

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



Faical Toubali Hadaoui
Ignacio Lucas Oros campo
Hamza Mouddene
Anass Tyoubi
Laurène Dardinier

5 Janvier 2021

Table des matières

1	Introduction	2
1.1	Présentation	2
1.2	Description des jeux	2
1.3	Exemple : un jeu d'énigme	2
2	Définition de la syntaxe concrète avec Xtext	3
2.1	Description de la syntaxe textuelle	3
2.2	Modélisation du jeu Enigme avec la syntaxe textuelle	3
2.3	Description du méta-modèle généré	5
3	Version Petri net et LTL de Enigme	9
4	Programme en Java de l'exemple énigme	11
5	Définition de la sémantique statique avec OCL	13
5.1	Exemples de contraintes OCL définies	13
5.2	Exemples de contraintes OCL non respectées pour un modèle de jeu	13
6	Définition d'une transformation M2T avec Acceleo - Propriétés LTL à partir d'un modèle	15
6.1	Exemples de transformation M2T d'un modèle de jeu en propriétées LTL	16
6.1.1	Modèle de jeu Dora :	16
6.1.2	Modèle de jeu FormationGuerrier	16
6.2	Vérification de la terminaison du jeu Enigme	17
7	Définition d'une transformation M2M avec ATL - Réseau de Pétri à partir d'un modèle	18
7.1	Les règles ATL	18
7.2	Limites	21
8	Conclusion	21

Table des figures

1	Syntaxe concrète avec Xtext	3
2	Exemple du jeu Enigme	4
3	Méta-modèle généré - 1ère partie	5
4	Méta-modèle généré - 2ème partie	5
5	Classes représentant une personne sur un lieu et les possibles choix et actions qui découlent d'une interaction avec elle	6
6	Classes représentant des objets et connaissances sur le jeu. Elles peuvent appartenir à un joueur ou à un lieu	6
7	Classes principales du jeu où on définit le nom du jeu et les caractéristiques du joueur	7
8	Classes des types énumérés pour définir le type des lieux et l'ouverture et visibilité des éléments ainsi que le type de comparaison	7
9	Classe représentant une condition définie soit sur un objet soit sur une connaissance	7
10	Classes représentant un chemin et le don d'objets et connaissances dans un chemin ou action	8
11	Petri net représentant le jeu Enigme avec Tina	9
12	enigme.net	10
13	enigme.ltl	10
14	Lancement de Enigme.java	11
15	Interagir avec le Sphinx	11
16	Terminaison avec un succès	12
17	Terminaison avec un échec	12
18	Le nom d'un élément du jeu ne doit être ni vide ni null	13
19	Le joueur ne peut pas porter une quantité d'objets supérieure à la taille de son inventaire	13
20	Un jeu ne peut pas disposer de plus d'un lieu de type début	13
21	Allocation de deux connaissances et un objet au guerrier dès le passage par le chemin Barriere2Soins	14
22	Définition du Joueur guerrier	14
23	La contrainte OCL Joueur : :connaissancejoueurExistant n'est pas respectée	14
24	Acceleo : Transformation M2T - LTL à partir d'un modèle de jeu	15
25	Transformation M2T - LTL à partir du modèle de jeu Dora	16
26	Transformation M2T - LTL à partir du modèle de jeu FormationGuerrier	16
27	Propriétés LTL pour le jeu Enigme	17
28	Outil Tina pour produire enigme.ktz	17
29	Vérification des propriétés LTL	17
30	Représentation Petri net d'un Lieu	18
31	Représentation Petri net d'un objet possédé par le joueur	18
32	Représentation Petri net d'une connaissance associée au joueur	19
33	Représentation Petri net d'un objet qui se trouve sur un lieu et les actions "prendre" et "déposer" associées	19
34	Représentation Petri net d'un chemin avec condition d'accès et allocateurs objets ou connaissances non conditionnés	20
35	Représentation Petri net d'une personne selon la condition de visibilité	20

1 Introduction

1.1 Présentation

Ce projet concerne les jeux de parcours/découverte et se divise en trois parties.

- Modélisation : concevoir un langage dédié pour décrire le jeu sous forme d'un modèle, implanter les outils d'édition, vérification et génération associés.
- Vérification : assurer qu'il existe une solution pour un jeu décrit par un modèle (outils de model-checking via TINA, il faut donc traduire un modèle de jeu en réseau de Pétri).
- Génération de code : construire un prototype avec une interface texte simple (pour tester le jeu décrit par un modèle, valider sa jouabilité et son intérêt avant d'aller plus loin).

1.2 Description des jeux

L'objectif d'un jeu d'exploration est pour le joueur nommé explorateur de visiter un territoire composé de lieux connectés par des chemins. Il possède des connaissances et des objets de certaines tailles en quantité limitée.

Les lieux peuvent quant à eux contenir des connaissances, des objets et des personnes, que l'explorateur peut recevoir/prendre s'ils sont visibles, et où il peut déposer les objets de son choix. L'explorateur peut alors emprunter un chemin (visible et ouvert), qui peut lui transmettre des connaissances et des objets, ou au contraire consommer certains de ses objets.

L'explorateur peut interagir avec les personnes présentes dans un lieu sous la forme de choix et cela peut lui rapporter des connaissances et des objets. Les actions proposées dépendent entre autre des choix précédents de l'explorateur et peuvent lui rapporter des connaissances et des objets, mais elles peuvent aussi lui consommer des objets.

1.3 Exemple : un jeu d'énigme

On modélise un jeu d'énigme.

Le territoire est composé de trois lieux, un de début nommé Enigme et deux de fin qui représentent le succès, nommé Succès, et l'échec, Echec. Ces trois lieux sont qualifiés par leur nom.

Le nombre de réponses possibles est représenté par un objet Tentative dont l'explorateur possède un nombre initial, par exemple 3.

Le lieu Enigme contient une personne Sphinx qui est visible. Cette personne est qualifiée par le texte de la question. Son interaction contient un choix dont chaque action est qualifiée par les réponses possibles. L'action associée aux mauvaises réponses consomme un objet Tentative. L'action associée aux bonnes réponses donne une connaissance Réussite.

Il existe un chemin allant du lieu Enigme au lieu Succès dont la visibilité est conditionnée par la possession de la connaissance Réussite. Il existe un chemin allant du lieu Enigme au lieu Echec dont la visibilité est conditionnée par la possession d'aucun objet Tentative (0 objet Tentative).

Le Sphinx n'est visible que si l'explorateur possède au moins un objet tentative et pas de connaissance Réussite

2 Définition de la syntaxe concrète avec Xtext

2.1 Description de la syntaxe textuelle

Nous avons défini avec Xtext une syntaxe concrète textuelle pour les modèles de jeu. Dans la construction de notre syntaxe concrète, le jeu se compose d'un joueur unique qui possède un nombre illimité de connaissances et un nombre limité d'objets. L'objectif de ce jeu d'exploration est de visiter un territoire composé de lieux connectés par des chemins.

```
1 grammar fr.n7.JEU with org.eclipse.xtext.common.Terminals
2
3 generate jEU "http://www.n7.fr/JEU"
4
5@Jeu: 'Jeu' name=ID '{'
6     joueur = Joueur
7     ( jeulement+=JeuElement ( "," jeulement+=JeuElement)* )?
8
9 '}';
10
11@JeuElement:
12     Lieu | Chemin | Objet | Connaissance;
13
14
15@Joueur: 'Joueur' name=ID
16     '{'
17     ('connaissance' '{' connaissance+= [Connaissance] ( "," connaissance+= [Connaissance])* '}' )?
18     ('inventaire' tailleinventaire = INT
19     ('objet' '{' objets+= [Objet] ( "," objets+= [Objet] )* '}' )?
20     'lieu' lieu=[Lieu]
21     '}';
22
23@Chemin : 'Chemin' name=ID
24     '{'
25     'de' source = [Lieu] 'a' destination = [Lieu]
26     'ouvertOuferme' ouvertOuferme=ouvertOuferme
27     'visibilite' visibilite = typevisibilite
28     ('accescondition' acces = condition)?
29     ('visiblecondition' visible = condition)?
30     ('description' description=STRING)?
31     ('connaissance' '{' allocateursConnaissance += AllocateurConnaissance ( "," allocateursConnaissance+=AllocateurConnaissance)* '}' )?
32     ('objet' '{' allocateursObjetLieu+=AllocateurObjet ( "," allocateursObjetLieu+=AllocateurObjet)* '}' )?
33
34 '}';
35
36@AllocateurConnaissance :
37     '{'
38     'connaissanceAlloue' = [Connaissance]
39     (estConditionne ?= 'condition' conditionAllocution = condition)?
40     '}';
41
42@AllocateurObjet:
43     '{'
44     'objetAlloue' = [Objet]
45     (consomme ?= 'consomme')?
46     quantite = INT
47     (estConditionne ?= 'condition' conditionAllocution = condition)?
48     '}';
```

FIGURE 1 – Syntaxe concrète avec Xtext

Les lieux explorés peuvent contenir des connaissances, des objets et des personnes. Le point de départ et les points de fin de l'exploration sont des lieux particuliers, les connaissances, les objets et les personnes contenus dans un lieu peuvent être visibles/actifs ou invisibles/inactifs selon des conditions. Les chemins peuvent être ouverts ou fermés selon des conditions, l'explorateur peut interagir avec les personnes présentes dans un lieu.

2.2 Modélisation du jeu Enigme avec la syntaxe textuelle

```

1 Deu EnigmeJeu {
2     Joueur Explorateur {
3         connaissance { Reussite }
4         inventaire 10
5         objet { Tentative }
6         lieu Enigme
7     }
8
9     Objet Tentative { taille 1 quantite 3 visibilite visible activite actif},
10
11    Connaissance Reussite {visibilite visible activite actif},
12
13    Lieu Enigme {
14        type debut
15        personne { nom Sphynx {
16            visibilite visible
17            activite actif
18            interaction Interaction {
19                personne Sphynx
20                question "Question de Sphynx"
21                choix {nom choix1 {actions { action A1 { description "BonneReponse" connaissance {{ Reussite }} finale }, action A2 { description "MauvaiseReponse" objet {{Tentative consomme 1}} finale }}
22                ChoixJoueur nom choixjoueur {}
23            }
24        }
25    },
26
27
28    Lieu Succes {
29        type fin
30    },
31    Lieu Echec {
32        type fin
33    },
34
35    Chemin E2S {
36        de Enigme a Succes
37        ouvertOferme ouvert
38        visibilite visible
39        accescondition nom AvoirSucces {connaissance Reussite}
40    },
41
42    Chemin E2E {
43        de Enigme a Echec
44        ouvertOferme ouvert
45        visibilite visible
46        accescondition nom AvoirEchec {objet Tentative comparateur = reference 0 }
47    }
48 }

```

FIGURE 2 – Exemple du jeu Enigme

2.3 Description du métamodèle généré

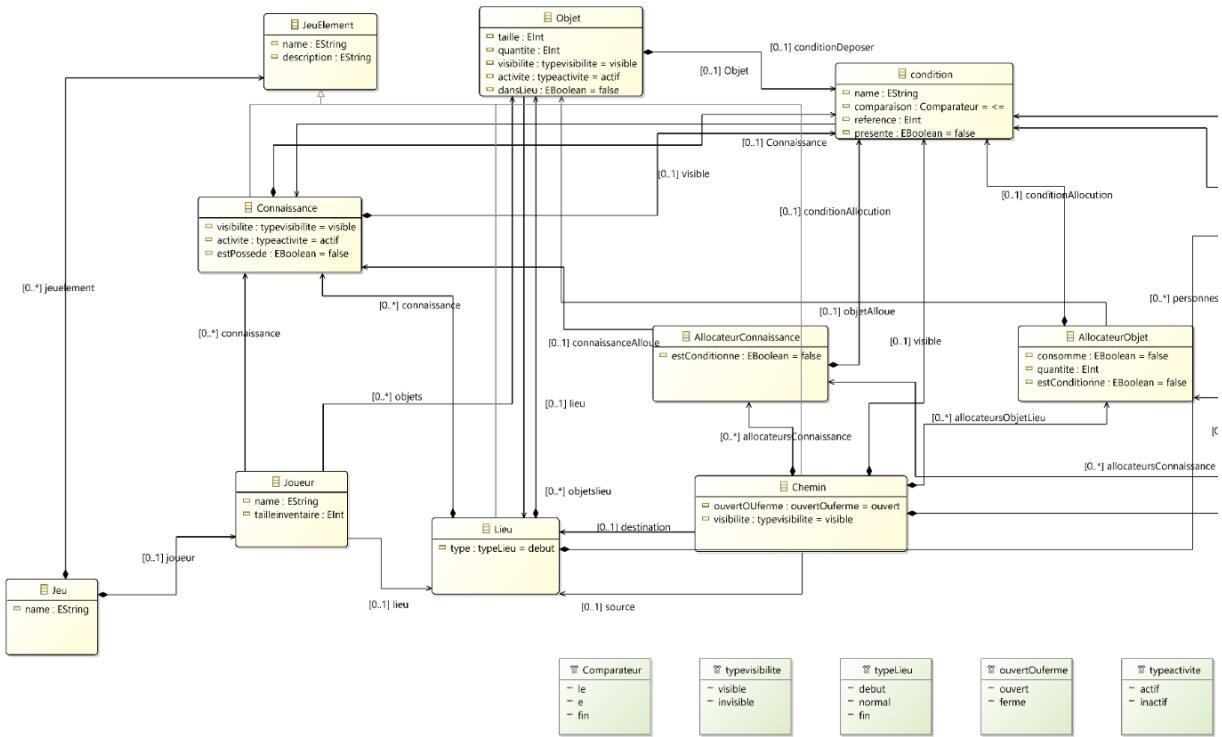


FIGURE 3 – Méta-modèle généré - 1ère partie

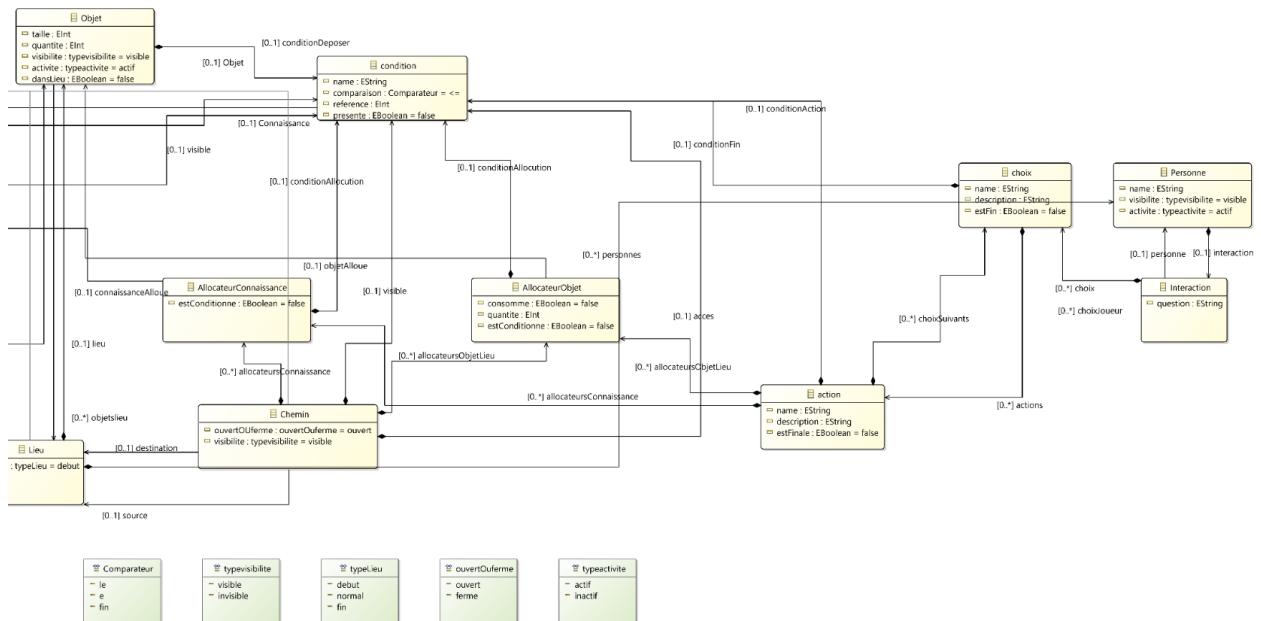


FIGURE 4 – Méta-modèle généré - 2ème partie

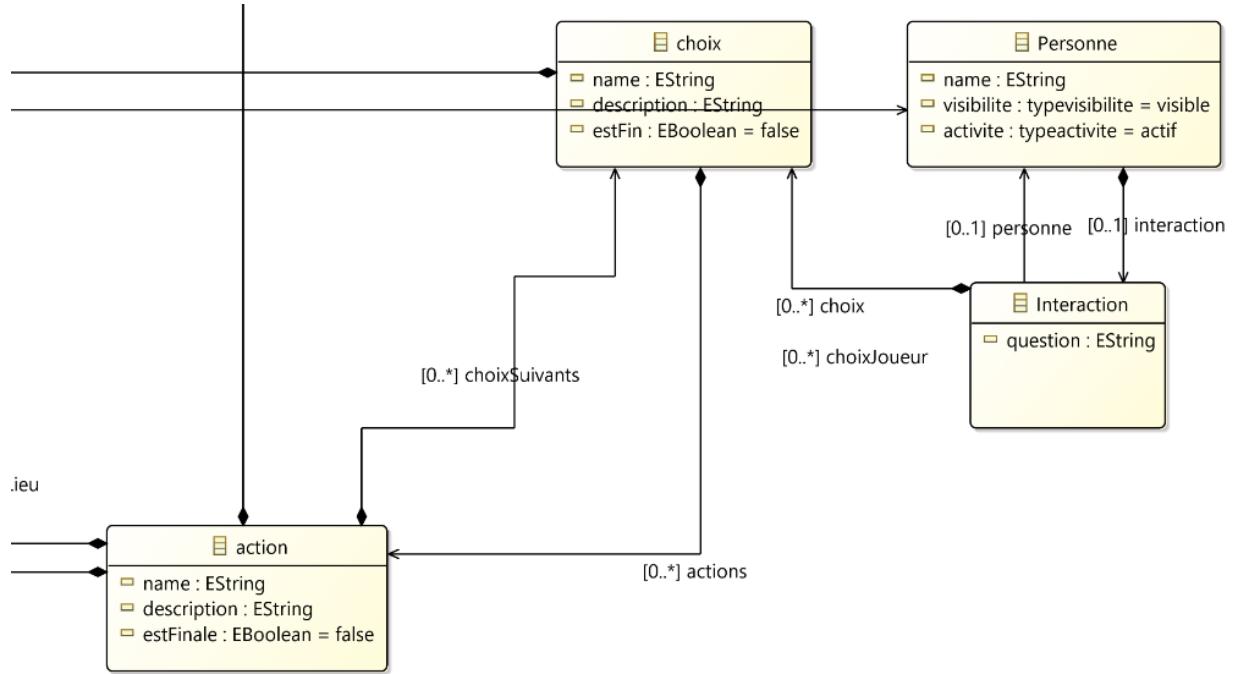


FIGURE 5 – Classes représentant une personne sur un lieu et les possibles choix et actions qui découlent d'une interaction avec elle

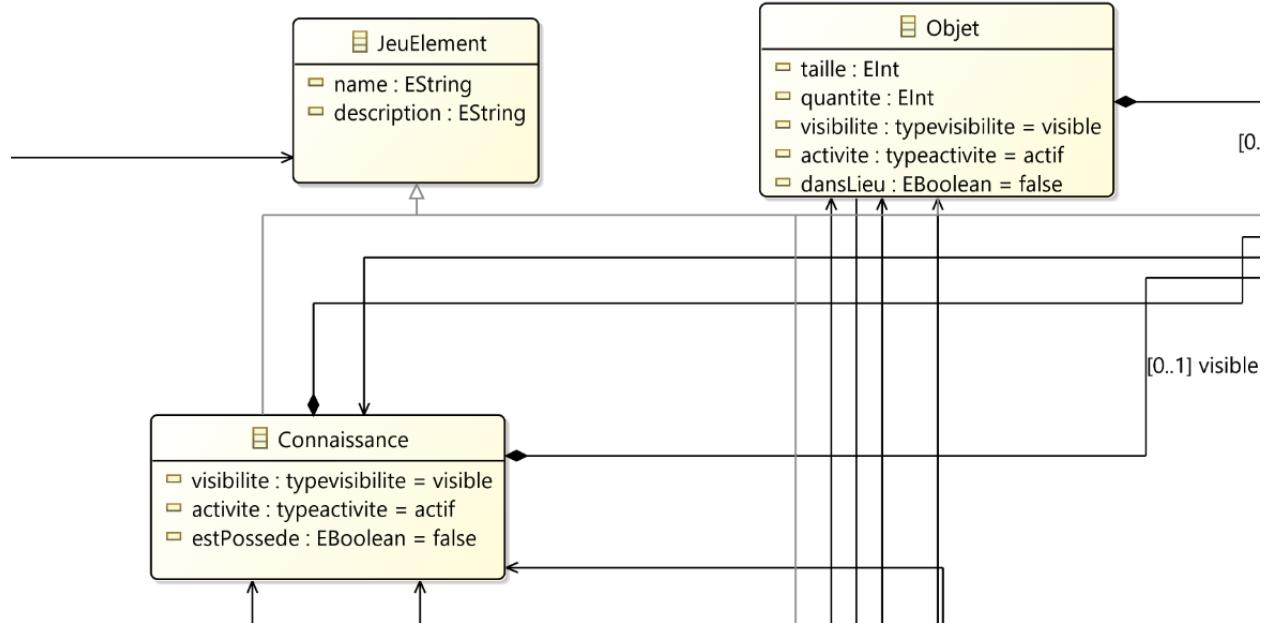


FIGURE 6 – Classes représentant des objets et connaissances sur le jeu. Elles peuvent appartenir à un joueur ou à un lieu

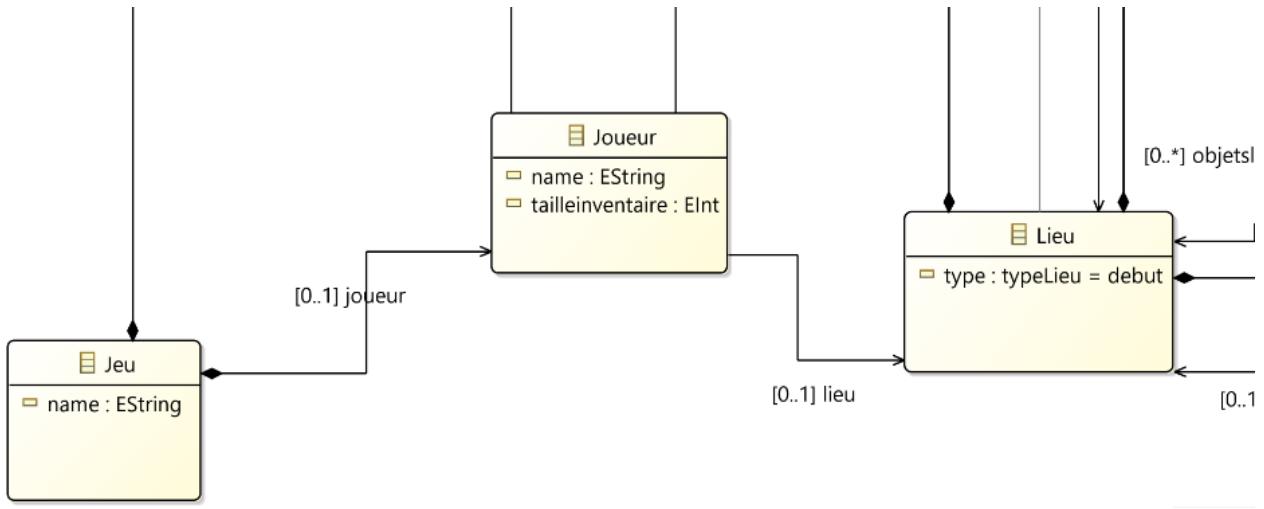


FIGURE 7 – Classes principales du jeu où on définit le nom du jeu et les caractéristiques du joueur



FIGURE 8 – Classes des types énumérés pour définir le type des lieux et l'ouverture et visibilité des éléments ainsi que le type de comparaison

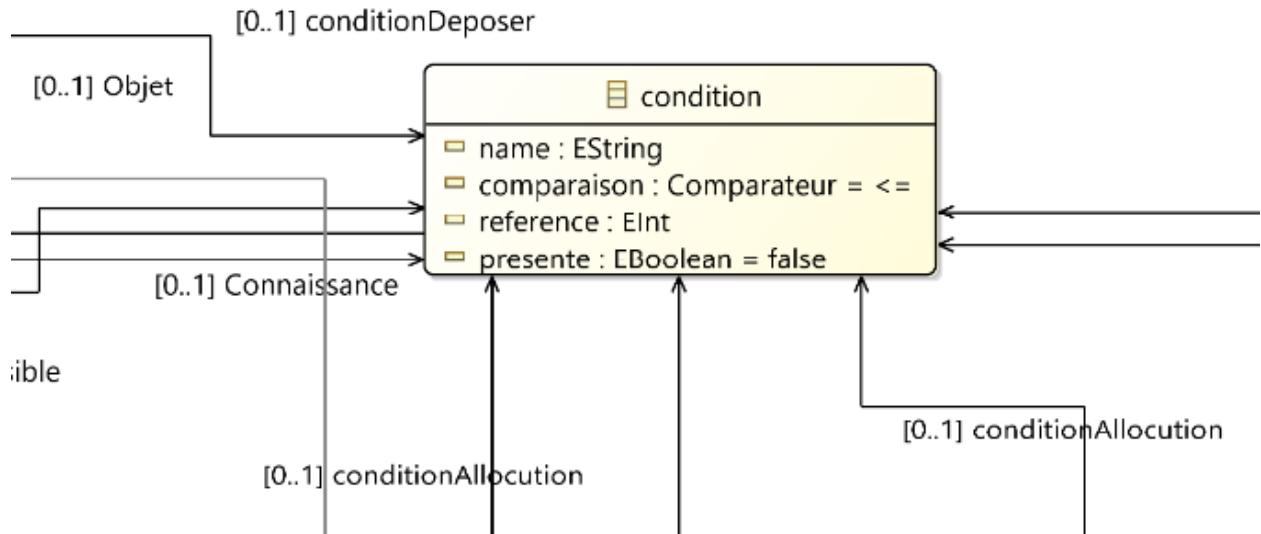


FIGURE 9 – Classe représentant une condition définie soit sur un objet soit sur une connaissance

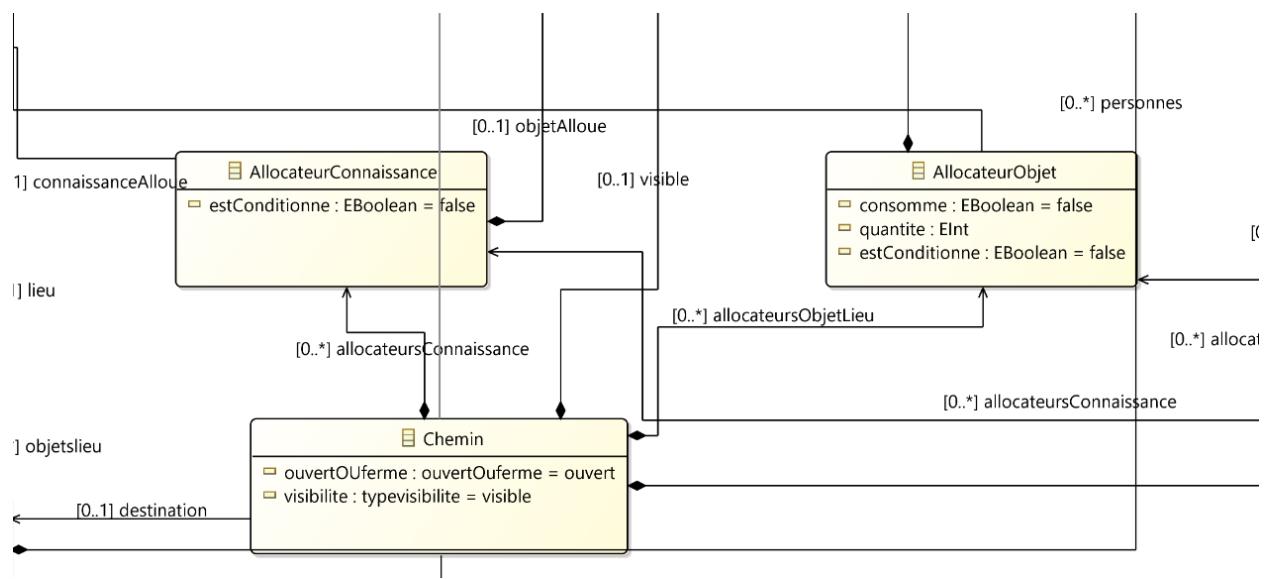


FIGURE 10 – Classes représentant un chemin et le don d'objets et connaissances dans un chemin ou action

3 Version Petri net et LTL de Enigme

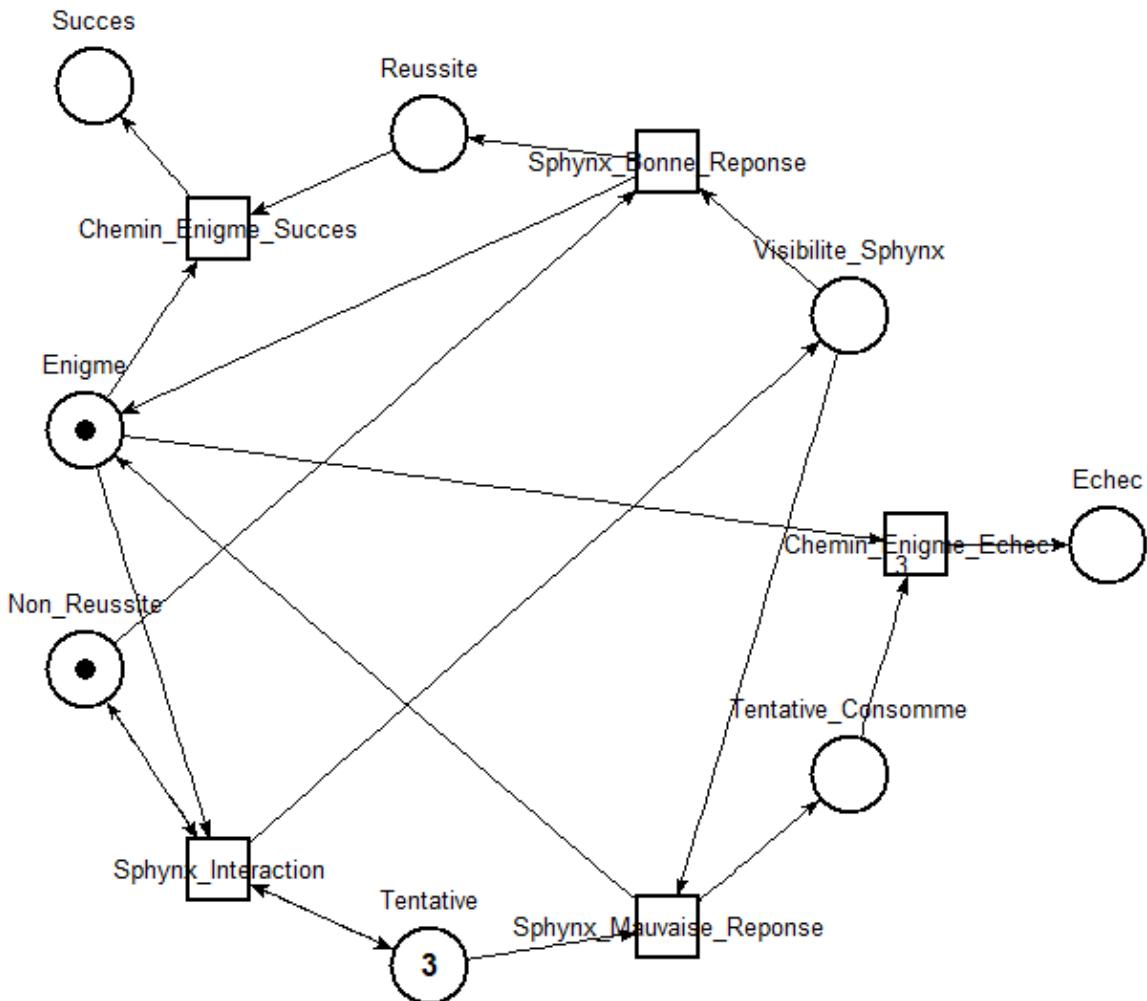


FIGURE 11 – Petri net représentant le jeu Enigme avec Tina

Nous avons représenté le jeu Enigme en suivant les principes décrits dans la section ATL mais en simplifiant un peu le réseau vu qu'on connaît le modèle à représenter.

En effet le chemin qui est d'habitude composé par plusieurs places et transitions ici n'est représenté que par une transition qui va de la place Enigme à la place Succes et de la place Enigme à la place Echec respectivement.

L'objet tentatives est représenté par une place ainsi que la connaissance Reussite. Nous avons ici besoin des places Non-Reussite pour représenter le fait que le joueur n'a pas de connaissance et la place Tentative-consomme pour représenter le fait que le joueur a consommé ses 3 tentatives pour pouvoir aller dans le lieu ECHEC.

La transition Sphynx-Interaction est activée lorsqu'on veut commencer une interaction avec le Sphynx (elle ne peut être activée que lorsqu'on a encore des tentatives et on n'a pas la réussite). La transition Sphynx-Mauvaise-Reponse enlève une tentative et la Sphynx-Bonne-Reponse donne une réussite.

Logiquement les transitions Chemin-Enigme-Succes et Chemin-Enigme-Echec sont activables que si on a respectivement une réussite ou trois tentatives consommées.

```

p1 Enigme (1)
p1 Echec (0)
p1 Succes (0)
p1 Tentative (3)
p1 Reussite (0)
p1 Visibilite_Sphynx (0)
p1 Tentative_Consumme(0)
p1 Non_Reussite (1)

tr Chemin_Enigme_Echec Enigme Tentative_Consumme*3 -> Echec
tr Chemin_Enigme_Succes Enigme Reussite -> Succes
tr Sphynx_Interaction Enigme Tentative Non_Reussite -> Visibilite_Sphynx Tentative Non_Reussite
tr Sphynx_Mauvaise_Reponse Visibilite_Sphynx Tentative -> Enigme Tentative_Consumme
tr Sphynx_Bonne_Reponse Visibilite_Sphynx Non_Reussite -> Reussite Enigme

```

FIGURE 12 – enigme.net

Nous avons représenté les propriétés LTL suivantes :

- ligne 3 - Si on est à une place représentant un lieu final le Petri net est dead c'est-à-dire il n'y a pas des Transitions activables.
- ligne 4 - A un moment le Petri net doit rentrer dans un état tel que aucune transition n'est activable.
- ligne 5 - Si on est à un état dead il faut qu'on soit dans une des places qui représentent des lieux finaux.
- ligne 6 - Le joueur n'arrive jamais à une place finale. C'est une propriété qui doit donner faux dans l'évaluation de selt.
- ligne 8 à 10 - Si on est dans un des lieux, on ne peut pas être dans les autres lieux c'est-à-dire qu'on ne peut pas être dans plusieurs lieux en même temps.

```

1  pp finished = Echec\Success;      #Etat de fin si le joueur se trouve sur Echec ou Succes
2
3  [] (finished => dead);          #Presence du joueur sur Echec ou Succes mène à une fin
4  [] <> dead;                   #Toujours, il y aura un fin
5  [] (dead => finished);         #Toujours, si fin alors il y a présence du joueur soit sur Echec soit sur Succes
6  - <> finished;
7
8  [] Enigme => -(Echec\Success);  #Un lieu à la fois
9  [] Succes => -(Enigme\Echec);
10 [] Echec => -(Succes\Enigme);

```

FIGURE 13 – enigme.ltl

4 Programme en Java de l'exemple énigme

Nous avons réalisé une conception préliminaire du générateur de code qui produit le prototype en Java correspondant à un modèle de jeu donné dans la section 1.1. Initialement, le jeu énigme se compose de trois lieux, un de début nommé Enigme et deux de fin qui représentent le succès, nommé Succès, et l'échec, Echec. Quand on lance le jeu, l'explorateur se trouve dans le lieu Enigme, il ne possède ni objets ni connaissances mais il a 3 tentatives, en revanche, le lieu Enigme possède 5 objets et 5 connaissances, le lieu Enigme contient une personne Sphinx qui peut interagir avec l'explorateur afin de débloquer un chemin. Toutes les actions précédemment décrites sont possibles grâce à un menu textuel qui s'affiche quand on lance le jeu comme sur la figure suivante.



FIGURE 14 – Lancement de Enigme.java

Dans le menu textuel, on peut exécuter plusieurs actions depuis le menu textuel. On peut demander le nombre d'objets ou de connaissances dont l'explorateur dispose, on peut avoir des informations sur le lieu courant éventuellement les objets, connaissances et personnes qu'il contient, on peut interagir avec le Sphinx, on peut prendre un chemin ou bien quitter le jeu.

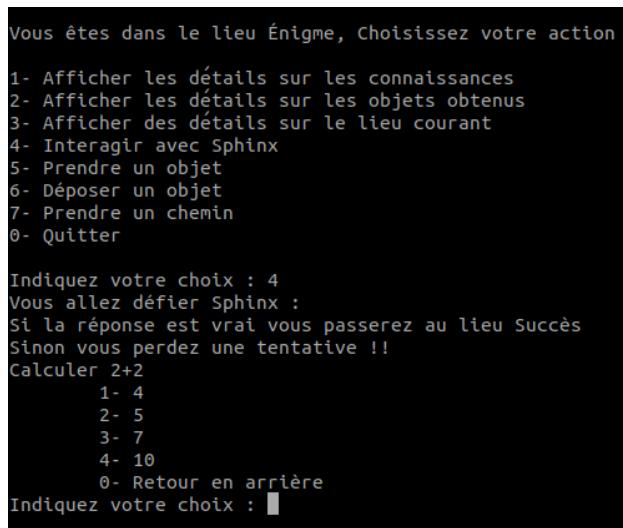


FIGURE 15 – Interagir avec le Sphinx

Dans la suite de cette partie, on va dérouler deux scénarios, le premier mène vers le lieu Succès et le deuxième mène vers le lieu Echec, le point commun entre les deux scénarios c'est que le jeu se termine quand on arrive dans le lieu Echec ou Succès, ceci sera démontré dans la partie LTL. On revient sur le menu textuel précédent et on va choisir le fait d'interagir avec le Sphinx, cette personne en retour va nous poser une question.

Si on donne une bonne réponse, on aura débloqué le chemin vers le lieu Succès, et puis on prendra le chemin pour aller à Succès et le jeu se termine.

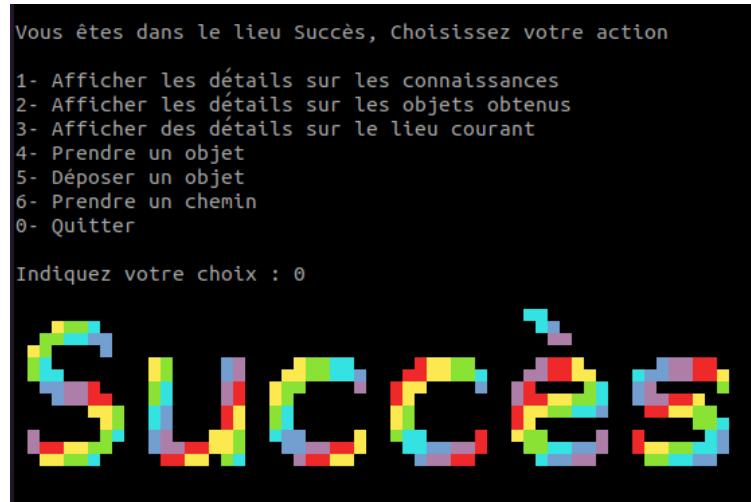


FIGURE 16 – Terminaison avec un succès

Sinon les tentatives que l'explorateur possède vont se décrémenter, si le nombre de tentatives atteint 0 on aura débloqué le chemin vers Echec, l'explorateur prendra le chemin échec et puis le jeu se termine.



FIGURE 17 – Terminaison avec un échec

5 Définition de la sémantique statique avec OCL

Le métamodèle défini en Xtext ne permet pas d'exprimer toutes les contraintes qu'un modèle de jeu doit respecter, ainsi, il serait judicieux de définir une sémantique statique avec OCL. La définition de ces contraintes seront dans un fichier *game.ocl* séparé du métamodèle.

5.1 Exemples de contraintes OCL définies

```
-- Vérifier que le nom d'un élément du jeu n'est pas null
inv nomjeuNul ('Le jeu n a pas un nom'):
    name <> null

-- Vérifier que le nom d'un élément du jeu n'est pas vide
inv nomjeuVide ('Le nom du jeu est vide'):
    name <> ''
```

FIGURE 18 – Le nom d'un élément du jeu ne doit être ni vide ni null

```
--Le joueur ne peut pas porter une charge d'objets dépassant sa taille d'inventaire
inv chargeLimite :
    AllocateurObjet.allInstances() -> collect (quantite*objetAlloue.taille)
        -> sum() <= tailleinventaire
```

FIGURE 19 – Le joueur ne peut pas porter une quantité d'objets supérieure à la taille de son inventaire

```
--Un jeu dispose d'un seul lieu de type début
inv debutUnique :
    JeuElement.allInstances() -> select(d|d.oclisTypeOf(Lieu))
        -> collect(d|d.oclaType(Lieu))
        -> select(d| d.type  = typeLieu::debut)
        -> size() = 1
```

FIGURE 20 – Un jeu ne peut pas disposer de plus d'un lieu de type début

5.2 Exemples de contraintes OCL non respectées pour un modèle de jeu

Nous prenons pour exemple d'un modèle de jeu, le jeu FormationGuerrier où un guerrier est formé selon des niveaux représentés par le passage d'un lieu à un autre par le biais d'un chemin qui alloue des *connaissances* et/ou *objets* au guerrier s'il réussit à les parcourir.

```

Chemin Barriere2Soins {
    de Entrainementbarriere a ApprendreSoins
    ouvertOuferme ouvert
    visibilite visible
    description "Le guerrier a appris à ouvrir et à fermer une barrière et obtient des clefs"
    connaissance {{Ouvrirbarriere}, {Fermerbarriere}}
    objet {{Clefbarriere 1}}
},

```

FIGURE 21 – Allocation de deux connaissances et un objet au guerrier dès le passage par le chemin Barriere2Soins

Lorsque le guerrier réussit à parcourir le chemin Barriere2Soins liant le lieu EntrainementBarriere à ApprendreSoins, il acquiert les deux connaissances Ouvrirbarrier et Fermerbarrier grâce aux allocateurs d'objets et de connaissances.

```

Joueur Guerrier {
    connaissance { MaitriseEpee , MaitriseArc , Soins , Ouvrirbarriere}
    inventaire 100
    objet { EpeeAlpha, EpeeBeta, ArcGamma , ArcZeta , Potionsdevie , Potionsenergie , Clefbarriere }
    lieu EntrainementArme
}

```

FIGURE 22 – Définition du Joueur guerrier

Selon notre définition du métamodèle jeu, la définition d'un joueur doit toujours comprendre toutes les connaissances ou objets possédés ou susceptibles d'être possédés par le joueur durant tout le jeu, avec une quantité d'un objet nulle si le joueur ne dispose pas de cet objet au début de jeu et une connaissance caractérisée par un booléen false si le joueur ne dispose pas de cette connaissance en début de jeu.

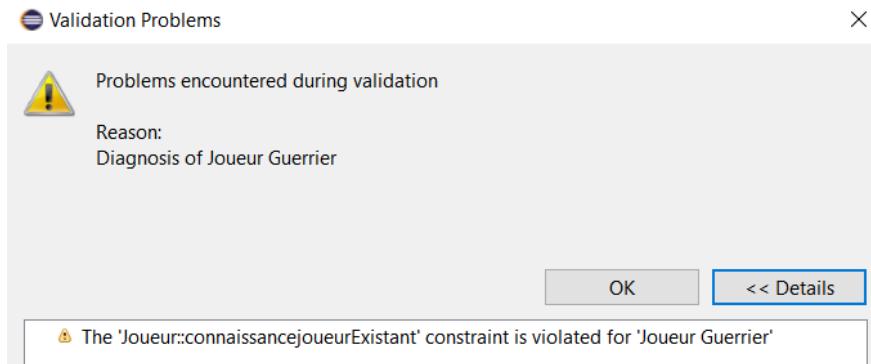


FIGURE 23 – La contrainte OCL Joueur : :connaissancejoueurExistant n'est pas respectée

Dans le modèle de jeu FormationGuerrier, la connaissance Fermerbarrier n'est pas définie dans la définition du joueur *Figure 9*, or le passage par le chemin Barriere2Soins alloue au joueur cette connaissance. La contrainte *Joueur : :connaissancejoueurExistant* qui stipule le fait que la définition d'un joueur doit toujours comprendre toutes les connaissances ou objets possédés ou susceptibles d'être possédés par le joueur durant tout le jeu n'est pas respectée et lève alors une erreur.

6 Définition d'une transformation M2T avec Acceleo - Propriétés LTL à partir d'un modèle

L'un des grands objectifs de ce projet est de vérifier l'existence d'une solution pour un jeu. En effet, la modélisation d'un jeu ne permet pas de déterminer si un jeu a une solution ou pas et surtout si la complexité du jeu est assez importante ou plusieurs classes sont définies. Ainsi, on se réfère à la boîte à outils TINA qui permet de vérifier si les états de fin d'un réseau de Petri sont atteignables, si c'est le cas alors le réseau de Petri aura une terminaison. Pour ce faire, la boîte à outils TINA s'appuie sur un réseau de Petri et les propriétés de logiques temporelles *LTL* qui lui sont associées.

Par conséquence, pour savoir si un modèle de jeu a une terminaison ou pas, on peut le transformer en un modèle de réseau de Petri et rédiger les propriétés de logique temporelle qui le caractérisent.

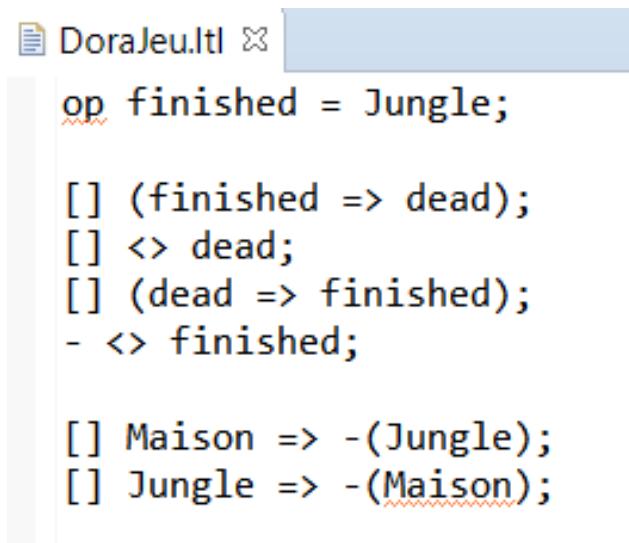
- On utilise l'outil ATL pour la transformation d'un modèle de jeu en un réseau de Petri. (*voir section 7*).
- On utilise une transformation modèle à texte qui permet d'obtenir les propriétés de logique temporelle *LTL* à partir d'un modèle de jeu.

```
1 [comment encoding = UTF-8 /]
2 [module ToLTL('http://www.n7.fr/JEU')]
3
4
5= [template public jeuToLTL(aJeu : Jeu)]
6 [comment @main/]
7 [file (aJeu.name +'.ltl', false, 'UTF-8')]
8 op finished = [for (l : Lieu | aJeu.getLieuxfin()) separator ('\\\/') after (';')] [l.name/] [/for]
9
10 ['['] '['']' /] (finished => dead);
11 ['['] '['']' /] <> dead;
12 ['['] '['']' /] (dead => finished);
13 - <> finished;
14
15 [ aJeu.getLieux() -> unlieualafois() /]
16 [/file]
17 [/template]
18
19= [template public unlieualafois(Lieux : OrderedSet(Lieu))]
20 [for(l : Lieu | Lieux)]
21 ['['] '['']' /] [l.name/] => -([for (lexclus : Lieu | (Lieux - Set{l})) separator ('\\\/')] [lexclus.name/] [/for]);
22 [/for]
23 [/template]
24
25
26 [query public getLieuxfin(jeu: Jeu) : OrderedSet(Lieu) =
27   jeulement -> select( l | l.oclIsTypeOf(Lieu))
28     -> collect ( l | l.oclAsType(Lieu))
29     -> select( l | l.type = typelieu::fin)
30     -> asOrderedSet()
31
32 /]
33
34 [query public getLieux(jeu: Jeu) : OrderedSet(Lieu) =
35   jeulement -> select( l | l.oclIsTypeOf(Lieu))
36     -> collect ( l | l.oclAsType(Lieu))
37     -> asOrderedSet()
38
```

FIGURE 24 – Acceleo : Transformation M2T - LTL à partir d'un modèle de jeu

6.1 Exemples de transformation M2T d'un modèle de jeu en propriétées LTL

6.1.1 Modèle de jeu Dora :



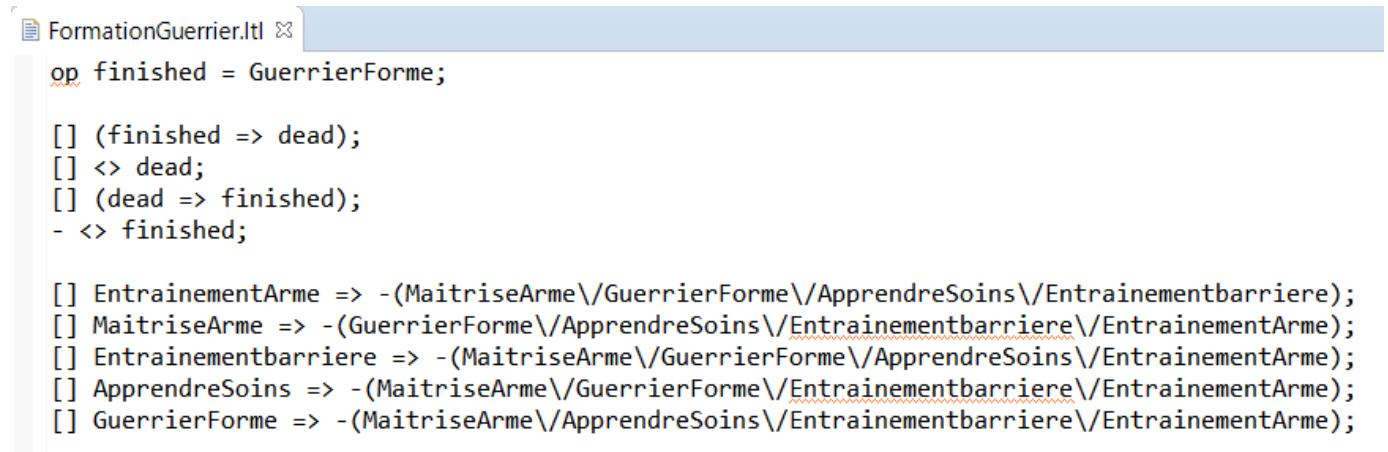
```
DoraJeu.ltl ✘
op finished = Jungle;

[] (finished => dead);
[] <> dead;
[] (dead => finished);
- <> finished;

[] Maison => -(Jungle);
[] Jungle => -(Maison);
```

FIGURE 25 – Transformation M2T - LTL à partir du modèle de jeu Dora

6.1.2 Modèle de jeu FormationGuerrier



```
FormationGuerrier.ltl ✘
op finished = GuerrierForme;

[] (finished => dead);
[] <> dead;
[] (dead => finished);
- <> finished;

[] EntrainementArme => -(MaitriseArme\GuerrierForme\ApprendreSoins\Entrainementbarriere);
[] MaitriseArme => -(GuerrierForme\ApprendreSoins\Entrainementbarriere\EntrainementArme);
[] Entrainementbarriere => -(MaitriseArme\GuerrierForme\ApprendreSoins\EntrainementArme);
[] ApprendreSoins => -(MaitriseArme\GuerrierForme\Entrainementbarriere\EntrainementArme);
[] GuerrierForme => -(MaitriseArme\ApprendreSoins\Entrainementbarriere\EntrainementArme);
```

FIGURE 26 – Transformation M2T - LTL à partir du modèle de jeu FormationGuerrier

6.2 Vérification de la terminaison du jeu Enigme

A partir du réseau de Petri du jeu Enigme qu'on a défini *enigme.net* et les propriétés de logique temporelles qui lui sont associés *enigme.ltl* on peut vérifier la terminaison du jeu grâce à la boîte à outil TINA :

```

1  bp finished = Echec\!/Succes;      #Etat de fin si le joueur se trouve sur Echec ou Succes
2
3  [] (finished => dead);           #Presence du joueur sur Echec ou Succes mène à une fin
4  [] <> dead;                   #Toujours, il y aura un fin
5  [] (dead => finished);         #Toujours, si fin alors il y a présence du joueur soit sur Echec soit sur Succes
6  - <> finished;
7
8  [] Enigme => -(Echec\!/Succes);  #Un lieu à la fois
9  [] Succes => -(Enigme\!/Echec);
10 [] Echec => -(Succes\!/Enigme);

```

FIGURE 27 – Propriétés LTL pour le jeu Enigme

```

ftoubali@hydre:~/Bureau/ProjetIDMPresentation$ tina enigme.net enigme.ktz
# net noname, 8 places, 5 transitions
# bounded, not live, not reversible
# abstraction      count      props      psets      dead      live #
#   states          14          8          14          4          4 #
# transitions       13          5          5          0          0 #

```

FIGURE 28 – Outil Tina pour produire enigme.ktz

```

Selt version 3.4.4 -- 01/05/16 -- LAAS/CNRS
ktz loaded, 14 states, 13 transitions
0.001s

- source enigme.ltl;
operator finished : prop
TRUE
TRUE
TRUE
FALSE
state 0: Debut Non_Reussite Tentative*3
-Sphynx_Interaction->
state 1: Non_Reussite Tentative*3 Visibilite_Sphynx
-Sphynx_Bonne_Reponse->
state 2: Debut Reussite Tentative*3
-Chemin_Enigme_Succes->
state 3: L.dead Succes Tentative*3
-L.deadlock->
state 4: L.dead Succes Tentative*3
[accepting all]
TRUE
TRUE
TRUE
0.003s

```

FIGURE 29 – Vérification des propriétés LTL

- Les trois premières propriétés sont vraies, en effet, la présence de l'explorateur sur l'un des lieux Succes ou Echec correspond à une fin puisqu'ils sont des lieux d'état fin. Aussi, on a toujours une fin, ensuite, si le jeu est terminé alors l'explorateur sera soit sur le lieu Succes soit sur le lieu Echec.

- La quatrième propriété est fausse, ainsi, toujours, à un moment il y aura une fin, et l'outil selt exhibe un contre exemple .

- Enfin, les trois dernières propriétés sont vraies, en effet, le joueur ne peut se trouver que dans un seul lieu à la fois.

7 Définition d'une transformation M2M avec ATL - Réseau de Pétri à partir d'un modèle

7.1 Les règles ATL

Tout d'abord nous représentons les éléments les plus généraux du jeu qui seront ensuite utilisés par le reste des éléments. Ces éléments de base sont regroupés dans le type JeuElement et sont traduits de la manière suivante :

Un lieu est transformé dans une place de même nom que le lieu, le nombre de tokens à cette place étant de 0 s'il s'agit d'un lieu normal ou final et de 1 s'il s'agit d'un lieu initial (à noter qu'on a adopté la convention de n'avoir qu'un seul lieu initial dans le jeu).

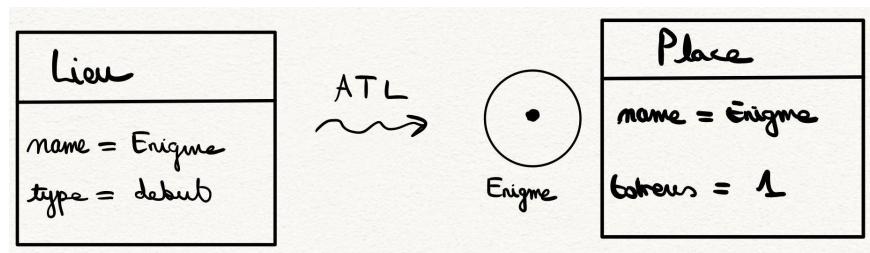


FIGURE 30 – Représentation Petri net d'un Lieu

Un objet qui figure dans la liste d'objets du joueur est transformé dans une place qui porte le nom : nomObjet-nomJoueur et qui a autant de tokens que l'attribut objet.quantite.

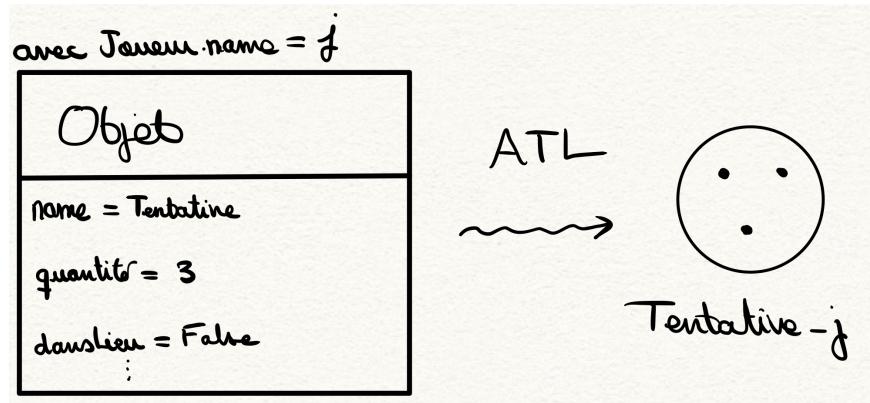


FIGURE 31 – Représentation Petri net d'un objet possédé par le joueur

Une connaissance qui figure dans la liste de connaissances du joueur est transformée en une place qui porte le nom : nomConnaissance-nomJoueur et qui a 1 token si cette connaissance est possédée par le joueur et aucun token sinon (à regarder selon connaissance.estPossede).

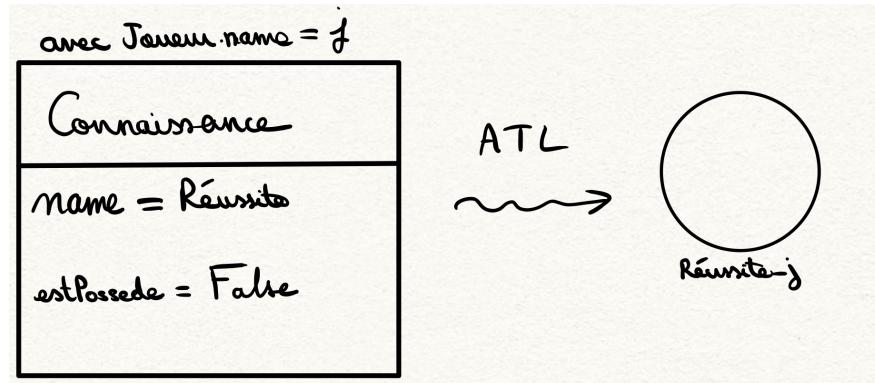


FIGURE 32 – Représentation Petri net d'une connaissance associée au joueur

Ensuite on transforme tous les objets qui se trouvent dans un lieu qu'on va appeler "lieu" dans ce qui suit. Ici on modélise l'action de "prendre un objet" par une transition qui s'appelle prendre-nomObjet-nomLieu, et l'action "déposer un objet" par la transition qui s'appelle deposer-nomObjet-nomLieu et qui est définie comme le décrit le dessin suivant. Enfin, l'objet présent dans un lieu est représenté par une place qui a autant de tokens que la quantité de l'objet et qui s'appelle nomObjet-nomLieu.

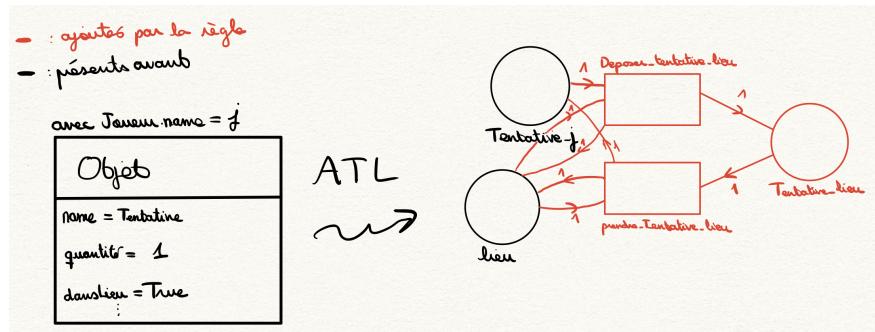


FIGURE 33 – Représentation Petri net d'un objet qui se trouve sur un lieu et les actions "prendre" et "déposer" associées

On pourrait avoir représenté la visibilité de l'objet en créant une place qui s'appellerait nomObjet-lieu-visible et une transition reliant cette place avec le lieu telle qu'elle n'est activée que si la condition de visibilité est vérifiée. Cette nouvelle place serait alors celle reliée aux transitions prendre et déposer. D'ailleurs, on aurait pu ajouter des arcs sur la transition deposer selon les conditions définies à objet.conditionDeposer tels que la transition n'est activée que si la condition pour déposer est vérifiée.

Ensuite nous avons la règle chemin2petrinet qui génère un ensemble de places arcs et transitions modélisant un chemin :

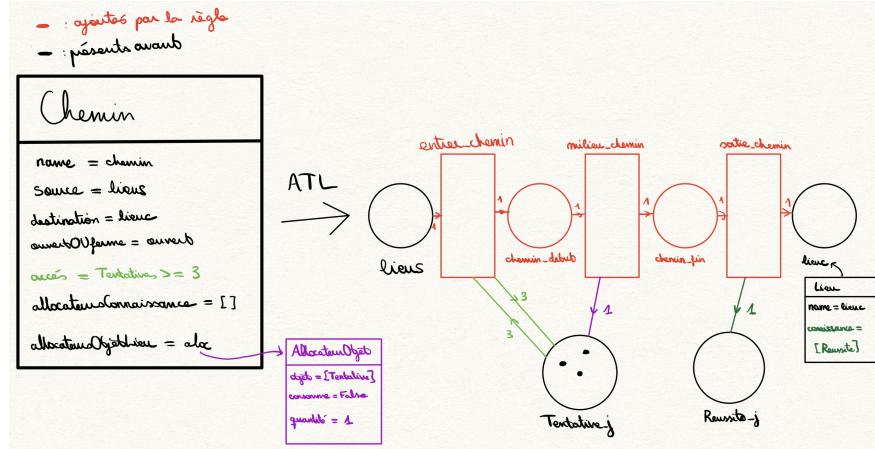


FIGURE 34 – Représentation Petri net d'un chemin avec condition d'accès et allocateurs objets ou connaissances non conditionnés

Le chemin décrit comment la condition d'ouverture est appliquée à la transition entree-chemin, et les dons/consommations d'objets et connaissances sont appliqués par la transition milieu-chemin. En particulier on a fait la simplification qui consiste à ignorer les conditions sur le don d'objets dans le passage, mais si ces conditions étaient prises en compte il faudrait définir une place et 2 transitions pour chaque allocateur objet et connaissance dans le chemin (une transition qui puisse être activée lorsque la condition est vérifiée et l'autre qui peut être activée sinon). Nous observons bien que lorsque le joueur active la dernière transition, on lui donne toutes les connaissances présentes dans le lieu cible (ce comportement n'a pas pu être implanté dans l'ATL par manque de temps).

Finalement il nous reste une règle qui définit les personnes dans les différents lieux et ses interactions, il s'agit de la règle personne2petrinet. Le schéma suivant représente comment est transformée une personne en Petri Net et nous verrons dans la suite comment transformer l'interaction associée à cette personne ainsi que les choix et actions associés (toutes ces générations sont faites à travers de called rules récursives mais le code n'aboutit pas parce que la définition du modèle ne permet pas ce type de code).

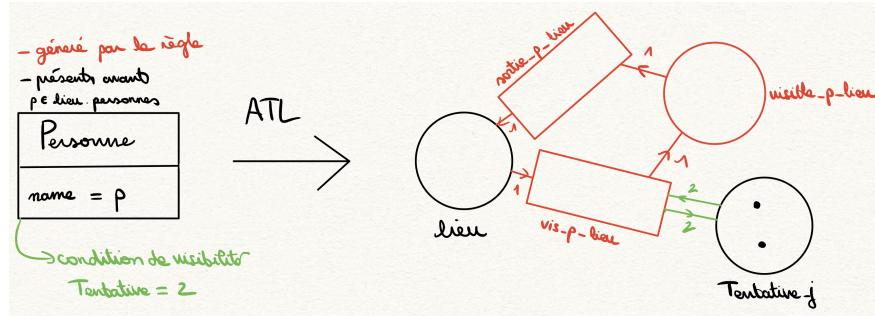


FIGURE 35 – Représentation Petri net d'une personne selon la condition de visibilité

Enfin, chaque choix sera représenté par une place et les choix initiaux seront reliés à la personne à travers une transition. Chaque action est représenté par une transition qui mène soit vers le lieu s'il s'agit d'une action finale, soit vers la place qui représente le choix suivant.

7.2 Limites

Le code ATL transforme bien le modèle en Petri net mais il ne prend pas compte des conditions de visibilité, d'accès aux chemins et d'allocation de ressources. Cette lacune est en partie due au manque de temps et aussi à la difficulté d'implanter de telles structures avec les métamodèles présents. Si on voulait introduire les conditions "inférieur que" par exemple il aurait fallu introduire dans le modèle Petri net un arc type qui soit arc inhibiteur.

D'autre part, les choix initiaux dans une interaction sont bien représentés mais les choix successifs qui découlent des actions ne sont pas bien initialisés dans le réseau de Petri, en effet il faudrait avoir prévu un attribut dans l'interaction qui regroupe tous les choix possibles qui vont être présentés au joueur qu'ils soient initiaux ou pas. De même pour les actions, parce que sinon j'ai eu des problèmes pour relier une action à la personne qui la réalise.

8 Conclusion

Ce projet nous a permis de mieux comprendre et exploiter la transformation de métamodèles. Nous avons pu mettre en pratique les notions développées en cours et nous avons pu à nouveau utiliser grâce à ce projet une grande variété d'outils que nous avons appris à utiliser et qui nous permettent de manipuler des objets très précis en les transformant dans des formats plus adaptés à certaines manipulations.