Domain Specific Languages

Kickoff Lab

Benzarti Zied

Destefanis Marc

Lozach Maxime

Introduction

Dans le cadre du cours Domain Specific Languages (DSL) nous avons réalisé une étude comparative de trois manière d'effectuer un langage de ce type. Un premier langage embarqué réalisé en Python, et deux externes avec xtext et MPS.
Dans notre cas, le but du DSL doit être de faciliter la création de code Arduino. Nos trois études reposent sur un exemple simple d'allumage d'une diode grâce à un bouton.
Afin d'avoir un avis le plus objectif, nous avons décidé que chaque membre devait s'occuper d'un langage et à la fin chacun devait faire un résumé et expliquer la maniere de fonctionner du langage. Cette methode permet d'avoir l'avis d'une personne qui connait le langage (celui qui a suivi le tutoriel) et les avis de deux personnes qui ne le connaissent pas du tout et donc qui pourront évaluer la simplicité de comprehension du langage par rapport à celui qu'ils ont étudiés.

I. Python

Description de l'implementation

Bien qu'ayant travaillé sur python dans ce cas-là, l'implémentation décrite ici est la meme pour tous les autres langages embarquées. Seule la syntaxe sera différente. Il n'y avait pas de tutoriel fourni pour les langages embarqués. Ce qui rendait la comprehension plus compliquée étant donné qu'on avait directement le code fonctionnel. Nous avons donc lu chaque fichier du projet afin de comprendre son fonctionnement. Tout d'abord, le modele doit être ecrit. Cela correspond aux differents objets que nous allons manipuler. Le modele est donc ecrit dans le meme langage que le code qui genère le code Arduino. L'ecriture du modele est la partie la plus simple du projet bien qu'il fasse faire attention à ce que chaque "classe" renvoie sur la console. Ensuite nous devons ecrire la methodchaining qui utilise les differents modeles. Cela permettra d'ajouter les methodes associés à chaque element comme la methode onPin qu'on associera à une Brick et non pas à une transition.
On peut voir que l'écriture à partir d'un code embarqué se fait donc en 2 etapes.

Pros

- Utilisation d'un langage que l'on connait. Dans cette exemple nous utilisons python mais nous pouvons utiliser d'autres langages que nous connaissons plus comme Java. De plus on a tous les avantages du langage utilisé (documentation, debug, etc…).
- Principe assez simple à comprendre : ecriture du modele puis methodechaining.
- Une fois ecrit, utilisation du code tres simple (but du DSL).

Cons

- On doit absolument tout écrire à la virgule près. La génération du code Arduino se fait à l'aide de print.
- Le code est facile à comprendre et utiliser mais bien plus compliqué pour ajouter de nouvelles fonctionnalités.

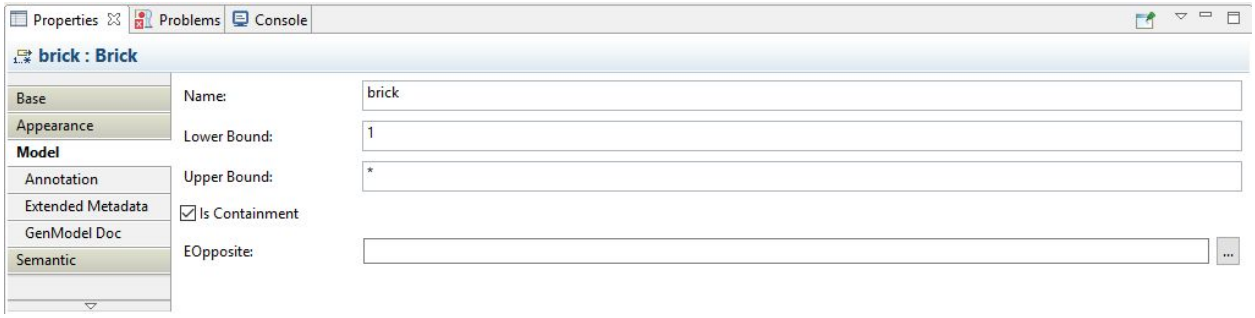
II. XText

Description de l'implementation

L'implémentation a été effectuée en suivant le tutoriel disponible sur le github correspondant au cours.
Dans un premier temps on créé graphiquement le model en sélectionnant les éléments dans la palette et en les glissant sur notre schéma représentant notre model. On ajoute ensuite les relations ainsi que les cardinalités adéquates.
Une fois les fichiers générés et les configurations terminées, on peut modifier la syntaxe de notre language en modifiant le fichier .xtext.
On ajoute ensuite les concepts nécessaires à notre model afin de mettre en oeuvre les comportements voulus.
Nous avons eu du mal à suivre le tutoriel et nous n'avons pas terminé l'implémentation avec XText.

Pros

- Création des méta éléments graphiquement. Cette couche d'abstraction supplémentaire facilite la création du model.
- La vue permettant de modifier les propriétés d'un élément comme par exemple la cardinalité ou autre est pratique et simple d'utilisation, il suffit de modifier les valeurs voulus dans les champs texte.



Cons

- Setup difficile à mettre en place.
- Sensation de perte de controle de par la génération des fichiers.

III. MPS

Description de l'implementation

Le langage se basant sur les AST pour définir le langage, le coeur de l'implémentation s'est faite autour de la définition des différents concept du langage qui formes les noeuds de l'arbre. Pour notre DSL, il va y avoir un concept racine qui va accepter les deux éléments qui vont former notre machine à état : les bricks et les states. Chacun ayant des sous-concept pour les définir plus en détails, comme des bricks spécifique, les actions et transitions des states . Avec la définition des concepts il y a le strict minimum pour écrire du code correspondant au DSL, mais il ne produira pas de code et sera un peu moche à écrire.
Pour produire du code, des template sont associé aux différents concepts afin de les traduire en code utilisable. Ici, les templates ont pour but de générer le code arduino correspondant aux différentes caractéristiques des noeuds de notre AST. Ainsi, le code static peut être directement écrit dans des prints et les parties qui dépendent des attributs des concepts sont généré grâce au langage.
MPS permet de créer également la syntaxe pour écrire le code du DSL. Ce qui permet de simplifier la lecture et l'écriture du code vers un modèle simple et épuré. La syntaxe de base correspondant a l'écriture de l'arbre, il peut être préférable de la changer vers quelque chose de plus concis et faisant ressortir les notions essentielle.

Pros

- S'abstraire de l'écriture du code pour se concentrer sur la definition des concept du langage ce qui facilite l'abstraction.
- Ecrire un DSL lisible avec une syntaxe personalisé
- Un editeur bien fait avec une auto-complétion "magique"

Cons

- Impossible de coder sans auto-complétion (ce qui empêche le copié/collé par exemple). Peu de chose s'écrivent sans faire un Ctrl + Espace.
- Un concept de programmation complètement nouveau pas facile à prendre en main

- Les templates sont un des rare aspect du langage à s'écrire sans auto-complétion et là c'est handicapant.

Conclusion

Suite à cette étude, nous avons pu nous rendre compte des différences entres les différentes manières de créer un DSL. Les DSL embarqué offrent un confort non négligeable du fait que l'on reste sur du code connu, mais la maintenabilité du code peut se montrer plus compliquée que sur les DSL externe. A choisir, nous nous orienterions soit sur de l'embarqué pour rester sur un langage maitrisé, quitte à faire face à la robustesse de l'implémentation. Soit sur MPS car c'est un DSL pour les DSL qui permet de s'abstraire de la partie code pour se concentrer sur l'essentiel du travail, à savoir : l'abstraction du langage.