ISAIP
18, rue du 8 Mai 1945
49000 St Barthélémy d'Anjou

Dossier de Conception

Développement d'une application de personnalisation de système d'exploitation : "OS A La Carte"

1 Février 2007

Mohammed BADISS, Jérôme CHARLOT
$2^{ème}$ année CPI
Promotion 2005 – 2007

Sous la responsabilité de :
M Alexandre LEPRIEULT ;

Table des matières

Table des figures

1 Traçabilité

Version	Date	Commentaires
1.00	05/01/07	Première version
1.20	21/01/07	Ajout des schémas de conception
1.30	03/02/07	Ajout de la partie Rappel
1.40	03/02/07	Redéfinition des modules “Select” et “Feed”
2.00	04/02/07	Correction orthographe

2 Introduction

2.1 But du document

Ce dossier présente la phase de conception du logiciel d’un point de vue méthodologique. Il dépend très peu de l’application elle-même.

2.2 Documentation

2.2.1 Documents de référence

Les documents de référence sont le dossier de **Cahier des Charges** et la dossier de **Specifications Fonctionnelles**.

3 Rappels

Il est nécessaire d’effectuer quelques rappels, pour rappeler les parties importantes du déroulement du projet.

3.1 Objectifs du projet

Notre projet nommé “OS A La Carte” a pour but de **personnaliser** un système d’exploitation depuis une page Internet, plus particulièrement de générer une image du DVD d’installation d’une distribution Linux.

Cette image au format ISO est destinée a être gravée sur un DVD. Une fois gravée, l’utilisateur insère le DVD dans son lecteur de DVD puis amorce le processus d’installation. L’utilisateur installe alors ce système d’exploitation sur sa machine.

Notre travail repose donc entièrement sur la **modification** selon les demandes et les besoins du client de la distribution Linux nommée **Ubuntu**.

Cette **personnalisation** s’éffectue à la base du système d’exploitation, tout d’abord au niveau des programmes qui sont courramment utilisés et mais également au niveau de l’installation d’applications additionnelles.

3.2 Procédé de fabrication

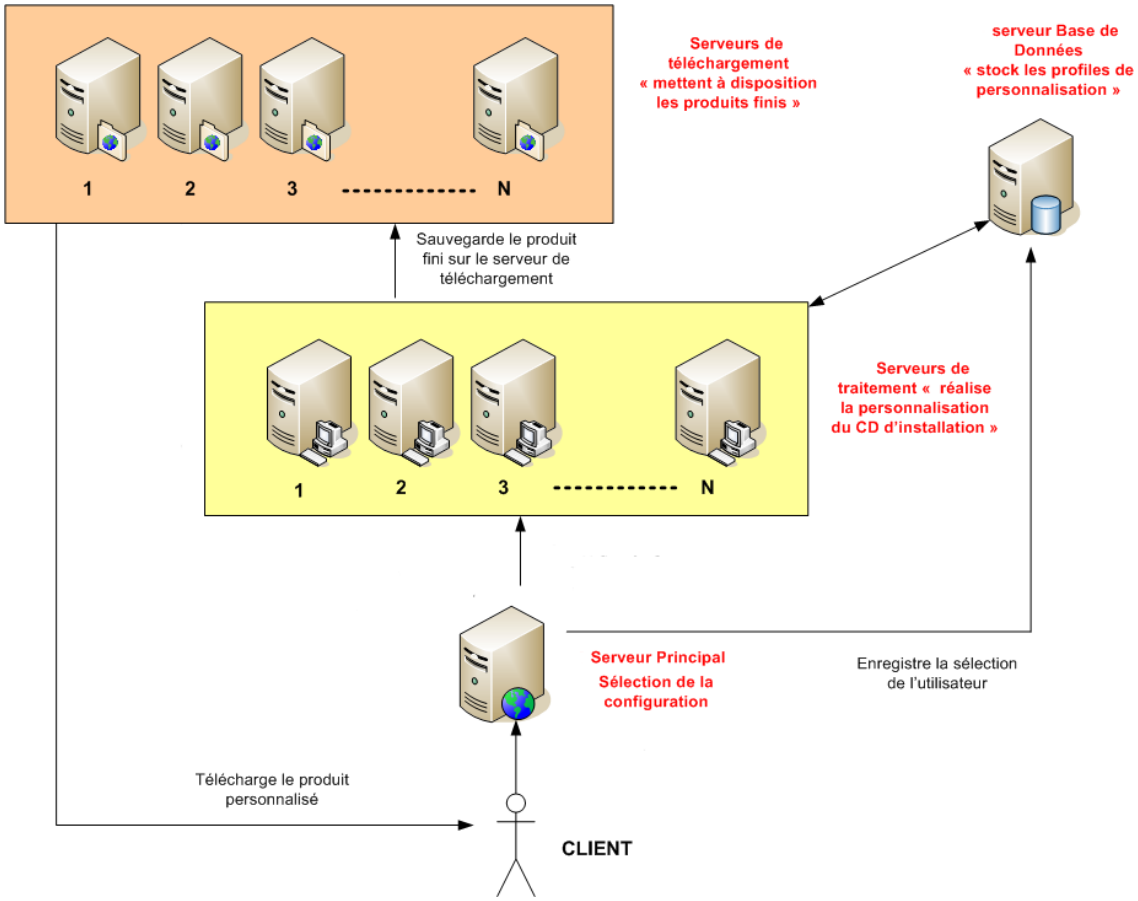

FIG. 1 – Schéma – Déroulement du procédé de fabrication.

Le procédé de fabrication se déroule en **trois** phases, qui sont les suivantes :

1. La première phase est la **sélection** de la configuration depuis notre interface Web. Cette opération est la seule effectuée par l'utilisateur. Le temps de réponse l'interface web dépend principalement de la bande passante du client, et de notre serveur. Mais avec les connexions actuelles, la technologie ADSL étant largement répendue, ce temps est **négligeable** (à peine quelques secondes).

2. Une fois la phase de personnalisation terminée. La phase de **génération** du DVD d'installation, réalisée par notre application, peut exiger environ **3** minutes sur le serveur dédié (un Pentium 4 ou équivalent).

3. Finalement, la phase de **téléchargement** de l'image du DVD est la plus **exigeante** au niveau temps, car comme la première phase, elle dépend de la bande passante que dispose le client mais surtout celle que nous allouons pour le serveur de téléchargement.

3.3 Les Fonctionnalités

Voici un schéma qui répresente l’étendue des choix pour lesquels l’utilisateur peut personnaliser son système d’exploitation :

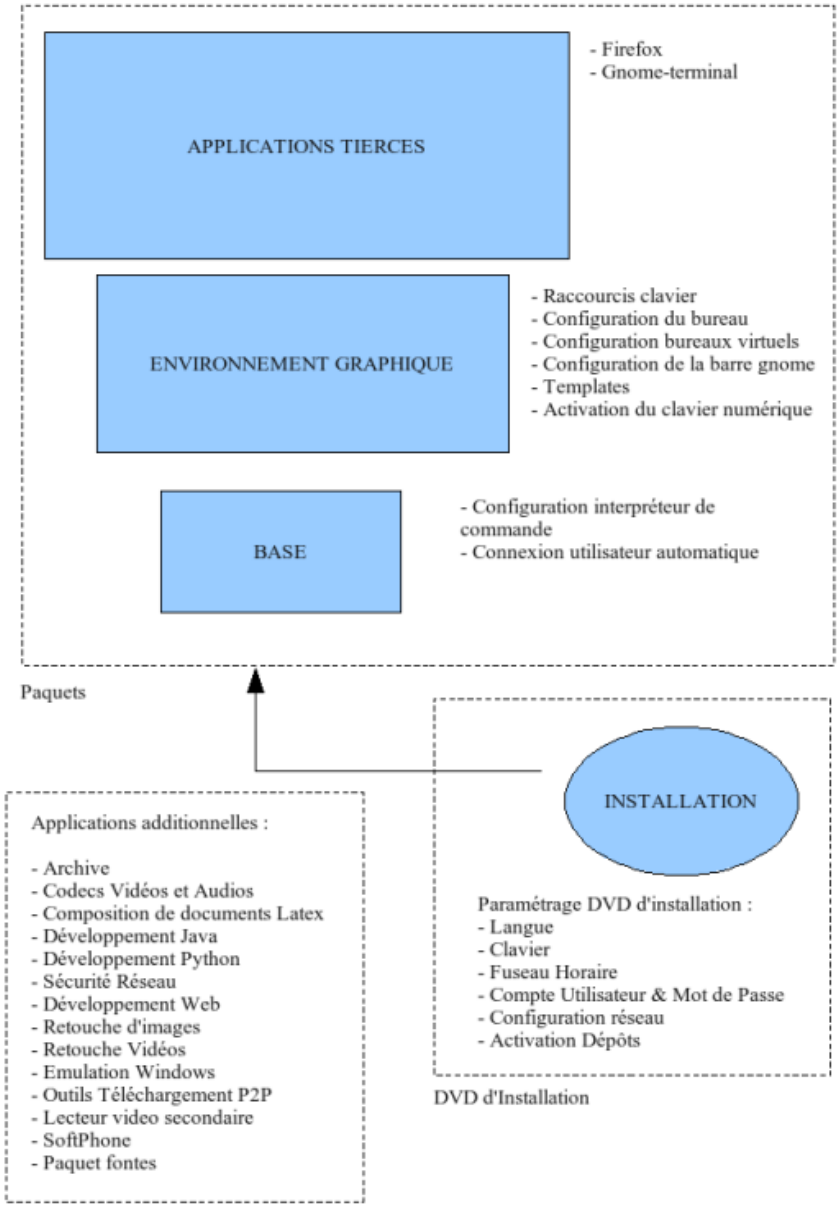

FIG. 2 – Schéma – Choix possibles de l’utilisateur.

L’utilisateur personnalise le DVD d’installation à **trois** niveaux différents.

Les trois niveaux sont :

1. Fonctionnalités au niveau du DVD d’installation.
2. Modifications des paquets les plus importants.
3. Choix des profiles d’applications.

Ces différents niveaux sont **abstraits** et **transparents** pour l’utilisateur, ils correspondent à un découpage en modules de l’application. Ce découpage est détaillé dans la partie “*Découpage de l’application en modules*”.

3.4 Découpage de l’application en modules

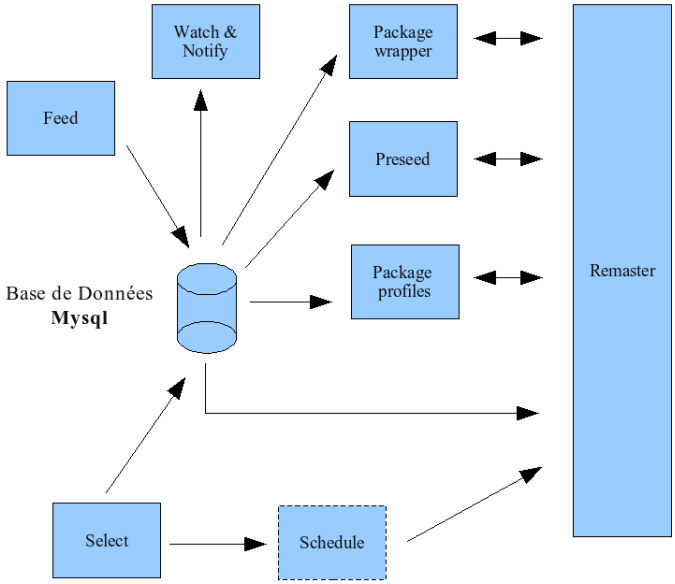

FIG. 3 – Schéma – Découpage de l’application OS A La Carte en modules.

3.5 Rôle de chaque module

– **“Feed”** prendra en compte le suivi des mises à jour proposées par les dépots officiels d’Ubuntu (Réponse à la contrainte de *sécurité*) ;
– **“Watch and Notify”** aura pour rôle d’afficher les changements des dépots ;
– **“Package Profiles”** s’occupera de la gestion des profiles d’applications ;
– **“Package Wrapper”** gérera les paquets, à savoir leur ouverture, leur modification, et finallement leur reconstruction ;
– **“Preseed”** concernera le paramétrage du CD d’installation ;
– **“Remaster”** aura pour tâche la création de l’image ISO en prenant en compte les modules **“Package Profile”**, **“Package Wrapper”** et **“Preseed”** ;

- **“Select”** sera l’interface ou l’utilisateur sélectionnera ses choix ;
- **“Schedule”** gérera les utilisateurs, avec une file d’attente, la sauvegarde des sélections des utilisateurs (Module Non Prioritaire).

4 Architecture dynamique du logiciel

4.1 Modèle Physique de Données

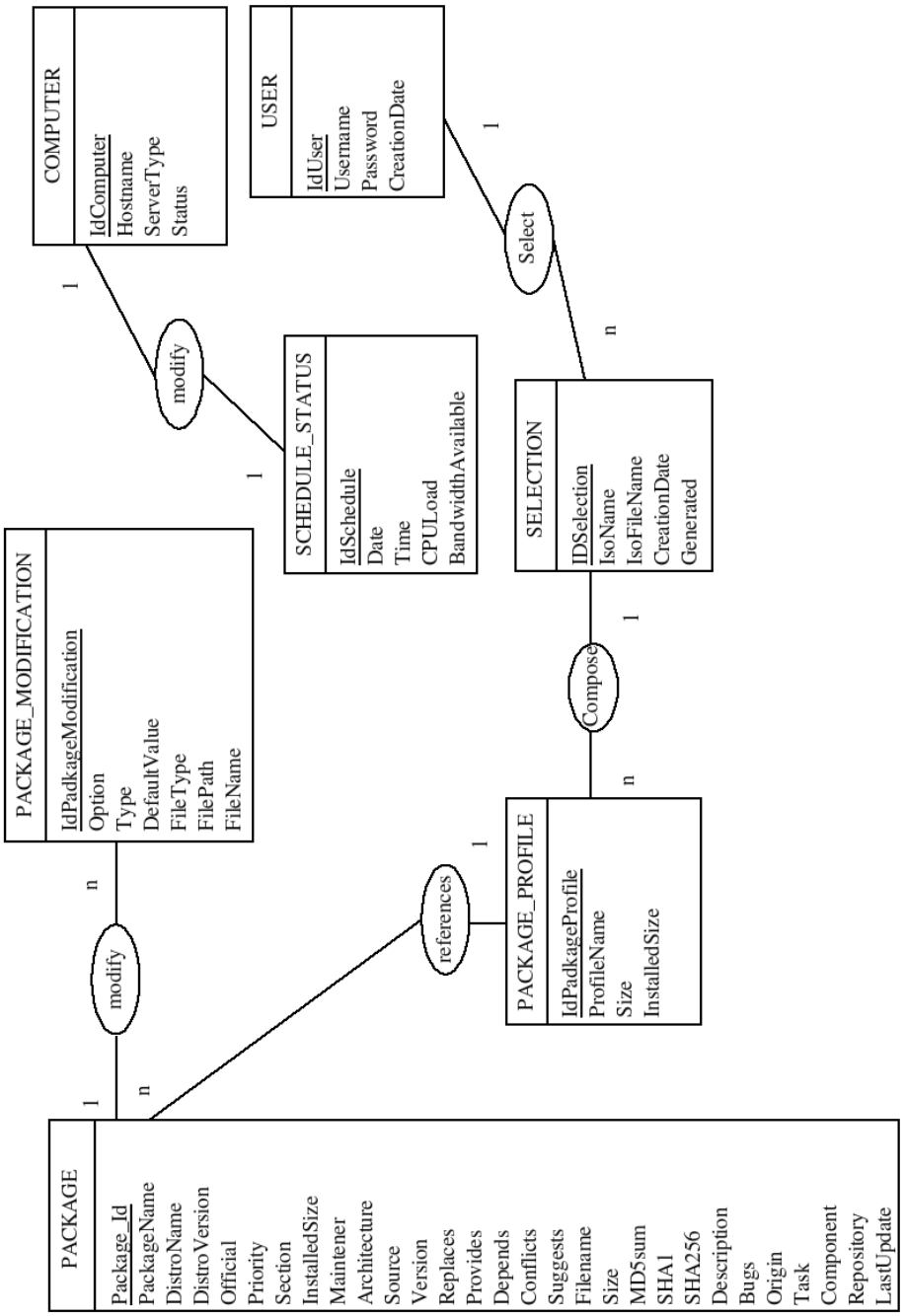

FIG. 4 – Schéma – Modèle Physique de Données pour l’application OS A La Carte.

Le **Modèle Physique de Données** est un modèle qui tient compte du moteur de base données pour gérer les données. Ainsi avec le MPD nous savons quels sont les types de chaque attribut et également les clés étrangères.

4.2 Tâche de préparation

4.2.1 Traitements effectués

Cette tâche est effectuée par l'administrateur, elle est **exceptionnelle** car réalisée qu'une seule fois lors de l'installation de l'application.
Elle consiste à **préparer** les répertoires et autres fichiers qui seront utilisés par la suite par le programme de génération de l'image DVD.

C'est également durant cette étape que nous allons utiliser la **cryptographie** avec l'aide des clés publique et privée. En effet lorsque le processus d'installation est en cours d'exécution, il va vérifier que les programmes présents sur le DVD d'installation n'ont pas été modifiés pour installer des applications dites “compromises” (contenant des trojans, backdoor, bombes logiques, ...).

Ainsi il est nécessaire de signer l'image DVD et de joindre la clé publique dans l'image. Cette étape est réalisée pendant la **tâche de préparation**.

4.2.2 Enchaînements et flots de données

Une fois tout le traitement réalisé, l'administrateur regroupe les fichiers sous forme d'archives. Ce seront ces archives qui seront utilisées comme **références** pour construire l'image du DVD d'installation.

4.2.3 Modules utilisés

Les modules qui seront utilisés sont “Feed” et “Remaster”.

4.3 Tâche de mise à jour des paquets

4.3.1 Traitements effectués

Notre image du DVD se base sur Ubuntu “Edgy Eft” qui est sortie au mois d'octobre 2006, or depuis ce temps il y a eu des **mises à jour** qui ont été diffusées sur les dépôts.

Tous les laps de temps réguliers, par exemple toutes les 6 heures ou 12 heures, nous réactualiserons le contenu de dépôts “Sécurité” et mettrons à jour notre archive de référence.

4.3.2 Enchaînements et flots de données

Le module “Feed” réactualisera les nouveaux paquets disponibles et une requête sur la base de données fera ressortir les paquets qui seront présent sur l'image DVD.
Il nous restera donc a mettre à jour les paquets sélectionnés et la liste qui les référence tous.

4.3.3 Modules utilisés

Les modules qui seront en relation avec cette tâche seront “Remaster”, “Watch and Notify” et “Feed”

5 Conception des modules

5.1 Module “Feed”

5.1.1 Objectifs du module

Ce module prendra en compte le suivi des mises à jour proposées par les dépots officiels d’Ubuntu (Réponse à la contrainte de *sécurité*).

Ce module sera développé en langage procédurale **Python**.

5.1.2 Relations d’utilisation avec d’autres modules

Ce module bien qu’étant très fortement lié au module **“Watch & Notify”** n’est pas en relation avec d’autres modules.
Cependant ce module sera en relation avec la base de données.

5.2 Conception du module “Feed”

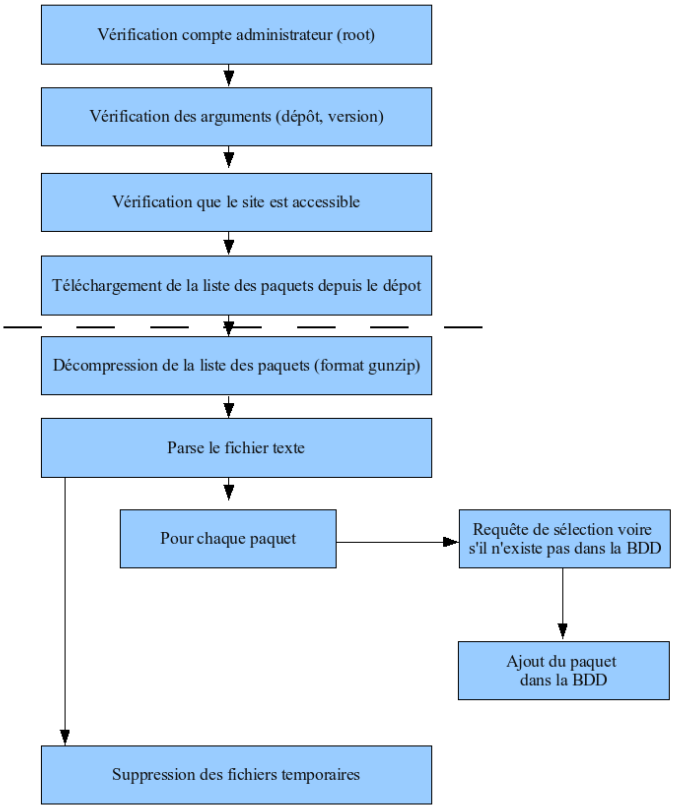

FIG. 5 – Schéma – Conception du module “Feed”.

Le modules “Feed” fonctionne indépendamment des autres modules et va rechercher des informations dans la base de données et en ajouter et/ou en mettre à jour.

5.2.1 Procédure de vérification des informations fournies en paramètre.

Tout d’abord le programme “Feed” pour fonctionner a besoin de **3** paramètres qui sont les suivants :

- l’**architecture** (par exemple : i386, ppc ou amd64) ;
- le nom du **dépôt** (par exemple : edgy, edgy-updates, edgy-security ou edgy-backports) ;
- les **composants** de chaque dépôts (par exemple : main, restricted, multiverse, universe).

Pour effectuer la vérification, tout d’abord nous comptons le nombre d’arguments avec le module **sys.argv** il faut que ce tableau ait une longueur de **4** éléments (le nom du programme et les 3 arguments).

Ensuite, nous essayons de vérifier que les informations fournies sont exactes, pour cela nous allons vérifier que le **fichier** correspondant au **composant du dépot** existe.

Nous allons pour cela, utiliser le module Python **httplib** et les fonctions suivantes :

- **httplib.HTTPConnection()** : qui instancie un objet de type **connexion HTTP** ;
- **request()** : qui émet une **requête** de type “GET” avec les paramètres fournis ;
- **getresponse()** : qui permet de savoir l’**état** de la requête **HTTP** * ;
- **close()** : qui permet de fermer la connexion **HTTP**.

* Note : Le résultat est celui défini par la norme **HTTP**, le code de retour normal est 200, tous les autres codes sont des erreurs (par exemple **404** pour fichier non trouvé).

Ainsi si la fonction **getresponse()** retourne un code différent de **200**, alors c’est que l’utilisateur se sera trompé à saisir les paramaètres, ce qui aura pour conséquence d’interrompre le programme avec une sortie d’erreur à l’aide de la fonction **sys.exit("erreur saisie de paramètre")**.

Une fois que nous nous sommes assurés que les paramètres transmis sont valides, nous pouvons passer au traitement des paquets contenu dans les dépots.

5.2.2 Traitement des dépots et alimentation de la base de données.

Nous allons à nouveau utiliser le module Python **httplib** pour télécharger localement le fichier qui liste l’ensemble des paquets pour un composant d’un dépot donné.

Pour cela on utilise les fonctions suivantes :

- **httplib.HTTPConnection()** : qui instancie un objet de type **connexion HTTP** ;
- **request()** : qui émet une **requête** de type “GET” avec les paramètres fournis ;
- **getresponse()** : qui permet de savoir l’**état** de la requête **HTTP** * ;
- **urllib2.Request()** : qui permet de récupérer une copie local d’un fichier accessible sur un serveur web ;
- **request.add_header()** : permet de spécifier le **type de fichier** que nous téléchargerons (le MIME TYPE), il s’agit ici d’une **archive compressée** au format **GZIP** (nous mettrons comme paramètres : ’Accept-encoding’, ’gzip’) ;
- **close()** : qui permet de fermer la connexion **HTTP**.

Ainsi, à ce stade, nous avons récupéré une copie du fichier qui contient la liste des paquets qui est compressé au format **GZIP**.

Pour traiter cette archive, nous allons utiliser le module Python **gzip**, et les fonctions suivantes :

- Déclaration du fichier en fichier **compressé StringIO.StringIO(compresseddata)** ;
- **gzip.GzipFile(fileobj=compressedata)**, permet de géré les fichiers compressés au format **gzip** ;
- **open()** permet d’ouvrir le fichier compressé ;
- **read()** permet de lire le contenu du fichier.

Ensuite, nous lisons le fichier de contenu du dépôt (la liste de ses paquets), on le “parse”, on analyse chaque ligne, un paquet est définit par le mot clé **Package** et toutes les lignes qui suivent sont des **informations relatives** à ce paquet.

```
Package: apache2
Priority: optional
Section: web
Installed-Size: 80
Maintainer: Ubuntu Core Developers <ubuntu-devel@lists.ubuntu.com>
Original-Maintainer: Debian Apache Maintainers <debian-apache@lists.debian.org>
Architecture: i386
Version: 2.0.55-4ubuntu4
Depends: apache2-mpm-worker (= 2.0.55-4ubuntu4) | apache2-mpm-prefork (= 2.0.55-
 | apache2-mpm-perchild (= 2.0.55-4ubuntu4)
Filename: pool/main/a/apache2/apache2_2.0.55-4ubuntu4_i386.deb
Size: 35976
MD5sum: b41e510af15a616746930c7ba86362c2
SHA1: be0399c61f2b3e3a27a0b1f2f558bc7218b7f7bc
SHA256: 2d1659ecb4512b257c933a2c9842f8a416641f939000f7d105ef9fa30c167a3b
Description: next generation, scalable, extendable web server
 Apache v2 is the next generation of the omnipresent Apache web server. This
 version - a total rewrite - introduces many new improvements, such as
 threading, a new API, IPv6 support, request/response filtering, and more.
Bugs: mailto:ubuntu-users@lists.ubuntu.com
Origin: Ubuntu
Task: lamp-server

Package: apache2-mpm-prefork
Priority: optional
Section: net
Installed-Size: 468
...
```

FIG. 6 – Schéma – Exemple d’un fichier dépot contenant une liste de paquets.

Avant de créer un nouvel enregistrement dans la base de données, on vérifie que cet enregistrement n’existe pas déjà avec les fonctions du module **Mysqldb** :

- **db.cursor()** : définit un curseur pour manipuler les données de la base de données ;
- **cursor.execute()** : permet d’exécuter une requête au niveau de la base de données et notamment la table “Package” ;
- **cursor.fetchall()** : permet de récuperer le résultat de la requête sous forme d’un tableau en Python.

Une fois que le script Python “Feed” a terminé son exécution, on se retrouve avec une **liste de paquets** dans la base de données qui **correspond** au contenu des **dépots** d’Ubuntu disponibles sur Internet.

5.2.3 Définitions de types

Ce module est composé des types suivants :

Nom variable	Type de données
Package_id	int(5)
PackageName	varchar(50)
DistroName	varchar(20)
DistroVersion	varchar(6)
Official	tinyint(1)
Priority	varchar(10)
Section	varchar(20)
InstalledSize	int(15)
Maintener	varchar(50)
Architecture	varchar(9)
Source	varchar(45)
Version	varchar(45)
Replaces	varchar(50)
Provides	varchar(75)
Depends	text
Conflicts	varchar(75)
Suggests	text
Filename	varchar(250)
Size	int(15)
MD5sum	varchar(32)
SHA1	varchar(60)
SHA256	varchar(60)
Description	text
Bugs	varchar(75)
Origin	varchar(20)
Task	varchar(125)
Component	varchar(20)
Repository	varchar(20)
LastUpdate	datetime
CD	tinyint(1)
PackageProfileSelection	tinyint(1)

FIG. 7 – Tableau – Liste des variables et leur type du module “Feed”

5.2.4 Dépendances du modules

Pour fonctionner ce module a besoin des dépendances suivantes :
- urllib2
- httplib
- StringIO
- gzip
- sysgzip
- MySQLdb
- time

5.3 Module “Watch and Notify”

5.3.1 Objectifs du module

Ce module aura pour rôle d’afficher le résultat du module “Feed” sous forme d’une page web. Ainsi il ne concerne pas directement la personnalisation du DVD d’installation.

5.3.2 Relations d’utilisation avec d’autres modules

Ce module sera en relation avec la base de données.

5.3.3 Définitions de types

Ainsi ce module partagera les mêmes variables que le module “Feed” qui sont les suivantes :

Nom variable	Type de données
Package_id	int(5)
PackageName	varchar(50)
DistroName	varchar(20)
DistroVersion	varchar(6)
Official	tinyint(1)
Priority	varchar(10)
Section	varchar(20)
InstalledSize	int(15)
Maintener	varchar(50)
Architecture	varchar(9)
Source	varchar(45)
Version	varchar(45)
Replaces	varchar(50)
Provides	varchar(75)
Depends	text
Conflicts	varchar(75)
Suggests	text
Filename	varchar(250)
Size	int(15)
MD5sum	varchar(32)
SHA1	varchar(60)
SHA256	varchar(60)
Description	text
Bugs	varchar(75)
Origin	varchar(20)
Task	varchar(125)
Component	varchar(20)
Repository	varchar(20)
LastUpdate	datetime
CD	tinyint(1)
PackageProfileSelection	tinyint(1)

FIG. 8 – Tableau – Liste des variables et leur type du module “Watch and Notify”

5.4 Module “Package Profiles”

5.4.1 Objectifs du module

Ce module s’occupera de la gestion des profiles d’applications. Il aura pour rôle de copier les paquets additionnels sélectionnés par l’utilisateur.

5.4.2 Relations d’utilisation avec d’autres modules

Ce module sera en relation avec la base de données.

5.5 Conception du module “Package Profile”

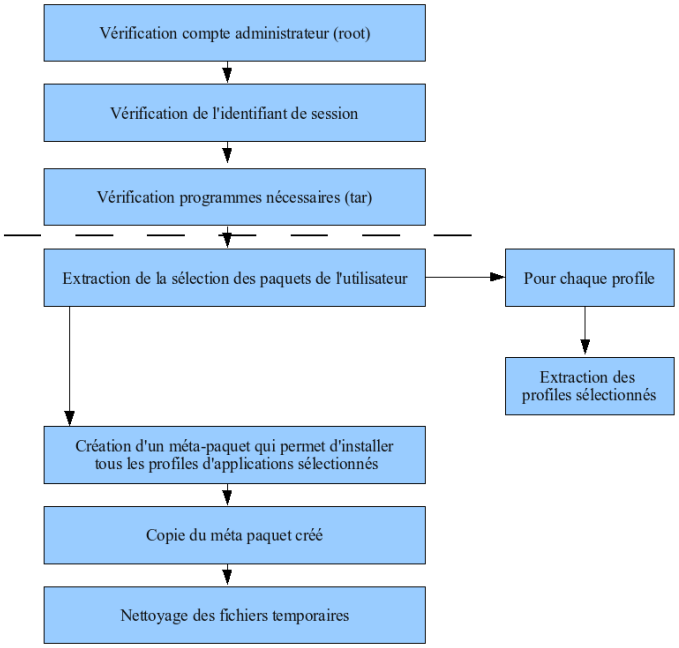

FIG. 9 – Schéma – Conception du module “Package Profile”.

5.5.1 Définitions de types

Ce paragraphe donne la liste des définitions de type. Dans le cas d’une conception orientée objets cette définition de type peut être unique et représenter les attributs de l’objet associé au module.

Nom variable	Type de données
IdPackageModification	int(8)
Option	varchar(250)
Type	tinyint
DefaultValue	varchar(30)
FileType	varchar(10)
FilePath	varchar(100)
FileName	varchar(35)

5.5.2 Dépendances du modules

Pour fonctionner ce module a besoin des dépendances suivantes :
– MySQLdb

5.6 Module “Package Wrapper”

5.6.1 Objectifs du module

Ce module gérera les paquets, à savoir leur ouverture, leur modification, et finallement leur reconstruction.

5.6.2 Relations d’utilisation avec d’autres modules

Ce module sera en relation avec la base de données.

5.6.3 Définitions de types

Ce paragraphe donne la liste des définitions de type. Dans le cas d’une conception orientée objets cette définition de type peut être unique et représenter les attributs de l’objet associé au module.

5.6.4 Dépendances du modules

Pour fonctionner ce module a besoin des dépendances suivantes :
– MySQLdb

5.7 Module “Preseed”

5.7.1 Objectifs du module

Ce module concerne le paramétrage du DVD d’installation avec la réalisation du fichier de configuration.

5.7.2 Relations d’utilisation avec d’autres modules

Ce module sera en relation avec la base de données.

5.8 Conception du module “Preseed”

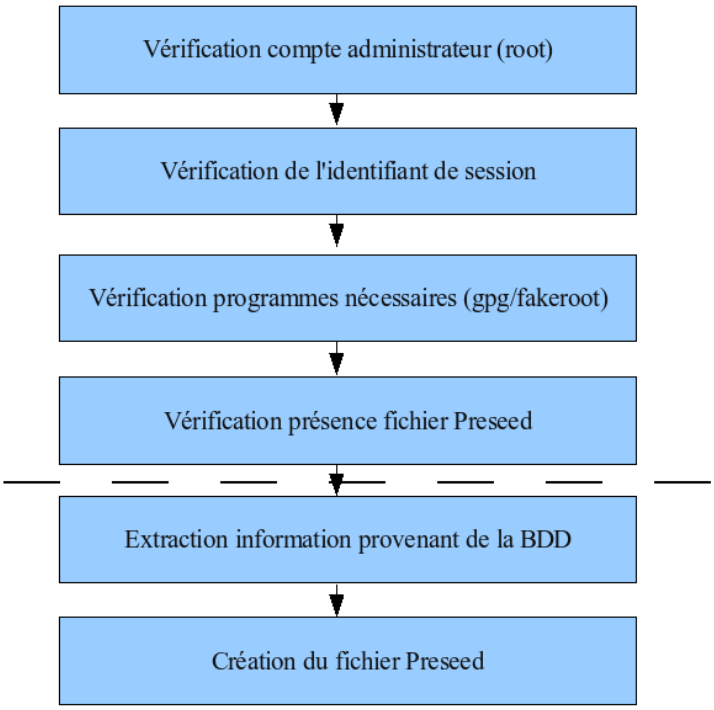

FIG. 10 – Schéma – Conception du module “Preseed”.

5.8.1 Définitions de types

Ce paragraphe donne la liste des définitions de type. Dans le cas d’une conception orientée objets cette définition de type peut être unique et représenter les attributs de l’objet associé au module.

Nom variable	Type de données
preseed_lang	varchar(20)
preseed_network	tinyint(1)
preseed_hostname	varchar(20)
preseed_domainname	varchar(20)
preseed_timezone	varchar(45)
preseed_user	varchar(30)
preseed_password	varchar(30)

5.8.2 Dépendances du modules

Pour fonctionner ce module a besoin des dépendances suivantes :
– MySQLdb

5.9 Module “Remaster”

5.9.1 Objectifs du module

Ce module aura pour tâche la création de l’image ISO en prenant en compte les modules “Package Profile”, “Package Wrapper” et “Preseed”.

5.9.2 Relations d’utilisation avec d’autres modules

Ce module sera en relation avec les modules “Package Profile”, “Package Wrapper” et “Preseed” mais également avec la base de données.

Pour fonctionner le module “Remaster” nécessite que certains fichiers soient préparer, pour cela nous avons créé un script qui s’appele **Préparation Remaster** qui a été concu, comme ceci :

5.10 Conception du script “Préparation Remaster”

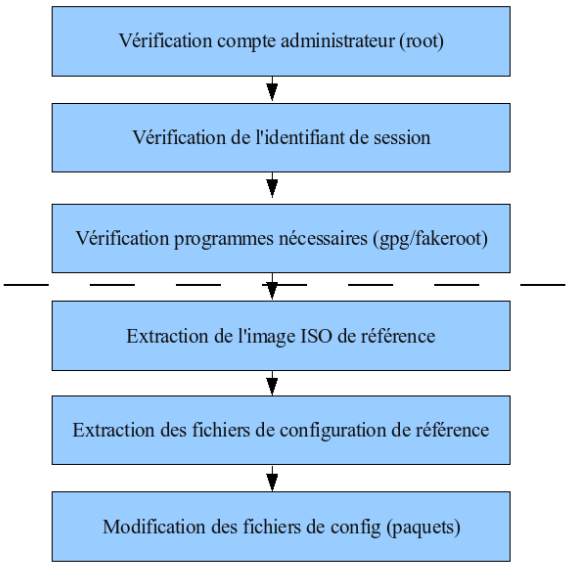

FIG. 11 – Schéma – Conception du script “Préparation Remaster”.

5.11 Conception du module “Remaster”

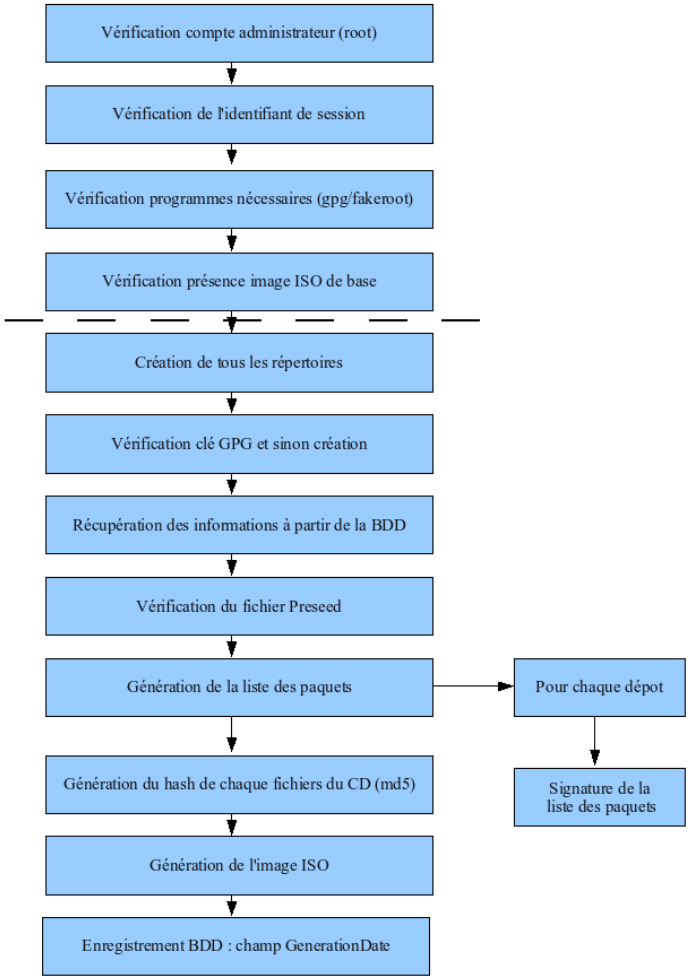

FIG. 12 – Schéma – Conception du module “Remaster”.

5.11.1 Définitions de types

Ce paragraphe donne la liste des définitions de type. Dans le cas d’une conception orientée objets cette définition de type peut être unique et représenter les attributs de l’objet associé au module.

Nom variable	Type de données
IDSelection	int(8)
IsoName	varchar(75)
IsoFileName	varchar(100)
CreationDate	datetime
GenerationDate	datetime

5.11.2 Dépendances du modules

Pour fonctionner ce module a besoin des dépendances suivantes :
- StringIO
- gzip
- sysgzip
- MySQLdb

5.12 Module “Select”

5.12.1 Objectifs du module

Ce module est le **point d’entrée** de notre logiciel. En effet, c’est dans cette interface que l’utilisateur effectue la sélection et les différents choix de sa configuration.

Cette partie fait référence à la spécification d’interface Homme – Machine définie dans le dossier de spécifications. Ainsi, pour que l’utilisateur ne se **perde** pas dans tous les paramètres, nous allons afficher une page web **simple**.

On retrouve la division en 3 parties : la configuration du système de base (au niveau du Preseed), la configuration détaillée de paquets et enfin la sélection de profiles d’applications.

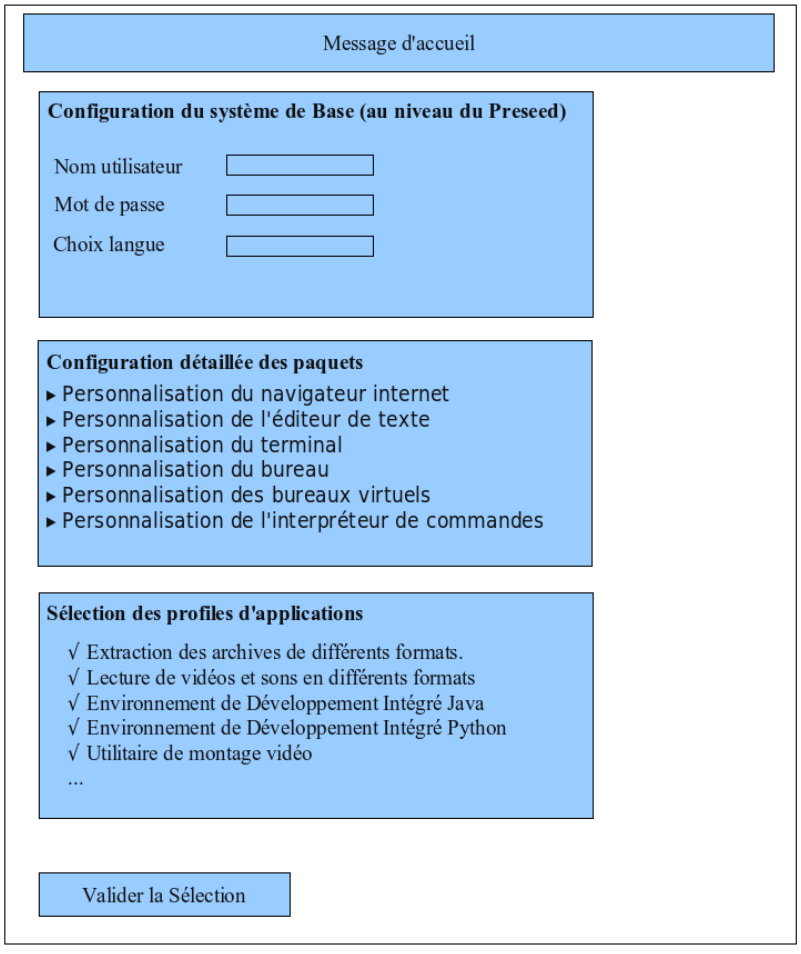

FIG. 13 – IHM – Interface non détaillée de sélection de configuration de l’utilisateur .

5.12.2 Relations d’utilisation avec d’autres modules

Après les vérifications nécessaires, ce module sera en relation avec la base de données et plus particulièrement la table “Selection”.

5.13 Conception de l’interface Homme – Machine

Pour concevoir cette spécification, nous utiliserons les technologies **Javascript** et **CSS**.

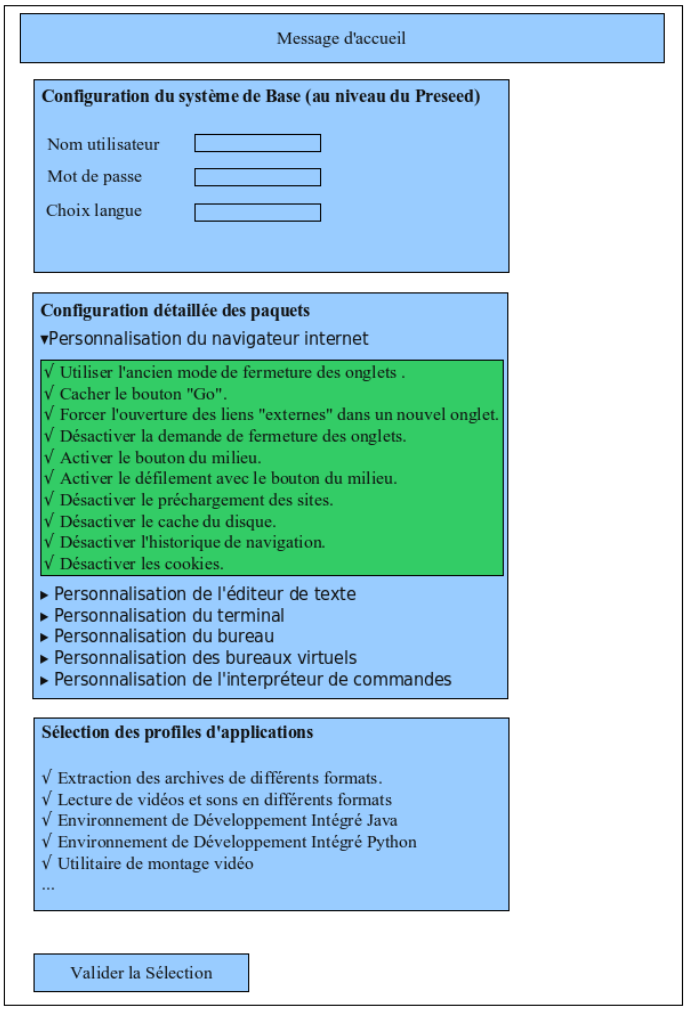

FIG. 14 – IHM – Interface détaillée de sélection de configuration de l’utilisateur .

Toutes les zones de choix de configuration masquées, représentées en **verte** sur l’IHM, seront délimitées par des balises HTML de type **...**, par défaut, elles seront masquées, nous utiliserons donc les attributs suivants : **style="visibility :hidden ;display :none"**.

Nous gérerons les événements, à l’aide du **Javascript**. Dès que l’utilisateur cliquera sur le titre du paquet à modifier par exemple “Personnalisation du navigateur internet” ou encore “Personnalisation de l’éditeur de texte” une fonction Javascript sera appelée.

L’événement déclenché appellera la fonction Javascript “toggleVisibilité” qui aura pour but d’**afficher** la zone si elle est masquée ou alors de la **cacher** si elle est affichée.

Pour afficher la zone, nous positionnerons les variables **Visibility** à “visible” et **Display** à “inline” de la zone selectionnée et au contraire pour masquer la zone nous positionnerons les variables **Visibility** à “hidden” et **Display** à “none”.

5.14 Procédure de vérification de la sélection de l’utilisateur

Pour transmettre les informations du formulaire nous utilisons la méthode **POST**. La vérification va s’effectuer dans la page **PHP** appelée **insert.php**.

Pour récuperer la valeur des informations transmises, nous allons utiliser la fonction PHP **$HTTP_POST_VARS**.

Les types suivants seront considérés comme **obligatoires** lors de l’enregistrement d’une nouvelle sélection dans la base de données et la page **insert.php** va donc vérifier qu’ils sont correctement renseignés :

- IDSelection ;
- preseed_lang ;
- preseed_network ;
- preseed_hostname ;
- preseed_domainname ;
- preseed_timezone ;
- preseed_user ;
- preseed_password ;
- IsoName ;
- IsoFileName.

5.15 Utilisation de fonctions PHP

Pour se connecter à la base de données et y réaliser toutes les opérations nécessaires, comme des sélections ou des insertions, nous utiliserons les fonctions **PHP/MySQL** suivantes :

- **mysql_connect()** : permet de se connecter à un serveur de base de données ;
- **mysql_select_db()** : permet de se connecter à une base de données ;
- **mysql_query()** : envoie une requête de sélection et d’ajout au SGBDR ;
- **mysql_fetch_row()** : réalise le traitement de chaque enregistrements du résultat d’une requête ;
- **mysql_fetch_array()** : récupère les informations sous forme de tableau PHP (array).

Dans le but de lancer la procédure de génération de l'image ISO, nous exécuterons alors le script de génération avec la fonction **popen()**, qui a pour but de lancer une commande au niveau non plus du serveur web mais du système d'exploitation.

La fonction **popen()** sera donc appelée avec en paramètre le **script Python** en charge de la génération et l'**identifiant** de l'utilisateur dans le but de récupérer les informations enregistrées dans la base de données.

5.15.1 Définitions de types

Les variables que nous utilisons pour ce module sont définies de la manière suivante :

Nom variable	Type de données
IDSelection	varchar(12)
preseed_lang	varchar(20)
preseed_network	tinyint(1)
preseed_hostname	varchar(20)
preseed_domainname	varchar(20)
preseed_timezone	varchar(45)
preseed_user	varchar(30)
preseed_password	varchar(30)
IsoName	varchar(200)
IsoFileName	varchar(200)