



open wide
INGENIERIE

Spécifications fonctionnelles
pour l'EDI Eclipse

Version: 1.0

Auteur: Thomas Monjalon

- PROJET RTEL4I -

RTEL4I
REAL TIME EMBEDDED LINUX FOR INDUSTRIES

Spécifications fonctionnelles pour l'EDI Eclipse (L1.2-a)



SAGEMCOM





open wide
INGENIERIE

Spécifications fonctionnelles
pour l'EDI Eclipse

Version: 1.0

Auteur: Thomas Monjalon

MODIFICATIONS

<i>VERSION</i>	<i>DATE</i>	<i>AUTEUR(S)</i>	<i>DESCRIPTION</i>
1.0	25/10/2009	T. Monjalon	Création



Table des matières

1.Présentation du document et du contexte.....	5
2.Généralités.....	6
2.1.Spécifications de l'interface.....	6
2.1.1Description du mock-up.....	6
2.2.Fonctionnement interne de l'interface.....	8
2.2.1Généralités.....	8
2.2.2Sauvegarde.....	9
2.2.3Changement de version/profil.....	9
2.2.4Composants.....	10
2.2.5Version détaillée.....	11
2.2.6Configuration d'un composant.....	11
2.2.7Propriétés d'un composant.....	12
2.2.8Ajout de composant.....	12
2.2.9Outils.....	13
2.2.10Création de release.....	13

Index des illustrations

Figure 1 . Mokup de l'interface Eclipse.....7



1. Présentation du document et du contexte

Ce document a pour but de présenter les spécifications fonctionnelles des extensions à l'EDI Eclipse utilisées dans RTEL4I. Comme nous l'avons évoqué précédemment, Eclipse est une des utilisations du middleware RTEL4I qui constitue un point fonctionnel clé du projet.

L'autre utilisation est l'interface web développée par Mandriva et décrite dans le livrable L5.1-a.

2. Généralités

L'extension développée s'intègre parfaitement à l'interface Eclipse du fait qu'elle est développée sous forme de greffon (plugin). La documentation concernant l'API des plugins Eclipse est disponible en ligne sur le site <http://eclipse.org>. Voici quelques liens utiles.

- <http://help.eclipse.org/helios/index.jsp?topic=/org.eclipse.platform.doc.isv/reference/api/org.eclipse.core.runtime/Plugin.html>
- <http://www.eclipse.org/articles/Article-Your%20First%20Plug-in/YourFirstPlugin.html>
- <http://beuss.developpez.com/tutoriels/eclipse/plugin/editor/bases/>
- <http://www.vogella.de/articles/EclipsePlugIn/article.html>

2.1. Spécifications de l'interface

Eclipse est avant tout une interface graphique. Le point principal concerne l'ergonomie du système. Les récents développements autour des IHM dans le domaine du web ou de la téléphonie (iPhone, Android, ...) ont vu l'apparition d'outils permettant de définir plus simplement des spécifications d'interface sous le nom de « mock-up ». Le terme indique une maquette à l'échelle 1:1 désignant un prototype d'interface utilisateur. Le but est de présenter les fonctionnalités d'une IHM. Certaines maquettes évoluées (exemple: sur iPhone) peuvent être dynamiques.

Le mock-up est souvent présenté sous forme d'un croquis similaire à ce qui pourrait être dessiné « à la main ».

Dans le cas présent, nous avons utilisé l'outil *Balsamiq* (<http://balsamiq.com>) très réputé dans le domaine. Ce n'est pas un logiciel libre, mais une version en ligne est disponible gratuitement.

2.1.1 Description du mock-up

L'interface Eclipse prévue est décrite sur la figure ci-dessous. Elle est pour l'instant en anglais mais on peut imaginer une traduction ultérieurement.

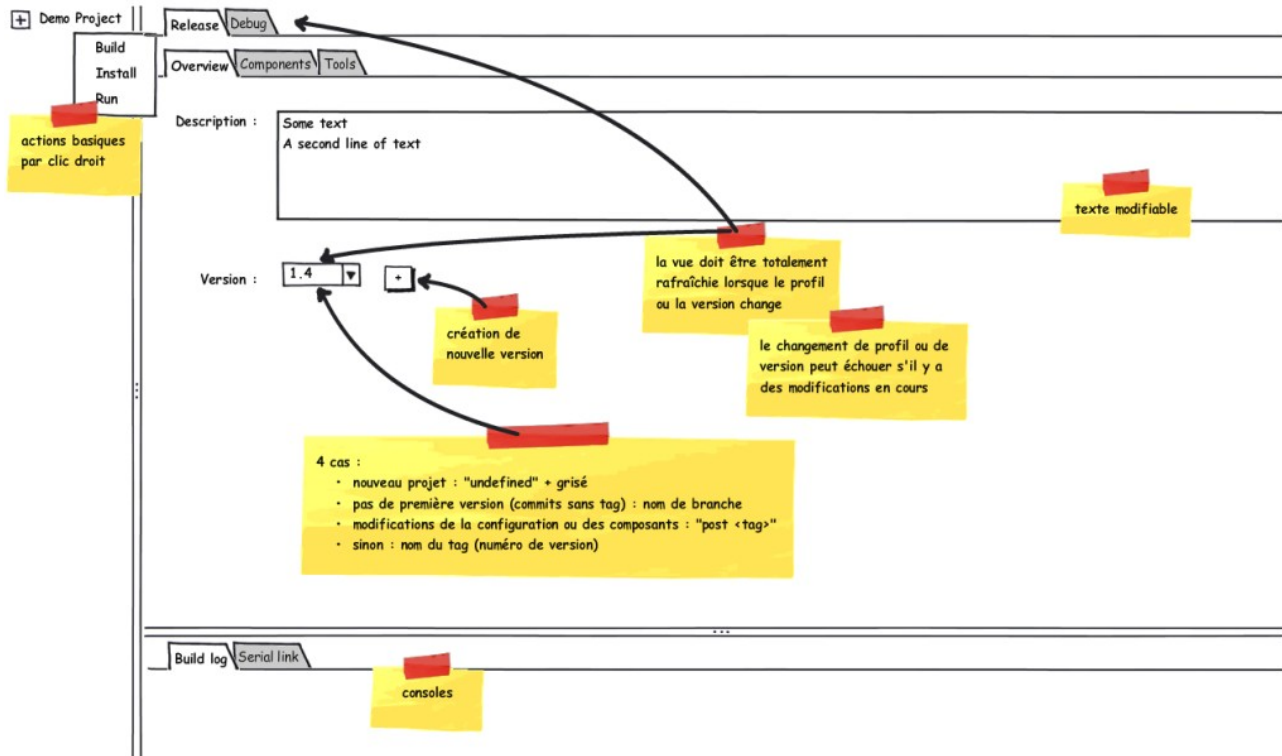


Figure 1 . Mock-up de l'interface Eclipse

La figure est « auto-documentée » sur les parties jaunes (ce qui est l'intérêt d'un mock-up) et indique les principales zones et actions disponibles sur l'interface utilisateur. De même, les fonctionnalités sont liées au « guideline » d'Eclipse (<http://www.eclipse.org/articles/Article-UI-Guidelines/Contents.html>) puisque nous avons affaire à une extension.

Nous notons les différents onglets :

- *Release/Debug*: type de firmware produit en version définitive (*Release*) ou développement (*Debug*)
- Les principales commandes concernant la production du firmware: *Build, Install, Run*
- Les onglets pour les zones de travail:
 - *Overview*: vue générale du projet, affichant des informations comme la version du firmware
 - *Components*: composants logiciels du projet



- *Tools*: outils disponibles (exemples: mise au point, simulation, ...). Certains outils comme la mise au point ou la compilation s'appuient bien entendu sur des plugins existants comme CDT (<http://www.eclipse.org/cdt>) qui fournit les fonctionnalités de base pour le développement C/C++.

2.2.Fonctionnement interne de l'interface

A l'ouverture de projet, il faut préciser au middleware s'il s'agit d'un projet *forge* ou *local*. L'interface web traite des projets de type *forge* (uniquement ce qui est centralisé/sous contrôle de version, i.e. « versionné »).

De même, le verbe *commiter* est un anglicisme très fréquemment utilisé par les développeurs. Il correspond à l'enregistrement de modifications sous forme d'une nouvelle version (`git commit`).

2.2.1Généralités

Dans le cas d'un projet de type *forge*,

- On donne l'URL du dépôt de configuration
- Le middleware crée une copie de travail du dépôt de configuration dans un répertoire temporaire unique à la session

L'interface de développement Eclipse traite les projets en local qui peuvent être différents de ce qui est versionné. On donne donc le répertoire projet à ouvrir.

L'affichage du numéro de version projet s'effectue comme suit :

- aucun commit => *undefined* + grisé
- aucun tag + modifications non commitées => *modified branch-version*
- aucun tag => *branch-version*
- tag + modifications non commitées => *modified post tag*
- tag + modifications => *post tag*
- sinon *tag*

Si on commite des configurations non publiées ou nouveaux commits sur le dépôt on affiche le message configuration *not synchronized with repository*.

Si on effectue des modifications de composants par rapport à la configuration (différent de la configuration ou des modifications locales) on affiche le message *configuration not synchronized with local modifications*.

L'ordre des composants est le suivant:

- non versionné (*not configured*)
- différent de la configuration
 - work in progress (*on branch or tag*)

	Spécifications fonctionnelles pour l'EDI Eclipse	Version: 1.0 Auteur: Thomas Monjalon
---	---	---

- local modifications (*from branch or tag*)
 - autre branche/tag
- branche non téléchargée
- branche
- tag non téléchargé
- tag

Si l'état d'un composant est différent de la configuration, on affiche alors deux lignes (la réalité et ce qui est configuré) ainsi que le statut *work in progress* et *local modifications* sur la première ligne. On ne peut changer de version/profil que s'il n'y a pas de *work in progress*.

2.2.2 Sauvegarde

Le projet ne peut pas être sauvegardé si la fonction `config_is_writeable()` renvoie `False`. La sauvegarde aura pour effet de modifier la valeur retournée par `config_has_changes()`.

La sauvegarde étant faite à l'aide de Git, un commentaire est obligatoire. Une boîte de dialogue surgit pour proposer de rentrer un commentaire. La zone de texte du commentaire peut être désactivée si l'on coche la case *merge with previous save*. En revanche, cette case à cocher doit être désactivée si `config_is_published()` renvoie `True`.

Lorsque la case *merge with previous save* est cochée, la zone de texte *comment* est remplie avec `get_last_config_comment()` et est désactivée (non modifiable).

La fonction à appeler est `config_commit(comment, amend)`. Le paramètre `comment` est une chaîne de caractères, `amend` est `True` (si la case est cochée) ou `False`.

Le paramètre `comment` ne peut pas être vide. Si l'utilisateur n'a rien saisi, il faut mettre *saved in Eclipse*.

Les sauvegardes peuvent-être publiées sur le serveur, cela aura pour effet de modifier la valeur renvoyée par `config_is_published()` en appelant `config_publish()`. Cette action doit être activée si la case *publish* est cochée.

2.2.3 Changement de version/profil

Pour changer de profil ou de version, il faut appeler deux fonctions. Il faut noter qu'une version peut-être majeure (*version_split*) ou mineure (*release*). Les deux types sont pris en charge sous le terme *version*.

La première fonction permet d'obtenir la référence où l'on souhaite basculer.

```
ref = Project.get_config_ref (version=new_version)
ref = Project.get_config_ref (profile=new_profile)
```

La référence doit ensuite être utilisée pour faire le changement.



Project.jump (ref)

Il n'est pas possible de faire un `jump()` si `config_has_changes()` est vrai.

2.2.4 Composants

On peut dénombrer quatre cas différents :

1. Par défaut, on affiche le composant sur 1 ligne en montrant ce qui est configuré (`get_configured_version`, `get_configured_ref`).
2. Si le composant n'a pas de dépôt configuré (`not has_repository`) mais a un répertoire local (`exists(path)`), alors c'est la version courante qui doit être affichée (`get_current_ref`).
3. S'il n'y a pas de dépôt configuré (`not has_repository`), ni de répertoire local (`not exists (path)`), alors il faut afficher *no repository* sur le reste de la ligne à la place des informations détaillées. La ligne doit être rouge.
4. Il peut y avoir deux lignes si l'état courant du composant est différent de ce qui est configuré.

Comment savoir si c'est différent ?

- `get_configured_ref['commit']` et `get_last_commit['commit']` sont différents
- ou `has_changes`

Dans le cas de différences, la version configurée est affichée en 2° et est barrée (ou grisée). Le nom du composant doit apparaître sur la première ligne.

Les colonnes correspondent à ces fonctions :

- `get_name`
- `get_current_version` et `get_configured_version`
- `get_current_ref ['type']` et `get_configured_ref ['type']`
- `get_current_ref ['name']` et `get_configured_ref ['name']`
- `get_current_ref ['commit'].get_age` et `get_configured_ref ['commit'].get_age`

Dans la colonne "version", `get_current_version` doit être remplacé par des messages en rouge si l'on est dans un de ces deux cas :

- "*work in progress*" (`has_changes`)
- "*local modifications*" (`exists(path)` and `not is_published`)

	Spécifications fonctionnelles pour l'EDI Eclipse	Version: 1.0 Auteur: Thomas Monjalon
---	---	---

Les lignes correspondant à un des ces deux derniers cas où correspondant à une branche doivent être mise en avant en colorant le fond en vert. Deux colonnes supplémentaires montrent des boutons *configure* et *properties*.

Le bouton *configure* n'apparaît que si `is_configurable()` est vrai. Dans ce cas, il doit être sur la première ligne.

Le bouton *properties* apparaît tout le temps sur la dernière ligne du composant, c'est-à-dire sur la deuxième s'il y en a une, sinon sur la première.

Si le composant a un dépôt configuré (`has_repository`) mais pas de dépôt local (`get_path` and `not exists(path)`), il faut afficher un bouton *Get* qui appelle la fonction `clone` pour faire le téléchargement. Ce bouton remplace le bouton *Configure*.

Les composants nécessitant l'attention de l'utilisateur sont affichés en premier. Pour un même niveau d'importance, c'est l'ordre alphabétique qui s'applique. On définit donc différents niveaux d'importance dans l'ordre :

- no repository
- work in progress
- local modifications
- différent de la configuration
- branche non téléchargée
- branche
- tag non téléchargé
- tag

2.2.5 Version détaillée

Ici, l'objet donnant les informations est `Project`. La version est renvoyée par `get_version`. Il faut ajouter des informations :

- profil (`get_profile`)
- config modified (`config_has_changes`)
- config not published (`not is_published`)

2.2.6 Configuration d'un composant

Les composants peuvent être configurables. Si ils le sont, la méthode `is_configurable()` renvoie `True`. Dans ce cas, un bouton *configure* doit être visible sur la ligne du composant.

L'appui sur le bouton *configure* déclenche l'ouverture d'une fenêtre de configuration externe.

	Spécifications fonctionnelles pour l'EDI Eclipse	Version: 1.0 Auteur: Thomas Monjalon
---	---	---

Pour cela, il faut appeler la méthode `configure()` de l'objet `component`.

2.2.7 Propriétés d'un composant

On clique sur le bouton de meta-configuration du composant. L'interface propose de rentrer une URL existante ou de créer un nouveau dépôt pour ce composant. On rentre donc une URL pour créer un nouveau dépôt. Concrètement, une petite fenêtre qui doit apparaître quand on clique sur le bouton *properties* d'un composant. Elle doit présenter un champ texte pour l'URI du dépôt, un deuxième champ pour préciser de manière optionnelle la branche ou le tag, un troisième pour décrire de manière optionnelle la version, un bouton pour valider et un deuxième bouton pour une autre option de création de dépôt. Ce dernier bouton, séparé du reste, ne doit apparaître que si le composant n'a pas de dépôt.

Côté API, les fonctions sont `set_repository()`, `set_version()` et `create_repository()`.

Il faut afficher la valeur des paramètres dans les zones de textes. Pour ça, il faut utiliser des fonctions `get_*` :

- `get_repository().get_uri()` renvoie une URI, suivie de manière optionnelle d'une branche/tag, c'est un espace qui sépare les 2 champs
- `get_configured_version(False)` renvoie uniquement la version telle que renseignée par `set_version()`

La condition pour afficher le bouton *Create repository* est :

```
not has_repository() and not exists(get_path() + '/.git')
```

2.2.8 Ajout de composant

Nous avons l'interface avec un bouton *Ajouter (+)*. Celui-ci, au clic, ouvre une nouvelle fenêtre.

La fenêtre affiche un premier bouton *Configure*, qui au clic doit appeler la fonction `project.configure()`, ce qui aura pour effet de lancer l'interface de configuration du plugin Buildroot.

Nous avons ensuite trois champs :

1. Nom : [Champ Texte pour le nom du Paquet]
2. Dossier à ajouter : [Boite de dialogue de type Parcourir]
3. Destination : [Champ texte permettant de saisir le dossier de destination sur la cible]

Un bouton [*Créer*] pour valider les informations saisies au dessus.

Lors du clic sur le bouton *Créer*, il faudra vérifier que le composant n'est pas déjà présent via la fonction `project.component_exists(name)` qui renverra dans ce cas `True` (Il faudra donc prévenir l'utilisateur de saisir un autre). Si la fonction renvoie `False`, on peut donc appeler la fonction `project.create_component(nom, dossier à ajouter, Destination)`.



2.2.9 Outils

Dans l'onglet *Tools*, il y a trois sections :

- Debug
 - Serial console
 - GDB
- Builder
 - Buildroot (Configure)
- Measures
 - Xenoscope
 - Get CPU load
 - Get RAM usage

Pour la partie *Debug*, il faut une étiquette par élément (*Serial console/GDB*) suivie d'un bouton *Configure* et *Run* par élément, le tout sur une ligne par élément.

Pour *Serial console*, la partie configure ne sera pas implémentée dans un premier temps. Il faut donc seulement associer le bouton *Run* à la fonction `Project.view_serial_line()`.

Pour *Buildroot*, il faut afficher *Buildroot* dans une étiquette et juxtaposer un bouton *Configure* qui appelle la fonction `Project.configure()`.

Pour *Measures*, chaque ligne adopte ce format :

étiquette : valeur dans une zone de texte non modifiable, bouton *Acquire*

La valeur est vide puis est rafraîchie lorsque l'on clique sur *Acquire*.

2.2.10 Création de release

À côté de la sélection de version, dans l'onglet "overview", il y a un bouton "+" permettant de faire une nouvelle release.

Si des changements ne sont pas sauvegardés (`config_has_changes`), une *popup* doit prévenir que les changements non sauvegardés ne seront pas inclus dans la release.

Une boîte de dialogue doit s'afficher et proposer un champ texte pour spécifier le nom de la release. Sans cette information, la création n'est pas possible. Ensuite, la fonction `Project.set_release(name)` doit être appelée.

Les informations de versions du projet doivent être rafraîchies (`get_version`).