

3.1.6 Daemon-Autoboot

A daemon is not a technology but it is an essential element to Polisave.

3.1.6.1 What is a Daemon?

In Unix and other computer multitasking operating systems, a daemon is a computer program that runs in the background, rather than under the direct control of a user; they are usually initiated as background processes.

In a Unix environment, the parent process of a daemon is often (but not always) the init process (PID=1). Processes usually become daemons by forking a child process and then having their parent process immediately exit, thus causing init to adopt the child process. This is a somewhat simplified view of the process as other operations are generally performed, such as disassociating the daemon process from any controlling tty.

Systems often start daemons at boot time: they often serve the function of responding to network requests, hardware activity, or other programs by performing some task. Daemons can also configure hardware (like devfsd on some GNU/Linux systems), run scheduled tasks (like cron), and perform a variety of other tasks. (16)

3.1.6.2 Why a Daemon?

Polisave must run in background and has to be properly initialized when the computer starts therefore, Polisave must be controlled by a daemon.

3.1.6.3 Programming details

As the definition says, when a program is initialized by a daemon, it must create a child process with a fork and kill the father process. The base directory must also be changed as the root computer directory. The following code can be viewed as Polisave creates a child process at the beginning (A1), killing the father process (this process always has a pid=0) (A2 and A3), changes directory (A4) and implements the other functions required for good management of the process (A5 and A6). Finally Polisave starts (A7).

```
        try:
A1:         pid_daemon = os.fork()
A2:         if pid_daemon > 0 :
A3:             sys.exit(0)
        except OSError, e:
            sys.exit(1)
A4:     os.chdir("/")
A5:     os.setsid()
A6:     os.umask(0)

A7:     Inizio()
```

It is important to know that this code is not the daemon, this is the first thing that runs when Polisave starts, which has been initialized by the daemon. The daemon internal structure is different in every Linux distribution. The daemon structure is detailed in Chapter 5.

3.1.7 LOGs

3.1.7.1 What is LOGs?

It is recommended that the program monitors all actions carried out: the important steps of the process, errors or warnings. This is done with LOG's files. Every time that an important action happens, a line is written in the log file annotating the description and the time it happened. Thus, if someone wants to

check if an error has occurred or what actions have been done, the LOGs file shows the information.

3.1.7.1 Programming details

Polisave only used LOGs in the brain of the program, in other words, the Polling.py file. Next code configures the LOGs system (Polling.py).

```
import logging
SELF = 'Polisave'
A1:  if os.getenv('MWC_DEBUG') != None:
        DEBUG = True
    if DEBUG:
A2:      DEBUG_LEVEL = logging.DEBUG #print all messages
    else:
A3:      DEBUG_LEVEL = logging.WARNING #print warnings
A4:  FORMATTER = '%(asctime)s %(levelname)s %(message)s'
A5:  logging.basicConfig(level=DEBUG_LEVEL,format= FORMATTER,
        filename = '/var/log/' + SELF + '.log',filemode = 'a')
```

To use LOGs it is necessary to import the logging library. Polisave LOGs defines two levels: the first one writes the error messages (A3) while the second one writes all the messages (A2). If the computer is in debug mode (A1), it writes all the messages, if it is not in debug mode, it writes only error messages. Polisave defines a message format, for example the date inclusion in the message (A4). Then it executes the LOG configuration (A5), indicating the kind of messages, the format, where the LOG file is stored (in this case /var/log/polisave.log) and if it wants to write all messages or only last message. Once configured, when Polisave wants to write a log message simply does:

```
A6:  logging.debug("Polisave started")
A7:  logging.critical('Error POLLING: invalid destination host')
A8:  logging.shutdown()
```

A6 writes a message only if debug mode is activated, but not in Warning mode, however, A7 message is always written. Finally, if an error has occurred, the LOG file will be closed (A8).

3.2 Client files

Previous section described the most important technologies that have been used and programming details. This section explains how these technologies have been implemented on the client.

3.2.1 General structure

The following diagram shows an overview of Polisave. Files are represented in squares and the most important actions are represented in circles.

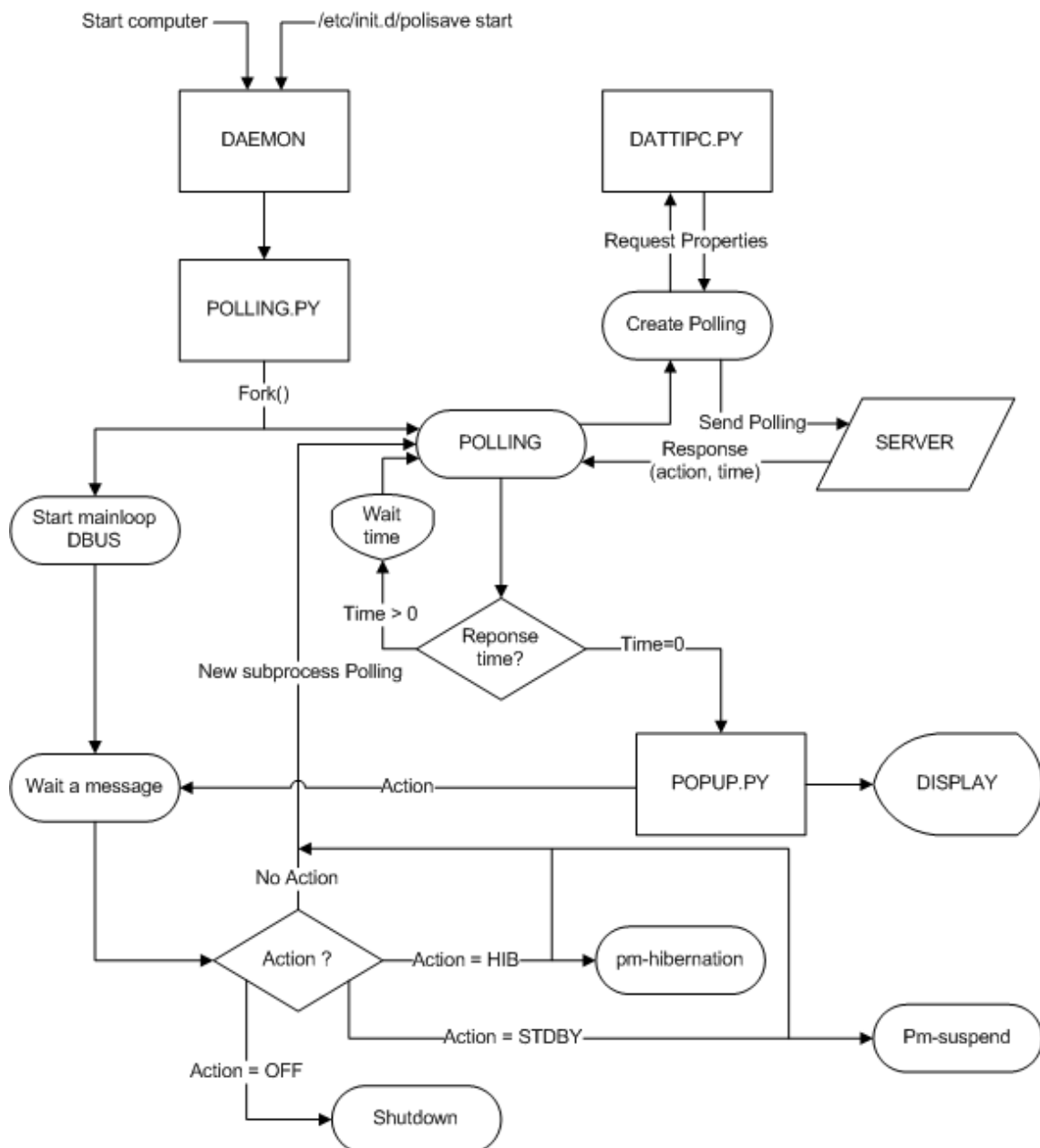


Fig. 3.2 General diagram of the Polisave client.

Understanding the diagram at first sight is a bit complicated, therefore, the main functions that implements each file are explained.

3.2.2 DattiPc.py

3.2.2.1 General function

The general function of this file is to obtain data from the computer. Data obtained are: hibernation and WoL capabilities, operating system, hostname and IP and MAC addresses.

The operation is basic: the client asks for a data and DattiPc.py returns the requested data. DattiPc.py is a library, that is, if you execute the file does nothing, since the file simply has methods.

Data are obtained by querying HAL (via DBUS) or through Shell commands. Data obtained from DattiPc.py are used to create POLLING messages and messages of the installation process.

3.2.2.2 Programming details

DattiPc.py has 6 methods:

- **Get_hib():** obtains if the computer is capable of hibernation. This is obtained using a HAL query to the object "/org/freedesktop/Hal/devices/computer" with interface "org.freedesktop.Hal.Device" and property "power_management.can_hibernate".
- **Get_so():** gets the machine's operating system using the Shell command "uname -o". To implement commands of Shell through Python is used *getstatusoutput()* method of the *commands* library.

```
commands.getstatusoutput('uname -o')[1]
```

- **Get_hostname():** gets the hostname using the Shell command "uname -n" a shell. Commands library is used.
- **Get_ip(interface):** gets IP address of the interface (input parameter) through the socket library.

```
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
ip=socket.inet_ntoa(fcntl.ioctl(s.fileno(),0x8915,struct.pack('256s', ifname[:15]))[20:24])
```

- **Get_tutti_ip_mac():** get all the IP and MAC address of the machine through HAL query.

The procedure is as it follows: A1 and A2 create a HAL manager. Manager filters all "net" devices and gets "device" objects (A3 and A4). After objects are interpreted by the appropriate interface and it obtains MAC address (A5) and network card name (A6) of each device. To obtain the IP address through the network card name, Polisave uses **Get_ip(interface)** (A7).

```
bus = dbus.SystemBus()
A1: objeto_hal = bus.get_object(hal_dot, hal_path+'/Manager')
A2: halmanager = dbus.Interface(objeto_hal, hal_dot+'.Manager')
```

```

A3: for dev_name in halmanager.FindDeviceByCapability("net"):
A4:     device = bus.get_object("org.freedesktop.Hal",dev_name)

A5:     mac = device.GetPropertyString('net.address',
        dbus_interface='org.freedesktop.Hal.Device')

A6:     net = device.GetPropertyString('net.interface',
        dbus_interface='org.freedesktop.Hal.Device')

        tutti_mac= tutti_mac + ',' + mac
A7:     tutti_ip= tutti_ip + ',' + get_ip(str(net))

```

- **Get_WOL(mac):** obtains, through HAL query, if the computer supports WOL capability.

First Polisave should get the "device" object which has the MAC, that is passed by parameter. This procedure is similar to that used in **Get_tutti_ip_mac()**, because it must analyze all the devices and select the device which has the correct MAC. Once the "device" object is correct, Polisave should check whether it has the "wake_on_lan" property.

```

prod = device.GetPropertyString('info.capabilities',
        dbus_interface='org.freedesktop.Hal.Device')

for property in prod:
    if property == 'wake_on_lan':
        return 'OK'

```

The complete listing of DattiPc.py file is detailed in the Annex.

3.2.3 Polling.py

3.2.3.1 General function

Polling.py is the Polisave brain, the client. Polling.py is controlled through the polisave.sh (daemon), it is what starts and stops Polling.py. The main function of Polling.py is to send POLLING messages to the server and it acts according to server response. Next, its operation is detailed; it can be seen graphically in Fig. 3.4.

The first step that Polling.py should do when it starts is to create a child process (using a fork()) and kill the father process. This is essential to do because it comes from a daemon.

Already in the child, it should create another thread, so now there are two threads: a "polling" thread and "DBUS" thread. It is essential to do this because otherwise it is not possible to have both parts running simultaneously.

Polling thread

The polling process has the function of sending POLLING messages to the server. The library DattiPC.py is used to create these messages. Then you can see the structure of a package POLLING:

```
GET /services/polisave/polisave.asp?ACTION=[POLLING]&NAME=[JoanVi]a]&COUNT_POLLING=[1]
&IP=[0.0.0.0,130.192.86.27]&MAC=[00:15:00:28:A6:F2,00:03:0D:37:02:CC]
Host: www.swas.polito.it\r\n
User-Agent: HTTP Fetcher/HTTP/1.0\r\n
Connection: close\r\n
```

Fig. 3.3 Polling packet

The most important fields are:

- **Action:** In this case always equate to POLLING.
- **Name:** Indicates the computer name.
- **Count_polling:** indicates the number of times it has sent a POLLING message and the server has responded WAIT. When you start Polisave or when the server response with an action, the count_polling restarts at 0.
- **IP:** indicates all IP addresses on the computer.
- **MAC:** indicates all MAC addresses on the computer.

When Polisave sends the POLLING message, it expects a response from the server. The different responses are described in Chapter 2, they are like this:

```
⊖ Hypertext Transfer Protocol
  ⊕ HTTP/1.1 200 OK\r\n
⊖ Line-based text data: text/html
  ANS=[200]\r\n
  IP=[130.192.86.27]\r\n
  MAC=[00:03:0D:37:02:CC]\r\n
  ACTION=OFF\r\n
  TIME=[120]\r\n
```

Fig. 3.4 Server response

If the "ACTION" field is equivalent to "WAIT", then "TIME" field is 1200s or 43000s. In contrast, if the "ACTION" is equivalent to another option, the "TIME" is a countdown time to run that action.

The first thing the client does when it receives the server response is to analyze the "TIME" field, then:

- If "TIME" is more than 60 seconds, the client waits this time and then it sends another POLLING message to the server, increasing "Count_Polling" field.
- If "TIME" is less than 60 seconds, the client initializes a Popup for the action specified in "ACTION". When this happens, polling thread ends, since it has no more tasks to run.

DBUS thread

DBUS process creates a system bus and waits Popup messages. There are 3 groups of messages that the server can receive:

- **NoAction:** This message occurs when the user rejects the action of the popup, then the client creates a new polling thread. (Previous polling thread was over).
- **Shutdown:** in this case the client shutdowns the computer.
- **Hibernation or Standby:** in this case, it creates a new polling thread because when computer is turned on again, Polisave is active. In addition to creating the polling thread, it runs the action (hibernation or standby).

3.2.3.2 Programming details

Polling.py file consists of 3 classes: Inizio, DBUSmanagement and Polisave and a method LOGs.

- **Inizio:** This class is responsible for creating the child process (A1) and creates Polling (A3) and DBUS (A4) threads.

```

try:
    shutil.copy('/tmp/polisave.txt',
                '/usr/local/bin/polisave.txt')
except OSError:
    logging.debug('Could not copy tmp file')
try:
A1:     pid_daemon = os.fork()
        if pid_daemon > 0 :
            sys.exit(0)
except OSError, e:
    logging.critical('Fork error')
    logging.shutdown()
    sys.exit(1)
A2:   os.chdir("/")
    os.setsid()
    os.umask(0)

pid = os.fork()

```

```

        if pid != 0:
A3:         Polisave()
        else:
            pid = str(os.getpid())
A4:         file("/var/run/polisave.pid", 'w+').write("%s\n"%pid)
A5:         DBUSmanagement()

```

Necessary post-daemon functions are also implemented, for example: changing root (A2), the pidfile (A5), etc.

- **DBUSmanagement:** This is the class which runs as DBUS thread. This class is basically composed by the methods needed to analyze the messages sent by the Popup (B1), so there are implemented actions here: off (B2), hibernation (B4) and standby (B6). Remember that when running an hibernation or standby action another polling process is opened (B3 and B5).

```

B1:     #@dbus.service.method(MY_BUS_NAME, in_signature='s',
        out_signature='as')
        def ExitAction(self, numAction):
            if numAction=='1':
                logging.debug('Azione wait')
            elif numAction=='2':
                logging.debug('Azione off')
B2:         commands.getstatusoutput('shutdown -h now')
            elif numAction=='3':
                logging.debug('Azione hib')
B3:         subprocess.Popen("/usr/local/bin/Polling.py
        rePolling", shell=True)
B4:         commands.getstatusoutput('pm-hibernate')
            elif numAction=='4':
                logging.debug('Azione stby')
B5:         subprocess.Popen("/usr/local/bin/Polling.py
        rePolling", shell=True)
B6:         commands.getstatusoutput('pm-suspend')

        return ["Ok"]

```

- **Polisave:** This is the class which runs as polling thread. There are three important methods:
 - **Polling():** This method performs the following actions: sending a Polling (C1), analyzing the response (C2) and opening a popup (C4).

```

        self.continuare=True
        while (self.continuare == True):
C1:         response = self.SEND_POLLING()
C2:         azione,msgAzione=self.RECEIVE_POLLING(response)
            self.countPolling+=1
            logging.debug('Open popup')
C3:         f=open("/usr/local/bin/polisave.txt", "r")
            XauthDir=f.readline()

```

```

        if os.path.exists(XauthDir)!=True:
            logging.critical('Error: could not detect X)
            logging.shutdown()
            exit().
        os.environ['XAUTHORITY']=XauthDir
        os.environ['DISPLAY']=":0.0"
C4:    child = subprocess.Popen( "/usr/local/bin/popup.py" +
        ' ' + str(azione) + ' ' + msgAzione, shell=True)

```

As you can see, before opening a popup it should verify that the user has correctly configured X11 Window Manager, if so, it looks what directory is the variable XAUTHORITY in (C3).

- **Send_Polling():** This method creates and sends the message POLLING to the server via HTTP. It simply uses the methods of the library DattiPc.py.
- **Recive_Polling():** This method analyzes the responses from the server. As mentioned above, it first sees if there are actions pending in the next 60 seconds (D1 and D2), if so, it returns the action to be performed. In case that TIME is more than one minute, the program sleeps during this time minus 30 seconds (D4) and returns 1 indicating that it must resent a POLLING message.

```

        if (R1.status!=200):
            exit()
        responseRead=R1.read()

D1:    time_wait=int((responseRead.split()[4]).strip(
        'TIME=[]'))

D2:    if time_wait < 60:
D3:        action=responseRead.split()[3]
            numAction = self.ACTION(action)
            self.continuare=False

            if numAction == 6:
                return numAction, responseRead.split()[6]
            elif 0<numAction<6:
                return numAction, 'tutto a posto'
            else:
                exit()
        else:
D4:        time.sleep(time_wait-30)
            return 1,'tutto a posto'

```

- **LOGs:** LOG file management. See chapter 3.1.7.

To view fully Polling.py file go to the Annex.

3.2.4 Popup.py

3.2.4.1 General function

The main function of this file is to inform the user that Polisave wants to perform an action. The user can accept or cancel the action. The following image shows the popup, it contains a countdown, if the user has not responded to the window after this time the action is executed.



Fig. 3.5 Popup

Another function of the popup is to communicate through DBUS, which is the user's response.

3.2.4.2 Programming details

The calling of the popup is done using a command like the following:

```
Subprocess.Popen("/usr/local/bin/popup.py Action Message, shell=True")
```

This command creates a new thread for the popup. The implementation has two input parameters: "Action" and "Message". "Action" indicates the action that it wants to run and "Message" is only used when the server wants to send a message to the user.

Below, the most important groups of methods Popup.py file are described:

- **Creation of window:** these are methods that contain all the properties and instructions to create the graphics window (buttons, dimensions ...).
- **Preparation of the message to display:** This method creates a message to display to the user depending on the type of action to execute.
- **Timer:** These methods are calculating and updating the counter value that is displayed to the user.

- **Action by user:** These methods show the actions executed by the program when the user presses the button.

The complete listing of Popup.py is reported in the Annex.

3.2.5 My.polisave.service and my.polisave.conf

3.2.5.1 General function

DBUS Polisave service is announced in advance in a particular directory in order to be known by the applications. Then my.polisave.service file is created. It contains the description of the service. Below there is the entire code, which shows the service name, the running file and the user who can run it:

```
[D-BUS Service]
Name=my.polisave
Exec=/usr/local/bin/Polling.py
User=root
```

Polisave also needs a service configuration file (my.polisave.conf). It contains interfaces, which users have permission to use, etc. The permissions and interfaces that root user have are detailed as follows:

```
<policy user="root">
  <allow own="my.polisave"/>
  <allow send_interface="my.polisave.On"/>
  <allow send_interface="my.polisave.Off"/>
</policy>
```

4 CLIENT INSTALLATION

This chapter explains in detail the process of installing Polisave on a Linux computer. The installation consists mainly of three steps: the first checks that the computer is ready to install Polisave; in the second step the client has to authenticate with the server; at last, in the third phase Polisave is installed in the client machine. It is important to mention that if the results of a phase are not satisfactory, the installation stops and does not run the next step.



Fig. 4.1 Client installation steps

4.1 First step: Requirements.

The first phase is the simplest one: Polisave has to verify that the machine where the client wants to install Polisave meets a set of requirements. Mainly, Polisave has to check three characteristics:

- **Python and all its accessories.** The computer must have Python installed, also all the libraries of the language that requires the installation (httplib, webbrowser, shutil, os, commnads, sys, platform, subprocess).
- **SuperUser.** Usually the installation of a program is not executed by a normal user but by the system administrator, who must run the installation. At this point, it checks that the user who executed the installation is the system administrator.
- **Power Management.** The Main function of Polisave is to control the power functions of the computer. Therefore, the installation must verify that the computer can execute these functions.

If the computer where the client wants to install Polisave meets the three characteristics exposed above, the computer is now ready to sign up for server.

4.1.1 Programming details

In this section are shown, in general, the requirements described above in terms of programming:

- **Python and all its accessories.** This feature is independent of programming. So, if Python is not installed on the computer it is not possible to run the installation program. If Python detects that some libraries are missing, it stops the installation.
- **SuperUser.** Linux classifies each user according to its level of privileges. The level assigned to a normal user depends on each Linux distribution, but the administrator level is equal to all distributions, this is level 0. To check this feature Polisave uses the Python library *os*, concretely its function *os.geteuid()*. This function returns the level of the user who executed it. Therefore, if the result of *os.geteuid()* is 0, it means that the user who executed the installation is the administrator. (17)
- **Power Management.** Specifically Polisave checks that is possible to run functions of hibernation and suspend. The option to turn off the computer is already implemented in all Linux using the command "*shutdown*". As mentioned in section 3, the programs chosen to execute these functions are *pm-hibernate* and *pm-suspend* (15). Polisave verifies if these programs are installed on the computer, checking that *pm-hibernate* and *pm-suspend* are in the computer's *Path*. Polisave uses the Python library *os*, concretely its function *os.path.exists()*. This library indicates whether the program is in the list of executable programs on the computer (17).

The complete listings of the Requirements step can be found in the Annex.

4.2 Second step: Sign up for Polisave Server.

Before a user can use Polisave this user must register its machine to the server. At this step the user sends data to the server and if it sees something that does not coincide with the stored data on the server database, the server answers with an error message. The following image shows a block diagram of the processes of communication between the client and server:

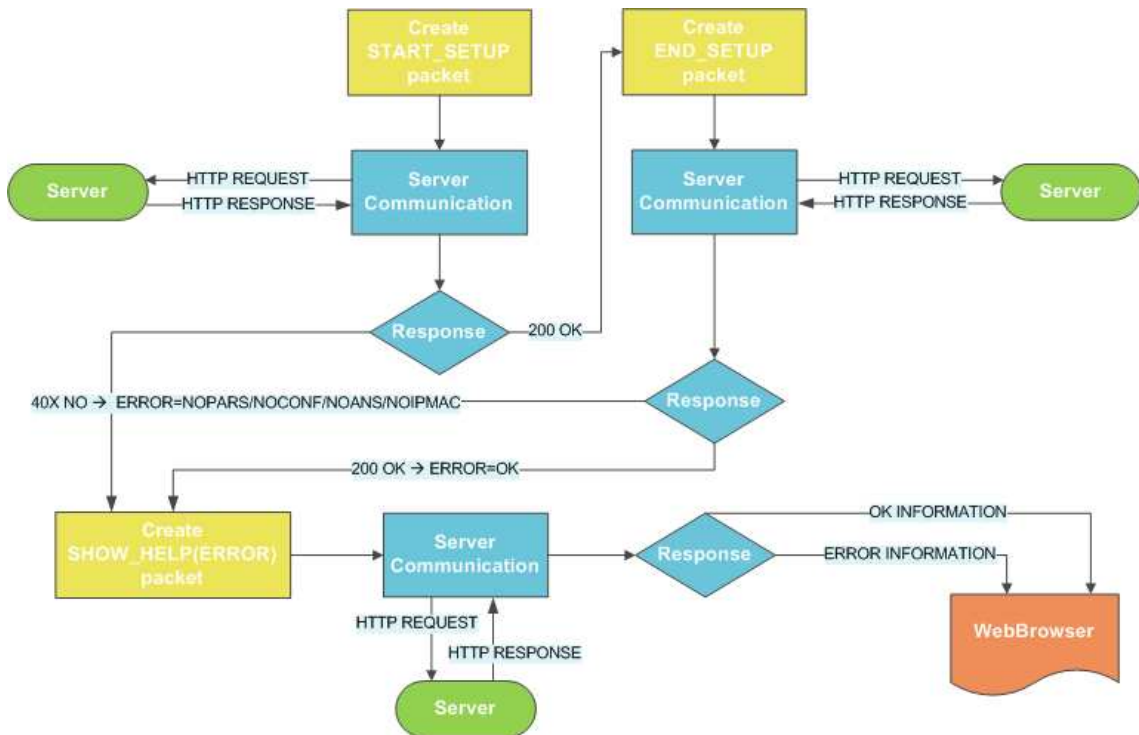


Fig. 4.2 Sign up diagram

The following graphic shows the three situations that can arise from the previous diagram. The graph Fig.4.3(a) shows a communication without any error. Graph Fig.4.3(b) shows an error in the final process. Finally, in Fig.4.3(c) there is an error in the initial point.

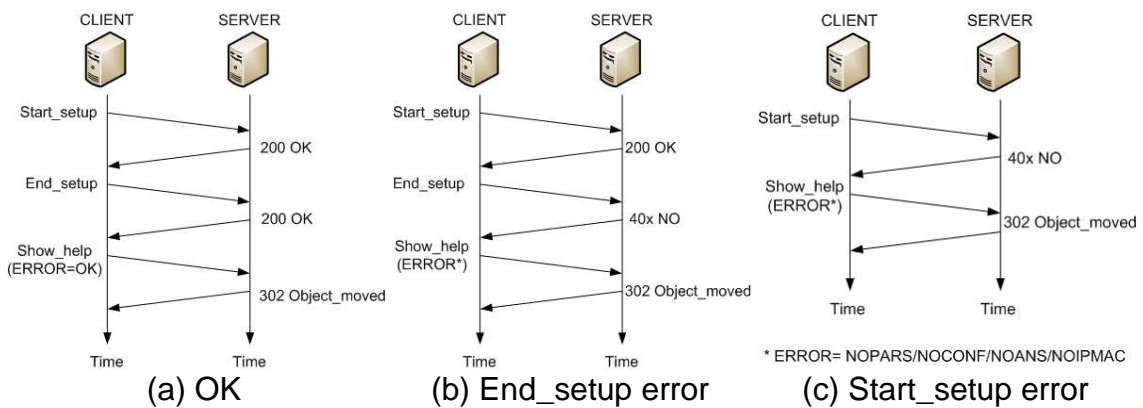


Fig. 4.3 Sign up communication

Below is a detailed description of packets sent:

- **START_SETUP:** HTTP Request (GET) with the following variables:
 - ACTION=[START_SETUP]
 - NAME=[Computer_name] → Computer name.
 - OS=[Operating_system] → Operating system.

- HIB=[OK/NO] → If the computer is able to hibernate.
- IP=[ip1, ip2,...] → All IPs of the computer.
- MAC=[mac1, mac2,...] → All MACs of the computer.
- **200 OK:** HTTP Response (body) of the *START_SETUP*:
 - ANS=[200/40x/500] → If the answer is a error (40x or 500), it sends a *SHOW_HELP* message and it aborts the installation.
 - IP=[selected_IP] → IP selected by the server.
 - MAC=[selected_MAC] → MAC selected by the server.
 - MSG=[optional_message] → Informational message server.

The server selects an IP and a MAC in accordance with the IP that has in its database. In other words, IP of the Politecnico network.

- **END_SETUP:** HTTP Request (GET) with the following variables:
 - ACTION=[END_SETUP]
 - WOL=[OK/NO] → If the computer is able to 'Wake Of Lan'
 - NAME=[Computer_name] → Computer name.
 - IP=[selected_IP] → IP selected by the server.
 - MAC=[selected_MAC] → MAC selected by the server.
- **200 OK:** HTTP Response (body) of the *END_SETUP*:
 - ANS=[200/40x/500] → If the answer is a error (40x or 500), it sends a *SHOW_HELP* message and it aborts the installation.
 - IP=[selected_IP] → IP selected by the server.
 - MAC=[selected_MAC] → MAC selected by the server.
 - ACTION=[action] → Indicates if there are any pending actions (*POWER OFF, HIB* o *STBY*) or not (*WAIT*) in the next 20 minutes.
 - TIME=[action_time] → Time needed for the action announced.
- **SHOW_HELP:** HTTP Request (GET) with the following variables:
 - ACTION=[SHOW_HELP]
 - ERROR=[OK/NOANS/NOPARS/NOIPMAC/NOCONF] → Indicates whether an error has occurred.
 - WOL=[OK/NO] → If the computer is able to 'Wake Of Lan'
 - OS=[Operating_system] → Operating system.
 - HF=[OK/NO] → If the computer is able to hibernate.
 - IP=[selected_IP] → IP selected by the server.
 - MAC=[selected_MAC] → selected by the server.
 - S4=[OK/NO] → If the computer is able to hibernate.
- **302 Object Moved:** HTTP Response (body) of the *END_SETUP*:
 - The answer is a dynamic website of Polisave that shows whether an error occurred previously or not.

During this exchange of messages may occur the following errors:

- **NOANS:** the answer from the server has not been received.
- **NOPARS:** the parsing of the answer of the server has failed.

- **NOIPMAC:** the answer of the server did not include the MAC and IP of the selected interface (required only for the first message of the setup program).
- **NOCONF:** the client was not able to activate the 'Wake Of Lan' or the Hibernation.

When a failure occurs, a *SHOW_HELP* message is sent back. In this case the *ERROR* variable contains the error code (*NOANS*, *NOPARS*, *NOIPMAC* or *NOCONF*). Otherwise, if registration is successful, the *ERROR* variable is set to *OK*.

4.2.1 Programming details

In this section is shown, in general, the process described before in terms of programming:

- Polisave use `httplib.HTTPConnection()` function of the `httplib` library to create a connection to exchange messages with the HTTP server. (18)
- To obtain the values of the variables of the requests, Polisave uses `DattiPc.py` described in chapter 3.2.2. As mentioned before, this file has been created expressly for Polisave. `DattiPc.py` provides all data about the machine that it is running Polisave, such as: IP and MAC address, if it can make WoL or Hibernation, computer name, operating system, etc.
- To display the final result of registration in a webbrowser, Polisave uses the library *webbrowser*. (19)

The complete listings are detailed in the Annex.

4.3 Third step: Installation on the computer.

Finally, once the machine is been registered on the server, it only needs to install the program on the computer. To install the program it has only to copy the necessary files to its destination. This location varies according to Linux distribution. It is important to know that all the files are done in standard form and therefore, files are useful for all the distributions Linux. The only two differences are the location where to store the files and the management daemon. Therefore, this section explains, in general, where to copy files each Linux distribution.

Then each file is related to its destination. The files have been described in chapter 3.2:

- ***DattiPc.py*:** placed in the directory of the Python libraries so that other files Python can be used.

- **Polling.py:** placed in a directory Bin, so that the computer is able to locate it and run it. The chosen directory is `/usr/local/bin/`.
- **Popupthread.py:** placed in a directory Bin, so that the computer is able to locate it and run it. The chosen directory is `/usr/local/bin/`.
- **my.polisave.service:** placed in a directory `/usr/share/dbus-1/services/` so that the DBUS is able to locate this service.
- **my.polisave.conf:** placed in a directory `/etc/dbus-1/system.d/` so that the DBUS is able to locate this configuration.
- **Polisave.sh:** The Linux daemons are located in the directory `/etc/init.d/`.
- **Autoboot:** For the daemon is initialized when the computer boots, Polisave creates a link to start directory of daemons. This directory varies according to the runlevel (computer boot mode) and the Linux distribution.
- **cpXauth:** to initialize a Daemon when session starts, Daemon has been placed in the directory `etc/X11/xinit/xinitrc.d/`.

4.3.1 Programming details

In this section we find, in general, the features described above in terms of programming:

- **Location directory of the python libraries:** Usually, the path is defined by `/usr/libXX/pythonYY`, where XX is 64 if a computer is 64 bits and YY is the version of Python, for example `/usr/lib/python2.6`. Therefore, the first step is to locate the version of Python. This is done by using the method `python_version()` of the platform library, being able to determine the version of Python this way. Finally, Polisave consults which directory has Python in its path and sees if some directory coincides with the default one (`/usr/libXX/pythonYY`). To see the directories that Python has in its path, Polisave uses the method `sys.path[]` from the library `sys`. When all these operations are done, Polisave knows which one is the directory where it must be stored `DattiPc.py`. (20) (21)
- **Location of other directories:** The other directories are common to all distributions. So, Polisave can manually put the paths.
- **Copy files:** The `shutil` library is used for copying files and in particular its method `shutil.copy()`. (22)
- **Create links:** The `os` library is used for linking files and in particular its method `os.symlink()`. (17)

4.4 Uninstall

Polisave has a script to uninstall Polisave. The script (`disinstallazione.py`) clears the files that are copied into the system during the installation. Before deleting, Polisave should check if the service is stopped. Here is a part of the code of this script:

```
A1: process = subprocess.Popen('/etc/init.d/polisave.sh stop',
    shell=True, stderr=subprocess.PIPE)
A2: process.wait()

A3: version_py = '/usr/lib/python'+platform.python_version()
    lib_path=sys.path[1]
    for i in sys.path:
        if version_py.startswith(i):
            lib_path=i

    try:
A4:         os.remove(lib_path+ '/DattiPc.py')
A5:         os.remove('/usr/local/bin/Polling.py')
A6:         os.remove('/usr/local/bin/popup.py')
A7:         os.remove('/etc/init.d/polisave.sh')
A8:         os.remove('/usr/share/dbus-1/services/my.polisave.service')
A9:         os.remove('/etc/dbus-1/system.d/my.polisave.conf')
A10:        os.remove('/var/log/Polisave.log')
A11:        os.remove(lib_path + '/DattiPc.pyc')
    except OSError:
        print "Error: Disinstallazione incorretta"
```

A1 stops Polisave service so it can be uninstalled successfully. Then, the service is expected to be fully stopped (A2). As in the installation, it finds out which is the Python path where the library `DattiPc.py` is located (A3) and, once Polisave has detected the path, it deletes the file (A4). The `os` library is used for removing files and in particular its method `os.remove()`. It also removes the files: `Polling.py` (A5), `popup.py` (A6), the script `polisave.sh` (A7), `my.polisave.service` (A8), `my.polisave.conf` (A9) and `Polisave.log` (A10). The file `DattiPc.pyc` (A11) is created the first time that Polisave runs.

5 UBUNTU, FEDORA AND OPENSUSE VERSIONS

Polisave has been created for 3 distributions of Linux: Ubuntu, Fedora and Opensuse:

- **Ubuntu:** Ubuntu is a computer operating system based on the Debian GNU/Linux distribution. New versions of Ubuntu are released every six months and supports Ubuntu for eighteen months. The latest version of Ubuntu, 9.10 (Karmic Koala), was released on October 29, 2009. (23)
- **Fedora:** Fedora is an RPM-based, general purpose operating system built on top of the Linux kernel, developed by the community-supported Fedora Project and sponsored by Red Hat. With 6 months between releases, the maintenance period is about 13 months for each version. The latest version, Fedora 12, was released on November 17, 2009. (24)
- **OpenSUSE:** OpenSUSE is a general purpose operating system built on top of the Linux kernel, developed by the community-supported openSUSE Project and sponsored by Novell. With 6 months between releases, the current stable release is openSUSE 11.2 and it was released on November 12, 2009. (25)

5.1 Polisave daemon

In previous chapters we have commented that all files are common for all platforms but the daemon, which differs from the others depending on Linux distribution. A daemon should have 3 main functions:

- **Start:** this function starts Polisave. First of all Polisave must verify that the Daemon is not already started.
- **Stop:** this function stops Polisave but before that it must verify that the Daemon is started.
- **Restart:** Combining the previous two features, in the first place it executes a stop and then a start.

To execute these actions manually, the user can simply launch the following commands:

```
/etc/init.d/polisave.sh start/stop/restart
```

To create the daemon as standard as possible, it is been decided to create a daemon with Shell language, in other words, scripts ".sh"

A Daemon creates child processes, ending processes, etc. This means that when Polisave wants to stop the service, it does not know exactly what process to stop, because Polisave is not the same process that has executed the start.

To perform these operations properly Polisave creates a pidfile. A pidfile is a file that contains the number of pid of the process that is actually running and so it is the process that it wants to finish later. This pidfile is created in Polling.py file and Polisave stores it in the directory /var/run/polisave.pid (directory where the majority of pidfiles are located).

```
pid = os.fork()
if pid != 0:
    Polisave()
else:
    pid = str(os.getpid())
    file("/var/run/polisave.pid", 'w+').write("%s\n" % pid)
    DBUSmanagement()
```

This Pidfile is created just as Polisave creates the DBUS thread, because the polling thread finishes and starts depending on the needs of Polisave. When Polisave executes a stop, it deletes the file "/var/run/polisave.pid". Then, there is a start action, where it checks whether the file exists or not, and if it exists it means that the daemon is started.

Every Linux platform has its own methods to perform the functions Start, Stop and Restart. The upcoming sections will explain these functions.

5.2 Ubuntu

This section is divided into two parts: Functions of the daemon and the autoboot when computer starts.

5.2.1 Functions of the daemon

The start and stop functions are managed with the script *start-stop-daemon*. This script is responsible for managing all the 'start/stop' of the process and is transparent to the user. Here is some of the code of the daemon:

```
A1: POLISAVE_BIN="/usr/local/bin/Polling.py"
A2: ARG="start"
A3: PIDFILE='/var/run/polisave.pid'
    case "$1" in
A4:         start)
A5:             if [ -f $PIDFILE ]; then
                    echo -n "Daemon gia aperto"
                else
A6:                 start-stop-daemon --start --pidfile $PIDFILE --
                    exec $POLISAVE_BIN $ARG
                fi
                ;;
A7:         stop)
A8:             start-stop-daemon --stop --pidfile $PIDFILE
A9:             rm -f $PIDFILE
```

```

                ;;
A10:            restart)
                $0 stop
                $0 start
                ;;
            *)
                echo "Usage: $0 {start|stop|restart}"
                exit 1
                ;;

```

First of all, the daemon defines the variables. These variables indicate the path of the file polling.py (A1), the arguments of start (A2) and location of the pidfile (A3).

When the user or the computer wants to implement the action 'start' (A4), the program must check for an existing pidfile (A5). If the pidfile exists, it means that the process is open and the daemon cannot start. On the contrary, if the pidfile does not exist, Polisave is able to start. The action that is used to start Polisave is shown in line A6 and it indicates to the script start-stop-daemon that Polisave starts (-start), the pidfile directory (A3), the file to execute (A1) and its arguments (A2).

When user wants to stop the process using the method 'stop' (A7), the client first stops the daemon using the script start-stop-daemon (A8) indicating that it wants to kill (-stop). The command also indicates that the pidfile (A3) for the script checks if the process is already started or not. Then it deletes the pidfile (A9).

The method 'restart' (A10) simply calls the methods stop (A7) and start (A4).

5.2.2 Autoboot

To create an autoboot in Ubuntu, Polisave must make a link to the daemon in the autoboot directory. This directory in Ubuntu is in /etc/rcX.d/ where X is the runlevel in which the computer starts (usually 2 or 4). This link is created when Polisave is installed and is called S99Polisave. The character S means to make a start when the computer starts and 99 is the boot order of daemons. Polisave is not a daemon priority for the system, so is the last level of boot (99).

5.3 Fedora

This section is divided into two parts: Functions of the daemon and the autoboot when computer starts.

5.3.1 Functions of the daemon

The start and stop functions are managed with the scripts *daemon* and *killproc*. This script is responsible for managing all the 'start/stop' of the process and is transparent to the user. Here is some of the code of the daemon:

```
A1:  MYDAEMON="/usr/bin/python /usr/local/bin/Polling.py start"
      RETVAL=0

A2:  start()
      {
A3:      if [ ! -f /var/lock/subsys/polisave ]; then
A4:          daemon $MYDAEMON && success || failure
              RETVAL=$?
A5:          [ "$RETVAL" = 0 ] && touch /var/lock/subsys/polisave
      else
              echo "Polisave gia iniziato"
      fi
      }

A6:  stop()
      {
A7:      killproc $MYDAEMON -TERM
              RETVAL=$?
A8:      [ "$RETVAL" = 0 ] && rm -f /var/lock/subsys/polisave
      }
```

The difference with Ubuntu is that here the daemon uses a particular file for the pidfile. The file is `/var/lock/subsys/polisave`.

The script defines the variable "MYDAEMON" indicating the file path `Polling.py` (A1). The `start()` method checks if the pidfile exists (A3) and if so, Polisave starts (A4). The daemon script is simple; it executes the action A1 and shows whether the command succeeded or if it has occurred an error. The A5 is the path of the pidfile when A4 was successful.

The `stop()` method uses the script `killproc` (A7). If the stop is correctly, then it deletes the pidfile (A8).

5.3.2 Autoboot

To create an autoboot in Fedora, Polisave must make a link to the daemon in the autoboot directory. This directory in Fedora is in `/etc/rcX.d/` where X is the runlevel in which the computer starts (usually 2 or 4). This link is created when Polisave is installed and is called `S99Polisave`. The character S means to make a start when the computer starts and 99 is the boot order of daemons. Polisave is not a daemon priority for the system, so is the last level of boot (99).

5.4 OpenSUSE

This section is divided into two parts: Functions of the daemon and the autoboot when computer starts.

5.4.1 Functions of the daemon

The start and stop functions are managed with the scripts *startproc* and *killproc*. This script is responsible for managing all the 'start/stop' of the process and is transparent to the user. Here is some of the code of the daemon:

```
A1: POLISAVE_BIN="/usr/local/bin/Polling.py"
A2: ARG="start"
A3: PIDFILE='/var/run/polisave.pid'

    case "$1" in
A4:         start)
A5:             if [ -f $PIDFILE ]; then
                    echo "Daemon gia aperto"
                else
A6:                     /sbin/startproc -f $POLISAVE_BIN $ARG
                    rc_status -v
                fi
                ;;

A7:         stop)
A8:             /sbin/killproc -L -p $PIDFILE $POLISAVE_BIN
A9:             rm -f $PIDFILE
                rc_status -v
                ;;
    esac
```

First of all, daemon defines the variables. These variables indicate the path of the file *polling.py* (A1), the arguments of start (A2) and location of the pidfile (A3).

When the user or the computer wants to implement the action 'start' (A4) must check for an existing pidfile (A5). If the pidfile exists, it means that the process is open and the daemon cannot start. On the contrary, if the pidfile does not exist, Polisave is able to start. The action that is used to start Polisave is shown in line A6 and it indicates to the script */sbin/startproc* that Polisave starts (-start), the pidfile directory (A3), the file to execute (A1) and its arguments (A2).

When user wants to stop the process using the method 'stop' (A7), first stops the daemon using the script */sbin/killproc* (A8) indicating the path of Polisave. The command also indicates that the pidfile (A3) for the script checks if the process is already started or not. Then it deletes the pidfile (A9).

5.4.2 Autoboot

In openSUSE to generate an autoboot Polisave must run the following action when user installs Polisave:

```
commands.getstatusoutput('insserv /etc/init.d/polisave.sh')
```

This shell command inserts Polisave service in the list of scripts that start automatically when user starts the computer. Therefore, when user uninstalls Polisave, Polisave have to remove the service from the list and this is done with the following command:

```
commands.getstatusoutput('insserv -r /etc/init.d/polisave.sh')
```

Polisave must create a link to the daemon in the autoboot directory. This directory in openSUSE is in */sbin/rcpolisave/*. As in the installation, this is done through the command `symlink` of the `os` library.

```
os.symlink('/etc/init.d/polisave.sh', '/sbin/rcpolisave')
```

6 USING POLISAVE

6.1 Polisave distribution

Polisave has been created for different Linux distributions. When Polisave has been created for Fedora and openSUSE, it has been done in virtual machines. For this reason, once finished Polisave has to be tested on other machines. Polisave for Linux has been sent to several people who helped with the project testing the new program. This testing process has helped to change some details and it can do a better Polisave. In the next picture you can see some users use Polisave.

POLISAVE - LOG				
Data	IP	Host	Azione	Messaggio
26/04/2010 20.43.25	130.192.85.195	ismb-perfect	[polling]	Esecuzione Polling macchina ismb-perfect IP=130.192.85.195 next action=[WAIT] next time=20 minuti
26/04/2010 20.40.49	130.192.9.146	lyapunov	[polling]	Esecuzione Polling macchina lyapunov IP=130.192.9.146 next action=[WAIT] next time=20 minuti
26/04/2010 20.40.17	130.192.85.204	ismb-defina	[polling]	Esecuzione Polling macchina ismb-defina IP=130.192.85.204 next action=[WAIT] next time=20 minuti
26/04/2010 20.36.54	130.192.86.233	ismb-falco	[polling]	Esecuzione Polling macchina ismb-falco IP=130.192.86.233 next action=[WAIT] next time=20 minuti
26/04/2010 20.36.23	130.192.164.142	rnt	[polling]	Esecuzione Polling macchina rnt IP=130.192.164.142 next action=[WAIT] next time=20 minuti
26/04/2010 20.35.21	130.192.9.189	fauniera	[polling]	Esecuzione Polling macchina fauniera IP=130.192.9.189 next action=[WAIT] next time=20 minuti
26/04/2010 20.34.44	80.35.6.169	joanvica	[polling]	Richiesta pervenuta da IP esterno al Politecnico
26/04/2010 20.30.36	130.192.15.118	theark	[polling]	Esecuzione Polling macchina theark IP=130.192.15.118 next action=[WAIT] next time=20 minuti
26/04/2010 20.28.45	130.192.9.197	polgum	[polling]	Esecuzione Polling macchina polgum IP=130.192.9.197 next action=[WAIT] next time=20 minuti

Fig. 6.1 Several users use Polisave

6.2 Polisave installation

The following image shows how the installation program copied the files in the correct directories.

```
joanvica@joanvica:/$ ls /usr/lib/python2.6/DattiPc.py
/usr/lib/python2.6/DattiPc.py
joanvica@joanvica:/$ ls /usr/local/bin/Polling.py
/usr/local/bin/Polling.py
joanvica@joanvica:/$ ls /usr/local/bin/popupThread.py
/usr/local/bin/popupThread.py
joanvica@joanvica:/$ ls /usr/share/dbus-1/services/my.polisave.service
/usr/share/dbus-1/services/my.polisave.service
joanvica@joanvica:/$ ls /etc/dbus-1/system.d/my.polisave.conf
/etc/dbus-1/system.d/my.polisave.conf
joanvica@joanvica:/$ ls /etc/init.d/polisave.sh
/etc/init.d/polisave.sh
joanvica@joanvica:/$ ls /etc/rc2.d/S99polisave
/etc/rc2.d/S99polisave
```

Fig. 6.2 Polisave files

6.3 Polisave execution

The following image shows the implementation of the daemon functions:

```
root@joanvica:/# /etc/init.d/polisave.sh start
Starting Polisave service
root@joanvica:/# /etc/init.d/polisave.sh start
Daemon gia aperto
root@joanvica:/# /etc/init.d/polisave.sh stop
Stopping Polisave service
root@joanvica:/# /etc/init.d/polisave.sh stop
Stopping Polisave service
No process in pidfile `/var/run/polisave.pid' found running; none killed.
root@joanvica:/# /etc/init.d/polisave.sh start
Starting Polisave service
root@joanvica:/# /etc/init.d/polisave.sh restart
Stopping Polisave service
Starting Polisave_service
```

Fig. 6.3 Daemon functions

With Polisave running, we see that the pidfile exists:

```
root@joanvica:/# cat var/run/polisave.pid
6257
```

Fig. 6.4 Polisave pidfile

The following image shows how the log file works correctly.


```
joanvica@joanvica:~$ cat /var/log/Polisave.log
2010-03-01 13:26:52,259 DEBUG Polisave started.
2010-03-01 13:26:53,337 DEBUG Inizio DBUS
2010-03-01 13:26:53,349 DEBUG Polling
2010-03-01 13:27:54,458 DEBUG Send Polling
2010-03-01 13:27:54,631 DEBUG Wait 1200s
2010-03-01 13:47:25,440 DEBUG Send Polling
2010-03-01 13:47:26,612 DEBUG Wait 120s
2010-03-01 13:48:56,702 DEBUG Send Polling
2010-03-01 13:48:57,003 DEBUG numAction:2
2010-03-01 13:48:57,003 DEBUG Open popup
2010-03-01 13:49:13,872 DEBUG Continuare
2010-03-01 13:49:14,477 DEBUG Continuare
2010-03-01 13:49:15,029 DEBUG Polling
2010-03-01 13:49:15,030 DEBUG Polling
2010-03-01 13:50:15,259 DEBUG Send Polling
2010-03-01 13:50:15,260 DEBUG Send Polling
2010-03-01 13:50:15,370 DEBUG Wait 1200s
```

Fig. 6.5 LOG file

6.4 Examples of the client-server communication

In this section, we can see several examples of clients using Polisave.

6.4.1 Example 1

The first example shows the messages sent when there is no scheduled action.

```
78 188. GET /services/polisave/polisave.asp?ACTION=[POLLING]&NAME=[Joanv11a]&COUNT_POLLING=[1]&IP=[
81 188. HTTP/1.1 200 OK (text/html)
63 1358 GET /services/polisave/polisave.asp?ACTION=[POLLING]&NAME=[Joanv11a]&COUNT_POLLING=[2]&IP=[
65 1358 HTTP/1.1 200 OK (text/html)
72 2529 GET /services/polisave/polisave.asp?ACTION=[POLLING]&NAME=[Joanv11a]&COUNT_POLLING=[3]&IP=[
75 2529 HTTP/1.1 200 OK (text/html)
81 3699 GET /services/polisave/polisave.asp?ACTION=[POLLING]&NAME=[Joanv11a]&COUNT_POLLING=[4]&IP=[
84 3699 HTTP/1.1 200 OK (text/html)
90 4869 GET /services/polisave/polisave.asp?ACTION=[POLLING]&NAME=[Joanv11a]&COUNT_POLLING=[5]&IP=[
94 4869 HTTP/1.1 200 OK (text/html)
00 6039 GET /services/polisave/polisave.asp?ACTION=[POLLING]&NAME=[Joanv11a]&COUNT_POLLING=[6]&IP=[
```

Fig. 6.6 Capture Wireshark

We can see how Polisave sends Polling messages every 1170 seconds (20 minutes minus 30 seconds) when there is no action scheduled. Polling counter increases. All messages "200 OK" have the ACTION=[WAIT] and TIME=[1200].

6.4.2 Example 2

The second example shows the exchange of the messages when there is a scheduled action.

```
49 2.5 GET /services/polisave/polisave.asp?ACTION=[POLLING]&NAME=[Joanv11a]&COUNT_POLLING=[1]
52 2.6 HTTP/1.1 200 OK (text/html)
58 92. GET /services/polisave/polisave.asp?ACTION=[POLLING]&NAME=[Joanv11a]&COUNT_POLLING=[2]
61 92. HTTP/1.1 200 OK (text/html)
67 122 GET /services/polisave/polisave.asp?ACTION=[POLLING]&NAME=[Joanv11a]&COUNT_POLLING=[3]
70 122 HTTP/1.1 200 OK (text/html)
```

Fig. 6.7 Capture Wireshark

The response messages are the following:

ANS=[200]\r\n	ANS=[200]\r\n	ANS=[200]\r\n
IP=[130.192.86.27]\r\n	IP=[130.192.86.27]\r\n	IP=[130.192.86.27]\r\n
MAC=[00:03:0D:37:02:CC]\r\n	MAC=[00:03:0D:37:02:CC]\r\n	MAC=[00:03:0D:37:02:CC]\r\n
ACTION=HIB\r\n	ACTION=HIB\r\n	ACTION=HIB\r\n
TIME=[120]\r\n	TIME=[60]\r\n	TIME=[0]\r\n
Packet 52	Packet 61	Packet 70

Fig. 6.8 Capture Wireshark

Here we see the operation explained in Chapter 3. When there is an action, the response indicates the time remaining to execute the action. The client program looks at whether this time (120 seconds) is less than 60 seconds. The client waits 120 seconds minus 30 seconds. After 90 seconds Polisave sends another Polling message. The following response (packet 61) indicates that TIME=60 seconds, as it is not minor than 60seconds, the client waits for 60-30=30seconds. When these 30 seconds are passed, client asks again and the server sends a response (packet 70) with a TIME=0 and then client opens the popup.

6.4.3 Example 3

The third example shows the exchange of the messages when there is a scheduled action and the user rejects the action via Popup.

```

49 2.548GET /services/polisave/polisave.asp?ACTION=[POLLING]&NAME=[Joanvila]&COUNT_POLLING=[1]&IP=
52 2.615HTTP/1.1 200 OK (text/html)
58 92.70GET /services/polisave/polisave.asp?ACTION=[POLLING]&NAME=[Joanvila]&COUNT_POLLING=[2]&IP=
61 92.79HTTP/1.1 200 OK (text/html)
67 122.8GET /services/polisave/polisave.asp?ACTION=[POLLING]&NAME=[Joanvila]&COUNT_POLLING=[3]&IP=
70 122.9HTTP/1.1 200 OK (text/html)
78 188.6GET /services/polisave/polisave.asp?ACTION=[POLLING]&NAME=[Joanvila]&COUNT_POLLING=[1]&IP=
81 188.7HTTP/1.1 200 OK (text/html)
63 1358.GET /services/polisave/polisave.asp?ACTION=[POLLING]&NAME=[Joanvila]&COUNT_POLLING=[2]&IP=
65 1358.HTTP/1.1 200 OK (text/html)
72 2529.GET /services/polisave/polisave.asp?ACTION=[POLLING]&NAME=[Joanvila]&COUNT_POLLING=[3]&IP=
75 2529.HTTP/1.1 200 OK (text/html)
81 3699.GET /services/polisave/polisave.asp?ACTION=[POLLING]&NAME=[Joanvila]&COUNT_POLLING=[4]&IP=

```

Fig. 6.9 Capture Wireshark

The package 70 is a response with ACTION=HIB and TIME=0. When the user rejects the Popup, the client waits 60 seconds and it sends another Polling message to the server. Here, the important thing is to see how the field count_polling is reset to 0 when Polisave opens a Popup.

7 PROBLEMS

While we have been developing this project, we have encountered many problems, some simple, because of lack of use of some technologies and other more complicated problems. This chapter mentions the three most difficult problems of this project.

7.1 Standardization

Polisave has to be completely standard for all Linux distributions. Polisave was originally created for Ubuntu, this version could not use perfect implementations for Ubuntu but unusable for Fedora and OpenSUSE. So when Polisave has been tested in other platforms, we have found that the platforms are very different. We have then modified Polisave according to the capabilities of each platform, since a possible solution to Fedora is not the same that for OpenSUSE. Finding a compatible implementation with all platforms has sometimes been a problem; it was not difficult but laborious and costly in terms of time.

7.2 HAL-DBUS

In general, DBUS has been one of the most difficult concepts to implement. It works with objects and interfaces, it is difficult to understand and moreover there are not many tutorials, manuals nor examples on the Internet. If DBUS is to communicate two new applications complicity is not as high, because the programmer always knows the interfaces and objects that he has to use, for example the communication between Polling.py and Popup.py. However, when you have to work with an independent application, as the case of HAL, the implementation is complicated. HAL organizes items but there is not a tree explaining the name of the interfaces, or objects anywhere. Because of that, it had to be tested interface by interface. For example, if you want to find if a particular interface is capable of execute WOL or not, it can be complicated.

7.3 Session or system DBUS? The graphical interface problem

According to the explanation of the creators we must use a Session Dbus if we want to communicate two applications and System Dbus when the application is communicating with an element of the system. At first Popup and Polling should communicate through a Session Dbus because they are two applications. The Session Dbus, as its name suggests, can only be used by users in the same Session. This, in principle, should not be a problem, Polisave is designed for only one user simultaneously using the computer. The problem comes when we want to initialize Polisave automatically after the system boot, to do this we create a daemon and we indicate the system to initialize the daemon automatically. The system initializes the daemon but not in the user's session

(like the popup) but in the “boot” session. As a result, the popup cannot find the bus daemon because they are in different sessions.

There are two possible solutions to this problem: implementing everything in the same user’s session or System DBUS:

- The implementation of the first option leads Polisave not to start until the user enters their username and password. It does not matter because if a computer is not logging, it also has to make energy saving actions. Moreover Polisave does not start as system daemon, but as the session daemon, a concept that is not simple in all Linux distributions.
- The second option results in a very difficult but solvable problem. Therefore, the implanted solution relies on this option. If the communication is done via a system Dbus, the Polling.py and the Popup.py and can communicate for sure, since there is only one system. The problem comes when Polisave tries to show the graphical interface, it has been run as root and therefore communication has nothing to do with the Session, then the Popup does not found the graphical user interface to represent the popup. The solution to this problem is to detect active graphical user interface, this is done by the DISPLAY environment variable where its value is typically "0.0". Once we have detected the DISPLAY, Polisave needs to have the authorization to represent windows; this is done with a session key. In most systems this key is stored in a directory which is indicated in the environment variable Xauthority, usually this key is in the Home of the user. So what is the problem? If we run everything as root, we cannot get the environment variables of users, because root has its own values of environment variables. To solve this we have implemented a solution a bit complicated but conceptually simple. A script starts when the user opens the session, this script copies the environment variables (DISPLAY and Xauthority) to a temporary file. Then, when it wants to display a popup to the user, it must obtain the environment variables. Polisave gets the variables from the stored file when the session starts and Polisave maps Display and Xauthority variables to the root user variables. Doing all this procedure, Polisave starts automatically by a daemon and it displays popup to the user.

Once solved, the problem seems simple, but the deduction of why popups were not open, after trying many solutions and finally think about an entirely new solution, has turned this into a problem for several months, but finally with a happy ending.

8 CONCLUSIONS

The final result of this thesis has been Polisave for Linux. During the carrying out of this project we have obtained the following conclusions.

It is a fact that nowadays there is too much unnecessary energy consumption. As an illustration, on a campus where there are more than 4000 devices, for sure it leads to a very high economic cost. That is why the carrying out of a program like Polisave is absolutely necessary and its implementation should be done as soon as possible. With Polisave we can reduce from 16 hours to 10 hours the power on of a computer every day. This results in saving 250,000 Euros per year just by reducing six hours daily on all computers.

We have seen too, how Python is now a very powerful language. In the case of online support information and libraries, Python, which is an interpreted language, it has been very helpfully at the moment to generate Polisave versions for different Linux distributions.

DBUS is a technology of great use, but its learning is complicated. It has allowed us to link multiple applications, whether at the hardware level and user level. DBUS is a powerful technology. In the near future applications will be generated by simply attaching processes of various applications using DBUS.

It is incredible to see how all the hardware devices and operating systems use HAL to describe their characteristics. This is a technology allowed to standard the information of all devices in order to be treated to other hardware devices or software. It is still a standard but it needs a better organization of its elements to make clearer his ease of use.

To carry out energy saving actions, in particular suspend and hibernate actions, Pm-utils is the component that has been used. Internally, this component uses HAL functions. Pm-utils is the most simple, effective and standard of all the technologies tested in this field.

We have studied the Linux world, seeing that the three most commonly used distributions are Ubuntu, Fedora and OpenSUSE. Each distribution has its peculiarities. Specially, we noted that Fedora and Ubuntu use the same structure for the daemons autoboot. OpenSUSE has a similar structure but we run more actions.

During this project we learned how to organize the GUI in Linux. Initially it seemed a simple issue but it has generated many problems. The GUI can cause problems if you do not choose the appropriate parameters, such as the display that it is currently active or the appropriate environment variables (different for each type of session, user, or Linux distribution.)

Our aim for this thesis is not only to give a tool for the implementation of a software, but to give a reference aid to all people who wants to learn the technologies explained here. We also hope that in a future Polisave achieve will

be able to reduce energy and economic costs of different organizations, not only of the Politecnico di Torino.

9 BIBLIOGRAPHY

1. <http://www.globalactionplan.org.uk/>. Global Action Plan Report, An inefficient truth,. [Online] 2007.
2. <http://www.theclimategroup.org>. SMART 2020 Report, Enabling the low carbon economy in the information age. [Online] 2008.
3. http://www.telematica.polito.it/chiaraviglio/papers/polisave_techreport.pdf. [Online]
4. <http://www.polisave.polito.it/>. [Online]
5. http://en.wikipedia.org/wiki/List_of_HTTP_status_codes. [Online]
6. http://en.wikipedia.org/wiki/Compiled_language. [Online]
7. http://en.wikipedia.org/wiki/Interpreted_language. [Online]
8. <http://www.gtk.org/>. [Online]
9. <http://www.pygtk.org/>. [Online]
10. http://en.wikipedia.org/wiki/Hardware_abstraction_layer. [Online]
11. <http://www.freedesktop.org/wiki/Software/dbus>. [Online]
12. http://en.wikipedia.org/wiki/Sleep_mode. [Online]
13. [http://en.wikipedia.org/wiki/Hibernation_\(computing\)](http://en.wikipedia.org/wiki/Hibernation_(computing)). [Online]
14. <http://wiki.archlinux.org/index.php/Pm-utils>. [Online]
15. <http://pm-utils.freedesktop.org/wiki/>. [Online]
16. [http://en.wikipedia.org/wiki/Daemon_\(computer_software\)](http://en.wikipedia.org/wiki/Daemon_(computer_software)). [Online]
17. <http://docs.python.org/library/os.html>. [Online]
18. <http://docs.python.org/library/httplib.html>. [Online]
19. <http://docs.python.org/library/webbrowser.html>. [Online]
20. <http://docs.python.org/library/platform.html>. [Online]
21. <http://docs.python.org/library/sys.html>. [Online]
22. <http://docs.python.org/library/shutil.html>. [Online]
23. <http://www.ubuntu.com/>. [Online]
24. <http://fedoraproject.org/>. [Online]
25. <http://www.opensuse.org>. [Online]
26. <http://www.python.org/doc/essays/blurb/>. [Online]

APPENDIX

A DattiPc.py

```
#!/usr/bin/python
# codice simple per sapere le proprieta del computer
import dbus
import socket
import fcntl
import struct
import commands

hal_dot = 'org.freedesktop.Hal'
hal_path = '/org/freedesktop/Hal'

def get_hib():
    bus = dbus.SystemBus()
    device =
bus.get_object("org.freedesktop.Hal", "/org/freedesktop/Hal/devices/com
puter")
    prod =
device.GetPropertyString('power_management.can_hibernate',
dbus_interface='org.freedesktop.Hal.Device')
    if prod==1:
        return 'OK'
    else:
        return 'NO'
    return 'NO'

def get_so():
    return commands.getstatusoutput('uname -o')[1]

def get_hostname():
    return commands.getstatusoutput('uname -n')[1]
    #return socket.gethostname()

def get_tutti_ip_mac():
    tutti_ip=''
    tutti_mac=''
    bus = dbus.SystemBus()
    objeto_hal = bus.get_object(hal_dot, hal_path+'/Manager')
    hal_manager = dbus.Interface(objeto_hal, hal_dot+'.Manager')

    for device_name in hal_manager.FindDeviceByCapability("net"):

        device = bus.get_object("org.freedesktop.Hal",device_name)

        prod = device.GetPropertyString('info.capabilities',
dbus_interface='org.freedesktop.Hal.Device')
        mac = device.GetPropertyString('net.address',
dbus_interface='org.freedesktop.Hal.Device')
        net = device.GetPropertyString('net.interface',
dbus_interface='org.freedesktop.Hal.Device')

        tutti_mac= tutti_mac + ',' + mac
        tutti_ip= tutti_ip + ',' + get_ip(str(net))
    return tutti_ip.lstrip(','), tutti_mac.lstrip(',')
```



```
#return tutti_ip+',130.192.86.27',
tutti_mac+',00:03:0d:37:02:cc'

def get_WOL(mac_selected):
    bus = dbus.SystemBus()
    objeto_hal = bus.get_object(hal_dot, hal_path+'/Manager')
    hal_manager = dbus.Interface(objeto_hal, hal_dot+'.Manager')

    for device_name in hal_manager.FindDeviceByCapability("net"):

        device = bus.get_object("org.freedesktop.Hal",device_name)

        mac = device.GetPropertyString('net.address',
dbus_interface='org.freedesktop.Hal.Device')
        if mac==mac_selected:
            prod = device.GetPropertyString('info.capabilities',
dbus_interface='org.freedesktop.Hal.Device')
            for lala in prod:
                if lala=='wake_on_lan':
                    return 'OK'
            print 'WARNING: WOL no e attivato'
            #return 'NO'
            return 'OK'

def get_ip(iframe):
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    try:

        ip=socket.inet_ntoa(fcntl.ioctl(s.fileno(),0x8915,struct.pack('2
56s', iframe[:15]))[20:24])
    except EnvironmentError:
        return 'null'
    return ip

def test():
    print get_hib()
    print get_so()
    print get_hostname()
    ip,mac = get_tutti_ip_mac()
    print ip
    print mac
    print get_WOL(('00:03:0D:37:02:cc').lower())

#test()
```

B Polling.py

```
#!/usr/bin/env python

import gobject
import subprocess
import dbus
import dbus.service
from dbus.mainloop.glib import DBusGMainLoop
import httplib
import time
import sys
import commands
import DattiPc
import os
import signal
import shutil

#
# Define global variables
#
MY_BUS_NAME = 'my.polisave'
MY_OBJECT_PATH = '/my/polisave'

pid_dbus=''

def logs():
    #
    # Logging & Debug
    #
    try:
        import logging
    except:
        print 'Failed to import logging module. Exiting...'
        sys.exit(1)

    SELF = 'Polisave'
    DEBUG = True #print all messages in file
    STREAMOUTPUT = False #print in console

    if os.getenv('MWC_DEBUG') != None:
        DEBUG = True

    if os.getenv('MWC_STREAMOUTPUT') != None:
        STREAMOUTPUT = True

    if DEBUG:
        DEBUG_LEVEL = logging.DEBUG #print all messages in file
    else:
        DEBUG_LEVEL = logging.WARNING #print warnings in file

    DEBUG_FORMATTER = '%(asctime)s %(levelname)s %(message)s'

    logging.basicConfig(level = DEBUG_LEVEL,
                        format = DEBUG_FORMATTER,
                        filename = '/var/log/' + SELF +
'.log',
                        filemode = 'a')
```

```

    if STREAMOUTPUT:
        console = logging.StreamHandler()
        console.setLevel(DEBUG_LEVEL)
        console.setFormatter(logging.Formatter(DEBUG_FORMATTER))
        self.logging.getLogger('').addHandler(console)

    return logging

logging=logs()

class DBUSmanagement:

    #@dbus.service.method(MY_BUS_NAME,in_signature='s',
out_signature='as')
    def Continuare(self, continuare):
        logging.debug(str(continuare))
        subprocess.Popen("/usr/local/bin/Polling.py rePolling",
shell=True)
        return ["Okkkkkkk"]

    #@dbus.service.method(MY_BUS_NAME,in_signature='s',
out_signature='as')
    def Uscire(self, numAction):
        if numAction=='1':
            logging.debug('Azione wait')
        elif numAction=='2':
            logging.debug('Azione off')
            commands.getstatusoutput('shutdown -h now')
            #subprocess.Popen("shutdown -h now", shell=True)
        elif numAction=='3':
            logging.debug('Azione hib')
            subprocess.Popen("/usr/local/bin/Polling.py
rePolling", shell=True)
            commands.getstatusoutput('pm-hibernate')
            #subprocess.Popen("suspend.sh hibernate", shell=True)
        elif numAction=='4':
            logging.debug('Azione stby')
            subprocess.Popen("/usr/local/bin/Polling.py
rePolling", shell=True)
            commands.getstatusoutput('pm-suspend')
            #subprocess.Popen("suspend.sh suspend", shell=True)
        elif numAction=='5':
            logging.debug('Azione power on')

        return ["Ok"]

    #@dbus.service.method("com.example.SampleInterface",in_signature
='', out_signature='')
    def Exit(self):
        mainloop.quit()
        exit()

    def __init__(self):
        logging.debug('Inizio DBUS')

        dbus.mainloop.glib.DBusGMainLoop(set_as_default=True)
        bus = dbus.SystemBus()

        bus.add_signal_receiver(self.Continuare,
dbus_interface=MY_BUS_NAME, signal_name="On")

```

```

        bus.add_signal_receiver(self.Uscire,
dbus_interface=MY_BUS_NAME, signal_name="Off")

        mainloop = gobject.MainLoop()
        mainloop = mainloop.run()

class Polisave:
    HOST = 'www.swas.polito.it'
    DIRECTION="/services/polisave/polisave.asp"

    #Datti di Dbus
    NAME="NAME=["+ DattiPc.get_hostname() +"]"
    tutti_IP=""
    tutti_MAC=""

    #Variables
    COUNT_POLLING_STR="COUNT_POLLING="
    countPolling=1
    conn=''
    continuare=True

    #PROGRAMME
    def __init__(self):
        logging.debug('Polling')
        try:
            self.conn = httplib.HTTPConnection(self.HOST)
        except httplib.HTTPException, e:
            logging.critical('Error POLLING: invalid destination
host')
            logging.shutdown()
            exit()
        time.sleep(60)
        #Datti di Dbus
        ipDbus, macDbus = DattiPc.get_tutti_ip_mac()
        self.tutti_IP="IP=["+ ipDbus + "]" #Unico importante
        self.tutti_MAC="MAC=["+ macDbus + "]"
        self.POLLING()

    def SEND_POLLING(self):
        logging.debug('Send Polling')
        ACTION="ACTION=[POLLING]"
        COUNT_POLLING=self.COUNT_POLLING_STR + '['+
str(self.countPolling) +']'
        cadena = self.DIRECTION + "?" + ACTION + "&" + self.NAME +
"&" + COUNT_POLLING + "&" + self.tutti_IP + "&" + self.tutti_MAC
        try:
            self.conn.request("GET", cadena)
        except httplib.HTTPException, e:
            logging.critical('Error send polling')
            logging.shutdown()
        return self.conn.getresponse()

    def RECEIVE_POLLING(self,Rl):
        if (Rl.status!=200):
            logging.critical('Error Polling: risposta diverse a
200-OK')
            logging.shutdown()
            exit()

```

```

        responseRead=R1.read()
        #time_wait=int(6)
        try:

time_wait=int((responseRead.split()[4]).strip('TIME=[]))
        except:
            logging.critical('Error Polling: This is not polito
ip')

            logging.shutdown()
            #logging.critical(pid_dbus)
            #os.kill(int(pid_dbus), signal.SIGTERM)
            sys.exit(1)

        if time_wait < 60:
            action=responseRead.split()[3] #ha de ser 3 (1 a
casa)

            numAction = self.ACTION(action)
            self.continuare=False
            logging.debug('numAction:' + str(numAction))
            #Amb Proces
            if numAction == 6:
                return numAction, responseRead.split()[6]
            elif 0<numAction<6:
                return numAction, 'tutto a posto'
            else:
                logging.critical('Error: response polling')
                logging.shutdown()
                exit()

        else:
            logging.debug('Wait ' + str(time_wait) + 's')
            #time.sleep(60)
            time.sleep(time_wait-30)
            return 1,'tutto a posto'

def POLLING(self):
    self.continuare=True
    while (self.continuare == True):
        response = self.SEND_POLLING()
        azione, msgAzione = self.RECEIVE_POLLING(response)
        self.countPolling+=1
        logging.debug('Open popup')
        user = commands.getoutput("who -
u").splitlines()[0].split()[0]
        XauthDir = "/home/"+user+"/.Xauthority"

        if os.path.exists(XauthDir)!=True:
            logging.critical('Error: could not detect Xwindows')
            logging.shutdown()
            exit()

        os.environ['XAUTHORITY']=XauthDir
        os.environ['DISPLAY']=":0.0"
        child = subprocess.Popen("/usr/local/bin/popupThread.py" +
' ' +
str(azione) + ' ' + msgAzione, shell=True)

    def ACTION(self,action):
        if action=='ACTION=[WAIT]': #hauria de ser wait (Polling a
casa)

            return 1

```

```
        elif action=='ACTION=OFF':
            return 2
        elif action=='ACTION=HIB':
            return 3
        elif action=='ACTION=STBY':
            return 4
        elif action=='ACTION=POWER ON':
            return 5
        elif action=='ACTION=[MSG]':
            return 6
        return 7

class Inizio:
    def __init__(self):
        logging.debug('Polisave started.')
        pid = os.fork()
        if pid != 0:
            Polisave()
        else:
            pid = str(os.getpid())
            file("/var/run/polisave.pid", 'w+').write("%s\n" %
pid)

            DBUSmanagement()

if __name__ == "__main__":
    if (len(sys.argv)>1 and sys.argv[1]=='rePolling'):
        Polisave()
    #else:
    elif (len(sys.argv)>1 and sys.argv[1]=='start'):
        try:
            shutil.copy('/tmp/polisave.txt',
'/usr/local/bin/polisave.txt')
        except OSError:
            logging.debug('Could not copy tmp file')
        try:
            pid_daemon = os.fork()
            if pid_daemon > 0 :
                sys.exit(0)
        except OSError, e:
            logging.critical('Fork error')
            logging.shutdown()
            sys.exit(1)
        os.chdir("/")
        os.setsid()
        os.umask(0)

        Inizio()

    else:
        print 'Command is not valid!'
```

C PopupThread.py

```
#!/usr/bin/env python

import pygtk
import commands
pygtk.require('2.0')
import gtk
import gobject
import time
import os
import sys
import dbus
import dbus.service
from dbus.mainloop.glib import DBusGMainLoop

MY_BUS_NAME = 'my.polisave'
MY_OBJECT_PATH = '/my/polisave'

class Eieruhr(dbus.service.Object):
    def __init__(self, numAction):
        bus = dbus.SystemBus(mainloop=DBusGMainLoop())
        dbus.service.Object.__init__(self, bus, MY_OBJECT_PATH)
        self.numAction = numAction
        self.msgPopUp = self.msgAction(numAction)
        self.temps='60'

    def main2(self):

        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.connect("destroy", self.destroy)
        self.window.set_title("PoliSave")
        self.window.set_default_size(300, 240)

        self.vbox = gtk.VBox()
        self.vbox.show()

        self.label=gtk.Label('PoliSave: spegnimento PC in corso.')
        self.label.show()
        self.vbox.add(self.label)

        self.label2=gtk.Label(self.msgPopUp)
        self.label2.show()
        self.vbox.add(self.label2)

        self.timedisp = gtk.Label("Time left: " + self.temps + "
seconds")
        #self.timedisp.set_alignment(0.5)
        self.timedisp.show()
        self.vbox.add(self.timedisp)

        self.hbox = gtk.HBox()
        self.hbox.show()
        self.vbox.add(self.hbox)

        self.start = gtk.ToggleButton("Continuare")
        self.start.connect("clicked", self.start_clicked)
        self.start.show()
```

```

self.hbox.add(self.start)

self.stop = gtk.Button("Realizzare l'azione")
self.stop.connect("clicked", self.stop_clicked)
self.stop.show()
self.hbox.add(self.stop)

self.lastTime = time.time()
self.timeout = GObject.timeout_add(500, self.timer_tick)

self.window.add(self.vbox)
self.window.show()

def start_clicked(self, widget, data=None):
    self.On('Continuare')
    sys.exit(0)

def stop_clicked(self, widget, data=None):
    self.Off(self.numAction)
    sys.exit(0)

def timer_tick(self):
    curdowntime=self.currentTime(self.temps)
    if(time.time() - self.lastTime) >= 1:
        # 1 sec has passed, so we count the time down
        self.temps="%i" % (curdowntime-1)
        self.timedisp.set_text("Time left: " + self.temps + "
seconds")
        self.lastTime = time.time()
    if curdowntime == 0:
        self.Off(self.numAction)
        sys.exit(0)
        return False
    return True

def currentTime(self, minsec):
    minsec = minsec.split(":")
    return int(minsec[0])

def destroy(self, widget, data=None):
    self.On('Continuare')
    sys.exit(0)

@dbus.service.signal(dbus_interface=MY_BUS_NAME,signature='s')
def Off(self,numAction):
    print("PoliSave: LogOff")

@dbus.service.signal(dbus_interface=MY_BUS_NAME,signature='s')
def On(self,user):
    print("Polisave: continue")

def msgAction(self,numAction):
    if numAction=='1':
        return 'Action: Force WAIT'
    elif numAction=='2':
        return 'Action: Force POWEROFF'
    elif numAction=='3':
        return 'Action: Force HIBERNATION'
    elif numAction=='4':
        return 'Action: Force STANDBY'
    elif numAction=='5':

```



```

        return 'Action: Force POWERON'
    elif numAction=='6':
        if len(sys.argv) > 2:
            j=2
            cadena = ''
            while j<len(sys.argv):
                cadena = cadena + ' ' + sys.argv[j]
                j+=1
            return cadena
        return 'Error'

    def main(self):
        Eieruhr.main2(self)
        gtk.main()

gui = Eieruhr(sys.argv[1])
gui.main()

```

D My.polisave.service

```

[D-BUS Service]
Name=my.polisave
Exec=/usr/local/bin/Polling.py
User=root

```

E My.polisave.conf

```

<?xml version="1.0" encoding="UTF-8"?> <!-- -*- XML -*- -->

<!DOCTYPE busconfig PUBLIC
"-//freedesktop//DTD D-BUS Bus Configuration 1.0//EN"
"http://www.freedesktop.org/standards/dbus/1.0/busconfig.dtd">
<busconfig>

  <!-- Only root can own the service -->
  <policy user="root">
    <allow own="my.polisave"/>
    <allow send_interface="my.polisave.On"/>
    <allow send_interface="my.polisave.Off"/>
  </policy>

  <policy group="users">
    <allow own="my.polisave"/>
    <allow send_interface="my.polisave.On"/>
    <allow send_interface="my.polisave.Off"/>
  </policy>

  <!-- Allow anyone to invoke methods on the interfaces -->
  <policy context="default">
    <allow send_interface="my.polisave.On"/>
    <allow send_interface="my.polisave.Off"/>
  </policy>

</busconfig>

```

F Installazione.py

Installazione.py (Ubuntu)

```
#!/usr/bin/env python

import httplib
import webbrowser
import DattiPc
import shutil
import os
import commands
import sys
import platform
import subprocess

HOST = 'www.swas.polito.it'
DIRECTION="/services/polisave/polisave.asp"

#Dati di Dbus
NAME="NAME=[" + DattiPc.get_hostname() + "]"
ipDbus, macDbus = DattiPc.get_tutti_ip_mac()
tutti_IP="IP=[" + ipDbus + "]" #Unico importante
tutti_MAC="MAC=[" + macDbus + "]"
OS="OS=[" + DattiPc.get_so() + "]"
HIB="HIB=[" + DattiPc.get_hib() + "]"
WOL="WOL=[NULL]"

IP="IP=[null]"
MAC="MAC=[null]"
HF="HF=[" + DattiPc.get_hib() + "]"
S4="S4=[" + DattiPc.get_hib() + "]"

conn = httplib.HTTPConnection(HOST)

def START_SETUP(conn):
    ACTION="ACTION=[START_SETUP]"
    cadena = DIRECTION + "?" + ACTION + "&" + NAME + "&" + OS + "&"
+ HIB + "&" + tutti_IP + "&" + tutti_MAC
    conn.request("GET", cadena)
    return conn.getresponse()

def END_SETUP(conn, IP, MAC):
    ACTION="ACTION=[END_SETUP]"
    cadena = DIRECTION + "?" + ACTION + "&" + WOL + "&" + NAME + "&"
+ IP + "&" + MAC
    conn.request("GET", cadena)
    return conn.getresponse()

def SHOW_HELP_ERROR(conn,reason):
    ACTION="ACTION=[SHOW_HELP]"
    ERROR="ERROR=[" + reason + "]"
    cadena = DIRECTION + "?" + ACTION + "&" + ERROR + "&" + NAME +
"&" + OS + "&" + HIB + "&" + IP + "&" + MAC
```

```

conn.request("GET", cadena)
return conn.getresponse()

def SHOW_HELP_OK(conn):
    ACTION="ACTION=[SHOW_HELP]"
    ERROR="ERROR=[OK]"
    cadena = DIRECTION + "?" + ACTION + "&" + ERROR + "&" + WOL +
"&" + OS + "&" + HF + "&" + IP + "&" + MAC + "&" + S4
    conn.request("GET", cadena)
    return conn.getresponse()

def AprireWeb(response):
    #print("Aprire explorer")
    webbrowser.open_new('http://www.swas.polito.it/services/polisave
/'+response.getheader('Location'))
    exit()

def ErrorToShell(response, responseRead, reason):
    print(response.status)
    print(responseRead)
    rError = SHOW_HELP_ERROR(conn,reason)
    AprireWeb(rError)

def ComprovazioneOK(response):
    responseRead= response.read();
    IP=responseRead.split()[1]
    MAC=responseRead.split()[2]

    if 'ANS=[200]' != responseRead.split()[0]:
        ErrorToShell(response, responseRead, 'NOIPMAC')
    else:
        return IP, MAC

def InstallazioneDir():
    version_py = '/usr/lib/python'+platform.python_version()
    lib_path=sys.path[1]
    for i in sys.path:
        if version_py.startswith(i):
            lib_path=i

    try:
        os.link('DattiPc.py', lib_path+'/DattiPc.py')
        os.link('Polling.py', '/usr/local/bin/Polling.py')
        os.link('popupThread.py', '/usr/local/bin/popupThread.py')
        shutil.copy('my.polisave.service', '/usr/share/dbus-
1/services/my.polisave.service')
        shutil.copy('my.polisave.conf', '/etc/dbus-
1/system.d/my.polisave.conf')

        #Ubuntu
        runlevel=(commands.getstatusoutput('runlevel'))[1].strip()[2]
        shutil.copy('polisaveUbuntu.sh', '/etc/init.d/polisave.sh')
        os.symlink('/etc/init.d/polisave.sh',
'/etc/rc'+runlevel+'.d/S99polisave')

    except OSError:
        print 'ERRORE: non puo copiare i file'
        sys.exit(1)
    subprocess.Popen('/etc/init.d/polisave.sh start', shell=True)

#====PROEDIMENT CORRECTE====

```

```

#Devi essere Root per fare l'installazione
if os.geteuid() != 0:
    print "ERRORE: Devi essere root per installare l'applicazione"
    sys.exit(1)

if os.path.exists('/usr/sbin/pm-suspend')!=True or
os.path.exists('/usr/sbin/pm-hibernate')!=True:
    print "ERRORE: E necessario installare 'pm-utils' per usare
Polisave."
    sys.exit(1)

r1 = START_SETUP(conn)

if r1.status==200:
    IP, MAC = ComprovazioneOK(r1)
    mac2 = MAC.lstrip('MAC=')
    mac2 = mac2.rstrip(']')
    WOL="WOL=["+DattiPc.get_WOL(mac2.lower())+"]"
elif r1.status==500:
    AprireWeb(r1)
else:
    ErrorToShell(r1, r1.read(),'NOPARS')

r2 = END_SETUP(conn,IP, MAC)

if r2.status==200:
    IP, MAC = ComprovazioneOK(r2)
else:
    ErrorToShell(r2, r2.read(),'NOPARS')

if WOL!="WOL=[OK]":
    rError = SHOW_HELP_ERROR(conn, 'NOCONF')
    AprireWeb(rError)

r3 = SHOW_HELP_OK(conn)
InstallazioneDir()
print "Installazione corretta!"
AprireWeb(r3)
conn.close()

```

Installazione.py (Fedora)

```

#!/usr/bin/env python

import httplib
import webbrowser
import DattiPc
import shutil
import os
import commands
import sys
import platform
import subprocess

HOST = 'www.swas.polito.it'
DIRECTION="/services/polisave/polisave.asp"

```

```

#Dati di Dbus
NAME="NAME=[" + DattiPc.get_hostname() + "]"
ipDbus, macDbus = DattiPc.get_tutti_ip_mac()
tutti_IP="IP=[" + ipDbus + "]" #Unico importante
tutti_MAC="MAC=[" + macDbus + "]"
OS="OS=[" + DattiPc.get_so() + "]"
HIB="HIB=[" + DattiPc.get_hib() + "]"
WOL="WOL=[NULL]"

IP="IP=[null]"
MAC="MAC=[null]"
HF="HF=[" + DattiPc.get_hib() + "]"
S4="S4=[" + DattiPc.get_hib() + "]"

conn = httplib.HTTPConnection(HOST)

def START_SETUP(conn):
    ACTION="ACTION=[START_SETUP]"
    cadena = DIRECTION + "?" + ACTION + "&" + NAME + "&" + OS + "&"
+ HIB + "&" + tutti_IP + "&" + tutti_MAC
    conn.request("GET", cadena)
    return conn.getresponse()

def END_SETUP(conn, IP, MAC):
    ACTION="ACTION=[END_SETUP]"
    cadena = DIRECTION + "?" + ACTION + "&" + WOL + "&" + NAME + "&"
+ IP + "&" + MAC
    conn.request("GET", cadena)
    return conn.getresponse()

def SHOW_HELP_ERROR(conn,reason):
    ACTION="ACTION=[SHOW_HELP]"
    ERROR="ERROR=[" + reason + "]"
    cadena = DIRECTION + "?" + ACTION + "&" + ERROR + "&" + NAME +
"&" + OS + "&" + HIB + "&" + IP + "&" + MAC
    conn.request("GET", cadena)
    return conn.getresponse()

def SHOW_HELP_OK(conn):
    ACTION="ACTION=[SHOW_HELP]"
    ERROR="ERROR=[OK]"
    cadena = DIRECTION + "?" + ACTION + "&" + ERROR + "&" + WOL +
"&" + OS + "&" + HF + "&" + IP + "&" + MAC + "&" + S4
    conn.request("GET", cadena)
    return conn.getresponse()

def AprireWeb(response):
    #print("Aprire explorer")
    webbrowser.open_new('http://www.swas.polito.it/services/polisave
/'+response.getheader('Location'))
    exit()

def ErrorToShell(response, responseRead, reason):
    print(response.status)
    print(responseRead)
    rError = SHOW_HELP_ERROR(conn,reason)
    AprireWeb(rError)

def ComprovazioneOK(response):
    responseRead= response.read();

```

```

IP=responseRead.split()[1]
MAC=responseRead.split()[2]

if 'ANS=[200]' != responseRead.split()[0]:
    ErrorToShell(response, responseRead, 'NOIPMAC')
else:
    return IP, MAC

def InstallazioneDir():
    version_py = '/usr/lib/python'+platform.python_version()
    lib_path=sys.path[1]
    for i in sys.path:
        if version_py.startswith(i):
            lib_path=i

    try:
        os.link('DattiPc.py', lib_path+'/DattiPc.py')
        os.link('Polling.py', '/usr/local/bin/Polling.py')
        os.link('popupThread.py', '/usr/local/bin/popupThread.py')
        shutil.copy('my.polisave.service', '/usr/share/dbus-
1/services/my.polisave.service')
        shutil.copy('my.polisave.conf', '/etc/dbus-
1/system.d/my.polisave.conf')

        #Fedora
        runlevel=(commands.getstatusoutput('runlevel'))[1].strip()[2]

        shutil.copy('cpXauth.sh', '/etc/X11/xinit/xinitrc.d/cpXauth.sh')
        shutil.copy('polisaveFedora.sh', '/etc/init.d/polisave.sh')
        os.symlink('/etc/init.d/polisave.sh',
'/etc/rc'+runlevel+'.d/S99polisave')

    except OSError:
        print 'ERRORE: non puo copiare i file'
        sys.exit(1)
    subprocess.Popen('/etc/init.d/polisave.sh start', shell=True)

#====PROEDIMENT CORRECTE====

#Devi essere Root per fare l'installazione
if os.geteuid() != 0:
    print "ERRORE: Devi essere root per installare l'applicazione"
    sys.exit(1)

if os.path.exists('/usr/sbin/pm-suspend')!=True or
os.path.exists('/usr/sbin/pm-hibernate')!=True:
    print "ERRORE: E necessario installare 'pm-utils' per usare
Polisave."
    sys.exit(1)

r1 = START_SETUP(conn)

if r1.status==200:
    IP, MAC = ComprovazioneOK(r1)
    mac2 = MAC.lstrip('MAC=')
    mac2 = mac2.rstrip(']')
    WOL="WOL=["+DattiPc.get_WOL(mac2.lower())+"]"
elif r1.status==500:
    AprireWeb(r1)
else:
    ErrorToShell(r1, r1.read(), 'NOPARS')

```

```

r2 = END_SETUP(conn,IP, MAC)

if r2.status==200:
    IP, MAC = ComprovazioneOK(r2)
else:
    ErrorToShell(r2, r2.read(),'NOPARS')

if WOL!="WOL=[OK]":
    rError = SHOW_HELP_ERROR(conn, 'NOCONF')
    AprireWeb(rError)

r3 = SHOW_HELP_OK(conn)
InstallazioneDir()
print "Installazione corretta!"
print "Per il buon funzionamento di Polisave devi riiniziare il
computer"
AprireWeb(r3)
conn.close()

```

Installazione.py (OpenSUSE)

```

#!/usr/bin/env python

import httplib
import webbrowser
import DattiPc
import shutil
import os
import commands
import sys
import platform
import subprocess

HOST = 'www.swas.polito.it'
DIRECTION="/services/polisave/polisave.asp"

#Dati di Dbus
NAME="NAME=[" + DattiPc.get_hostname() + "]"
ipDbus, macDbus = DattiPc.get_tutti_ip_mac()
tutti_IP="IP=[" + ipDbus + "]" #Unico importante
tutti_MAC="MAC=[" + macDbus + "]"
OS="OS=[" + DattiPc.get_so() + "]"
HIB="HIB=[" + DattiPc.get_hib() + "]"
WOL="WOL=[NULL]"

IP="IP=[null]"
MAC="MAC=[null]"
HF="HF=[" + DattiPc.get_hib() + "]"
S4="S4=[" + DattiPc.get_hib() + "]"

conn = httplib.HTTPConnection(HOST)

def START_SETUP(conn):
    ACTION="ACTION=[START_SETUP]"
    cadena = DIRECTION + "?" + ACTION + "&" + NAME + "&" + OS + "&"
+ HIB + "&" + tutti_IP + "&" + tutti_MAC
    conn.request("GET", cadena)
    return conn.getresponse()

```

```

def END_SETUP(conn, IP, MAC):
    ACTION="ACTION=[END_SETUP]"
    cadena = DIRECTION + "?" + ACTION + "&" + WOL + "&" + NAME + "&"
+ IP + "&" + MAC
    conn.request("GET", cadena)
    return conn.getresponse()

def SHOW_HELP_ERROR(conn,reason):
    ACTION="ACTION=[SHOW_HELP]"
    ERROR="ERROR=[" + reason + "]"
    cadena = DIRECTION + "?" + ACTION + "&" + ERROR + "&" + NAME +
"&" + OS + "&" + HIB + "&" + IP + "&" + MAC
    conn.request("GET", cadena)
    return conn.getresponse()

def SHOW_HELP_OK(conn):
    ACTION="ACTION=[SHOW_HELP]"
    ERROR="ERROR=[OK]"
    cadena = DIRECTION + "?" + ACTION + "&" + ERROR + "&" + WOL +
"&" + OS + "&" + HF + "&" + IP + "&" + MAC + "&" + S4
    conn.request("GET", cadena)
    return conn.getresponse()

def AprireWeb(response):
    #print("Aprire explorer")
    webbrowser.open_new('http://www.swas.polito.it/services/polisave
/'+response.getheader('Location'))
    exit()

def ErrorToShell(response, responseRead, reason):
    print(response.status)
    print(responseRead)
    rError = SHOW_HELP_ERROR(conn,reason)
    AprireWeb(rError)

def ComprovazioneOK(response):
    responseRead= response.read();
    IP=responseRead.split()[1]
    MAC=responseRead.split()[2]

    if 'ANS=[200]' != responseRead.split()[0]:
        ErrorToShell(response, responseRead, 'NOIPMAC')
    else:
        return IP, MAC

def InstallazioneDir():
    version_py = '/usr/lib/python'+platform.python_version()
    version_py2 = '/usr/lib64/python'+platform.python_version()
    lib_path=sys.path[1]
    for i in sys.path:
        if version_py2.startswith(i):
            lib_path=i
        if version_py.startswith(i):
            lib_path=i

    try:
        shutil.copy('DattiPc.py', lib_path+'/DattiPc.py')
        shutil.copy('Polling.py', '/usr/local/bin/Polling.py')
        shutil.copy('popupThread.py',
'/usr/local/bin/popupThread.py')

```



```

        shutil.copy('my.polisave.service', '/usr/share/dbus-
1/services/my.polisave.service')
        shutil.copy('my.polisave.conf', '/etc/dbus-
1/system.d/my.polisave.conf')

        #OpenSuse
        shutil.copy('polisaveOpenSuse.sh', '/etc/init.d/polisave.sh')
        shutil.copy('cpXauth.sh', '/etc/X11/xinit/xinitrc.d/90-
cpXauth.sh')
        commands.getstatusoutput('insserv /etc/init.d/polisave.sh')
        os.symlink('/etc/init.d/polisave.sh', '/sbin/rcpolisave')

    except OSError:
        print 'ERRORE: non puo copiare i file'
        sys.exit(1)
    subprocess.Popen('/etc/init.d/polisave.sh start', shell=True)

#====PROEDIMENT CORRECTE====

#Devi essere Root per fare l'installazione
if os.geteuid() != 0:
    print "ERRORE: Devi essere root per installare l'applicazione"
    sys.exit(1)

if os.path.exists('/usr/sbin/pm-suspend')!=True or
os.path.exists('/usr/sbin/pm-hibernate')!=True:
    print "ERRORE: E necessario installare 'pm-utils' per usare
Polisave."
    sys.exit(1)

r1 = START_SETUP(conn)

if r1.status==200:
    IP, MAC = ComprovazioneOK(r1)
    mac2 = MAC.lstrip('MAC=')
    mac2 = mac2.rstrip(']')
    WOL="WOL=["+DattiPc.get_WOL(mac2.lower())+"]"
elif r1.status==500:
    AprireWeb(r1)
else:
    ErrorToShell(r1, r1.read(), 'NOPARS')

r2 = END_SETUP(conn, IP, MAC)

if r2.status==200:
    IP, MAC = ComprovazioneOK(r2)
else:
    ErrorToShell(r2, r2.read(), 'NOPARS')

if WOL!="WOL=[OK]":
    rError = SHOW_HELP_ERROR(conn, 'NOCONF')
    AprireWeb(rError)

r3 = SHOW_HELP_OK(conn)
InstallazioneDir()
print "Installazione corretta!"
AprireWeb(r3)
conn.close()

```

G Disinstallazione.py

Disinstallazione.py (Ubuntu)

```
#!/usr/bin/env python

import commands
import sys
import os
import platform
import subprocess
import time

if os.geteuid() != 0:
    print "ERRORE: Devi essere root per disinstallare
l'applicazione"
    sys.exit()

process = subprocess.Popen('/etc/init.d/polisave.sh stop', shell=True,
stderr=subprocess.PIPE)
process.wait()

print 'Disinstallazione...'

runlevel=(commands.getstatusoutput('runlevel'))[1].strip()[2]
version_py = '/usr/lib/python'+platform.python_version()
lib_path=sys.path[1]
for i in sys.path:
    if version_py.startswith(i):
        lib_path=i

try:
    os.remove(lib_path+ '/DattiPc.py')
except OSError:
    print 'WARNING: non puo cancellare il file ' + lib_path +
'/DattiPc.py'
try:
    os.remove('/usr/local/bin/Polling.py')
except OSError:
    print 'WARNING: non puo cancellare il file
/usr/local/bin/Polling.py'
try:
    os.remove('/usr/local/bin/popupThread.py')
except OSError:
    print 'WARNING: non puo cancellare il file
/usr/local/bin/popupThread.py'
try:
    os.remove('/etc/init.d/polisave.sh')
except OSError:
    print 'WARNING: non puo cancellare il file
/etc/init.d/polisave.sh'
try:
    os.remove('/usr/share/dbus-1/services/my.polisave.service')
    os.remove('/etc/dbus-1/system.d/my.polisave.conf')
except OSError:
    print 'WARNING: non puo cancellare i file my.polisave.service e
my.polisave.service'
```

```

try:
    os.remove('/var/log/Polisave.log')
    os.remove(lib_path + '/DattiPc.pyc')
except OSError:
    disinstallazione = True

try:
    os.remove('/etc/rc'+runlevel+'.d/S99polisave') #Fedora e Ubuntu
except OSError:
    disinstallazione = True

print 'Disinstallazione corretta!'

```

Disinstallazione.py (Fedora)

```

#!/usr/bin/env python

import commands
import sys
import os
import platform
import subprocess
import time

if os.geteuid() != 0:
    print "ERRORE: Devi essere root per disinstallare
l'applicazione"
    sys.exit()

process = subprocess.Popen('/etc/init.d/polisave.sh stop', shell=True,
stderr=subprocess.PIPE)
process.wait()

print 'Disinstallazione...'

runlevel=(commands.getstatusoutput('runlevel'))[1].strip()[2]
version_py = '/usr/lib/python'+platform.python_version()
lib_path=sys.path[1]
for i in sys.path:
    if version_py.startswith(i):
        lib_path=i

try:
    os.remove(lib_path+ '/DattiPc.py')
except OSError:
    print 'WARNING: non puo cancellare il file ' + lib_path +
'/DattiPc.py'
try:
    os.remove('/usr/local/bin/Polling.py')
except OSError:
    print 'WARNING: non puo cancellare il file
/usr/local/bin/Polling.py'
try:
    os.remove('/usr/local/bin/popupThread.py')
except OSError:
    print 'WARNING: non puo cancellare il file
/usr/local/bin/popupThread.py'
try:

```

```

        os.remove('/etc/init.d/polisave.sh')
except OSError:
    print 'WARNING: non puo cancellare il file
/etc/init.d/polisave.sh'
try:
    os.remove('/usr/share/dbus-1/services/my.polisave.service')
    os.remove('/etc/dbus-1/system.d/my.polisave.conf')
except OSError:
    print 'WARNING: non puo cancellare i file my.polisave.service e
my.polisave.service'

try:
    os.remove('/var/log/Polisave.log')
    os.remove(lib_path + '/DattiPc.pyc')
except OSError:
    disinstallazione = True

try:
    os.remove('/etc/rc'+runlevel+'.d/S99polisave') #Fedora e Ubuntu
    os.remove('/etc/X11/xinit/xinitrc.d/cpXauth.sh') #Fedora
    user = commands.getoutput("who -u").splitlines()[0].split()[0]
    os.remove("/home/"+user+"/.Xauthority")
except OSError:
    print 'WARNING: non puo cancellare i file del daemon'

print 'Disinstallazione corretta!'

```

Disinstallazione.py (OpenSUSE)

```

#!/usr/bin/env python

import commands
import sys
import os
import platform
import subprocess
import time

if os.geteuid() != 0:
    print "ERRORE: Devi essere root per disinstallare
l'applicazione"
    sys.exit()

process = subprocess.Popen('/etc/init.d/polisave.sh stop', shell=True,
stderr=subprocess.PIPE)
process.wait()

print 'Disinstallazione...'

version_py = '/usr/lib/python'+platform.python_version()
lib_path=sys.path[1]
for i in sys.path:
    if version_py.startswith(i):
        lib_path=i
try: #OpenSuse
    commands.getstatusoutput('insserv -r /etc/init.d/polisave.sh')
except e:
    disinstallazione = True

```

```
try:
    os.remove(lib_path+ '/DattiPc.py')
except OSError:
    print 'WARNING: non puo cancellare il file ' + lib_path +
'/DattiPc.py'
try:
    os.remove('/usr/local/bin/Polling.py')
except OSError:
    print 'WARNING: non puo cancellare il file
/usr/local/bin/Polling.py'
try:
    os.remove('/usr/local/bin/popupThread.py')
except OSError:
    print 'WARNING: non puo cancellare il file
/usr/local/bin/popupThread.py'
try:
    os.remove('/etc/init.d/polisave.sh')
except OSError:
    print 'WARNING: non puo cancellare il file
/etc/init.d/polisave.sh'
try:
    os.remove('/usr/share/dbus-1/services/my.polisave.service')
    os.remove('/etc/dbus-1/system.d/my.polisave.conf')
except OSError:
    print 'WARNING: non puo cancellare i file my.polisave.service e
my.polisave.service'

try:
    os.remove('/var/log/Polisave.log')
    os.remove(lib_path + '/DattiPc.pyc')
except OSError:
    disinstallazione = True

try:
    os.remove('/sbin/rcpolisave') # OpenSuse
except OSError:
    disinstallazione = True

print 'Disinstallazione corretta!'
```

H Polisave daemon

PolisaveUbuntu.sh

```
#!/bin/sh
#
# Startup/shutdown script for Polisave.
#
# @Author Joan Vila <email>

### BEGIN INIT INFO
# Provides: polisave
# Required-Start:
# Required-Stop:
# Should-Start:
# Should-Stop:
# Default-Start: 2 3 5
# Default-Stop:
# Short-Description:
# Description:
### END INIT INFO

POLISAVE_BIN="/usr/local/bin/Polling.py"
ARG="start"
PIDFILE="/var/run/polisave.pid"
case "$1" in

    start)
        # start daemon
        if [ -f $PIDFILE ]; then
            echo -n "Daemon gia aperto"
        else
            echo -n "Starting Polisave service\n"
            start-stop-daemon --start --pidfile $PIDFILE --exec
$POLISAVE_BIN $ARG
        fi
        ;;

    stop)
        # stop daemon
        echo -n "Stopping Polisave service\n"
        start-stop-daemon --stop --pidfile $PIDFILE
        rm -f $PIDFILE
        ;;

    restart)
        $0 stop
        $0 start
        ;;

    *)
        echo "Usage: $0 {start|stop|restart}"
        exit 1
        ;;

esac
```

PolisaveFedora.sh

```
#!/bin/bash
#
# Init file for mydaemon
#
#
#           Run-Level Start Stop
#           vvvv   vv   vv
# chkconfig: 2345   99   25
#
# description: Polisave
#
# processname: Polisave
# config: /etc/polisave.conf
# pidfile: /var/run/polisave.pid

# source function library
. /etc/rc.d/init.d/functions

# you may keep some variables in an external file
# for easy access so pull in sysconfig settings
[ -f /etc/sysconfig/polisave ] && . /etc/sysconfig/polisave

# Some more variables to make the script readable
MYDAEMON="/usr/bin/python /usr/local/bin/Polling.py start"
PID_FILE=/var/run/polisave.pid
RETVAL=0
prog="Polisave"

start()
{
    if [ ! -f /var/lock/subsys/polisave ]; then
        echo $"Starting $prog:"
        daemon $MYDAEMON && success || failure
        RETVAL=$?
        [ "$RETVAL" = 0 ] && touch /var/lock/subsys/polisave
    else
        echo $"$prog gia iniziato"
    fi
}

stop()
{
    echo -n $"Stopping $prog:"
    killproc $MYDAEMON -TERM
    RETVAL=$?
    [ "$RETVAL" = 0 ] && rm -f /var/lock/subsys/polisave
    echo
}

reload()
{
    echo -n $"Reloading $prog:"
    killproc $MYDAEMON -HUP
    RETVAL=$?
    echo
}
```

```
}
case "$1" in
    #DDD='444'
    #export $DDD
    start)
        #status $MYDAEMON && exit 0
        start
        ;;
    stop)
        stop
        ;;
    restart)
        stop
        start
        ;;
    reload)
        reload
        ;;
    status)
        status $MYDAEMON
        RETVAL=$?
        ;;
    *)
        echo $"Usage: $0 {start|stop|restart|reload|status}"
        RETVAL=1
esac
exit $RETVAL
```

PolisaveOpenSuse.sh

```
#!/bin/sh
#
# Startup/shutdown script for Polisave.
#
# @Author Joan Vila <email>

### BEGIN INIT INFO
# Provides: polisave
# Required-Start:
# Required-Stop:
# Should-Start:
# Should-Stop:
# Default-Start: 2 3 5
# Default-Stop:
# Short-Description:
# Description:
### END INIT INFO

. /etc/rc.status

# First reset status of this service
rc_reset

POLISAVE_BIN="/usr/bin/python /usr/local/bin/Polling.py"
ARG="start"
PIDFILE="/var/run/polisave.pid"
```



```
case "$1" in
start)
    # start daemon
    if [ -f $PIDFILE ]; then
        echo "Daemon gia aperto"
    else
        echo "Starting Polisave service "
        /sbin/startproc -f $POLISAVE_BIN $ARG
        rc_status -v
    fi
    ;;

stop)
    # stop daemon
    echo "Stopping Polisave service"
    /sbin/killproc -L -p $PIDFILE $POLISAVE_BIN
    rm -f $PIDFILE
    rc_status -v
    ;;

status)
    # status daemon
    echo -n "Status Polisave service\n"
    /sbin/checkproc -L $POLISAVE_BIN
    rc_status -v
    ;;

restart)
    $0 stop
    $0 start
    rc_status
    ;;

*)
    echo "Usage: $0 {start|stop|restart|status}"
    exit 1
    ;;

esac
rc_exit
```