Grimm, comment ça marche ?

Adel Ferdjoukh, Lirmm

Table des matières

1 Qu'est ce que Grimm ? **2**

2 Conditions de fonctionnement **2**

3 Exécution **2**

4 Workflow **3**

5 Exemple **4**

Références **4**

1 Qu'est ce que Grimm ?

Grimm est un outil codé en *EMF/Java* permettant de générer des modèles conformes à des méta-modèles écrits en *ecore*.

L'outil prend en entrée un méta-modèle conforme à *ecore* et des données de configuration sur les modèles à générer.

Il fournit deux types de sorties. La première est un modèle au format *xmi* manipulable dans eclipse et la deuxième est la représentation visuelle du modèle généré qui est réalisée grâce à l'outil *dot* de *GraphViz*.

2 Conditions de fonctionnement

La machine d'un utilisateur qui veut faire fonctionner Grimm doit réunir les conditions nécessaires et suffisantes suivantes :

1. Installer la JVM (*The Java Virtual Machine*).
2. Télécharger l'archive Grimm qui embarque aussi le solveur CSP Abscon.
3. Installer *GraphViz* pour pouvoir visualiser les modèles générés.

3 Exécution

Après avoir réuni les conditions de fonctionnement de Grimm, vous pouvez passer à l'étape de fonctionnement. Pour cela, il faut d'abord extraire les fichiers de l'archive Grimm. Ensuite, il faudra suivre les étapes suivantes :

1. Choisir un méta-modèle parmi ceux qui sont joints à l'archive.
2. Regarder le tableau de la figure 1 pour récupérer la classe racine du méta-modèle choisi.
3. Lancer un terminal et exécuter la commande :

```
java -jar grimm.jar mm.ecore root lb ub rb sym
```

ou :
mm.ecore : le chemin du fichier du méta-modèle choisi.
root : la classe racine du méta-modèle mm.
lb : la borne inférieure du nombre d'instances par classe.
ub : la borne supérieure du nombre d'instances par classe.
rb : la borne supérieure du nombre d'instances par référence.
sym : Casser ou non les symmétries entre les variables des références $\{0, 1\}$.

4. Suivre les instructions sur le terminal, les modèles générés sont sauvegardés dans un dossier *root* (différent pour chaque méta-modèle).

Méta-modèle	Chemin	Classe racine
Test	test.ecore	Compo
PetriNet	PetriNet.ecore	PetriNet
Entities & Relashionships	ER.ecore	Schema
Royal and Loyal	RoyalAndLoyal.ecore	LoyaltyProgram
BibTex	BIBTEXML.ecore	BibtexFile
BMethod	BMethod.ecore	BSpec
Sad	Sad3.ecore	DocumentRoot
Diagraph	Diagraph.ecore	GraphModel
Business Process	BusinessProcessModel.ecore	CoumpoundTask
MyUML	MyUML.ecore	Model
Ecore	Ecore.ecore	EClass
Jess	jess.ecore	JessModel

FIGURE 1 – Liste des méta-modèles avec leurs classe racine.

4 Workflow

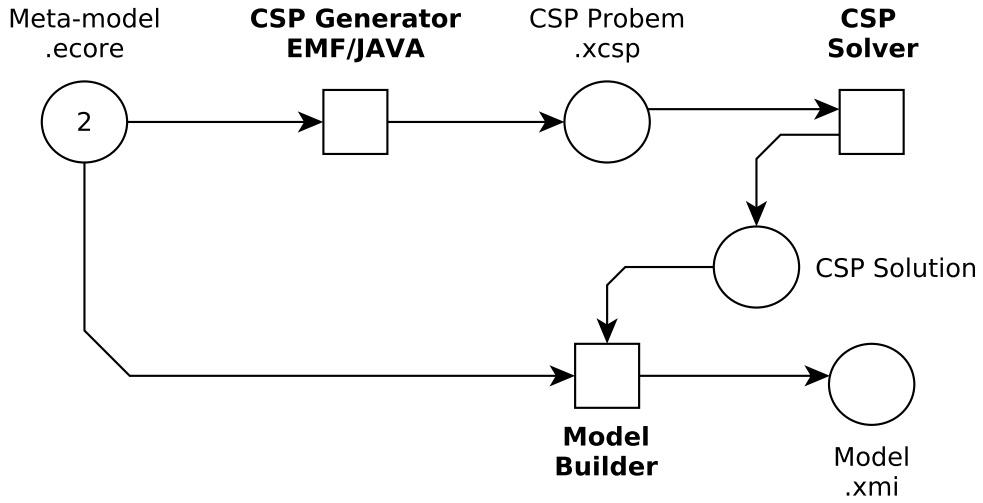

FIGURE 2 – Étapes du processus de génération de modèles par l'outil Grimm.

Voici les étapes du processus de génération de modèles à l'aide de l'outil Grimm :

1. À partir d'un méta-modèle en entrée, générer l'instance CSP correspondante et cela en suivant la modélisation d'un méta-modèle en CSP expliquée dans notre article [FBC+13].
2. Résoudre l'instance CSP obtenue par le solveur *Abscon* ([MLB01]).
3. Construire un modèle valide (au format xmi et visuel) en partant de la solution retournée par le solveur de contraintes.

Ces trois différentes étapes sont également montrées dans le réseau de Petri, figure 2.

5 Exemple

Pour générer un modèle conforme au méta-modèle *test.ecore*, vous devez exécuter la commande suivante :

```
java -jar grimm.jar test.ecore Compo 2 2 4
```

Le modèle généré sera sauvegardé dans le dossier *Compo* dans le répertoire courant au format pdf.

Références

[FBC+13] Adel Ferdjoukh, Anne-Elisabeth Baert, Annie Chateau, Rémi Coletta, and Clémentine Nebut. A csp approach for metamodel instantiation. In *ICTAI 2013, Internationnal Conference on Tools with Artificial Intelligence, November 4-6, Whashington D.C., USA.*, 2013.

[MLB01] Sylvain Merchez, Christophe Lecoutre, and Frédéric Boussemart. Abscon : A prototype to solve csps with abstraction. In *CP*, pages 730–744, 2001.