ETNA
Projet de Fin d'Étude 2005-2007
RimElse
Cahier des charges technique

4 mai 2006

Table des matières

1 Norme de codage **4**
- 1.0.1 Nommage . . . 4
- 1.0.2 Présentation . . . 4
- 1.1 Commentaires . . . 4
  - 1.1.1 Entête des fichiers . . . 5
  - 1.1.2 Fichier Ruby . . . 5
  - 1.1.3 Autres langages . . . 6

2 Séquence de démarrage **7**
- 2.1 Chargeur d'amorçage . . . 7
- 2.2 Initramfs . . . 8
- 2.3 Reconnaissance matérielle . . . 8
- 2.4 Init script . . . 9
  - 2.4.1 Étapes . . . 9
  - 2.4.2 Structure d'un de démarrage de service . . . 10
- 2.5 Bootsplash . . . 10
- 2.6 Démarrage du serveur X . . . 10

3 Gestionnaire de paquets (rebgen) **13**
- 3.1 Structure des paquets . . . 13
  - 3.1.1 Configuration du paquet . . . 13
  - 3.1.2 Génération de la base de donnée . . . 14
  - 3.1.3 Compilation . . . 15
  - 3.1.4 Épuration . . . 15
  - 3.1.5 Création de l'arborescence . . . 15
  - 3.1.6 Création du paquet . . . 16
  - 3.1.7 Signature . . . 16
  - 3.1.8 Publication . . . 16
- 3.2 Identification des patches . . . 16
- 3.3 Gestion des dépendances . . . 16
- 3.4 Application des patches . . . 16

4 Gestionnaire de configuration **17**
- 4.1 Gestionnaire de paquets . . . 17
- 4.2 Gestionnaire de configuration graphique . . . 17
- 4.3 Éditeur de configurations . . . 17

5 Outils **18**
- 5.1 Communication . . . 18
  - 5.1.1 Intra Else Team . . . 18
  - 5.1.2 Communication externe . . . 18

5.2 Développement . . . 19
5.2.1 Git . . . 19

A SQLite **21**

Introduction

Ce document fait suite au cahier des charges fonctionnel de la distribution RimElse, rédigé par la Else Team.

Après avoir décrit dans le précédent document les fonctionnalités générales de RimElse, celui-ci expose les différents paramètres intervenant dans son élaboration.

Pour ce faire, ce document décrit dans un premier temps la norme de codage utilisée dans les différentes productions. Puis dans un second, il définit les étapes de la séquence de démarrage. La troisième partie, analyse la gestion des paquets et les développement nécessaires à sa réalisation. Pour finir, ce document expose les caractéristiques de l'implémentation du gestionnaire de configuration.

Chapitre 1

Norme de codage

Dans le cadre de la conception de la RimElse, le groupe ELSE utilisera la norme de codage suivante. Celle-ci a pour but de faciliter la compréhension ainsi que la ré-utilisation et le développement des différentes productions.

Pour faciliter l'accès au code à une plus grande communauté, tous les commentaires seront en anglais.

1.0.1 Nommage

Les différente fonctions, variables et autres éléments à nommer seront basés sur des mots de la langue anglaise. Dans le cas de nom composés tel que "my example" on notera la variable "my_example".

1.0.2 Présentation

Toujours par soucis de lisibilité, toutes nos productions auront des lignes composées au maximum de 80 caractères. Les fichiers de codes respecteront aussi l'indentation définie dans la section suivante.

Indentation

L'indentation sera faite à partir du caractère de Tabulation et non du caractère Espace. L'exemple suivant illustre le cas d'une fonction.

```
type my_function()
{
        type my_object

        if(my_object)
        {
                do_action()
        }
}
```

1.1 Commentaires

Dans un souci de productivité, de rapidité de codage et pour garantir la propriété intellectuelle du code à la Else Team, le code source sera commenté.

1.1.1 Entête des fichiers

L'entête des fichiers inclura une identification de celui-ci ainsi que le rappel des droits s'appliquant à ceux-ci.

Identification du fichier

```
File: rim/else.rb
Description: Description rapide du fichier
Projet: RimElse
```

Inclusion de la GNU GENERAL PUBLIC LICENSE (GPL)

```
Copyright (C) 2006 Else Team

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301, USA.
```

1.1.2 Fichier Ruby

Pour générer la documentation se rapportant aux parties de code en Ruby, nous utiliserons le générateur de documentation "rdoc"[1].

Rdoc générera à partir du code source une documentation sous forme de fichiers HTML, XML...Rdoc extraira automatiquement les définitions des classes, modules, méthodes et attributs. Il générera aussi l'arbre des différentes classes et de leurs dépendances.

Les commentaires suivants devront donc être ajoutés, pour les :

- Classes

```
# Author:: Else Team
# Copyright::  Copyright (c) 2006 Else Team
# License:: GNU General Public License version 2 (GPLv2)

# Description of the classe
# on multiple lines
```

- Méthodes

[1] http ://rdoc.sourceforge.net/

```
# Description of function
# on multiple lines
def function(text)
      @first_variable == other_function(text)
end
```

1.1.3 Autres langages

Dans les fichiers sources des autres langages, tel que le C, la documentation sera extraite à l'aide de balises doxygen incluses dans les commentaires. La documentation peut être générée dans plusieurs formats tels que : HTML (compressé ou non), LaTeX, RTF, PostScript, PDF. Il permet de tirer une multitude d'informations du code source dont seulement certaines seront utiles :

- Prototypes et documentation des fonctions, qu'elles soient locales, privées ou publiques.
- Liste des fichiers inclus
- Prototypes, documentation des classes et leurs hiérarchies

Exemple : le cas d'une fonction

```
/**
 * @brief description of function
 * @param parameter_name description of entry parameter
 * @return description of return
 */
```

Chapitre 2

Séquence de démarrage

La séquence de démarrage est le point le plus crucial de RimElse. C'est lors de cette étape que le système va devenir opérationnel. Pour ce faire nous aurons besoin premièrement d'un chargeur d'amorçage ainsi que d'une partition racine pour notre noyau. Dans un second temps, nous adapterons RimElse à son environnement en effectuant une reconnaissance du matériel ainsi qu'une initialisation des services. Pour rendre plus conviviales ces actions une animation au démarrage (bootsplash) sera mis en place. Enfin, RimElse lancera un serveur X définit dans la dernière section de ce chapitre.

2.1 Chargeur d'amorçage

Le démarrage du système requiert un chargeur d'amorçage (bootloader) qui s'exécute hors système d'exploitation. Il s'agit d'un programme appelé par le BIOS qui charge l'image du noyau d'un système d'exploitation dans la RAM. Le schéma suivant illustre ce séquencage.

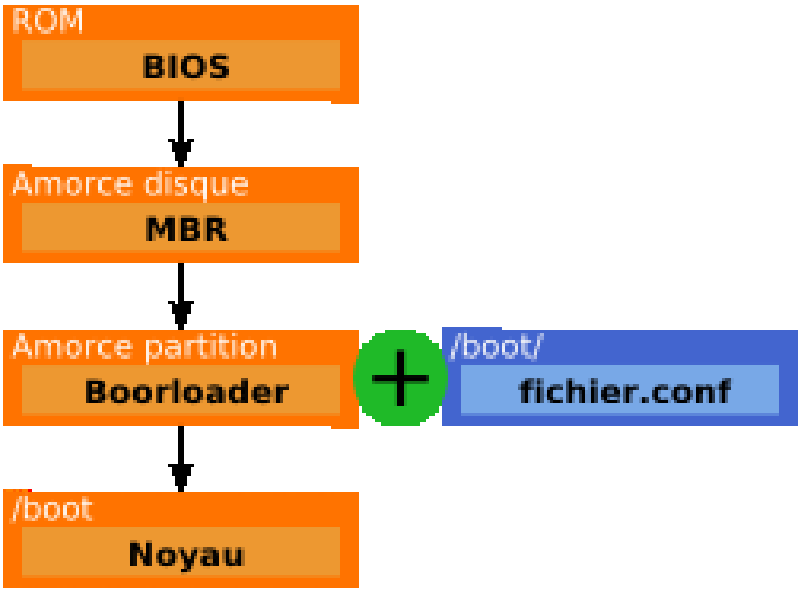

Fig. 2.1 – Chargeur d'amorcage

Dans le cadre de la RimElse, le chargeur d'amorçage doit être présent dans les premiers secteurs de notre support (clé USB, CD-ROM, ...).

Actuellement, trois solutions de bootloader sont abouties : Isolinux, LILO et Grub.

Isolinux

Isolinux qui fait partie du projet Syslinux, permet de charger un noyau Linux sans contrainte de taille. Mais celui-ci dispose d'un grand inconvénient, il n'accepte que les systèmes de fichiers ISO-9660 (CD-ROM). Or, RimElse a pour principal support une clé USB avec un système de fichiers différent et ne sera pas restreint à ce seul média et/ou système de fichiers.

LILO

LILO (LInux LOader) est un programme de multiboot et multisupports : il permet de charger plusieurs systèmes d'exploitations. Ce qui répond à nos besoins cependant il requiert une attention particulière lors d'une modification de la configuration et/ou changement de noyau.Ceci est du au fait que Lilo stocke tout simplement l'adresse et la taille du noyau à charger.

Grub

Acronyme de Grand Unified Bootloader, Grub est développé par la Free Software Foundation, n'a pas ces limitations, et est plus avancé. Grub est donc beaucoup plus adapté aux différents changements de configurations qui seront nombreux avec RimElse.

Aux vus des caractéristiques énoncées précédemment, Grub sera le bootloader de RimElse.

2.2 Initramfs

Le noyau Linux version 2.6 que nous allons utiliser dispose de trois solutions pour monter son "root".

La première, basique, s'effectue sans Initial Ramdisk (initrd). Pour monter sa partition root et exécuter /sbin/init, le noyau va chercher dans les paramètres qui lui auront été fournis par le bootloader.

Ce type d'initialisation se restreint à une configuration matérielle précise. Pour pallier à ce problème, l'initrd, est chargé au démarrage par le noyau. Dans ce système de fichier en mémoire vive, le noyau va exécuter /linuxrc. Celui-ci va s'occuper du montage du root (/), en fonction de l'architecture physique et lancer /sbin/init.

Le noyau 2.6 apporte une nouvelle solution. L'initramfs ajoute à l'initrd les avantages de la compression dans une archive cpio. De plus, l'archive cpio autorise un redimensionnement plus facile que le ramdisk. La mémoire utilisée peut donc être restreinte à son strict minimum.

2.3 Reconnaissance matérielle

Afin d'assurer la reconnaissance du matériel par le noyau nous allons devoir développer un script qui sera responsable de la configuration des périphériques PCI, ainsi que de la carte son ISA si installée. Dans un premier temps à l'aide de lspci on affiche les informations concernant chaque périphérique PCI soit ses identifiant Vendor (ID constructeur sur 4 caractères hexadécimaux) et device (ID périphérique sur 4 caractères hexadécimaux) ainsi que la classe du périphérique (catégorie de périphérique : storage, network, display, et memory).

Puis on lit le fichier /lib/modules/¡version du noyau¿/modules.pcimap dans lequel sont listées les correspondances entre les modules et les vendor ID. Ce qui permettra d'attribuer le module correspondant à chaque périphérique et de traiter celui-ci de manière appropriée. Le script permettra aussi de prendre en compte les paramètres passés par l'utilisateur lors du démarrage.

Dans le tableau suivant sont indiquées les classes et les types de contrôleur que le script traitera.

Classe	Type de Controlleur
0x0100	SCSI
0x0101	IDE
0x0102	Floppy
0x0c03	USB
0x0c00	IEEE1394
0x0200	Ethernet
0x0401	Son
0x0605	PCMCIA
0x0607	Cardbus

Une fois que les périphériques seront configurés, un test permettra de déterminer si le son fonctionne. Si ce n'est pas le cas le script essayera de trouver une carte de son en ISA. Le traitement des périphériques se fait classe par classe, le script utilisera une fonction qui cherchera les modules correspondant à un périphérique. Si un seul module est trouvé, il sera choisi par défaut sinon un programme "askUser" permettra de donner à l'utilisateur le choix du module à utiliser.

2.4 Init script

Une fois que le noyau est décompressé et il faut reveiller le systeme. Pour ce faire on va lancer le premier processus dit d'initialisation. Il s'agit du père de tous les processus, c'est lui qui va par la suite, lancer tous les autres programmes ou scripts qui se trouvent lister dans /etc/rc Il est possible de le vérifier en utilisant la commande ps dans un shell, on remarque que le processus Init a bien le premier PID et est le parent des autres.

```
elseteam@rimelse:/$ ps -eaf

UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 Apr04 ?        00:00:00 init [2]
root         2     1  0 Apr04 ?        00:00:00 [keventd]
root         3     1  0 Apr04 ?        00:00:00 [ksoftirqd_CPU0]
root         4     1  0 Apr04 ?        00:00:09 [kswapd]
root         5     1  0 Apr04 ?        00:00:00 [bdflush]
```

Init prend donc en charge la fin de la procédure de démarrage et se décompose en deux étapes principales.

2.4.1 Étapes

Le schéma suivant, illustre ces étapes qui sont commentées dans la suite de la section.

- Étape 1 :
  Le script d'initialisation va commencer par contrôler que la partion de démarrage ("/") n'a pas d'erreurs. Si necessaire le script d'init donnera un shell à l'utilisateur pour effectuer des opérations de maitenance. Ensuite, le script fera appel au script de reconnaissance matérielle décrit précédement et chargera les modules utiles au noyau. Cette étape ce finira par une suppression de tous les fichiers temporaires.

– Étape 2 :
Cette correspond au lancement des services de RimElse. Elle commence par initialiser les différents fichiers en supprimant le contenus des dossiers “/etc/initin/” et “/etc/initout”. Le script regénere ensuite le contenu de “/etc/initin/” en créant des liens vers les différents fichiers contenus dans “/etc/init/”.
Les fichiers initialisés, le script lit le contenu de “/etc/rc” qui contient tous les services devant être lancés au démarrage. Il interroge la fonction “dependances ?” des scripts de chaque service et contruit une table de dépendances correspondante. Si un service a été homit dans “/etc/rc”, le script d'initialisation le rajoutera à sa séquence de démarage.
La séquence de démarage crée, le script lance chaque service via leur fonction “start” et supprime le lien “/etc/initin/¡nom_du_service¿”. Un nouveau lien est crée a la place entre “/etc/init/¡nom_du_service¿” et “/etc/initout/¡nb_service¿.¡nom_du_service¿”. ¡nb_service¿ correspond à une valeure incrémentée à chaque démarrage d'un service.

2.4.2 Structure d'un de démarrage de service

Le script d'initialisation est codé en Ruby, il en sera donc de même pour les fichiers de démarrage des services. Ces derniers contiendrons au moins les trois methodes suivantes :
– “dependances ?” : recense les services dont celui-ci dépend.
– “start” : démarre le service
– “stop” : stop le service

2.5 Bootsplash

Bootsplash peut booter en différentes résolutions, dans le cas de RimElse, le mode par défaut sera le 800x600 16bits. Ce mode présente l'avantage d'exister sur 99.9% des combinaisons ordinateurs+écrans. Bootsplash dispose de modes de fonctionnement ”silent” où l'on masque la console et ”verbose” où l'on affiche cette dernière. Le passage de l'un à l'autre pourra se faire facilement durant la séquence d'initialisation.
Par défaut, le noyau linux ne supporte pas bootsplash, par conséquent, il est nécessaire de patcher celui-ci. Le patch est diponible sur le site bootsplash.org[1]. On notera aussi que chaque version du noyau dispose de son propre patch.

2.6 Démarrage du serveur X

La dernière tâche de notre script d'initialisation sera de lancer l'interface graphique qui n'est rien d'autre qu'un programme.
Le projet X.org est le plus utilisé sur Linux, il offre une implémentation libre et ouverte du standard X Window System, souvent abrégé en X11 ou X.
Avant de démarrer, le script d'initialisation demandera à l'utilisateur s'il souhaite conserver son ancienne configuration. Si l'utilisateur répond par la négative, un utilitaire de configuration sera lancé.
X.org fournit un outil de configuration appelé xorgcfg qui essaie de lancer Xorg -configure et lance ensuite le serveur X pour permettre des ajustements supplémentaires.
Pour ne pas se soucier des problèmes de bas niveau relatifs au matériel, nous utiliserons un tampon de mémoire vidéo. Celui-ci définit une abstraction logicielle d'accès aux périphériques vidéos qui correspond à la mémoire d'affichage de certains contrôleurs graphique et propose une interface unifiée aux logiciels.

[1] http ://www. bootsplash.org/index.html

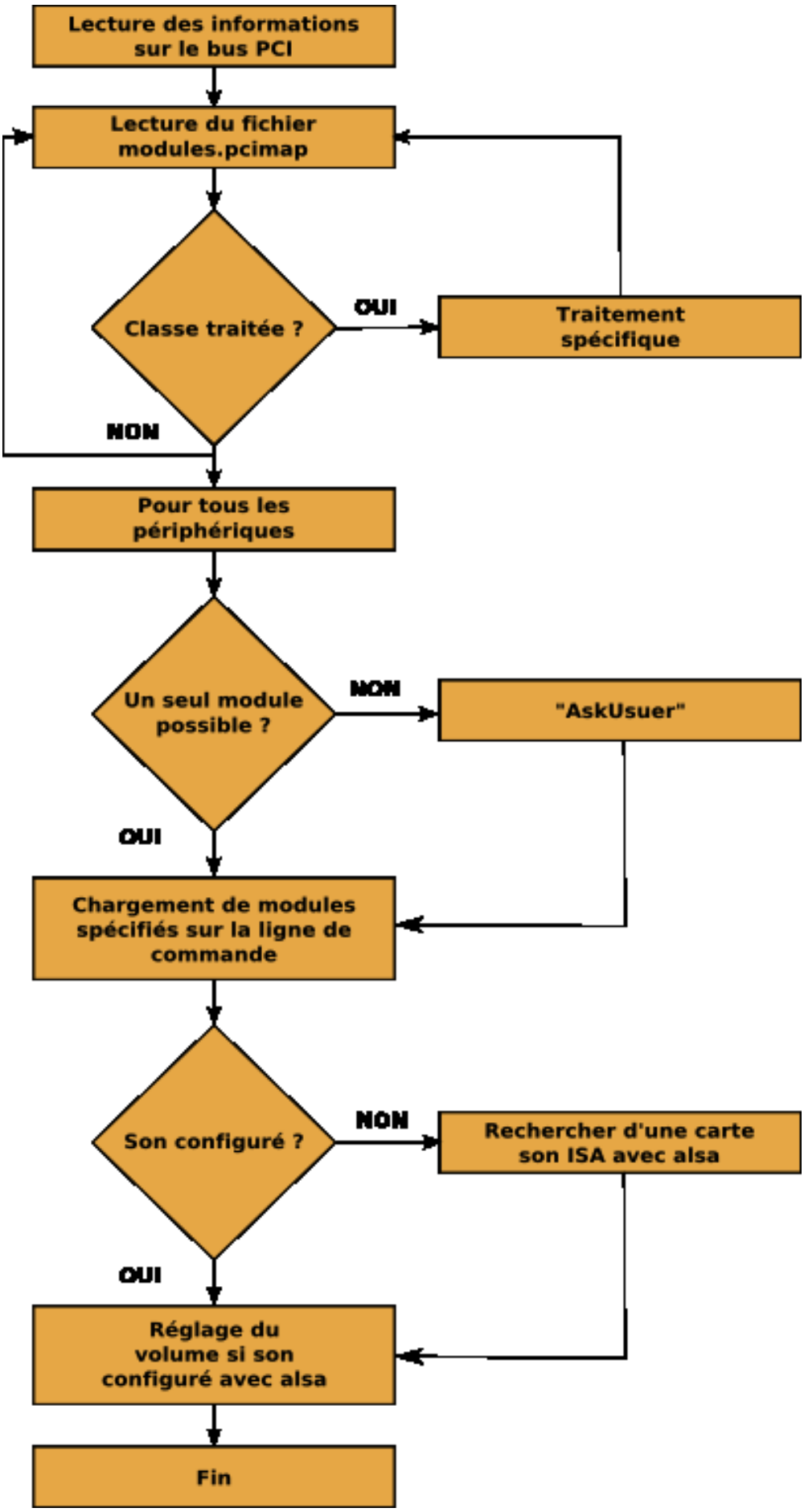

Fig. 2.2 – Ordonnancement de la reconnaissance matérielle

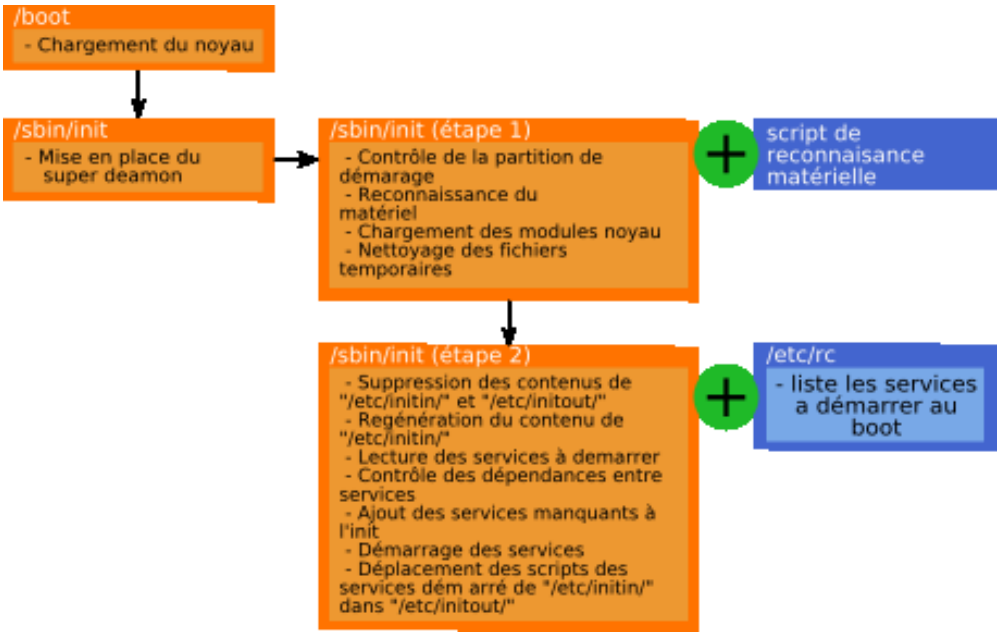

Fig. 2.3 – Script d'initialisation

Chapitre 3

Gestionnaire de paquets (rebgen)

3.1 Structure des paquets

Nous avons besoin de redéfinir un format bien particulier pour nos paquets. RimElse a besoin de légèreté. Pour ce faire nous avons besoin de réduire la taille des binaires, par conséquent, nous allons devoir opérer ce qui est appelé le strip. Stripper est un moyen de supprimer d'un fichier binaire tous symboles de débogues et informations inutiles pour l'utilisateur final. Ceci n'a aucune incidence sur le fonctionnement du logiciel.

Les paquets doivent forcément être compréssé. Pour des raisons de performances nous avons besoin d'un format qui compresse fortement mais qui reste léger durant la décompression.

Quelques tests de performance ont été réalisés sur les trois formats de compressions les plus utilisés dans le monde du logiciel libre.

Debian utilise gzip, les .deb sont des paquets comptactés avec tar et compréssé avec Gzip. Les rpm sont quant à eux des paquets compressés grâce à cpio. Les .deb et les .rpm sont les deux plus importants formats de paquets. Dans notre cas et aux vus des tests que nous avons réalisé sur des répertoires hétéroclites contenants des fichiers textes et des binaires ;

Le plus performant pour une utilisation mémoire et cpu réduitei, est à notre sens gzip.

Il allie un taux de compression élevé pour une consommation réduite.

Pour comparer les formats de compression. Voici un tableau récapitulatif d'un même répertoire compressé avec 3 méthodes, ce répertoire fait 21 Mo.

Type de compression	CPU	USER	SYSTEM	TOTAL	TAILLE
Gzip	48%	0,26s	0,09s	0,723	12 Mo
Bz2	98%	5,43s	0,10s	5,604	11 Mo
cpio	88%	0,03s	0,15s	0,209	15 Mo

Obtenir un paquet installable sur une distribution RimElse nécessite pour le paquageur 8 étapes :

- Étape 1 Configuration du paquet.
- Étape 2 Génération de la base de donnée
- Étape 3 Compilation.
- Étape 4 Épuration.
- Étape 5 Création de l'arborescence.
- Étape 6 Création du paquet.
- Étape 7 Signature.
- Étape 8 Publication.

3.1.1 Configuration du paquet

Cette phase de configurationi, est le moment où le développeur créé un fichier texte contenant toutes les informations utiles.

Il comprendra les parties qui sont les suivantes :

Information

- NAME Nom du paquet
- Soft-VERSION Version officielle du logiciel.
- Else-VERSION Version du packaging.
- DESCRIPTION Description du logiciel.
- Homepage Site web
- Else-Home Information sur le paquet sur le site RimElse
- License License du logiciel.

Dépendance

- DEPENDS Dépendances
- NULL Dépendance obligatoire
- **?** Dépendance optionnelle

```
? <Nom de la dépendance> (<paquet1>|<paquet2>) Description
Soit le paquet1 ou paquet2 sont nécessaire pour
cette dépendances optiotnnel
? <Nom de la dépendance> (<paquet1>&<paquet2>) Description
Soit le paquet1 et paquet2 sont nécessaire pour
cette dépendances optiotnnel
? <Nom de la dépendance> (<paquet1>^<paquet2>) Description
Soit le paquet1 ou ex paquet2 sont nécessaire pour
cette dépendances optiotnnel
```

- **!** Conflit

Dépendance

- PRE/POST-INSTALL Définie la suite de procédures à executer avant ou après l'installation. Certaines procédures courantes seront fournies par le gestionnaire. Comme par exemple la création d'un utilisateur ou le changement de droit d'un fichier ...

3.1.2 Génération de la base de donnée

Pour des raisons de rapidité d'accés à l'information, il serait pertinant de traduire la configuration décrite plus haut dans un fichier de binaire simple.

Il s'avère que la meilleure manière d'obtenir une information rapidement est de l'avoir dans une base de données.

C'est pourquoi il faudrait créer des tables simples contenants les informations.

Nous avons besoin d'un outil rapide, robuste et léger.

Diverses options s'offre à nous pour créer et lire cette "mini base de données" de configuration.

- 1 SQlite : Léger Robuste et répendu.
- 2 kirbybase : Léger, complétement en ruby (par ruby pour ruby), peut répendu
- 3 cdb : Rapide, simple d'utilisation.
- 4 etc ...

Au vus des differents retours de la communauté et de la presse spécialiséé, SQLite se pose comme une référence en la matière.

Cette petite bibliothèque en C propose un moteur de BDD SQL compatible SQL92. Avec l'énorme avantage de ne pas être construit sur le shéma client-serveur, mais sur une intégration au programme.

Voici un exemple nous montrant comment s'intègre SQLite dans un programme écrit en Ruby.

```
require 'sqlite'
db = SQLite::Database.new( "data.db" )

db.execute( "select * from table" ) do |row|
p row
end
db.close
```

Grâce à cette interface, nous pourrons avoir un accès rapide aux données via le gestionnaire de paquets.

3.1.3 Compilation

Une fois l'étape de configuration du paquet terminée, il faut compiler le binaire. Pour RimElse cette étape est très particulière car relativement complexe. En effet nous n'allons pas nous contenter de compiler une fois le logiciel. Nous allons le compiler autant de fois que nécessaire pour obtenir tous les supports édictés dans le paramètrage sous forme de patchs.

Pour tout autre support, il suffira d'ajouter une dépendance (optionnelle) dans le paramètrage.

A chaque dépendance optionnelle correspondra un patch qui sera la différence entre le logiciel minimal et l'application avec le support optionnel.

Ainsi toutes les possibilités offertes dans la déclaration des dépendances optionnelles seront couvertes.

3.1.4 Épuration

Cette étape consiste en l'épuration des fichiers pour n'avoir que le strict nécessaire. La plus part des scripts contiennent des commentaires, ces derniers sont très utiles pour la compréhension du script lorsqu'on le lit. En ce qui concerne son fonctionnement ces commentaires sont tout simplement inutiles, de plus ils augmentent considérablement la taille des fichiers. Un simple utilisateur n'en a aucun besoin, l'intérêt pour lui étant que le script fonctionne correctement.

Dans le cas des scripts (ruby, perl, python, shell, ...) cette étape consistera donc à supprimer tous les commentaires, toutes les lignes blanches et tout autre caractère inutile.

```
cat /usr/sbin/gaze | wc -c
  48835
cat /usr/sbin/gaze | grep -vE '^#' | git-stripspace | wc -c
  38593
```

Sur cet exemple on gagne 10242 caractères. Reproduit sur le nombre de scripts et de fichiers dits "pollués", l'espace gagné n'est pas négligeable.

Pour les binaires la même chose se reproduit, en effet nos binaires sont chargés de symboles de déboguage et autres éléments que nous qualifierons de "polluants" pour un système qui se charge entièrement en mémoire vive.

L'outil "GNU strip" retire tous les symbloles des fichiers binaires qui ne sont pas indispensables.

Ainsi pour les scripts nous pourrons gagner jusqu'a 50% d'espaces par rapport a un script non épuré.

3.1.5 Création de l'arborescence

Une fois le logiciel compilé avec tous ces patchs il ne reste plus qu'a créer l'arborescense du paquet.

```
./<nom_du_paquet_version>.dbc : configuration du paquet au format binaire.
./binaries : contient les binaires.
./scripts/pre_install : contient les scripts de pré-installation.
./scripts/post_install : contient les scripts de pré-installation.
```

3.1.6 Création du paquet

Lorsque toutes les étapes précédentes sont terminées il ne manque plus que la compression de l'arborescence dans le format proposé plus haut.

3.1.7 Signature

Cette étape permettera de fournir à l'utilisateur une signature du paquet. Pour des raisons de sécurité nous préférons signer les paquets, pour attester qu'ils ont été validés par la Else Team. Celà évitera que l'on reproche à l'équipe qu'un paquet ait brisé le système. L'équipe ne sera donc pas responsable de tout .reb non signé. Il reste à déterminer la méthode de signature.

3.1.8 Publication

Pour que le paquet soit publié, il faut dans un premier temps qu'il soit validé par la Else Team. Après validation, il sera mis à dispostion des utilisateurs via le site internet. Les utilisateurs pourront ainsi le télécharger et l'installer sur leur systéme.

3.2 Identification des patches

La phase de compilation étant relativement complexe nous allons revenir dessus. Chaque patch devras être identifié en fonction de ses composants. Cette identification de niveau de patch à appliquer sera crée gràce au fichier de base de données .dbc

La liste des dépendances optionnelles qu'il contient permettra de créer un identifiant unique pour chaque combinaison de support optionnel.

3.3 Gestion des dépendances

Les dépendances seront résolues grâce au fichier binaire de configuration traduit lors de l'étape deux de la création d'un paquet.

Un par un, le gestionnaire de dépendances va chercher les paquets cités dans le paramètrage. Après avoir vérifié qu'ils soient présents, il va lancer l'installation des binaires abscents.

3.4 Application des patches

L'utilisateur dispose donc d'un paquet "basique" avec le minimum de supports. Il va pouvoir appliquer la patch contenant les fonctionnalités désirées.

L'utilisateur va pouvoir choisir ses supports. Ainsi le système choisira le patch à appliquer.

Chapitre 4

Gestionnaire de configuration

RimElse est, comme nous l'avons vu, une distribution orientée utilisateur. Il faut donc fournir des outils pour faciliter son utilisation.

Le Gestionnaire de configuration fournira les différentes interfaces qui facilite la vie d'un utilisateur.

- Configuration réseau
- Configuration graphique
- Configuration des paquets
- Éditeur de configurations logiciels
- ...

4.1 Gestionnaire de paquets

Cet outil devra offrir la possibilité à l'utilisateur de choisir ses paquets et ses supports. Il choisira automatiquement les paquets nécessaires au bon fonctionnement de l'application. En fonction des choix des supports, l'outil déterminera le patch à appliquer.

4.2 Gestionnaire de configuration graphique

La RimElse sera doté d'un serveur 'X' afin de disposer d'interfaces graphiques. Cet outil permettera à l'utilisateur de choisir divers élements relatifs à l'affichage, comme la résolution, le pilote graphique utilisé, la fréquence de rafraîchissement de l'écran. ...

4.3 Éditeur de configurations

Les logiciels installés qui nécessiteront une configuration, seront fournis avec un module de configuration Else.

Ce module sera compris dans le paquet.

Une API sera mise à disposition par la Else Team afin de permettre le développement facile de ces modules de configuration.

Chapitre 5

Outils

Pour faciliter la gestion du projet au niveau des ressources humaines ainsi que des différentes productions, plusieurs outils sont à disposition des membres de la Else Team.

5.1 Communication

La Else Team dispose de divers outils essentiels pour la communication au sein du groupe. Elle a aussi mis en place des moyens d'information pour les utilisateurs et toutes personnes souhaitant prendre connaissance des activités de la Else Team et du projet RimElse.

5.1.1 Intra Else Team

gna.org

Else Team a choisi d'utiliser le site gna.org pour héberger, entre autre, la partie d'assignement de tâches. Gna.org est un centre de développement mettant à disposition des dépôts de code source, des espaces de téléchargement, des sites web, des listes de discussion et des outils de suivi(anomalies, tâches, support technique,patches). Via ce site le groupe dispose d'une "roadmap" en ligne accessible à tous. Chaque contributeur au projet RimElse peut ainsi choisir la partie sur laquelle il souhaite focaliser son temps et voir le travail restant.

"Mailing list"

Le site gna.org permet aussi la création de "mailing list". Actuellement, deux "mailing list" sont affectées au projet RimElse :

- else-devel@gna.com Cette "mailing list" est accessibles à tous les contributeurs du projet RimElse. Elle contient toutes les interrogations du groupe et les idées de développement.
- else-commit@gna.com Dans cette "mailing list" sont recensés tous les développements faits sur les différentes production par la Else Team. Elle est principalement constituée de de mails envoyés automatiquement par le système de gestion de code Git décrit dans la suite de ce document.

5.1.2 Communication externe

Pour la communication externe nous avons choisi d'utiliser un blogueur (Wordpress). WordPress est un logiciel de blog abouti qui permet à plusieurs auteurs de publier des billets, lesquels seront classés par date et par catégories. De multiples catégories, elles-même imbricables, peuvent être affectées à un billet donné. De plus, WordPress inclut un système de gestion des commentaires. Ainsi tout à chacun pourra laisser des commentaires sur les nouvelles publiées. WordPress permet également l'exportation de flux de syndication aux formats RSS. En outre, WordPress respecte nativement les standards du web XHTML et CSS.

La documentation concernant RimElse se trouvera sur un wiki dédié afin de structurer l'information pour permettre d'y naviguer plus commodément.

5.2 Développement

5.2.1 Git

Conclusion

Annexe A

SQLite

Source please
Citation wikipedia :

> SQLite est une petite bibliothèque écrite en C qui propose un moteur de base de données SQL et implémentant en grande partie le standard SQL92 et les Propriétés ACID. Contrairement aux serveurs de bases de données comme MySQL ou PostgreSQL, sa particularité est de ne pas reproduire le schéma habituel client-serveur mais d'être intégré directement aux programmes en utilisant des fichiers de bases de données. D. Richard Hipp, le créateur de SQLite, a choisi de distribuer cette bibliothèque dans le domaine public. Parmi ses autres spécificités, on notera : l'absence de procédure d'installation et de configuration. toute la base est stockée dans un seul fichier le type de chaque donnée stockée en base est une propriété de la donnée, pas de la colonne. Une colonne peut donc contenir des données de types différents. SQLite implémente la majorité de la norme SQL 92 à l'exception de : la gestion des droits avec GRANT et REVOKE la gestion des clés étrangères les jointures de type RIGHT OUTER JOIN et FULL OUTER JOIN - les triggers ne sont que partiellement pris en compte les possibilités de modifier la structure d'une table sont limitées : on peut renommer une table et y ajouter des colonnes, mais pas modifier ou supprimer de colonnes. SQLite peut se révéler intéressant au niveau des performances et être utile dans bien des cas (impossibilité d'utiliser un serveur de bases de données, pour des sites internet ou dispositifs et applications embarquées, ...), mais il ne permet pas à différents processus ou thread d'accéder en écriture à la même base de données et n'est donc pas conçu pour gérer de nombreux accès concurrentiels. La bibliothèque peut être utilisée en C et C++ mais des modules pour TCL et d'autres langages de scripts sont disponibles. Dans notre cas nous allons utiliser l'interface Ruby pour commander SQLite.

Table des figures

2.1 Chargeur d'amorcage 7
2.2 Ordonnancement de la reconnaissance matérielle 11
2.3 Script d'initialisation 12