Mastériale : Model Checking

Mathieu ARLES
Samy DAOUD
William ATGER

13 février 2018

Table des matières

Introduction

Ce projet s'inscrit dans le thème du mastériale, à savoir, les stations de vélos Vélib/Smoovengo. Cependant, le model checking ne peut s'appliquer sans avoir de modélisation ou de code. Ainsi, ce travail n'est pas directement lié aux autres projets du mastériale. Nous ne faisons donc pas de mise en commun. En revanche, nous proposons ici une spécification qui respecte certaines propriétés prouvées mathématiquement. Ceci peut donc être considéré comme un travail préliminaire et servir pour un développement futur. Pour exposer ce qui a été fait, nous allons suivre le plan suivant : Tout d'abord, nous verrons les outils utilisés, rodin et Event-B. Ensuite, nous expliquerons notre spécification et ce qui a pu motiver nos différents choix. Pour finir, nous parlerons brièvement des preuves apportées.

Chapitre 1

Outils

1.1 Rodin

La plate-forme Rodin est un IDE basé sur Eclipse pour Event-B qui fournit un support efficace pour le raffinement et la preuve mathématique. La plate-forme est open source, contribue au framework Eclipse et est extensible avec des plugins tels que : le prouveur Atelier B ou encore ProB, que nous utilisons pour le model checking.

1.2 Event-B

Event-B est une méthode formelle pour modéliser et analyser un système à différents niveaux d'abstraction. De plus, on peut utiliser des preuves mathématiques pour vérifier la cohérence entre les différents niveaux de raffinement.

1.3 Fonctionnement

Un projet contient le développement mathématique complet d'un système de transitions discrètes. Il est composé de deux sortes de composants :

Contexte : Les contextes contiennent les ensembles porteurs, les constantes, les axiomes et les théorèmes d'un projet.

Machine : Les machines contiennent les variables, les invariants, les théorèmes et les événements d'un projet.

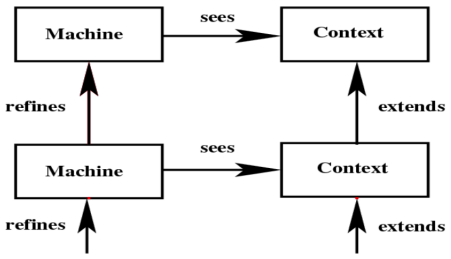

FIGURE 1.1 – Liens entre les différents types de composant

Comme on peut le voir sur le schéma ci-dessus, les machines peuvent « voir »les contextes, afin de pouvoir utiliser les ensembles définis par exemple. Ces contextes peuvent s'étendre entre eux pour être plus précis, ou plus complets. Les machines peuvent, elles, être raffinées afin de diminuer le niveau d'abstraction et se rapprocher le plus possible de la réalité. Cela ressemble de loin au principe d'héritage que l'on retrouve en orienté objet.

Chapitre 2

Spécification

2.1 Spécification informelle

Au tout début de notre projet, nous avons commencer par déterminer une spécification informelle. Cette dernière correspond à ceci :

- Nous avons plusieurs grands ensembles comme VELOS, PERSONNES et SITES. Nous avons défini SITES comme un grand ensemble car ils peuvent évoluer au cour du temps (ajout/ suppression).
- Un site a un nombre de places définies.
- Un site a un nombre de places libres.
- Un site a un nombre de vélos disponibles.
- On doit être capable d'identifier un vélo et savoir dans quelle station il est.
- On doit être capable de déplacer des vélos d'un site à l'autre.
- On doit être capable de mettre des vélos en réparation.
- Un vélo ne peut être emprunté que par un seul abonné. Un abonné ne peut emprunter qu'un seul vélo.
- On doit être capable de bloquer un utilisateur (non rendu de vélo)
- On doit être capable de débloquer un utilisateur s'il a justifié son non-rendu.

2.2 Contexte

Notre contexte, se découpe en deux parties. Il y a d'abord, les ensembles définis puis les axiomes. Ces derniers permettent de définir nos ensembles comme des ensembles finis. Dans le contexte, nous ne définissons que des ensembles qui nous permettront par la suite de « typer »des variables. Ces ensembles sont :

- PERSONNES : Ensemble des êtres humains.
- VELOS : Ensemble des vélos.
- SITES : Ensemble des stations de vélos.

Il faut bien avoir en tête qu'un élément d'un de ces ensembles ne fait pas obligatoirement partie de notre système. Ils servent, comme dit précédemment, à typer nos variables.

2.3 Machine Vélib

Notre machine vélib se découpe en quatre parties. La première consiste à spécifier quel contexte on va utiliser. La seconde permet de définir nos variables. La troisième concerne les invariants. Ces derniers permettent de typer nos variables mais aussi de spécifier des contraintes que notre système doit respecter. On peut par la suite faire des preuves mathématiques pour s'en assurer. Pour finir, la dernière partie est consacrée aux évènements. Il y a bien évidemment l'initialisation, mais aussi des évènements qui vont modifier l'état de notre système. Les preuves permettent donc de vérifier la cohérence entre les invariants et les différents évènements disponibles.

2.3.1 Variables

Nous avons au total six variables :

abonnes : Ensemble des abonnés Vélib.

velos : Ensemble des vélos possédés par la société Vélib.

sites : Ensemble des stations de vélos possédées par Vélib.

nbplaces : Ensemble des sites auxquels on attribue un nombre de places.

velos_in_site : Ensemble des sites auxquels on attribue un ensemble de vélos. On peut utiliser le cardinal afin d'obtenir le nombre de vélos dans un site et d'en déduire le nombre de places libres.

emprunts : Ensemble des abonnés auxquels on a attribué un vélo emprunté.

Les variables emprunts et velos_in_site permettent de savoir où se trouve un vélo à chaque instant.

2.3.2 Invariants

Nous avons déterminé les invariants suivants :

1. $abonnes \subseteq PERSONNES$
2. $velos \subseteq VELOS$
3. $sites \subseteq SITES$
4. $nbplaces \in sites \rightarrow \mathbb{N}$
5. $velos\_in\_site \in sites \rightarrow \mathbb{P}(velos)$
6. $\forall s \cdot s \in SITES \Rightarrow (s \in sites \Rightarrow nbplaces(s) \geq card(velos\_in\_site(s)))$

7. $emprunts \in abonnes \rightarrowtail velos$
8. $\forall s1, s2 \cdot ((s1 \in sites \wedge s2 \in sites \wedge s1 \neq s2) \Rightarrow (velos\_in\_site(s1) \cap velos\_in\_site(s2)) = \varnothing)$
9. $\forall v, s \cdot v \in velos \wedge s \in sites \wedge v \in ran(emprunts) \Rightarrow (v \notin velos\_in\_site(s))$

Comme vous pouvez le constater, certains invariants servent à typer une variable, d'autres sont des propriétés à vérifier. Nous pouvons lire les invariants comme suit :

1. abonnes est un sous-ensemble de PERSONNES.
2. velos est un sous-ensemble de VELOS.
3. sites est un sous-ensemble de SITES.
4. nbplaces est une fonction totale qui attribue à chaque site un nombre de place.
5. velos_in_site est une fonction totale qui attribue un sous-ensemble de velos à chaque site.
6. Pour tous les sites, le nombre de places doit être supérieur ou égal au nombre de vélos au sein-même du site. On ne peut pas avoir plus de vélos que de places dans un site.
7. emprunts est une fonction injective partielle qui attribue au plus un vélo à un abonné. La fonction injective partielle permet de vérifier qu'un abonné peut emprunter au plus un vélo.
8. Si un vélo est dans un site, alors il ne peut pas être présent dans un autre.
9. Si un vélo est emprunté, alors il ne peut pas être présent dans un site.

2.3.3 Évènements

Nous avons défini un ensemble d'évènements qui nous semblaient pertinents dans la spécification de notre système. Dans le cadre de notre projet, nous sommes restés très abstrait.

Nous avons au total dix évènements. La majeure partie de nos évènements possèdent les mêmes éléments. Nous avons d'abord les paramètres d'entrée. Ensuite, les gardes qui permettent de typer les paramètres mais aussi de rajouter des contraintes (un peu comme un invariant). Enfin, il y a les actions qui permettent de modifier notre système. Voici nos différents évènements :

Initialisation : Cet évènement permet de définir l'état initial de notre système. Par défaut, nous avons décidé de considérer qu'il est vide. Toutes nos variables sont des ensembles vides.

AddSite : Cet évènement permet d'ajouter un site. Pour cela, il prend en paramètre un site, un nombre de places et un ensemble de vélos de départ. Pour s'assurer que l'on respecte nos invariants, on vérifie que le nombre de places est supérieur ou égal au nombre de vélos. De plus, on vérifie qu'aucun des vélos de départ n'est emprunté ou dans un autre site.

AcheterVelos : C'est l'un des premiers évènements que l'on déclenche lors d'une exécution. En effet, il permet d'acheter des vélos qui pourront être ajoutés à un site déjà existant ou à un nouveau.

DeplacerVelos : Cette méthode permet de déplacer des vélos d'un site a vers un site b. Cet évènement découle directement de notre spécification. Il nous suffit de vérifier que le site b a assez de places pour recevoir ces vélos.

RemoveSite : Cet évènement est un peu exceptionnel car cela ne doit pas se faire tous les jours. Cependant, afin d'éviter des blocages lors d'une exécution, il faut faire certaines vérifications. On supprime le site, seulement s'il est plein, ou s'il reste de la place dans un autre site. Cela évite à un abonné de ne pas pouvoir rendre son vélo.

AddVelosToSite : Cet évènement permet d'ajouter des vélos à un site. On vérifie pour cela, que le site possède assez de places. De même, on vérifie que les vélos ajoutés ne sont pas empruntés, ni dans un autre site.

AddAbonne : Cela permet d'ajouter un abonné. Pour cet évènement, il suffit juste que cette personne ne soit pas déjà abonnée.

RemoveAbonne : On supprime un abonné. Pour cela, on vérifie qu'il n'ait pas un vélo emprunté. Cela permet de toujours savoir où se trouve nos vélos.

EmprunterVelo : Cet évènement permet à un abonné d'emprunter un vélo à un site donné. Pour cela, on vérifie qu'il n'ait pas déjà fait un emprunt. Cette vérification va bloquer notre utilisateur sinon.

RendreVelo : Un abonné rend un vélo. Pour cela, on vérifie que le site de destination possède bien le nombre de places nécessaire, mais aussi que cet abonné a effectivement un emprunt.

On remarque que la majeure partie des spécifications informelles sont représentées par des évènements, des variables, des invariants. Ensuite, les gardes permettent aussi de respecter nos spécifications. Certaines n'ont pas été formalisées, comme le fait de pouvoir mettre hors-service un vélo.

Chapitre 3

Model Checking et Preuves

3.1 LTL

La partie LTL est la plus facile à appliquer car cela se fait automatiquement. Le model checker nous a permis de revoir certaines gardes de nos évènements mais aussi un dead lock. Ainsi, le model checker nous apprend que nous ne violons jamais d'invariant, et que le système ne se bloque jamais.

3.2 Preuves

Les preuves mathématiques permettent de vérifier la cohérence de notre système, mais aussi le respect de nos invariants. Ainsi, si tout est prouvé, nous devrions avoir une spécification formelle correcte. Dans notre cas, la majeure partie des preuves ont pu être faites automatiquement. Pour cela, nous avons reformulé certaines gardes et invariants. Par exemple, les preuves ne fonctionnent pas très bien avec l'union généralisée. Dans ce cas, nous avons reformulé avec des quantificateurs. De cette manière, le prouveur s'en sort tout seul. Pour les autres preuves, nous avons ajouté les hypothèses manquantes au prouveur pour l'aider. En revanche, il nous restait trois preuves non faites. Pour ces dernières, nous avons demandé au prouveur de revoir l'objectif en fonction de toutes les hypothèses. Le prouveur a validé cette démarche.

Conclusion

Ce mastériale fut l'occasion pour nous de revoir des notions concernant la théorie des ensembles. De plus, il nous a permis d'apprendre de nouveaux outils. Nous avons pu pratiquer le model checking (même s'il est fait automatiquement), et voir un peu de preuves. Nous avons fini par obtenir une spécification qui semble correcte, même si nous n'avons pas formalisé tout ce qui était prévu. Les principales fonctionnalités ont été formalisées et vérifiées. Pour la suite, il est possible de raffiner notre machine afin d'intégrer le mécanisme de transactions avec la carte navigo. De même, il est toujours possible d'ajouter des critères à notre spécification.

CONTEXT VelibContext

SETS

PERSONNES

VELOS

SITES

PLACES

AXIOMS

axm1: $finite(PERSONNES)$

axm2: $finite(VELOS)$

axm3: $finite(SITES)$

END

MACHINE Velib
SEES VelibContext
VARIABLES

abonnes
velos
sites
nbplaces
velos_in_site
emprunts

INVARIANTS

inv1: $abonnes \subseteq PERSONNES$
inv2: $velos \subseteq VELOS$
inv3: $sites \subseteq SITES$
inv8: $nbplaces \in sites \to \mathbb{N}$
inv10: $velos\_in\_site \in sites \to \mathbb{P}(velos)$
inv14: $\forall s \cdot s \in SITES \Rightarrow (s \in sites \Rightarrow nbplaces(s) \geq card(velos\_in\_site(s)))$
inv11: $emprunts \in abonnes \rightarrowtail velos$
inv12: $\forall s1, s2 \cdot ((s1 \in sites \land s2 \in sites \land s1 \neq s2) \Rightarrow (velos\_in\_site(s1) \cap velos\_in\_site(s2)) = \varnothing)$
inv13: $\forall v, s \cdot v \in velos \land s \in sites \land v \in ran(emprunts) \Rightarrow (v \notin velos\_in\_site(s))$

EVENTS

Initialisation

begin

act1: $abonnes := \varnothing$
act2: $velos := \varnothing$
act3: $sites := \varnothing$
act4: $nbplaces := \varnothing$
act5: $velos\_in\_site := \varnothing$
act6: $emprunts := \varnothing$

end

Event AddSite ⟨ordinary⟩ $\widehat{=}$

any

site
nb_places
init_velos

where

grd1: $site \in SITES \setminus sites$
grd2: $nb\_places \in \mathbb{N} \land nb\_places > 0$
grd3: $init\_velos \subseteq velos \land card(init\_velos) > 0$
grd6: $init\_velos \cap ran(emprunts) = \varnothing$
grd4: $nb\_places \geq card(init\_velos)$
grd5: $\forall v, s \cdot v \in velos \land s \in sites \land v \in velos\_in\_site(s) \Rightarrow (v \notin init\_velos)$

then

act1: $sites := sites \cup \{site\}$
act2: $nbplaces(site) := nb\_places$
act3: $velos\_in\_site(site) := init\_velos$

end

Event AcheterVelos ⟨ordinary⟩ $\widehat{=}$

any

v

where

grd1: $v \subseteq VELOS \setminus velos$
grd2: $card(v) > 0$

then

act1: $velos := velos \cup v$

end

Event DeplacerVelos ⟨ordinary⟩ $\widehat{=}$

any
site_a
site_b
velos_a
where
grd1: $site\_a \in sites$
grd2: $site\_b \in sites$
grd5: $site\_a \neq site\_b$
grd3: $velos\_a \subseteq velos\_in\_site(site\_a)$
grd6: $card(velos\_a) > 0$
grd4: $nbplaces(site\_b) - (card(velos\_in\_site(site\_b))) \geq card(velos\_a)$
then
act2: $velos\_in\_site := velos\_in\_site \Leftarrow \{site\_a \mapsto velos\_in\_site(site\_a) \setminus velos\_a, site\_b \mapsto velos\_in\_site(site\_b) \cup velos\_a\}$
end

Event RemoveSite ⟨ordinary⟩ $\widehat{=}$
any
s
where
grd1: $s \in sites$
grd2: $card(velos\_in\_site(s)) = nbplaces(s) \vee (\exists s2 \cdot s2 \in sites \wedge s2 \neq s \wedge nbplaces(s2) > card(velos\_in\_site(s2)))$
then
act1: $velos\_in\_site := velos\_in\_site \setminus \{s \mapsto velos\_in\_site(s)\}$
act2: $nbplaces := nbplaces \setminus \{s \mapsto nbplaces(s)\}$
act3: $sites := sites \setminus \{s\}$
end

Event AddVelosToSite ⟨ordinary⟩ $\widehat{=}$
any
new_velos
site
where
grd1: $site \in sites$
grd2: $new\_velos \subseteq velos \wedge card(new\_velos) > 0$
grd4: $new\_velos \cap ran(emprunts) = \varnothing$
grd5: $\forall v, s \cdot v \in velos \wedge v \in new\_velos \wedge s \in sites \Rightarrow v \notin velos\_in\_site(s)$
grd3: $nbplaces(site) \geq card(new\_velos) + card(velos\_in\_site(site))$
then
act1: $velos\_in\_site(site) := velos\_in\_site(site) \cup new\_velos$
end

Event AddAbonne ⟨ordinary⟩ $\widehat{=}$
any
new_abonne
where
grd1: $new\_abonne \in PERSONNES \setminus abonnes$
then
act1: $abonnes := abonnes \cup \{new\_abonne\}$
end

Event RemoveAbonne ⟨ordinary⟩ $\widehat{=}$
any
a
where
grd1: $a \in abonnes$
grd2: $a \notin dom(emprunts)$
then
act1: $abonnes := abonnes \setminus \{a\}$
end

Event EmprunterVelo ⟨ordinary⟩ $\widehat{=}$

any
abonne
site
velo
where
grd1: $abonne \in abonnes$
grd4: $abonne \notin dom(emprunts)$
grd3: $site \in sites$
grd2: $velo \in velos \land velo \in velos\_in\_site(site)$
then
act1: $emprunts(abonne) := velo$
act2: $velos\_in\_site(site) := velos\_in\_site(site) \setminus \{velo\}$
end

Event RendreVelo ⟨ordinary⟩ $\widehat{=}$
any
abonne
site
where
grd1: $abonne \in abonnes$
grd2: $site \in sites$
grd3: $abonne \in dom(emprunts)$
grd4: $nbplaces(site) \geq card(velos\_in\_site(site)) + 1$
then
act2: $velos\_in\_site(site) := velos\_in\_site(site) \cup \{emprunts(abonne)\}$
act1: $emprunts := emprunts \setminus \{abonne \mapsto emprunts(abonne)\}$
end

END