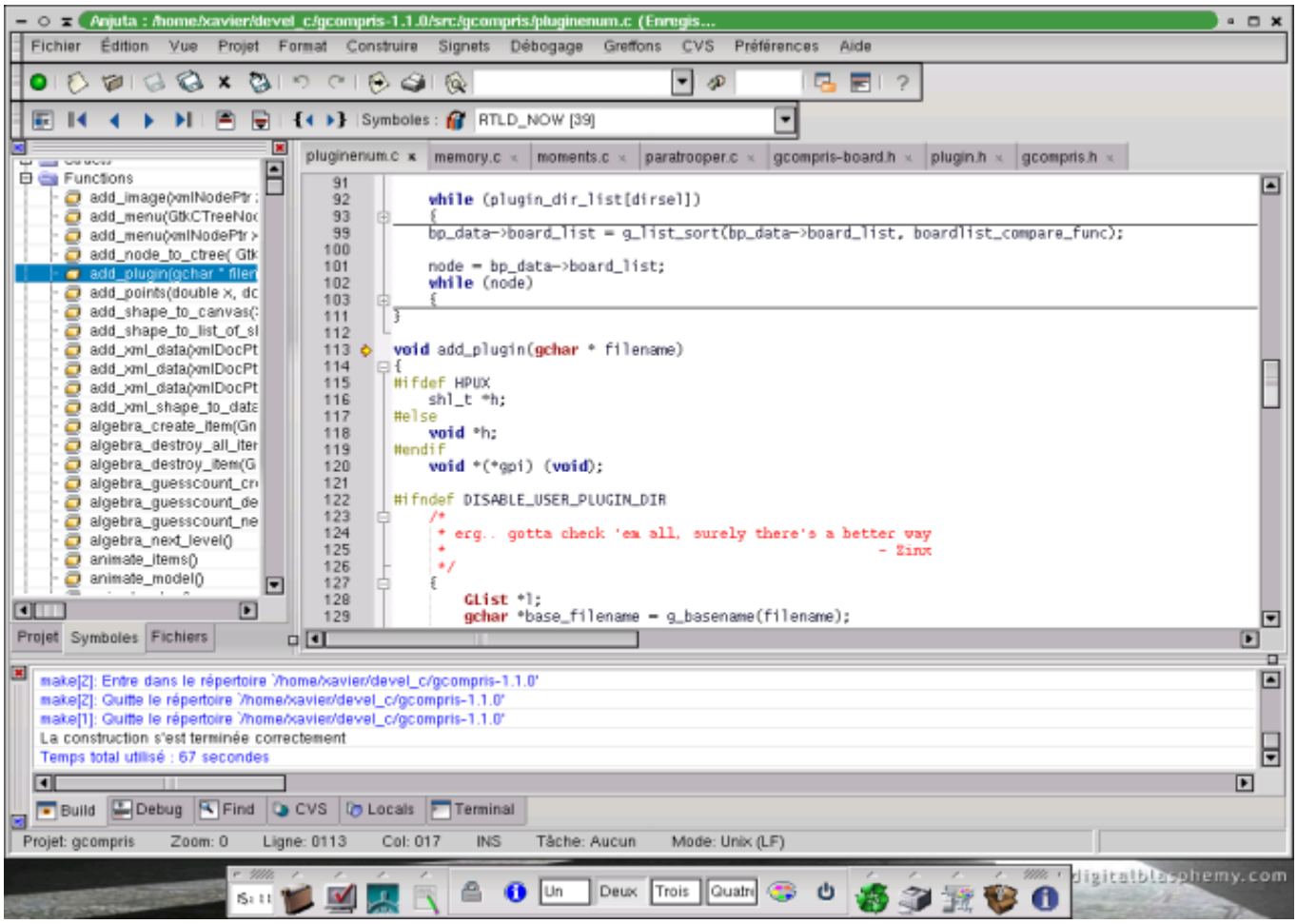
ANJUTA

xavier garreau

Cet article a pour objectif de vous faire découvrir un EDI (Environnement de Développement Intégré) dont on parle peu, à tort. Je m'adresserai à ceux qui aiment disposer de facilités pour construire leurs programmes tels que la colorisation de syntaxe, la recherche rapide des lignes correspondant aux erreurs et warnings de gcc par simple clic, possibilité de trouver une fonction dans un fichier uniquement en cliquant sur son nom dans un navigateur de code.
Anjuta fait tout cela pour les développeurs C/C++ et bien plus encore comme nous allons le voir.

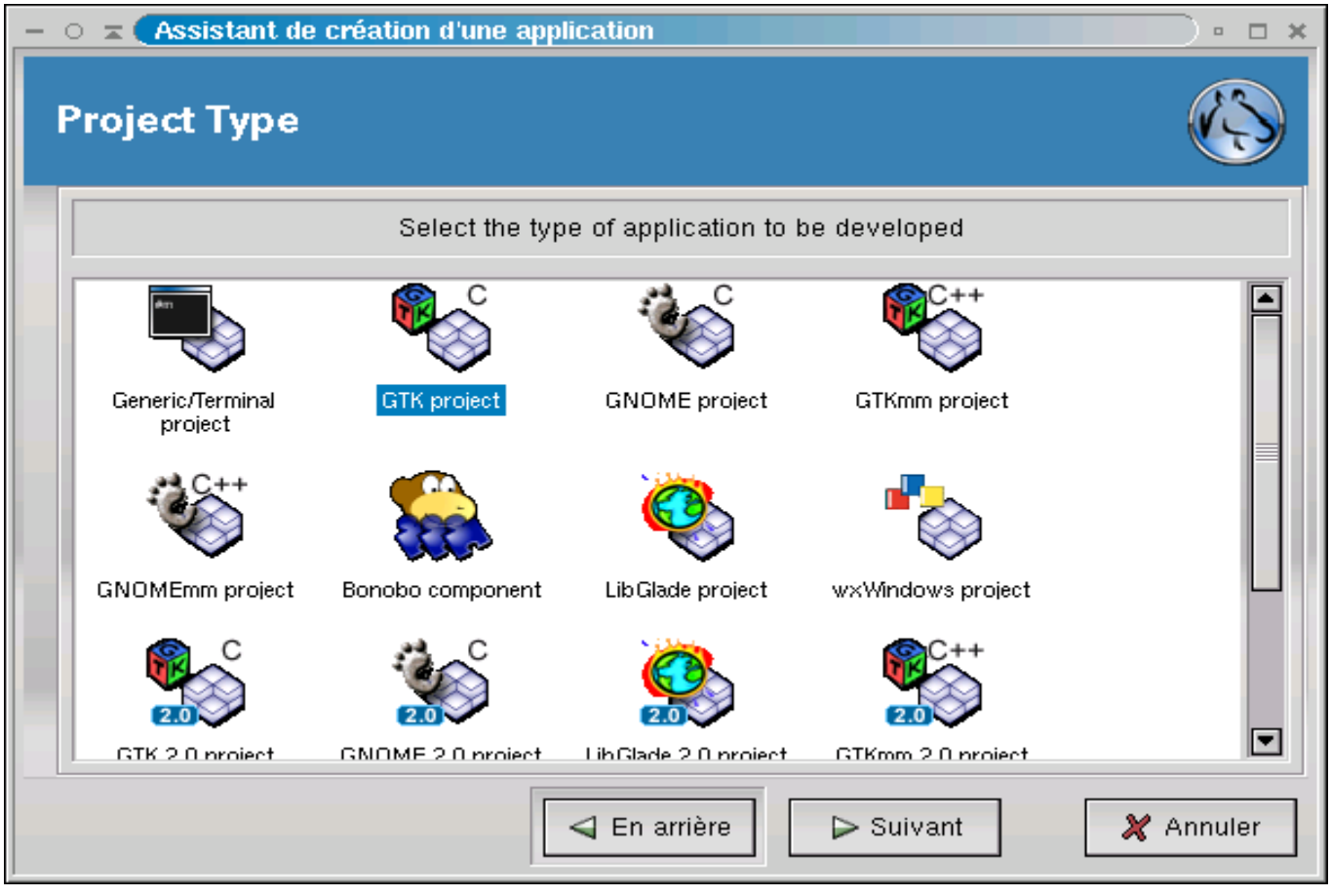
Les présentations



Anjuta vous offre :

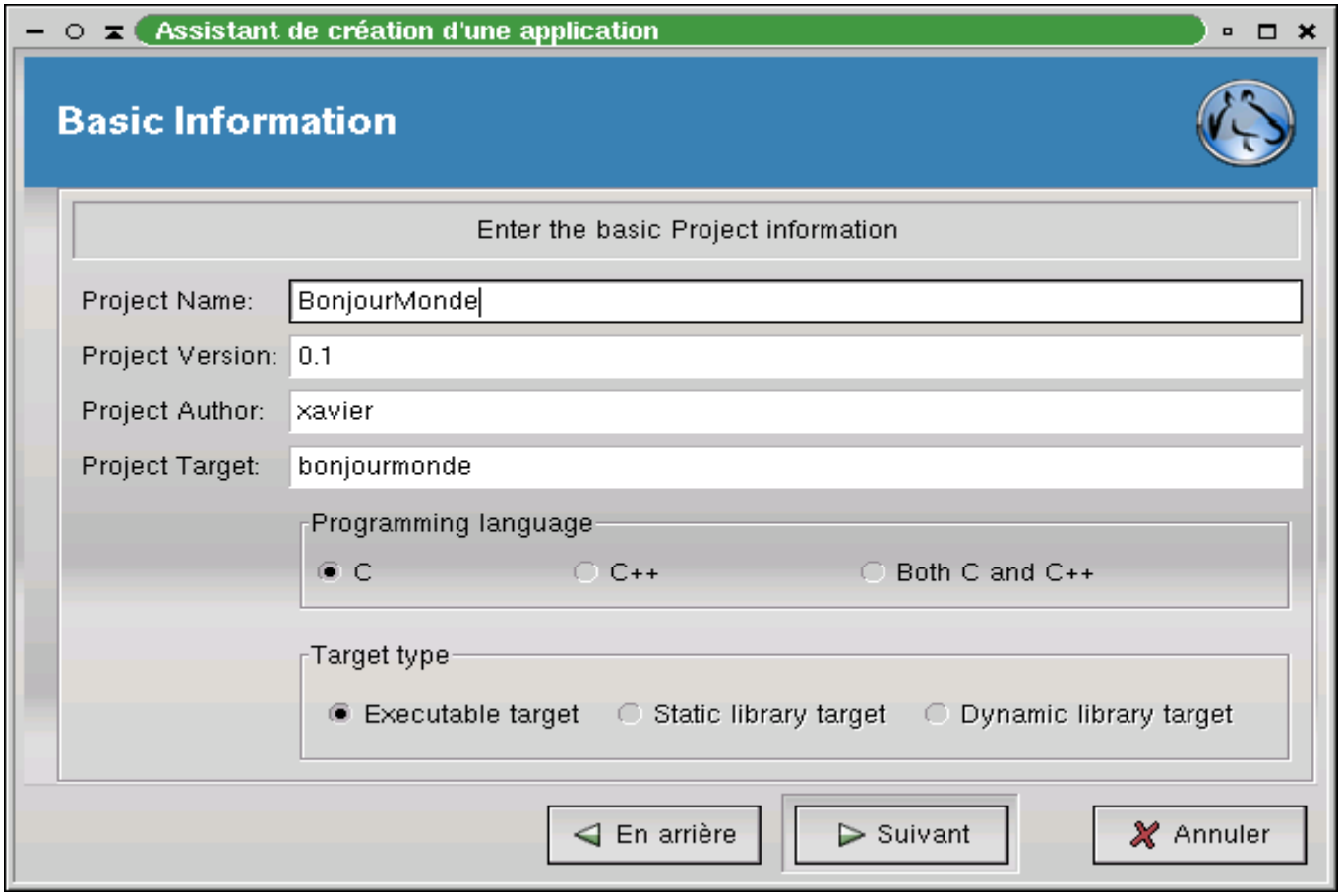
- Un éditeur convivial avec des lignes numérotées, la colorisation syntaxique, la possibilité de masquer les blocs de codes comme dans un explorateur de répertoires. Cet éditeur peut également être utilisé seul et supporte de multiples langages (C/C++/java/perl/php...).
- Le couplage avec glade pour l'édition de l'interface de votre application.
- Un outil de débogage intégré utilisant gdb qui apporte plusieurs facilités comme le placement de points d'arrêt dans le code.

- La possibilité d'éditer plusieurs documents dans la même fenêtre (à onglet, type gedit ou gnotepad+) et/ou de les "détacher" dans des fenêtres indépendantes.
- Le rappel des prototypes de fonctions lors de la saisie.
- La possibilité d'importer un répertoire de code pour en faire un projet dans anjuta. Les fichiers de configuration ne sont toutefois alors pas gérés.
- Un navigateur de code rappelant les classes, structures, fonctions, variables et macros présentes dans votre projet et permettant d'y accéder en un double clic, quelque soit le fichier dans lequel il se trouve. Cela s'avère très utile si vous récupérez le code d'un projet touffu auquel vous voulez collaborer (je l'ai utilisé récemment pour tenter de comprendre le fonctionnement de GCompris). Il reste toutefois à finaliser ; quelques problèmes subsistent encore comme l'"oubli" de fonctions, inexpliqué.
- Une fonction d'indentation du code entièrement paramétrable qui utilise le programme indent (man indent ou info indent vous en apprendra plus).
- La gestion des fichiers nécessaires à la construction du projet (configure, Makefile, etc...).
- Une belle interface en français.
- Des assistants pour créer vos projets (console/gtk+/gnome/gtkmm/gnomemm/composant bonobo/libglade) et les modifier ensuite par le biais de boîtes de dialogues.
- La possibilité de créer des signets dans votre code pour faciliter la navigation.
- La gestion partielle de CVS. (Vous devrez notamment faire le premier import vous même.)
- L'intégration de patches.
- Et bien d'autres choses encore ...



Le "Bonjour Monde d'anjuta"

Création du projet



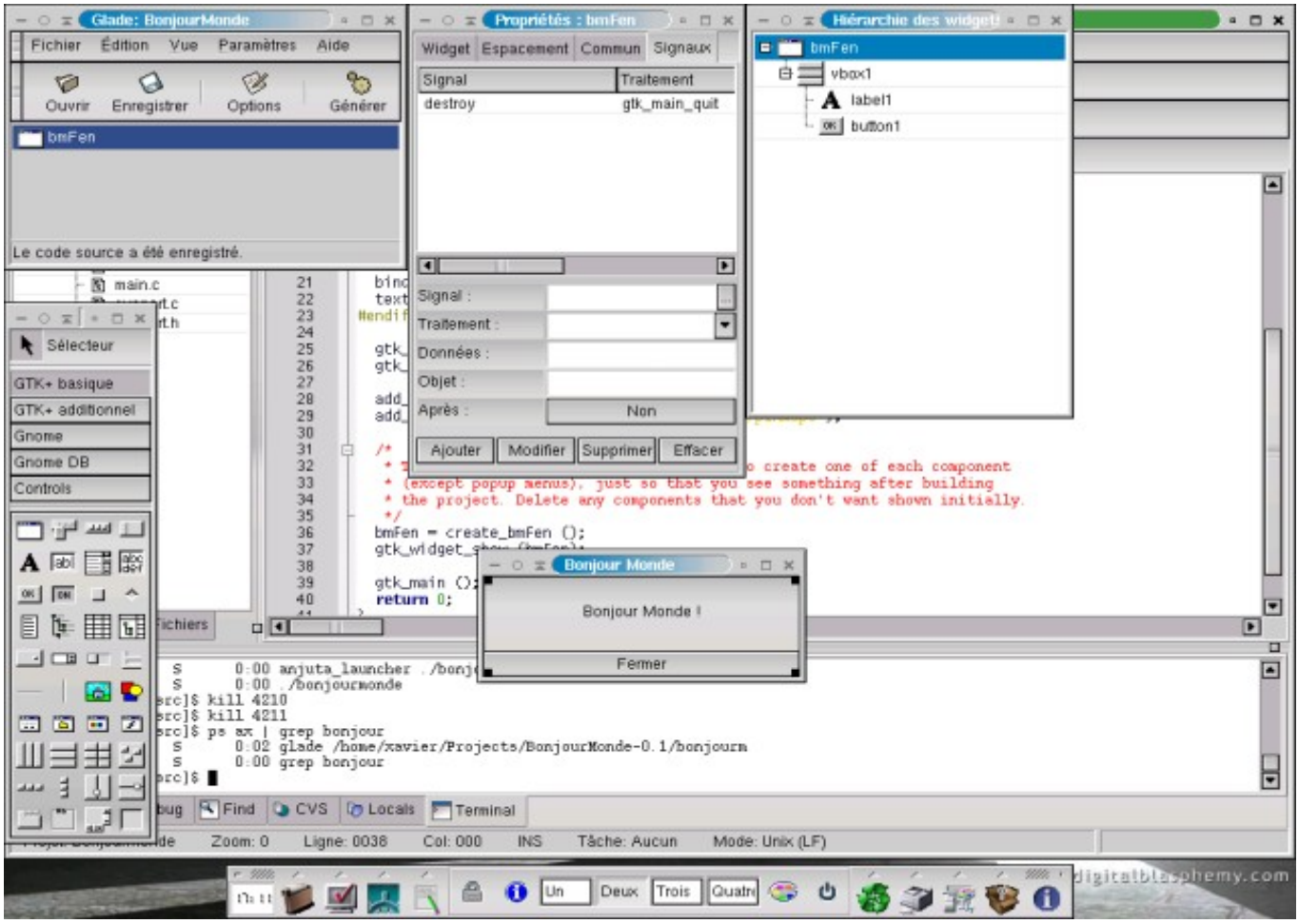
1. Démarrez anjuta.
2. Créez un projet en choisissant Nouveau Projet dans le menu Fichier.
3. Dans l'assistant, cliquez sur 'Suivant' et choisissez GTK Project (l'internationalisation n'est pas terminée et toute contribution est la bienvenue) puis recliquez sur 'Suivant'
4. Saisissez les informations demandées, laissez les options telles qu'elles sont et après quelques clics supplémentaires, vous devriez être en présence d'un message (en bas) vous annonçant que vous pouvez construire votre projet pour voir à quoi il ressemble ainsi que d'un arbre (à gauche) représentant les fichiers créés.
5. En double-cliquant sur le nom d'un fichier, vous pouvez l'ouvrir dans l'éditeur.
6. En choisissant l'onglet 'Symboles' du navigateur, vous obtenez non plus le nom des fichiers mais celui des structures, fonctions, macros, variables et classes (dans le cas du c++).
7. On construit le projet en choisissant 'Construire' dans le menu Construire (ou F10). Des messages défilent alors dans la zone dédiée à cet effet et vous indiquent fnalement, que tout s'est bien déroulé en N secondes.
8. On lance l'exécutable généré en choisissant 'Exécuter' dans le même menu (ou F3). On se retrouve avec à l'écran une fenêtre gtk+ vierge, ce qui correspond bien dans mon esprit à un début de projet gtk ;-)

Modification de l'interface

On va améliorer un petit peu tout celà en ajoutant un message et un bouton à cette

vilaine fenêtre. On choisit donc 'Editer IHM ..' dans le menu Projet et on se retrouve dans glade (si il est installé, sinon faites-le).

1. Cliquez sur window1 dans la fenêtre principale pour pouvoir éditer les propriétés de la fenêtre de l'application. Changez en le titre et le nom de façon à pouvoir vous y retrouver facilement.
2. Un double-clic sur un nom de fenêtre dans la fenêtre principale de glade permet de l'afficher afin d'en modifier le contenu, faites donc cela pour notre fenêtre puis, à l'aide de la boîte à outils, ajoutez une vbox de deux lignes et placez y un GtkLabel et un GtkButton.
3. Si la hiérarchie des widget n'est pas affichée, sélectionnez 'Montrer l'arborescence des widgets' dans le menu Vue et sélectionnez label1. Changez en l'Etiquette en "Bonjour Monde !". De même, changez celle de button1 en "Fermer"
4. Toujours avec button1 sélectionné, cliquez sur l'onglet 'Signaux' du dialogue des propriétés et affectez au signal 'released' le traitement gtk_main_quit (choisissez le signal puis le traitement et cliquez sur 'Ajouter').
5. Faites de même pour le signal destroy de la fenêtre (pour sélectionner les fenêtres on ne clique pas dans l'arborescence des widgets mais dans la fenêtre principale de glade).
6. Choisissez 'Enregistrer' puis 'Générer' et quittez glade. **ATTENTION !** Les fichiers modifiés par glade ne sont pas automatiquement rechargés dans anjuta ! C'est à vous de le faire si vous voulez les éditer. Quoiqu'il en soit, anjuta vous propose de recharger le fichier lorsque vous tentez de l'éditer si ce dernier est plus récent sur le disque que dans anjuta.
7. Vous devez éditer le fichier main.c pour qu'il reflète le nouveau nom de la fenêtre. Si vous l'avez renommée en bm_fen par exemple, changez create_window1 en create_bm_fen et window1 en bm_fen. Ensuite, construisez et lancez le projet.



Ajout d'un widget et personnalisation d'un callback

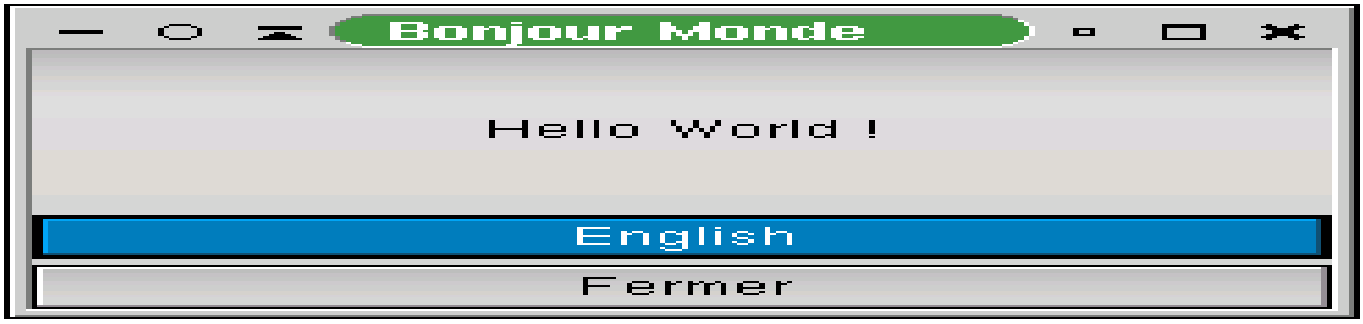
1. Relancez glade par le biais d'anjuta (Projet / Editez IHM ..).
2. Sélectionnez le label et nommez le 'message_label'.
3. Sélectionnez la vbox et passez sa taille à 3. et ajoutez un bouton dans l'espace ainsi créé.
4. Affectez à ce bouton l'Etiquette "English" et le nom lang_button.
5. Reliez enfin le signal clicked au callback 'on_lang_button_clicked' (valeur par défaut) et tapez 'message_label' dans le champ 'Objet'.
6. Générez et enregistrez

Glade a généré du code dans les fichiers interface.c et son en-tête interface.h, ces fichiers sont réécrits à chaque génération de code, il ne faut donc pas les modifier. Les fichiers que l'on édite sont callbacks.h et callbacks.c

Nous avons connecté le signal clicked du bouton lang_button à la fonction on_lang_button_clicked. Toutefois le premier paramètre de la fonction ne sera pas le bouton mais le label 'message_label' comme c'est le cas quand on utilise la fonction gtk_signal_connect_object. Vous pouvez consulter le code à la fin de la fonction create_bm_fen du fichier interface.c.
Etant donné que l'on passe un GtkObject en paramètre au callback, il faut remplacer son prototype en conséquence (celui qui a été créé marche mais ce n'est pas logique). Donc, dans callbacks.c et callbacks.h, remplacez le premier paramètre `GtkButton* button` par `GtkObject* object`.
Complétez ensuite le code du callback dans callbacks.c pour qu'il fasse quelque chose d'intelligent :

```
void on_lang_button_clicked (GtkObject *object, gpointer user_data) {
        gtk_label_set_text ((GtkLabel*)object, "Hello World !");
}
```

Je vous laisse taper F10 puis F3 pour constatez à quel point vous êtes forts en développement. Vous venez de créer une application gtk fonctionnelle en tapant **UNE** ligne de code. Pas mal non ?



Quelques fonctionnalités utiles non abordées dans cet exemple :

- Pour ajouter des librairies, chemins de recherche à gcc et ld sans éditer les makefile.am, la boîte de dialogue qu'il vous faut se trouve sous le menu 'Préférences' et s'appelle 'Compilateur/Editeur de liens'
- Pour ajouter des fichiers à votre projet, utilisez l'entrée 'Importer' du menu 'Projet'.
- Une fois votre projet prêt à distribuer, utilisez l'entrée 'Préparer distribution'

du menu Construire pour obtenir une archive du projet au format tar.gz dans le répertoire principal dudit projet.

A ce stade vous avez les bases pour utiliser anjuta et glade. Il vous reste à explorer les possibilités de ces logiciels.

Les améliorations à venir

L'équipe de développement d'anjuta souhaite une interaction plus "intime" entre anjuta et glade, en incorporant glade dans anjuta, sous forme d'un composant bonobo.
Un composant logiciel de création d'aide au format SGML est également dans les priorités des développeurs.

Les outils recommandés

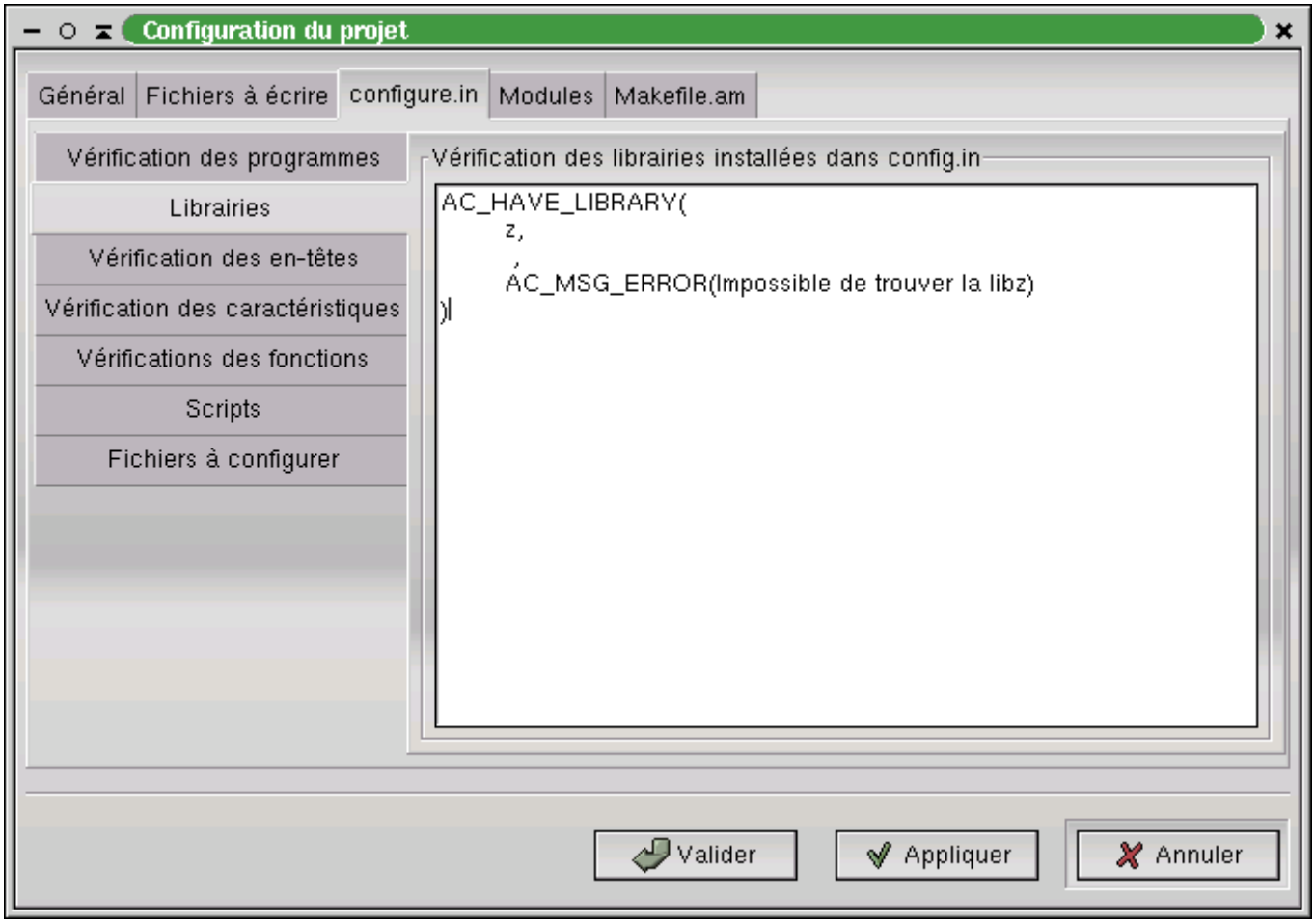
La page "Resources" du site regroupe diverses applications pouvant être utilisées en complément d'anjuta pour vous aider dans vos tâches quotidiennes.

- RUST, un utilitaire permettant de créer des RPM très facilement à partir de vos sources.
- DevHelp : Un navigateur d'aide simple mais efficace permettant de naviguer simplement dans vos manuels de références. Devhelp utilise des "livres" téléchargeables sur le site de devhelp et sur lidn (ci-dessous). C'est le système d'aide officiel d'anjuta.
- lidn : Le site sur lequel vous pourrez consultez et télécharger une multitude de livres pour devhelp.
- Dia et AutoDia : Le premier sert à créer des diagrammes, notamment UML mais aussi de circuits électriques, des grafcets, etc.... Le second créée à partir d'un répertoire de sources un diagramme UML vous permettant de vous y retrouver, au fromat Dia. Lors de l'écriture de cet article, les langages supportés étaient perl, c, c++ et java mais la documentation parle de php et on trouve un modpule PHP.pm dans le répertoire des Handlers. Il n'est toutefois pas installé lors du make install et n'est pas pris en compte dans le fichier de configuration 'Autodia.pm'.
- GTranslator : Un utilitaire complet pour l'internationalisation de vos programmes.

Et pour conclure

Evidemment, il y aura toujours des gens pour dire qu'il y a mieux que ce qu'utilisent les autres. Je laisse à chacun ses choix d'environnement de développement et évite de les critiquer. Le meilleur environnement est celui qui vous convient, que vous soyez partisan de vi, emacs, xemacs, nedit, kdevelop ou anjuta (ou tout autre). Comme le dit le splash screen d'anjuta : "the best is in you"
J'espère toutefois vous avoir convaincu qu'anjuta possède les qualités que l'on s'attend à trouver dans un EDI performant. La chose qui manque le plus à anjuta, outre l'intégration des composants annexes, est à mon avis une interface plus conviviale de modification du fichier configure.in. En effet, pour l'instant il vous faudra taper les macros comme vous le feriez de façon classique (mais tout de même dans une fenêtre à onglet, en fonction de la section que vous souhaitez

modifier). A titre d'exemple, si vous ajoutez une librairie dans les options de gcc et ld, le test de sa présence sur le système cible devrait être ajouté dans le script configure.in.



Xavier GARREAU - http://www.xgarreau.org/ - <xavier@xgarreau.org>

Ingénieur de recherche PRIM'TIME TECHNOLOGY
http://www.prim-time.com/

Président du ROCHELUG
http://lug.larochelle.tuxfamily.org/

Fiche d'identité :

Version : 0.9.99 alias 1.0beta1
Licence : GNU/GPL

Liens:

Anjuta : http://anjuta.sourceforge.net/
RUST : http://www.rusthq.com/
DevHelp : http://devhelp.codefactory.se/
lidn : http://lidn.sourceforge.net/
Dia : http://www.lysator.liu.se/~alla/dia/
AutoDia : http://droogs.org/autodia/
gtranslator : http://www.gtranslator.org/