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

7 mai 2006

Table des matières

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.

1.1 Nommage

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

1.2 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.

1.2.1 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()
        }
}
```

1.3 Commentaires

Dans une optique de productivité, de rapidité de codage ainsi que pour garantir la propriété intellectuelle du code à la Else Team, le code source sera commenté.

1.3.1 Entête des fichiers

Les en-têtes incluront une identification des fichiers ainsi que le rappel des droits s'y appliquant.

Identification du fichier

```
File: rim/else.rb
Description: Short file description
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.
```

1.3.2 Fichier Ruby

Pour fournir de la documentation se rapportant aux fichiers code Ruby, le générateur de documentation "rdoc"[1] est préconisé.

Rdoc génére à partir du code source une documentation sous forme de fichiers HTML, XML...Il extrait automatiquement les définitions des classes, modules, méthodes et attributs. Pour finir un arbre des classes et de leurs dépendances est créé.

Les commentaires suivants doivent 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

```
# Description of function
# on multiple lines
```

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

```
def function(text)
        @first_variable == other_function(text)
end
```

1.3.3 Autres langages

Dans les fichiers sources des autres langages, tel que le C, la documentation est extraite à l'aide de balises doxygen incluses dans les commentaires. La documentation peut être générée dans plusieurs formats tels que : HTML, LaTeX, RTF, PostScript, PDF. Il permet de tirer une multitude d'informations du code source qui 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, premièrement, un chargeur d'amorçage ainsi qu'une partition racine sont nécessaires au noyau. Dans un second temps, RimElse est adaptée 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) est mis en place. Enfin, RimElse lance un serveur X définit dans la dernière section de ce chapitre. D'autre part, la Else Team recommande l'utilisation du langage Ruby pour le développement des différents scripts 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 séquençage.

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 parti 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 dû 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, il n'a pas limitations des précédents et est plus avancé. Grub est donc beaucoup plus adapté aux différents changements de configurations qui seront nombreux avec RimElse.

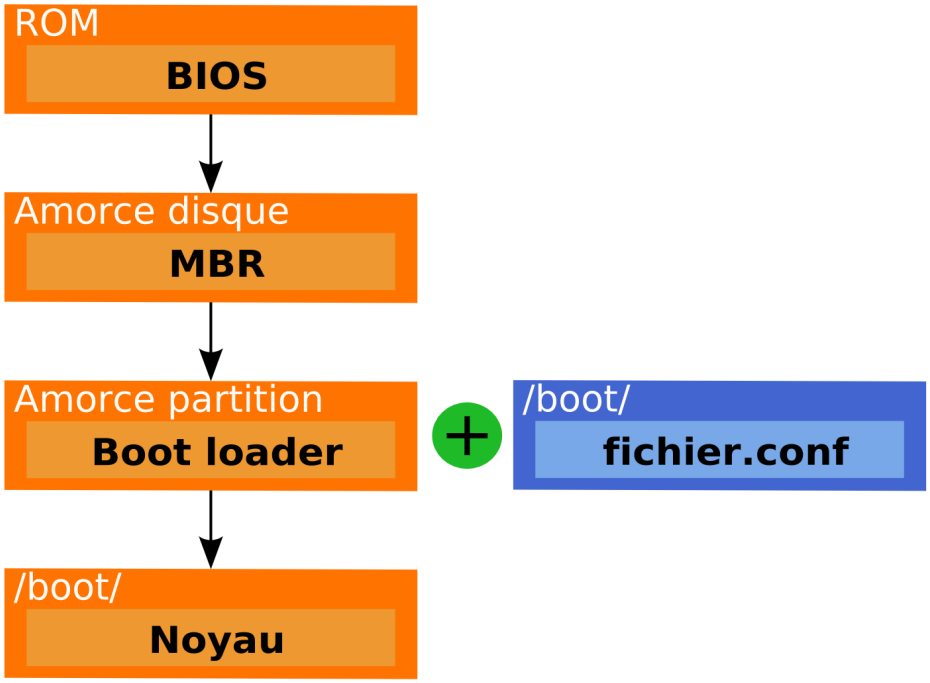

FIG. 2.1 – Chargeur d'amorcage

Aux vues des caractéristiques énoncées précédemment, Grub est le bootloader préconisé pour RimElse.

2.2 Initramfs

Le noyau Linux version 2.6 utilisé dispose de trois solutions pour monter sa racine ("root") ("/").

La première, basique, s'effectue sans Initial Ramdisk (initrd). Pour monter sa partition root et exécuter /sbin/init, le noyau va utiliser 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 fichiers en mémoire vive, le noyau va exécuter /linuxrc. Celui-ci va s'occuper du montage du root en fonction de l'architecture physique puis 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. De plus, cette archive 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, un script doit être développer. Il 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 la commande "lspci", les informations concernant chaque périphériques PCI sont recueillies. Les identifiants "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).

Le script 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 permet d'attribuer le module correspondant à chaque périphérique et de traiter celui-ci de manière appropriée. Le script peut aussi de prendre en compte les paramètres passés par l'utilisateur lors du démarrage.

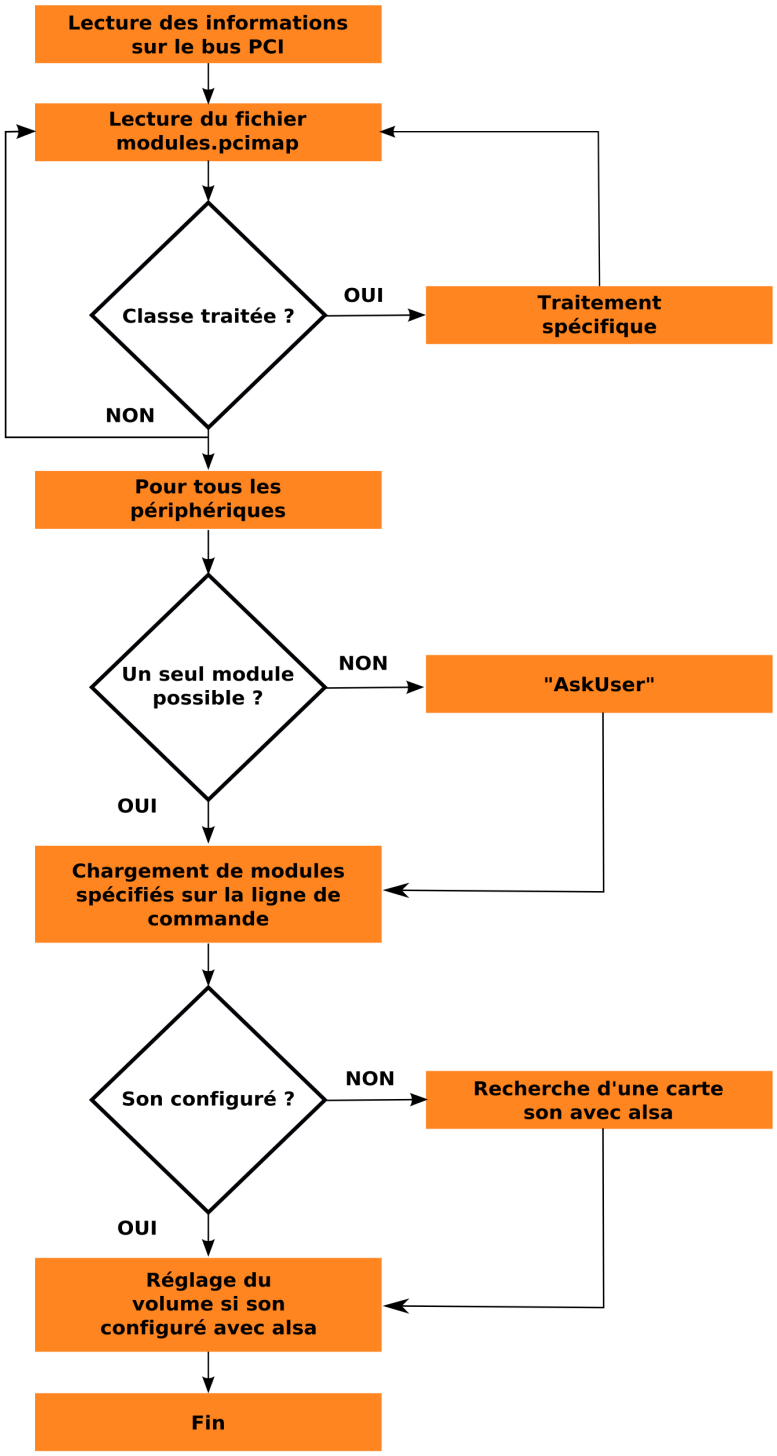

FIG. 2.2 – Ordonnancement de la reconnaissance matérielle

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

Tab. 2.1 – Classes et types de contrôleur traité lors de la reconnaissance matérielle

Une fois que les périphériques sont configurés, un test s'assure que le son fonctionne. Si ce n'est pas le cas, le script essayera de trouver une carte son sur un autre bus.

Le traitement des périphériques se fait classe par classe, le script utilise une fonction qui cherche les modules correspondant à un périphérique précédement détecté. Si un seul module est trouvé, il est choisi par défaut. Dans le cas contraire, un programme "askUser" permettra de donner à l'utilisateur le choix du module à utiliser.

2.4 Init script

Une fois le noyau décompressé, il faut réveiller le système. Pour ce faire, le premier processus dit d'initialisation est lancé. Il s'agit du père de tous les processus. C'est lui qui, par la suite, lance tous les autres programmes, ou scripts, qui se trouvent listés 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.

```
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]
```

Fig. 2.3 – Arbre des processus

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. Elles sont commentées dans la suite de la section.

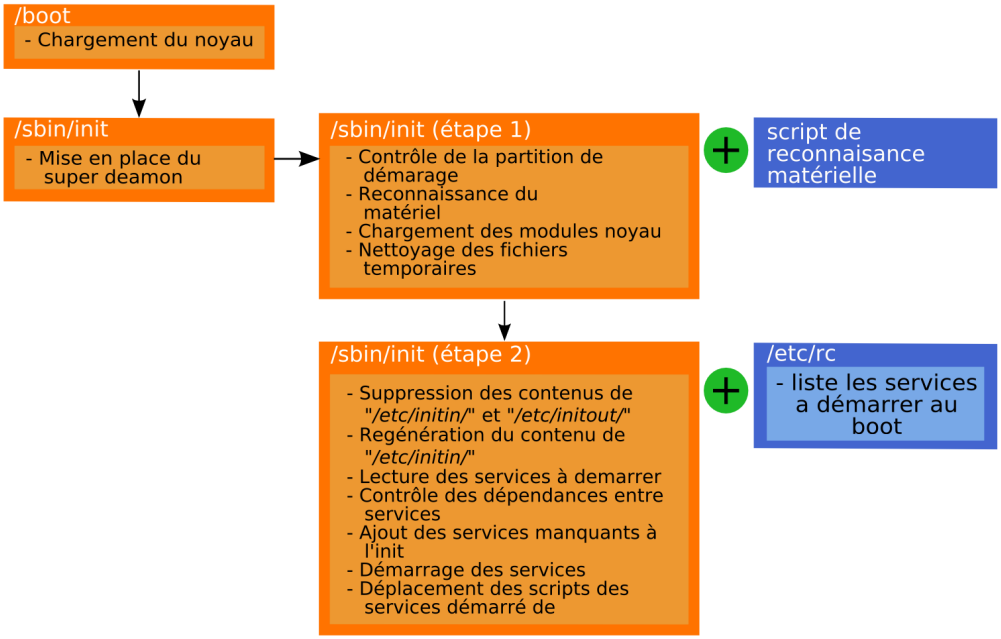

FIG. 2.4 – Script d'initialisation

– Étape 1 :
Le script d'initialisation commence par contrôler que la partion de démarrage ("/") n'est pas corrompue. Si nécessaire le script d'init donnera un shell à l'utilisateur pour effectuer des opérations de maintenance. Ensuite, le script fait appel à celui de reconnaissance matérielle décrit précédement et chargera les modules utiles au noyau. Cette étape se finie par une suppression de tous les fichiers temporaires.
– Étape 2 :
Cette phase 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 omis dans /etc/rc, le script d'initialisation le rajoutera à sa séquence de démarrage.
La séquence de démarrage crée, le script lance chaque service via leur fonction "start" et supprime le lien /etc/initin/`<nom_du_service>`. Un nouveau lien est crée à la place entre /etc/init/`<nom_du_service>` et /etc/initout/`<nb_service>.<nom_du_service>.<nb_service>` correspond à une valeur incrémentée à chaque démarrage d'un service.

2.4.2 Structure d'un script de démarrage de service

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

2.5 Bootsplash

Bootsplash peut s'éxecuter en différentes résolutions, dans le cas de RimElse, le mode initial sera le 800x600 16bits. Il 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 le patcher. Le patch est diponible sur le site bootsplash.org[1]. On notera aussi que chaque version du noyau dispose de son propre patch.

FIG. 2.5 – Visualisation du bootsplash de RimElse

2.6 Démarrage du serveur X

L'une des dernières 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 X.

La configuration du X11 sera regénérée à chaque démarrage, sauf si l'utilisateur ne l'a pas voulu. X.org fournit des outils de configuration automatiques. 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

Chapitre 3

Génération et gestion des paquets

RimElse, avec ses besoins de légéreté, nécessite la redéfinition d'un format de paquets. Rebgen, le générateur de paquet de RimElse a pour fonction de générer des paquets .reb. Cet outil devra être développé en langage Ruby.

3.1 Format des paquets

Les paquets sont des conteneurs de données (fichiers binaires, de configuration,...). Pour certains logiciels la taille des paquets peut être très importante. L'efficacité de ces derniers peut s'avérer fortement réduite par cette contrainte. Un moyen d'optimiser les performances est de compresser.

Il existe plusieurs formats de compression. 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 compactés avec tar et compressés 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.

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

Tab. 3.1 – Tableau comparatif récapitulant les tests de compression

Les tests précédents ont été réalisé sur des répertoires hétéroclites de 21 Mo contenants des fichiers textes et binaires ; le plus performant pour une utilisation mémoire et cpu réduite, s'avère être gzip. Ce dernier, allie un taux de compression élevé pour des consommations réduites.

3.2 Création d'un paquet

L'obtention d'un paquet installable sur une distribution RimElse nécessite pour le paquageur 8 étapes :

1. Configuration du paquet
2. Génération de la base de données
3. Compilation
4. Épuration
5. Création de l'arborescence
6. Création du paquet

7. Signature
8. Publication

3.2.1 Configuration du paquet

Cette phase de configuration est le moment où le développeur crée un fichier texte contenant toutes les informations nécessaires.

Il comprend les parties suivantes :

Informations

- 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.

Dépendances

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

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

- **! :** Conflit

Installation

- PRE/POST-INSTALL : Définie la suite de procédures à exécuter 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

3.2.2 Génération de la base de données

L'accès à l'information décrite précédement, pourrait être optimisée par le stockage de la configuration dans un fichier binaire simple. À ce jour, la meilleure manière d'obtenir une ressource rapidement est de la stocker dans une base de données (BDD). Celle-ci doit contenir les informations dans des tables simples.

La création et la lecture de cette "mini base de données" requiert un outil rapide, robuste et léger. Diverses solutions existent :

- SQLite : Léger, robuste et répandu.
- Kirbybase : Léger, complétement en Ruby (par Ruby pour Ruby), peu répandu
- cdb : Rapide, simple d'utilisation.

Au vus des différents retours de la communauté et de la presse spécialisée, 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 schéma client-serveur, mais sur une intégration au programme.

(La définition de Wikipedia est disponible à la page 21.)

Exemple 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
```

Cette interface autorise un accès rapide aux données via le gestionnaire de paquets.

3.2.3 Compilation

Une fois l'étape de configuration du paquet terminée, le binaire est compilé. Pour RimElse cette étape est très particulière car relativement complexe. En effet, Rebgen ne peut pas se contenter de compiler une fois le logiciel. Il devra compiler les sources autant de fois que nécessaire pour obtenir tous les supports édictés dans le paramètrage sous forme de patchs binaires.

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.

3.2.4 Épuration

Cette étape consiste en l'épuration des fichiers pour n'avoir que le strict nécessaire. La plupart des scripts contiennent des commentaires, ces derniers sont très utiles pour leur compréhension. 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 un gain de 10242 caractères est obtenu. 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 les binaires sont chargés de symbroles de débogage et autres éléments que la Else Team qualifie de "polluants" pour un système qui se charge entièrement en mémoire vive.

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

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

3.2.5 Création de l'arborescence

Une fois le logiciel compilé avec tous ces patchs, il ne reste plus qu'à 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 post-installation.
```

3.2.6 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. Les scripts de pré et post installation définis dans le fichier de paramètres devront également y être inclus.

3.2.7 Signature

Cette étape permet de fournir à l'utilisateur une signature du paquet. Pour des raisons de sécurité, la signature des paquets attestera qu'ils ont été validés par la Else Team. Cela évitera que l'on reproche à l'équipe la corruption du système par un paquet .reb non signé. Il reste à déterminer la méthode de signature.

3.2.8 Publication

Le paquet sera publié par la Else Team après sa validation. Il sera ensuite mis à dispostion des utilisateurs via le site internet de RimElse. Les utilisateurs pourront ainsi le télécharger et l'installer sur leur système.

3.3 Identification des patchs

La phase de compilation étant relativement complexe, elle impose d'identifier chaque patch en fonction de ses composants. Cette identification au niveau des patchs à appliquer sera créée grâce 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.

3.4 Gestion des dépendances

Les dépendances sont résolues grâce au fichier binaire de configuration traduit lors de l'étape 2 de la création d'un paquet.

Un par un, le gestionnaire de dépendances cherche les paquets cités dans le paramètrage. Après avoir vérifié qu'ils sont présents, il lance l'installation des binaires absents.

3.5 Application des patchs

L'utilisateur dispose donc d'un paquet "basique" avec le minimum de supports. L'application d'un patch ajoutera les fonctionnalités désirées.

Au moyen du gestionnaire de paquets, l'utilisateur pourra choisir ses supports, ainsi le système choisira et appliquera le patch correspondant.

Chapitre 4

Gestionnaire de configuration

RimElse est, comme nous l'avons vu, une distribution orientée utilisateurs. Elle doit donc fournir des outils pour faciliter son utilisation.

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

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

4.1 Gestionnaire de paquets

Cet outil offre la possibilité à l'utilisateur de choisir ses paquets et ses supports. Il choisit automatiquement les paquets nécessaires au bon fonctionnement de l'application ainsi que les patchs à appliquer.

4.2 Gestionnaire de configuration graphique

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

4.3 Éditeur de configurations

Les logiciels installés nécessitant une configuration, seront fournis avec un module de configuration Else. Ce module sera compris dans le paquet et viendra s'ajouter dans l'interface de configuration.

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 intra Else Team

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 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,patchs). 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.

5.1.2 Liste de diffusion

Le site gna.org permet aussi la création de liste de diffusion. Actuellement, deux listes sont affectées au projet RimElse :

- else-devel@gna.org : Cette liste est accessible à tous les contributeurs du projet RimElse. Elle contient toutes les interrogations du groupe et les idées de développement.
- else-commits@gna.org : Des courriers électronique sont envoyés automatiquement sur cette liste lorsque qu'une modification dans les développement est faite. Cette liste informe de tous les développements faits sur les différentes productions par la Else Team. Elle est principalement constituée de de mails envoyés automatiquement par le système de gestion de version Git décrit dans la suite de ce document.

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

Toutes les productions seront centralisées sur un serveur de gestion de version (Source Code Manager - SCM). Le SCM utilisé est "git". Il gère les accès concurrent sur un même fichier. Il sera très utile afin

de garder un historique de toutes les modifications apportées. De plus, “git” est un SCM distribué ce qui permet de créer et de gérer des branches de façon performante, rapide et simple.

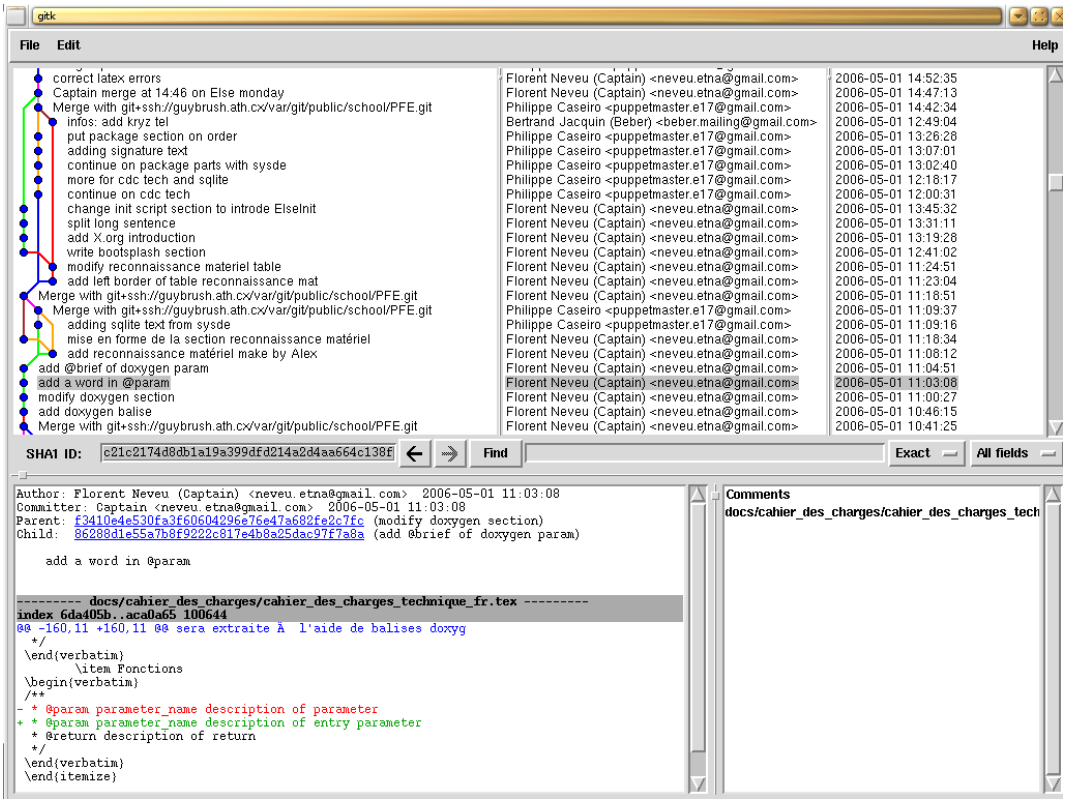

Fig. 5.1 – Visualisation de l’arbre de la Else Team

Conclusion

L'utilisation de la norme de codage définie dans la première section de ce cahier des charges technique, permettra d'élaborer la distribution RimElse. De plus, l'utilisation du language Ruby rendra les scripts performants et légers. RimElse bénéficie également d'une méthode d'initialisation originale pour un système Linux. Ensuite, lors de la génération des paquets, le procédé d'épuration et de compression permettra une optimisation de ces derniers. Le système de gestion des patchs binaires offrira la possiblité à l'utilisateur d'avoir des logiciels performants. Il ne sera plus encombré par les supports inadapté à ses besoins. Rebgen offrira, quant à lui, une solution simple et performante pour générer les paquets.

Pour finir l'utilisateur pourra configurer sa RimElse via les différentes interfaces du gestionnaire de configuration. Son, graphismes, logiciels, RimElse sera tout aussi configurable qu'une distribution classique, peut être même plus.

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

Liste des tableaux