Système de Gestion Ferroviaire

YOUSSEF MNAOUI,Mohammed NAHI,Missouri Taha

February 2025

1 Introduction

La conception des systèmes de gestion ferroviaire se base sur différents principes similaires à ceux des méthodes formelles,en particulier la méthode Event B. Dans ce projet ,nous allons utiliser Rodin pour modéliser la conception d'un système de gestion Ferroviaire,tout en respectant les exigences fournis sur le cahier des charges.

Figure 1: Schéma de la dynamique du train

2 Modélisation

Pour la modélisation de ce projet, nous allons concevoir un modèle Event-B .Ce modèle est composé de plusieurs machines reliées par des relation de raffinement.Par rapport aux contextes ,ils seront reliées par des relation extends .

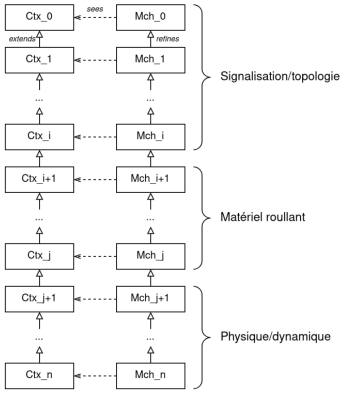

Figure 2: Schéma expliquant la modélisation

3 Couche 1: Topologie et Signalisation

Dans cette partie, deux machines ont été mises en place :

3.1 Mch_Topologie

Le but principale de cette machine est de déterminer la position de chaque train

- **Position des Trains :** La position de chaque train est déterminé par la fonction `train_position`, qui associe à chaque train le ou les deux tronçons où le train se trouve . Chaque tronçon est occupé par au plus un train cela est traduit par le fait que la fonction est injective.

```
inv1:   train_position ∈ TRAINS ⤖  ℙ1(TRONCONS) not theorem
inv3:   ∀train · card(train_position(train)) ≤ 2 not theorem
```

Figure 3: Invariants sur la fonction `train_position` dans le modèle Event-B

- **Déplacement des Trains :** Pour modéliser le déplacement des trains on autilsé deux events :

– **DEBUT_FRANCHISSEMENT** : C'est l'événement modélisant le début du franchissement d'un tronçon par un train vers un autre tronçon .

La dernière garde assure que le tronçon que le train veut franchir ne doit pas être occupé.

```
DEBUT_FRANCHISSEMENT:  not extended ordinary ›
ANY
  train  ›
  tr1  ›
  tr2  ›
WHERE
  grd1:  train ∈ TRAINS not theorem ›
  grd2:  train_position(train) = {tr1} not theorem ›
  grd3:  tr1 ↦ tr2 ∈ Reseau not theorem ›
  grd4:  ∀other_train ·(other_train∈dom(train_position)∧ tr2 ∉ train_position(other_train)) not theorem ›
THEN
  act1:  train_position(train) ≔ {tr1, tr2} ›
  act2:  sens_deplacement(train) ≔ (tr1↦ tr2) ›
END
```

Figure 4: L'événement `DEBUT_FRANCHISSEMENT`

– **FIN_FRANCHISSEMENT** : C'est l'événement qui modélise la fin du franchissement d'un tronçon par un train.Après cet événement le train occupe un seul tronçon.

```
FIN_FRANCHISSEMENT:  not extended ordinary ›
ANY
  train  ›
  tr1  ›
  tr2  ›
WHERE
  grd1:  train ∈ TRAINS not theorem ›
  grd2:  train_position(train) = {tr1, tr2} not theorem ›
  grd3:  sens_deplacement(train) = (tr1↦ tr2) not theorem ›
THEN
  act1:  train_position(train) ≔ {tr2} ›
  act2:  sens_deplacement≔{train}⩤sens_deplacement ›
END
```

Figure 5: L'événement `FIN_FRANCHISSEMENT`

Une fonction appelée `sens_deplacement` a été mise en place pour suivre la direction du déplacement du train . On pourra donc par la suite savoir et déterminer le tronçon à supprimer de la position du train .

3.2 Mch_Topologie1

Dans cette machine, on a introduit les deux notions suivantes.

- **Destination :** Chaque train est affecté par l'événement `Affecter_Destination` à une destination qui est un tronçon.

```
Affecter_Destination:  not extended ordinary
ANY
   train
   dest
WHERE
   grd1:  train ∈ TRAINS not theorem
   grd4:  {dest} ≠ train_position(train) not theorem
   grd5:  train ∉ dom(destination) ∨ (train ∈ dom(destination) ∧ train_position(train) = {destination(train)}) not theorem
   grd3:  dest∈ TRONCONS not theorem
   grd2:  train ∈ dom(train_position) not theorem
THEN
   act1:  destination(train) ≔ dest
END
```

Figure 6: L'événement `Affecter_Destination`

Chaque train doit avoir une destination différente de sa position actuelle et cette destination appartient à l'ensemble TRONCONS.

- **Route :**

  La route est une suite de tronçons. Elle peut être incomplète. Dans ce cas, si le train arrive à la fin de la route, on doit attendre jusqu'à ce qu'on lui associe une nouvelle route. On peut modifier la route d'un train. Dans notre modélisation, nous avons introduit une fonction `route` qui associe à chaque train $train_i$ un ensemble de couples (n_i, t_i), où n_i représente l'ordre dans lequel le tronçon t_i doit être parcouru par le train $train_i$. La position actuelle du train est donc $(1, train\_position(train_i))$.

```
inv1:   route ∈ TRAINS ⇸ ℙ(N1×TRONCONS) not theorem
inv2:   ∀t·(t ∈ dom(route) ⇒ train_position(t) ⊆ ran({i ↦ tr | (i ↦ tr) ∈ route(t)})) not theorem
```

Figure 7: Invariants sur la fonction `route` dans le modèle Event-B

L'événement `Affecter_Route` permet d'affecter une route à chaque train tout en respectant les exigences de notre fonction `route`.

```
Affecter_Route:  not extended ordinary
ANY
   train
   r
   trl
WHERE
   grd1:  train ∈TRAINS not theorem
   grd2:  train ∈dom(destination) not theorem
   grd3:  train_position(train) = {trl} not theorem
   grd4:  destination(train)≠ trl not theorem
   grd6:  (1↦trl)∈r not theorem
   grd7:  r ∈ ℙ(N1 × TRONCONS) not theorem
   grd8:  card(r) > 1 not theorem
   grd9:  (3index·(index ↦ destination(train))∈r ∧(index>1)∧ (∀i,tr·(i ↦ tr)∈r ⇒ isindex))∨ (destination(train) ≠ ran(r)) not theorem
   grd10: ∀i,tr·((i ↦ tr) ∈ r ∧ i<card(r)) ⇒ (∃tr2·((i+1)↦tr2)∈r∧ (tr↦tr2 ∈ Reseau)) not theorem
   grd11: ∀i·(∃tr, tr2 · (i ↦ tr) ∈ r ∧ (i ↦ tr2) ∈ r ⇒ tr = tr2) not theorem
THEN
   act1:  route(train) ≔ r
END
```

Figure 8: L'événement `Affecter_Route`

N.B. : on a pa pu implémenter le fait que le système peut aussi changer la route du train alors qu'il ne l'a pas terminée.

Concernant les deux autres événements raffinés : `Début_franchissemenet` : on a ajouté deux autres gardes permettant de s'assurer les éléments (1,t1), (2,t2)

appartiennent à la route (t1 est le tronçon où se trouve le train actuellement et t2 est le tronçon suivant que le train veut franchir).
`Fin_franchissemenet`: on supprime le dernier tronçon que le train vient de sortir, et on décrémente les autres tronçons de 1 , et donc le tronçon que le train vient de franchir sera de rang 1.

3.3 Validation et vérification

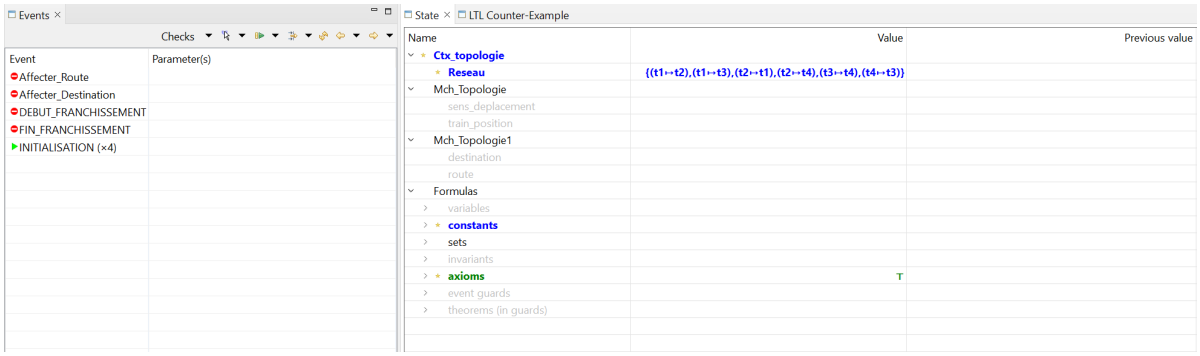

Figure 9: État initial du modèle ferroviaire avant toute affectation

Figure 10: Affectation de la destination pour le train train1 vers t1

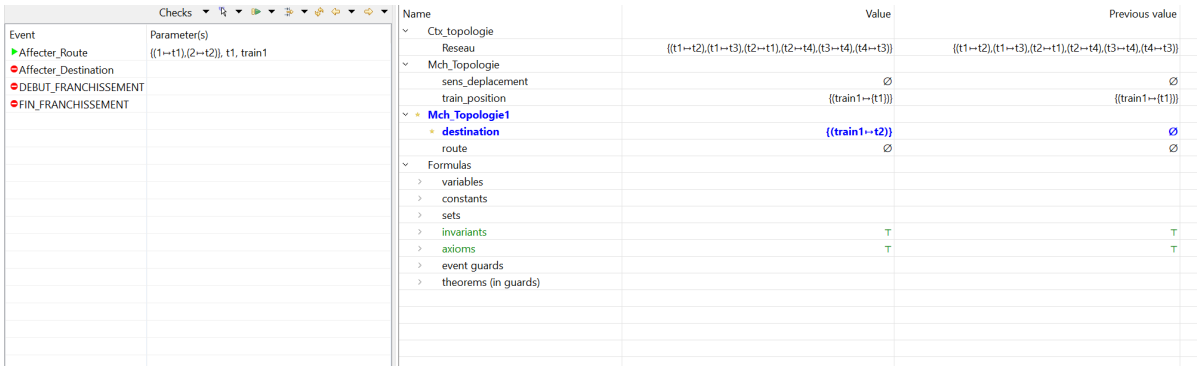

Figure 11: Affectation d'une route au train train1 à travers les tronçons t1 et t2

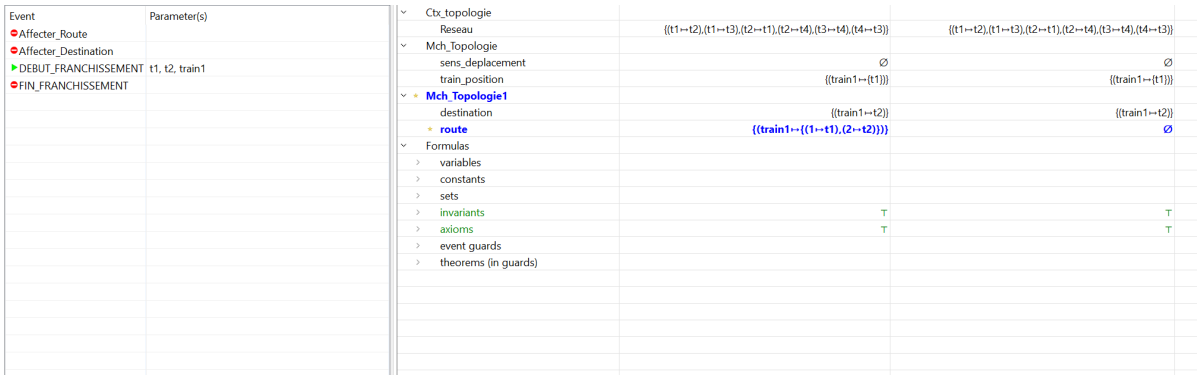

Figure 12: Début du franchissement de t1 vers t2 pour train1

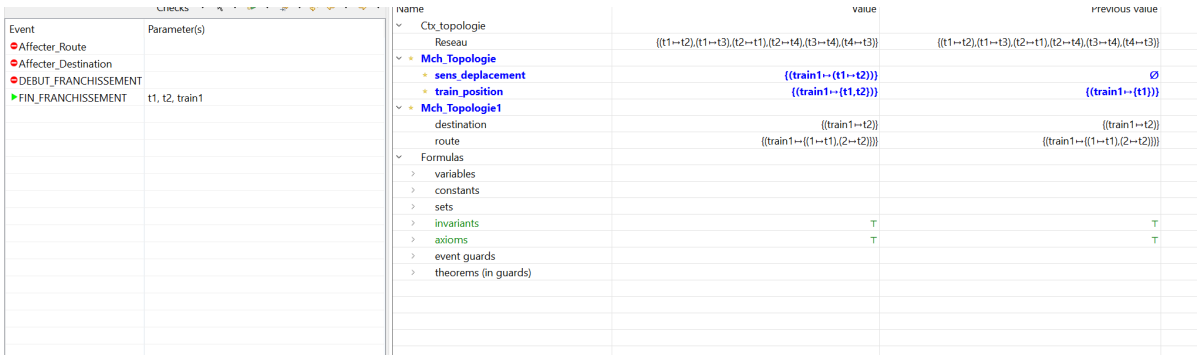

Figure 13: Fin du franchissement, mise à jour de la position du train train1

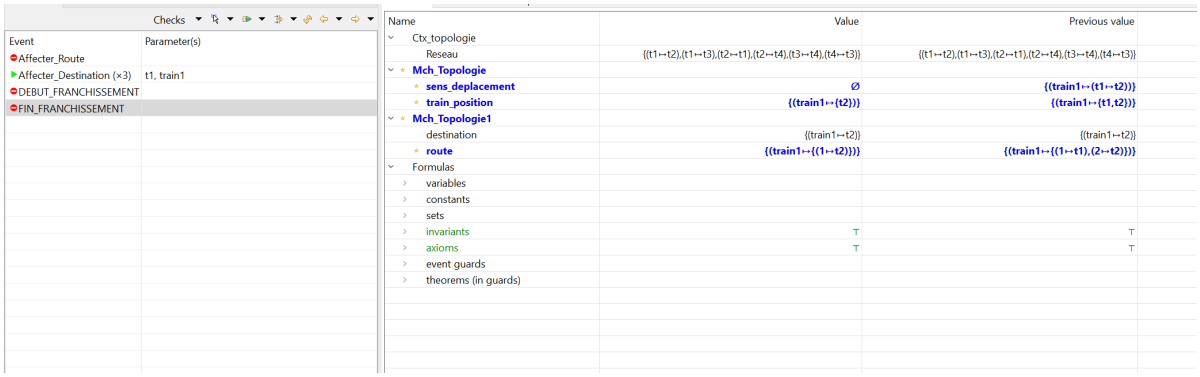

Figure 14: Affectation d'une nouvelle destination pour le train train1

On a effectué le model checking pour vérifier s'il y a des violations des invariants : aucune violation n'a été trouvée.

Les images suivantes représentent un cas d'animation Event-B permettant d'expliquer et de vérifier notre modélisation.

4 Couche 2: Matériel Roulant

Dans cette partie,le train est représentée par plusieurs wagons.Le nombre de wagon attribué à chaque train est aléatoire mais supérieur strictement à 3. Un train occupe un tronçon dés que sa motrice y est positionnée. le passage de tout les wagons d'un train d'un tronçon vers un autre représente Event Franchissement.

```
CONTEXT
    plus_wagon
EXTENDS
    Ctx_topologie_Instance
SETS
    WAGONS
CONSTANTS
    train_wagons     not symbolic
AXIOMS
    axm1:   train_wagons ∈ TRAINS ⟶ (N ⤖ WAGONS) not theorem
    axm8:   card(WAGONS)>10 not theorem
    axm7:   ∀tr ·(tr∈TRAINS)⇒(∃wagon·train_wagons(tr)(0)=wagon) not theorem
    axm5:   ∀train · train ∈ TRAINS ⇒ (∀i · i ∈ dom(train_wagons(train)) ∧ i ≠ 0 ⇒ (i−1) ∈ dom(train_wagons(train)))
             not theorem
    axm6:   ∀t1, t2 ·((t1 ∈ TRAINS)∧(t2∈TRAINS)∧(t1≠t2))⇒(ran(train_wagons(t1))∩ran(train_wagons(t2))=∅) not theorem
    axm9:   ∀train·card(train_wagons(train))>3 not theorem
END
```

Figure 15: Contexte wagon

Pour implémenter cela ,on a commencé par définir le contexte plus wagons. le but de cette partie est de respecter certaines exigences:

- Chaque train a une séquence de wagons ordonnée.
-Chaque train a au moins un wagon.
- Un wagon ne peut pas appartenir à plus qu'un train.

- L'indexation des wagons est continu et cela est fait en associant à chaque train une séquence (N , WAGONS) de wagons .On aura donc par la suite un ordre et une continuité des wagons pour chaque train .

Figure 16: Mch Topologie2

Dans cette figure ,on voit que la Mch Topologie2,raffine Mch Topologie1. Pour l'événement affecter Route,il n'y a pas eu de changements,vu que l'attribution d'une route est une étape essentielle et le fait d'ajouter des wagons ne changera pas la logique de l'affectation des routes et l'attribution des destinations .

Pour le début de franchissement,on avait ajouté deux gardes:

garde 8:le booléen finFrPossible doit être nul,car on peut pas avoir une fin de franchissement possible dans le début de franchissement.

garde 9:l'indice de passage du train doit être strictement inférieur au cardinal du train wagons(train).

Après cela on a ajouté deux nouvelles actions:

act3 :indice de passage doit être incrémenté.

act4 :Le booléen finFrPossible prend la valeur true.

```
DEBUT_FRANCHISSEMENT:  extended ordinary ›
REFINES
◦    DEBUT_FRANCHISSEMENT
ANY
◦   train    ›
◦   tr1  ›
◦   tr2  ›
WHERE
◦   grd1:   train ∈ dom(route) not theorem ›
◦   grd2:   train_position(train) = {tr1} not theorem ›
◦   grd3:   tr1 ↦ tr2 ∈ Reseau not theorem ›
◦   grd4:   ∀other_train ·(other_train∈TRAINS∧ tr2 ∉ train_position(other_train)) not theorem ›
◦   grd6:   (1 ↦ tr1) ∈ route(train) not theorem ›
◦   grd5:   (2 ↦ tr2) ∈ route(train) not theorem ›
◦   grd8:   finFr_possible(train)=0 not theorem ›
◦   grd9:   indice_de_passage(train)<card(ran(train_wagons(train))) not theorem ›
THEN
◦   act1:   train_position(train) ≔ {tr1, tr2} ›
◦   act2:   sens_deplacement(train) ≔ (tr1↦ tr2) ›
◦   act4:   finFr_possible(train)≔1 ›
◦   act3:   indice_de_passage(train)≔indice_de_passage(train)+1 ›
END

FIN_FRANCHISSEMENT_WAGON:  not extended ordinary ›
ANY
◦   train    ›
◦   tr1  ›
◦   tr2  ›
WHERE
◦   grd1:   train ∈ dom(route) not theorem ›
◦   grd2:   train_position(train) = {tr1, tr2} not theorem ›
◦   grd3:   (2 ↦ tr2)∈ route(train) not theorem ›
◦   grd4:   sens_deplacement(train) = (tr1↦ tr2) not theorem ›
◦   grd5:   train_position(train) = {tr1, tr2} not theorem ›
◦   grd6:   sens_deplacement(train) = (tr1↦ tr2) not theorem ›
◦   grd8:   finFr_possible(train)=1 not theorem ›
◦   grd9:   indice_de_passage(train) < card(ran(train_wagons(train))) not theorem ›
THEN
◦   act1:   finFr_possible(train)≔0 ›
END
```

Figure 17: Mch Topologie2 (suite 1)

Concernant l'événement Fin Franchissement Wagon ,il représente la fin du franchissement d'un wagon ,les gardes à respecter sont les suivantes:

- grd1: le train doit avoir une route .
- grd2: le train est positioné sur deux tronçons.
- grd4: le train se déplace d'un sens tr1 vers tr2.
- grd8 :Vérifier que le franchissement est possible.

Si tous les gards sont valides on pourra passer à act1 qui consiste à remettre le booléen finFrpossible à 0.

```
FIN_FRANCHISSEMENT_TRAIN:  not extended ordinary
REFINES
    FIN_FRANCHISSEMENT
ANY
   train
   tr1
   tr2
WHERE
   grd5:   train ∈ dom(route) not theorem
   grd6:   train_position(train) = {tr1, tr2} not theorem
   grd7:   (2 ↦ tr2)∈ route(train) not theorem
   grd1:   sens_deplacement(train) = (tr1↦ tr2) not theorem
   grd3:   finFr_possible(train)=1 not theorem
   grd4:   train_position(train) = {tr1, tr2} not theorem
   grd8:   indice_de_passage(train) = card(ran(train_wagons(train))) not theorem
THEN
   act1:   train_position(train) ≔ {tr2}
   act2:   sens_deplacement≔{train}⩤sens_deplacement
   act3:   route(train) ≔ {i − 1 ↦ tr | (i ↦ tr) ∈ route(train) ∧ i > 1}
   act4:   indice_de_passage(train)≔0
END

END
```

Figure 18: Mch Topologie2 (suite 2)

L'événement FIN FRANCHISSEMENT TRAIN est un raffinement de FIN FRANCHISSEMENt TRAIN. On vérifie d'abord que l'indice du passage est égale au cardinal des wagons du train , donc on s'assure que tous les wagons sont passés. Ainsi le train prend une nouvelle position,l'indice du passage est réinitialisé à 0,et on retire le train du sens de déplacement.

4.1 Validation et vérification

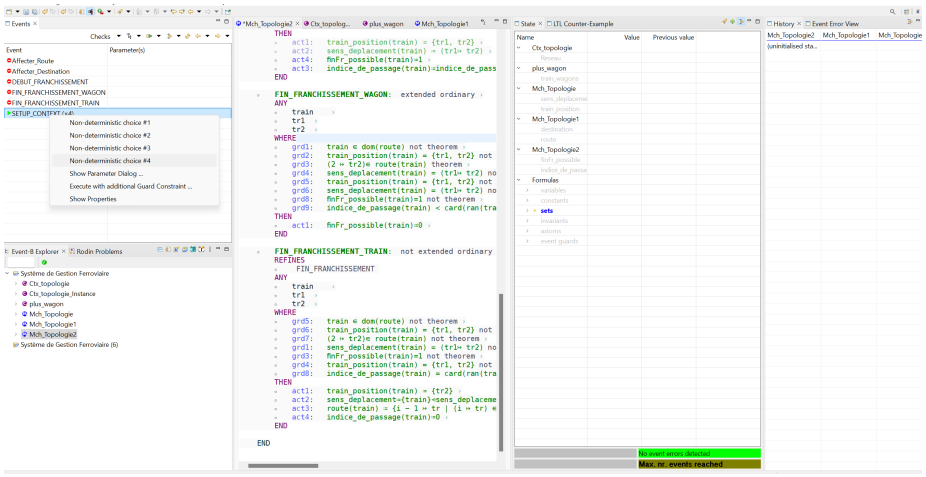

Figure 19: Initialisation

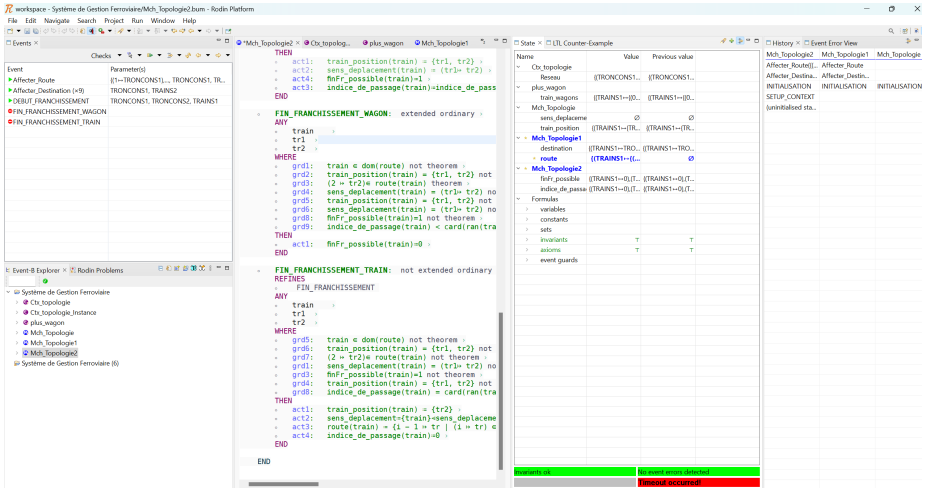

Figure 20: Affecter route

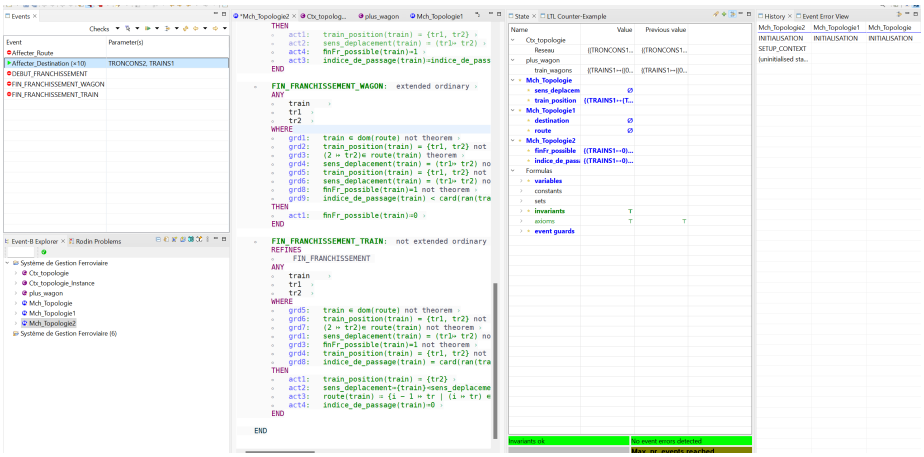

Figure 21: Affecter destination

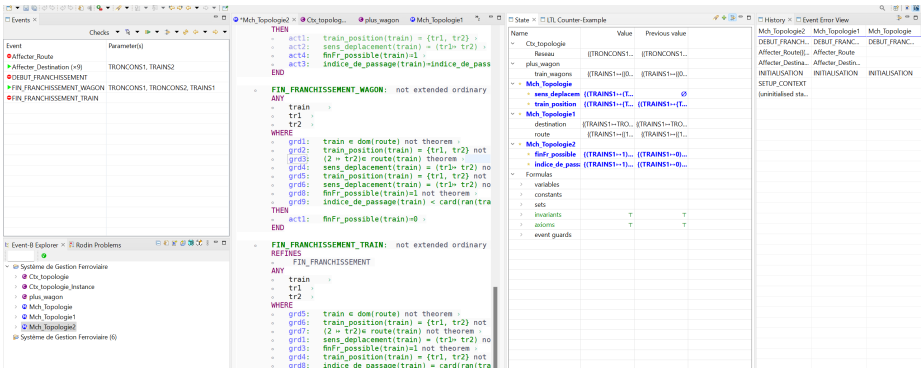

Figure 22: Fin franchissement wagon

On voit bien que notre modèle respecte les exigences définies dans le cahier des charges. Les évenements respectent bien les invariants et l'évolution des variables est comforme aux résultats attendues

5 Couche 3: Physique et Dynamique

5.1 contexte

Dans cette dernière partie ,chaque train est représenté par sa position,sa vitesse et son accélération. Une nouvelle notion a été introduite dans cette section c'est l'autorité de mouvement qui limite le mouvement du train.Chaque train passe par trois modes :mouvement libre,freinage et attente.

```
CONTEXT
    Dynamique   ›
EXTENDS
    plus_wagon
SETS
    Modes     ›
CONSTANTS
    longueur_wagon   not symbolic ›
    longueur_tronçon     not symbolic ›
    Mouvement_libre  not symbolic ›
    Freinage     not symbolic ›
    Attente  not symbolic ›
    DeltaT   not symbolic ›
AXIOMS
    axm1:   Modes  ={Mouvement_libre, Freinage,Attente} not theorem ›
    axm3:   longueur_tronçon=100 not theorem ›
    axm4:   longueur_wagon=30 not theorem ›
    axm5:   DeltaT=1 not theorem ›
END
```

Figure 23: Contexte dynamique

On a commencé par définir un nouveau contexte Dynamique . Les modes de trains sont Mouvement libre,Freinage,Attente, Pour modéliser les caractéristiques physiques des mouvements on a défini des nouvelles constantes: longueur tronçon qui prend la valeur 100 et longueur wagon de valeur 30 et un detlaT qui représente une unité de temps , ces valeur peuvent être adaptées pour simuler la réalité. Ce contexte modélise le comportement dynamique du train en prenant le temps et les distances comme nouveaux paramètres dans la conception .

5.2 Machine

Dans cette machine on a ajouté plusieurs variables pour décrire l'état physique du train .

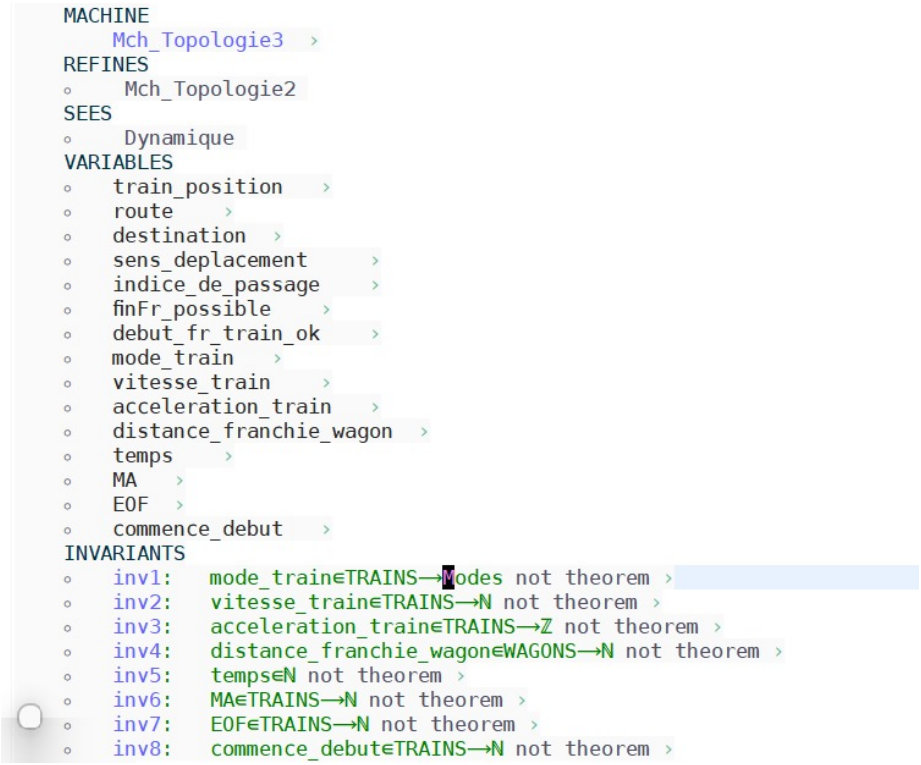

Figure 24: Contexte dynamique

- mode_train : fonction qui donne pour chaque train son mode (accélère , freine ou en attente)
- vitesse_train:associe à chaque train une vitesse , initialement nulle.
- acceleration_train:associe à chaque train une accélération négative ou positive selon est ce qu'il freine ou accélère.
- distance_franchie_wagon: cette fonction associe à chaque wagon la longueur de sa partie qui a franchit le jonction entre un tronçon et l'autre .
- MA ,EOF: l'autorité de mouvement , même si pas explicité mais c'est conçu dans les évènements puisque un train ne peux pas entrer dans un tronçons tant qu'il y'a un autre train qui l'occupe
- commence_debut : ce booléan spécifie le fait que le train a décidé de bouger pour permettre de faire quelques évènement (on développe plus bas)

```
DEBUT_FRANCHISSEMENT_WAGON:   extended ordinary
REFINES
    DEBUT_FRANCHISSEMENT_WAGON
ANY
   train
   tr1
   tr2
   wagonPrecedant
WHERE
   grd1:   train ∈ dom(route) not theorem
   grd2:   tr1 ↦ tr2 ∈ Reseau not theorem
   grd3:   ∀other_train ·(other_train∈TRAINS∧ tr2 ∉ train_position(other_train)) not theorem
   grd4:   (1 ↦ tr1) ∈ route(train) not theorem
   grd6:   finFr_possible(train)=0 not theorem
   grd7:   indice_de_passage(train)<card(ran(train_wagons(train))) not theorem
   grd8:   debut_fr_train_ok(train)=1 not theorem
   grd9:   (indice_de_passage(train) = 0 ⇒ wagonPrecedant = train_wagons(train)(0))
           ∧
           (¬(indice_de_passage(train) = 0)) ⇒ wagonPrecedant = train_wagons(train)(indice_de_passage(train) − 1) not theorem
THEN
   act1:   finFr_possible(train):=1
   act2:   indice_de_passage(train):=indice_de_passage(train)+1
   act3:   commence_debut(train):=1
   act4:   distance_franchie_wagon(wagonPrecedant):=0
END
```

Figure 25: Debut Franchissement Wagon

- grd9 : check l'ordre de passage des wagons pour remettre le calculateur de longueur de la partie qui a franchit au zero
- act3: spécifie que le train va commancer à franchir donc les wagons vont commencer à franchir aussi , ce qui nous permet à frener accelerer ou mettre à jour
- act4: remet le compteur de longueur de la partie qui a franchit du wagon (wagon a déja franchit totalement) à zéro , pour préparer au nouvel franchissement potentiel

```
Accelerer:  not extended ordinary
ANY
   train
   wagon
WHERE
   grd1:   (train_wagons(train))(indice_de_passage(train) − 1) =wagon  not theorem
   grd2:   vitesse_train(train)<Vmax not theorem
   grd3:   distance_franchie_wagon(wagon)≤(longueur_wagon − (vitesse_train(train)*DeltaT)) not theorem
   grd4:   commence_debut(train)=1 not theorem
THEN
   act1:   vitesse_train(train):=vitesse_train(train)+acceleration*DeltaT
   act2:   distance_franchie_wagon(wagon):=distance_franchie_wagon(wagon)+vitesse_train(train)*DeltaT
   act3:   mode_train(train):=Mouvement_libre
   act4:   acceleration_train(train):=acceleration
END
```

Figure 26: Accélération

Accélèrer veille a incrémenter la longueur de la partie de wagon qui est entrain de franchir et aussi à incrémenter la vitesse , en usilsant l'accélèration et le pas temporel DeltaT.
La vitesse doit rester inférieur à une vitesse max Vmax.
Le train est donc en mode libre.

```
freiner:    not extended ordinary
ANY
  train    
  wagon    
WHERE
  grd1:   train∈TRAINS not theorem
  grd3:   (train_wagons(train))(indice_de_passage(train) − 1) =wagon  not theorem
  grd2:   vitesse_train(train)>0 not theorem
  grd4:   distance_franchie_wagon(wagon)≤(longueur_wagon − (vitesse_train(train)*DeltaT)) not theorem
  grd5:   commence_debut(train)=1 not theorem
THEN
  act1:   vitesse_train(train)=vitesse_train(train)−acceleration*DeltaT
  act2:   distance_franchie_wagon(wagon)=distance_franchie_wagon(wagon)+vitesse_train(train)*DeltaT
  act3:   mode_train(train)=Freinage
  act4:   acceleration_train(train)=−acceleration
END
```

Figure 27: Freinage

Freiner faits la même chose sauf qu'elle décrémente la vitesse , avec l garde que ça doit être toujours positive
le train est donc en freinage

```
FIN_FRANCHISSEMENT_WAGON:  extended ordinary
REFINES
  FIN_FRANCHISSEMENT_WAGON
ANY
  train    
  tr1    
  tr2    
  wagon    
  wagonNext    
WHERE
  grd1:   train ∈ dom(route) not theorem
  grd2:   train_position(train) = {tr1, tr2} not theorem
  grd3:   (2 ↦ tr2)∈ route(train) not theorem
  grd4:   sens_deplacement(train) = (tr1↦ tr2) not theorem
  grd5:   train_position(train) = {tr1, tr2} not theorem
  grd6:   sens_deplacement(train) = (tr1↦ tr2) not theorem
  grd8:   finFr_possible(train)=1 not theorem
  grd9:   indice_de_passage(train) < card(ran(train_wagons(train))) not theorem
  grd12:  wagon∈WAGONS not theorem
  grd10:  distance_franchie_wagon(wagon)>(longueur_wagon − (vitesse_train(train)*DeltaT)) not theorem
  grd11:  (train_wagons(train))(indice_de_passage(train) − 1) = wagon not theorem
  grd13:  wagonNext=(train_wagons(train))(indice_de_passage(train)) not theorem
  grd14:  distance_franchie_wagon(wagonNext)=0 not theorem
THEN
  act1:   finFr_possible(train)=0
  act3:   distance_franchie_wagon(wagonNext) = ((vitesse_train(train) * DeltaT) − (longueur_wagon − distance_franchie_wagon(wagon)))
END
```

Figure 28: Fin franchissement

fin franchissement wagon mets a jour la longueur de la partie du wagons suivant qui va franchir , si la longueur qui reste du wagon actuelle est plus petite que le pas vitesse*DeltaT

```
MISE_A_JOUR:    not extended ordinary
ANY
  train    
  wagon    
WHERE
  grd3:   (train_wagons(train))(indice_de_passage(train) − 1) =wagon  not theorem
  grd1:   commence_debut(train)=1 not theorem
  grd2:   distance_franchie_wagon(wagon)≤(longueur_wagon − (vitesse_train(train)*DeltaT)) not theorem
THEN
  act1:   distance_franchie_wagon(wagon)=distance_franchie_wagon(wagon)+vitesse_train(train)*DeltaT
END
```

Figure 29: Mise à jour

Mise à jour faits une mise à jour des variables , justement comme freiner ou accélérer mais sans freiner ni accélérer , elle garde la vitesse constante

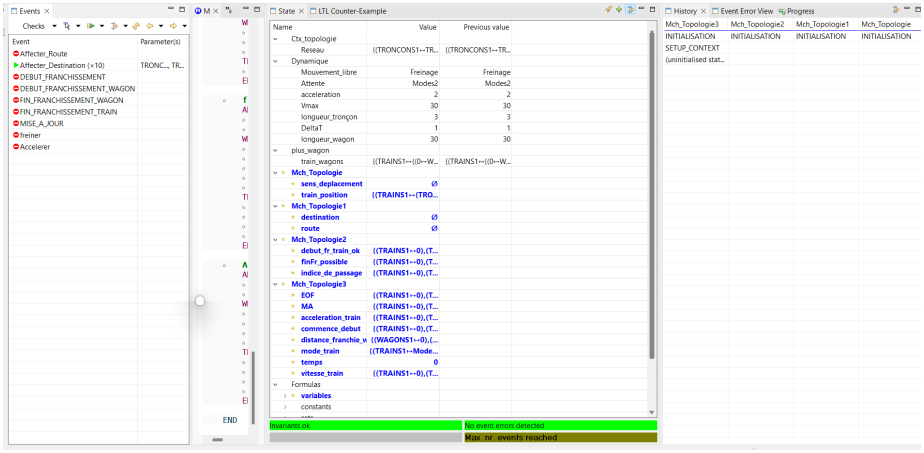

Figure 30: Partie 1

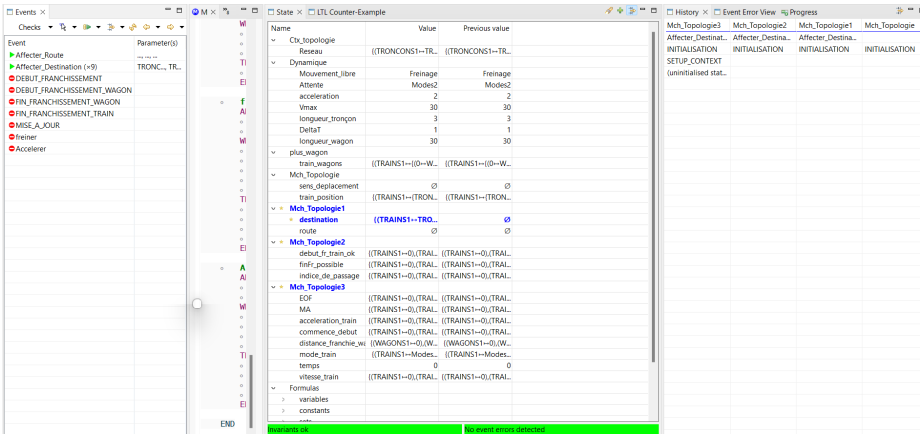

Figure 31: Partie 2

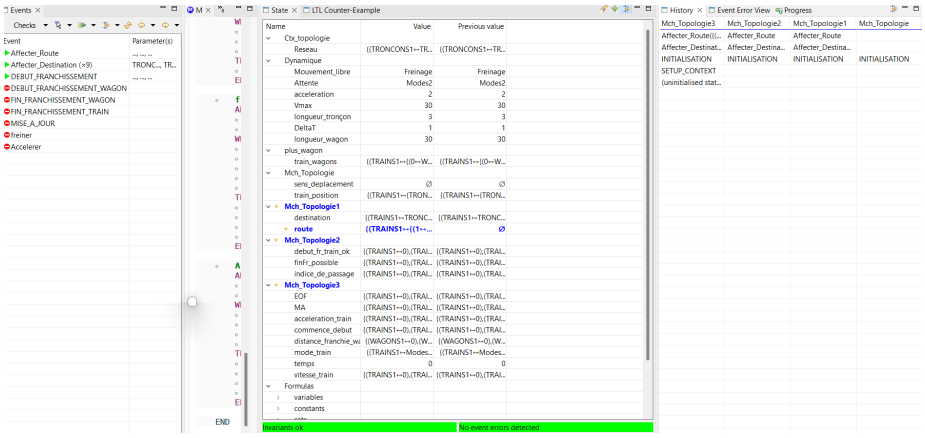

Figure 32: Partie 3

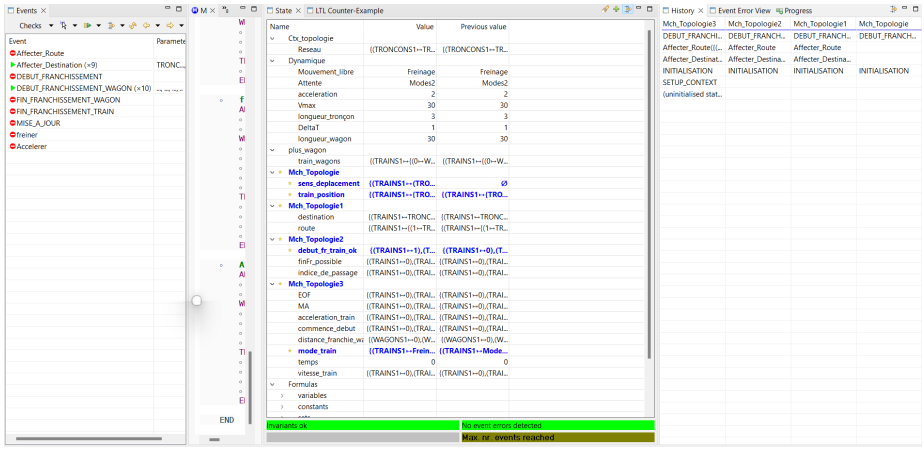

Figure 33: partie 4

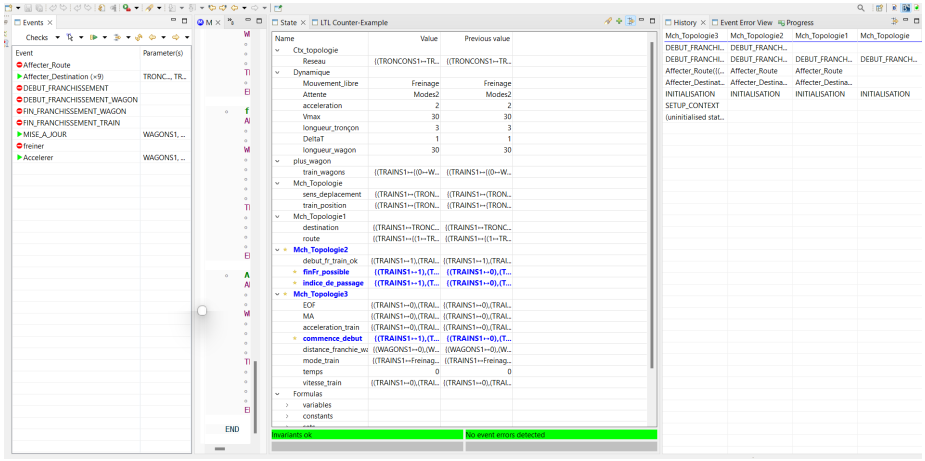

Figure 34: partie 5

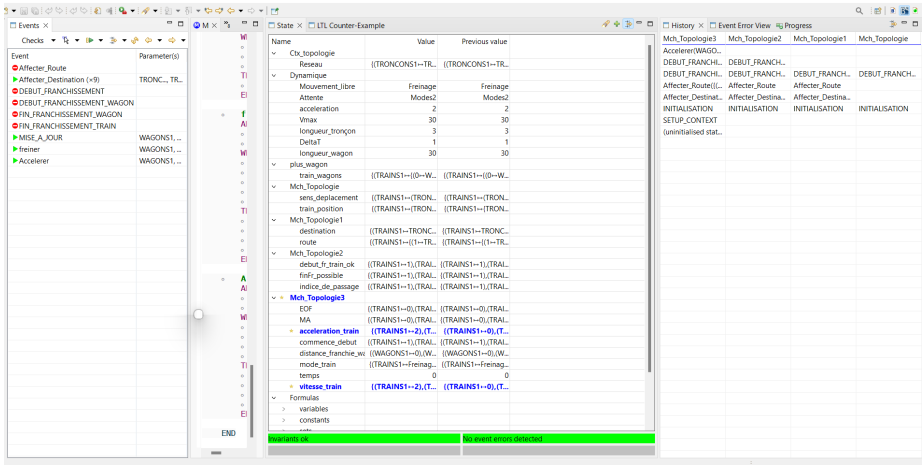

Figure 35: partie 6

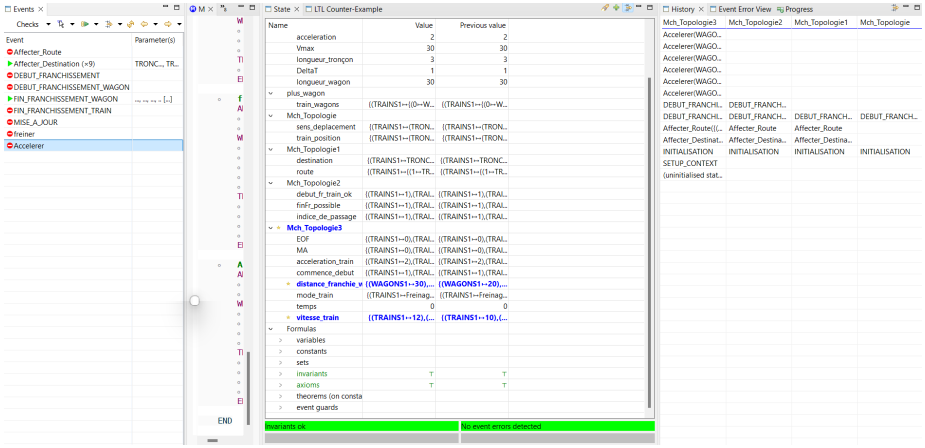

Figure 36: partie 7

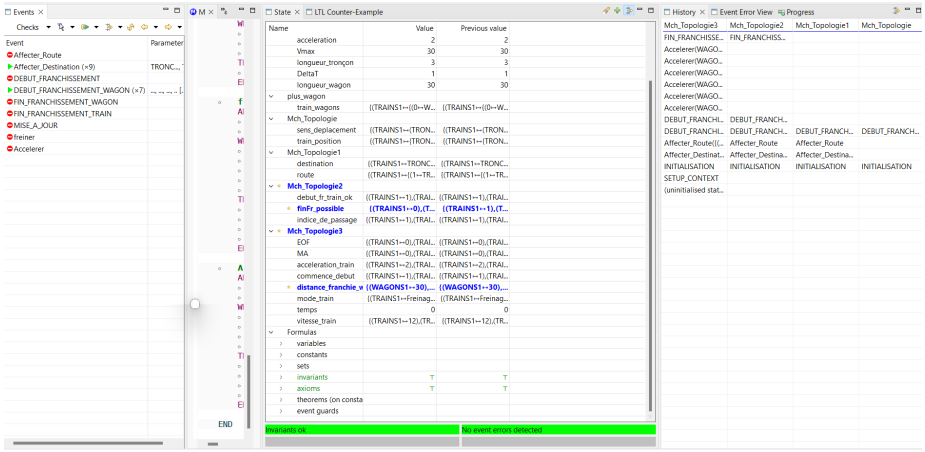

Figure 37: partie 8

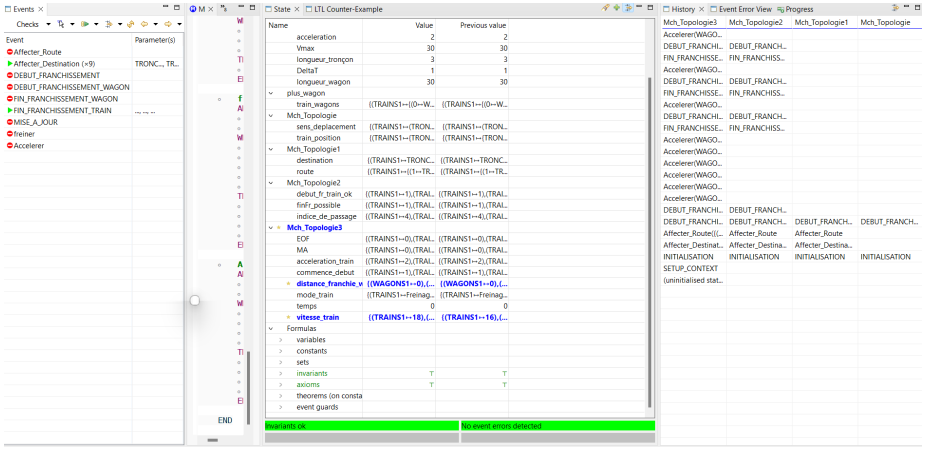

Figure 38: partie 9

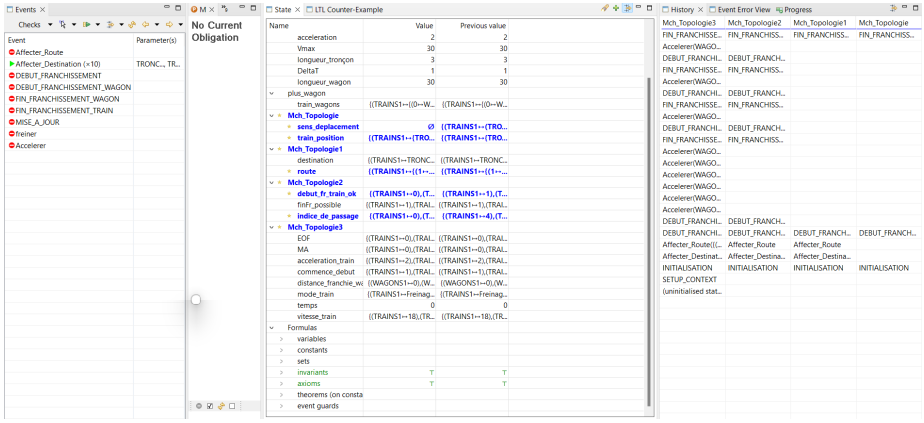

Figure 39: partie 10

On voit dans notre modèle que les exigences sont bien respectées. EN accélérant, on voit que la vitesse ne peut pas dépasser la vitesse seuil et on ne peut pas freiner dans le cas où la vitesse est nulle . On commence toujours par affecter une destination , puis une route , puis le train décide de commencer de franchir , ses wagons eux aussi , et donc on peut commencer d'accélerer et freiner , et mettre a jour jusqu'à ce que tous les wagons passent et on peut finalement annoncer le fin de franchissement du train. continuer comme ça jusqu'atteindre la destiantion

6 Conclusion

Ce projet était une bonne opportunité pour appliquer les connaissances vues en cours. On a passé par différentes étapes de conception, c'était pas facile au début de réaliser un tel modèle mais en avançant les choses sont devenues plus claires.Le raffinement nous a permis de diviser le problème en sous-problèmes plus faciles à gérer. Y.M

6.1 Nahi Mohammed

Dans ce projet, j'ai travaillé sur la première couche, qui concerne la topologie et la signalisation des trains. Mon rôle était de modéliser les éléments clés du réseau ferroviaire, notamment la gestion des segments de voie, la position des trains et les itinéraires tout en garantissant les contraintes de sécurité pour éviter les collisions. La plupart des fonctionnalités concernant cette couche ont été couvertes par la modélisation, comme la structuration des itinéraires et des mouvements des trains et la réalisation d'affinements progressifs qui soutiennent l'évolution d'un modèle. J'ai également prouvé formellement plusieurs propriétés importantes pour assurer la cohérence de ce système. La fonctionnalité qui restait à implémenter complètement concernait les changements dynamiques d'un itinéraire de train lorsqu'il est en route. Au final, ce projet m'a enrichi : d'abord, il m'a permis de comprendre les méthodes formelles et d'être plus rigoureux pour la modélisation d'un système complexe.

6.2 bilan :M'naoui Youssef

Dans ce projet, j'avais travaillé sur la deuxième couche, j'ai donc introduit la notion des wagons.L'ajout de wagon était une partie importante car cela permet de rendre notre modèle plus réaliste, je pense que j'ai réussi à bien définir les événements associés à cette partie. L'indexation des wagons était l'élément clé pour assurer la continuité et le bon fonctionnement du modèle.Des variables booléen ont été introduite pour assurer le bon fonctionnement du modèle. Pour les points faibles de cette partie, je pense que la définition ds éventements ajout et retrait de wagons à la destination auraient étaient une bonne chose donnant à un aspect beaucoup plus dynamique au modèle.

6.3 bilan :Missouri Taha

Dans cette partie ,j'ai essayé de voir le modèle d'une façon différente (d'un point de vue temporelle et physique)de ce qui précède. J'avais introduit la notion de vitesse et d'accélèration.j'ai ajouté aussi les modes de fonctionnement (mouvement libre, freinage et attente). La partie trois me

semble la plus difficile des trois partie , ayant déjà contribué au deux partie précédente .
L'idée la plus intéressante est la logique d'attribuer des longueur au tronçons et wagons , et gérer le franchissement de chaque wagon avec une variable propre à lui et au train auquel il appartient , et aussi gérer les événement de débordement (quand la longueur restante dans le premier tronçons et plus petite que la distance parcouru pendant un deltaT) , et aussi l'idée de faire trois événement , un de mise à jour , un de freinage et le dernier d'accélération , et l'enchaînement des événements qui fait que tout marche comme il faut en harmonie , sans avoir des évènement en conflits