



open wide
INGENIERIE

Spécifications techniques des
outils, partie 2

Version: 1.0

Auteur: Frédéric Ferrandis

- PROJET RTEL4I -



Spécifications techniques des outils (L1.2-b_2)





open wide
INGENIERIE

Spécifications techniques des
outils, partie 2

Version: 1.0

Auteur: Frédéric Ferrandis

MODIFICATIONS

<i>VERSION</i>	<i>DATE</i>	<i>AUTEUR(S)</i>	<i>DESCRIPTION</i>
1.0	7/09/2009	F.Ferrandis	Création



Table des matières

Introduction.....	4
1.1.Objectif du document.....	4
2.Fonctions « projets » :.....	5
2.1.Création d'un nouveau projet.....	6
2.2.Sauvegarde modification de projet.....	8
2.3.Duplication de projet.....	9
2.4.Destruction de projet.....	9
3.Fonctions « Lecture informations ».....	10
3.1.Lecture des cartes validées.....	10
3.2.Lecture des crosscompiler validés.....	11
3.3.Lecture des kernel validés.....	12
3.3.1Lecture des associations « configuration kernel - description ».....	13
3.3.2Récupération de la liste des fichiers de configuration de base d'une version de noyau.....	13
3.3.3Lecture des informations d'extensions temps-réel.....	14
3.3.3.1.Lecture des informations d'extensions temps-réel xenomaï/adeos.....	14
3.3.3.2.Lecture des informations d'extensions temps-réel preempt-rt.....	15
4.Fonctions « Production de composant ».....	17
4.1.Ajout d'une nouvelle carte.....	19



Introduction

1.1. Objectif du document

Le document présenté a pour objectif de décrire l'ensemble des fonctionnalités que devra couvrir le logiciel RTEL4I. Cet ensemble sera divisé en deux parties distinctes :

- les fonctions afférentes à la création de composants tel que :
 - une carte
 - un cross-compileur
 - un root-filesystem
 - le noyau
 - les extensions temps-réel
 - ...
- les fonctions afférentes à l'instrumentation des composants produits, par exemple :
 - exécuter le profiling d'un noyau
 - démarrer une session de débogage via qemu
 - analyser les fuites mémoires via valgrind
 -

Cette division se justifie par la nature des fonctions, et également par la présentation graphique [note : insertion d'une image] retenue.

Pour chaque sous-ensemble, une distinction supplémentaire sera faite entre les fonctionnalités externes, qui forment le tout cohérent visible pour l'utilisateur, et le moyen de les réaliser, à savoir les fonctionnalités internes.



2.Fonctions « projets » :

Les fonctions projets vont définir comment une entité projet est gérée. Outre la description des fonctions basiques (new, destroy, duplicate, save), l'analyse va permettre de rendre compte des informations qui devront être sauvegardé et de la mécanique dans l'enchaînement des actions.



2.1.Création d'un nouveau projet

identifiant	create_new_project
visibilité	externe
acteur	utilisateur
dépendances	
déclencheur	Action de click sur bouton ou bien ctrl+N
paramètre	aucun
Retour	0 si par d'erreur

L'un des objectifs majeurs du projet RTEL4I est la gestion de projet, afin de reproduire un environnement de la manière la plus simple qui soit. Pour cela, toutes les informations afférentes à un projet seront stockées dans un fichier XML, que l'utilisateur pourra conserver (1 projet = 1 fichier XML)

Le fichier XML devra contenir toutes les informations nécessaires à la reproduction d'un projet. L'analyse technique décrira d'avantage les informations obligatoires, mais en clair, sont nécessaires :

- Informations sur la machine de production : il est nécessaire qu'un utilisateur sache sur quelle machine de son parc de production a été généré son projet (utilisation de gethostname + lecture de /proc/cpuinfo, voir utilisation de la fonction sysinfo ou de la fonction C uname)
- Informations utilisateurs
 - lieu de sauvegarde des composants
 - uid, gid
 - nom
- Informations sur le programme RTEL4I:
 - version de rtel4I
 - plugins installés
 - plugins actifs
- Informations sur le projet :
 - description du projet
 - date de création
 - carte associée



- cross compilateur associé
 - date de création si cross_compilateur créé par utilisateur
- kernel associé
 - date de création si kernel créé par utilisateur
- rootfs associé
 - date de création
 - fichier de configuration

Une fois le projet créé, l'utilisateur peut accomplir les actions suivantes :

- créer une nouvelle carte
- créer un nouveau cross_compilateur pour une carte
- créer un kernel par un cross_compilateur
- créer un rootfs par un cross_compilateur.
- utiliser une carte existante
- utiliser un cross_compilateur existant
- utiliser un projet existant
- utiliser un kernel existant

L'utilisateur aura en visibilité :

- toutes les cartes certifiées par RTEL4I en couleur noir, et toutes les cartes créées par l'utilisateur en rouge.
- Idem pour les cross_compilateur
- idem pour les kernels

Le projet fait une distinction entre ce qui est produit par l'utilisateur et ce qui est fourni avec le logiciel RTEL4I. Cette distinction se fait via le biais de fichier XML séparé mais à la structure identique : un fichier usine, sur lequel nous reviendrons plusieurs fois, et contenant l'ensemble des éléments proposés et certifiés par RTEL4I (noyau, cc, carte, extension rt) et un (ou plusieurs) fichier utilisateur, contenant les mêmes éléments, mais non-certifiés par RTEL4I, correspondant aux créations de composants (noyau, cc, carte, extension rt) de l'utilisateur.

Ces parties seront notamment abordés dans le chapitre « Création de composants »



2.2. Sauvegarde modification de projet

identifiant	save_project
visibilité	externe
acteur	utilisateur
dépendances	
déclencheur	Action de click sur bouton ou bien ctrl+S
paramètre	aucun
Retour	0 si par d'erreur


Un projet n'a besoin d'être sauvegardé que sur conditions suivantes :

1. la machine de production n'est plus la même
2. l'utilisateur a fait la demande explicite de re-sauvegarder son projet
3. un des composants de rtel4i a été modifié
4. l'utilisateur a changé quelque chose au niveau du cross_compilateur
5. l'utilisateur a changé quelque chose au niveau du noyau
6. l'utilisateur a changé quelque chose au niveau du root-fs.

Le cas (3) entraîne la recompilation du cross_compilateur si l'utilisateur a changé la recompilation du kernel et du rootfs.

Le cas (4) entraîne la recompilation du kernel.

Le cas (5) entraîne la recompilation du rootfs.

	Spécifications techniques des outils, partie 2	Version: 1.0 Auteur: Frédéric Ferrandis
---	--	--

2.3. Duplication de projet

identifiant	duplicate_project
visibilité	externe
acteur	utilisateur
dépendances	
déclencheur	Action de click sur bouton
paramètre	aucun
Retour	0 si par d'erreur

Description : Il doit être possible de créer un nouveau projet à partir d'un projet existant. Cela doit se faire à l'aide de cette fonction.

2.4. Destruction de projet

identifiant	destroy_project
visibilité	externe
acteur	utilisateur
dépendances	
déclencheur	Action de click sur bouton
paramètre	aucun
Retour	0 si par d'erreur

Description : l'utilisateur peut souhaiter supprimer un projet. Cela supprimera le fichier XML, et le contenu du répertoire (sur confirmation de l'utilisateur) du projet



3.Fonctions « Lecture informations »

3.1.Lecture des cartes validées

identifiant	read_card_list
visibilité	interne
acteur	N/A
dépendances	N/A
déclencheur	Appel de la fonction
paramètre	aucun
retour	Une liste de carte, ou NULL en cas d'erreur

Description: RTEL4I est un logiciel, qui outre les fonctionnalités qu'il offre et/ou package, apporte un certain nombre de composants validés. Chaque version de RTEL4I permettra l'ajout de nouvelles cartes ayant été validées.

Cet ensemble de cartes constitue une partie des paramètres usines de RTEL4I. Elles sont décrites dans un fichier XML. A titre d'illustration, un exemple de description de carte, au sens XML du terme, est donné ci-dessous. Le fichier sera clairement décrit dans l'analyse technique.

```
<?xml version='1.0'?>
...
...
<carlist>
  <card id='1' name='card_name'>
    <date_creation>20100808</date_creation>
    <description>describe card here</description>
    <hwinfo>
      <arch type='arm'/>
      <cpu type='arm'/>
      <abi type='eabi' subtype='apcs-gnu'/>
      <tune type='arm'/>
    </hwinfo>
    <hwinfo>
    <crosscompilerlist>
      <crosscompiler idref='1'/>
      <crosscompiler idref='7'/>
    </crosscompilerlist>
    <kernellist>
      <kernel idref='37'/>
    </kernellist>
  </card>
</carlist>
```

....



....
<!-- fin du xml -->

La fonction interne va lire l'ensemble des noeuds « card » du fichier. Les informations récoltées sont destinées à apparaître graphiquement sur l'interface.

3.2.Lecture des cross-compiler validés

identifiant	read_cc_list
visibilité	interne
acteur	N/A
dépendances	N/A
déclencheur	Appel de la fonction
paramètre	aucun
retour	Une liste de cc, ou NULL en cas d'erreur

Description : De la même manière que précédemment, tous les cross-compilateurs validés par rtel4i seront lus depuis le même fichier « usine ».

A titre d'illustration, un exemple de description de cross-compilateur, au sens XML du terme, est donné ci-dessous. Le fichier sera clairement décrit dans l'analyse technique.

```
<?xml version='1.0'?>
...
<cardlist>
...
</cardlist>
<crosscompilerlist>
  <crosscompiler>
    <target type='powerpc'/>
    <abi type='eabi'/>
    <endian type='little'/>
    <flags>
      <cpuflags>...</cpuflags>
      <tuneflags>...</tuneflags>
      <archflags>...</archflags>
      <cflags>...</cflags>
    </flags>
    <binutils>
      <version>X.X.X</version>
    </binutils>
  </crosscompiler>
</crosscompilerlist>
```



```

<kheader>
  <version>X.X.X</version>
</kheader>
<options>
  <usecpp val='false'/>
  <usejava val='false'/>
</options>
<libc type='glibc'>
  <version>X.X.X</version>
  <thread type='ntpl'/>
</libc>
<cc>
  <version>X.X.X</version>
</cc>
</crosscompiler>
</crosscompilerlist>

```

3.3. Lecture des kernel validés

identifiant	read_kernel_list
visibilité	interne
acteur	N/A
dépendances	N/A
déclencheur	Appel de la fonction
paramètre	aucun
retour	Une liste de kernel, ou NULL en cas d'erreur

Description : De manière similaire, une liste de kernel apparaît dans le fichier usine.

```
<?xml?>
```

```

<cardlist>...</cardlist>
<crosscompilerlist>...</crosscompilerlist>
<kernellist>
<kernel id='X' build_by='id_cc_ref' >
<version>2.6.22</version>
<rt_extension type='preempt'>
</rt_extension>
<kernel_config filepath='RTEL4I_ROOT/usr/share/kernelconfigs/config_XX_YY_ZZ_ww'/>
</kernellist>

```



3.3.1 Lecture des associations « configuration kernel - description »

identifiant	read_kernel_defconfig_xml
visibilité	interne
acteur	
dépendances	
déclencheurs	démarrage
paramètre	aucun
retour	Un ensemble de {arch, kstart, kend, config_name, description }

Description : L'objectif majeur de RTEL4I est de masquer un grand nombre de complexités inhérentes à la création de kernel. Un répertoire de source du noyau contient, pour chaque architecture, un très grand nombre de configuration prêt à l'emploi, sur lesquelles il n'y aura guère que quelques détails à mettre au point. Le soucis d'un utilisateur est de savoir concrètement pour une architecture donnée, quelle est la configuration de base à choisir. Le fichier xml usine contiendra des informations supplémentaires :

```
<kh_conf_base>
  <arch type='powerpc'
    <kh_conf kstart='2.6.0' name='holly_defconfig'> use this configuration if cpu
type is XXX and card is YYYY </kh_conf>
  </arch>
</kh_conf_base>
```

Pour bien comprendre l'utilité de cette fonction, mettons-nous dans le contexte d'un utilisateur souhaitant créer une nouvelle carte. Cet utilisateur n'a que des vagues connaissances de sa carte, et arrive au moment de la configuration du noyau. Plutôt que de devoir tout configurer à la main, une première étape lui permettra de choisir parmi une liste de configuration prête à l'emploi, celle dont la description est la plus proche de sa connaissance de la carte. Ce point sera évoqué de nouveau dans le paramétrage du kernel.

3.3.2 Récupération de la liste des fichiers de configuration de base d'une version de noyau

identifiant	read_kernel_defconfig
visibilité	interne
acteur	



dépendances	
déclencheurs	Lors de la configuration d'un nouveau kernel
paramètre	aucun
retour	Un ensemble {arch, config_name }

Description : RTEL4I utilise fortement la notion de fichier XML dans un souci d'extensibilité. Un utilisateur, durant la phase de configuration d'un noyau, peut avoir la possibilité de choisir sa propre version de noyau. Il n'est pas certain que cette version de noyau ait les mêmes fichiers de configuration de base, que ceux renseignés dans le fichier de configuration usine. Le but est donc de récupérer pour une architecture donnée, l'ensemble de fichier de configuration (dans arch/\${arch}/configs) et de les matcher avec ceux trouvé dans le XML

3.3.3 Lecture des informations d'extensions temps-réel

identifiant	read_rtext_compatibility_list
visibilité	interne
acteur	
dépendances	read_rtext_compatibility_xeno read_rtext_compatibility_preempt
déclencheurs	Démarrage de l'application
paramètre	aucun
retour	Un ensemble de triplet {kvers,arch,rtext}


Description : RTEL4I dispose d'un certain nombre de ressources. Parmi celles-ci, les patches d'extensions temps-réel xenomai et preempt-rt.

L'objectif de cette fonction est de lire pour chaque architecture, les versions de noyaux auxquelles peuvent s'appliquer le patch.

Pour cela, les fonctions spécialisées read_rtext_compatibility_xeno et read_rtext_compatibility_preempt vont être appelée

3.3.3.1. Lecture des informations d'extensions temps-réel xenomai/adeos

identifiant	read_rtext_compatibility_xeno
visibilité	interne
acteur	

	Spécifications techniques des outils, partie 2	Version: 1.0 Auteur: Frédéric Ferrandis
---	--	--

dépendances	
déclencheurs	Démarrage de l'application
paramètre	aucun
retour	Un ensemble de triplet {kvers,arch,rtext}

Description : Parmi les ressources installées avec RTEL4I, un répertoire adeos/patches sera fourni. Ce répertoire contiendra l'ensemble des patches adeos par architecture et contiendra l'arborescence suivante :

adeos

```

|__ patches
    |__ 2.6
        |__ x86
            |__ adeos-ipipe-2.6.24-rc6-x86-2.0-02.patch
            |__ adeos-ipipe-2.6.26-x86-2.0-16.patch
        |__ arm
  
```

...

Cet arborescence sera le miroir de celle présente sur le site :

<http://download.gna.org/adeos/patches/v2.6>


La récupération des triplets se fera donc de la manière suivante :

- pour chaque architecture
 - lire le nom du fichier patch
 - extraire version du noyau du nom du fichier

Se faisant, la fonction pourra retourner l'ensemble des triplets.

3.3.3.2. Lecture des informations d'extensions temps-réel preempt-rt

identifiant	read_rtext_compatibility_preempt
visibilité	interne
acteur	
dépendances	
déclencheurs	Démarrage de l'application

	Spécifications techniques des outils, partie 2	Version: 1.0 Auteur: Frédéric Ferrandis
---	--	--

paramètre	aucun
retour	Un ensemble de triplet {kvers,arch,rtext}

Description : En se basant sur le site officiel
https://rt.wiki.kernel.org/index.php/CONFIG_PREEMPT_RT_Patch
 l'idée est de reproduire la même arborescence que précédemment.

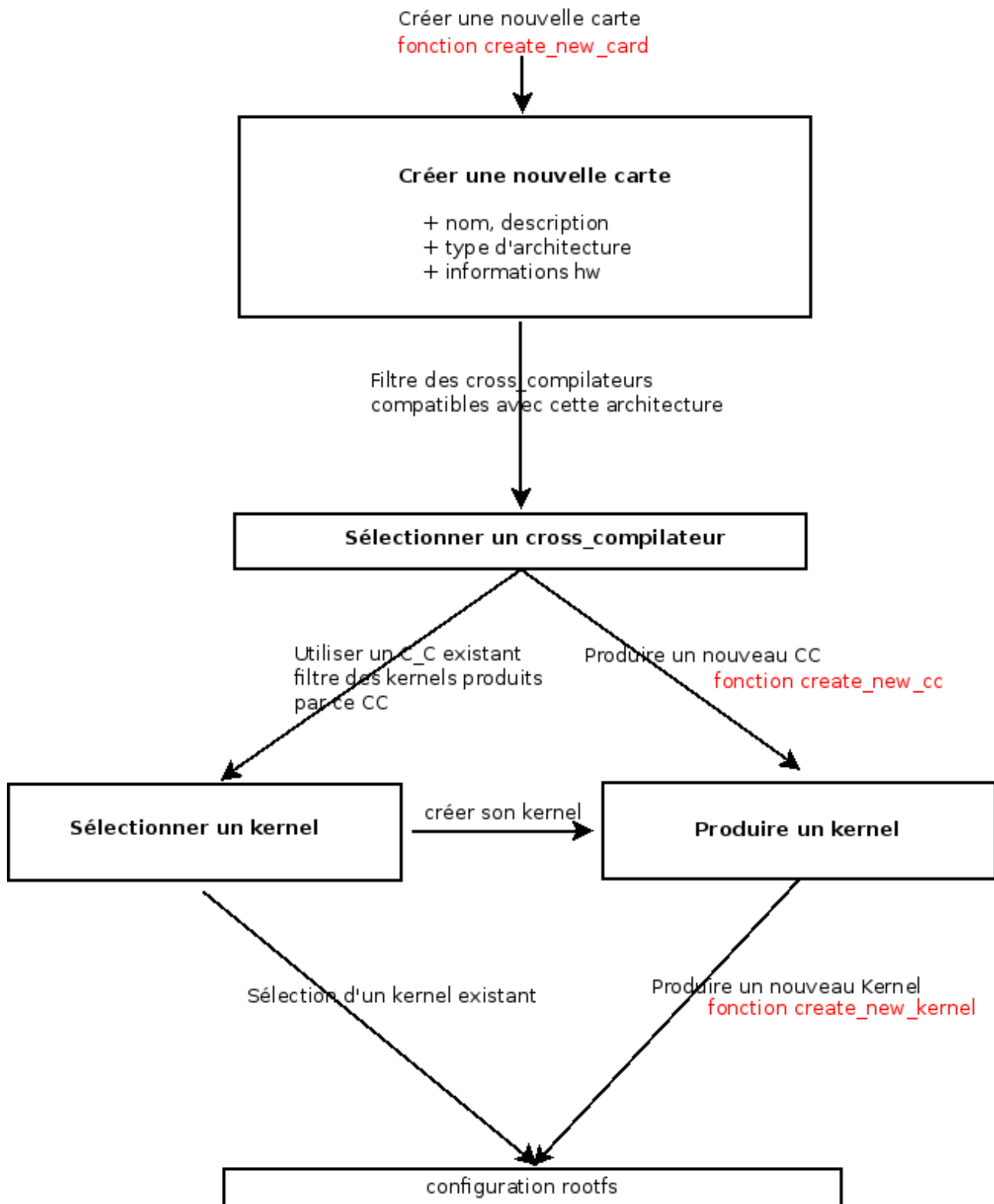


4.Fonctions « Production de composant »

Le gros des fonctionnalités technique de RTEL4I, mettant notamment en oeuvre l'architecture en couches, se trouve décrit dans cette partie.

Ces fonctionnalités vont de la création d'une nouvelle carte, à la création d'un kernel en passant par la création d'un nouveau cross_compilateur.

Le diagramme ci-dessous fait un fort résumé de ce qui va être décrit par ailleurs





4.1. Ajout d'une nouvelle carte

identifiant	add_new_card
visibilité	externe
acteur	utilisateur
dépendances	<ul style="list-style-type: none">• set_general_info• set_hw_info• set_cross_compiler• set_kernel• build_card• install_card
déclencheur	Action de click sur bouton (ou appel explicite en ligne de commande)
paramètre	aucun
retour	0 si par d'erreur

Description : Au démarrage de l'interface, un certain nombre de carte, validées par rtl4i, sont proposées à l'utilisateur. Chaque carte a ses propres spécificités, divisées en trois catégories :

- spécificités générales : description, date de création, nom de la carte, projets auxquels est rattachée la carte ...
- spécificités matérielles/bas-niveau : architecture de la carte, sous architecture, type d'abi, type d'endian, type de cpu ...
- spécificités logicielles : un ensemble de kernel produit par des cross-compileur compatible avec la carte

Un utilisateur doit avoir la possibilité d'ajouter une nouvelle carte. La création d'une nouvelle carte se caractérise donc par la séquence d'actions suivantes

1. cliquer sur le bouton « add card »
2. affichage du popup wizard de création
 1. premier volet : spécificités générales
 1. appel de la fonction set_general_info
 2. second volet : spécificités matérielles
 1. appel de la fonction set_hw_info
 3. troisième volet : spécificités logicielles



1. appel de la fonction set_cross_compiler
2. appel de la fonction set_kernel
3. demande de validation de l'utilisateur
4. création de la carte
 1. appel de la fonction build_card
 2. appel de la fonction install_card

Chaque action est chaînée à la suivante, à la condition qu'elle se termine sans erreur. Par exemple, si la fonction build_card (voir par ailleurs) renvoie un code erreur, la fonction install_card ne sera évidemment pas appelée.

4.2. Ajout d'un nouveau cross_compilateur

identifiant	add_new_cc
visibilité	externe
acteur	utilisateur
dépendances	build_new_cc install_new_cc
déclencheurs	- Action de click sur bouton « + » sous la liste des cross-compilateurs (une carte doit être sélectionné) ou - Lors de la création d'une carte, au passage de la fonction « set_cross_compiler », l'utilisateur choisit l'option « create custom »
paramètre	aucun
retour	0 si par d'erreur

Description : Il est possible pour une carte donnée, et donc pour un ensemble d'informations hardware, il est possible d'ajouter un nouveau cross-compilateur (qui n'appartiendra donc PAS aux paramètres usines).

Le déclenchement de cette fonction se fait de deux manières possibles :

- Par click explicite sur le bouton « + » situé sous la liste des cross-compilateurs compatibles avec la carte couramment sélectionnée.
- Lors de l'étape de sélection de cross-compilateurs lors de la création d'une carte, en choisissant de créer une configuration custom plutôt que d'en sélectionner un existant

L'utilisateur voit donc un panneau de configuration, avec :



- Les options de bases : version de gcc, version de libc, type de libc, type de thread, version de kheaders, version de binutils
- Les options avancées : les flags

Les options de bases apparaîtront sous forme de liste de choix.

Les options avancées apparaîtront sous forme de zone de texte, avec une info-bulle d'aide.

Les options validées sont enregistrées et validées par le plugin bas-niveau.

4.2.1 Construction du nouveau cross_compiler

identifiant	add_new_cc
visibilité	interne
acteur	utilisateur
dépendances	install_new_cc
déclencheurs	
paramètre	aucun
retour	0 si par d'erreur

Description : Une fois configuré, le cross_compiler est prêt à être produit. Il est du ressort du plugin d'effectuer ce travail

4.2.2 Installation du nouveau cross_compiler

identifiant	install_new_cc
visibilité	interne
acteur	utilisateur
dépendances	
déclencheurs	
paramètre	aucun
retour	0 si par d'erreur

Description : Si l'étape précédente s'est bien déroulée, le cross_compiler produit est copié dans le répertoire utilisateur de sauvegarde, et une entrée est ajoutée dans un fichier XML utilisateur, dont la structure est similaire à celle du fichier usine. Deux fichiers différents sont utilisés pour distinguer les cross_compilers de base, de ceux produits par l'utilisateur.



4.3. Ajout d'un nouveau kernel

identifiant	add_new_kernel
visibilité	externe
acteur	utilisateur
dépendances	select_kernel_type set_misc_options set_boot_method set_debug_options set_devices_options build_new_kernel install_new_kernel
déclencheurs	- Action de click sur bouton « + » sous la liste des kernel (une crosscompiler doit être sélectionné) ou - Lors de la création d'une carte, au passage de la fonction « set_kernel », l'utilisateur choisit l'option « create custom »
paramètre	aucun
retour	0 si par d'erreur

Description : Un kernel va être produit pour une carte cible, et par un cross_compilateur identifié. Lorsque l'utilisateur sélectionne un cross_compilateur pour une carte donnée, une liste de noyaux produits par ce dernier, pour la carte ciblée, apparaît.

Il est possible d'ajouter un nouveau kernel (qui n'appartiendra donc PAS aux paramètres usines).

Le déclenchement de cette fonction se fait de deux manières possibles :

- Par click explicite sur le bouton « + » situé sous la liste des noyaux compatibles avec le cross_compilateur couramment sélectionné..
- Lors de l'étape de sélection de kernel lors de la création d'une carte, en choisissant de créer une configuration custom plutôt que d'en sélectionner un existant

La configuration du noyau est l'élément le plus complexe, et se fait de deux manières possibles :

- via un wizard
- via la configuration classique

La configuration classique étant relativement standard, l'analyse fonctionnelle va décrire les étapes par wizard.



- Sélection du type de kernel
- Sélection des options de base (obligatoire)
- Sélection d'options diverses (scheduling, nom du noyau ...)
- Sélection de la stratégie de démarrage
- Sélection des options de debuggage
- Sélection du matériel

4.3.1 Configuration du type de kernel

identifiant	select_kernel_type
visibilité	externe
acteur	utilisateur
dépendances	set_basic_config register_rt_handler
déclencheurs	Phase de configuration d'ajout d'un nouveau noyau
paramètre	Architecture cible
retour	0 si par d'erreur

Description : A la configuration du noyau, la première étape va consister à qualifier le type de noyau à créer (standard, soft-rt, hard-rt)

Pour cela, le programme connaît l'architecture ciblée (arm, x86 ...). Un panneau de trois choix apparaît :

- création d'un noyau temps-réel dur (xenomai)
- création d'un noyau temps-réel mou (preempt-rt)
- création d'un noyau standard

Les deux premiers choix n'apparaissent que si la phase de lecture effectuée par la fonction `read_rtext_compatibility_preempt` a retourné au moins un triplet xenomai et un triplet preempt-rt pour l'architecture ciblée.

La sélection d'un noyau xenomai affiche la liste des version noyau 2.6 compatibles avec xenomai pour cette architecture. L'utilisateur sélectionne une de ses versions.

Le mécanisme est similaire pour PREEMPT-RT.

En ce qui concerne un noyau standard, l'utilisateur sélectionne la version qu'il souhaite.

A cette étape, les éléments de configuration sélectionnés sont :

- la version des sources du noyau (récupéré implicitement de la version du patch xenomai/preempt-rt ou explicitement dans le cas d'un noyau standard)
- la version du patch RT, si nécessaire

Cette étape n'effectue pas l'application des patchs, mais enregistre une callback de post-configuration, qui permettra d'exécuter le script xenomai ou le script preempt-rt. (via la dépendance `register_rt_handler`)

4.3.2 Configuration basique du kernel

identifiant	set_basic_config
visibilité	externe
acteur	utilisateur
dépendances	apply_kernel_type_selection_xeno ou apply_kernel_type_selection_premprt ou apply_kernel_type_selection_base
déclencheurs	Phase de configuration d'ajout d'un nouveau noyau
paramètre	Architecture cible
retour	0 si par d'erreur

L'étape précédente (`select_kernel_type`) a permis à l'utilisateur de sélectionner son extension temps-réel, et par la même de manière implicite, la version du noyau.

Ce noyau va être téléchargé s'il ne figure pas dans le repository RTEL4I. Une fois téléchargé et décompressé, la fonction `read_kernel_defconfig` (voir précédemment) est appelée pour scanner l'ensemble des fichiers de configuration de base possible pour l'architecture cible. De plus ces données sont mises en corrélation avec celles figurant dans le fichier XML et extraites via la fonction `read_kernel_defconfig_xml`, afin d'obtenir une description suffisamment explicite pour que l'utilisateur puisse sélectionner son choix.

L'utilisateur sélectionne donc parmi la liste, la configuration basique qu'il souhaite utiliser.

Une fois ceci fait, la sélection est récupérée, et la commande shell : `make ARCH=cible_arch CROSS_COMPILE=cc_current config_base_selected` est exécutée.

Ensuite, si une callback pour appliquer une extension temps-réel a été enregistrée dans la fonction `select_kernel_type`, elle est exécutée.

Cette fonction peut être :

- `apply_kernel_type_selection_xeno`
- ou bien
- `apply_kernel_type_selection_premprt`

4.3.2.1. Lecture du fichier de configuration de base

identifiant	read_basic_config
-------------	-------------------



visibilité	interne
acteur	
dépendances	
déclencheurs	Fin de configuration de base
paramètre	
retour	0 si par d'erreur

Description : Le fichier .config créé par l'étape précédente va être lue ligne à ligne, afin de vérifier l'existence de certaines options à configurer

4.3.2.2. Application du type de kernel xenomai


identifiant	apply_kernel_type_selection_xeno
visibilité	interne
acteur	
dépendances	
déclencheurs	Validation de la fonction select_kernel_type
paramètre	Version des sources du noyau Architecture cible extension temps-réel
retour	0 si par d'erreur

Description: L'exécution de cette fonction se fait en étapes :

- téléchargement des sources du noyau
- application d'une configuration de base, en fonction de l'architecture (make xxxoldconfig)
- application du patch xenomai : *prepare_kernel.sh -kernel path_to_src -adeos patch_selected_previously*

4.3.2.3. Application du type de kernel preempt

identifiant	apply_kernel_type_selection_preempt
visibilité	interne
acteur	
dépendances	

	Spécifications techniques des outils, partie 2	Version: 1.0 Auteur: Frédéric Ferrandis
---	--	--

déclencheurs	Validation de la fonction <code>select_kernel_type</code>
paramètre	Version des sources du noyau Architecture cible extension temps-réel
retour	0 si par d'erreur

Description: L'exécution de cette fonction se fait en étapes :

- téléchargement des sources du noyau
- application d'une configuration de base, en fonction de l'architecture (`make xxxoldconfig`)
- application du patch `preempt-rt` : `patch -p1 -i patch_selected_previously`

4.3.3 Configuration options « misc » du noyau

identifiant	<code>set_misc_options</code>
visibilité	externe
acteur	Utilisateur
dépendances	<code>apply_misc_options</code>
déclencheurs	Validation de la fonction « <code>set_rtt_extension</code> »
paramètre	Version des sources du noyau Architecture cible extension temps-réel
retour	0 si par d'erreur

Description : L'utilisateur verra apparaître un panneau sur lequel il pourra configurer diverses options

- le nom du noyau
- l'allocateur de slab
- gestion ou non des modules
- le model de preemption du noyau
- l'utilisation du smp
- les filesystems à utiliser (`ext3`, `jffs2`, `xfs` ...)

L'application devra être capable de fournir des options et des valeurs compatibles avec la version du noyau, ainsi qu'à l'architecture. En effet, certaines options, par exemple le support du SMP ne sont pas forcément compatibles avec toutes les versions et architecture du kernel. Pour faire ce filtre, la fonction `read_basic_config` (voir précédemment) a été appelée, afin de voir, parmi la liste des options décrites ci-dessus, celles qui existent vraiment.



4.3.3.1. Application options « misc » du noyau

identifiant	apply_misc_options
visibilité	interne
acteur	
dépendances	
déclencheurs	Validation de la fonction « set_misc_options »
paramètre	Version des sources du noyau Architecture cible extension temps-réel
retour	0 si par d'erreur


Description : Après validation de la fonction set_misc_options, les paramètres sélectionnés sont appliqués à la configuration du noyau.

4.3.4 Configuration options de boot

identifiant	set_boot_method
visibilité	externe
acteur	Utilisateur
dépendances	apply_boot_method
déclencheurs	Validation de la fonction « set_misc_options »
paramètre	Version des sources du noyau Architecture cible extension temps-réel
retour	0 si par d'erreur

Description : Cette option va permettre à l'utilisateur de décider de sa stratégie de boot. Ainsi il aura le choix entre :

- boot standard
- boot via nfs, ce qui lui fera choisir
 - l'adresse IP du serveur NFS
 - l'adresse IP de la gateway
 - le nom de l'interface réseau sur la carte, par défaut « eth0 »

	Spécifications techniques des outils, partie 2	Version: 1.0 Auteur: Frédéric Ferrandis
---	--	--

- le nom du répertoire exporté sur le serveur NFS

4.3.4.1.Applications des paramètres de boot sur le noyau

identifiant	apply_boot_method
visibilité	interne
acteur	
dépendances	
déclencheurs	Validation de la fonction « set_boot_method»
paramètre	Version des sources du noyau Architecture cible extension temps-réel
retour	0 si par d'erreur

Description : La validation de la fonction set_boot_method entraîne l'application de ceux-ci sur les sources du noyau, à savoir :

- dans le cas d'un boot via NFS :
 - sélection de l'option de support client NFS dans le noyau
 - sélection de l'option de support de rootfs over NFS du noyau
 - ajout des paramètres sur la ligne de commande du noyau (par exemple :
`root=/dev/nfs rw nfsroot=IP_SRV_NFS:NAME_DIR_EXPORT
ip=IP:::255.255.255.0::ITF_NAME:off)`

4.3.5 Configuration options de debug et instrumentations

identifiant	set_debug_options
visibilité	externe
acteur	utilisateur
dépendances	apply_debug_options
déclencheurs	Validation apply_boot_method
paramètre	Version des sources du noyau Architecture cible
retour	0 si par d'erreur

Description : Ce volet de configuration a pour objectif de guider l'utilisateur dans les



multitudes d'options possibles du noyau.

Les options apparaîtront sous forme de case à cocher, avec un descriptif fonctionnel, et un moyen d'utilisation de ces options.

Les options suivantes sont attendues :

- activation du debug
- activation du profiling
- debug du noyau
- monitoring
 - collection des statistiques de scheduling (pour le temps-réel)
 - collection des statistiques de timer
- traces
 - trace d'appel de fonction noyau
 - trace de latence de scheduler
 - trace de latence de changement de contexte de tache

De la même manière que pour les « misc » options, la version du noyau impactera sur les propositions. Une liste de compatibilité pourra être tenue de la même manière que proposé précédemment.

4.3.5.1.Applications des paramètres de debug du noyau

identifiant	apply_debug_options
visibilité	interne
acteur	
dépendances	
déclencheurs	Validation set_debug_options
paramètre	Version des sources du noyau Architecture cible
retour	0 si par d'erreur

Description : L'application des paramètres se feront, si activés, sur la configuration du noyau, notamment:

- sélection de l'option profiling_support et oprofile_support
- sélection de kernel_hacking et de frame_pointer
- sélection du système de fichier debugfs
- ...



4.3.6 Configuration matérielle

identifiant	set_devices_options
visibilité	externe
acteur	utilisateur
dépendances	apply_devices_options
déclencheurs	Validation de la fonction set_debug_options
paramètre	Architecture, version du noyau
retour	0 si par d'erreur

Description : L'utilisation d'une configuration de base aura permis de dégrossir le travail de l'utilisateur en ce qui concerne la configuration des devices drivers. Néanmoins, pour un modèle de carte particulier, il existera toujours certaines spécificités (par exemple pour le sous-système MTD, ou bien PCI). Pour le moment, le volet de configuration proposé pour cela ressemblera à ce qui est proposé en standard. Il sera possible d'organiser différemment et de manière plus intelligente cette configuration, mais cela reste trop spécifique pour pouvoir définir une seule manière de faire..


A la fin de cette étape devra être affiché la configuration générée. La validation de celle-ci amène à l'étape de construction du nouveau kernel (build_new_kernel)

4.3.7 Construction du nouveau kernel

identifiant	build_new_kernel
visibilité	interne
acteur	utilisateur
dépendances	
déclencheurs	Validation de la fonction set_devices_options
paramètre	Architecture, version du noyau
retour	0 si par d'erreur

Description : Cette fonction lancera simplement la commande de compilation du noyau. La sortie de la commande devra être visible depuis l'interface graphique pour que l'utilisateur puisse suivre l'avancée des opérations.

A la fin de cette étape, le noyau est prêt à être installé.

	Spécifications techniques des outils, partie 2	Version: 1.0 Auteur: Frédéric Ferrandis
---	--	--

4.3.8 Installation du nouveau kernel

identifiant	install_new_kernel
visibilité	interne
acteur	utilisateur
dépendances	
déclencheurs	Validation de la fonction set_devices_options
paramètre	Architecture, version du noyau
retour	0 si par d'erreur

Description : le fichier noyau ainsi que les modules sont copiés dans le répertoire spécifié par l'utilisateur.

En plus de cela, cet ensemble est sauvegardé dans un fichier XML structuré de la même manière que le fichier usine.

5.Fonctions « Instrumentation »

L'objectif de cette section est de décrire, d'une part les fonctionnalités d'instrumentations, leur utilité et leur mise en oeuvre, et d'autre part, de justifier les choix retenus, parmi le très grand nombre de possibilités offertes par GNU/Linux.

Afin de pouvoir tirer parti de toutes les possibilités, le noyau, ainsi que le root-file system doivent avoir des options et applicatifs supplémentaires, par rapport à une configuration initiale.

Afin de simplifier ce paramétrage, l'utilisateur aura la possibilité de sélectionner des options macroscopique (voir ci-dessous) permettant d'ajouter ce qui est nécessaire aussi bien pour le kernel que pour les applicatifs.

5.1.Génération version de test

identifiant	generate_test_version
visibilité	externe



acteur	utilisateur
dépendances	
déclencheurs	Click sur bouton
paramètre	Paramètres du projet
retour	0 si par d'erreur

Description : L'utilisateur, lorsqu'il le souhaite, peut à partir de sa version de travail, générer un kernel et un root file system destiné à être instrumenté.

L'objectif des instruments proposés sera de permettre de :

- debug de module noyau
- détection de fuite mémoire
- suivi de consommation mémoire par processus
- profiling du système