

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

Résumé

Ce document décrit le travail demandé aux étudiants du département Sciences du Numérique de l'ENSEEIHt inscrits pour la session 1 de l'année universitaire 2020-2021 de l'UE GLS.

Dans ce projet sont testées les facultés des étudiants à construire une infrastructure basée modèle autour d'une problématique concrète. Cette approche s'effectuera en deux axes : une **partie conception** et une **partie réalisation** en utilisant les outils du projet Eclipse Modelling pris en main au cours du module.

Deux livraisons auront lieu au cours du projet : une livraison correspondant à une conception préliminaire qui sera suivi d'un retour des enseignants et une livraison finale avant la recette du projet (oral, démonstration).

Les dates de remise sont précisées à la fin de ce sujet. Il est impératif de rendre les travaux dans les délais. **Tout retard sera sanctionné.** Les étudiants sont encouragés à travailler de manière régulière, et ce dès la publication de ce sujet.

Table des matières

1	Objectifs du projet	2
1.1	Modélisation de jeux d'exploration	2
1.2	Vérification de l'existence de la solution dans un jeu	4
1.3	Validation de la transformation en réseau de Petri	4
1.4	Génération de prototypes	5
2	Déroulement du projet	5
2.1	Tâches à réaliser	5
2.2	Documents à rendre	7
2.3	Dates importantes	7



Ce projet consiste à produire une chaîne de modélisation, de vérification et de génération de code pour des jeux de parcours/découverte.

La modélisation consiste à concevoir un langage dédié pour décrire le jeu sous la forme d'un modèle et à implanter les outils d'édition, vérification et génération associés.

La vérification permet d'assurer qu'il existe une solution pour le jeu décrit par un modèle. Pour répondre à cette question, nous utilisons les outils de *model-checking* définis sur les réseaux de Petri au travers de la boîte à outils TINA. Il nous faudra donc traduire un modèle de jeu en un réseau de Petri.

La génération de code permet de construire 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.

Vous utiliserez les outils suivants étudiés en TP et exploités pendant le mini-projet SimplePDL2TINA : Xtext, Acceleo, ATL. Vous réutiliserez le métamodèle des réseaux de Petri et la traduction vers le format TINA qui ont été réalisés pendant les TPs/mini-projet.

Contrairement aux TPs/mini-projet, le métamodèle sera généré à partir de la syntaxe concrète du langage définie en Xtext.

1 Objectifs du projet

Ce projet consiste pour l'essentiel à définir une chaîne de modélisation, vérification et génération de code pour des modèles de jeux (leur description est donnée en section 1.1). Les principales étapes sont les suivantes :

1. Définition de la syntaxe concrète du langage de modélisation de jeu en utilisant Xtext et génération du métamodèle Ecore. Cette solution permet de travailler plus facilement en équipe et de préserver, autant que possible, les exemples réalisés précédemment lors d'évolution du métamodèle (contrairement au format XMI).
2. Définition de la sémantique statique avec OCL dans un fichier séparé du métamodèle créé avec l'éditeur CompleteOCL.
3. Utilisation de l'infrastructure fournie par EMF pour manipuler les modèles. Celle-ci est générée automatiquement par Xtext.
4. Définition d'une transformation de modèle à texte (M2T) avec Acceleo, pour engendrer les propriétés LTL à partir d'un modèle de jeu.
5. Définition d'une transformation de modèles à modèles (M2M) avec ATL pour produire un réseau de Petri à partir d'un modèle de jeu.
6. 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. Le code généré utilisera le langage et les bibliothèques choisis par l'équipe.

1.1 Modélisation de jeux d'exploration

Nous vous proposons de concevoir un langage dédié (Domain Specific Modeling Language) pour modéliser les jeux d'exploration. L'analyse des besoins/recueil des exigences a permis d'obtenir les informations suivantes :

- E₁ L'objectif d'un jeu d'exploration est de visiter un territoire composé de lieux connectés par des chemins.
- E₂ Le joueur unique est l'explorateur.
- E₃ L'explorateur possède un nombre illimité de connaissances et un nombre limité d'objets.
- E₄ Un explorateur peut posséder plusieurs exemplaires d'un même objet.
- E₅ Chaque objet est qualifié par sa taille.
- E₆ Le nombre d'objets que peut posséder un explorateur est limité par la taille cumulée des objets possédés.
- E₇ Les lieux explorés peuvent contenir des connaissances, des objets et des personnes.
- E₈ Le point de départ et les points de fin de l'exploration sont des lieux particuliers.
- E₉ Les connaissances, les objets et les personnes contenus dans un lieu peuvent être visibles/actifs ou invisibles/inactifs selon des conditions.
- E₁₀ Les chemins peuvent être ouverts ou fermés selon des conditions.
- E₁₁ Lorsqu'il se trouve dans un lieu, l'explorateur reçoit les connaissances visibles de ce lieu.
- E₁₂ Lorsqu'il se trouve dans un lieu, l'explorateur peut prendre les objets visibles de son choix.
- E₁₃ Lorsqu'il se trouve dans un lieu, si cela est autorisé par des conditions, l'explorateur peut déposer les objets de son choix.
- E₁₄ Les objets déposés sur un lieu resteront sur place et pourront être repris ultérieurement par l'explorateur.
- E₁₅ Lorsqu'il se trouve dans un lieu, l'explorateur peut emprunter un chemin visible et ouvert de son choix.
- E₁₆ Le passage par un chemin peut transmettre à l'explorateur des connaissances et des objets.
- E₁₇ Le passage par un chemin peut consommer des objets possédés par l'explorateur.
- E₁₈ Les connaissances transmises et les objets transmis et consommés lors du passage par un chemin peuvent dépendre de conditions.
- E₁₉ Les conditions sont des combinaisons logiques des connaissances et objets possédés par l'explorateur. Les conditions peuvent dépendre du nombre d'exemplaires d'un objet (relation de comparaison avec des constantes). Pour des raisons de simplicité, les conditions seront écrites en forme normale disjonctive.
- E₂₀ L'explorateur peut interagir avec les personnes présentes dans un lieu.
- E₂₁ Les interactions permettent à l'explorateur de recevoir des connaissances et des objets.
- E₂₂ Les interactions prennent la forme de choix.
- E₂₃ Le choix de début de l'interaction peut dépendre de conditions.
- E₂₄ Un choix consiste à proposer à l'explorateur plusieurs actions.
- E₂₅ L'explorateur doit choisir une action ou quitter l'interaction s'il s'agit d'un choix de fin.
- E₂₆ Le fait qu'un choix soit un choix de fin peut dépendre de conditions.
- E₂₇ Les actions proposées dépendent de conditions et des choix précédents de l'explorateur.
- E₂₈ Une action peut attribuer à l'explorateur des connaissances et des objets.

- E₂₉ Les connaissances et objets attribués dépendent de conditions.
- E₃₀ Une action peut consommer des objets de l'explorateur.
- E₃₁ Les lieux, chemins, connaissances et objets sont qualifiés par une description textuelle qui peut dépendre de conditions.

Ces éléments permettent de modéliser un grand nombre de jeux différents. A titre d'exemple, les objets peuvent représenter des ressources (points de vie, aliments, soins, carburant, munitions, etc) ; les interactions peuvent représenter des combats, des déplacements complexes, des énigmes, etc ;

Exemple *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*.

1.2 Vérification de l'existence de la solution dans un jeu

La complexité d'un modèle de jeu peut être telle que le concepteur ne pourra plus déterminer si le jeu possède une solution. Nous proposons d'utiliser les outils de vérification de modèles de la boîte à outils TINA pour vérifier que les états de fin sont atteignables. Il est pour cela nécessaire de traduire les modèles de jeu en réseau de Petri et de générer les propriétés de logique temporelle exprimant l'existence d'une ou plusieurs solutions pour le jeu.

Vous pourrez également générer une propriété permettant de synthétiser un contre-exemple constituant une solution pour chaque lieu de fin du jeu.

1.3 Validation de la transformation en réseau de Petri

Comme pour tout programme écrit, il est important de valider la transformation de modèle. Afin de valider la transformation des modèles de jeu en réseau de Petri, une possibilité est de vérifier que la sémantique des éléments d'un jeu est préservée par le modèle de réseau de Petri correspondant. Ces invariants sont appelés *propriétés de sûreté*. En voici quelques exemples :

- l'explorateur ne peut être présent que dans un seul lieu à la fois ;
- une interaction ne peut présenter qu'un seul choix à la fois.

On peut alors écrire une transformation modèle à texte qui traduit ces propriétés de sûreté sur le modèle de Petri. L'outil *selt* permettra alors de vérifier si elles sont effectivement satisfaites sur le modèle de réseau de Petri. Si ce n'est pas le cas, c'est que la traduction contient une erreur ou que l'invariant n'en est pas un !

1.4 Génération de prototypes

Le concepteur doit déterminer si son jeu est intéressant avant de démarrer le développement de la partie multimédia du jeu (images, sons, musiques, vidéos, etc) qui est la plus coûteuse.

Nous proposons dans ce but de générer un prototype interactif en mode texte dans le langage de programmation de votre choix, en utilisant les bibliothèques de votre choix. L'objectif est la simplicité du code généré et de la transformation qui génère le code. Le prototype peut être rudimentaire. Il ne s'agit donc pas de faire la modélisation par objet la plus satisfaisante d'un jeu d'aventure, mais de générer le code le plus simple possible. En particulier, ce code n'a pas pour objectif d'être générique, extensible, réutilisable. Il sera réengendré pour chaque modification du modèle de jeu.

L'analyse des besoins/recueil des exigences a permis d'obtenir les connaissances suivantes :

- P₁ Le prototype doit afficher à la demande du joueur les informations sur les connaissances et les objets (nom et quantité) de l'explorateur ainsi que sa capacité de stockage disponible.
- P₂ Le prototype doit afficher le lieu courant dans lequel se trouve l'explorateur.
- P₃ Le prototype doit afficher à la demande du joueur les connaissances, les objets, les personnes et les chemins présents et visibles dans le lieu courant.
- P₄ Le prototype doit demander au joueur ce qu'il souhaite faire quand il se trouve dans un lieu :
 - afficher des détails sur les connaissances et les objets qu'il possède.
 - afficher des détails sur le lieu courant.
 - interagir avec une personne présente et visible dans le lieu courant.
 - prendre un objet présent et visible dans le lieu courant.
 - déposer un objet dans le lieu courant.
 - emprunter un chemin visible et ouvert dans le lieu courant.
- P₅ Les données affichées correspondent au qualificatif texte associé aux lieux, connaissances, objets et personnes.
- P₆ Lors d'une interaction, le prototype doit présenter au joueur les différentes actions possibles dans le choix courant et lui demander soit de choisir une de ces actions soit de quitter l'interaction si le choix courant est un choix de fin.
- P₇ Quand le joueur a choisi une action, le prototype doit modifier les connaissances et les objets de l'explorateur en conséquence ainsi que le choix courant puis poursuivre l'interaction.

2 Déroulement du projet

Ce projet est un travail **en équipe** dont les membres font partie du même groupe de TD.

Les techniques mises en œuvre doivent être celles présentées dans le module de GLS : Ecore, EMF, OCL, Xtext, Acceleo et ATL ainsi qu'un langage de programmation cible de la génération de code qui sera choisi librement par chaque équipe.

Pour chaque partie, voici les tâches à réaliser, les documents à rendre et les dates de remise.

2.1 Tâches à réaliser

- T₁ Définir avec Xtext une syntaxe concrète textuelle pour les modèles de jeu. Le métamodèle sera celui engendré par Xtext. Une attention particulière sera portée à la facilité d'utilisation

de la syntaxe concrète proposée.

- T₂ Modéliser l'exemple donné section 1.1 avec la syntaxe textuelle proposée en T₁.
- T₃ Réaliser une conception préliminaire de la transformation de modèles de jeux vers les réseaux de Petri. Il s'agit de prendre l'exemple précédent et de construire le réseau de Petri correspondant. Le réseau de Petri pourra être construit avec l'outil `nd` de TINA.
- T₄ Réaliser une conception préliminaire du générateur de code qui produit le prototype (dans le langage de votre choix) correspondant à un modèle de jeu. Il s'agit d'abord d'écrire directement le programme qui correspondant à l'exemple donné section 1.1 pour en déduire ensuite des règles générales qui s'appliqueront sur les éléments d'un modèle de jeu.
Attention, il ne s'agit pas de construire un interprète d'un modèle de jeu mais un programme aussi simple que possible permettant d'exécuter le jeu. Le modèle du jeu ne sera pas lu par le programme à l'exécution mais il aura été représenté dans le code de ce programme. Il ne doit donc pas y avoir de structures de données similaires aux classes du métamodèle. Par exemple, les connaissances acquises peuvent être représentées par des attributs booléens du nom de la connaissance. De même les objets possédés et les quantités associées peuvent être représentés par des attributs entiers du nom de l'objet. En particulier, il n'est pas utile ni pertinent de s'appuyer sur le code Java généré par EMF pour programmer le jeu si le langage cible choisi par l'équipe est Java.
- T₅ Valider la pertinence, la complétude et la facilité d'utilisation du langage défini en T₁ en rédigeant des tests plus simples pour chaque élément du langage ainsi qu'un exemple réaliste combinant toutes les capacités du langage.
Il n'est pas utile de faire des tests au format XMI avec l'éditeur arborescent. Les fichiers XMI seront générés à partir des modèles textuels avec l'outil générique de traduction de modèles textes gérés par Xtext vers du XMI inclus dans la distribution Eclipse GLS (dernière entrée du menu en faisant un clic droit sur un fichier texte valide géré par Xtext).
- T₆ Définir les contraintes OCL pour capturer les contraintes sur les modèles de jeu, contraintes qui ne sont pas déjà présentes sur le métamodèle.
- T₇ Valider la pertinence et la complétude des contraintes en rédigeant des tests élémentaires ne respectant pas les contraintes et en montrant que les tests et exemples rédigés précédemment respectent les contraintes.
- T₈ Implanter la transformation de modèles de jeux vers les réseaux de Petri en utilisant ATL.
- T₉ Valider cette transformation en utilisant les tests rédigés précédemment.
- T₁₀ Implanter le générateur de code pour produire les prototypes dans le langage de votre choix en utilisant Acceleo.
- T₁₁ Valider cette transformation en utilisant les tests et exemples rédigés précédemment.
- T₁₂ Implanter un générateur de propriétés LTL permettant de vérifier l'existence d'une solution pour un modèle de jeu.
- T₁₃ Valider ce générateur en utilisant les tests et exemples rédigés précédemment.
- T₁₄ Implanter un générateur de propriétés LTL correspondant aux invariants des modèles de jeu pour valider la transformation écrite (voir section 1.3).
- T₁₅ Valider ce générateur en utilisant les tests et exemples rédigés précédemment.

2.2 Documents à rendre

Les consignes pour rendre les documents suivants seront données sur la page du module (les documents seront rendus via SVN).

- D₁ Le modèle Xtext décrivant la syntaxe concrète textuelle des modèles de jeux (`game.xtext`).
- D₂ Une vue graphique mise en page du métamodèle généré par Xtext (`game.png`).
- D₃ Le modèle de jeux de l'exemple *enigme* donné en section 1.1 exprimé dans la syntaxe concrète textuelle définie avec Xtext (`enigme.SUFFIXE`, utilisez le suffixe choisi lors de la création du projet Xtext).
- D₄ Un réseau de Petri au format Tina représentant le déroulement du jeu de l'exemple *enigme* et les propriétés LTL exprimant l'existence d'une solution pour ce jeu (`enigme.net` et `enigme.ltl`).
- D₅ Le programme écrit dans le langage de votre choix qui exécute le jeu de l'exemple *enigme* (`enigme.SUFFIXE`, utilisez le suffixe correspondant au langage de votre choix).
- D₆ D'autres exemples de modèles de jeux (en expliquant l'intérêt de chaque exemple). Ces modèles serviront à illustrer les différents éléments réalisés dans le cadre du projet.
- D₇ Les fichiers de contraintes OCL au format CompleteOCL associés à ce métamodèle, avec des exemples (et contre-exemples) qui montrent la pertinence de ces contraintes (`game.ocl`).
- D₈ Le code ATL de la transformation modèle à modèle des modèles de jeu vers les réseaux de Petri (`game2petrinet.atl`).
- D₉ Le code Aceleo des transformations modèle à texte vers les propriétés LTL (`game2ltl.mtl`) et vers le prototype dans le langage de votre choix (`game2prototype.mtl`).
- D₁₀ Un document concis (`rapport.pdf`) qui explique le travail réalisé. Attention, c'est ce document qui servira de point d'entrée pour lire les éléments rendus.

2.3 Dates importantes

Mercredi 9 décembre 2020 : Publication du sujet

11 décembre 2020 : Séance de travail sur le projet

Vendredi 18 décembre 2020 : Remise de la syntaxe textuelle et du métamodèle correspondant (format Xtext et image, livrable D_1 et D_2) et du dossier de conception préliminaire (exemple *enigme* écrit dans la syntaxe textuelle, livrable D_3), traduction manuelle en réseau de Petri et logique temporelle au format TINA (livrable D_4), traduction manuelle dans le langage de programmation choisi (livrable D_4)

Semaine du 6 janvier 2020 : Point d'avancement au cours d'une séance de travail projet

Mercredi 20 janvier 2020 : Rendu des livrables.

Jeudi 21 janvier 2020 : Oral du projet : démonstration et questions/réponses.

Vendredi 22 janvier 2020 : Version finale des livrables (les modifications faites depuis la version du 20 janvier doivent être détaillées dans une section changements du rapport).