Rapport de Projet d'IDM : Modélisation, Vérification et Génération de Jeux

Nom des auteurs: Imane Mriziga et Chaimaa Lotfi
Anas Titah et Amine Akby

Département Sciences du Numérique - Deuxième année
2020-2021

Contents

Introduction et Objectif

Le but de ce projet est de définir une chaîne de modélisation, vérification et génération de code pour des modèles de jeux. L'objectif essentiel est de modéliser un jeu dans un langage propre, afin de s'assure qu'il est faisable et que ce jeu peut se terminer tout en respectant un ensemble d'exigences.

1 Définition du syntaxe Textuelle

Nous avons fait le modèle Xtext décrivant la syntaxe concrète textuelle des modèles de jeux, nous avons essayé au maximum de respecter toutes les exigences demandées. On a remarqué qu'il y a quelques exigences qu'on va les modéliser sous forme des contraintes OCL.
On a effectué quelques modifications au fur et à mesure de l'implantation de la transformation ATL, ainsi que l'implantation de jeu en langage JAVA. Par exemple, pour ATL, on a ajouté un élément GameElement qui facilite l'accès au jeu, Pour Implantation du jeu, on a ajouté deux relations de référence unityItem et unityKnowledge pour en savoir le nombre d'occurrence de chaque objet qui possède l'explorateur.
Nous avons exprimé dans la syntaxe concrète textuelle définie avec Xtext l'exemple de jeu Enigme. Aussi, pour le même exemple on a associé à chaque élément de game un élément dans le modèle petriNet.

- **Jeu Enigme** : correspond à un réseau de Petri de même nom.
- **Explorateur**: le joueur a été modélisé par une place de trois tentatives (nombre maximum de nombre d'objet).
- **Objet Tentative**: correspond à une place liée aux transitions correspondant aux personnes qui sont dans ce lieu.
- **La connaissance Sphinx**: correspond à une transition liée aux places correspondant aux interactions de cette personne.
- **Points d'arriver succès ou échec**: Ces deux points sont modélisés par deux places.
- **Une bonne réponse et la mauvaise réponse**: Les réponses sont modélisées par des places et chaque réponse donne un chemin différent.
- **Interaction**: correspond à une place liée aux transitions de choix.
- **Choix**: correspond à une transition liée aux objets consommés ou acquis.
- **Chemin**: correspond à une transition de même nom liée à 2 places correspondant à sa visibilité et son ouverture. On lie la transition aux places correspondant aux lieux origines et cibles du chemin.

Nous avons visualisé à l'aide de l'outil nd, le réseau de petri résultant de jeu enigme.

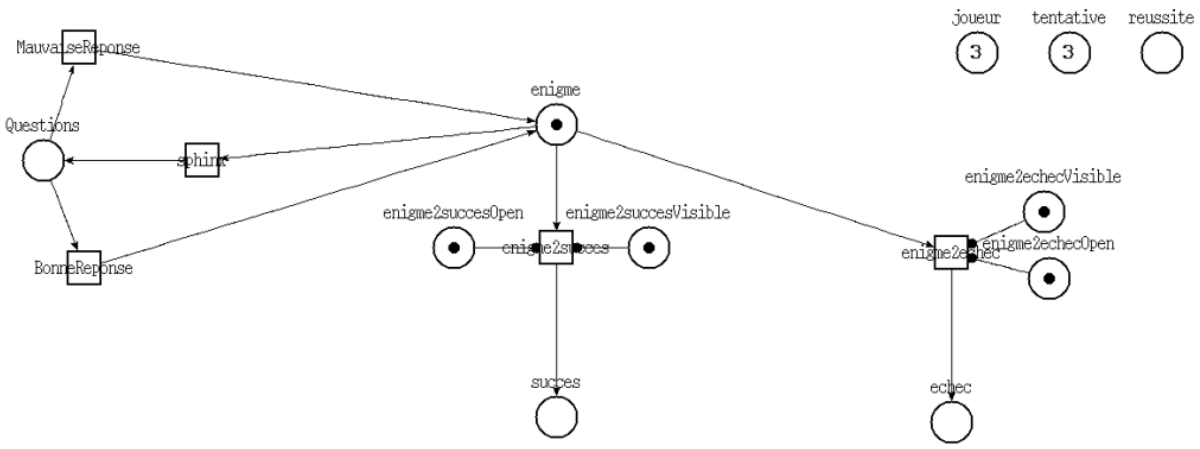

Figure 1: le réseau de petri résultant de jeu enigme.

Une vue graphique mise en page du métamodèle généré par Xtext

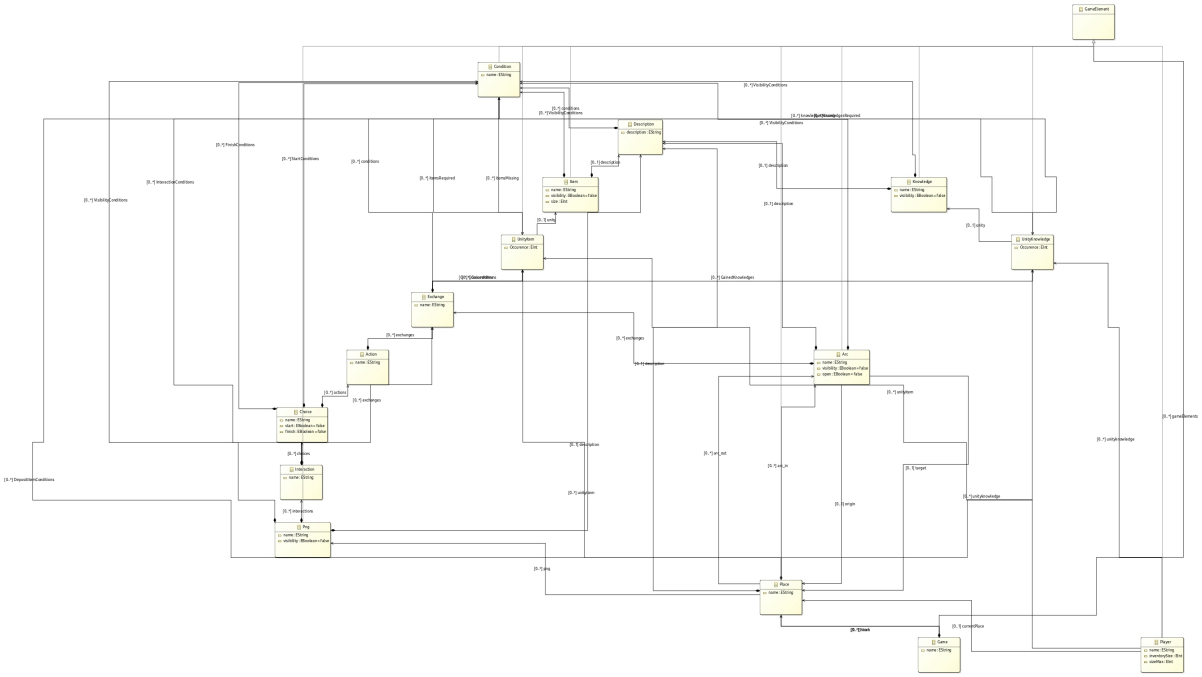

Figure 2: Une vue graphique mise en page du métamodèle généré par Xtext

Dans cette image, les classes ne sont pas vraiment visible, vous trouverez une image plus claire sur les livrables.

2 Contraintes OCL

Nous avons modélisé plusieurs contraintes en langage OCL,

- Le nom du jeu, de places, d'explorateur, des objets, des personnages, ne doit être ni vide ni null.
- L'exigence 6 : Le nombre d'objets que peut posséder un explorateur est limité par la taille cumulée des objets possédés.
- L'exigence 8 : On doit avoir une seule place de départ, donc size doit valoir 1. Pas de contraintes sur le nombre de places d'arriver donc il suffit que le size soit supérieur ou vaut à 1
- L'exigence 25 : L'explorateur doit choisir une action ou quitter l'interaction s'il s'agit d'un choix de fin.

On a ajouté d'autres contraintes dont on peut citer :

- Deux choix ne doivent pas avoir la même action
- Deux conditions, connaissances, ne doivent pas avoir le même nom
- Les descriptions ne doivent être ni vide ni null.
- Le nombre des connaissances ou des objets est non nulle.
- La taille des objets est non nulle.
- Une place start n'a pas d'arc entrant

3 Transformation M2M avec ATL

Pour la transformation du jeu en réseau de Petri, nous avons défini les principales correspondances suivantes:

- **Jeu** : correspond à un réseau de Petri de même nom.
- **Joueur**: correspond à une place en réseau de Petri avec un nombre de jetons égal au nombre maximum d'objets qu'il peut avoir.
- **Place**: correspond à une place liée aux transitions correspondant aux personnes qui sont dans ce lieu.
- **Personne**: correspond à une transition liée aux places correspondant aux interactions de cette personne.
- **Interaction**: correspond à une place liée aux transitions de choix.
- **Choix**: correspond à une transition liée aux objets consommés ou acquis.
- **Chemin**: correspond à une transition de même nom liée à 2 places correspondant à sa visibilité et son ouverture. On lie la transition aux places correspondant aux lieux origines et cibles du chemin.

4 Prototype

Nous avons décider d'implanter ce jeu en Java.

Nous avons décidés de ne pas utiliser le code Java généré par EMF pour programmer le jeu.

Pour chaque élément du jeu, on a crée une classe java correspondante.

Nous avons crées deux classes java pour manipuler les différentes classes et pouvoir gérer les actions à faire.

- Afficheur.java : Cette classe gère l'affichage des différents actions demandés par l'utilisateur. Elle affiche tous les connaissances, les objets, et les personnes disponibles et les chemins ouverts.
- Contrôleur.java : Cette classe gère toutes les fonctionnalités du modèle. Elle gère les dépôts et les prises des objets, effectue les changements entre les places, traite les interactions entre le joueur et les personnes.

```
Vous êtes dans Enigme

Que voulez vous faire ?
(1) Lister les connaissances et les objets possédées
(2) Lister ce qui est visible dans le lieu (personnes + objets)
(3) Déposer un objet
(4) Prendre un objet
(5) Interagir avec une personne
(6) Choisir un chemin
(7) Choisir un objet à transformer

1
********** Connaissances Maitrisées **********
Il n'y a aucune Knowledges
**********************************************************

****************** Items Possedes ******************
Id  Quantite Nom                                  Poids
 1)        3 Tentative                           1
**********************************************************
Poids Total : 3/3
**********************************************************


Vous êtes dans Enigme

Que voulez vous faire ?
(1) Lister les connaissances et les objets possédées
(2) Lister ce qui est visible dans le lieu (personnes + objets)
(3) Déposer un objet
(4) Prendre un objet
(5) Interagir avec une personne
(6) Choisir un chemin
(7) Choisir un objet à transformer
```

5 Transformation M2T avec Acceleo

Afin de vérifier les contraintes du jeu via l'outil Boîte Tina, il était nécessaire de pouvoir réaliser un réseau de pétri qui modélise ce jeu.

5.1 A partir d'un modèle jeu :

Définition d'une transformation de modèle à texte (M2T) avec Acceleo, pour engendrer les propriétés LTL à partir d'un modèle de jeu. Cette transformation est faite pour vérifier quelques contraintes en utilisant l'outil Tina. Plus précisément, nous voulons vérifier la terminaison d'un tel jeu est-il possible.

5.1.1 Propriétés de terminaison de jeu

Un jeu se termine si il y a plus d'interactions ou s'il s'agit d'un choix de fin. La propriété correspondante peut s'écrire ainsi:

op finished = on recupere toutes les places de fin ;

(finished ⇒ dead);⇒ quand un jeu est terminé, il n'y a plus d'action à exécuter.
(dead ⇒ finished); ⇒ si il n'y a plus d'action à exécuter alors le jeu est terminé.
- <> finished; ⇒ un jeu n'est jamais terminé.

Nous avons pour la dernière propriété qu'un jeu n'est jamais terminé plutôt qu'un jeu se termine pour le jeu puisse nous exhiber des scénarios de terminaison du jeux comme contre-exemple, et ainsi comprendre ce qui se passe exactement.
Le fichier *game2ltl.mtl* contient le code pour engendrer les propriétés ci-dessus pour chaque modèle de jeu.
Nous engendrons ainsi les propriétés pour le modèle de jeu Enigme et à l'aide de l'outil *selt* de la boîte à outils Tina que nous appliquons sur la transformation du modèle en *.net*, nous obtenons les résultats suivants de la figure 3

```
- source game.ltl;
operator finished : prop
TRUE
TRUE
FALSE
  state 0: enigme enigme2echecOpen enigme2echecVisible enigme2succesOpen enigme2succesVisible joueur*3 tentative*3
  -enigme2echec->
  state 1: L.dead echec enigme2echecOpen enigme2echecVisible enigme2succesOpen enigme2succesVisible joueur*3 tentative*3
  -L.deadlock->
  state 2: L.dead echec enigme2echecOpen enigme2echecVisible enigme2succesOpen enigme2succesVisible joueur*3 tentative*3
  [accepting all]
0.001s
```

Figure 3: Résultat des propriétés pour le jeu Enigme

5.2 A partir d'un prototype :

Définition d'une transformation de m odèle à texte (M2T) avec Acceleo, pour générer un prototype d'implantation du jeu à partir d'un modèle de ce jeu. Cette transformation crée une classe java avec une méthode main qui initialise tout l'environnement du jeu à partir du modèle donné en paramètre, et lance le jeu à partir de la classe Java Contrôleur.
On commence par initialiser le jeu, les places, les arcs entre ces places et le joueur. Ensuite, on crée les objets et les connaissances du modèle et on initialise les échanges, les choix et les actions associés à chaque interaction des personnes du modèle. Enfin, on ajoute tous les éléments crées au jeu et on lance le jeu.

6 Conclusion

En conclusion, nous avons pu faire fonctionné tout et vérifier la terminaison d'un jeu, Notre ATL maintenant fonctionne bien. Ce projet nous a permet de partir à zero et contruire notre métamodèle qui nous facilitera les transformation et l'élaboration de notre prototype.