Institut Supérieur d'Informatique, de Modélisation et de leurs Applications

Complexe des Cézeaux
BP 10125
63173 Aubière Cedex
France

CERN Organisation Européenne pour la Recherche Nucléaire

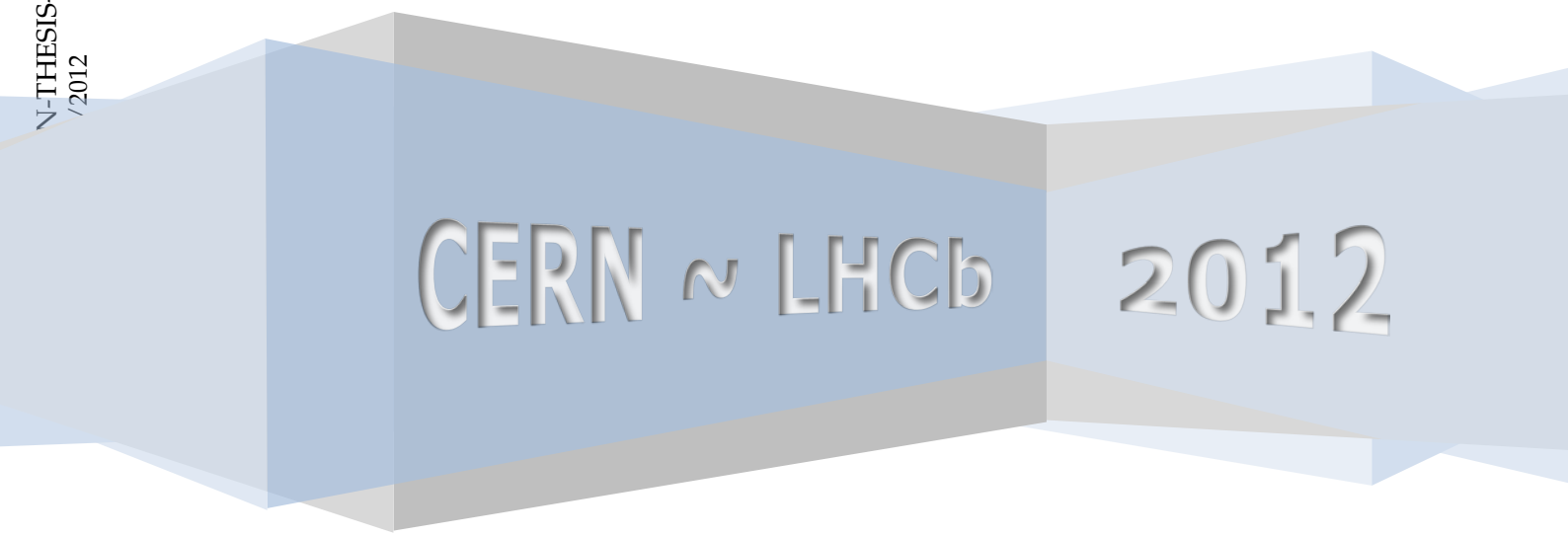
CERN CH-1211
Genève 23

Suisse

SYSTÈME DE CONSTRUCTION AUTOMATISÉ D'ARCHIVES D'INSTALLATION

Rapport de stage – Filière 2 : Génie logiciel

Du Lundi 2 Avril 2012 au Vendredi 31 Août 2012



Présenté par
Damien Vessière

Sous la direction de
Loïc Brarda

Remerciements

Je remercie tout particulièrement Monsieur Loïc Brarda pour son accompagnement tout au long de ce stage.

Ses conseils et ses réponses à mes différentes questions m'ont permis d'avancer sur ce projet passionnant. En conséquence j'ai acquis une expérience enrichissante dans de nombreux domaines de l'ensemble informatique.

Je tiens par ailleurs à remercier également l'ensemble des collaborateurs du LHCB pour leur accueil et pour leurs explications passionnées sur le fonctionnement du CERN.

Je remercie enfin mon tuteur ISIMA, Monsieur Emanuel Menard pour son suivi tout au long du stage et sa venue sur le site.

Résumé

Afin d'optimiser le temps des personnes en charge de la gestion du parc de machines, j'ai réalisé un logiciel en Python capable d'automatiser la création d'archives d'installation sur les différentes architectures utilisées par le LHCb.

Pour mener à bien cette réalisation, j'ai eu accès à différents serveurs et machines virtuelles du CERN ainsi que sur mon propre poste de travail. La manipulation de ces machines a été effectué par la bibliothèque « libvirt » sous python ainsi qu'a de nombreux modules comme par exemple « pyGTK » pour l'interface graphique

Grace à mon logiciel, il est à présent possible de lancer des commandes sur des postes distants afin de créer des archives d'installations pour enfin les rapatrier sur le poste de travail.

Mots-clés :

CERN, LHCb, Machines virtuelles, Python, rpm, rpmbuild, cross-compilation, contrôle à distance.

Abstract

In order to optimize people's time in charge of asset computer management, I created Python software able to automate the creation of install archives on various LHCb's computer in use.

To conduct this realization, I had access to different servers and virtual machines at CERN and on my own workstation. The handling of these machines was performed by the "libvirt" library in python and has many modules like "pyGTK" for the graphical user interface.

With my software it is now possible to run commands on remote systems to create installer and finally bring them back on the workstation.

Keywords:

CERN, LHCb, Virtual Machines, Python, rpm, rpmbuild, cross-compilation, remote control.

Table des matières

Table des figures

Introduction

L'Organisation Européenne pour la Recherche Nucléaire, et plus particulièrement le groupe du « Large Hadron Collider beauty experiment (LHCb) » se penche sur l'opportunité d'intégrer un système de construction automatisé d'archives d'installation afin de réduire le temps requis pour cette opération. En effet, actuellement, de nombreux logiciels et mises à jour sont déployés régulièrement sur les machines utilisées par l'expérience.

La commande actuellement mise en place pour réaliser ces archives sous linux redhat est rpmbuild. Le but est de lancer ces rpmbuild sur les différentes architectures linux distantes et de trouver un moyen de faire la même démarche sous un système d'exploitation Windows. Le logiciel doit donc être évolutif et permettre par la suite d'inclure de nouvelles méthodes de construction à distance comme par exemple une pour le système Macintosh.

Pour parvenir à un tel résultat, il est donc très important de réfléchir dans un premier temps à la structure du programme. En effet la clause de modularité est sans doute la plus importante du projet puisqu'elle permettra la pérennité du programme. Par la suite, il me faut apprendre à maitriser les nombreux éléments qui vont se retrouver au sein du programme : machines virtuelles, interfaces graphique/log, bibliothèque de virtualisation, la commande rpmbuild et les scripts de création de programmes d'installation sous Windows (Nullsoft NSIS[1]).

Contexte du projet

Le CERN est un organisme de recherche fondamentale qui emploie près de 10'000 collaborateurs. Son axe de recherche vise à comprendre l'univers et les particules qui le composent. L'expérience est matérialisée par le « Large Hadron Collider (LHC)» qui accélère des milliers de paquets de protons dans 4 principaux accélérateurs circulaires :

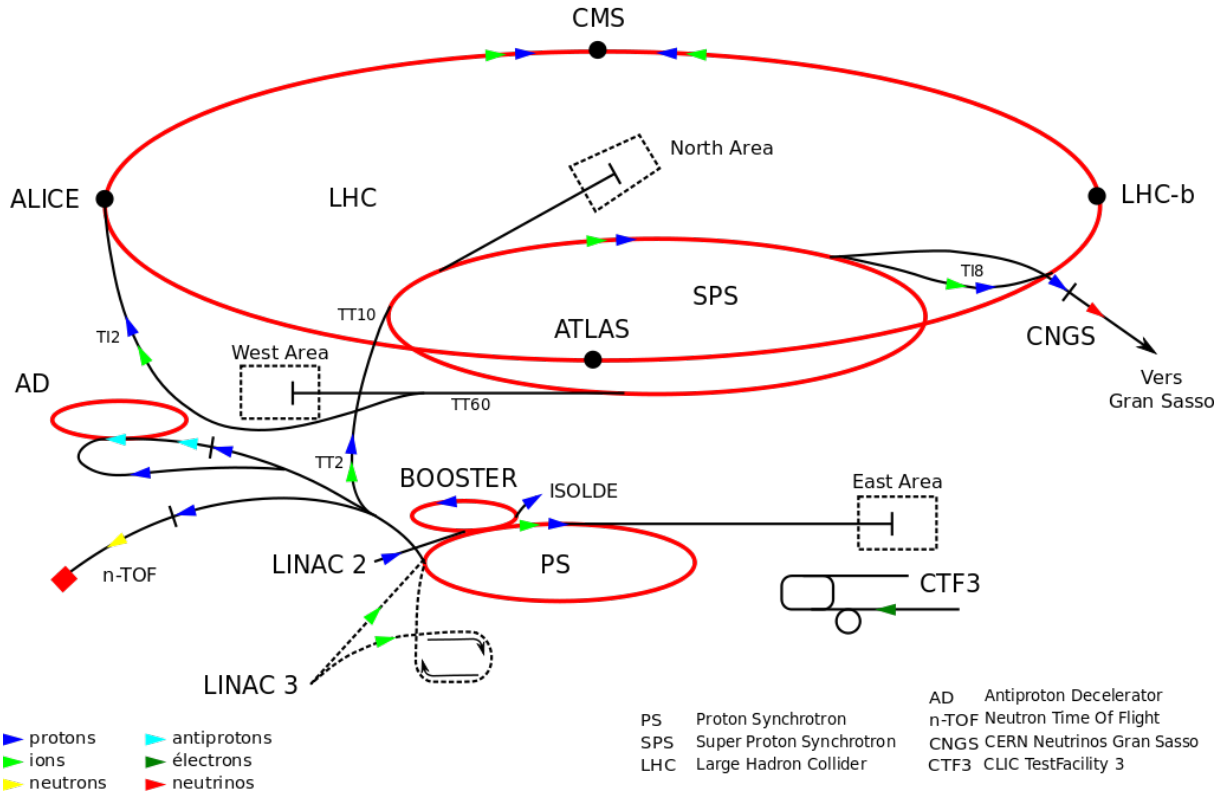


Figure 1 - Les 4 principaux accélérateurs à particule du CERN

Une fois les faisceaux de protons proches de la vitesse de la lumière et chargés d'une quantité énorme d'énergie, grâce à des aimants supraconducteurs placés le long des 30 km de parcours du LHC, ils entrent en collision avec un deuxième faisceau subissant le même phénomène en sens inverse.

Ces collisions sont analysées dans de gigantesques centres, dont le LHCb, qui est constitué d'une équipe d'environ 150 collaborateurs. J'ai eu la chance d'intégrer une de ces équipes durant mon stage.

Au sujet des serveurs, dont la charge revient de filtrer et traiter les données, ils sont analysés par une poignée d'administrateurs système dont dépend M. Loïc Brarda. De nombreuses missions sont assignées à ces administrateurs en vue de maintenir le bon fonctionnement de tous les serveurs et stations de travail.

Des logiciels sont développés par des membres du LHCb ou plus généralement du CERN et sont mis à disposition des administrateurs système pour être déployés. Les nombreux profils de machines entraînent un travail fastidieux de conception d'archives d'installation en vue du déploiement. Ce travail étant répétitif, son automatisation a pu être possible et fera gagner un temps précieux aux administrateurs qui pourront mettre à profit ce gain de temps sur leurs autres missions. C'est donc dans ce contexte que ce sujet de stage m'a été présenté.

Pour mener à bien ce logiciel une première analyse du sujet en début de stage m'a conduit à ce diagramme prévisionnel ci-dessous (Figure 2).

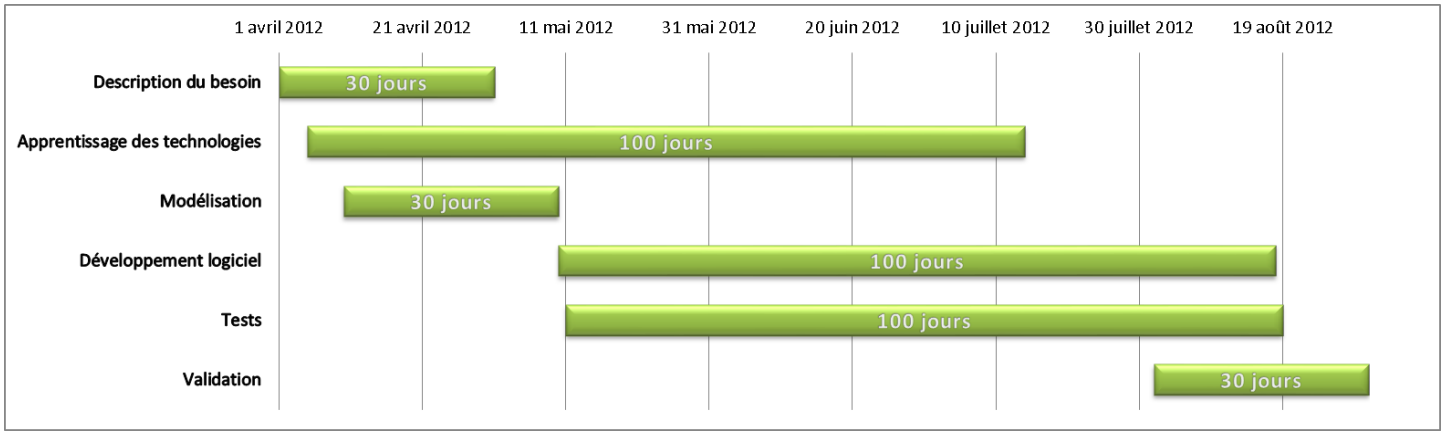


Figure 2 - Diagramme de GANTT Prévisionnel

I. Les archives d'installations

« Archives d'installation » est le terme générique qui définit le fichier nécessaire à l'installation d'un programme. Il porte bien souvent les noms : « setup » ou « installer » et s'exécute la plupart du temps grâce à une succession de fenêtres définissant les options d'installation. On retrouve par exemple :

- L'emplacement du répertoire d'installation.
- Les composants à inclure/exclure.
- La clef d'activation de la licence.
- Etc…

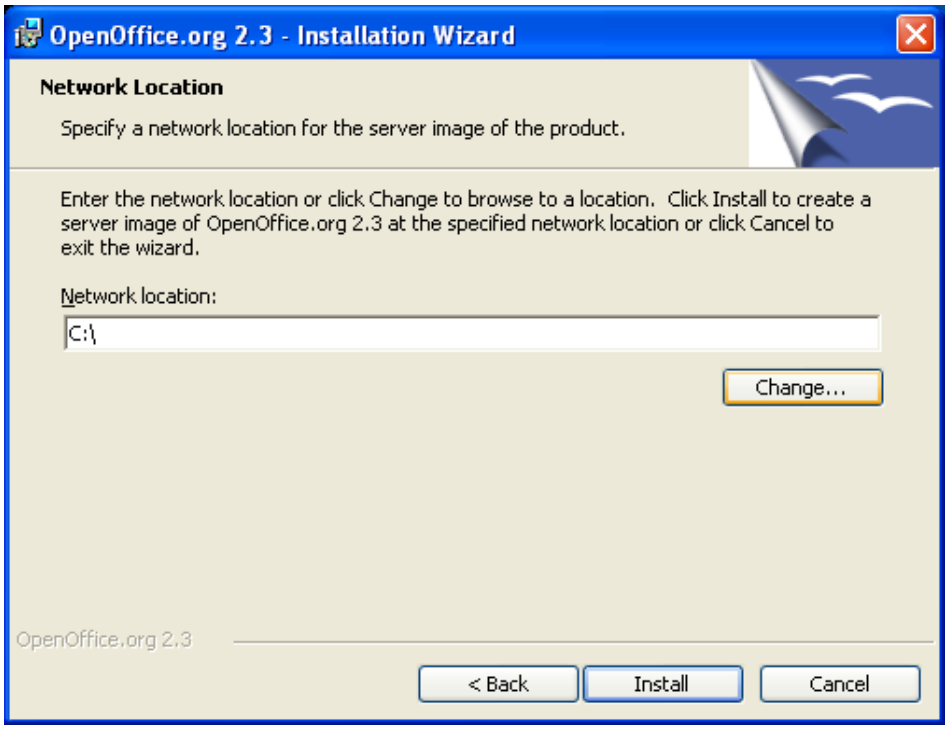


Figure 3 - Interface d'installation

Les divers paramètres rentrés dynamiquement ou extraits d'un fichier de configuration vont copier les différents fichiers et bibliothèques à un emplacement dédié dans le système de fichier.

a. Le RPM

Pour les systèmes Linux, il existe beaucoup de méthodes pour installer un logiciel. Parmi eux figure le « Red Hat Package Manager » RPM qui au fil du temps est devenu un jeu de mot, l'acronyme récursif : « RPM Package Manager ».

Cette méthode d'installation est développée par Red-Hat et sa communauté. C'est un format ouvert doublé d'un logiciel libre de manipulation des fichiers.

Lorsque nous disposons d'une machine Linux gérant les rpm nous pouvons les fabriquer. Pour ce faire il existe la commande « rpmbuild ».

Cette commande lancée nue, nous demande un fichier specfile en paramètre. Ce fichier constitue le cœur de la création d'archives RPM, et permet de programmer les différentes opérations à réaliser. Les premières informations fournies par ce fichier sont les différentes informations consacrées au package.

```
Summary: The DIM communication package
Name: dim
Version: v19r35
URL: http://cern.ch/dim
Source0: %{name}_%{version}.zip
License: GPL
Packager: Loic Brarda & Niko Neufeld
Group: Applications/Communications
BuildRoot: %{_tmppath}/%{name}-root
Prefix: /usr/local
Distribution: 1.RHEL5
```

Figure 4 - Les différentes variables du specfile

Dans l'exemple ci-dessus, nous pouvons observer les informations remarquables de ce package, notamment les champs Source0 et BuilRoot. Ces champs contiennent des directives spécifiques qui seront interprétées à la création de l'archive. Ici, « %{name} » sera directement remplacé par sa valeur définie au-dessus « dim ». Il va de même pour la version. Par contre en ce qui concerne « %{_tmppath} », c'est une directive qui est définie par la configuration du système dans l'un de ces fichiers :

- /usr/lib/rpm/macros
- ~/.rpmmacros
- ...

Ces fichiers permettent de définir des variables propres au système et à l'environnement d'exécution du rpmbuild.

La suite du specfile se divise en différentes catégories mentionnées par un '%' en début de ligne suivi d'un nom tel que :

- Package
- Description
- Prep
- Build
- ...

En fonction des paramètres passés à la commande « rpmbuild », les différentes étapes mentionnées ci-dessus seront réalisées.

```
%prep
/bin/rm -fr %{name}_%{version}
/usr/bin/unzip -o -a %{_topdir}/SOURCES/%{name}_%{version}.zip
#%setup -q # because @#$%@%@@#$# DIM needs the -a option grrrrrrh!!!
cd %{_topdir}/BUILD/%{name}_%{version}
%patch0 -p1

%build
export SHELL=/bin/bash
cd %{name}_%{version}
export DIMDIR=$PWD
export PREFIX="$RPM_BUILD_ROOT"/usr/local
export OS=Linux
```

Figure 5 - Etapes d'un specfile

b. L'exécutable Microsoft

Microsoft dispose d'un système analogue au RPM pour la création d'archives d'installations. C'est un ensemble d'exécutables avec une interface graphique, sorte d'environnement de développement intégré (visant à) se documenter et à tester nos scripts générateurs d'archives d'installation.

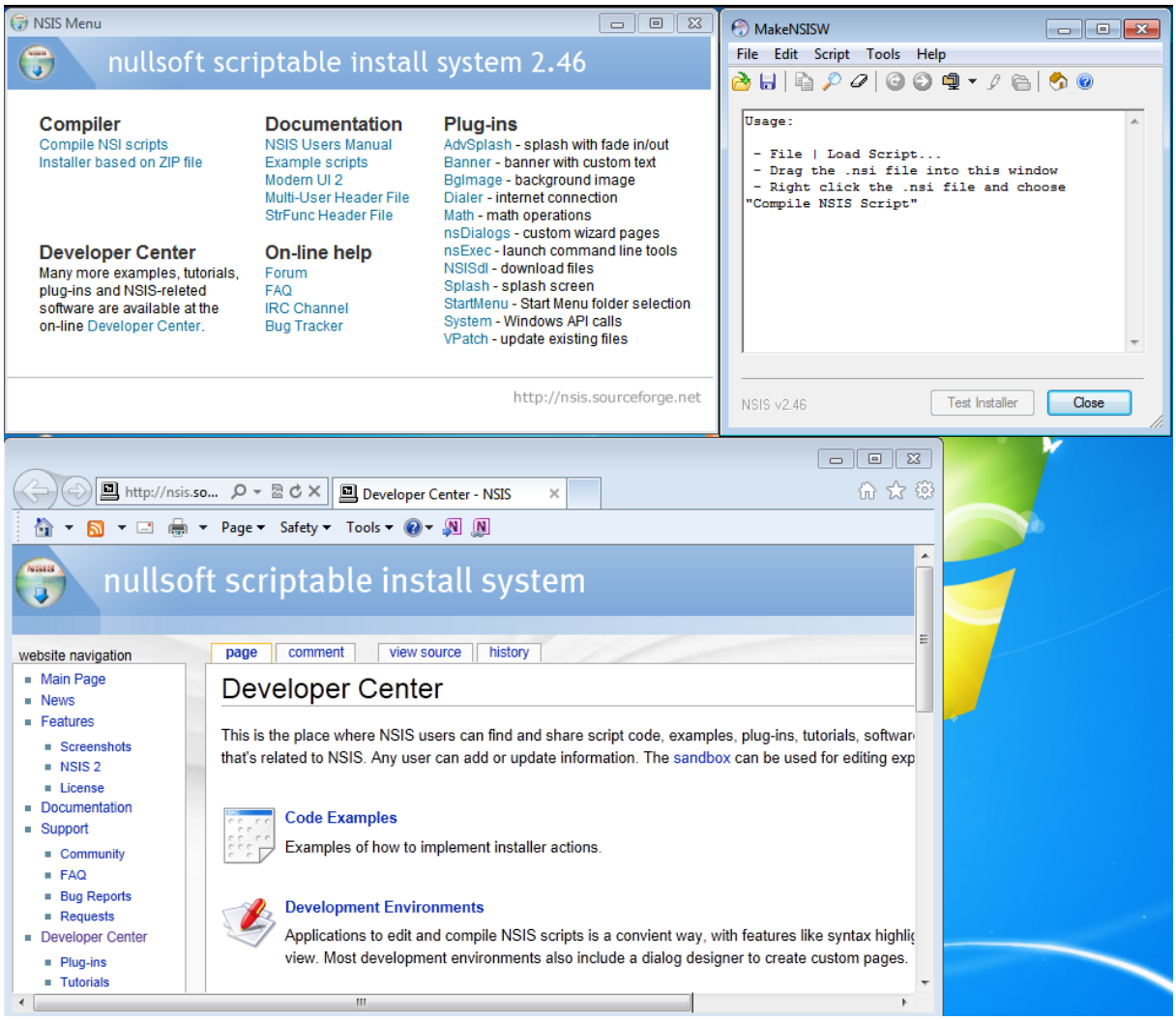


Figure 6 - Nullsoft scriptable install system

Ce logiciel peut être installé sur les différentes architectures Microsoft Windows présentes au CERN. Il se chargera ensuite de générer un exécutable compatible avec le système sur lequel il est exécuté.

Le script de création d'une archive peut être comparé au spec file de RPM. En effet il décrit lui aussi les différentes opérations à effectuer sur le système lors de l'installation du programme.

Le script va aussi pouvoir préciser des opérations propres à Windows telle que la modification de la base des registres afin de définir ou redéfinir les variables de configuration du système et de l'ensemble de ses programmes.

Lorsque le script a été façonné, le programme compile le tout et génère un exécutable prêt pour installer le logiciel voulu. L'exécutable se présente

sous la forme d'une interface module à partir d'un modèle de base que l'on peut deviner sur la figure 7.

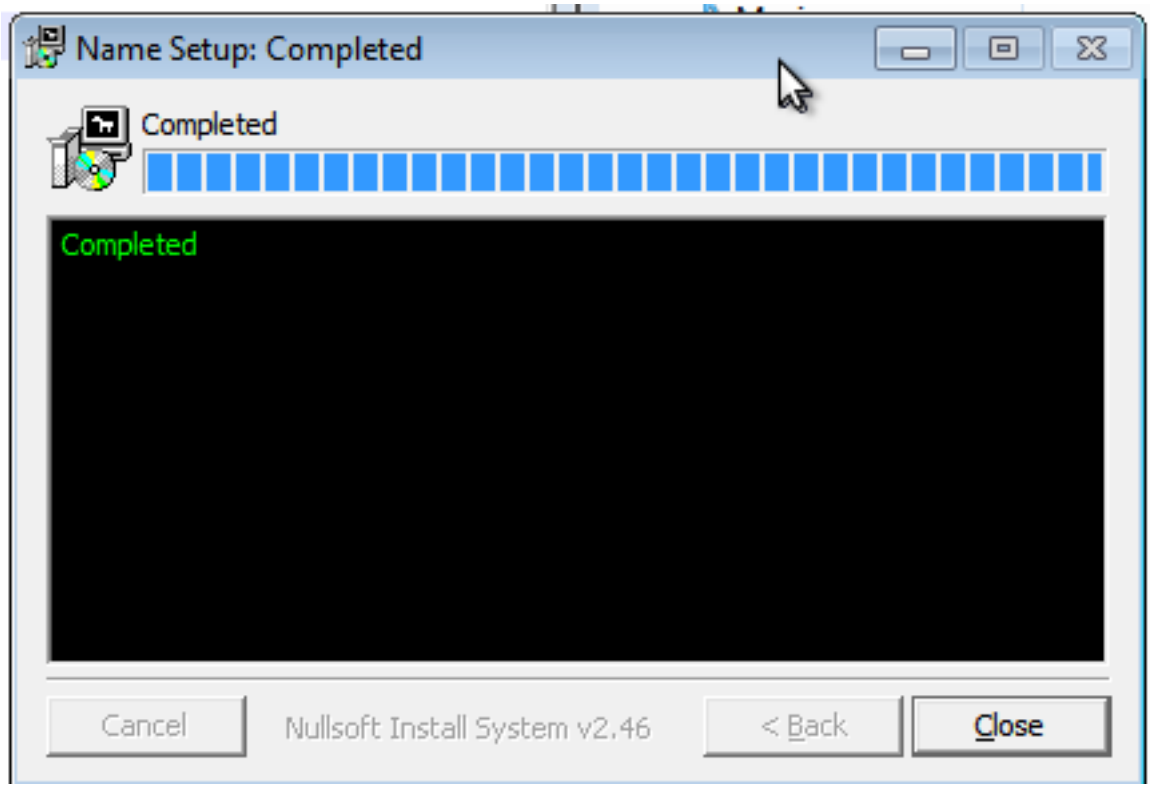


Figure 7 - Exécution de l'installateur généré par NSIS

Une fois les différents paramètres choisis, le technicien va procéder à l'installation du logiciel.

Dans le cadre de mon projet, c'est ce fichier exécutable qui, à terme, doit être rapatrié sur la machine exécutant l'automatisation de création d'archives d'installation.

II. Les machines virtuelles

Afin d'assurer la réussite du projet, il est nécessaire d'utiliser diverses architectures machines. En particulier les architectures de machines présentes au LHCb. Cependant la procédure de création d'archives d'installation, aussi bien sous Linux que sous Windows implique une utilisation d'une bonne quantité de ressources mémoires et processeurs.

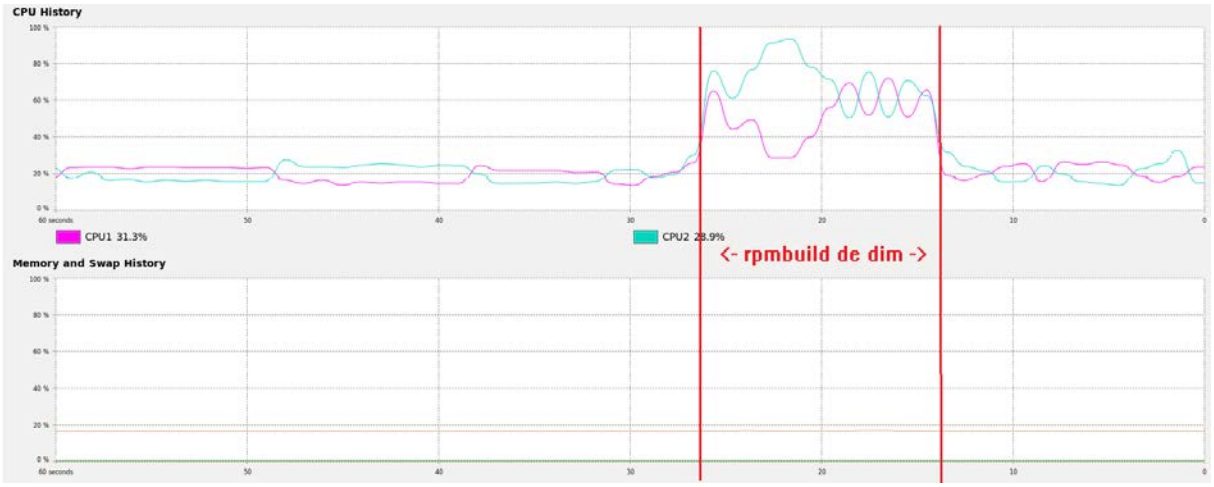


Figure 8 - Utilisation des ressources lors de la commande rpmbuild du paquet "dim"

Il est donc inconcevable de faire cette opération sur des serveurs traitant la quantité énorme de données de l'expérience LHCb. En conséquence, et dans le but d'éviter que le parc de machines n'emphatise pas de cette situation, tout doit être centralisé sur la machine exécutant l'automatisation. Cette centralisation est aussi très intéressante pour la phase de développement et de tests, qui implique des dommages envisageables sur les machines exécutants ces constructions d'archives.

La solution la mieux adaptée à ce cas de figure est donc l'utilisation de machines virtuelles (une par système d'exploitation actif au LHCb).

a. Principe

Les machines virtuelles, c'est l'imbrication d'un système d'exploitation dans un autre.

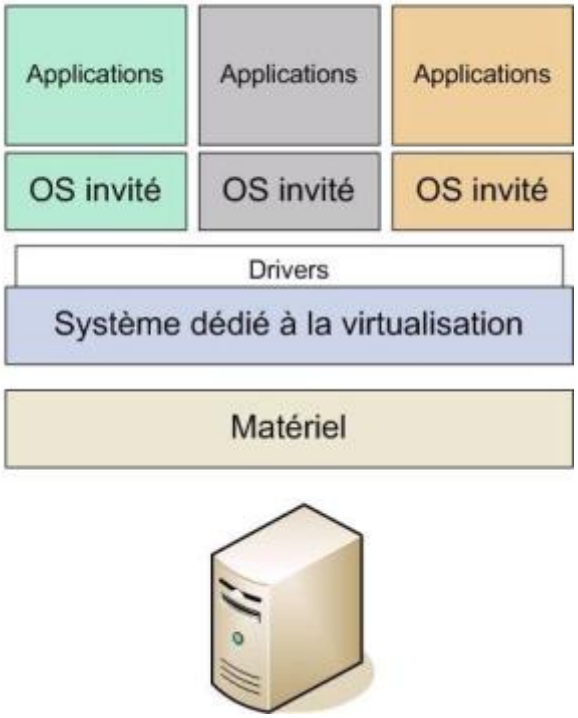


Figure 9 - Schéma de la virtualisation

Pour créer une machine virtuelle, il faut faire croire à celle-ci qu'elle possède du matériel propre. C'est donc en définissant des drivers virtuels, que l'interconnexion entre le matériel réel et celui simulé par l'application hébergeant les machines virtuelles se fait.

La machine virtuelle va ainsi pouvoir fonctionner de manière autonome. Le disque dur sera remplacé par un fichier du système hôte et ce sont donc les drivers virtuels qui se chargeront de convertir les opérations de lecture et d'écriture sur le disque sous forme d'écriture et de lecture binaire du fichier « disque dur ».

Tout l'intérêt d'un tel système réside dans sa capacité à cohabiter sur une même machine hôte avec d'autres systèmes invités. Ainsi l'ensemble des systèmes d'exploitation invités tournent indépendamment les uns des autres. En cas de crash ou d'utilisation excessive de ressources processeur ou mémoire d'une des machines virtuelles, les autres ne seront pas pour autant impactées.

Cette fonctionnalité présente néanmoins un inconvénient : c'est le fait qu'il est nécessaire allouer des quantités non négligeables d'espace disque, de mémoire et de puissance processeur. Ainsi si nous souhaitons faire fonctionner en parallèle quatre machines sous Windows 7, il nous faudra, pour avoir une utilisation confortable, un minimum de 10 Gigas octets de mémoire (2 chacune et 2 pour l'hôte), un processeur 4 cœurs et une taille d'espace disque de 4 fois 30 Giga octets. Les machines virtuelles ont donc besoin de matériel de qualité et fiable afin de pouvoir fonctionner efficacement. Cependant il est à noter que de nos jours, le coût de la mémoire et de la puissance de calcul diminue continuellement. Ce frein, lié au coût, est donc de moins en moins réel. Dans cette perspective, le CERN a d'ailleurs décidé de migrer son parc de serveurs en un ensemble de machines virtuelles dans les années à venir.

b. Kvm & Qemu

KVM est l'acronyme de « Kernel Virtual Machine », pouvant être traduit par Noyau pour machines virtuelles. Ce noyau permet donc l'interfaçage entre les appels système de la machine virtuelle, tels que des demandes d'écriture et de lecture sur le disque. Cette liaison permet donc à la machine virtuelle d'utiliser les composants standards d'un ordinateur via cette couche noyau d'abstraction.

KVM fonctionne depuis quelque temps en collaboration avec Qemu pour l'émulation de la machine virtuelle.

J'ai installé le paquet « qemu-kvm » qui regroupe les deux technologies. Ensuite j'ai téléchargé le paquet « Virtual Machine Manager » qui m'a permis de gérer et configurer les machines virtuelles hébergées en local sur ma station de travail.

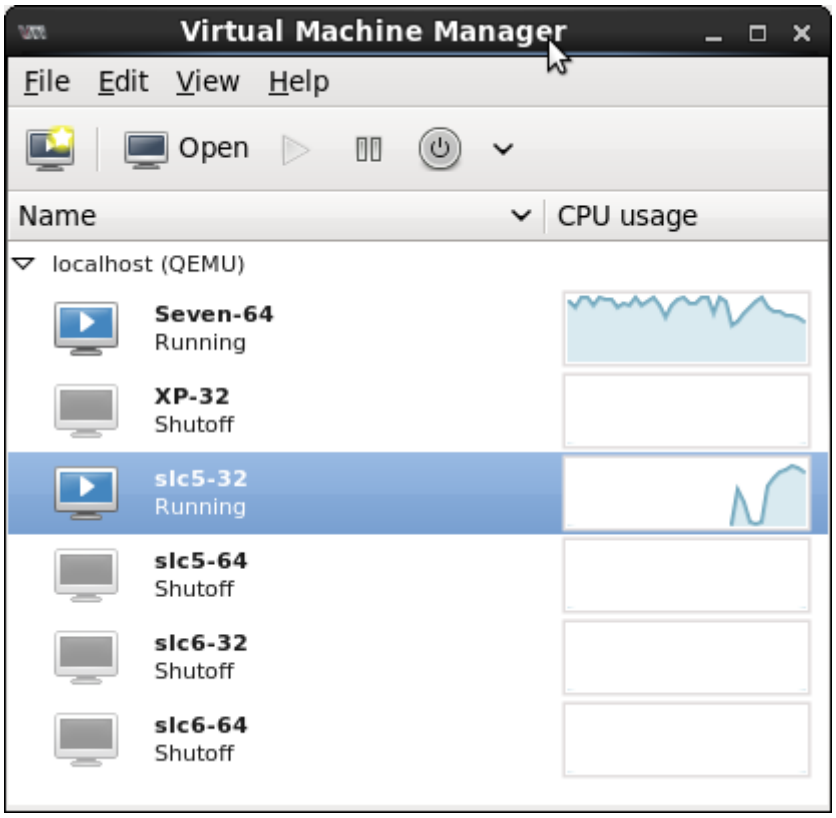


Figure 10 - Le Virtual Machine Manager

Sur le schéma ci-dessus, le manager graphique donne l'état des machines virtuelles présentes sur le PC avec notamment la quantité de CPU utilisée par celles-ci en fonction des quotas alloués à sa création.

Pour créer une machine virtuelle grâce à ce logiciel, il convient d'apporter certaines précisions comme par exemple le moyen utilisé pour installer la machine. Cette option configurera le matériel virtuel en conséquence. Il existe différentes méthodes :

- Par un Cd-Rom. Il est possible de définir un lecteur réel de la machine hôte qui sera « prêté » à la machine virtuelle ou utiliser un fichier ISO (International Organization for Standardization) qui est plus connu sous le nom d'image disque. C'est un fichier qui va contenir les mêmes éléments qu'un CD, un DVD ou encore un Blue Ray. Ce fichier possède des secteurs ainsi que le système de fichier d'un disque.
- Par le réseau. C'est le même principe que l'option du dessus mais en téléchargeant cette image disque directement sur un serveur WEB à définir. C'est le BIOS (« Basic Input Output System » en français : « système élémentaire d'entrée/sortie) qui se charge d'implémenter une telle fonctionnalité.
- Par PXE. Le « Preboot Execution Environment » utilise la même procédure que par le réseau, mais cette fois-ci, c'est un serveur (Serveur PXE) qui se chargera de fournir l'interface à l'utilisateur pour savoir ce qu'il veut installer ou encore installer automatiquement et de manière autonome le système. Au LHCb, le serveur PXE fournit un fichier appelé Kickstart qui définit tout ce qui doit être fait pendant l'installation (définition du matériel, installation de paquet, …). Ce protocole permet une administration efficace d'un grand nombre de machines.
- Importer un fichier disque dur déjà existant.

Dans le cadre de ce projet, des tests primaires ont été effectués en installant les machines via une image disque téléchargée dans les répertoires et mise à disposition au CERN.

Dans un second temps, j'ai préféré utiliser PXE pour avoir l'ensemble des configurations utilisées au LHCB et pour leur ressembler au plus près et utiliser leur même méthode de travail. Pour se faire, il a fallu configurer la carte réseau de la machine hôte afin qu'elle transfère les données à destination des machines virtuelles vers leur carte virtuelle propre. Par défaut, l'élément utilisé pour le réseau des machines virtuelles est la translation d'adresse « network address translation » (NAT). Celle-ci consiste à déterminer des règles afin que le routeur principal gère un réseau secondaire.

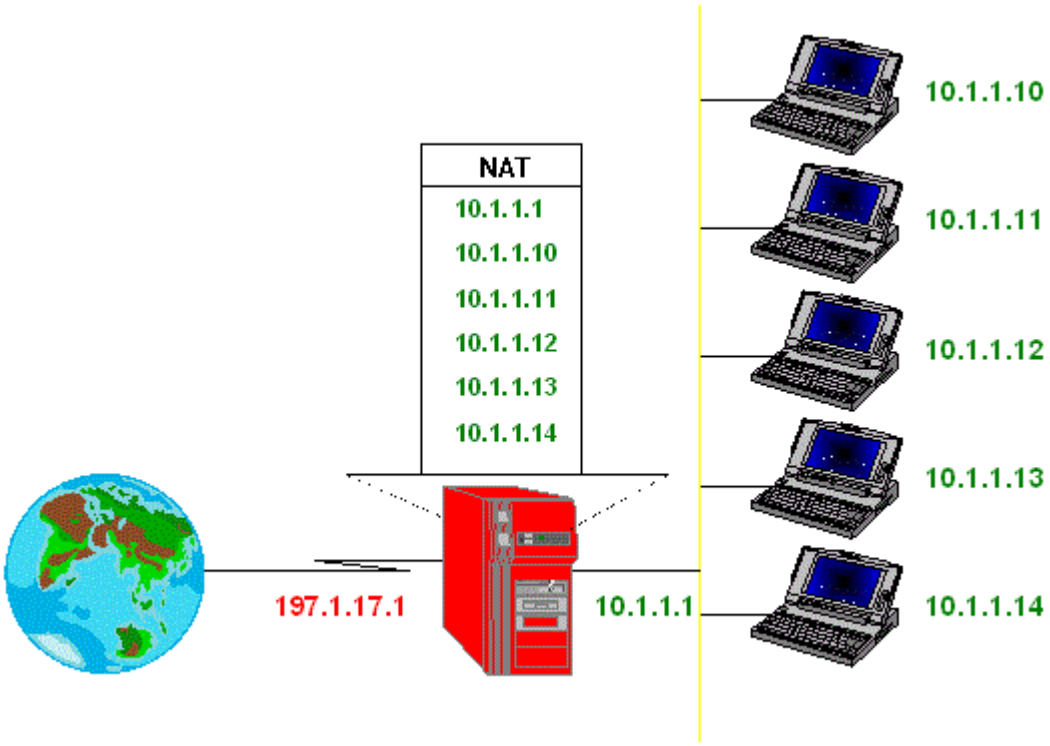


Figure 11 - Protocole NAT

Par contre, ce mode ne convenait pas pour les serveurs PXE, qui ne reconnaissaient qu'un port précis pour une adresse IP enregistrée. Le serveur PXE recevait la trame de la machine virtuelle avec l'adresse mac de celle-ci. Par conséquent, le serveur ne pouvait donc pas identifier la machine puisque son adresse mac n'existait pas dans la base des PC du CERN.

Pour corriger ce problème, l'interface réseau de la machine hôte a été configurée en mode « bridge » vers le réseau des machines virtuelles. Ainsi la carte réelle capturera toutes les trames reçues pour les recopier sur le réseau des machines virtuelles. En enregistrant dans la base de données des PC du CERN les différentes adresses mac virtuelle des machines, le client PXE fonctionne.

III. Python

a. Le langage

Python est un langage de programmation à la fois disponible en orienté objet et en séquentiel. Il est considéré comme un langage facile à apprendre et très rapide à mettre en application. En contrepartie, c'est un langage interprété qui sera donc plus lent à s'exécuter que d'autres langages comme le C ou encore le C++. Pour pallier à ce problème, les librairies les plus utilisées de python sont écrites en C.

Le choix de ce langage a été établi grâce aux éléments suivants :

- C'est un des langages le plus utilisé au CERN. Il me fut fréquent d'entendre parler de ce langage lors des différents échanges avec l'équipe du LHCb. En choisissant ce langage, j'ai pu trouver rapidement des solutions aux phases de bugs et blocages au cours de l'implémentation.

- C'est un langage disposant d'une énorme communauté. Outre le site de la documentation Python qui est clair, simple et précis, il existe de nombreux forums et sites annexes traitant des différents problèmes que peut rencontrer le développeur Python.

- C'est un langage objet. Cette particularité permet une modularité des différentes parties d'un programme. Il est possible d'appliquer le principe des boîtes noires. Comme représenté sur la figure 12, chaque module possède un rôle prédéfini et on peut ignorer les différentes opérations d'une méthode tant qu'on en connaît les tenants et les aboutissants, autrement dit les entrées et les sorties. Dans mon cas il est donc intéressant de pouvoir séparer les différentes parties du programme comme la gestion des machines virtuelles. Il suffira donc d'appeler une méthode « try_to_boot('system_name') » qui se chargera de nombreuses vérifications que l'utilisateur n'aura à sa connaissance que dans le

cas ou l'une des parties rencontre un éventuel problème. C'est un moyen efficace pour des programmes longs et complets disposant de nombreuses fonctionnalités.

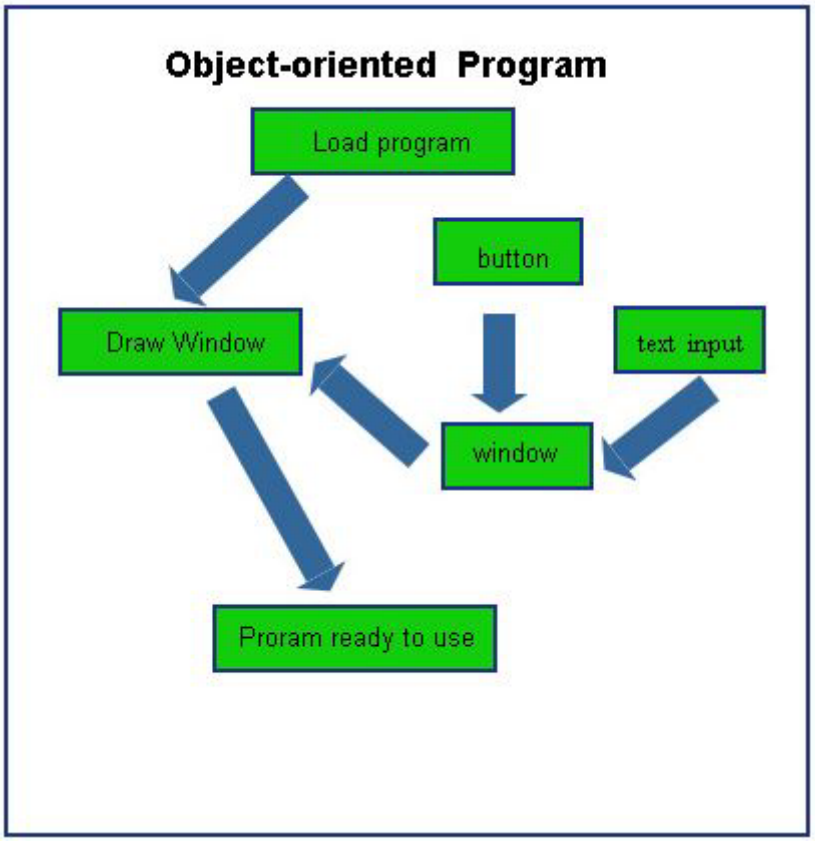


Figure 12 - La programmation orientée objet

- C'est un langage facilement modélisable. Directement lié au fait qu'il s'agit d'un langage objet et donc modulable, il est possible de modéliser un programme avant de l'implémenter. Si la modélisation peut s'avérer être une perte de temps dans un premiers temps, c'est aussi et surtout un moyen de pérenniser un programme. La bonne conception initiale des différents diagrammes permettra des ajouts futurs sans risque d'avoir l'ensemble du programme à remodeler. C'est alors dans ce second temps que nous pourrons observer un gain de temps et une diminution importante des bugs liés aux mises à jour de programmes.

Ces quatre critères déterminants m'ont donc encouragé à choisir ce langage pour mener à bien ce projet. Mais il existe cependant d'autres particularités à ce langage qui peuvent être considérées comme des inconvenants ou sans importance majeure dans mon cas précis.

- C'est un langage faiblement typé. Ceci permet une liberté d'écriture de code importante puisque n'importe quel variable peut héberger n'importe quel type de données En pratique cela peut engendrer quelque problèmes lors de l'écriture de codes relativement longs. Cette liberté implique que rapidement on ne sait plus quel type de données est passé en paramètre à une fonction. Evidemment il est possible de palier à ce problème en documentant son travail ou en nommant explicitement les noms de paramètres. Cette dernière option, bien que simple d'aspect, peut entrainer un code lourd à relire. Par exemple un paramètre de méthode qui aurait pour nom : tableau_entier_identifiant_machines_virtuelles.
  L'autre problème rencontré à cause de ce typage faible est qu'aucun environnement de développement ne permet d'auto complétion puisqu'on ne peut connaitre le type d'objet manipulé. On perd en conséquence un temps important à rechercher la nomenclature exacte des méthodes et fonctions sur la documentation de python ainsi que dans son propre code.

- C'est un langage multi plateforme. Dans le cas de ce projet, l'application sera destinée à fonctionner sur un système d'exploitation Linux. Mais Python est écrit dans l'optique de fonctionner sur l'ensemble des architectures dominant le marché : Linux, Microsoft Windows et Apple Mac OS. Les fonctions sont au maximum adaptées pour avoir le même comportement sur l'ensemble de ces systèmes.

- Il dispose d'un interpréteur en ligne de commande. Sous un système Linux, il est possible d'installer la commande python, elle permet de lancer un interpréteur basique ligne par ligne. Mais c'est aussi la commande qui nous permettra de lancer des programmes complexes par l'intermédiaire de fichiers Python. Cette commande est très pratique pour tester les fonctionnalités de certains modules Python.

b. Libvirt

Python ne possède qu'un nombre restreint d'instructions propres à son langage. Ce qui en fait un langage puissant, c'est la multitude de bibliothèques qui le composent.

On peut trouver une bibliothèque pour les mathématiques, qui contiendra toutes les fonctions mathématiques, de la plus simple à la plus complexe, du cosinus à l'exponentiel en passant par des cosinus hyperboliques par exemple.

Mais dans le cas de ce projet, la bibliothèque qui sera la plus fortement sollicitée est la libvirt, traduisible par bibliothèque virtuelle. C'est une librairie puissante et très modulable permettant de gérer des machines virtuelles.

Une fois inclus dans notre script python, il suffit de préciser de quel type est notre gestionnaire de machines virtuelles afin de profiter d'un ensemble de fonctions pour les manipuler. Pour stipuler ce choix de manageur, c'est l'instruction :

```
conn = libvirt.openReadOnly("qemu:///system")
```

Figure 13 - Code Python pour créer le manageur de connexions virtuelles

Cette fonction retourne un objet nommé dans cet exemple « conn ». Il possède de nombreux attributs donnant un ensemble d'informations sur ce manageur de machines virtuelles mais aussi de nombreuses méthodes pour les manipuler. On en dénombre 110, dont en voici quelques-unes grâce à la directive dir() propre au langage, permettant de lister l'ensemble des éléments (attributs et méthodes) :

```
['__del__',
 '__doc__',
 '__init__',
 '__module__',
 '_dispatchDomainEventCallbacks',
 '_dispatchDomainEventDiskChangeCallback',
 '_dispatchDomainEventGenericCallback',
 [...]
 'nwfilterDefineXML',
 'nwfilterLookupByName',
 'nwfilterLookupByUUID',
 'nwfilterLookupByUUIDString',
 'restore',
 'restoreFlags',
 'saveImageDefineXML',
 'saveImageGetXMLDesc',
 'secretDefineXML',
 'secretLookupByUUID',
 'secretLookupByUUIDString',
 'secretLookupByUsage',
 'setKeepAlive',
 'storagePoolCreateXML',
 'storagePoolDefineXML',
 'storagePoolLookupByName',
 'storagePoolLookupByUUID',
 'storagePoolLookupByUUIDString',
 'storageVolLookupByKey',
 'storageVolLookupByPath',
 'suspendForDuration',
 'virConnGetLastError',
 'virConnResetLastError']
```

Figure 14 - Les différentes méthodes applicables à l'objet gérées par la libvirt

Avec ces nombreux opérateurs, il est possible d'obtenir des informations sur les machines virtuelles administrées par QEMU. Cependant dans le cadre de mon projet, il me faudra utiliser une directive quelque peu différente pour pouvoir démarrer, arrêter et accéder à un maximum d'informations sur les machines virtuelles.

Cette nouvelle directive moins restreinte est :

```
conn = libvirt.open("qemu:///system")
```

Figure 15 - Créer le manageur de connexions virtuelles sans restrictions

Cependant, elle nécessite de rentrer le mot de passe de l'administrateur. C'est donc une interactivité qui ne peut être admise au sein de ce programme.

Il est possible de contourner ce problème en manipulant un des fichiers de configuration du système.

Il nous faut donc créer ou ouvrir le fichier de configuration de libvirt afin d'y ajouter l'utilisateur autorisé à manipuler les machines.

Pour simplifier la suite du projet et correspondre à la programmation par module, j'ai créé un ensemble de fonctions afin de permettre de manipuler les machines virtuelles sans traiter des cas particuliers.

Le tout est regroupé dans un fichier nommé « virtual manager ». Les deux principales fonctions présentes sont :

- `start_local_vm`
- `stop_local_vm`

« start-local-vm » permet, comme son nom l'indique, de démarrer une machine virtuelle, située sur le poste de travail.

Il faut passer en paramètre un descriptif de la machine contenant principalement son adresse mac et son nom afin que le gestionnaire puisse donner les informations nécessaires à « libvirt » pour rechercher cette machine afin de la manipuler.

Il est également nécessaire d'avoir un pointeur vers l'interface graphique de contrôle. Celle-ci peut être une série de logs au format texte dans un fichier, ou dans une console, ou un format graphique conçu spécialement pour le développement de l'application et afin de mieux comprendre les étapes du programme.

Un identifiant graphique de système est aussi requis pour que la différente interface graphique possible identifie le système dont il est question en cas d'erreur ou de notification.

Le dernier paramètre obligatoire se nomme « asFastAsPossible » traduit littéralement par aussi vite que possible. C'est une variable binaire qui prend la valeur vrai ou faux et qui détermine si la fonction doit être exécutée ou non. En effet si l'utilisateur décide d'arrêter le programme et qu'un thread de celui-ci est en train de démarrer la machine virtuelle, il faut pouvoir lui notifier un arrêt de la procédure.

S'en suivent deux paramètres facultatifs : préciser si on souhaite que des statistiques soient produites au court de ce démarrage. Un fichier sera alors créé ou complété du nom de la machine, suivi de son temps de démarrage. Un nombre variable définissable sous forme de constantes dans ce fichier déterminera le nombre de mesures maximum à prendre. Ceci permet au programme de devenir de plus en plus « intelligent » au fil du temps en précisant sur la base des enregistrements précédents quelle durée il reste pour le démarrage de la machine.

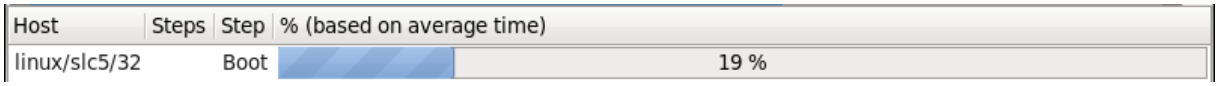


Figure 16 - Etat d'avancement du démarrage d'une machine virtuelle

Le second paramètre facultatif de cette méthode permet la récursivité de la fonction. En effet la librairie virtuelle peut nous indiquer l'état de la machine virtuelle mais en aucun cas l'état du système d'exploitation hébergé dessus. Pour ce faire, une fonction « attente de démarrage » a été créée dans l'optique d'interroger à intervalles réguliers le système d'exploitation invité, seule la réponse de ce dernier pourra déterminer si la machine a terminé son démarrage.

Pour démarrer une machine via Python, rien de plus simple, il suffit d'appeler la fonction « create » sur l'objet « domain » qui représente l'ensemble des éléments disponibles pour une machine virtuelle.

La fonction boot est donc écrite simplement :

```
def _boot(system, window, sysId):
domain = conn.lookupByName(system.host)
domain.create()
```

Figure 17 - Code de démarrage d'une machine virtuelle

On récupère simplement le « domain » grâce au nom de la machine puis on la lance.

Toute la difficulté réside dans la fonction chargée d'attendre la fin de ce démarrage. Son algorithme de principe est le suivant :

- Récupérer le « domain »
- Récupérer l'adresse mac grâce à l'info extraite via la « libvirt »
- Récupérer grâce au module de statistique précédemment créé, le temps moyen de démarrage de cette machine.
- Notification utilisateur du démarrage de cette machine.
- Enregistrer le temps actuel (en millisecondes depuis 1970)
- Récupérer l'adresse IP depuis le module conçu pour l'occasion (depuis l'adresse mac ou depuis le nom d'hôte de la machine référencée dans la base de données des périphériques du CERN)
- Tant que le ping[2] n'aboutit pas et que le temps limite n'est pas atteint :
  - Attendre une seconde
  - Notifier l'utilisateur qu'une seconde est écoulée (pour la barre de progression)
- Si le temps est écoulé et que le ping n'a pas abouti :
  - Notifier l'utilisateur
  - Remonter l'existence d'une erreur (Faux)
- Sinon
  - Notifier l'utilisateur
  - Remonter que la machine est prête pour la suite (Vrai)

Ce dernier paramètre nous permet alors de recommencer l'opération plusieurs fois en précisant le nombre d'essais maximums et en retranchant un à ce nombre à chaque échec.

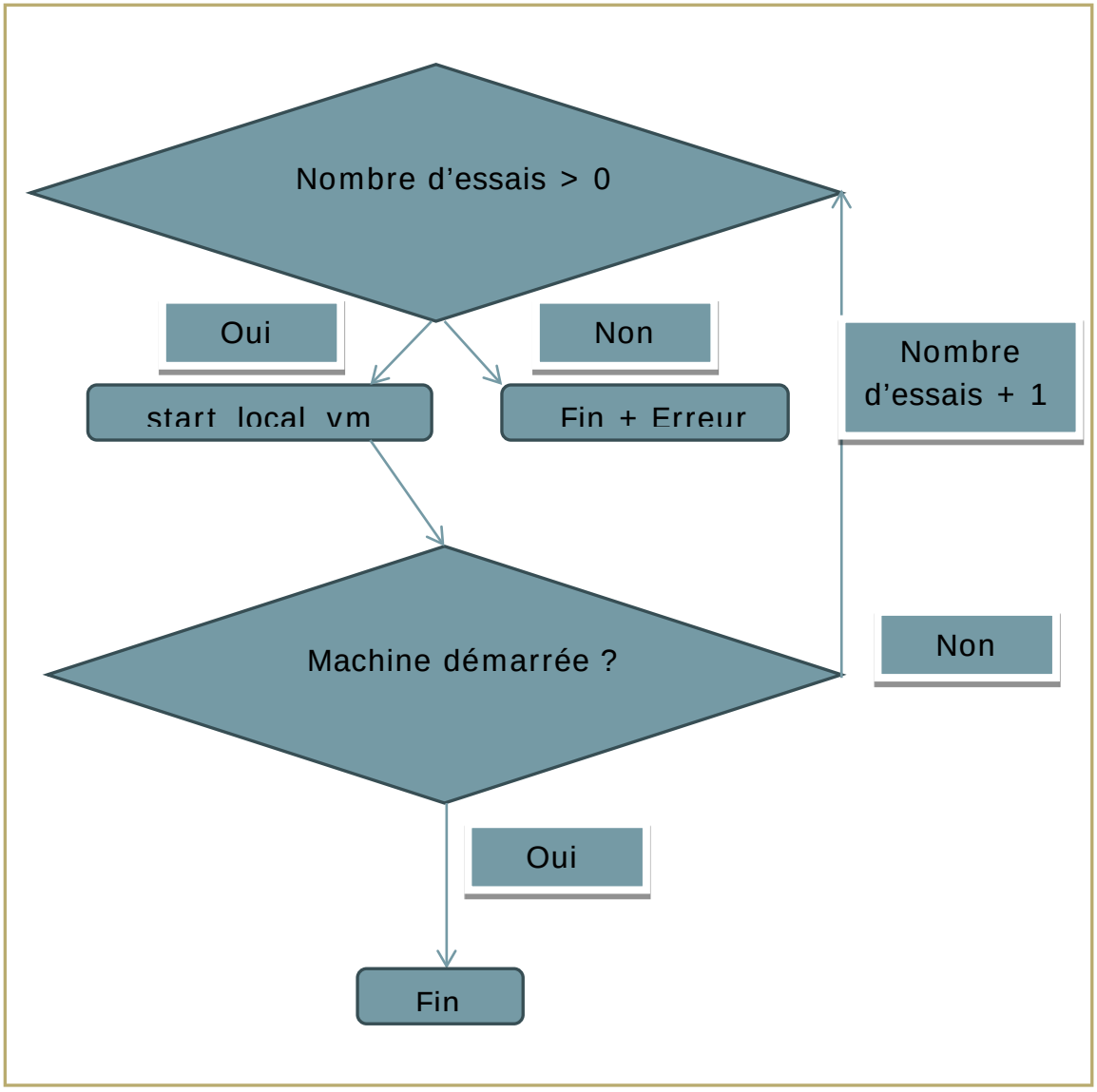


Figure 18 - Schéma récursif de démarrage d'une machine virtuelle

C’est le même principe pour « stop-local-vm » que pour démarrer la machine, dans le sens où il est très simple d’envoyer le signal d’extinction, mais par la suite il est très compliqué de savoir quand la machine va elle-même rendre l’ensemble des ressources qu’elle a réservé pour son fonctionnement.

Dans un souci de compréhension de la fonction, les mêmes paramètres sont requis lors de son appel.

Le principe de cette fonction est structuré sur le même modèle que la fonction de démarrage des machines virtuelles :

- Notifier l’utilisateur de l’arrêt de la machine
- Si la machine est déjà en statut inactif, on sort du programme
- Récupérer, grâce au module de statistique, le temps moyen de démarrage de cette machine.
- Envoi du message d’extinction grâce à la méthode de l’objet « domain » : « shutdown »
- Tant que la machine virtuelle n’a pas le statut « inactif », et que le temps maximum n’est pas dépassé.
    - Si la fermeture du programme principal est demandée, sortir de la boucle.
    - Notifier l’utilisateur qu’une seconde est écoulée (pour la barre de progression)
- Si le « domain » est encore actif
    - Notifier l’utilisateur que l’arrêt va être forcé.
    - Forcer l’arrêt de la machine et rendre les ressources de force (équivalent à couper le courant)
- Sinon
    - Notifier l’utilisateur de la réussite de l’arrêt de la machine.
    - Sauvegarder le temps d’arrêt de cette machine pour les futures exécutions du programme.

Il existe dans ce module de diverses autres fonctions telles que suspendre un système pour rendre les ressources processeurs (mise en pause).

IV. Assemblage des modules et automatisation

Ce projet comporte de nombreux aspects informatiques et de l'administration système. L'étape la plus compliquée est de mettre en cohabitation tous ces éléments.

Pour ce faire, nous avons la modélisation. Celle-ci permet une programmation à la méthode des jeux de construction. En effet il faut prévoir quelle sera l'interface des différents éléments composant le programme pour pouvoir modifier aisément le code sur de futures retouches.

J'ai eu besoin d'une notion importante de la modélisation pour mener à bien ce projet : l'héritage. En effet pour piloter une machine virtuelle à distance, on utilise toujours le même modèle d'algorithme :

- Démarrage
- Initialisation
- Connexion
- Transfert des données générées
- Arrêt

Ces quelques étapes clés étant présentes quelque soit le système d'exploitation contrôlé, il est donc logique qu'il représente l'objet de base des constructeurs d'archives d'installation à distance.

Ensuite un deuxième niveau apparait séparant les systèmes d'exploitation des deux distributions principales en deux classes de constructions distinctes présentant les mêmes méthodes que la classe de base mais en redéfinissant tout ou partie du code des méthodes.

Ensuite nous avons du code encore plus spécialisé qui peut être défini pour une distribution spéciale du système d'exploitation qui demande un traitement particulier.

Cette hiérarchie de classe peut se définir comme sur le schéma suivant :

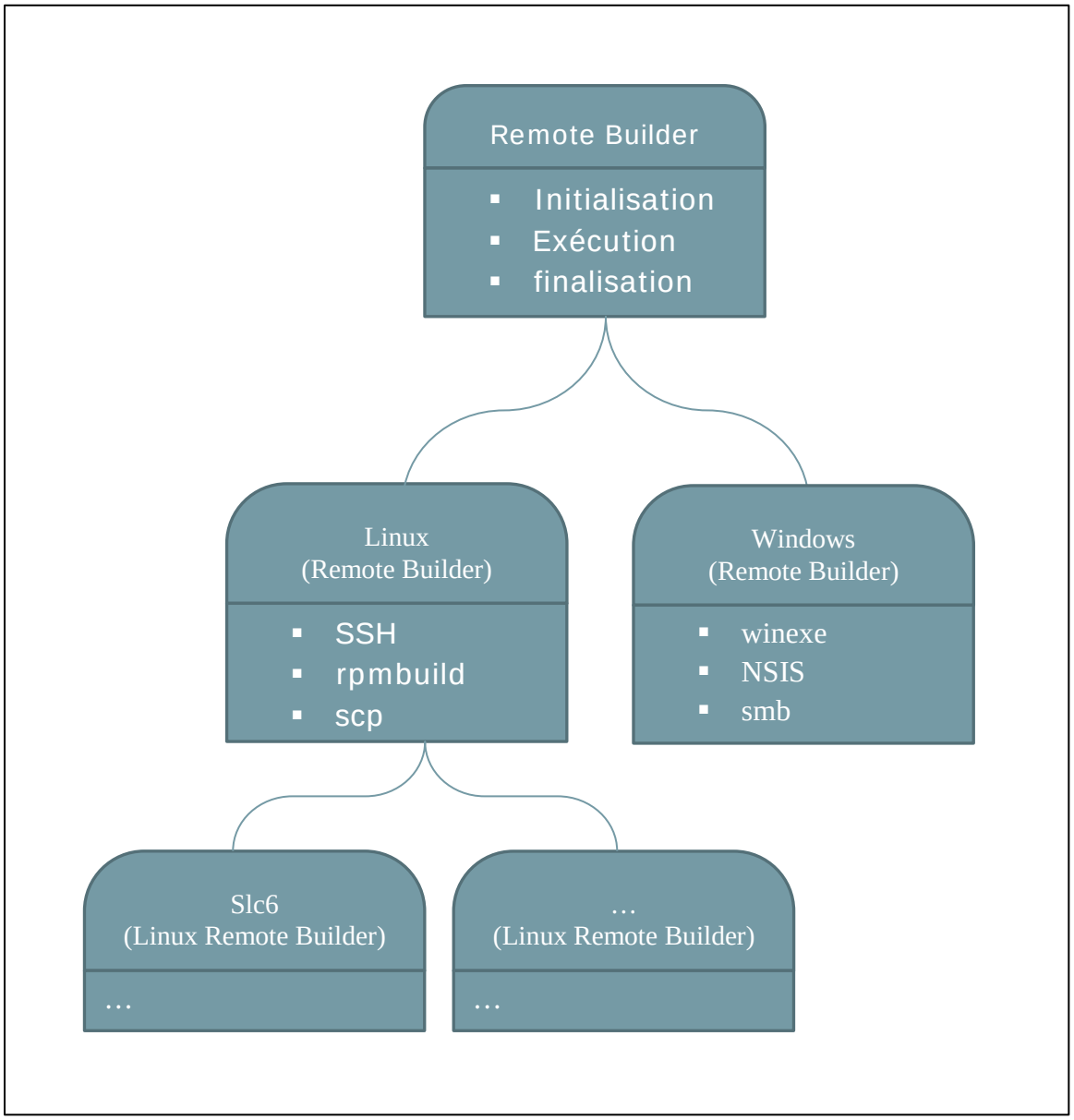


Figure 19 - Hiérarchie des constructeurs à distance

On remarque que la classe de base ne possède que trois méthodes de base. Cela permet de s'adapter à la fois aux machines virtuelles et aux machines physiques.

Un simple booléen dans la description de la machine à contrôler permet de savoir si c'est une machine virtuelle, et un autre permet de savoir s'il faut l'arrêter ou simplement la mettre en pause. C'est l'initialisation qui va dans un premier temps tester ce booléen pour savoir s'il faut lancer la procédure de démarrage de machine virtuelle. Dans la finalisation le même test est effectué pour savoir si la machine virtuelle doit être stoppée ou non. Cela permettra à l'avenir de tourner sur des machines virtuelles de serveur, qui restent en fonctionnement plusieurs jours à la suite.

Le deuxième niveau de ce schéma explicite les technologies mises en œuvre dans les deux cas de figure.

Il y a pour Linux et Windows respectivement :

- SSH et Winexe

Ce sont les protocoles de connexion aux machines distantes (virtuelles ou non) qui sont appliqués pour communiquer et donner les ordres de construction au poste de travail client.

Winexe a été développé sous licence GPL3 le 30 juin 2006 et présente une bonne solution pour exécuter diverses commandes Windows depuis un client Linux.

SSH (*Secure Shell*) quant à lui, est un protocole développé depuis bien plus longtemps (1995) et offre la meilleure optimisation pour se connecter sur un serveur Linux distant. Il est intégré par défaut dans la quasi-totalité des distributions Linux, à la fois au format serveur et au format client.

- Rpmbuild et NSIS

Comme présenté précédemment, ils permettent donc de réaliser l'archive d'installations.

- Scp et smb

Ce sont les protocoles utilisés pour la finalisation, ils permettent l'acheminement de fichiers au travers du réseau.

scp (*Secure copy*), du nom de sa commande qui comme ssh est une commande intégrée depuis longtemps dans la majeure des distributions Linux. Son avantage est qu'elle présente un niveau de sécurité excellent.

Smb (*Samba*) est aussi un protocole de transfert de fichiers mais est bien moins souvent présent par défaut sur les distributions. Cependant il a l'avantage d'être présent à la fois sur Linux et sur Windows, ce qui en fait l'intermédiaire de choix pour rapatrier l'archive d'installation sur la machine maître lors de la finalisation.

a. Méthode

Tous les modules du programme sont assemblés entre eux pour être appelés via leur interface.

Pour les constructeurs d'archives à distance appelés *Remote Builders,* ils ont d'abord été instanciés sous forme de classe avec les diverses méthodes à appeler. Mais après réflexion, il a paru plus judicieux de faire hériter tous les *Remote Builders* des Threads.

Les threads : notion informatique de plus en plus en vogue de nos jours puisqu'ils sont l'élément de base de la programmation parallèle. Ils disposent d'un code à exécuter séquentiellement mais sont susceptibles d'être interrompus et repris à tout moment. L'autre caractéristique d'un thread est qu'il peut recevoir des messages et notifications d'autres threads à n'importe quel moment.

Il est donc plus difficile de programmer une application via des threads puisqu'il faut gérer les interruptions. Mais les threads permettent dans bien des cas un gain énorme en performance, bien souvent multiplié par le nombre de threads puisque un thread en attente d'une ressource quelconque ne bloquera plus le reste du programme. De plus les threads permettent une bonne lisibilité du code par rapport à un seul programme essayant tant bien que mal de gérer tous les cas particuliers.

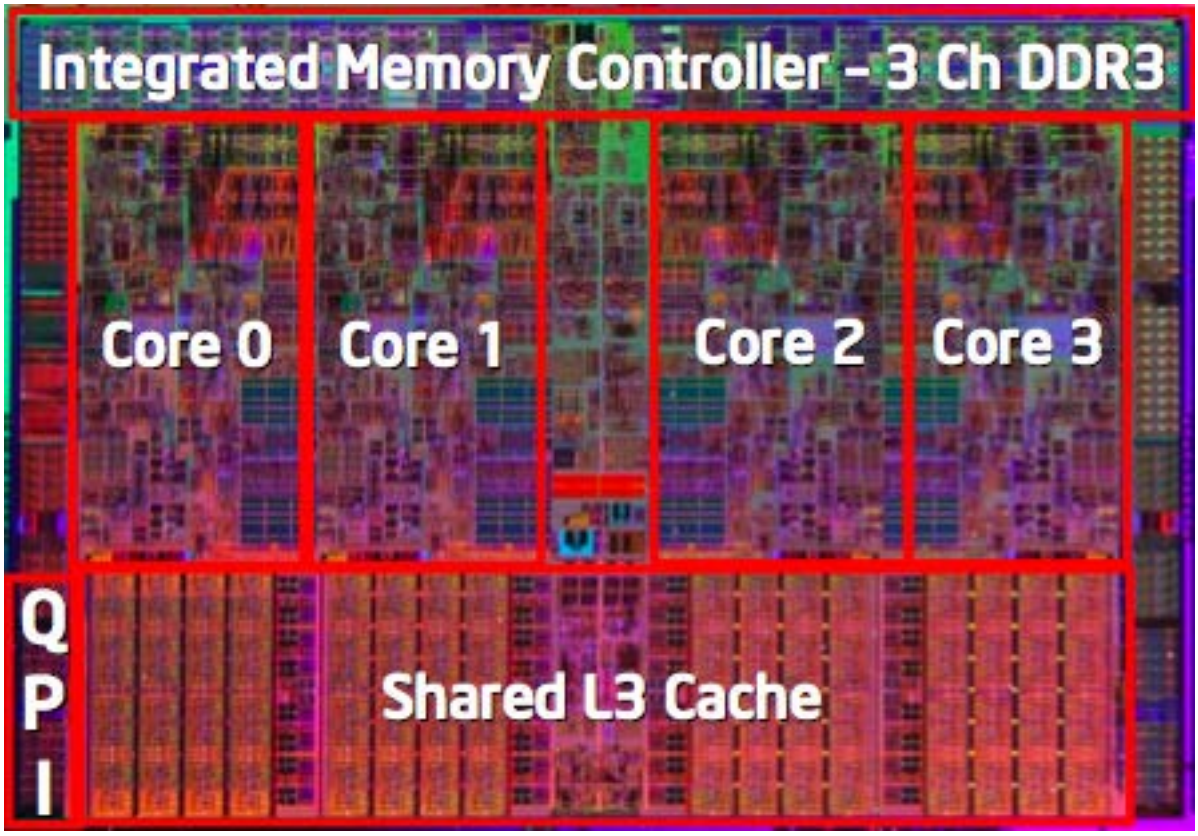


Figure 20 - Processeur 4 cœurs

Sur cette figure est exposé un processeur récent : on note la présence de quatre cœurs et d'une mémoire partagée à accès extrêmement rapide pour le code et les données du programme en cours.

Aujourd'hui, ces processeurs modernes disposent d'une capacité de deux threads par cœur. En effet un procédé permet de changer quasi instantanément de contexte d'exécution pour émuler un fonctionnement parallélisé de ses deux threads. Cette technologie se nomme hyperthreading.

Il est donc opportun désormais de programmer en tenant compte de cette technologie et ainsi de maximiser la parallélisassions ; d'autant que la tendance continue de s'accroitre dans cette direction.

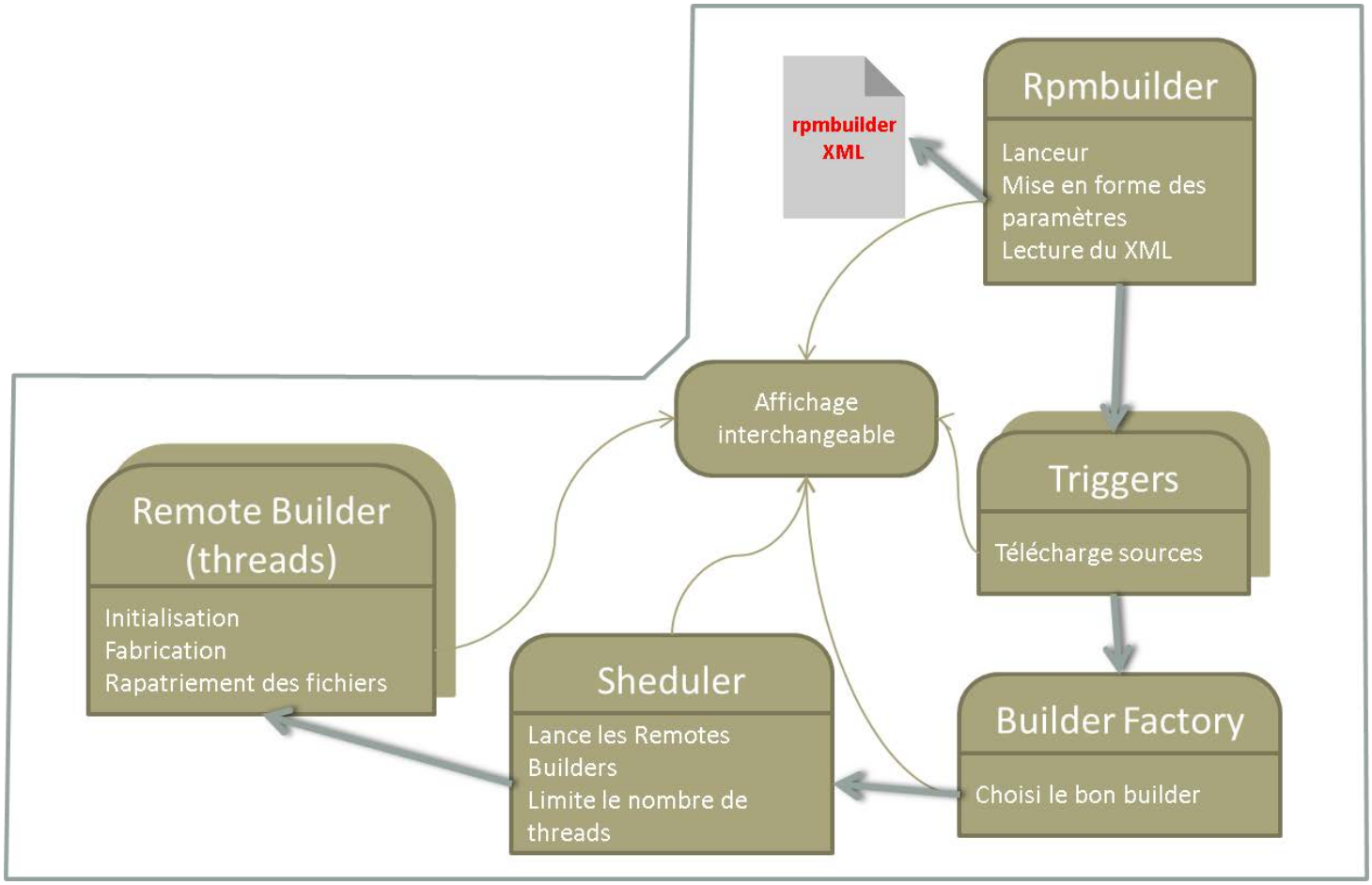


Figure 21 - Modélisation du programme "rpmbuilder"

Toute la méthode de développement du programme final repose sur ce schéma de modélisation.

Dans un premier temps il y a eu deux autres versions du programme qui ont permis d'aboutir à ce troisième mieux structuré.

Le tout premier programme m'a permis d'appréhender les diverses facettes du langage ainsi que de me familiariser avec la bibliothèque « libvirt ». C'était un programme entièrement séquentiel qui se chargeait de démarrer et de se connecter sur une machine Linux.

Dans un second temps, après quelque semaines de travail sur le projet, le logiciel a été repensé de façon à exploiter le modèle objet et modulaire de Python. Le programme se découpait en divers modules et classes de manière à interfacer les éléments du code. La notion de script associée à une machine et à un OS a commencé à être abordée dans cette version.

C'est aussi les débuts de la découverte du module GTK de python permettant de gérer et concevoir des interfaces graphiques.

Enfin ce troisième et dernier programme qui regroupe de nombreux modules du deuxième mais adaptés au principe des threads. L'interface graphique est cette fois un simple contrôleur d'exécution qui permet de mieux appréhender les exécutions des divers threads sur les manipulations des machines virtuelles.

Cette notion de configuration est déportée dans un fichier édité au format XML.

i. Le fichier de configuration XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
Default user = root
Default pass = 123456
-->
<racine>

  <packages>

    <package name="dim">
      <trigger script="web_check.py">
        <param name="url"          value="http://dim.web.cern.ch/dim/dim_unix.html"/>
        <param name="tag_contain" value="dim.zip"/>
        <target name="linux" />
        <!-- You can use a special builder inherit from BaseRemoteBuilder -->
        <!--
        <target name="windows" />
        <target name="linux" />
        <target name="linux/slc6" />
        <target name="linux/slc5/32" builder="slc" />
        <target name="windows/xp" />
        -->
      </trigger>
      <trigger script="script_innexistant.py"></trigger>
    </package>
```

Figure 22 - Configuration XML des paquets

Sur cette figure, nous avons le début du fichier XML qui nous permet de paramétrer le programme.

Comme on peut le voir dans le premier commentaire, caractérisé par les balises ‘<!--’ et ‘-->’, le mot de passe par défaut est 123456 et le programme se connecte en tant qu’administrateur. Cependant il sera possible de changer ses valeurs dans la suite de la configuration.

S’en suivent les diffèrent paquets qui vont être traités par le programme principale.

Chacun des paquets à traiter contient des « triggers ». Ce sont des modules que l’on peut greffer sur le programme principal. Un répertoire triggers va accueillir le script donné dans l’option « script= ». Ce trigger est chargé de préparer les données pour les threads communiquant avec les machines virtuelles.

Pour l’instant il n’existe qu’un seul de ces plugin[3]s, son nom est « webcheck », il permet :

- D’analyser une page web.
- De rechercher une balise contenant un certain texte passé en paramètre du trigger.
- D’en extraire la version du paquet, contenu dans le texte de la balise via une expression régulière[4] passée aussi en paramètre.
- De comparer cette version avec celles déjà présentes dans le répertoire prévu pour recevoir les rpm.
- Si elles n’existent pas, de déléguer les constructions d’archives d’installation aux différents threads de contrôle des machines à distance.
- De rendre la main au programme principal pour l’exécution des triggers suivants et le traitement des paquets restants.

Chaque paramètre est passé au trigger via la balise « param », l’attribut « value » correspond à sa valeur. Le tout est extrait dans un dictionnaire python passé au trigger qui se chargera de vérifier s’il dispose des paramètres nécessaires à son bon fonctionnement. Dans le cas contraire, il prendra la valeur par défaut du paramètre si elle est disponible. Sinon il affichera un message d’erreur dans le journal d’erreur[5].

Suite à ces paramètres, il se trouve les « targets » (cibles) : ce sont les systèmes qui doivent être visés par les threads de constructions d'archives d'installation.
La description de ces systèmes se situe plus bas dans le fichier.

On note la présence d'un script nommé explicitement « script inexistant ». Le programme notifiera donc ce problème dans le journal d'erreur.

```
<systems>
  <os name="linux">
    <os name="slc5">
      <os name="32" host="slc5-32" vm='True' shutoffvm='False' /><!-- shutoffvm default is True -->
      <os name="64" host="slc5-64" vm='True' />
    </os>
    <os name="slc6">
      <os name="32" host="slc6-32" vm='True' shutoffvm='False' />
      <os name="64" host="lbgw01" user="dvessier" />
    </os>
  </os>
  <os name="windows">
    <os name="xp">
      <os name="32" host="XP-32" vm='True' />
    </os>
    <os name="seven">
      <os name="64" host="Seven-64" vm='True' />
    </os>
  </os>
</systems>

</racine>
```

Figure 23 - Configuration XML des machines

Cette nouvelle figure est la fin du fichier de configuration du programme, il renseigne les machines virtuelles ou non qui devront produire les archives d'installation.

Le tout est organisé de manière hiérarchique pour permettre une factorisation des moyens de création des archives.

Sur chaque machine configurée, il existe différentes options :

- Name, c'est le nom de la machine, il est le seul paramètre obligatoire. Il est régulièrement utilisé dans le code pour identifier une machine. Cependant il peut ne pas être unique puisque le nom complet d'une machine dans le code prend en compte toute l'arborescence. D'où les multiples « 32 » ou « 64 » correspondant à la valeur finale de l'arbre des systèmes.

  Exemple : Windows-seven-64 ou Windows-XP-32

- Vm, est l'attribut qui permet de designer si le système d'exploitation est une machine virtuelle hébergée en local. Dans le cas contraire il suffit de ne pas mentionner cet attribut. C'est-à-dire que la valeur de l'attribut, ici a True, n'a en réalité aucune incidence, ce n'est que la présence de vm que vérifie le programme.

- Host a deux significations, si vm est actif, alors il constitue le nom de la machine virtuelle locale à utiliser.

Dans le cas contraire, c'est le nom de la machine sur le réseau. Le programme utilisera donc un protocole propre au CERN pour transformer ce nom en adresse IP. Ce protocole est géré par un nouveau module python spécialement conçu pour l'occasion. Il se charge d'effectuer les opérations nécessaires sur le protocole SOAP (*Simple Object Access Protocol*). Ce protocole SOAP est une interface délivrée par un web service développé par la section administration du parc informatique. Ce système fonctionne comme sur le schéma ci-dessous.

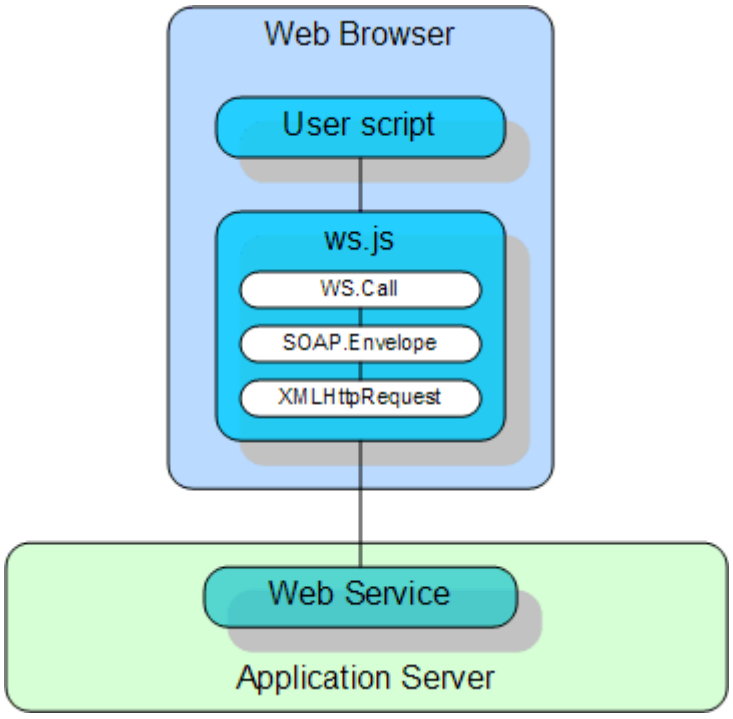


Figure 24 - Protocole SOAP via un web service

- Il existe aussi l'attribut shutoffvm. Sur le même principe que vm, il est optionnel et permet de spécifier si on ne souhaite pas éteindre la machine virtuelle après utilisation. Dans ce cas, le thread de contrôle des machines n'appliquera pas l'arrêt mais la suspension d'exécution du système d'exploitation. Toute la mémoire de la machine virtuelle sera alors conservée dans la mémoire vive de la machine hôte mais le processeur sera lui libéré. L'intérêt de ce système est de pouvoir réutiliser la machine par la suite en quelques millisecondes seulement. Dans le cas contraire, il faut attendre plus de deux minutes sur ma station de travail pour démarrer une machine virtuelle.

- Enfin nous avons user et pass qui sont respectivement l'utilisateur et le mot de passe pour la connexion sur le système. Dans le cas de linux, il est possible de se passer du mot de passe si le protocole est configuré correctement. Par exemple avec ssh il suffit de placer la clef de sécurité public sur la machine distante, ainsi grâce au protocole rsa, la connexion est sécurisée, sans demande de mot de passe.

Toutes ces configurations permettent de modeler le programme à l'utilisation que l'on souhaite en faire.

Cette configuration est évolutive puisqu'il peut être rajouté d'autres balises qui seront interprétées par d'autres modules qui viendront se greffer sur le programme principal.

ii. Le programme principal

C'est le module python avec le moins de lignes, il est en charge de lire le fichier de configuration pour lancer les différents triggers du XML.

Il possède aussi une fonction de tri des systèmes à lancer. En effet si l'utilisateur souhaite lancer la construction de l'archive d'installation sur tous les linux, le programme recherchera les différents systèmes dans le fichier XML de manière récursive. De plus l'utilisateur peut facilement demander plusieurs fois la construction sur un même système. Pour illustrer ce cas :

- Inclure tous les Linux SLC 5 (Linux-slc5-32, Linux-slc5-64)
- Inclure tous les Linux (Linux-slc5-32, Linux-slc5-64, Linux-slc6-32, Linux-slc6-64)

On constate que la liste des systèmes à produire contient alors des doublons après fusion des consignes :

(**Linux-slc5-32**, **Linux-slc5-64**, **Linux-slc5-32**, **Linux-slc5-64**, Linux-slc6-32, Linux-slc6-64)

Pour résoudre ce dysfonctionnement, le programme principal dispose d'une fonction de distinction des éléments de la liste. A chaque ajout d'élément, cette fonction vérifie que l'élément n'existe pas déjà puis insère l'élément de manière triée. La liste reste donc triée et la recherche dichotomique de l'élément à insérer permet des performances optimales.

Par ailleurs ce module principal permet aussi d'écrire des erreurs dans le journal en cas d'inexistence d'un trigger.

Enfin il compose au format texte la commande associée au trigger qui sera lancée par la suite pour que l'utilisateur puisse réutiliser simplement cette commande dans le futur sans repasser par le programme principal.

C'est encore un avantage de la programmation modulaire par plug-in, les triggers peuvent être lancés séparément du programme principal sous forme de ligne de commande Python.

iii. La « builder factory »

Un autre module essentiel pour le projet est la builder factory. Son rôle est minimaliste mais c'est le cœur de répartition des constructeurs d'archives d'installation. En fonction du nom du système cette « factory » va générer le thread constructeur associé. Un thread « constructeur linux » pour le Linux-slc6-32 ou encore un thread « constructeur Windows » pour le Windows-xp-32.

Ce procédé pourra être complexifié par la suite pour intégrer des notions plus complètes d'héritages de constructeur spécifié dans le fichier de configuration XML.

Cette notion s'inspire du pattern de la Factory. Le principe est de retourner un objet en fonction d'une chaine de caractères.

b. Interface

i. Interface utilisateur

La majorité des enseignants de l'ISIMA considère que la partie la plus importante d'un logiciel est son interface avec l'utilisateur. En effet c'est la seule partie que l'utilisateur visualise et manipule. Cela signifie qu'elle doit être des plus intuitives et simple. Cependant il ne faut évidemment pas négliger le reste du programme sans quoi le programme se résumera à une belle image.

Le programme manipulant un des objets relativement complexe en parallèle il est très compliqué de visualiser et comprendre ce que fait le programme avec une simple sortie textuelle.

Pour illustrer ce phénomène, prenons programme simple qui se charge d'afficher un compteur de secondes :

Figure 25 - Simple compteur avec et sans interface graphique

Sur cette figure les deux sorties (graphique à gauche et console à droite) sont tout a fait compréhensibles par l'utilisateur. Cependant dès qu'un programme dispose de threads, il devient bien plus complexe de comprendre ce qui se passe avec l'affichage console. Sur ce même exemple de programme avec trois threads compteur :

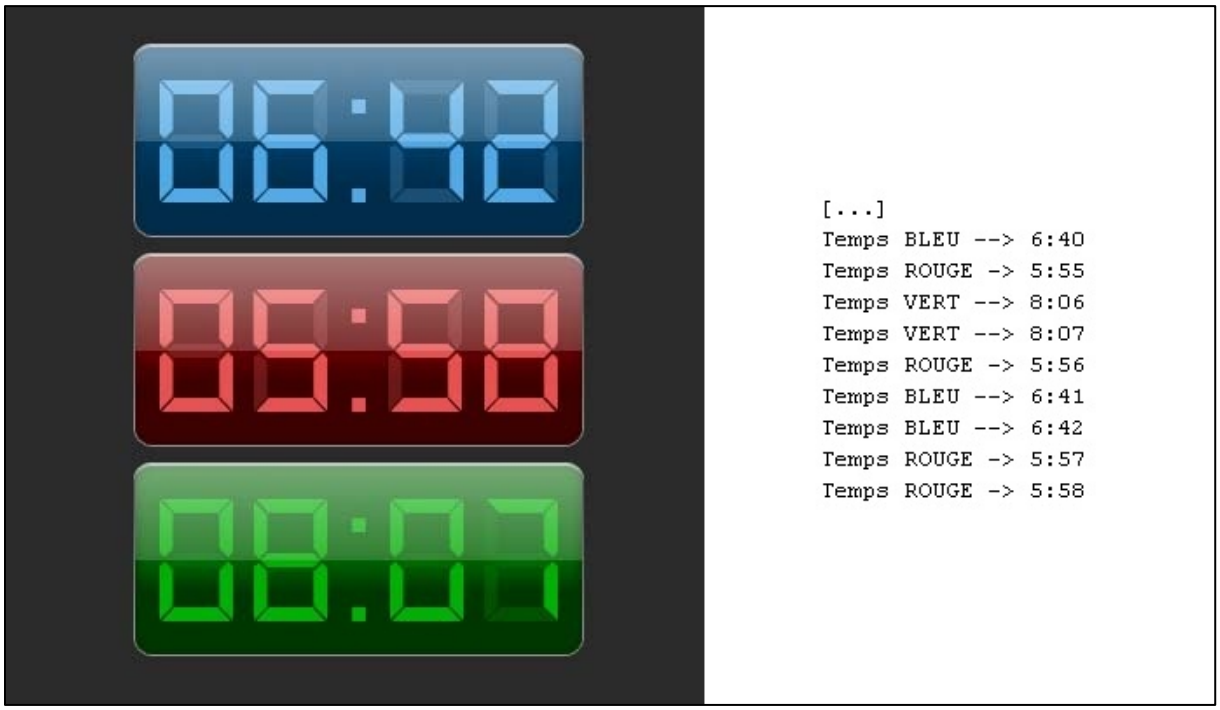


Figure 26 - Trois compteurs avec et sans interface graphique

C’est donc dans cette optique de compréhension et simplification du programme que j’ai développé un contrôleur graphique pour appuyer l’explication du déroulement du programme mais aussi pour comprendre et corriger des bugs potentiels en cours de développement bien plus rapidement.

La version deux du logiciel intègre cette notion de contrôleur directement dans le programme et est donc fortement couplée à celui-ci.

Mais après concertation et analyse, on s’est rendu compte que le but final est de faire fonctionner le programme sur un serveur. Il n’existe donc pas d’écran pour cette machine et il faudra donc pouvoir s’adapter à ce problème.

Pour pallier à cela, une hiérarchie d’interface graphique a été mise en place avec une classe de base « BaseWindow » qui se charge de référencer l’ensemble des fonctions interfacées avec le programme. Ce sont ensuite des classes qui dérivent de cette interface qui implémenteront toutes ses fonctions. On peut donc développer des interfaces graphiques pour le programme de manière interchangeable. Cette notion est tirée du pattern de modélisation modèle vue contrôleur.

Nous pourrons donc avoir un contrôleur graphique et un système de logs interchangeable.

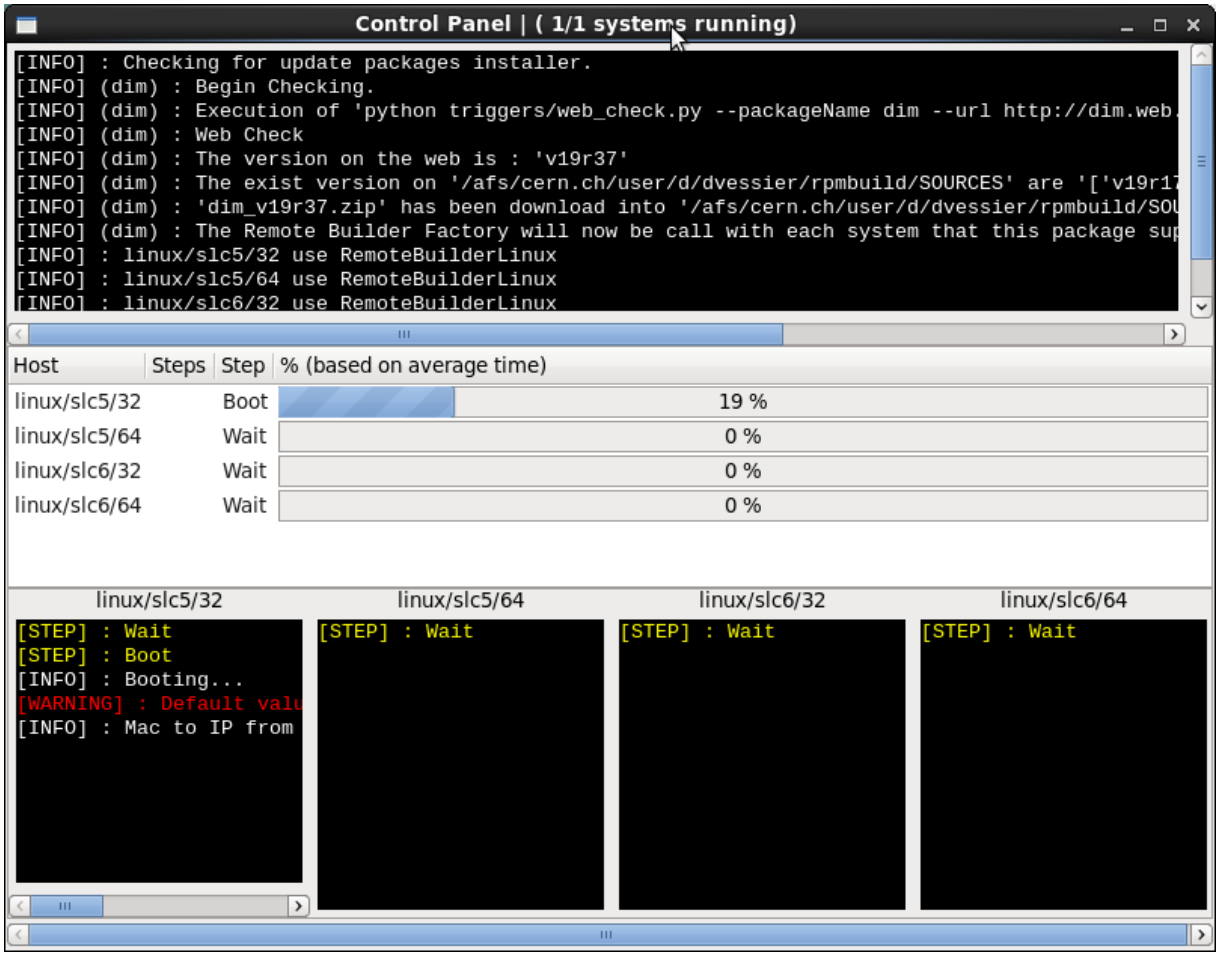


Figure 27 - Le contrôleur graphique du projet

Sur cette figure se trouve tous les éléments de contrôle du bon déroulement du programme :

- Des consoles (texte sur fond noir) créées à partir d'une boite textuelle. Les tags permettent de coloriser le texte en fonction de son caractère (étapes, infos, erreurs, …). Elles disposent aussi d'une vue disposant de barre de défilement pour pouvoir tout voir, mais aussi un système de défilement automatique pour que le message le plus s'affiche reste à l'écran.
  La console du haut est globale à tout le programme et donne des informations sur le paquet et l'exécution du trigger.
  Les consoles inferieures représentent les messages générés par chaque machine.

- Un titre comportant le nom du programme suivi du nombre de threads constructeurs d'archives d'installation en cours d'exécution.

- Une table d'avancement des étapes. Elle indique les étapes en cours sur les différents systèmes et propose aussi une estimation du temps que va prendre cette étape en se basant soit sur une valeur par défaut, soit sur la moyenne des dernières exécutions. Cette estimation est illustrée par cette barre de progression.

ii. Interface de log

Dans un second temps, lorsque le programme fonctionne bien, on peut remplacer l'interface graphique par une interface de log. Pour rendre ce journal le plus interprétable possible, il est découpé en un fichier général et un fichier par machine.

Le module de log propose plusieurs fonctionnalités.

Dans un premier temps, il permet à l'utilisateur de spécifier la sortie du journal. Celle-ci peut être console, dans un fichier ou bien personnalisée ce qui permettrait par exemple d'afficher les informations dans des pages Web.

Il permet aussi de définir l'importance des messages émis.

Level	When it's used
DEBUG	Detailed information, typically of interest only when diagnosing problems.
INFO	Confirmation that things are working as expected.
WARNING	An indication that something unexpected happened, or indicative of some problem in the near future (e.g. 'disk space low'). The software is still working as expected.
ERROR	Due to a more serious problem, the software has not been able to perform some function.
CRITICAL	A serious error, indicating that the program itself may be unable to continue running.

Figure 28 - Les niveaux du journal d'erreur

Ainsi, il est possible de ne faire apparaître que les informations les plus importantes en ne changeant qu'un seul paramètre dans le programme.

Cela implique aussi une programmation soignée et précise pour classifier chaque message dans une catégorie. Etant donné que c'est l'interface graphique qui a été développé en premier, ces niveaux n'ont pas été prévus initialement. Cependant, le principe d'affichage des messages était analogue puisqu'il existait une fonction pour :

- Afficher une information
- Afficher une alerte
- Afficher une erreur

Il a donc été possible d'adapter le code à ce nouveau système de journal sans avoir à répertorier l'ensemble des messages transmis.

L'information est devenue « INFO ».

L'alerte est devenue « WARNING »

Et l'erreur est devenue « ERROR »

On note ainsi que 2 catégories ne sont pas utilisées mais comme celles-ci sont les deux plus extrêmes, elles seront tout de même utilisées à de rares occasions.

c. Parallélisassions

Les différents constructeurs à distance s'exécutent de manière parallèle. Ce système présente beaucoup d'avantages mais nécessite une rigueur absolue. Les ressources d'une machine ne sont pas illimitées et l'utilisation de machines virtuelles en local consomme facilement plus de 30% de ses ressources sur une station de travail standard.

Il est donc inconcevable que cinq, six, ou plus de machines virtuelles, hébergées localement, soient lancées en parallèle.

Dans le cas de machines distantes, le nombre de constructions à distance est bien plus élevé mais reste limité.

Pour pallier à ce problème, il existe dans le programme un nouveau thread dont la tâche est uniquement l'ordonnancement des threads constructeurs d'archives.

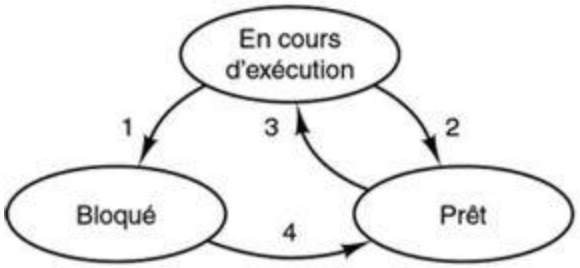


Figure 29 - Principe de l'ordonnancement des threads

Pour ce faire, ce thread ordonnanceur compte en permanence le nombre de threads constructeurs d'archives actif et si ce nombre est inferieur à une constante définie, il lance un nouveau constructeur d'archives.

Tous les objets thread constructeurs d'archives sont stockés dans une liste que l'ordonnanceur contrôle exclusivement.

Ainsi lorsque l'utilisateur demande l'interruption du programme, l'interface graphique ou le programme en lui-même, vont demander à cette ordonnanceur de s'arrêter. Il va à son tour émettre un signal sur les threads actifs afin qu'il termine au plus vite leur travail (dans le cas d'une machine virtuelle à arrêter, c'est l'arrêt forcé de la machine). Une fois que les threads en cours ont terminé leur exécution, l'ordonnanceur s'arrête à son tour.

Résultat

Ce projet a été développé en Python grâce à l'environnement de développement d'Eclipse dans le but d'optimiser la vitesse de développement et la modularité du code.

Il se compose de plusieurs grandes parties :

- La manipulation des machines virtuelles et son exploitation grâce à la bibliothèque « libvirt ».
- L'ordonnancement de la construction automatique des archives sur les machines distantes.
- Les triggers chargés de préparer les fichiers nécessaires à la réalisation.
- L'interface de contrôle pour communiquer les résultats à l'utilisateur.

Ce projet a permis de mettre en exergue les avantages et inconvénients d'une automatisation pour la construction d'archives d'installation. L'inconvénient majeur qui ressort, reste le temps passé pour programmer un tel système ainsi que l'hétérogénéité des manières de construire une archive impliquant le développement de nombreux plugins en fonction du système. Vient s'ajouter à cela les façons de concevoir une archive pour un paquet donné, ce qui implique une grande rigueur de mise en conformité de ceux-ci pour éviter une croissance exponentielle du nombre de plugins.

Dans le contexte du LHCb, ce projet présente surtout l'avantage d'un gain de temps à long terme pour la construction récurrente d'archives d'installations sur les différents systèmes exploités sur l'expérience. Nous avons aussi pu noter que de par son caractère modulaire le programme peut aussi permettre d'automatiser d'autre procédé, comme la cross-compilation[6] d'un code source.

Les tests ont été effectué à ce jour sur le paquet « dim », développé au CERN, dont l'automatisation de la construction de son archive d'installation fonctionne sur l'ensemble des systèmes d'exploitation Linux et Scientifique Linux du CERN ainsi que sur les machines virtuelles hébergées localement. La construction pour Windows est déjà structurée

et les différentes opérations nécessaires sont connues. Il reste à relier toutes ses opérations et à les intégrer dans le constructeur d'archive à distance.

Il est aussi prévu d'automatiser encore plus le processus en proposant un fonctionnement régulier géré par le système des tâches cron[7]. Ainsi les triggers vérifieront régulièrement la présence de nouvelles versions et si nécessaire la construction de nouvelles archives.

Conclusion et perspectives

Dans le cadre de l'administration du parc informatique, ce projet pourra apporter un gain de temps réel quant à l'automatisation de la construction d'archives d'installations.

Le projet est cependant amené à être mis à jour continuellement par la greffe de divers plugins. Ces ajouts seront basés sur un modèle d'héritage simple à comprendre. Si le code moteur est amené à être modifié, il sera simple d'apporter la modification car les parties les plus complexes sont bien commentées et rigoureusement formatées.

Le planning fixé au départ a été en partie respecté même si dans le détail de nombreux points d'analyse du langage ont pris plus ou moins de temps que prévu :

Figure 30 - Diagramme de GANTT effectif

La perspective d'avenir de ce projet résulte dans son maintien à jour et son exploitation au quotidien pour accroître continuellement le nombre d'heures économisé grâce au projet.

Pour ma part, ce projet m'a permis d'acquérir de nouvelles connaissances et de m'immiscer dans le milieu de la recherche. A ce titre, j'ai été agréablement surpris de la passion, du dévouement et de l'intérêt dégagé par l'équipe. Je me suis tout naturellement intégré avec plaisir dans cet univers.

Lexique

[1] **NSIS** : *Nullsoft Scriptable Install System* (Système d'installation scriptable)

[2] **Le ping** : Protocole pour vérifier la présence d'une machine sur un réseau. Par exemple, le client envoie une trame au serveur, celui-ci répond simplement qu'il a bien reçu la trame.

[3] **Un plugin** : Tiret de l'anglais « insérer dans », c'est un module, périphérique ou composant quelconque qui va pouvoir se greffer.

[4] **Une expression régulière** : C'est une suite de caractères destiné à correspondre à un motif. Il permet la recherche de ce motif dans un texte. Son avantage est qu'il possède un codage tel que l'on peut rechercher toute sorte de motif.

Par exemple « le [0-3][0-9] du mois » recherchera toutes les expressions contenant « le » suivi de deux nombres (entre 0 et 3 puis entre 0 et 9) puis de « du mois ».

Ainsi si un texte respecte un certain format, il est possible d'en extraire les informations via une expression régulière.

[5] **Journal d'erreur** : C'est une sortie (console, fichier ou affichage) qui recense les erreurs d'un programme.

[6] **Cross-compilation** : Méthode de compilation d'un code source sur une machine donnée, générant un exécutable binaire destiné à fonctionner sur une autre architecture. Par exemple, compiler une application Smartphone depuis un ordinateur.

[7] **Tâche cron** : Cron est un service linux s'exécutant en tâche de fond, il permet de lancer à intervalle régulier des tâches et programmes définis dans des fichiers de configuration.

Références

Bibliotheque GTK.

Récupéré sur www.gtk.org

***KVM*.**

Récupéré sur www.linux-kvm.org

***La bibliotheque 'libvirt'*.**

Récupéré sur libvirt.org

***Le CERN*.**

Récupéré sur www.cern.ch

***L'equipe informatique du CERN*.**

Récupéré sur information-technology.web.cern.ch

***Projet Linux Fedora*.**

Récupéré sur fedoraproject.org

***QEMU*.**

Récupéré sur qemu.org

***RPM*.**

Récupéré sur www.rpm.org