Prise en main de l'outil RTDS: illustration avec le bit alterné

Marc Boyer

Novembre 2018

1 Lancement de l'outil

Avant de lancer l'outil, il faut dire à votre shell où le trouver, où trouver le serveur de licences, etc.

D'abord, il faut savoir quel est votre shell :

```
echo $SHELL
```

Si votre shell est `bash`, il vous faut taper un terminal la commande

```
source /mnt/n7fs/rtds5.0/pragmastudio-config.sh
```

sinon, si c'est `tcsh`, la commande est

```
source /mnt/n7fs/rtds5.0/pragmastudio-config.csh
```

2 Commencer un projet

2.1 Créer un projet

- créer un répertoire de travail (`TP-SDL` par exemple) et vous y positionnez
- créer un nouveau système SDL par le menu *Project* → *New Projet* → *SDL Z100 project* (ne pas choisir SDL-RT, qui est un autre langage !)
- donner un nom au système par le menu *Projet* → *Save as*

2.2 Créer le système

- Click-Droit sur le nom du projet, sélectionner *Add child element*. Dans la fenêtre qui s'ouvre, sélectionner *Active architecture* → *System*, lui donner un nom, puis clicker sur *New* pour créer le fichier dans lequel sera sauvé le composant.
  C'est un principe général de l'outil : chaque composant est placé dans un fichier. Par défaut, chaque composant d'un système est sauvé dans un fichier qui porte le nom du composant avec l'extension `.rdd`.
  Ceci rend difficile le fait d'avoir plusieurs composants ayant le même nom, alors que le langage le permet.

— Double-Click sur le composant, cela ouvre une nouvelle fenêtre, dans laquelle on peut dessiner le composant

3 Mettre les premiers composants dans le système

3.1 Créer l'architecture du nouveau système

Les différents éléments que l'on peut ajouter sont sous forme graphique dans la barre gauche de l'éditeur.

— Placer un bloc `Text` et utiliser le pour entrer la définition de deux signaux
`SIGNAL Data, Ack;`
Pour éditer le contenu d'un bloc texte, vous faites un Double-Click dessus et entrez le texte.
— Placer deux *Block* en vis à vis, l'un nommé `Emetteur` et l'autre `Recepteur`
— tirer un canal (*Channel*) entre les deux blocs ; pour cela, sélectionner *Channel*, Click-Droit sur un des deux blocs puis tirer le fil jusqu'à l'autre en maintenant le bouton appuyé
— nommer le canal (qui s'appelle par défaut `(new channel)`)
— mettre les signaux dans les crochets aux deux extrémités du fil ; pour cela deux méthodes
  — Double-Click sur la zone et taper le texte depuis le clavier
  — Double-Click sur la zone, puis taper *F8*, qui vous propose une liste de choix possibles.

3.2 Définir le bloc `Emetteur`

— Double-Click dans le cadre de l'`Emetteur` mais pas sur son nom ; accepter la création du composant avec son type et son nom par défaut ; une nouvelle fenêtre d'édition graphique apparaît
— insérer un processus dans le bloc, nommé `ProcEmetteur`
— tirer un canal entre le processus et un bord du bloc :
  — nommer le canal
  — nommer son connecteur vers l'extérieur (*F8* fonctionne)
  — entrer les signaux
— revenir à la fenêtre *Project* et faite un *Save all in project*
Pour naviguer entre les fenêtres, vous pouvez utiliser les onglets en haut de chaque fenêtre, ou l'arbre de vue générale du projet.
— Tetourner à l'`Emetteur` (onglet *Emetteur*), et vérifier que le bloc est correct : *Diagram* → *Check syntax/semantics...*
Il doit se plaindre que le processus `ProcEmetteur` n'existe pas. Créer le avec un Double-Click sur le rectangle aux angles bisautés représentant le processus dans le bloc. Sauvez tout et refaites la vérification du bloc `Emetteur`. Un message *Everything's fine* devrait apparaître en bas.

3.3 Définir le processus `ProcEmetteur`

Nous allons faire un processus simpliste, à deux états. Depuis l'état initial, il lance une occurrence de `Data`, passe dans un état `wait`. Depuis cet état, il peut recevoir un `Ack` et se termine alors.

— dans l'éditeur graphique de `ProcEmetteur`, insérer un état *start*
— garder cet état sélectionné ; Double-Click sur le symbole d'envoi de signal (*Send message*) ; donner `Data` comme type de signal envoyé (au clavier ou par *F8*)
De manière générale, vous pouvez insérer automatiquement un nouveau symbole sous un autre symbole en gardant le symbole sélectionné et en en faisant un Double-Click sur le nouveau symbole.
— insérer en dessous un état d'arrivée `wait_ack`
— créer un état `wait_ack` (au clavier ou par *F8*) ; insérer en dessous un symbole de réception de signal `Ack` ; terminer la transition par la terminaison du processus

3.4 Définir la partie réceptrice

Suivant le même principe, faites le bloc `Recepteur` et son processus.

4 Prise en main de l'outil de *debug*

4.1 Contourner un bug de l'outil

Une petite manipulation pour contourner un bug de l'outil : fenêtre *Projet*, dans le *Generation*, sélectionnez *Options...*. Une fenêtre *Generation options* s'ouvre, sélectionnez le profil *Simulation options* (colonne de gauche), puis activez l'option *Manage all types in a single system-wide scope*.

4.2 Compiler le système

— dans la fenêtre *Projet*, sélectionner le système
— compiler en appuyant sur le bouton *Build*

Il est important de sélectionner ce qu'on cherche à simuler, car l'outil permet de ne simuler qu'un sous-composant.

4.3 Utiliser l'outil de *Debug*

— dans la fenêtre *project*, appuyer sur *Debug* ; cela ouvre une nouvelle fenêtre dans laquelle nous allons désormais travailler.
Nous allons voir plusieurs méthodes de simulation.
— lancer l'outil de trace MSC par *Start MSC trace* ; une fenêtre de trace MSC se crée

Déroulement automatique appuyer sur *Run* ; le système s'exécute automatiquement et les échanges apparaissent dans la trace MSC.

Fermer la trace sans la sauver, et relancer l'outil de *Debug* par *Debugeur* → *Restart*.

Déroulement pas à pas appuyer sur *Run until end of the transition* ; cela fait avancer le système d'une transition ; la trace MSC est mise à jour ; répéter l'action jusqu'à la fin du système

Déroulement petit pas par petit pas utiliser cette fois *Step in code* ; vous pouvez alors visualiser dans les fenêtres d'édition les états intermédiaires (surlignés en jaunes)

Point d'arrêt par une fenêtre d'édition, ajouter un *Breakpoint* dans votre modèle, puis exécuter le avec *Run*

Remarque générale : la vue n'est pas toujours automatiquement mise à jour. Pour le faire, faite *Stop* suivit de *Refresh*.

Lors du projet lui même, lorsque de nombreux processus seront présents, les traces MSC peuvent être difficile à lire. Vous pouvez configurer deux choses :

— choisir les éléments qui seront affichés dans la trace : bouton *Configure MSC trace*, l'usage est intuitif

— choisir l'ordre d'affichage des processus : c'est un peu plus délicat. En fait, il apparaissent dans leur ordre de création, et on peut changer cet ordre en allant modifier l'ordre des signaux de création des processus. Quand une nouvelle simulation est crée, vous voyez dans la zone *SDL system queue* une liste de `RTDS_startMessage`. Avec un Double-Click (gauche) sur le chiffre du `Pid` destinataire, vous faite remonter le signal de création en tête de la liste, et donc, par effet de bord, l'ordre d'apparition dans la fenêtre *MSC Tracer*.

4.4 Envoi d'occurrence de signal par l'utilisateur

Commençons par un signal sans paramètre.

— Modifier le modèle pour que l'émetteur attende de l'environnement un signal `Start` pour envoyer `Data`.

— Recompiler et relancer le *Debugeur*.

— Pour envoyer le message, appuyer sur *Send an SDL message*. Une fenêtre apparaît. Sélectionner le type de message, le récepteur (notez que la liste des récepteurs se réduit lorsque vous avez sélectionné un type de message aux seuls processus capable de recevoir un message de ce type).

Les signaux peuvent aussi transporter des valeurs en paramètre. La fenêtre *Send an SDL message* possède sur la droite une zone qui permet d'entrer les valeurs des paramètres. Si l'ensemble des paramètres devient long, on peut sauvegarder le message dans un fichier texte.

5 Un modèle plus réaliste

Nous vous demandons de faire une modification plus réaliste du protocole du bit alterné, en plusieurs étapes.

L'objectif est d'avoir une petite pile protocolaire, qui gère un multiplexage dynamique, la fiabilisation du transfer, le tout sur un médium à perte et dédoublement.

Nous vous proposons de procéder par étapes :

1. Gérez un protocole qui reçoit des données (chaînes de caractères) de l'utilisateur, les insère dans les messages de donnée et les délivre à l'arrivée.

   Ne pas oublier de délivrer les messages à l'utilisateur.

2. Branchez entre les deux entités une couche qui dédouble certains paquets. Soit de manière déterministe (un sur trois pas exemple), soit de manière aléatoire (avec la commande `ANY`).

   Elle doit pouvoir dédoubler les messages de données comme les acquittements.

3. Ajoutez à votre couche la possibilité de *perdre* des messages, de donnée comme d'acquittements.

4. Fiabilisez maintenant votre connexion suivant le principe du "bit alterné", dit aussi "fenêtre glissante de taille 1". Ajoutez un bit dans les messages de donnée et d'acquittement, et la gestion des ré-émissions.

   À ce niveau de votre formation, nous ne rappellerons pas le fonctionnement du protocole : si vous avez oublié, réfléchissez un coup ou chercher dans Google, en fonction de vos capacités.

   Si vous décidez d'utiliser des timers, dans la fenêtre "SDL simulator", allez activer l'option 'Options → Timer → Real time timers'.

   Pensez à sauvegarder des traces MSC pour illustrer le fonctionnement.

5. Gérer plusieurs connections fiabilisées par le bit alterné suivant le modèle vu en cours.

   Essayer de réfléchir au niveau de multiplexage : est-ce qu'il faut une valeur de bit par connexion active, ou est-elle partagée entre les différentes connexions.

Nous vous rappelons que l'objectif n'est pas que "ça marche" mais que cela soit bien conçu (cf. cours).

Annexe

Vous pouvez trouver la documentation du logiciel à `/mnt/n7fs/rtds/doc/`