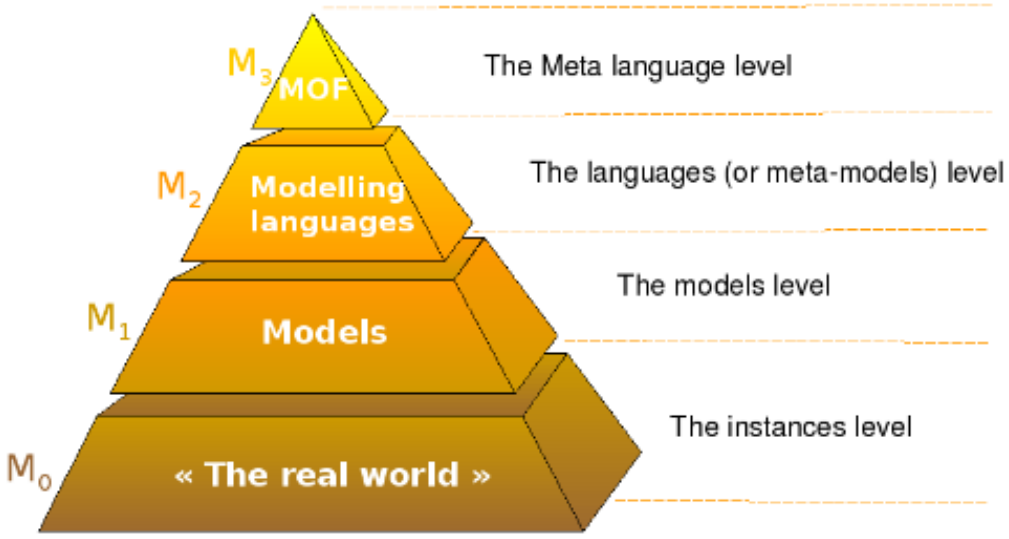
Modélisation, Vérification et Génération De Jeux
Projet d'Ingénierie Dirigée par les Modèles

Chaimaa Louahabi - Ahmed Ghanim
Manal Hajji - Thibaud Merieux

January 21, 2021



Département Sciences du Numérique - Deuxième année 2020-2021

Contents

List of Figures

1 Modélisation de jeux d'exploration

1.1 Définition de la syntaxe concrète textuelle

Pour pouvoir décrire les jeux de parcours/découverte et manipuler ses modèles, un language dédié (Domain Specific Modeling Language)a été défini en utilisant Xtext. Ce language répond aux exigences tirées de l'analyse des besoins des clients (cf $E1 -> E31$ sur le sujet).
Le fichier *game.xtext* définit une grammaire pour créer des modèles de SimpdlePDL avec les règles de construction simples suivantes :

- ***game* { ... }** : dévlarer le jeu.
- **player** *nom* **{ capacity** = k, **current place** = *place***}**: ajouter je joueur unique et déclarer ses attribuuts.
- **place** nom **{ type** = debut/fin/intermidiaire
  has { objets /connaissances / personnes**}**
  allocations
  interactions
  chemins**}**
- **path from** source **to** destination: définir un chemin.
- **act** nom **: qualified with "**..." : une action a un nom et une qualification.

La figure 1 montre le modèle de jeu Enigme écrit dans la syntaxe concrète textuelle définie.

enigme.game

```
1  game {
2      player anonyme {
3          capacity= 2; current place = Enigme
4      }
5      place Enigme {
6          type = begin
7          has
8              (knowledge Reussite state V);
9              (obj Tentative mesure 3 state V);
10             (person Sphinx state V if {~Reussite and Tentative >= 1}
11                 qualified with "Combien de gouttes d'eau peut-on mettre dans un verre vide")
12         allocate 2 of Tentative to anonyme
13
14         interact with Sphinx {
15             choice answer [act reponse1 : qualified with "une";
16                            act reponse2 : qualified with "deux";
17                            act reponse3: qualified with "trois" ]
18
19             if reponse1 then gain Reussite
20             if reponse2 then consume 1 of Tentative
21             if reponse3 then consume 1 of Tentative
22         }
23
24         path from Enigme to Succes open if {Reussite}
25         path from Enigme to Echec open if {Tentative = 0}
26     }
27     place Succes {
28         type = end
29     }
30     place Echec {
31         type = end
32     }
33 }
```

Figure 1: Le modèle de jeux 'enigme' exprimé dans la syntaxe concrète textuelle définie

.

1.2 Génération du métamodèle

Xtext permet également de générer le métamodèle à partir de la définition du language(voir figure 2). Explication de Quelques élèments de ce métamodèle :

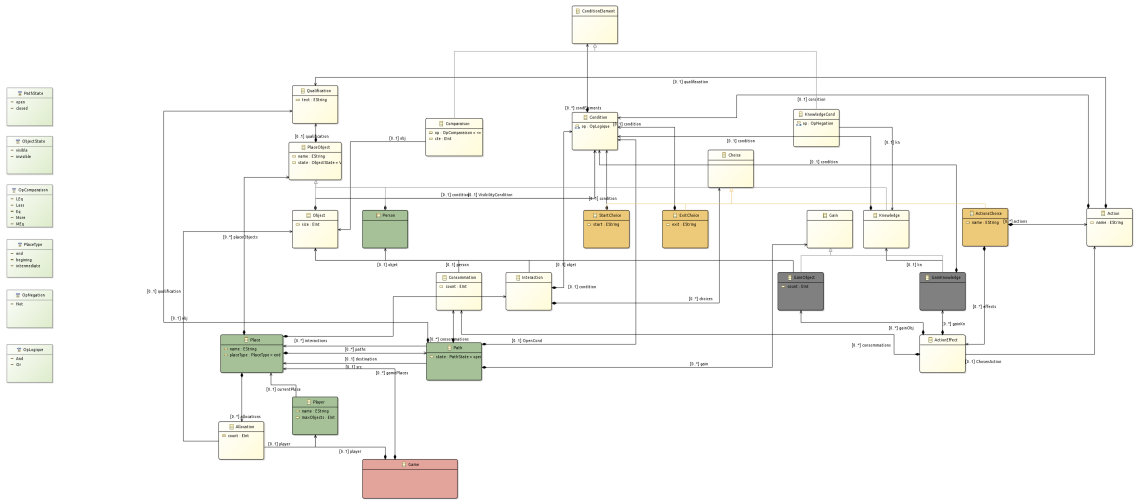


Figure 2: Une vue graphique mise en page du métamodèle généré par Xtext

.

- **ActionsChoice** : sous type se Choice, reprèsente le choix à actions multiples (qui sont les réponses possibles dans l'exemple d'enigme)
- pour décrire proprement une condition sur les objets possédés par le joueur ou sur ses connaissances, nous avons eu recours à deux type de conditions: **KnowledgeCond** (la possession ou non d'une connaissance), et **Comparaison** (égalité ou inégaté comparantle nombre d'un objet à une constante)
- Les effets du choix d'une action sont traduite par **ActionEffect** qui est soit attribution d'une connaisssance/objet (**GainKnowledge** /**GainObject**), ou consommation d'un objet possédé par le joueur (**Consommation**)

2 Définition de la sémantique statique avec OCL

OCL est un moyen pour préciser la sémantique du métamodèle en limitant les modèles conformes. Nous l'avons utilisé pour imposer quelques invariants sur les modèles de Game, comme :

- L'invariant *SizeLimit_objects* : Une allocation ne peut pas dépasser le nombre maximal d'objet permis pour le joueur.
- L'invariant *Interaction_Need_NonNull_Person* : Une intéraction ne peut avoir lieu sans une personne à intéragir avec.
- L'invariant *SourceAndDestinationOfPathAreDifferent*: garantit que les places source et destination d'un chemin soient différentes.
- Les invariants *NonNull_Object_Consommation* (resp Gain): une consammation (resp attribution) n'a aucun sens si le nombre d'objets à consommer (resp à attribuer) est nul.
- L'invariant Effect *_Must_Include_Gain_Or_Consommation* : impose que l'on ajoute obligatoirement au moins une consommation ou une attribution à un élèment actionEffect, qui represente l'effet lié au choix d'une action.

3 La transformation M2M de Game en Réseau de Petri

Pour utiliser les outils de simulation et de vérification proposés par la boîte à outils TINA sur un modèle de jeu d'exploration, il faut le traduire en réseaux de Petri. Cette tranformation peut-être réalisée en utilisant ATL.
Une difficulté particulière rencontrée : Pour pouvoir traduire les conditions sur les objets d'un lieu de type $objet \leq k$ en des élèments de réseau de Petri, il fallait introduire un nouveay type d'arc " inhhibitor". Un arc inhibiteur de poids k n'est validé que si la place de départ a au plus (k-1) jetons.

Donc, nous avons dû modifier le métamodèle de petrinet réalisé au mini-projet ainsi que sa transformation vers la syntaxe de TINA, pour introduire ce nouveau type d'arcs.
La figure 3 montre une vue graphique du réseau de petri résultant de la transformation M2M du modèle du jeu enigme.

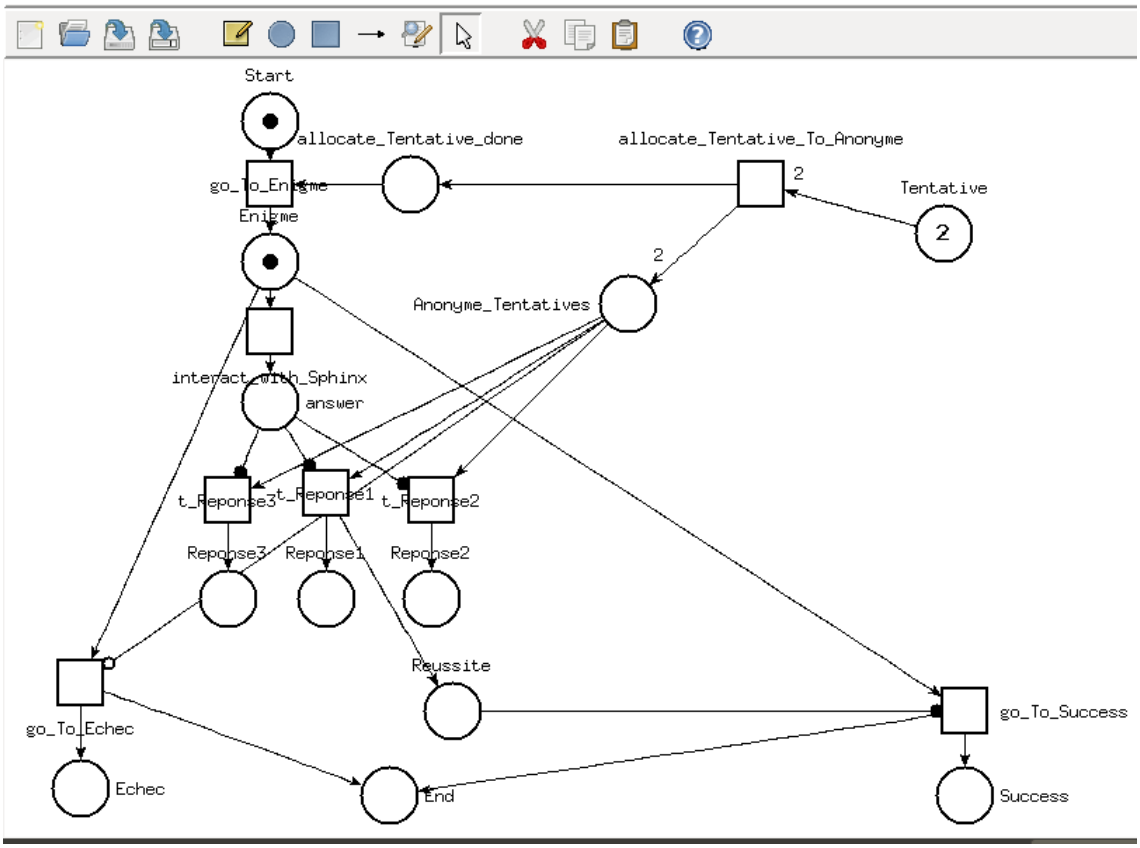


Figure 3: Le réseau de petri équivalent au modèle de jeux 'enigme'

4 Engendrer les propriétés LTL à partir d'un modèle de jeu

La question de savoir si un jeu est réalisable peut se ramener à une étude de la terminaison de l'exécution du jeu. On considère qu'un jeu est terminé s'il est dans son état final, que le joueur soit dans l'un des lieux finaux.
Le code de "game2ltl.mtl" est une transformation modèle à texte qui prend en entrée un modèle de Game et traduit les propriétés suivantes qui seront vérifiées par le model checker selt.
$[] <> dead$: toute exécution se termine.
$- <> finished$:Si le jeu peut finir, cette propriété sera fausse et le model checker exhibera un contre-exemple qui correspond à un scénario où le jeu se termine.

5 Conclusion

Ce projet nous a permis d'intervenir sur l'ensemble de la chaîne de vérification des modèles de jeux en passant par les modèles PetriNet. A savoir, la génération du métamodèles avec Xtext, la complétion de leurs propriétés structurelles par des contraintes (invariants, pré-, post-conditions) spécifiées à l'aide du language d'expression de propriétés OCl, la tasformation M2M avec ATL et finalement la tansformation M2T pour pouvoir profiter de la boîte à outils TINA qui permet de savoir si le processus décrit est réalisable ou pas avant de démarrer le développementde la partie multimédia du jeu (images, sons, musiques, vidéos, etc) qui est la plus coûteuse.