Réalisation d' une interface graphique d'aide à la spécification de comportements de composants

Contact
Annie Ressouche : annie.ressouche@sophia.inria.fr - Tel 04 92 38 79 44

5 Octobre 2006

1 Contexte scientifique

L' équipe Orion s'intéresse à la conception de systèmes intelligents réutilisables. En particulier, la réutilisation d'outils de conception de systèmes à base de connaissances est un sujet de recherche étudié depuis longtemps à Orion. En nous appuyant sur les résultats de l'état de l'art en génie logiciel et en langages à objet, nous proposons une plate-forme de composants (**Lama**) afin de faciliter la construction de systèmes à base de connaissances (SBC).

1.1 La plate-forme Lama

Lama est une plate-forme générique de composants extensibles et portables. Le coeur de la plate-forme est une librairie de composants réutilisables (Blocks) implémentant des structures de données génériques et des méthodes pour la définition de SBC. L'objectif recherché avec l' utilisation de Blocks est d'écrire de nouveaux moteurs de SBC par extension et adaptation de l'existant en réécrivant un minimum de code. Lama a déjà été utilisé pour construire différents moteurs dans des domaines variés: pilotage de programmes, classification, calage de modèles numériques. Son utilisation a simplifié le développement de façon importante en comparaison avec la quantité de travail nécessaire pour développer un moteur à partir de rien.

1.2 Vérification des moteurs

Pour assurer une utilisation correcte des composants de la librairie, nous avons spécifié un outil de vérification du comportement des moteurs construits, et nous aimerions réaliser cet outil. Les composants de Blocks sont utilisés par composition ou par dérivation et nous avons choisi d'utiliser des techniques de "model-checking" pour vérifier la cohérence de l'assemblage des composants de Blocks. Nous avons proposé un modèle mathématique et un langage pour décrire le comportement des moteurs. Un outil de vérification peut ensuite assurer l'utilisation sûre de la librairie.

1.3 Outil de vérification des moteurs

La réalisation pratique d'un outil de vérification de la correction de l'utilisation de la librairie est un développement nécessaire pour Orion dans sa volonté d'offrir des outils d'aide à la construction de moteurs. Un développeur de moteur doit pouvoir utiliser la librairie en dérivant et composant ses éléments tout en gardant la cohérence globale de celle-ci.

Les travaux théoriques réalisés ces deux dernières années à Orion s'appuient sur une représentation effective des comportements des composants de la librairie. Pour exprimer ces comportements nous avons défini un langage (**BDL**) décrivant des automates (ou machines à états finis) hiérarchiques et concurrents. Un programme BDL représente le comportement d'un composant, éventuellement à partir de la représentation des comportements de ses sous-composants. La sémantique de BDL repose sur le modèle des systèmes réactifs synchrones ("à la Esterel") et le langage peut servir d'entrée à un model-checker comme **NuSMV**.

2 Description du projet

Le but du projet est de créer une interface graphique pour spécifier des automates hiérarchiques parallèles. Ces automates ont un modèle de calcul synchrone et l'outil graphique souhaité devra aussi permettre leur simulation en respectant cette sémantique. Dans le cadre du projet, les automates concernés seront des programmes BDL mais nous souhaitons une interface générique car le projet Orion (et le futur projet Pulsar) est demandeur de ce type d'outil dans d'autres domaines, par exemple la définition de scénarios dans une plate-forme dédiée à la reconnaissance de comportements.

2.1 L'interface graphique

L'interface graphique que nous souhaitons a 3 fonctionalités principales

1. dessiner des automates hiérarchiques et parallèles;
2. simuler leur fonctionnement suivant un modèle de calcul synchrone;
3. optionnellement générer du code BDL pour une utilisation par d'autres outils;

2.1.1 Le langage BDL

Dans un premier temps, cette interface graphique devra nous permettre de décrire des programmes BDL. BDL est un langage complètement défini avec une syntaxe précise spécifiée à l'aide de l'outil de génération d'analyseurs syntaxiques YACC et dont la sémantique mathématique a été longuement étudiée (un rapport de recherche Inria et des papiers de conférences existent). Un programme BDL est soit un automate de base, soit une composition hérarchique et parallèle de programmes. Plus précisément, un automate de base est composé d'un nombre fini d'états et d'une relation de transition entre les états. Les transitions portent des étiquettes composées de deux ensembles de signaux: les signaux d'entrée qui déclenchent la transition et les signaux de sortie qui

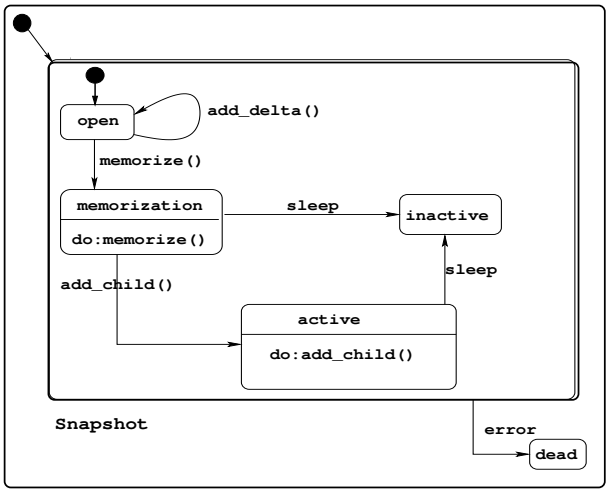

Figure 1: Exemple de représentation d'un automate hiérarchique. Le programme principal est composé de 2 états. L'état initial est raffiné en 4 sous-états.

sont émis. La figure 1 représente un programme BDL. Les opérateurs du langage sont au nombre de 3:

1. l'opérateur de parallélisme. Sémantiquement, il peut être considéré comme un produit d'automates. D'un point de vue graphique c'est la juxtaposition de sous-programmes.

2. l'opérateur de hiérarchie: il permet de raffiner un état d'un automate en un sous programme. C'est le plus délicat à représenter graphiquement car on doit pouvoir effectivement changer le statut d'un état de simple état à état raffiné. L'ergonomie de l'outil graphique est à spécifier pour cet opérateur.

3. l'opérateur d'encapsulation qui permet de cacher des signaux. Il est surtout utilisé pour assurer la communication entre les deux branches d'un programme parallèle. Graphiquement, il s'agit de représenter la liste des signaux encapsulés.

De plus, chaque programme a une interface de signaux d'entrée et de signaux émis qui doit pouvoir être visualisée.

2.1.2 Dessiner avec l'interface graphique

La première fonction de l'outil est la **représentation de programmes BDL**. La figure 2 décrit succintement ce que l'on voudrait dessiner et maintenir. Il faut tout d'abord pouvoir spécifier graphiquement un programme. Pour cela, l'outil devra offrir des fonctionnalités appropriées, c'est-à-dire:

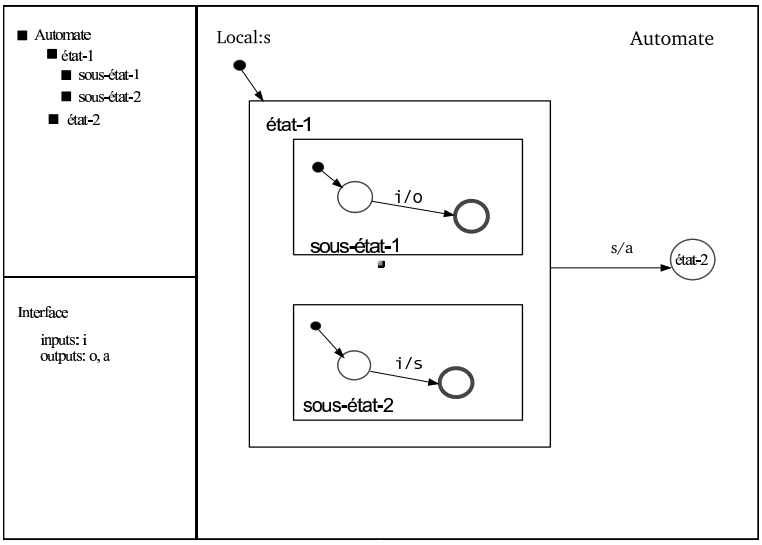

Figure 2: Exemple de présentation d'automate hiérarchique parallèle

- donner plusieurs vues de l'objet: on doit avoir simultanément le dessin et la structure dessinée sous forme d'arbre (comme le montre la vue en haut à gauche de la figure 2);
- tenir compte du coté hiérarchique des automates: un simple état doit pouvoir être raffiné en un sous programme, sur le dessin et/ou en ouvrant une autre fenêtre (les deux possibilités doivent coexister, le choix étant laissé à l'utilisateur);
- être facile d'utilisation au cours de la construction: on doit pouvoir ajouter et détruire des états et des transitions suivant différentes politiques : nous voulons respecter la sémantique de BDL tout en facilitant l'édition graphique (par exemple, la destruction d'une transition ne s'accompagne pas toujours de la destruction de l'étiquette et de celle de l'état d'arrivée);
- vérifier la cohérence du dessin: par exemple les signaux des étiquettes doivent appartenir à la sorte de l'état [1], on ne peut pas traverser les frontières des macro-états avec une transition, etc..;
- sauvergarder et recharger des spécifications à l'identique, y compris graphiquement;

La liste ci-dessus n'est pas exhaustive et la spécification exacte des fonctionalités de l'éditeur graphique fera l'objet d'un travail commun entre le développeur et nous. Pour les définir , on peut s'inspirer d'outils existants comme **Rose-RT**, **EsterelStudio** ou

[1] La sorte d'un état est l'ensemble des signaux qui apparaissent dans les étiquettes des transitions issues de cet état.

Scade. L'outil que l'on envisage est très proche de ces deux derniers environnements et on peut s'en inspirer de façons positive et négative pour définir les spécifications précises de notre éditeur.

2.1.3 Simuler avec l'interface graphique

La deuxième fonctionnalité fondamentale est la possibilité **de simuler le comportement d'un programme**. Les programmes BDL obéissent à la sémantique synchrone que nous étudions depuis longtemps. L'algorithme qui fait le calcul d'un pas de simulation pour les automates synchrones est connu et nous nous engageons à fournir son implémentation. Intégrer des fonctionnalités de simulation à l' éditeur graphique signifie:

- permettre à l'utilisateur de spécifier une séquence d'entrée et des suites de séquences d'entrée;
- afficher à l'utilisateur les résultats de la simulation : états actifs dans chaque sous programmes, signaux émis, ect...;
- simuler pas à pas sur une séquence d'entrée ou tout un scenario;
- permetter à l'utilisateur de poser des points d'arrêt au cours d'une simulation sur un état, une transition, l'émission d'un signal;
- proposer à l'utilisateur les entrées succeptibles de provoquer un changement d'état dans un état actif;
- etc....

Les fonctionalités du simulateur sont incrémentables presque à l'infini.... Le travail consiste également à faire un tri et à définir des priorités avec nous. La caractéristique la plus importante pour nous est l'ouverture du simulateur à d'autres fonctionnalités implémentées par d'autres personnes que son concepteur.

2.1.4 La génération de code (optionnelle)

Enfin, la dernière fonctionnalité importante est la **génération de code textuel BDL** à partir de la syntaxe abstraire manipulée par l'outil graphique. Les outils d'analyse et de vérification que nous développons acceptent comme entrée des programmes BDL textuel. Par exemple, pour l'utilisation de NuSMV, nous avons déjà une traduction de BDL textuel vers le format accepté par NuSMV.

2.2 Conception de l'interface graphique

Afin d'avoir une interface graphique réutilisable et extensible, nous suggérons d'utiliser le meta environnement de développement intégré ECLIPSE et son générateur d'éditeur graphique GEF (Graphical Editor Framework) couplé avec EMF (Eclipse Modeling Framework). Cet environnement fournit un squelette d' éditeur graphique à partir

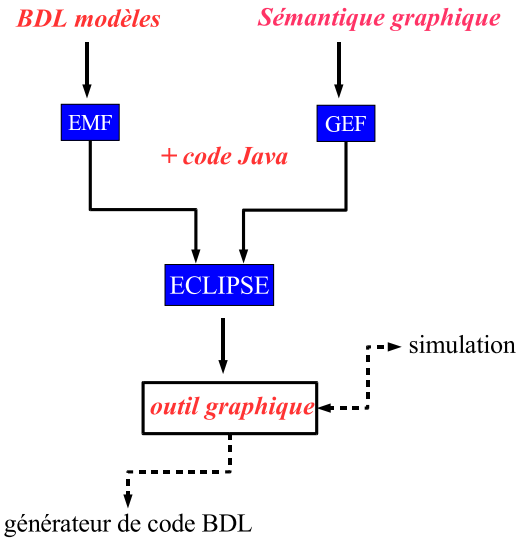

Figure 3: Conception de l'outil graphique

d'un modèle conceptuel et permet de créer l'éditeur moyennant toutefois la programmation de certaines parties en Java. Il assure la correspondance entre les éléments du modèle et leurs représentations graphiques à l'aide de "modèle vue controleurs" (MVC). ECLIPSE est écrit en Java et l'implémentation de ces controleurs fournit automatiquement un ensemble de classes abstraites que l'on peut étendre. L'éditeur utilise la librairie graphique SWT d'ECLIPSE. L'avantage d'une telle approche est de bénéficier d'un certain nombres d'opérations générales fournies par ECLIPSE et d'avoir seulement à les étendre.

La figure 3 suggère un schéma de développement de l'outil graphique. La plateforme ECLIPSE est construite autour de la notion de "plugins". Un plugin est un ensemble de code java et de données qui peut ajouter des fonctionnalités à la plate-forme. Cette dernière est elle-même construite à partir de la notion de plugins. Deux frameworks complémentaires permettent de construire des éditeurs graphiques: GEF est un framework pour construire des éditeurs graphiques à partir de modèles et EMF permet de construire et de manipuler ces modèles.

Dans le cadre de notre application, il faudra définir un modèle EMF du langage BDL. EMF est un framework dédié à la modélisation et à la génération de code pour des applications basées sur des données structurées (modèles). Il produit des classes Java pour ces modèles et un ensemble de classes pour visualiser et éditer un modèle ainsi qu'un éditeur de base. Les modèles peuvent être décrits en Java annoté, en XML, entre autres formats possibles.

GEF permet de créer un éditeur à partir d'un modèle. Il comporte une boite à

outil pour visualiser des graphes et le développeur peut utiliser un certain nombre d'opérations de base et les étendre. Il faut définir une sémantique graphique pour faire le lien entre la visualisation et le modèle.

Néanmoins, il reste une partie de code Java à écrire pour implémenter réellement l'interface graphique à partir des classes produites par EMF et GEF.

EMF et GMF semblent bien adaptés à nos besoins car BDL peut se décrire sous forme de modèles EMF et un générateur de code BDL textuel pourra être facilement connecté à l'outil graphique ainsi qu' un simulateur. On pourra se baser sur GEF pour la partie purement graphique et cette approche permettra la généricité nécessaire à la réutilisation de l'outil.

Par ailleurs, l'environnement ECLIPSE, EMF/GEF fait l'objet d'une abondante documentation sur le Web.

2.3 Compétence requises

La connaissance de l'environnement intégré Eclipse et du langage Java sont nécessaires à l'élaboration du projet.

3 Conclusion

La vérification automatique d'une utilisation correcte de la librairie Blocks de Lama est un outil fondamental pour rendre la plate-forme robuste et fiable. L'utilisation de méthodes formelless nous assure du bien fondé de la démarche. Toutefois, cette approche repose sur la description des comportements des composants de la bibliothèque et un outil graphique est primordial pour exprimer ces comportements et obtenir un vérifieur convivial qui sera à terme incorporé à Lama.