ProActive Screensaver

Table of contents

Philippe Gouttefarde – Nice University
Intership 2011 in ActiveEon Company

I - Introduction:

To develop a screensaver means to be able to generate image on the XSCREENSAVER_WINDOW object. This object is managed by xscreensaver on server X.

Historic of screensaver: http://www.jwz.org/xscreensaver/man1.html

What do the screensaver must provide:

- Trace the non-using of system (absence of keyboard input and events from the mouse);
- Starts the screen saver chosen by user;
- Automatic switch off of the mouse's cursor;
- If necessary locks the screen;
- Provide user's autentification;
- Finishes work of the screen saver.

The developer of screensaver must do:

- To provide display of graphic demonstration in a window with the given identifier;
- To provide a choice of working mode of the screen saver with means of the command line;

KDE spec:

- Developer must to realize the user graphic interface for setting of screen saver's parameters;
- Screensaver is written in QT(C++)
- Source : http://qt.osdn.org.ua/screensaver.en.html

Gnome spec:

- Screensaver is written in GTK+(C or Python)
- Source : http://live.gnome.org/GnomeScreensaver

Debian like spec (ex: Ubuntu):

- Startup script in /etc/init.d (add it with “update-rc.d <script name> defaults” commands).
- Screensavers are in /usr/lib/xscreensaver
- The configurations file .desktop for screensaver in /usr/share/applications/screensavers folder.

Red Hat like spec (ex: Fedora):

- Screensavers are in /usr/lib/xscreensaver
- The configurations file .xml for screensaver in /usr/share/xscreensaver/config folder.

II - Configuration:

Each screensaver get a configuration file to make know the screensaver to the windows-manager.
The configuration file get .desktop extension and get into some information (name, execution commands, options, comments …).
The emplacement:
KDE: /usr/share/kde4/services/ScreenSavers/
Gnome: /usr/share/applications/screensaver/

A configuration file feels like:

```
[Desktop Entry]

Exec=/path/to/ProActiveScreenSaver
Icon=preferences-desktop-screensaver
Type=Service
X-KDE-ServiceTypes=ScreenSaver
Actions=Setup;InWindow;Root;
# X-KDE-Category=Miscellaneous
Name= ProActiveScreenSaver
Name[ca]= ProActiveScreenSaver
```

KDE

```
[Desktop Entry]
Name= ProactiveScreensaver
Exec=/usr/lib/xscreensaver/ProActive
-root
TryExec=/usr/lib/xscreensaver/ProAct
ive
Comment=ProActive Screensaver
StartupNotify=false
Terminal=false
Type=Application
Categories=Screensaver;
OnlyShowIn=GNOME;
```

Gnome

Access rights: -rwxr-xr-x (755)
Owner: root

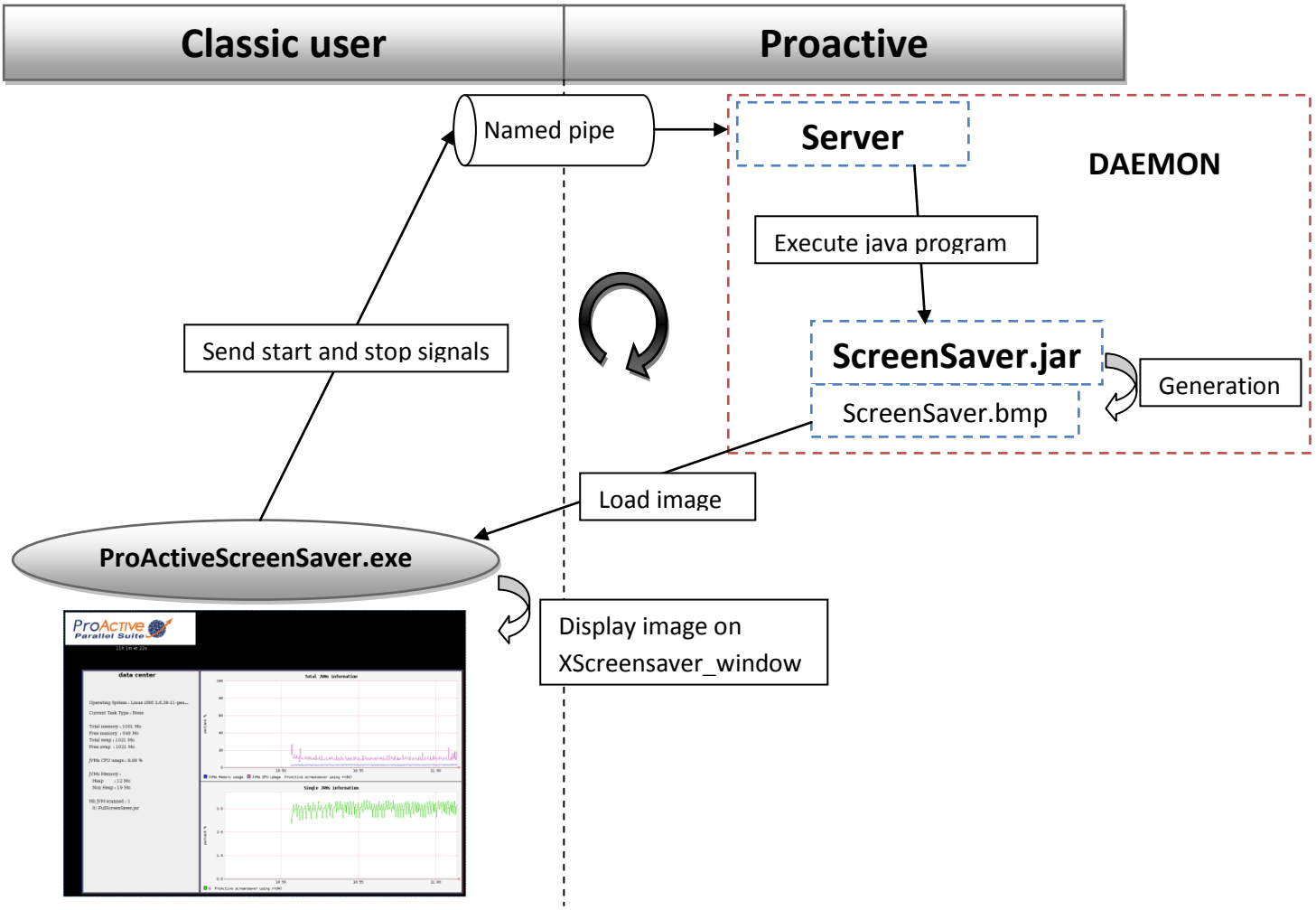
This configuration file allows our new screensaver to be displayed in preference pop-up and be selected.

The different windows manager are built on X server. To be system independent, it will be preferable to get a direct access to the X window.
So, the screensaver code must display picture on the XSCREENSAVER_WINDOW. This is the anid (a native pixmap handle) of the screensaver layer.

The screensaver executable is written in Python; owner is root and gets the same access rights: -rwxr-xr-x (755)

III - Application:

The ProActive Screensaver is build around of a server. It will manage all mechanism and centralize log files.



The daemon:

The daemon is written in python and JAVA language.
It allows establishing different link between system screensaver and ProActive suite.
The daemon is composed of a server, a Java file and a screensaver executable.
The communication between this entities are realized in named pipe: /tmp/ss.pipe
The screensaver executable write into named pipe state information about the screensaver status (start and stop).
This named pipe is a management of user access rights, in fact, the users getting no one access of ProActive Screensaver will be forbidden to write in the named pipe.

Server:

The server start during system boot by proactive user (same that the agent user).
This server listen signals send by proxy and react at 2 different signals.
The signals are built in this syntax: [START/STOP] [X] [Y]

START/STOP: Main signal sent by proxy, a START mean server have to execute screensaver mechanism. A STOP means to stop it…

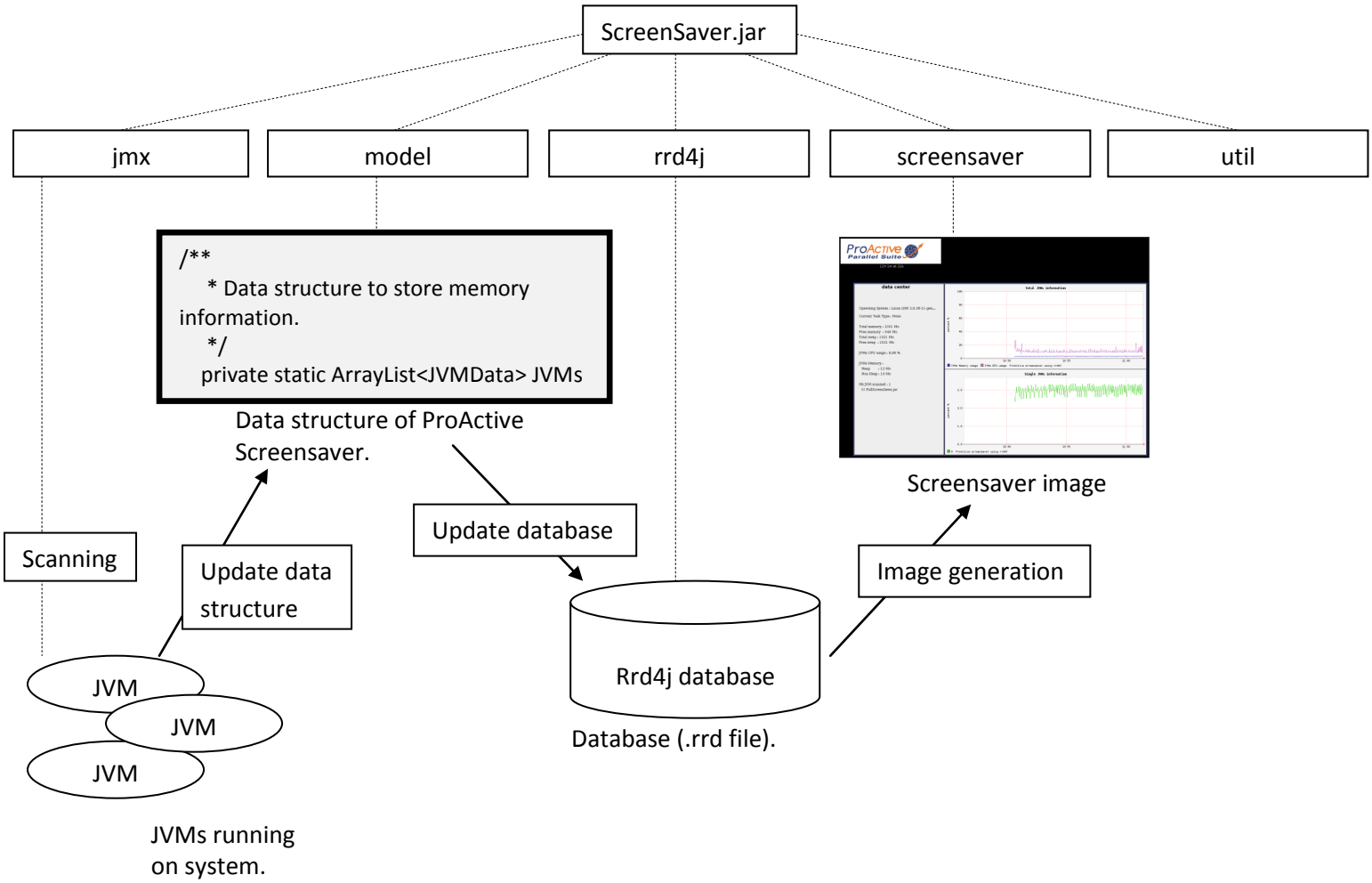
X: The screen length in pixel of graphical user session.

Y: The screen height in pixel of graphical user session.

The server knows how many users are logged on it (get a client counter). It run the mechanism at the first START and cut Java process at the last STOP.

JAVA:

This JAVA application intended to create picture synthesizing ProActive agent JVMs state. This picture will be drawn on screensaver. This code is executing in parallel of screensaver, and generate regular BMP file. The connections are established with JMX. A database RRD is set to keep data between 2 screensavers. JAVA code is executed by proactive user in order to scan agent's JVMs.



Screensaver package:

Remember that the ProActive screensaver displays on screen two distinct zones. One of these show digit data about system and the second draw some chart about system and agent's JVMs. This package contains ProActive picture. The graphical part is composing of four classes. The main checks arguments and initializes components. The three other represent the screen elements.

DrawingClass: This class initializes the 2 geographic areas adapting them to screen dimensions.

DataAera: The digit area.

ChartAera: The chart area, itself cut in 2 little charts. On about system and second draws one curve for each JVM scanned.

JMX package:

This package contains three classes. Its role is to centralize all the information necessary for the screensaver from different JVMs running on system. The information checked to represent a JVM is the following:

String name: The JVM name
Int PID: The process PID
Long startTime: The start timestamp of process
MemHeap: The heap memory of process
MemNonHeap: The non heap memory of process
Double CPU: The load CPU of process on system
JMXConnector connector: JAVA objects keeping connection between the JVM and screensaver.

All information provides a unique representation of each scanned JVM by screensaver. The memory data and CPU data are displaying on chart.

The JVMDetector class establish link with JVMs running on system regularly. Its tools are in "bin" folder of JDK (1.6 release at least). The file jps.jar allows to establish a connection with JVM and to get back process PID. The JVMDetector class builds 2 different objects, a process map and a PID array then.

The ClientJMX class get main role in screensaver. It manages all items give by JVMDetector, update Model, and send compute data to database RRD.

RRD4J Package:

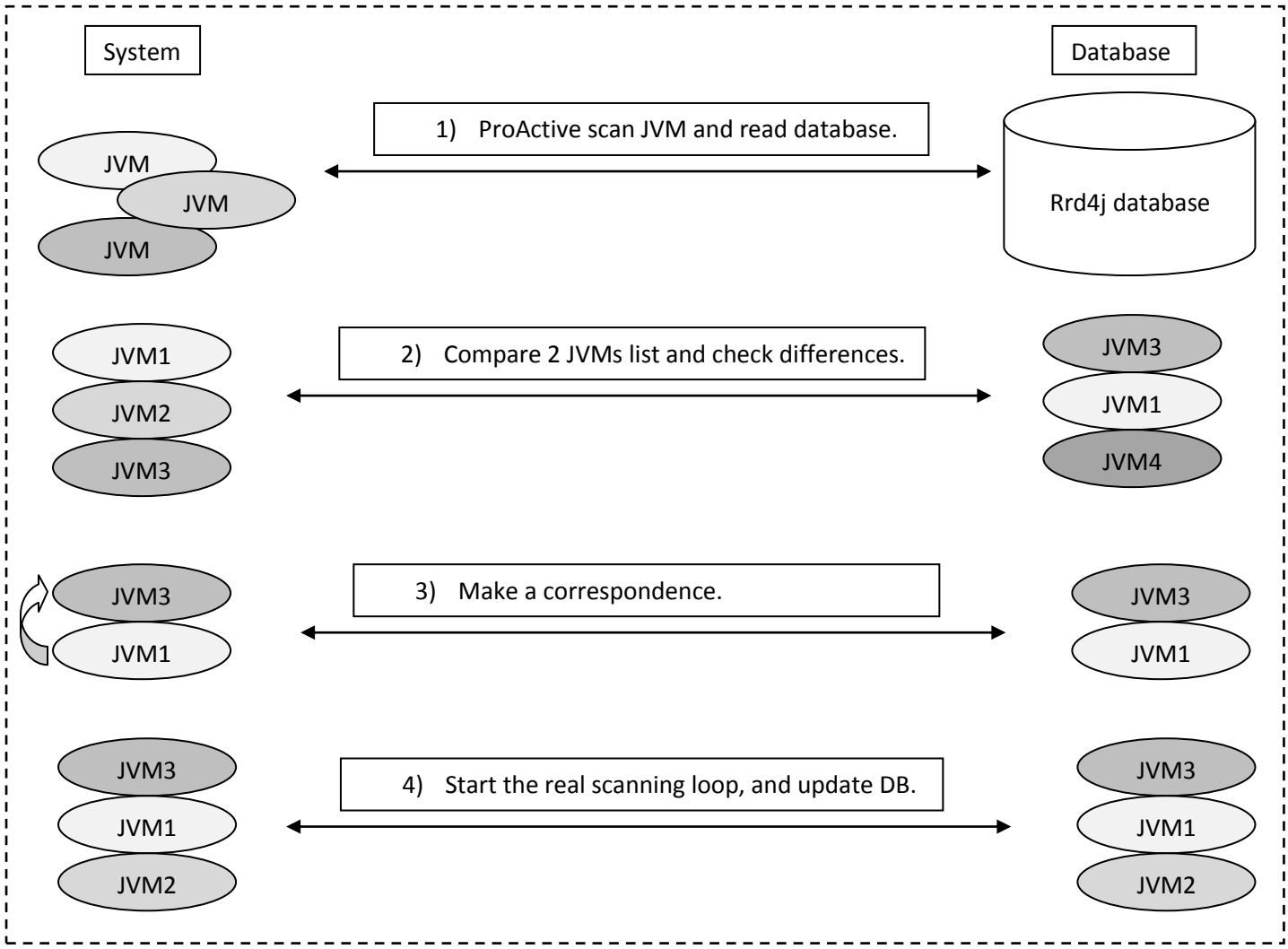
This package links application to database. Remember that database is a RoundRobin database, which is adapted with our monitoring context. The database is used with RRD4J API: http://code.google.com/p/rrd4j/

We can set maximal duration of the loop (before erase first record), the data precision and minimal duration between 2 records (heart beat). The data base use source<->value couple. A data source can inserts some of values at a certain frequency. In our case, data source corresponding at JVMs and we will record their memory heap.

A class represents a JVM in database is used and get 2 attributes: the name and the color of the curve on the chart.

Util package:

The package has been added for initialization of JAVA code. Indeed, the first synchronization between current state of system and the last record in database need to update data. It's why Util class gets "synchronization" method which update different connections and data record in database. The goal is to get a JVM list in database structure and in model. One get numeric information (memory etc…), other get graphical information (color etc…).



The JVM names in database need to be unique, the compute of data source name is the following then:

Name + 'space' + PID + 'space' + Start_TimeStamp_in_second

Model package:

This package contains application model, i.e. all screensaver data. All attributes and methods are static. It contains system information (system name, total memory, CPU load …) and JVM list in JMX format (name, PID, startTimeStamp…).

The configuration file/script:

The files:

The configuration files get ".desktop" extension. There is one into ProActive Screensaver application. It informs screensaver gnome manager there is a new screensaver. It is placed in /usr/share/applications/screensavers folder and gets ProActive.desktop name. It gets some information like path to screensaver, the name in graphic screensaver manager, etc…

The scripts:

There are three scripts. One of these automates installation of application on the system. It installs application in /usr/bin/ProActiveScreenSaver folder. One other contain uninstall code. The third allow server to launch during system boot. It is placed in /etc/init.d folder, and it starts server as proactive user.

The Screensaver:

A first version of the ProActive Screensaver is written for Gnome.

The screensaver is written in python language and use "gtk" library. The code initializes connection with XSCREENSAVER_WINDOW window which represent screensaver window layer on ubuntu. The code will loop in drawing JAVA picture on screen.

It starts in sending a start signal to the server in named pipe, and listen SEGTERM signal. This signal is sent by system when an event is detected by user. When signal is received, screensaver executable sends a stop signal to server and the JAVA process could be stopped.

IV - Installer:

An installer deploys and can remove all files of an application on a system. On Linux, there is 2 family of installer: Red Hat and Debian.

A composition of a .deb:

- DEBIAN
  - control (configuration script with package name and dependencies)
  - postinst (script to run after deployment, 755 rights)
  - preinst (script to run before deployment, 755 rights)
  - postrm (script to run after removing, 755 rights)
  - prerm (script to run before removing, 755 rights)
- etc
  - init.d
    - ServerProActiveScreenSaver (autostart server)
- usr
  - bin
    - ProActiveScreenSaver (the application)
  - lib
    - xscreensaver
      - ProActive (the screensaver)
  - share
    - applications
      - screensavers
        - ProActive.desktop (the screensaver)

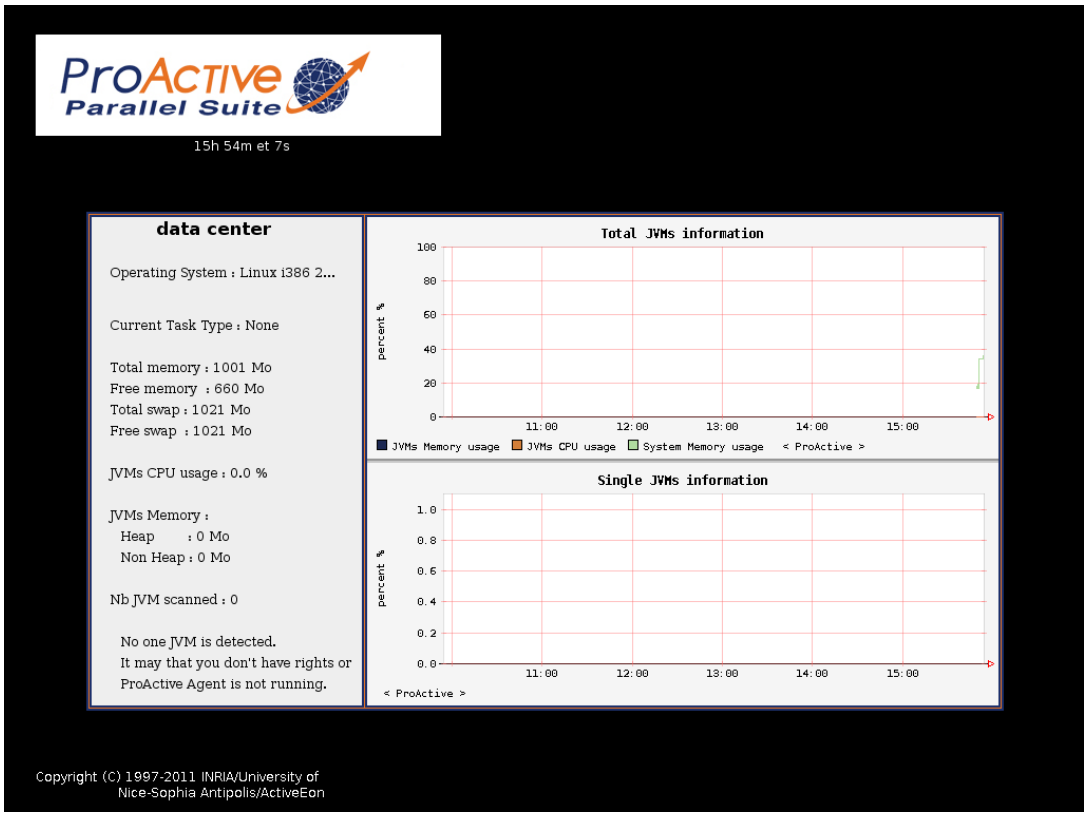
A first approach to create a .deb installer: http://alp.developpez.com/tutoriels/debian/creer-paquet/

The goal is to simulate the deployment of each file of the application in the tree view of the package. So the file in mypackage/etc will automatically be sent in /etc folder.

However, an .rpm feels like: http://fedoraproject.org/wiki/How_to_create_an_RPM_package

V - Annexes:

Final view without ProActive JVM on the system.



Final view with ProActive JVM on the system.