Rapport Projet d'Ingénierie Dirigée par les Modèles

Modélisation, Vérification et Génération de Jeux

Ayoub LOUDI
Kamal HAMMI
Mohamed IKICH
Mohamed Hamza LOTFI

Groupe B-01

22 January 2022

Table des matières

1 Introduction

Ce document représente un rapport concis expliquant le travail réalisé lors du projet de la matière Ingénierie Dirigée par les Modèles.

L'objectif du projet est de produire une chaîne de modélisation, de vérification et de génération de code pour des jeux de parcours/découverte, en s'appuyant sur :

1. Conception d'un langage dédié pour décrire le jeu sous la forme d'un modèle.
2. Garantie d'existence d'une solution pour le jeu décrit par un modèle.
3. Construction d'un prototype avec une interface texte simple permettant de tester le jeu décrit par un modèle, et de valider la jouabilité et l'intérêt du jeu avant de développer le contenu multimédia.

2 Plan

1. Définition de la syntaxe concrète du langage de modélisation de jeu en utilisant Xtext et génération du métamodèle Ecore associé.
2. Définition de la sémantique statique avec OCL.
3. Définition d'une transformation de modèle à texte (M2T) avec Acceleo, pour engendrer les propriétés LTL à partir d'un modèle de jeu. En réutilisant la transformation des réseaux de Petri vers la syntaxe concrète de TINA réalisée pendant les TPs.
4. Définition d'une transformation de modèles à modèles (M2M) avec ATL pour produire un réseau de Petri à partir d'un modèle Game2PetriNet.
5. Définition d'une transformation de modèle à texte (M2T) avec Acceleo, pour générer un prototype d'implantation du jeu à partir d'un modèle de ce jeu, et ceci dans le language du Java.
6. Définition des syntaxes concrètes graphiques avec Sirius pour un modèle de Game

3 La syntaxe concrète Xtext

Le fichier game.xtext contient la grammaire décrivant les différents elements d'un Game, en respectant la plupart des exigences données.

Le metamodèle généré est donné dans la figure 1.

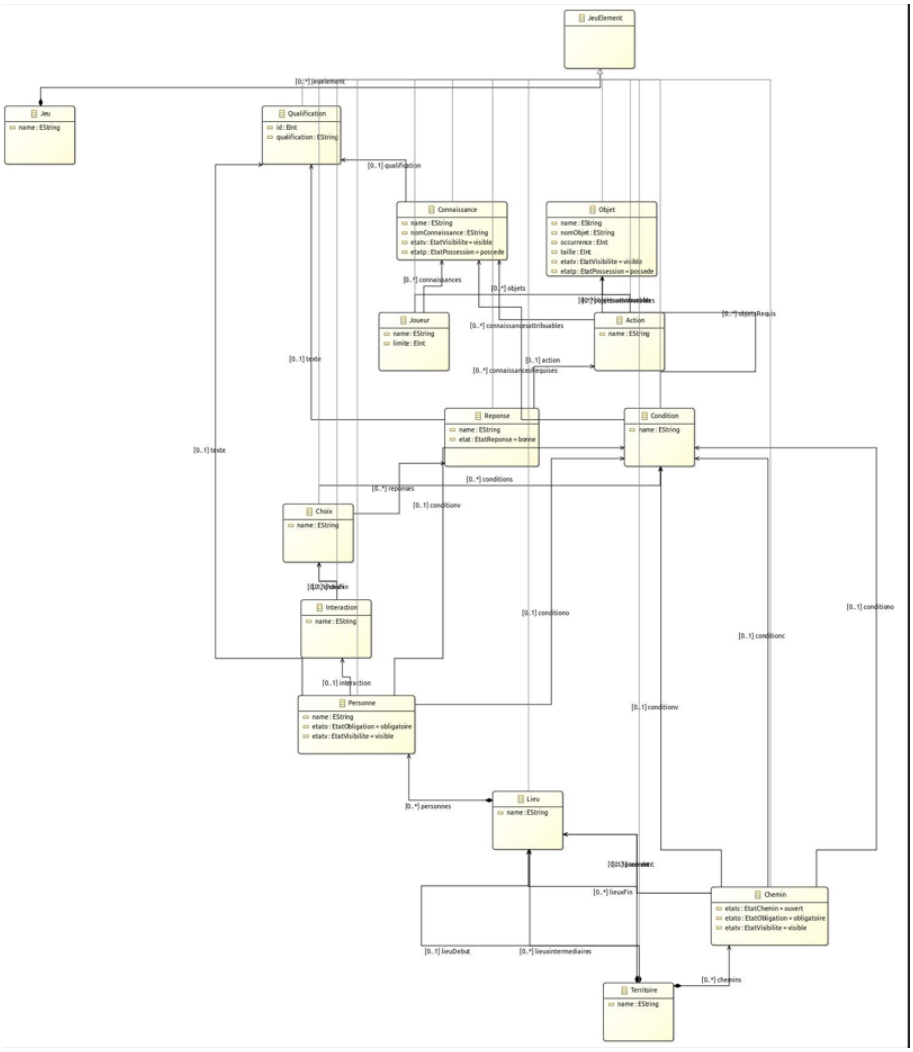

FIGURE 1 – Diagramme de la classe Game

4 Sémantique statique avec OCL

Les contraintes abordées par OCL on été celles concernant la capacité maximale que peut porter le joueur, les poids des objets (positifs), l'unicité si existence d'une personne obligatoire par lieu, et la vérification que tout les objets/ connaissances que possède le joueur comme étant tous visibles.

Les contraintes d'unicité du joueur, du lieu de départ, etc ..., sont gérés directement dans le métamodèle Xtext, en définissant les bonnes arités pour chaque composantes de la grammaire.

5 Transformation M2T avec Acceleo, pour engendrer les propriétés LTL du modèle

Le fichier Game2ltl.mtl contient la transformation vers la Syntaxe LTL à partir d'un modèle de Game en vérifiant l'existence d'une solution, et que le joueur arrive quoi qu'il soit en un Lieu de Fin.

6 Transformation M2M avec ATL pour produire un réseau de Petri à partir d'un modèle Game.

Le fichier Game2PetriNet.atl contient la transformation du modèle de Game vers un réseau de Petri avec ATL. Le fichier enigme.net contient le résultat de transformation du modèle de l'exemple Enigme contenu dans le fichier EnigmeOK.xmi.

7 Transformation M2T avec Acceleo, pour générer un prototype d'implantation du modèle Game dans le language Java

Le fichier **ToJava.mtl** contient le code de la transformation ModelToText du modèle de Game vers un prototype dans le langage Java.
Le principe utilisé est le suivant :
- Tout le programme sera écrit dans un seul fichier qui contiendra la méthode **main** représentant le moteur du jeu.
- Le fichier contiendra également des sous classes qui font référence aux éléments du modèle : Pour chaque élément du modèle, on associe une classe dont les attributs sont exactement les attributs qui lui sont associés au niveau du métamodèle. Ce sont les constructeurs de ces classes qui nous permettrons ensuite d'instancier les différents éléments du modèle à transformer.
- On commence d'abord par initialiser le territoire et tous ces éléments et puis le moteur commence son interaction avec l'utilisateur.

Il est à noter qu'on a rencontré plusieurs problèmes lors de cette façon à savoir le nommage des différents éléments du modèle : Il faut normalement avoir un nom unique pour chaque élément, or les noms sont eux-même des éléments générés automatiquement, il faut donc trouver des combinaisons de nom qui soient irrépétables. De plus puisque dans notre implantation les noms des instances des sous-classes sont tirés des noms des éléments du modèle, on exige qu'il n'y ait pas d'espace dans les noms des éléments du modèle.

8 Définition des syntaxes concrètes graphiques avec Sirius pour un modèle de Game

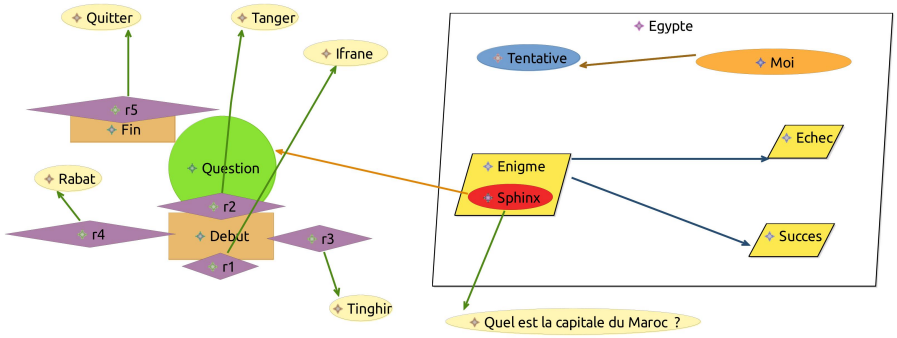

FIGURE 2 – Diagramme Crée par Sirius pour l'exemple Enigme