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 2021-2022 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

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, Sirius, 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 généré 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. Vous réutiliserez la transformation des réseaux de Petri vers la syntaxe concrète de TINA réalisée pendant les TPs/BE.
5. Définition de syntaxes concrètes graphiques avec Sirius pour les points de vue Territoire et Dialogue.
6. Définition d'une transformation de modèles à modèles (M2M) avec ATL pour produire un réseau de Petri à partir d'un modèle de jeu. Vous réutiliserez le métamodèle des réseaux de Petri réalisé pendant les TPs/BE.
7. 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 :

E_1 L'objectif d'un jeu d'exploration est de visiter un territoire composé de lieux connectés par des chemins.

E_2 Le joueur unique est l'explorateur.

E_3 L'explorateur possède un nombre illimité de connaissances et un nombre limité d'objets.

E_4 Un explorateur peut posséder plusieurs exemplaires d'un même objet.

E_5 Chaque objet est qualifié par sa taille.

E_6 Le nombre d'objets que peut posséder un explorateur est limité par la taille cumulée des objets possédés.

E_7 Les lieux explorés peuvent contenir des connaissances, des objets et des personnes.

E_8 Le point de départ et les points de fin de l'exploration sont des lieux particuliers.

E_9 Les connaissances, les objets et les personnes contenus dans un lieu peuvent être visibles/actifs ou invisibles/inactifs selon des conditions.

E_{10} Les chemins peuvent être ouverts ou fermés selon des conditions.

E_{11} Les chemins peuvent être visibles ou invisibles selon des conditions.

E_{12} Lorsqu'il se trouve dans un lieu, l'explorateur reçoit les connaissances visibles de ce lieu.

E_{13} Lorsqu'il se trouve dans un lieu, l'explorateur peut prendre les objets visibles de son choix.

E_{14} Lorsqu'il se trouve dans un lieu, si cela est autorisé par des conditions, l'explorateur peut déposer les objets de son choix.

E_{15} Les objets déposés sur un lieu resteront sur place et pourront être repris ultérieurement par l'explorateur.

E_{16} Les chemins dans un lieu peuvent être obligatoires ou choisis par l'explorateur.

E_{17} Il ne peut y avoir qu'un seul chemin obligatoire visible et ouvert par lieu.

E_{18} Un chemin obligatoire, visible et ouvert est franchi automatiquement par l'explorateur dès qu'il arrive dans le lieu après les interactions avec les personnes présentes obligatoires.

E_{19} Lorsqu'il se trouve dans un lieu, l'explorateur peut emprunter un chemin visible et ouvert de son choix.

E_{20} Le passage par un chemin peut transmettre à l'explorateur des connaissances et des objets.

E_{21} Le passage par un chemin peut consommer des objets possédés par l'explorateur.

E_{22} Les connaissances transmises et les objets transmis et consommés lors du passage par un chemin peuvent dépendre de conditions.

E_{23} 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.

E_{24} L'explorateur peut interagir avec les personnes présentes dans un lieu.

E_{25} Les personnes peuvent être obligatoires ou choisies par l'explorateur.

E_{26} Il ne peut y avoir qu'une seule personne obligatoire par lieu.

E_{27} Une personne obligatoire débute son interaction dès que le joueur arrive dans le lieu.

E_{28} Les interactions permettent à l'explorateur de recevoir des connaissances et des objets.

E_{29} Les interactions prennent la forme de choix.

E_{30} Le choix de début de l'interaction peut dépendre de conditions.

E_{31} Un choix consiste à proposer à l'explorateur plusieurs actions.

E_{32} Les actions proposées peuvent dépendre de conditions.

E_{33} L'explorateur doit choisir une action proposée ou quitter l'interaction lorsqu'il s'agit d'un choix de fin.

E_{34} Le fait qu'un choix soit un choix de fin peut dépendre de conditions.

E_{35} Les actions proposées dépendent de conditions ainsi que des choix précédents de l'explorateur.

E_{36} Une action peut attribuer à l'explorateur des connaissances et des objets.

E_{37} Les connaissances et objets attribués dépendent de conditions.

E_{38} Une action peut consommer des objets de l'explorateur.

E_{39} Les lieux, chemins, connaissances et objets sont qualifiés par une description textuelle qui peut dépendre de conditions.

E_{40} Les objets peuvent être transformables selon des conditions.

E_{41} L'explorateur peut décider de transformer des objets qu'il possède. Ceux-ci sont alors consommés et remplacés par d'autres objets.

E_{42} Le résultat de la transformation d'objets transformables peut dépendre de conditions.

E_{43} Les objets transformables peuvent se transformer en objets qui peuvent être eux-même transformables.

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, les objets transformables peuvent représenter des soins qui se transforment en points de vie, des aliments qui se transforment en énergie, des recettes de fabrication d'objets à partir de ressources, etc.

Exemple *enigme* : On modélise un jeu d'énigme. Le territoire est composé de trois lieux, un de début nommé *Énigme* et deux de fin qui représentent le succès, nommé *Succès*, et l'échec, *Échec*. 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 *Énigme* contient une personne *Sphinx* qui est visible et obligatoire. 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

obligatoire allant du lieu *Énigme* au lieu *Succès* dont la visibilité est conditionnée par la possession de la connaissance *Réussite*. Il existe un chemin obligatoire allant du lieu *Énigme* au lieu *Échec* 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 boite 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 :

P_1 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.

P_2 Le prototype doit afficher le lieu courant dans lequel se trouve l'explorateur.

P_3 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.

P_4 Le prototype doit lancer les interactions automatiques avec les personnes présentes dans un lieu dès que l'explorateur entre dans ce lieu, y compris pour le lieu de départ du jeu.

P_5 Le prototype doit prendre les chemins automatiques visibles et ouverts après avoir lancé les interactions automatiques.

P_6 Le prototype doit demander au joueur ce qu'il souhaite faire quand il se trouve dans un lieu après avoir exécuté les interactions automatiques :
- 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.

P_7 Les données affichées correspondent au qualificatif texte associé aux lieux, connaissances, objets et personnes.

P_8 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.

P_9 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 (exceptionnellement, en majeure SIL, il est possible qu'une équipe contienne des membres de L12 et de L34). Les équipes doivent normalement être composées de 4 membres (ou de 3 membres si le nombre d'étudiants de la majeure n'est pas un multiple de 4). La composition des équipes sera saisie sur la page Moodle de la matière.

Les techniques mises en œuvre doivent être celles présentées dans le module de GLS : Ecore, EMF, OCL, Xtext, Sirius, 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

T_1 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. Cette facilité fera parti des critères de notation.

T_2 Modéliser l'exemple donné section 1.1 avec la syntaxe textuelle proposée en T_1.

T_3 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.

T_4 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 ni 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. La simplicité de ce programme fera parti des critères de notation.

T_5 Valider la pertinence, la complétude et la facilité d'utilisation du langage défini en T_1 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).

T_6 Définir les contraintes OCL pour capturer les contraintes sur les modèles de jeu, contraintes qui ne sont pas déjà présentes sur le métamodèle.

T_7 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.

T_8 Développer un éditeur graphique avec une vue présentant la partie territoire d'un modèle de jeux, et une vue présentant les parties interaction d'un modèle de jeux. Ces vues seront complétées par des contrôleurs pour pouvoir saisir graphiquement des éléments dans les modèles de jeux visualisés. Une attention particulière sera portée à la facilité d'utilisation de la syntaxe concrète proposée. Cette facilité fera parti des critères de notation.

T_9 Implanter la transformation de modèles de jeux vers les réseaux de Petri en utilisant ATL.

T_{10} Valider cette transformation en utilisant les tests rédigés précédemment.

T_{11} Implanter le générateur de code pour produire les prototypes dans le langage de votre choix en utilisant Acceleo.

T_{12} Valider cette transformation en utilisant les tests et exemples rédigés précédemment.

T_{13} Implanter un générateur de propriétés LTL permettant de vérifier l'existence d'une solution pour un modèle de jeu.

T_{14} Valider ce générateur en utilisant les tests et exemples rédigés précédemment.

T_{15} Implanter un générateur de propriétés LTL correspondant aux invariants des modèles de jeu pour valider la transformation écrite (voir section 1.3).

T_{16} 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).

D_1 Le modèle Xtext décrivant la syntaxe concrète textuelle des modèles de jeux (`game.xtext`).

D_2 Une vue graphique mise en page du métamodèle généré par Xtext (`game.png`).

D_3 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).

D_4 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`).

D_5 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.

D_6 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.

D_7 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`).

D_8 Le code ATL de la transformation modèle à modèle des modèles de jeu vers les réseaux de Petri (`game2petrinet.atl`).

D_9 Le code Acceleo des transformations modèle à texte vers les propriétés LTL (`game2ltl.mtl`) et vers le prototype dans le langage de votre choix (`game2prototype.mtl`).

D_{10} Les modèles Sirius décrivant l'éditeur graphique pour les deux points de vue sur les modèles de jeux (le point de vue Territoire et le point de vue Interaction).

D_{11} 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

Lundi 15 novembre 2021 : Publication du sujet.

Mercredi 17 décembre 2021 : Saisie de la composition des groupes sur la page Moodle de la matière.

Mercredi 24 novembre 2021 : Séance de travail sur le projet. Réponse aux questions sur le sujet.

Mercredi 1 décembre 2021 : Remise de la syntaxe textuelle et du métamodèle correspondant (format Xtext et image, délivrable D_1 et D_2) et du dossier de conception préliminaire (exemple *enigme* écrit dans la syntaxe textuelle, délivrable D_3), traduction manuelle en réseau de Petri et logique temporelle au format TINA (délivrable D_4), traduction manuelle dans le langage de programmation choisi (délivrable D_4).

Vendredi 3 décembre 2021 : Séance de travail sur le projet. Point d'avancement avec les enseignants.

Mercredi 10 décembre 2021 : Séance de travail sur le projet. Point d'avancement avec les enseignants.

Mardi 14 décembre 2021 : Rendu de la version pré-soutenance des livrables.

Jeudi 16 décembre 2021 : Oral du projet : démonstration et questions/réponses.

Mercredi 22 décembre 2021 : Version définitive des livrables (les modifications faites depuis la version du 14 décembre doivent être détaillées dans une section changements du rapport).