
Mémoire de Master

# *DSL pour le model-checking d'un réseau de transports publics par voie ferrée*

---

par
Marius NGUENA KENGNI
(*nguenak3@etu.unige.ch*)

Superviseur:
Professeur Didier BUCHS
(*Didier.Buchs@unige.ch*)

Co-Superviseurs:
M. Steve HOSTETTLER et M. Alexis MARECHAL
(*Steve.Hostettler@unige.ch, Alexis.Marechal@unige.ch*)

29 septembre 2009

.

A Tim Jordan,

A Carine,

A toute ma famille

# *Remerciements*

Je tiens en premier lieu à remercier le bon DIEU de nous avoir offert tout ce que nous possédons.

Je remercie vivement toutes les personnes qui ont contribué au bon déroulement de ce travail.

En particulier, j'exprime ma gratitude à mon encadreur M. Didier BUCHS, professeur au CUI (Centre Universitaire d'informatique) de l'Université de Genève, pour avoir suivi de très près l'évolution du mémoire. Ses conseils avisés m'ont donné la possibilité de dépasser des longs moments d'hésitation et surtout, de recadrer mon énergie afin d'obtenir de meilleurs résultats. C'est grâce à lui que j'ai retiré autant de satisfaction tout au long de ce projet.

Un grand merci à toute l'équipe SMV (Software Modeling and Verification) de l'université de Genève, en particulier à M. Steve HOSTETTLER et à M. Alexis MARECHAL pour avoir largement contribué, à l'approfondissement de mes connaissances techniques, dans le domaine ô combien complexe du Software Engineering.

Je remercie toute ma famille pour ses encouragements.

Je ne saurais clore ce chapitre sur les remerciements sans mentionner deux personnes qui partagent ma vie au quotidien et qui me sont chères. Je dédie ce mémoire à :

- Mon petit Tim Jordan, qui est arrivé au monde pendant la réalisation de ce travail.

- Ma très chère et douce Carine, qui a fait preuve de beaucoup de patience durant ces sept longs mois ou je n'étais qu'un fantôme à la maison.

# *Résumé*

*« DSL pour le model-checking d'un réseau de transport public par voie ferrée ».*

Dans ce mémoire, nous présentons un DSL (Domain Specific Language) ou *Langage dedié* en français, dont la vocation est de fournir un langage de modélisation efficace, pour faire du model-checking. Le système que nous analysons est un réseau de transport public par trams.

Le DSL que nous développons est appliqué au contexte des réseaux de transport public par trams de la ville de Genève en SUISSE, nous l'avons baptisé "***TramLang***", qui signifie *Langage de Trams*. Le langage ***TramLang*** défini par sa grammaire et son métamodèle, est transposable à d'autres réseaux de transport public par voie ferrée et permet de créer des modèles de réseaux de trams sur lesquels vont être appliqués des techniques de model-checking. Il s'agit de vérifier automatiquement si un modèle donné, le système lui même ou une abstraction du système satisfait un ensemble de propriétés, souvent formulées en terme de logique mathématique .

**Mots clé :** Model-checking, DSL, MDA, métamodélisation, syntaxe abstraite, syntaxe concrète, sémantique d'une transformation, sémantique d'un langage, sémantique opérationnelle, Réutilisabilité, Modularité.

# *Abstract*

*« A DSL for the model-checking of public transport network by rail system ».*

In this paper, we present a DSL (Domain Specific Language) whose purpose is to provide an effective modeling language to make model-checking. The system we analyze is a network of public transport by trams.

DSL we develop is applied to the context of public transport by tram from the city of Geneva in Switzerland, we have called "**TramLang**", which means *language of trams*. The language **TramLang** defined by its grammar and its metamodel is transferable to other public transport by rail, and allows you to create network models of trams that will be applied the techniques of model-checking. The aim is to check automatically whether a given model, the system itself or an abstraction of the system satisfies a set of properties, often formulated in terms mathematical logic.

**Keywords :** Model-checking, DSL, MDA, metamodelling, abstract syntax, concrete syntax, semantic processing, semantic of a language, operationnal semantic, Reusability, Modularity.

# Table des matières

*Remerciements* **2**

*Résumé* **3**

*Abstract* **4**

*Introduction* **11**

**1 TPG : Transports Publics Genevois** **14**
1.1 Historique . . . . . 14
1.2 Statistiques . . . . . 15

**2 Modélisation d'un réseau de transport public par trams et transformation en RdP** **18**
2.1 Les Réseaux de Petri . . . . . 20
2.1.1 Les composants élémentaires . . . . . 20
2.1.2 Places, Transitions, Arcs et jetons . . . . . 21
2.1.3 Un Exemple de RdP . . . . . 23
2.2 Les Extensions des réseaux de Petri . . . . . 23
2.2.1 Réseaux de Pétri Colorés(RdpC) . . . . . 24
*un exemple de RdpC* . . . . . 25
2.2.2 Réseaux de Pétri Temporisés (RdpT) . . . . . 26
2.2.3 Réseaux de Pétri Colorés Temporisés (TCPN) . . . . . 28
*un exemple de TCPN* . . . . . 29
2.2.4 Réseaux de Pétri Algébriques(APN) . . . . . 31
*un exemple d'APN* . . . . . 33
2.3 Modélisation d'une maquette de ***TramwayNET*** . . . . . 35
2.3.1 Description de modèles Physiques de TramwayNET . . 35
2.3.2 Décomposition des entités (TrackElement) de TramwayNET . . . . . 36
2.4 Transformation d'une maquette de ***TramwayNET*** en RdP . 41

**3 Model-checking** **43**
3.1 Model-checking : Etat de l'art . . . . . 44
3.2 Spécification formelle des besoins d'un TramwayNET . . . . . 46
3.2.1 Expression des propriétés en langage naturel . . . . . 49
3.2.2 Expression des propriétés en logique mathématique . . 50
3.3 Validation d'un modèle de TramwayNET . . . . . 51
3.3.1 Application du Model-checker CPNtools . . . . . 52
3.3.1.1 Modélisation . . . . . 53
3.3.1.2 Expérimentation . . . . . 56
3.3.2 Application du Model-checker HELENA . . . . . 58
3.3.2.1 Modélisation . . . . . 58
3.3.2.2 Expérimentation . . . . . 58
3.3.3 CPNtools vs HELENA . . . . . 61

**4 MDA et DSL** **67**
4.1 MDA : Etat de l'art . . . . . 67
4.1.1 Principe du MDA (d'après l'OMG) . . . . . 67
4.1.2 Architecture du MDA . . . . . 70
4.1.3 Standards de l'OMG . . . . . 71
4.2 DSL : Etat de l'art . . . . . 72
4.2.1 Avantages des DSLs . . . . . 73
4.2.2 Inconvénients des DSLs . . . . . 74
4.2.3 Exemples de DSL . . . . . 74

**5 Conception du DSL textuel (TramLang)** **75**
5.1 Overview conception du DSL . . . . . 75
5.2 Syntaxe abstraite du DSL . . . . . 78
5.2.1 métamodélisation : Etat de l'art . . . . . 78
5.2.2 "TramwayMM" : Metamodèle du réseau de trams . . . 79
5.2.3 "apnMM" : Metamodèle des réseaux de Petri algébriques 85
5.2.4 "timedAPNMM" : Metamodèle des réseaux de Petri algébriques temporisés . . . . . 86
5.3 Syntaxe concrète du DSL . . . . . 88
5.4 Sémantique du DSL . . . . . 90
5.4.1 Etape 1 : Transformation ATL de TramwayMM vers timedAPNMM . . . . . 90
5.4.2 Etape 2 : Transformation ATL de timedAPNMM vers apnMM . . . . . 93
5.4.3 Etape 3 : Transformation Java de apnMM vers HELENA 94

**6 Implémentation du DSL textuel (TramLang) 95**
6.1 Implémentation de la syntaxe abstraite . . . . . 95
6.2 Implémentation de la syntaxe concrète . . . . . 96
6.3 Implémentation de la sémantique . . . . . 98
6.4 DSL et Support d'implémentation . . . . . 100

**7 Conclusion et Perspectives 102**

***Acronymes*** **106**

# Table des figures


1.1 un tram devant le site des TPG au Bachet-De-Pesay à Genève 14
1.2 Réseau des tramways TPG avec ses principales stations jusqu'au 14 décembre 2008 . . . . . 17

2.1 Propriétés des RdPs . . . . . 19
2.2 Exemple de graphe non orienté(A gauche) et graphe orienté (A droite) . . . . . 21
2.3 Un Rdp avant et après le tir d'une transition . . . . . 24
2.4 Un exemple de RdpC : Liste de nombres pairs et impairs entre n=1 et k . . . . . 25
2.5 Processus d'écoulement du temps sous CPNTools . . . . . 30
2.6 ADT simplifié des naturels . . . . . 32
2.7 exemple d'APN : Paiement et Pourboire . . . . . 34
2.8 Une partie du tramway de la ville de Genève partagée par cinq lignes de trams : Le carrefour de Plainpalais . . . . . 35
2.9 Les Elements de base("TrackElement") qui composent un tramwayNET . . . . . 36
2.10 Les Elements de base("TrackElement") avec leurs ports . . . . 37
2.11 Processus de décomposition d'un TramwayNET . . . . . 38
2.12 [A gauche] Une "grande jonction" à Toronto(Canada), entre les lignes Spadina et Queen, [A droite] la maquette de cette "grande jonction" . . . . . 39
2.13 Un TrackElement **[C/1,3]** : Croisement 1 entrée et 3 sorties . 39
2.14 [A gauche] Une jonction de lignes de tram à la gare Cornavin de Genève, Photo prise le Jeudi 20 Août 2009, [A droite] Superposition de deux TrackElements **[C/2,1] et [C/1,1]** . . 40
2.15 [A gauche] Un croisement de rails de trams à Leidsestraat, Amsterdam, [A droite] Un TrackElement **[C/1,2]** : Croisement 1 entrée et 2 sorties . . . . . 40
2.16 Un TrackElement **[C/3,3]** : Croisement 3 entrées et 3 sorties . 41
2.17 Transformation d'un TramwayNET en Rdp . . . . . 42

3.1 Processus général de model-checking . . . . . 45
3.2 Table horaire Ligne 12, Direction "Palette" ⟶ "Moillesulaz" 48
3.3 Scénario Propriété securité : Collision entre deux trams . . . . 49
3.4 Scénario Propriété régularité : Retard de tram à un arrêt . . . 49
3.5 maquette de tramwayNET ("***SIMPLETramway***") pour l'application du model-checking et son RdP équivalent . . . . . 51
3.6 "***SIMPLETramway***" dans CPNTools : Declaration des variables et des couleurs . . . . . 54
3.7 "***SIMPLETramway***" dans CPNTools : La page "d'initialisation" . . . . . 54
3.8 "***SIMPLETramway***" dans CPNTools : La page du "traffic Ligne12" . . . . . 55
3.9 "***SIMPLETramway***" dans CPNTools : La page du "traffic Ligne13" . . . . . 55
3.10 "***SIMPLETramway***" : Résultat exécution des requêtes en ML sous CPNTools . . . . . 57
3.11 Le language de description d'Helena : Exemple . . . . . 59
3.12 "***SIMPLETramway***" dans HELENA . . . . . 60
3.13 Modélisation de l'écoulement du temps sous HELENA : Deadlock . . . . . 64
3.14 Modélisation de l'écoulement du temps sous HELENA : Pas de Deadlock . . . . . 65

4.1 Transformation de modèles du MDA . . . . . 69
4.2 Architecture du MDA . . . . . 71
4.3 Standard OMG : Les quatres couches de l'architecture MDA . 72

5.1 Chaine des transformations sémantiques . . . . . 76
5.2 Modèles utilisés pour la transformation de TramwayNET vers RdP au format HELENA . . . . . 77
5.3 Metamodèle de TramwayNET . . . . . 79
5.4 Les métarelations entre un modèle de TramwayNET et son métamodèle . . . . . 80
5.5 Métamodèle pour les APN(carré bleu) et Métamodèle pour la représentation des ADT des APN (carré rouge) . . . . . 87
5.6 Les quatres inscriptions possibles sur une transition dans un TCPN . . . . . 88
5.7 Métamodèle pour les timedAPN(carré bleu) et Métamodèle pour la représentation des ADT des timedAPN (carré rouge) . 89
5.8 Un exemple de règle ATL . . . . . 91

6.1 Grammaire du DSL TramLang avec TMFxtext . . . . . 97
6.2 Liens entre les outils et les formats de données : EMF, ATL , XMI, Ecore, TMFXtext . . . . . 101

# *Introduction*

L'utilisation croissante des systèmes informatiques automatisés, dans tous les domaines de notre vie, nécessite une garantie maximale de leurs bons fonctionnements et de leurs performances. Ceci est particulièrement vrai pour les applications critiques (de contrôle, de commande, embarquées, etc...) qui entraînent parfois des pertes humaines ou financières. Aussi la vérification automatique de ses applications est naturellement importante. Aujourd'hui, une part importante des efforts fournis dans le domaine du test et vérification de systèmes informatiques est consacrée à l'activité de *validation* d'un modèle du système communément appelé *model-checking*. La valeur ajoutée du model-checking est maintenant reconnue dans les milieux universitaires. Ces dernières années ont vu naître de nombreuses méthodes de génération de tests basées sur les techniques de vérifications formelles.

Les données à fournir à un *model-checker*[I], sont la description du modèle à analyser, donnée par un langage de modélisation et la propriété à vérifier, exprimée par un langage de spécifications. Il existe plusieurs langages de modélisation dont le champ d'application est général, parmis lesquels :

- EXPRESS (ISO 10303-11) : Standard international de langage de modélisation de données, utilisé entre autre, pour modéliser différents modèles de données du standard ISO.

- UML (*Unified Modeling Langage*) langage graphique de modélisation de données et de leur traitement.

- etc...

Il existe aussi plusieurs langages de modélisation spécifiques à un domaine particulier, fournissant des abstractions et des notations appropriées comme construction de base : Ce sont les DSL (*Domain Specific Language*) ou *Lan-*

I. Outil de model-checking

*gages dédiés* en français. Ils permettent aux utilisateurs de se concentrer sur leur métier, en manipulant un formalisme spécifique à leur activité.

Dans ce contexte, pour combiner les avantages des DSLs et des model-checker, nous présentons un DSL spécifique aux systèmes de réseaux de transport public par trams. L'utilisation de ce DSL via sa syntaxe abstraite, sa syntaxe concrète et sa sémantique, permettra la spécification et la validation des modèles de réseaux de transport public par trams.

Le présent mémoire est constitué de huit chapitres :

- Dans le **premier chapitre** 1, nous présentons brièvement l'entreprise qui exploite le réseau de transport par trams de la ville de Genève dont le nom est **TPG** [II]. Nous donnons également quelques statistiques et chiffres clés sur son activité.

- Le **second chapitre** 2 introduit le formalisme des réseaux de Petri et ses extensions. Ce chapitre est également consacré à la modélisation, au sens physique (des paires de rails !), d'un réseau de transport par trams. Nous présentons notamment une maquette de ce réseau dont nous décrivons, syntaxiquement et sémantiquement, les principaux composants. Dans la troisième partie de ce chapitre, nous allons définir des règles sémantiques de transformation d'un modèle de réseau de transport public par trams en modèle réseau de Petri, dont le formalisme sera utilisé comme automate pour l'exploration de l'espace d'états dans le processus de model-checking.

- Le **troisième chapitre** 3 fait un état de l'art des techniques de model-checking, ainsi qu'une application concrète de ces techniques sur un modèle de réseau de transport par trams. Nous présentons les résultats que nous avons obtenus par l'application de deux model-checker (CPN-Tools et HELENA).

- Le **quatrième chapitre** 4 fait un état de l'art sur le **MDA** (**M**odel **D**riven **A**rchitecture) ou **IDM** (**I**ngenierie **D**irigée par les **M**odèles) en français et sur les **DSL** (**D**omain **S**pecific **L**anguage) ou *Langage dedié* en français. Nous y présentons notamment le lien entre les principes généraux de **MDA** et le processus de création des **DSL**.

II. **T**ransports **P**ublics **G**enevois

– Dans le **cinquième chapitre** 5, nous présentons toutes les étapes détaillées de la conception du DSL textuel "***TramLang***" qui signifie *Langage de Trams* et dont l'objectif est de fournir un langage de modélisation, spécifique au réseau de transport public par trams, pour faire du model-checking. Nous donnons tous les détails de la conception de la syntaxe abstraite, la syntaxe concrète et de la sémantique de notre DSL.

– Le **sixième chapitre** 6 décrit différentes phases de l'implémentation du DSL textuel "TramLang".

– Le **septième chapitre** 7 présente la conclusion sur ce travail et quelques idées d'approfondissement de recherche sur le sujet.

# Chapitre 1

# TPG : Transports Publics Genevois

`<< Les TPG sont ancrés dans le coeur des Genevois.`
`Modernes, publics et dynamiques, ils sont au service de toute une région. >>`

[TPG 09]

FIGURE 1.1 – un tram devant le site des TPG au Bachet-De-Pesay à Genève

## 1.1 Historique

L'entreprise publique autonome de transport en commun nommée **TPG** couvre la région de Genève en SUISSE ainsi que les départements français de l'Ain et de la Haute-Savoie. Sa création remonte au $1^{er}$ Janvier 1977 [TPG 09] et succède à la **CGTE** (**C**ompagnie **G**enevoise des **T**ramways **E**lectriques). Elle emploie 1550 personnes et exploite un réseau 1.2 en pleine expansion, desservi par des *tramways*, des *autobus* et des *trolleybus*.

## 1.2 Statistiques

Pour satisfaire ses clients aux attentes forcément diverses, Les TPG doivent faire face à des défis d'exploitation variés, tant au niveau de la fréquence de circulation de ses véhicules, qu'au niveau de la couverture de tout l'ensemble du canton, depuis les zones urbaines jusqu'aux régions éloignées du centre ville.

Les résultats présentés ci-dessous illustrent l'activité, en quelques nombres, des TPG et sont tirés du rapport "*Faits marquants et Chiffres Clés 2008*" [TPG 09]

### Voyageurs

| | |
|---|---|
| Millions de voyages | 168.0 |
| Millions de voyages x Km | 388.3 |
| Voyages en moyenne journalière | 460 |
| Millions de Km productifs parcourus | 20.5 |
| Millions de CHF de recettes voyageurs | 118.9 |

### Marché

| | |
|---|---|
| Population du Canton de Genève | 453'459 |
| Population de la zone desservie par les TPG | 446'388 |

### Prestations des TPG

| | |
|---|---|
| **Kilomètres-convois productifs parcourus** | |
| Tramways | 3'090'000 |
| Trolleybus | 3'942'000 |
| Autobus | 13'499'000 |
| Total | 20'531'000 |
| **Voyageurs transportés** | |
| Total du nombre de voyages par année | 167'967'000 |
| dont : sur les lignes urbaines | 155'145'000 |
| sur les lignes régionales | 9'897'000 |
| Moyenne par jour | 460'184 |
| **Voyages par habitant de la zone desservie** | **376** |
| **Réseau Tram** | |
| Total par année | 61'335'000 |
| Moyenne par jour | 168'041 |

## Offre

| | | |
|---|---|---|
| **Nombre de lignes** | | |
| Tramways | | 6 |
| Trolleybus | | 6 |
| Autobus | | 47 |
| dont: | urbaines | 32 |
| | régionales | 15 |
| Total | | 59 |
| **Longueur du réseau exploité** (y compris les parcours de service) | | |
| Tramways | | 21.1 km |
| Trolleybus | | 36.3 km |
| Autobus | | 327.2 km |
| dont: | réseau urbain | 172.0 km |
| | réseau régional | 155.2 km |
| Total | | 384.6 km |
| **Parc de véhicules** | | |
| Tramways articulés | | 67 |
| Trolleybus articulés | | 89 |
| Minibus | | 5 |
| Autobus | | 47 |
| Autobus articulés | | 185 |
| Total parc | | 393 |
| **Age moyen des véhicules** | | |
| Tramways | | 13 ans |
| Trolleybus | | 12 ans |
| Autobus | | 7 ans |
| **Nombre de places-voyageurs dans les véhicules** base: places debout 4 personnes/m² | | |
| Tramways | | 17'836 |
| Trolleybus | | 13'008 |
| Autobus | | 33'113 |
| Total | | 63'957 |
| **Nombre de places-km offertes** | | **2'451'194'000** |
| **Nombre de km-convoi offerts par habitant de la zone desservie** | | **45.99** |

Le tramway a historiquement toujours été présent à Genève, mais depuis les années 1990, il bénéficie d'une large revalorisation avec la création de nouvelles lignes et de nouveaux tronçons.

Le réseau est organisé autour de trois pôles :

- Bel-Air(Cité)
- Plainpalais
- Gare Cornavin.

Six lignes de trams sont exploitées par le tramway de la ville de Genève :

- La ligne 12 : Moillesulaz ⟷ Palettes(via Carouge)
- La ligne 13 : Nations ⟷ Palettes(via Carouge)
- La ligne 14 : Bachet-de-pesay/Augustins ⟷ Avanchet
- La ligne 15 : Nations ⟷ Palettes(via Acacias)
- La ligne 16 : Avanchet ⟷ Moillesulaz
- La ligne 17 : Moillesulaz/Gare de Chêne-Bourg ⟷ Lancy-Pont-Rouge.

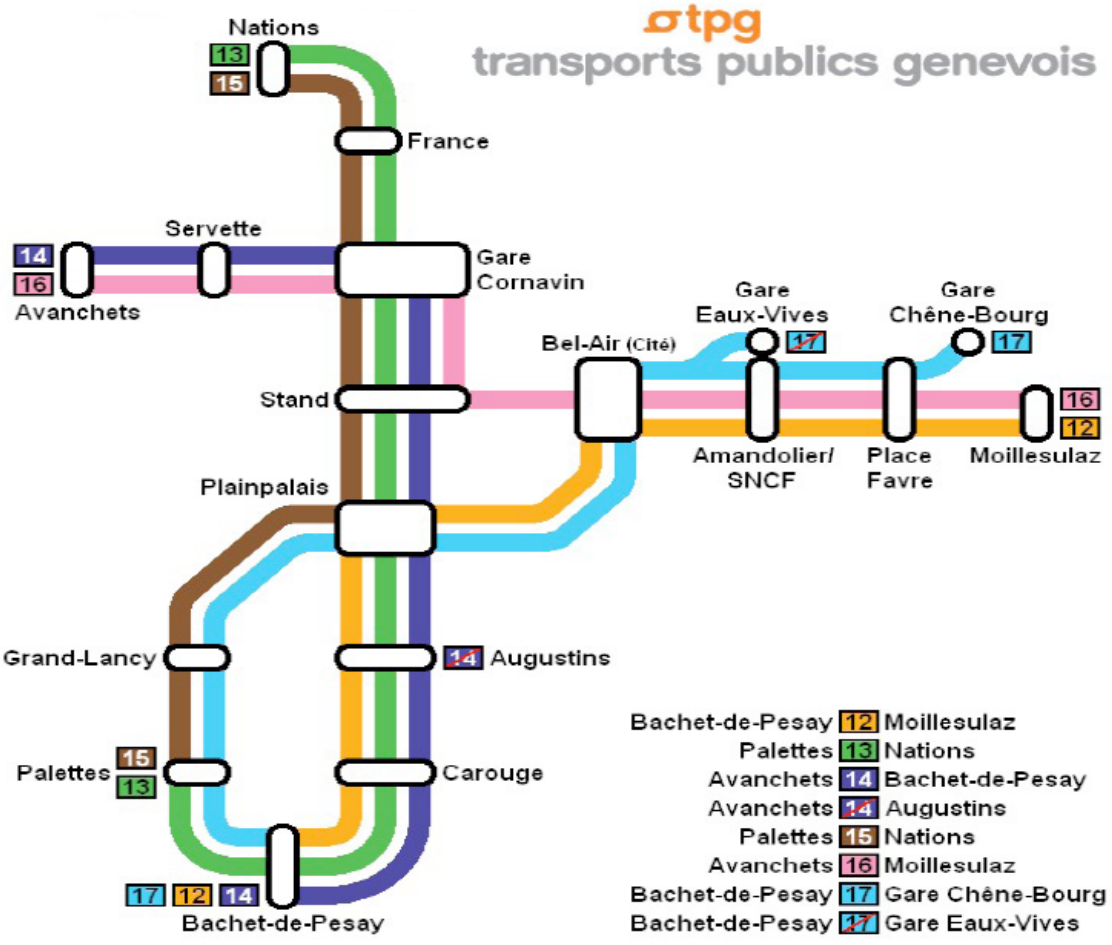



FIGURE 1.2 – Réseau des tramways TPG avec ses principales stations jusqu'au 14 décembre 2008

# Chapitre 2

# Modélisation d'un réseau de transport public par trams et transformation en RdP

*Qu'y a-t-il de commun entre une carte Michelin, un programme informatique, un texte de loi, une partition musicale, la célèbre formule* $E = mc^2$*, le plan d'une maison, la double hélice de l'ADN, les équations chimiques* $HCl + NaOH \longrightarrow NaCl + H_20$ *ou acide + base* $\longrightarrow$ *sel + eau ou encore un planning d'activités ? En quoi ces "objets" se ressemblent-ils ? Ce sont tous des modèles, des constructions élaborées par l'humain à des fins particulières. S'il existe des motifs de les rapprocher, il faut également reconnaitre qu'ils diffèrent par bien des points. En premier lieu, par le "langage" utilisé pour les exprimer : si certains sont graphiques (la carte, le plan par exemple), mathématiques (la formule d'Einstein), d'autres sont écrits en langage naturel (français, anglais, etc...) ou dans des codes symboliques (langages de programmation,...). Ils diffèrent également par leurs rôles et usages : les uns décrivent une "réalité", présente ou en devenir, physique ou immatérielle, concrète ou abstraite ; d'autres sont le support de théories spéculatives ou de normes à respecter. Les uns spécifient, d'autres décrivent, proposent ou imposent.*

*Mais, au-delà de leur diversité, les modèles présentent une caractéristique commune : ils sont tous porteurs d'informations et véhiculent de la connaissance ; ils sont l'expression dans un langage/formalisme donné d'un point de vue particulier porté sur un sujet d'études.* [CAP, 08]

Pour commencer ce chapitre, nous allons introduire le formalisme des modèles de réseaux de Petri (RdP), ses principales caractéristiques et ses extensions. Nous présentons ensuite la syntaxe abstraite [I] textuelle et graphique d'un TramwayNET [II]. Nous présentons enfin les règles sémantiques de transformation d'un TramwayNET en RdP.

La modélisation et la simulation d'un réseau de transport public urbain est une tâche délicate et compliquée. En effet, les systèmes distribués tels que les TramwayNET sont caractérisés par des comportements parallèles, synchrones, indéterministes et concurrents. Le formalisme des modèles de Rdp permet l'expression de ces comportements, qui sont représentés sur la figure 2.1 :

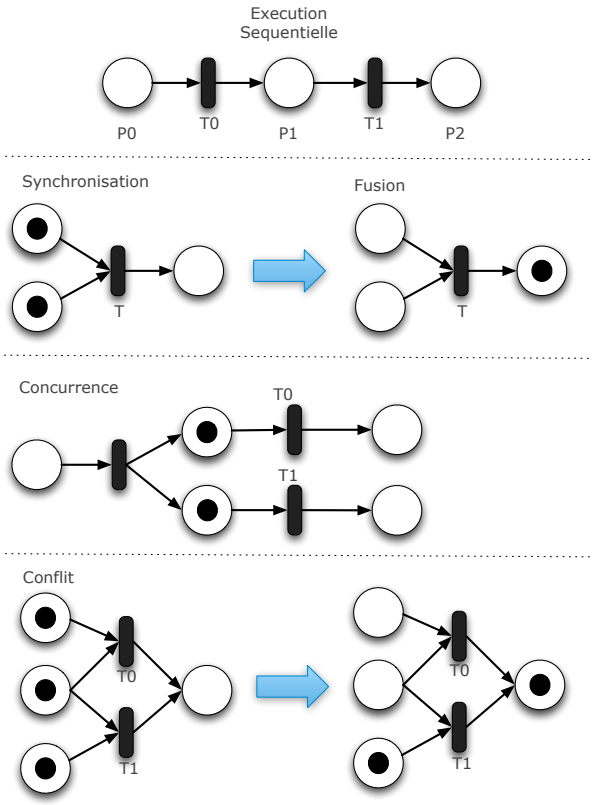



FIGURE 2.1 – Propriétés des RdPs

I. La structure interne du réseau et les liens hiérarchiques entre les éléments qui le composent.

II. Réseau de transport par trams

• **Exécution séquentielle** : le tir de la transition T1 est possible uniquement après le tir de la transition T0.

• **Synchronisation** : La transition T est tirable uniquement si il y a au moins un jeton dans toutes ces places en entrées.

• **Fusion** : Se produit lorsque les jetons des places en entrée d'une transition sont disponibles au même moment.

• **Concurrence** : les transitions T0 et T1 sont concurrentes. Cette propriété permet aux RdPs de modéliser des systèmes dans lesquels plusieurs processus s'exécutent de façon concurrente à un instant donné.

• **Conflit** : Les transitions T0 et T1 sont toutes les deux « *tirables* », mais le tir de l'une « *désensibilise* » l'autre et vice-versa.

## 2.1 Les Réseaux de Petri

Les réseaux de Petri (RdP) ont été introduits en 1962 par le mathématicien Allemand Carl Adam Petri [Petri, 62]. Ils présentent trois caractéristiques importantes :

– La modélisation et la représentation graphique de comportements intégrant du parallélisme, de la synchronisation et du partage de ressources. Ils décrivent à la fois la structure du système et son comportement.

– La vérification des propriétés possibles car basé sur un formalisme mathématique rigoureux.

– La simulation du comportement du réseau.

### 2.1.1 Les composants élémentaires

Un ***Graphe*** simple **G** est un couple formé de deux ensembles : un ensemble $\mathbf{X} = \{X_1, X_2, ..., X_n\}$ dont les éléments sont appelés *noeuds*, et d'un ensemble $\mathbf{A} = \{A_1, A_2, A_3, ....., A_m\}$ dont les éléments sont appelés *arêtes*. Dans le graphe tel que représenté par la figure 2.2, on peut attribuer des

pondérations aux arêtes et/ou des "noms" (valeurs) aux noeuds.

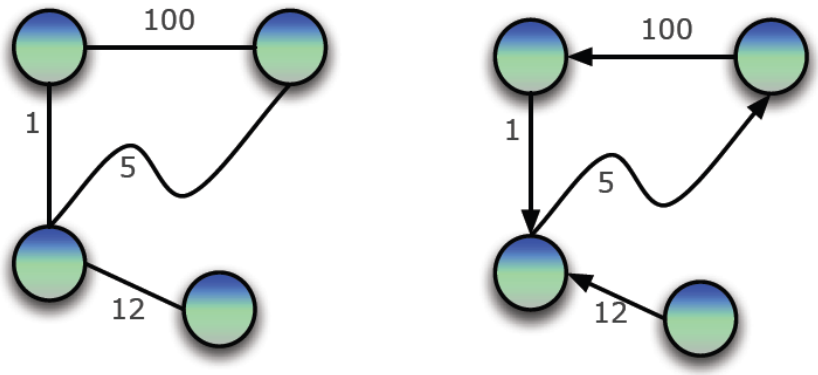



FIGURE 2.2 – Exemple de graphe non orienté(A gauche) et graphe orienté (A droite)

### 2.1.2 Places, Transitions, Arcs et jetons

Les réseaux de Petri simples sont des graphes - des figures formées de lignes (*arêtes*) et de points de jonction de ces lignes (*noeuds*) - qui représentent des systèmes à évènements discrets. Ces graphes sont dits *orientés* et sont composés de deux types de noeuds :

- des **Places**.
- des **Transitions**.

Ces noeuds contiennent des marques appelées **jetons** et sont réliés entre eux par des arcs orientés. Un **Arc** relie soit une transition à une place, soit une place à une transition et doit absolument avoir à son extrémité une place ou une transition.

Formellement [BUC 08], un réseau de Petri simple ou réseau de Petri Place-Transition est défini comme un quintuplet

R = <P, T, A, Pre, Post> :

ou

– P = $\{P_1, P_2, P_3, ..., P_n\}$ est l'ensemble fini des places.

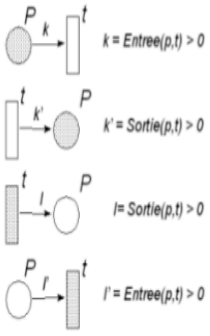
– T = $\{T_1, T_2, T_3, ..., T_m\}$ est l'ensemble fini des transitions.

– A = $\{A_1, A_2, ....., A_n\}$ est l'ensemble d'arcs A $\subset \{P \times T\} \cup \{T \times P\}$

– Entrée est une application,
Entrée : P $\times$ T $\longmapsto \mathbb{N}$ est appelée application d'incidence avant.

– Sortie est une application,
Sortie : P $\times$ T $\longmapsto \mathbb{N}$ est appelée application d'incidence arrière.

Vocabulaire : Soit p une place($p \in P$) et t une transition ($t \in T$),




PLACES :

- p est une *place d'entrée* de t si Entrée$(p, t) > 0$ :
- p est une *place de sortie* de t si Sortie$(p, t) > 0$ :

TRANSITIONS :

- t est une *transition d'entrée* de p si Sortie$(p, t) > 0$ :
- t est une *transition de sortie* de p si Entrée$(p, t) > 0$ :

k ,k',l,l' sont les valuations des arcs

Marquage : Définition

Le Marquage d'un Rdp est son état. Formellement [BUC 08], un marquage est une application

$$M : P \longmapsto \mathbb{N}$$

donnant pour chaque place le nombre de jetons qu'elle contient. Le marquage initial est généralement noté $M_0$.

Fonctionnement du RdP (Sémantique)

• une transition T est *tirable* pour un marquage M si et seulement si

$$\forall\ p \in P,\ M(p) \geq \text{Entrée}(p,T)$$

Il s'agit de la condition de franchissement de T depuis M.

• une transition T est franchissable depuis M, le tir (ou le franchissement) de T produit un nouveau marquage $M'$ donné par

$$\forall\ p \in P,\ M'(p) = M(p) - \text{Entrée}(p,T) + \text{Sortie}(p, T)$$

### 2.1.3 Un Exemple de RdP

Considérons le marquage du Rdp à gauche de la figure 2.3 avec le marquage $M_k$ tel que k = 0 ($M_0$ est le marquage initial). La transition T est *tirable*. A droite de cette figure on a le marquage après le tir de la transition T (marquage $M_1$). Lorsqu'on "tire" la transition T chaque place en sortie contient un nombre de jetons supplémentaires égal au poids de l'arc qui la relie avec T.

## 2.2 Les Extensions des réseaux de Petri

Les réseaux de Petri possèdent certaines extensions nécessaires à la vérification ou au contrôle d'applications temps-réels.

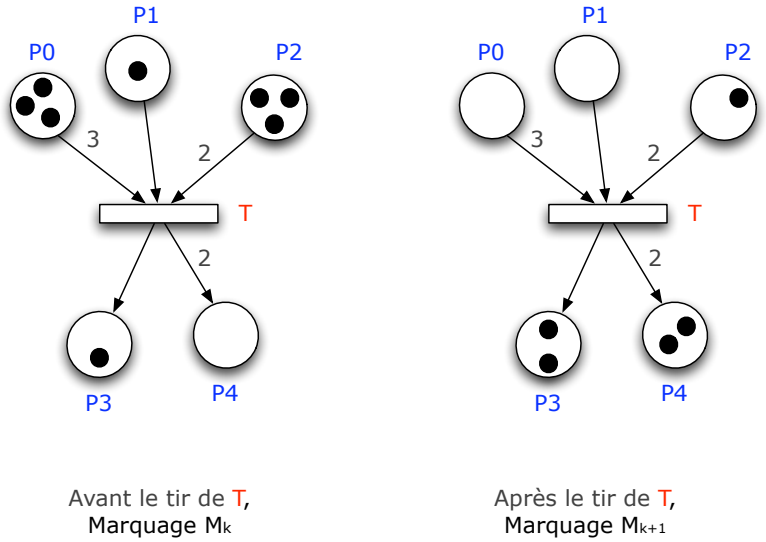



FIGURE 2.3 – Un Rdp avant et après le tir d'une transition

### 2.2.1 Réseaux de Pétri Colorés(RdpC)

Les RdP classiques ou RdP Places-Transitions peuvent rapidement devenir "***très***" larges tant au niveau de la quantité de places qu'au niveau du nombre d'arcs. Avec l'extention de ces réseaux qu'on appelle "réseaux de petri colorés"(RdpC), on modélise de manière compacte des systèmes ayant des composantes au comportement indentique. On réduit ainsi la taille des modèles crées.

les principales caractéristiques des RdpC :

- Les jetons sont "typés" par des ***couleurs*** en quantité finie.
- On associe à chaque transition un ensemble de couleurs "valides". Ce sont les couleurs de franchissement de transition.
- Tous les jetons d'une place appartiennent à un même type de données.
- On spécifie une *relation* entre les jetons consommés et les jetons produits. En d'autre termes, la valeur des jetons produits doit se référer à la valeur des jetons consommés.

**un exemple de RdpC**

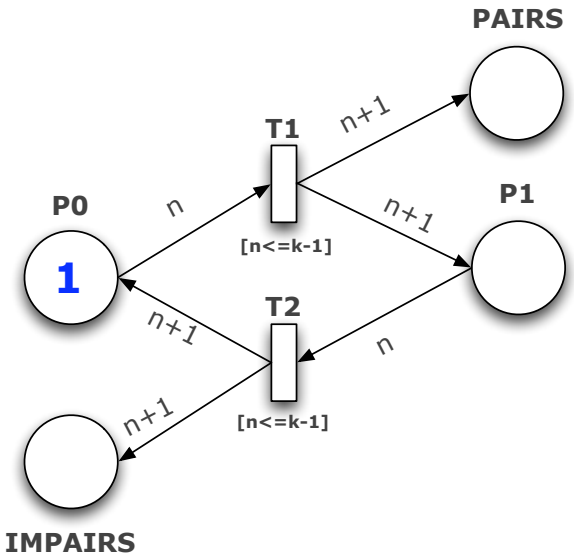



FIGURE 2.4 – Un exemple de RdpC : Liste de nombres pairs et impairs entre n=1 et k

Comme exemple, on considère un système qui modélise la liste des tous les nombres pairs et impairs compris entre des entiers **n** et **k** (On initialise le réseau avec **n** = 1). Le système est modélisé par le RdpC de la figure 2.4. Ce RdpC est composé :

- des places P0, P1, PAIRS et IMPAIRS qui contiennent des jetons de type entier. Les nombres pairs transitent par la place P1 et les nombres impairs transitent par la place P0. Les places PAIRS et IMPAIRS vont contenir respectivement les nombres pairs et les nombres impairs à la fin de la simulation du RdpC.

- des transitions T0 et T1 ayant une garde($[n \leq k-1]$) qui conditionne le "tir" de la transition. On aimerait obtenir la liste des nombres pairs et impairs compris entre **n** et **k**, donc la transition est *tirable* lorsqu'il y a un jeton (couleur) dans la place en entrée et lorsque **n** n'est pas plus grand que **k**.

- des arcs auxquels sont associés des expressions ("**n**" pour consommer la donnée des places en entrées et "**n+1**" pour incrémenter la valeur

de la donnée et la mettre dans les places en sorties).

Si on initialise le RdpC de la figure 2.4 avec le marquage initial $M_0$ tel que :
- $M_0$(P0) = {1}
- $M_0$(P1) = $\phi$
- $M_0$(PAIRS) = $\phi$
- $M_0$(IMPAIRS) = $\phi$.

Après l'exécution de la séquence de transitions T1 T2 T1 T2 T1 T2 T1 T2 T1, on obtient le marquage $M_8$ tel que :

- $M_8$(P0) = $\phi$
- $M_8$(P1) = {10}
- $M_8$(PAIRS) = $\{2, 4, 6, 8, 10\}$
- $M_8$(IMPAIRS) = $\{3, 5, 7, 9\}$.

Le marquage $M_8$ pour les places PAIRS et IMPAIRS nous donne respectivement, la liste de tous les nombres pairs et impairs entre n=1 et k=10.

### 2.2.2 Réseaux de Pétri Temporisés (RdpT)

Dans la littérature [MER, 74], [RAM, 74], [SIF, 77], Il existe plusieurs extensions temporelles des Rdp classiques. Selon l'élément du réseau qui porte l'information temporelle, on parle de :

↪ Réseaux A-temporisés (les temporisations sont associées aux arcs).

↪ Réseaux P-temporisés (à une Place, on associe un temps qui est la durée d'indisponibilité d'un jeton arrivant dans cette place).

↪ Réseaux T-temporisés (l'information temporelle est associée à la transition T). On distingue deux types d'information temporelle :

- On associe à la transition un temps **t** qui est la durée associée à l'exécution de la dite transition

- On associe à la transition un intervalle de temps [ $\mathbf{t}_{min}$, $\mathbf{t}_{max}$ ] : $\mathbf{t}_{min}$ est la date de tir ou plus tôt et $\mathbf{t}_{max}$ est la date de tir au plus tard.

Le modèle de cette dernière variante a été développé par P.M. Merlin [MER, 74] en 1974. Le modèle de Merlin a été conçu pour l'étude des problèmes de recouvrement, pour les protocoles de communications. Dans ce modèle, à chaque transition est attachée une contrainte temporelle de type intervalle.

De façon informelle, l'intervalle associé à la transition T est relatif au moment ou la transition devient validée. Soit [ $\mathbf{t}_{min}$, $\mathbf{t}_{max}$ ] cet intervalle, supposons que T soit validée à l'instant $\tau$, alors elle peut être franchie dans l'intervalle matérialisé par les quantités ($\mathbf{t}_{min} + \tau$) et ($\mathbf{t}_{max} + \tau$), sauf si elle est « *désensibilisée* » à cause du franchissement d'une autre transition avec laquelle elle était en conflit.

La définition formelle [BON 01] de ces réseaux est la suivante :

Un réseau de Petri T-temporisé à intervalle est un doublet <R, IS> où :

R est un RdP marqué.

• IS : T $\longrightarrow \mathbb{Q}^+$ x ($\mathbb{Q}^+ \cup \infty$) avec $\mathbb{Q}^+$ l'ensemble des nombres rationnels positifs.

• $t_i \longrightarrow \mathrm{IS}(t_i) = [\ \mathbf{t}_{min_i}, \mathbf{t}_{max_i}\ ]$ avec $\mathbf{t}_{min_i} \leq \mathbf{t}_{max_i}$.

IS($t_i$) définit l'intervalle statique associé à la transition $t_i$. Ainsi, une transition $t_i$ doit être sensibilisée (validée) pendant au moins $\mathbf{t}_{min_i}$ unités de temps avant de pouvoir être tirée et ne peut rester validée au-delà de $\mathbf{t}_{max_i}$ unités de temps sans être tirée. Le tir d'une transition est alors de durée nulle (instantanée).

Nous pouvons noter que ce modèle comporte deux sémantiques temporelles :

⊙ Une sémantique *forte*, où les contraintes temporelles (en particulier la contrainte maximale $\mathbf{t}_{max}$) peuvent forcer le tir d'une transition : *La transition doit être tirée.*

⊙ Une sémantique temporelle *faible*, où, bien que toutes les conditions nécessaires au franchissement d'une transition soient réunies, cette dernière peut ne pas être franchie : *La transition peut être tirée.*

### 2.2.3 Réseaux de Pétri Colorés Temporisés (TCPN)

Nous nous intéressons particulièrement à la **sémantique temporelle** des Réseaux de Petri colorés temporisés non-hierarchiques (TCPN) ou encore timed non-hierarchical Colored Petri Net en anglais. les modèles de TCPN permettent d'exprimer le comportement des systèmes tels que les réseaux de transport par trams dans lesquels les horaires de circulation attribués à chaque véhicule (tram) sont synchronisés par rapport à une heure de référence. Le model-checker CPNTools [CPNTools] permet de créer, analyser et simuler des TCPN.

**Définition formelle des TCPN** [Jen 97]

Un TCPN est un tuple $<$ **CPN**, **R**, $\mathbf{r}_0>$ :

$\hookrightarrow$ **CPN** est un réseau de Petri coloré (non-hiérarchique [a]).

$\hookrightarrow$ **R** c'est l'ensemble des valeurs du temps aussi appelées "**TimeStamp**". Le "**TimeStamp**" c'est le temps modèle (model time) où le jeton (couleur) peut être utilisé (consommé).

$\hookrightarrow$ $\mathbf{r}_0 \in \mathbf{R}$ (l'ensemble des "**TimeStamp**") et est appelé "**Start Time**" ou "**Global Clock**". Sous CPNTools, sa valeur est en générale initialisée à 0 et tout au long de l'exécution du réseau il prend la valeur du minimum de l'ensemble **R**.

*a.* un RdP hiérarchique est une construction imbriquée. Un réseau à l'intérieur d'un réseau. En général, tout élément de RdP peut être substitué par réseau imbriqué, CPNTools utilise les opérations de substitution de transitions et de fusion de places pour créer des RdP hiérarchiques

**Les Multisets et le temps**

Les multisets sont largement utilisés dans les RdPC pour la représentation du marquage dans les places. Dans un multiset, à la différence des Sets (ensembles), chaque élément a une multiplicité définie, i.e chaque élement possède un nombre défini d'occurences. Un multiset peut être vu comme une liste de paires, chaque paire contient alors un élément et sa cardinalité.

L'opérateur du construction de multiset est le *back-quote* (‘). Par exemple, 2‘3 est le multi-set avec deux occurences de la couleur 3.

Les "*timed multi-sets*" sont utilisés pour représenter les retards (délais) dans le modèle. La déclaration de la couleur correspondante devrait être accompagnée du mot-clé *timed*. Les opérateurs @, @+ et @@+ sont utilisés pour ajouter des **TimeStamp** aux couleurs. L'ajout d'un temps de retard x à une couleur C attachera un **TimeStamp** avec une valeur qui est égale au temps modèle actuel + x à la couleur C. Voici les opérations valident pour les "*timed multi-sets*" :

- C @ t : ajout du **TimeStamp** t à la couleur C.
- ms @+ i : ajout l'entier i, représentant le delai, à toutes les couleurs du multi-set ms.

**un exemple de TCPN**

L'exemple de la figure 2.5 montre la trace du processus d'écoulement du temps sous CPNTools. On a les déclarations suivantes :

**colset** NAME = with CLOVIS | MARIUS | INGRID | JOSEE timed ;

**var** *id* : NAME ;

*id* = 1‘JOSEE@+300++ 1‘MARIUS@+100++1‘MARIUS@+200++ 1‘CLOVIS@+100++1‘CLOVIS@+200 ;

Ces déclarations signifient que le jeton JOSEE@+300 pourrait être consommé seulement après le temps modèle égale à 300, le jeton MARIUS@+100 pourrait être consommé seulement après le temps modèle égale à 100, etc...

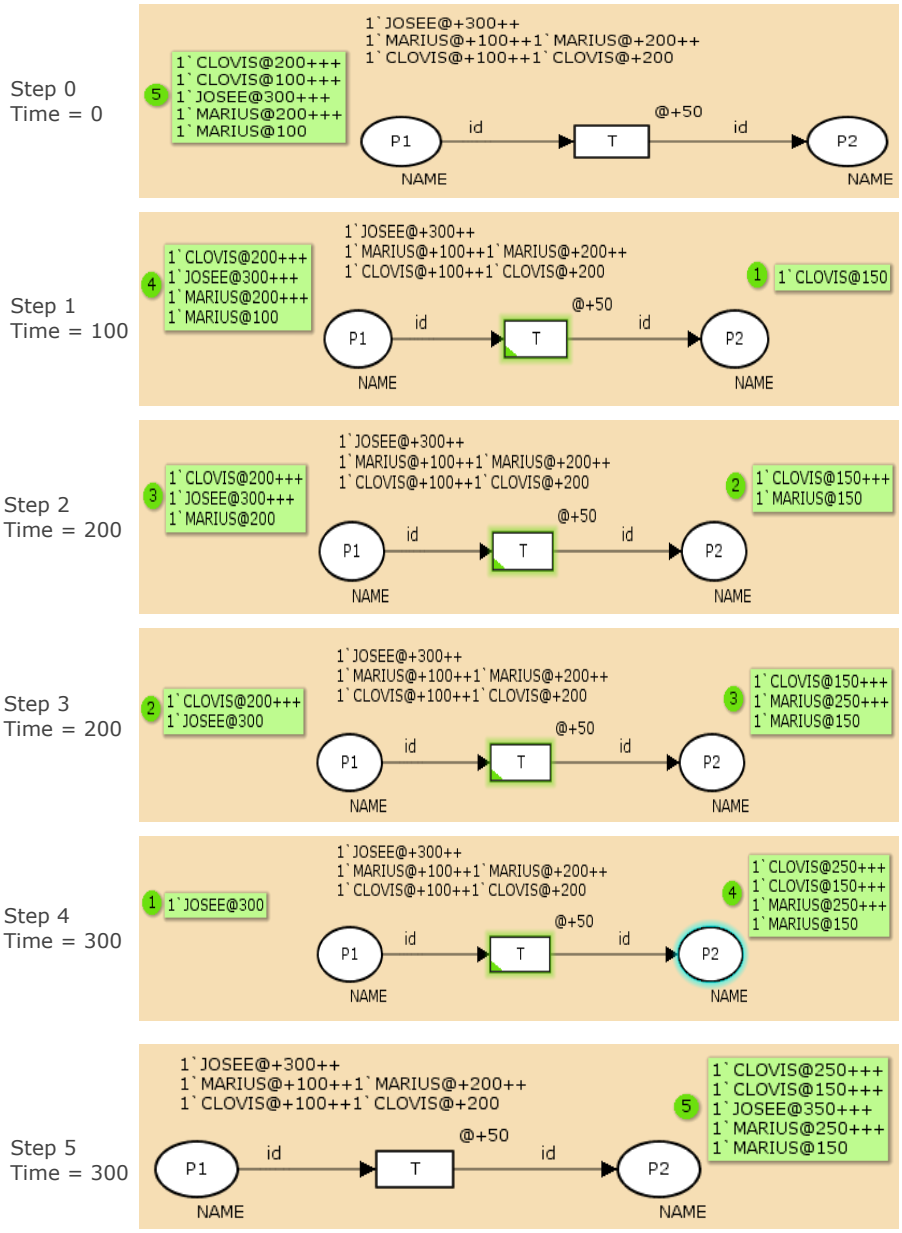



FIGURE 2.5 – Processus d'écoulement du temps sous CPNTools

**Description de l'écoulement du temps de la figure** 2.5

Au temps modèle égal à 0, aucune transition n'est « sensibilisée ». La prochaine instance du temps modèle égale à 100. CPNTools exécute deux étapes (steps) pour déplacer les jetons MARIUS@+100 et CLOVIS@+100 qui ont leur **TimeStamp** initial égale à 100. A l'instance du temps modèle égale à 200, CPNTools exécute deux étapes (steps) pour déplacer les jetons MARIUS@+200 et CLOVIS@+200 qui ont leur **TimeStamp** initial égale à 200. Et finallement à l'instance du temps modèle égale à 300, CPNTools déplace le jeton JOSEE@+300 qui a son **TimeStamp** initial égale à 300.

### 2.2.4 Réseaux de Pétri Algébriques(APN)

Les APN sont une évolution bien connue des RdPs dans laquelle les données sont représentées grâce à des types abstraits algébriques ou encore *Algebraic Data Types* en anglais. Ce formalisme peut être comparé au formalisme des RdPC (Réseaux de Petri colorés). Dans le cas des APN, les données sont définies inductivement et leur sémantique est donnée par une axiomatisation permettant des preuves et calculs automatiques.

Les APN ont été inventés [APN] par Jacques Vautherin [VAU 85] dans sa thèse de doctorat et améliorés par Wolfgang Reisig [WOL 91].

Pour faciliter la compréhension, on peut considérer les termes des ADT comme des **jetons typés**. On suppose que seules les opérations définies par l'ADT seront applicables aux jetons. Considérons par exemple l'ADT de la figure 2.6 qui définit les naturels. Les jetons de ce type pourront être additionnés, soustraits, multipliés, divisés, etc... On pourra donc appliquer toutes les opérations possibles prédéfinies sur les naturels. Les ADT permettent de construire des modèles « *corrects* » : L'ADT des entiers garanti que des entiers ne prennent que des valeurs entières, dans le même ordre d'idées, l'ADT des booléens garanti que les booléens ne prennent que deux valeurs. *Pour résumer, les ADTs permettent de définir le domaine des types de données et leur comportement. Certains ADTs permettent de limiter le domaine des types de données pour mieux faire la vérification de propriétés.*

La syntaxe de l'ADT de la figure 2.6 qui définit les naturels est utilisée dans l'outil ALPINA ([ALPINA]), qui est un model-checker développé au sein du group SMV (Software Modeling and Verification) à l'Université de

Genève.

```
// The naturals ADT

Adt naturals;
Interface
        Sorts nat;
        Generators
                zero : nat;
                suc : nat  -> nat;
        Operations
                plus : nat, nat  -> nat;
                minus : nat, nat  -> nat;
                times : nat, nat  -> nat;
Body
        Axioms
                //plus
                plus(zero, $x) = $x;
                plus(suc($x), $y) = suc(plus($x, $y));

                //minus
                minus($x, zero) = $x;
                minus(suc($x), suc($y)) = minus($x, $y);

                //times
                times($x, zero) = zero;
                times($x, suc($y)) = plus($x, times($x, $y));

        Where
                x : nat;
                y : nat;
End naturals;
```

FIGURE 2.6 – ADT simplifié des naturels

La définition formelle des APN que nous citons ci-dessous est tirée du cours "*Modeling and Verification*" du cursus Bachelor ès sciences informatiques à l'université de genève.

**Definition : *Algebraic Petri net specification***

An *algebraic Petri net specification* is defined as $N - SPEC =< Spec, T, P, X, AX >$, where :

- $Spec =< \Sigma, X', E >$ is an algebraic specification extended in $< [\Sigma], X', E >$, where $\Sigma =< S, F >$.
- $T$ is the set of transition names.
- $P$ is the set of place names, and we define a function $\tau : P \longrightarrow S$ which associates a sort to each place.
- $AX$ is a set of axioms defined below.

**Definition : *Algebraic Petri net axioms***

Given an algebraic Petri net N-SPEC $=< Spec, T, P, X, AX >$, an axiom of $AX$ is a 4-tuple $< t, Cond, In, Out >$, s.t. :

- $t \in T$ is the transition name for which the axiom is defined.
- $Cond \subseteq T_{\Sigma,X} \times T_{\Sigma,X}$ is a set of equalities attached to transition name $t$ for this axiom, which are satisfied iff all the relations $a = b$ of $Cond$ are satisfied.
- $In = (In_p)_{p \in P}$ is a $P$-sorted set of terms, s.t. $\forall p \in P, In_p \in (T_{[\Sigma],X})_{[\tau(p)]}$ is the label of the arc from place $p$ to transition $t$.
- $Out = (Out_p)_{p \in P}$ is a $P$-sorted set of terms, s.t. $\forall p \in P, Out_p \in (T_{[\Sigma],X})_{[\tau(p)]}$ is the label of the arc from transition $t$ to place $p$.

**Definition : *Algebraic specification***

A *many sorted algebraic specification* $Spec =< S, F, X, AX >$ is a signature extended by a collection of axioms $E$ on variables $X$.

**un exemple d'APN**

Considérons l'APN de la figure 2.7. Cet APN modélise le scénario du retour de monnaie et du pourboire [III] :

Le modèle d'APN est composé de :

⊙ l'ADT **PiecesDeMonnaie** avec un nombre fini de valeurs possibles. On mentionne cela car les ADTs peuvent avoir un nombre infini de valeurs comme les *naturals* et les *integer*.

⊙ La place **MONNAIE**, dont les jetons sont typés ***piece***, initialisée avec le multiset 2'20cCH ++ 2'1CH ++ 1'5CH. Ce multiset est composé de deux occurences de la terme *20cCH*, deux occurences du terme *1CH* et une

III. Le pourboire est une somme d'argent que l'on verse à une personne en remerciant d'un service ou de la qualité de celui-ci.

occurence de la terme *5CH* ; qui modélisent respectivement 2 pièces de 20 centimes CH, 2 pièces de 1 CH et 1 pièce de 5 CH.

⊙ La place **POURBOIRE**, dont les jetons sont typés ***piece***, qui est initialement vide.

⊙ La transition **T**, avec un condition de tir [(x = 20cCH) or (x = 1CH)] qui est un *terme* booléen.

⊙ La variable **x** qui doit être du même sort (type) que les jetons des deux places.

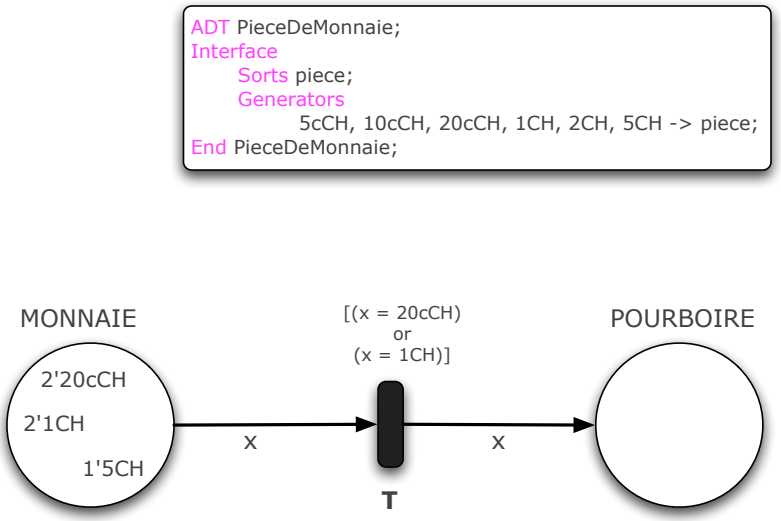



FIGURE 2.7 – exemple d'APN : Paiement et Pourboire

Le tir de la transition prend de façon indeterministe un jeton de la place **MONNAIE** et ajoute ce jeton dans la place **POURBOIRE** si et seulement si il vérifie la condition de tir de la transition **T**. Après l'éxécution de la séquence de transitions T T T T, on obtient le marquage :

$M_4$(**MONNAIE**) = 1'5CH
$M_4$(**POURBOIRE**) = 2'20cCH + 2'1CH.

## 2.3 Modélisation d'une maquette de *TramwayNET*

Dans un TramwayNET, les trams roulent sur des voies ferrées (des paires de rails) spécialement conçues à cet effet. les lignes de trams qui composent le tramway se croisent plusieurs fois et s'entremêlent pour finalement, dans certains cas, n'irriguer qu'une toute petite partie du réseau. Considérons un modèle de tramway au sens physique (ou il n'y a que des paires de rails). Pour concevoir ce modèle, il faut identifier la ou les plus petites entités qui le composent.

### 2.3.1 Description de modèles Physiques de TramwayNET

Après avoir étudié une petite maquette du modèle physique de tramwayNET de la ville de Genève 1.2, nous avons dessiné fidèlement, une partie (figure 2.8) de cette maquette en terme de paires de segments, nous l'avons analysé et nous la décrivons comme une composition d'entités de base que nous avons appelés "**TrackElement**".

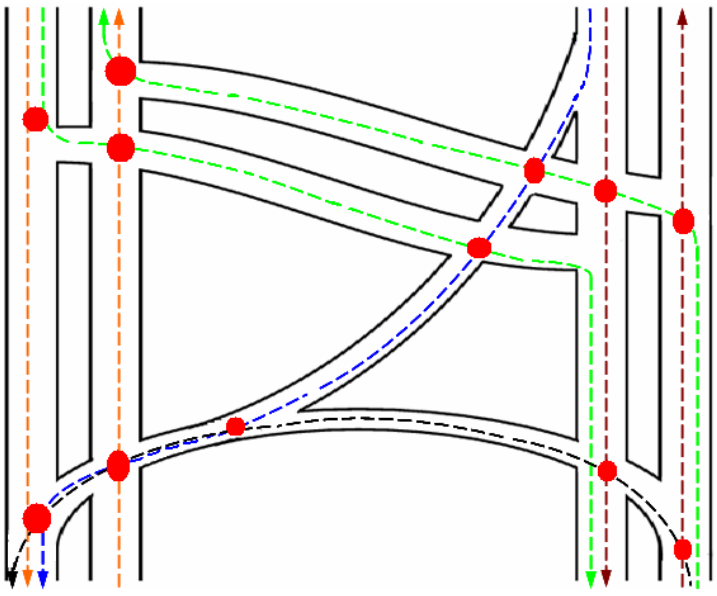


FIGURE 2.8 – Une partie du tramway de la ville de Genève partagée par cinq lignes de trams : Le carrefour de Plainpalais

Par exemple, pour identifier les composants élémentaires ("TrackElement") de la maquette de la figure 2.8, on sélectionne chaque point rouge et on compte de nombre de lignes en entrée qui ont des directions différentes et le nombre de lignes en sortie qui ont des directions différentes. Après analyse, on obtient quatre "TrackElement", qui sont représentés sur la figure 2.9. On distingue notamment :

- Une paire de rails(**[C/1,1]**) et
- Un croisement une entrée et deux sorties (**[C/1,2]**)
- Un croisement deux entrées et une sortie (**[C/2,1]**)
- Un croisement deux entrées et deux sorties (**[C/2,2]**).

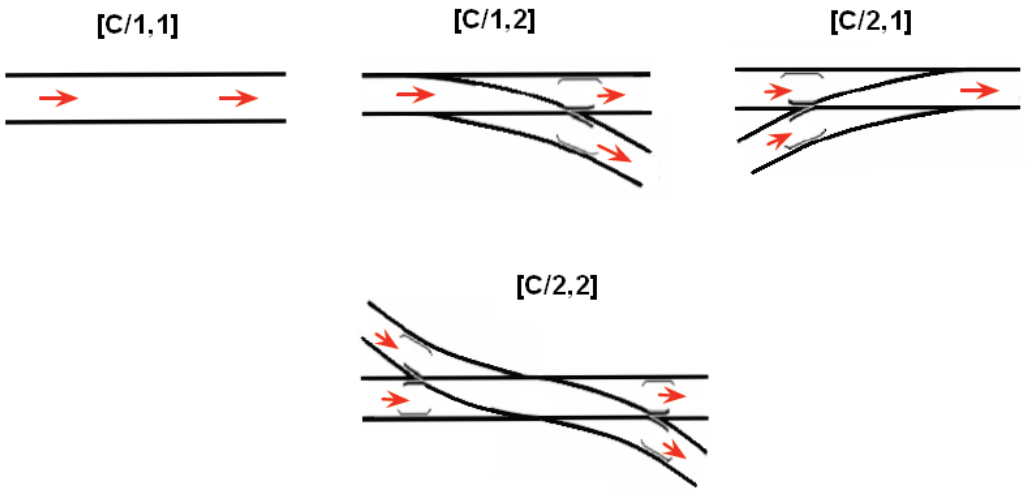



FIGURE 2.9 – Les Elements de base("TrackElement") qui composent un tramwayNET

### 2.3.2 Décomposition des entités (TrackElement) de TramwayNET

Tous les TrackElements (figure 2.9) possèdent au moins une entrée et une sortie. La relation entre un TrackElement et ses entrées est une relation entre un Composant et ses composés. Nous allons formaliser ces entrées par des "PORTS", qui sont de types "InputPORT" (pour entrer dans un TrackElement) et "OutputPORT" (pour sortir d'un TrackElement).

La figure 2.10 regroupe les quatre "TrackElement" avec leurs "PORTS"

respectifs.

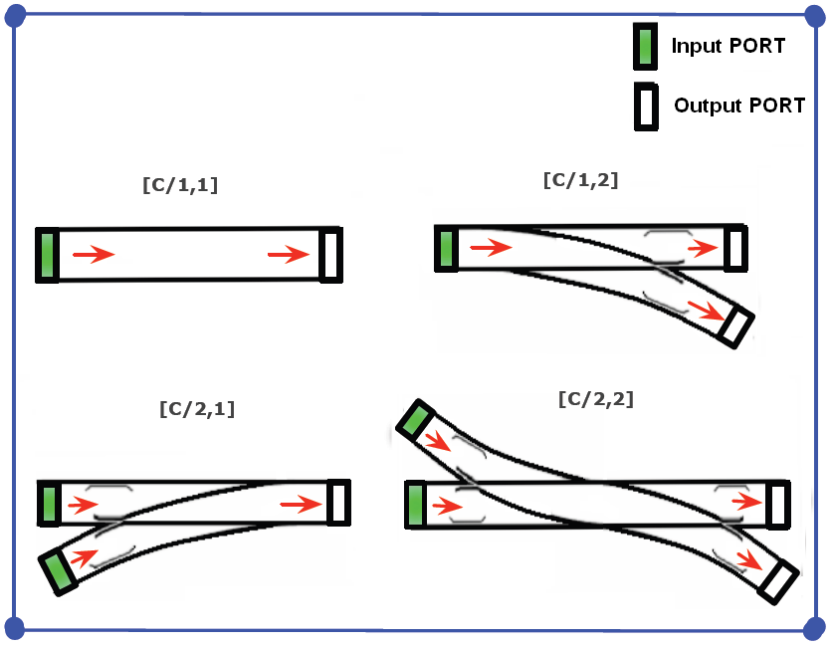



FIGURE 2.10 – Les Elements de base("TrackElement") avec leurs ports

Considérons un tramwayNET quelconque, le processus de la figure 2.11, montre toutes les étapes de décomposition de ce TramwayNET :

- **Etape 1** : On découpe le tramwayNET en TrackElements.
- **Etape 2** : On crée les composants (ports) de chaque TrackElement et on crée une liaison entre chaque ports.

Pour illustrer la réalité, nous présentons quelques images de TramwayNET dans le monde qui peuvent être modélisé par ces TrackElement :

- La figure 2.12 représente une petite partie du TramwayNET de la ville de Toronto au Canada, cette jonction est composée de plusieurs TrackElement de type **[C/1,3** qui peuvent être modélisés par la figure 2.13.

- La figure 2.14 représente une petite partie du TramwayNET de la ville de Genève en Suisse. Sur cette image, on observe un TrackElement de type **[C/2,1]**, et un TrackElement de type **[C/1,1]**. Ceci représente une superposition de deux TrackElement et peut être représentée par

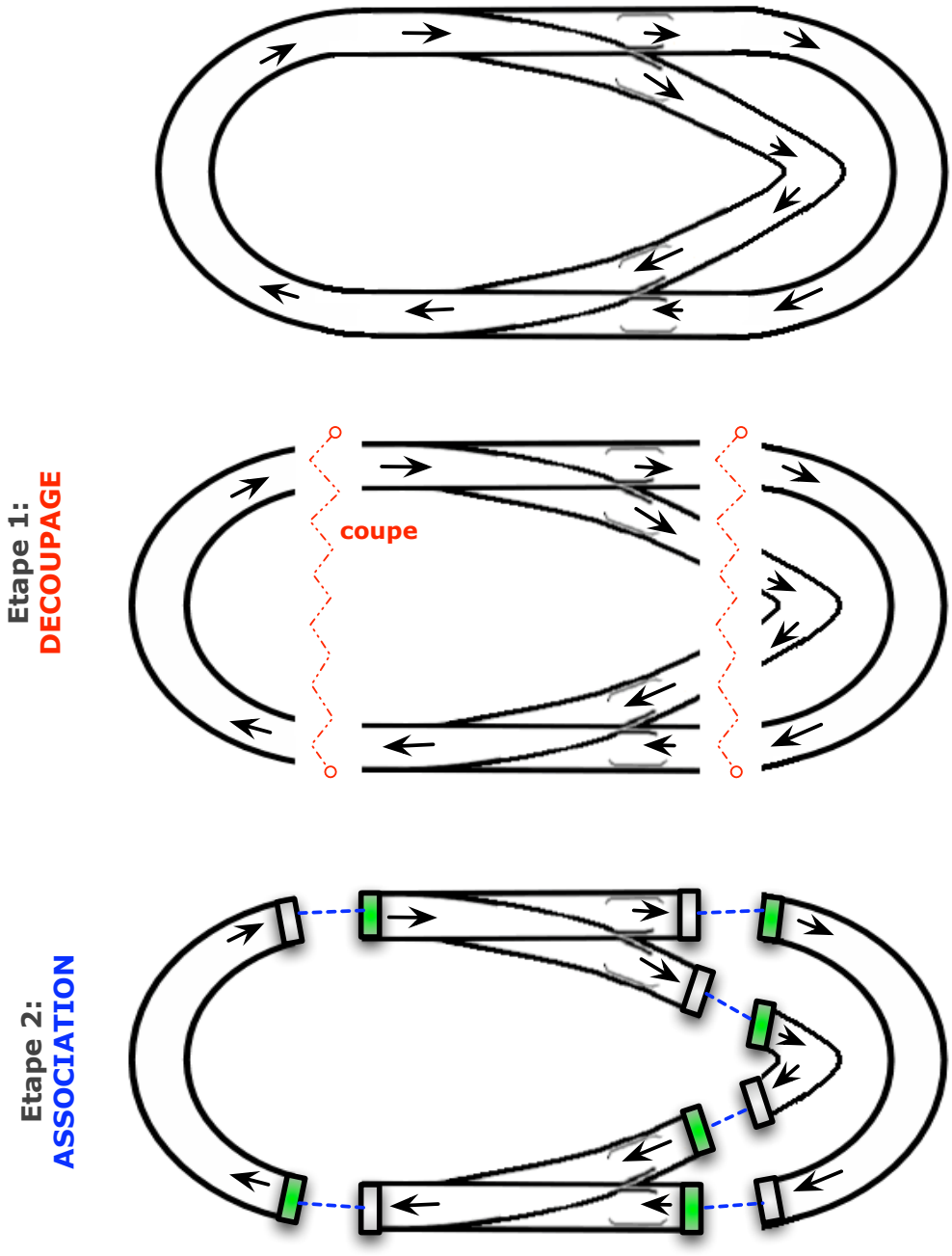



FIGURE 2.11 – Processus de décomposition d'un TramwayNET

l'image de droite la figure 2.14. Sur cette figure, la paire de rails en bleu n'a qu'une seule direction en entrée et en sortie. Dans le RdP équivalent à ce TracElement, on gère la direction des trams en mettant une garde (condition) aux transtions en sortie. La garde vérifie que l'identificateur (nom) du tram appartient bien à une liste prédéfinie.

– La figure 2.15 représente une petite partie du TramwayNET de la ville d'amsterdam (rue Leidsestraat), On y observe un TrackElement de type **[C/1,2]** (ce TrackElement pourrait aussi être de type **[C/2,1]**, auquel cas, il suffirait de changer la direction des flèches). On le modélise par l'image à droite de la figure 2.15

Il est à noté qu'il existe des TrackElement (**[C/n,m]**) tel que $n \geq 3$ et $m \geq 3$. De tels composants pourraient être modélisés par la figure 2.16.

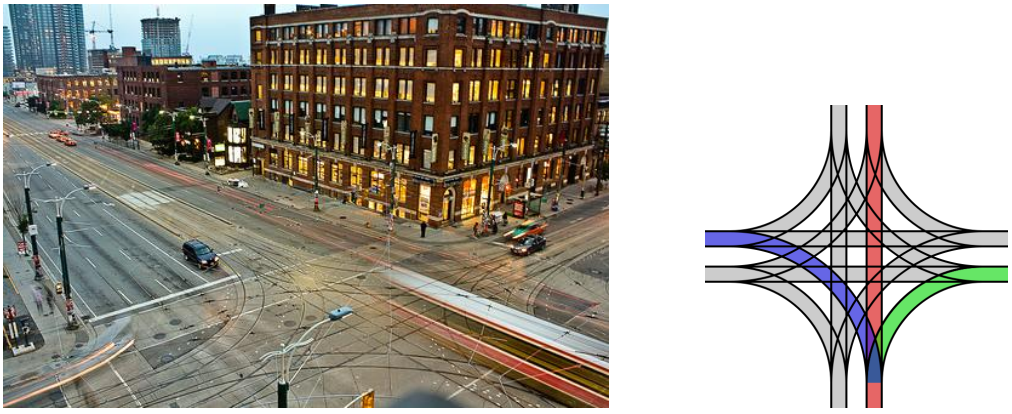


FIGURE 2.12 – [A gauche] Une "grande jonction" à Toronto(Canada), entre les lignes Spadina et Queen, [A droite] la maquette de cette "grande jonction"

FIGURE 2.13 – Un TrackElement **[C/1,3]** : Croisement 1 entrée et 3 sorties

FIGURE 2.14 – [A gauche] Une jonction de lignes de tram à la gare Cornavin de Genève, Photo prise le Jeudi 20 Août 2009, [A droite] Superposition de deux TrackElements **[C/2,1] et [C/1,1]**

FIGURE 2.15 – [A gauche] Un croisement de rails de trams à Leidsestraat, Amsterdam, [A droite] Un TrackElement **[C/1,2]** : Croisement 1 entrée et 2 sorties

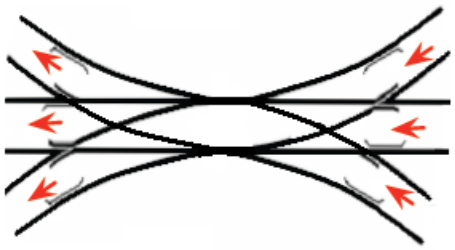


FIGURE 2.16 – Un TrackElement **[C/3,3]** : Croisement 3 entrées et 3 sorties

## 2.4 Transformation d'une maquette de *TramwayNET* en RdP

Après avoir modélisé le système, une question naturelle que l'on se pose généralement est « *Que peut-on faire avec ce modèle* ? ». Nous allons transformer le modèle de TramwayNET en RdP dont le formalisme nous donne les techniques pour analyser les propriétés prédéfinies sur le système.

Dans cette partie, nous présentons les régles sémantiques de transformation de toute maquette de TramwayNET en RdP :

- Un TrackElement est représenté par une Place.
- Un lien entre deux TrackElement est représenté par une Transition.
- Un InputPORT d'un TrackElement est représenté par un arc qui relie une Transition à une Place (InputArc).
- Un OutputPORT d'un TrackElement est représenté par un arc qui relie une Place à une Transition (OutputArc).

Considérons la maquette décomposée de la figure 2.11. La figure (2.17) illustre la transformation, via les règles sémantiques que nous venons de définir, de cette maquette en RdP.

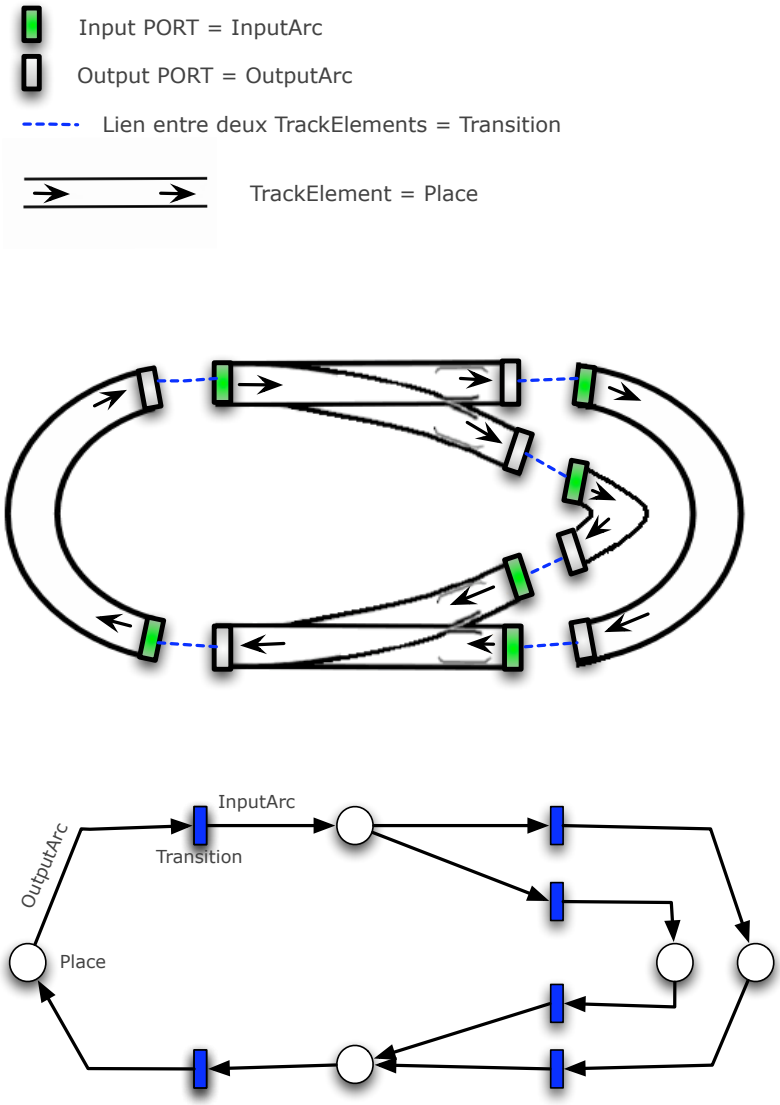



FIGURE 2.17 – Transformation d'un TramwayNET en Rdp

# Chapitre 3

# Model-checking

Le model-checking, aussi appelée *validation*, est une méthode de vérification exhaustive des systèmes, i.e au lieu de tester des prototypes physiques, on vérifie exhaustivement un modèle mathématique du système que nous voulons analyser, en utilisant un algorithme qui explore toutes les configurations possibles du système.

Dans le domaine du génie logiciel, la vérification c'est répondre à la question " faisons-nous le produit correctement ? " (Par exemple on fait des tests sur le logiciel, on traque les bugs, etc...). La validation c'est répondre à la question " faisons-nous le bon produit ? " (On valide un modèle du système par rapport à une propriété exprimée sur ce système).

Les premiers travaux [Model-checking Tools] sur le model-checking de formules logiques temporelles ont étés menés par Edmund M. Clarke et E.Allen Emerson en 1981, ainsi que par Jean-Pierre Queille et joseph Sifakis en 1982.

Formellement, le model-cheking c'est : $\boldsymbol{M} \overset{?}{\models} \varphi$

* $\boldsymbol{M}$ c'est le modèle du système sous observation
* $\varphi$ est la propriété à vérifier
* Pré-requis : Un langage de spécification avec sa sémantique.

Le plus gros avantage de la méthode de model-checking est qu'il est (idéalement) complètement *automatique* et un contre-exemple est toujours retourné lorsqu'une propriété n'est pas vérifiée. Ce dernier point a été déterminant pour des applications industrielles.

Cependant, les techniques du model-checking souffrent du problème de l'explosion du nombre d'états. L'automate explicite est généralement infini(ou énorme) et donc impossible à explorer. Pour résoudre ce problème, l'idée c'est de travailler sur une abstraction finie de l'automate. Cette abstraction doit conserver certaines propriétés du système étudié.

## 3.1 Model-checking : Etat de l'art

Le processus (figure 3.1) d'application du Model-checking comporte trois phases :

- **Phase I** : La spécification formelle des besoins (sécurité, qualité, maintenabilité, réduction des coûts, utilisabilité, robustesse, fiabilité, modularité, etc...) que l'on désire assurer sur le système.

- **Phase II** : La modélisation formelle du système.

- **Phase III** : La vérification formelle, que le système répond bien à ces besoins :

↪ Dans la **phase I**, on spécifie les propriétés à vérifier par des formules logiques, par exemple la logique temporelle c'est à dire une logique avec une notion de temps (qui permet, par exemple d'exprimer des propriétés sur les états précédents/suivants d'un système)

↪ Dans la **phase II**, on modélise formellement le système par un mécanisme de flot d'informations.

Pour la construction du modèle considéré, il convient en général de l'exprimer au moyen d'un graphe orienté, formé de noeuds et de transitions. Chaque noeud représente un état du système et chaque transition représente une possible évolution du système, d'un état donné vers un autre état.

Dans ce mémoire, nous utilisons les modèles de RdP, pour les raisons que nous avons évoquées au chapitre 2.1.

↪ **Phase III** : Analyse de l'espace d'états du système. L'algorithme de model-checking combine alors le modèle et la formule pour calculer l'ensemble des états accessibles du système. Une propriété est vérifiée s'il existe

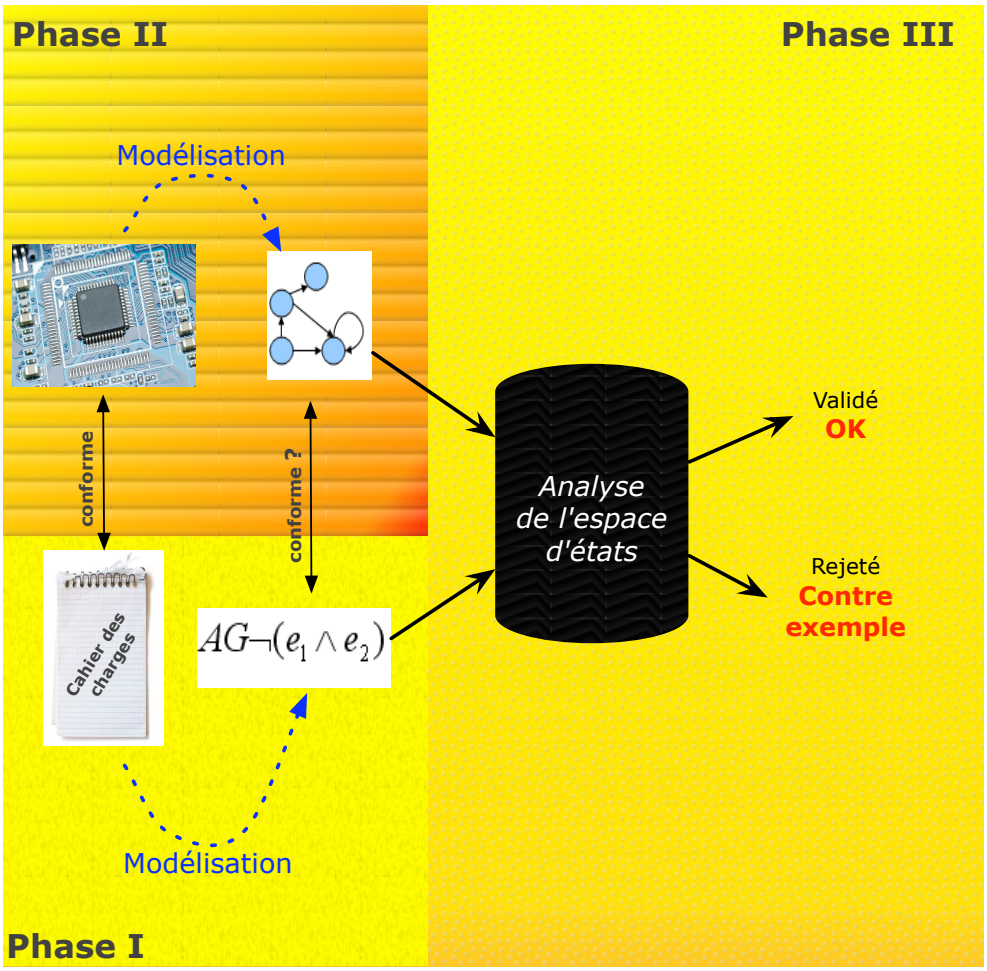



FIGURE 3.1 – Processus général de model-checking

au moins un chemin qui relie l'état initial à l'ensemble des états qui vérifient cette propriété.

Le model-checking a aujourd'hui de nombreuses applications, notamment, dans la détection de bugs dans les logiciels et sur le hardware, l'analyse statique pour la compilation, la vérification des processus métiers, la vérification de système critiques embarqués (avions, trains, satellites, etc...), l'industrie des circuits intégrés, etc...

Dans la littérature [Model-checking Tools], il existe plusieurs model-checkers. Nous avons choisi CPNTools et HELENA dont nous détaillons leur usage ; dans les sections 3.3.1 et 3.3.2.

## 3.2 Spécification formelle des besoins d'un TramwayNET

Lorsqu'on modélise un système, il est d'usage de réfléchir d'abord à ce qu'on aimerait vérifier avant de créer des modèles. Dans cette section, nous allons définir les propriétés que nous aimerions garantir sur notre TramwayNET.

Le tramway cumule tous les atouts d'un mode de transport en commun efficace : *Accessibilité*, *Securité*, *Protection de l'environnement*, *Regularité*, etc...

Dans ce mémoire, nous définissons des propriétés pour la sécurité et la regularité sur le réseau :

- **SECURITE** : Les trams couvrent 21.1 kilomètres [1.2] sur les 384.6 kilomètres couverts par tout le réseau des transports publics de la ville de Genève. il est important de s'assurer que ces véhicules sont bien répartis pour une bonne cohabitation sur tout le réseau.
  Dans ce mémoire, nous analysons les croisements qui existent sur le réseau et notre objectif , c'est de proposer un modèle de tramwayNET qui garanti qu'il n'y ai jamais de collision entre les trams.

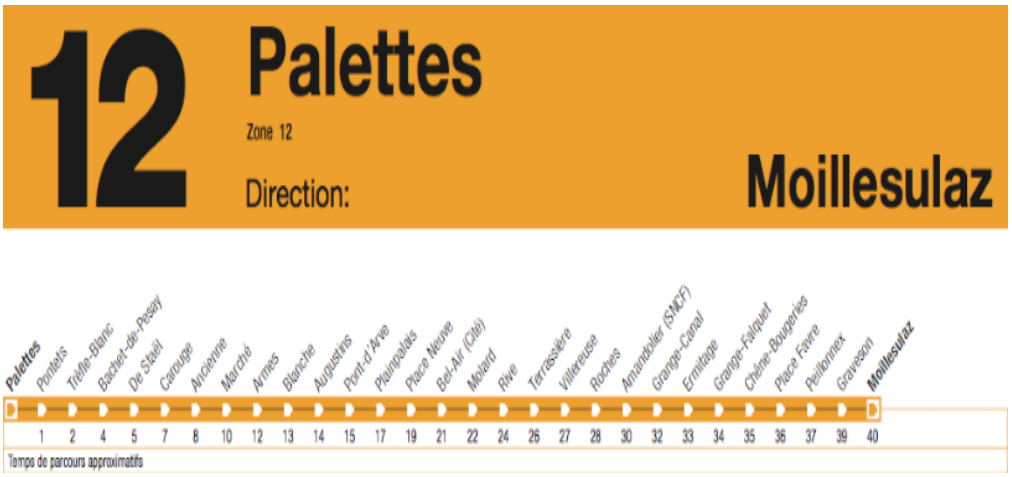
– **REGULARITE** : Le tram, c'est l'assurance de ne jamais arriver en retard. Sauf, bien sûr, en cas de perturbations inattendues sur le réseau. L'ensemble des véhicules du réseau de transports publics de la ville de Genève parcourt plus de 20 millions de kilomètres par années. Un tram circule environ toutes les 6 à 8 minutes en journée et toutes les 15 à 30 minutes en soirée, les dimanches et les jours fériés. La priorité donnée aux trams par rapport aux autres modes de déplacement contribue à réduire les distances.

  Nous analysons conjointement le TramwayNET de la ville de Genève et les tables horaires pré-définies par les TPG, ensuite nous créons et validons un modèle de TramwayNET pour la propriété "Les heures d'arrivée des trams aux différents arrêts sont conforment aux tables horaires". Les tables horaires (figure 3.2) donnent les valeurs approximatives, des heures d'arrivée à chaque arrêts de trams, à partir d'une heure de départ fixée. Par exemple pour aller de l'arrêt "*Palettes*" à l'arrêt "*Treffle-blanc*" un tram met approximativement deux minutes, pour aller de l'arrêt "*Palettes*" à l'arrêt "*Carouge*" un tram met approximativement sept minutes, etc...

  Nous allons attribuer des valeurs concrètes à ces délais approximatifs. On considère que chaque délai appartient à un intervalle discret de valeurs. La borne inférieure, respectivement supérieure de cet intervalle représente le délai minimum, respectivement maximum, pour aller d'un arrêt à un autre.

  Pour un point de départ A et un point d'arrivée B, la valeur de l'heure d'arrivée au point B se calcule en fonction de l'heure de départ au point A e de la somme des valeurs des délais pour aller du point A au point B. La discrétisation de l'intervalle des valeurs possibles des délais permet de modéliser l'indéterminisme sur la valeur de l'heure d'arrivée d'un tram à un arrêt. Ce scénario correspond bien au fonctionnement dynamique du réseau qui est régulièrement exposé aux perturbations de natures diverses.

  Pour la simulation du réseau, on considère, par l'hypothèse, que l'heure d'arrivée d'un tram à un arrêt appartient à un intervalle de valeurs connues, dans le cas contraire, un message d'alerte est généré.



# 12 Palettes

Zone 12

Direction: **Moillesulaz**

| Palettes | Pontets | Trèfle-Blanc | Bachet-de-Pesay | De Staël | Canouge | Ancienne | Marché | Armes | Blanche | Augustins | Pont-d'Arve | Plainpalais | Place Neuve | Bel-Air (Cité) | Molard | Rive | Terrassière | Villereuse | Roches | Amandolier (SMCF) | Grange-Canal | Ermitage | Grange-Falquet | Chêne-Bourgeries | Place Fèvre | Peillonnex | Gravesson | Moillesulaz |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 4 | 5 | 7 | 8 | 10 | 12 | 13 | 14 | 15 | 17 | 19 | 21 | 22 | 24 | 26 | 27 | 28 | 30 | 32 | 33 | 34 | 35 | 36 | 37 | 39 | 40 |

Temps de parcours approximatifs

a: Départ à l'arrêt de la ligne 13, situé dans la boucle des Palettes.
c: Arrêts desservis jusqu'à Bachet-de-Pesay. Ne circule pas les nuits du vendredi au samedi.
v: Arrêts desservis jusqu'à Bachet-de-Pesay.

## Horaire valable jusqu'au 23 août 2009

L'HORAIRE GRANDES VACANCES est appliqué du 27 juin au 23 août 2009.
L'HORAIRE PETITES VACANCES est appliqué du 20 au 26 juin 2009.
L'HORAIRE DIMANCHE est également appliqué le 1er août 2009.

### Horaire Grandes Vacances

| Lundi / Vendredi | Samedi |
|---|---|
| **05**h a a a 25 38 50 | a a 42 57 |
| **06**h 01 17 28 40 51 | 12 26 41 56 |
| **07**h 02 13 25 37 48 59 | 11 26 41 56 |
| **08**h 11 22 34 47 59 | 11 26 41 56 |
| **09**h 10 a v 22 31 34 47 59 | 09 21 33 46 59 |
| **10**h 11 23 36 48 | 11 23 36 49 |
| **11**h 00 12 25 37 49 | 01 14 26 39 51 |
| **12**h 01 14 26 38 50 | 04 16 29 41 54 |
| **13**h 03 15 27 39 52 | 06 19 31 44 56 |
| **14**h 04 16 29 40 53 | 09 21 33 46 58 |
| **15**h 05 17 30 42 53 | 11 23 36 48 |
| **16**h 05 15 27 38 50 | 01 13 26 38 51 |
| **17**h 02 14 25 36 48 | 03 16 28 41 53 |
| **18**h 00 11 23 34 46 58 | 06 19 32 45 57 |
| **19**h 10 22 34 47 | 10 22 34 47 |
| **20**h 02 v 04 av 07 v 15 17 av 18 av 31 v 35 46 av 48 | 00 15 30 46 |
| **21**h 01 20 v 22 av 28 40 | 01 20 v 23 v 31 40 |
| **22**h 00 av 03 av 17 20 40 | av 00 14 20 40 |
| **23**h 00 20 40 | 00 20 40 |
| **00**h 00 c 17 c 37 c 58 | 00 |
| **01**h c 03 | |

### Horaire Petites Vacances

| Lundi / Vendredi | Samedi | Dimanche / Fériés |
|---|---|---|
| a a a 25 38 50 | a a 42 57 | a 39 **05**h |
| 01 17 28 40 51 | 12 26 41 56 | a 01 20 40 **06**h |
| 02 13 25 37 48 59 | 11 26 41 56 | 00 20 40 **07**h |
| 11 22 34 47 59 | 11 26 41 56 | 00 20 41 59 **08**h |
| 10 22 av 31 34 47 59 | 09 21 33 46 59 | a 14 28 43 58 **09**h |
| 11 23 36 48 | 11 23 36 49 | 13 28 43 58 **10**h |
| 00 12 25 37 49 | 01 14 26 39 51 | 13 28 43 58 **11**h |
| 01 14 26 38 50 | 04 16 29 41 54 | 13 28 43 58 **12**h |
| 03 15 27 39 52 | 06 19 31 44 56 | 13 28 43 58 **13**h |
| 04 16 29 40 53 | 09 21 33 46 58 | 13 28 43 58 **14**h |
| 05 17 30 42 53 | 11 23 36 48 | 13 28 43 58 **15**h |
| 05 15 27 38 50 | 01 13 26 38 51 | 13 28 43 58 **16**h |
| 02 14 25 36 48 | 03 16 28 41 53 | 13 28 43 58 **17**h |
| 00 11 23 34 46 58 | 06 19 32 45 57 | 13 28 43 58 **18**h |
| 10 22 34 47 | 10 22 34 47 | 13 30 45 **19**h |
| 02 v 04 av 07 v 15 17 av 18 31 av 35 v 46 av 48 | 00 15 30 46 | 00 15 30 46 **20**h |
| 01 20 v 22 av 28 40 | 01 20 v 23 v 31 40 | v 01 20 23 av 28 40 **21**h |
| av 00 03 av 17 20 40 | av 00 14 20 40 | av 00 17 20 40 **22**h |
| 00 20 40 | 00 20 40 | 00 20 40 **23**h |
| 00 c 17 c 37 c 58 | 00 | v 00 17 v 37 v 58 **00**h |
| c 03 | | av 03 **01**h |

FIGURE 3.2 – Table horaire Ligne 12, Direction "Palette" ⟶ "Moillesulaz"

### 3.2.1 Expression des propriétés en langage naturel

Nous présentons dans ce qui suit les deux principales propriétés d'un réseau de transport public par trams que nous allons analyser. Ces propriétés sont exprimées en langage naturel :

↪ Propriété 1 : SECURITE

"***Les trams ne se croisent jamais***".

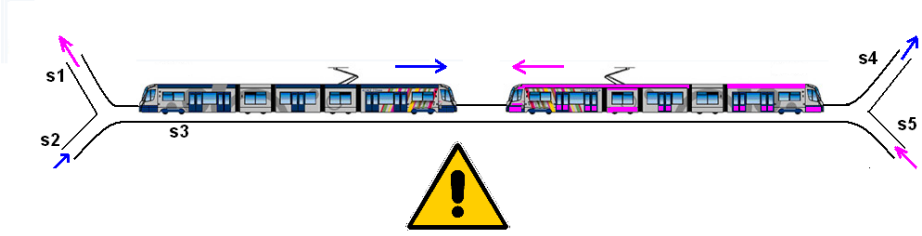



FIGURE 3.3 – Scénario Propriété securité : Collision entre deux trams

↪ Propriété 2 : REGULARITE

"***Les heures d'arrivée des trams aux différents arrêts sont conformes aux tables horaires.***"

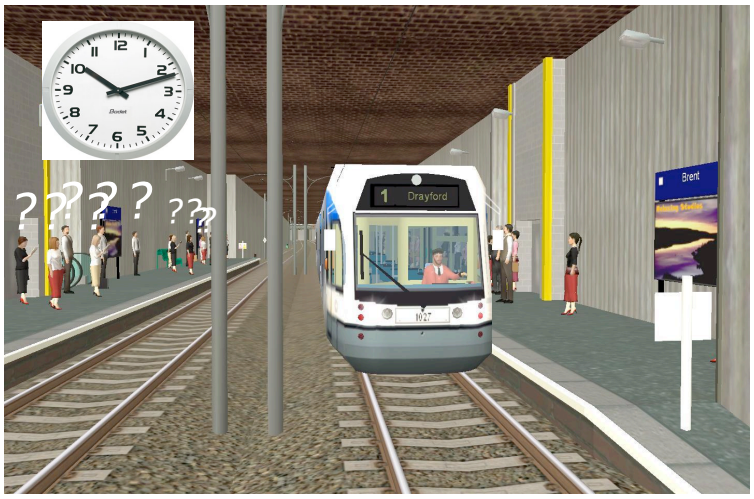



FIGURE 3.4 – Scénario Propriété régularité : Retard de tram à un arrêt

### 3.2.2 Expression des propriétés en logique mathématique

La logique mathématique est la science qui est à la base de la formalisation de différents domaines scientifiques, elle ne s'intéresse pas au contenu des objets étudiés mais à leur structure formelle. En réalité, une affirmation n'est vraie que dans un domaine particulier, il faut donc préciser le domaine de validité et préciser pour quels éléments c'est vrai. Dans ce contexte, la logique mathématique s'avère indispensable pour l'expression de nos propriétés :

↪ Propriété 1 : SECURITE

$\forall s \in \boldsymbol{S}$, card(s) $\leq 1$.

$\boldsymbol{S}$ est l'ensemble des segments ("TrackElement") qui composent le réseau.

*card* est l'opérateur de cardinalité [I].

↪ Propriété 2 : REGULARITE

$$\begin{cases} \forall t_{id} \in Trams, \\ \forall P_n, P_m \in Places | n, m \in \mathbb{N}^*, n < m, \\ \forall d_{P_i P_{i+1}} \in D, \\ \forall \delta^-, \delta^+ \in D_\delta, \\ Harr_{t_{id}}(P_m) \in [Harr_{t_{id}}(P_n) + \sum_{i=n}^{m} d_{P_i P_{i+1}} - \sum_{i=n}^{m} \delta^-_{P_i P_{i+1}}, \\ \qquad\qquad Harr_{t_{id}}(P_n) + \sum_{i=n}^{m} d_{P_i P_{i+1}} + \sum_{i=n}^{m} \delta^+_{P_i P_{i+1}}] \end{cases}$$

$Trams$ est l'ensemble identificateurs (noms) des trams du réseau.

***Places*** est l'ensemble des places ("TrackElement") du réseau.

$\boldsymbol{D} \in \mathbb{N}^*$ est l'ensemble des delais pour aller d'un TrackElement à un autre.

$\boldsymbol{D_\delta} \in \mathbb{N}^*$ est l'ensemble des entiers positifs, qui sont des paramètres permettant de calculer les delais minimum et maximum pour aller d'un TrackElement à un autre.

I. En mathématiques, la **cardinalité** est une notion de taille pour les ensembles

$P_n$ et $P_m$ sont respectivement les places de départ et d'arrivée d'un tram .

$d_{P_iP_{i+1}}$ est le delai entre la place $P_i$ et $P_{i+1}$

$Harr_{t_{id}}(P_n)$ est l'heure d'arrivée du tram $t_{id}$ dans la place $P_n$

## 3.3 Validation d'un modèle de TramwayNET

Nous avons imaginé et construit une petite maquette de tramwayNET pour appliquer les techniques de model-checking. Cette maquette que nous avons baptisée "***SIMPLETramway***" est illustrée sur la figure 3.5 et est composée des éléments suivant :

- La place "STOCK" pour l'insertion des trams dans le réseau.
- La place "C22" qui représente un croisement 2 entrées et 2 sorties.
- Les places "P12" et "P13" qui sont respectivement les segments des lignes 12 et 13
- un ensemble de transitions pour lier ces places.

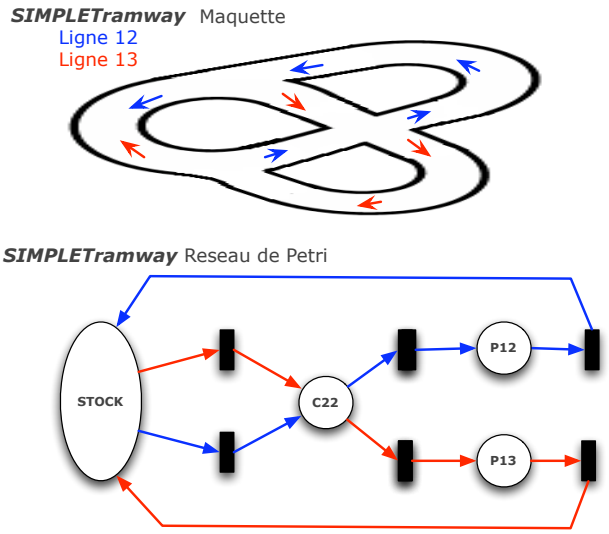



FIGURE 3.5 – maquette de tramwayNET ("***SIMPLETramway***") pour l'application du model-checking et son RdP équivalent

Dans ce mémoire, nous avons choisi deux model-checkers ***CPNTools*** et ***HELENA***, que nous utilisons pour valider nos modèles par rapport aux propriétés prédéfinies sur notre TramwayNET.

– CPNTools est un logiciel gratuit qui, après inscription, peut être téléchargé à l'adresse suivante : http ://wiki.daimi.au.dk/cpntools/cpntools.wiki

– HELENA est un logiciel gratuit, disponible selon les conditions de la GNU General Purpose Licence, il peut être téléchargé à l'adresse suivante : http ://helena.cnam.fr

### 3.3.1 Application du Model-checker CPNtools

Parmis les outils qui existent dans la littérature [Model-checking Tools], nous avons opté pour CPNTools, successeur de design/CPN, qui a été développé par des chercheurs au sein de l'université Aarhus au Danemark.

Notre choix s'est porté sur cet outil de vérification, tout d'abord parce qu'il permet grâce à une sémantique du temps formellement définie ([Jen 97]), de gérer des évènements temporels et aussi, à cause sa facilité d'utilisation. Celle-ci se justifie par son interface ergonomique, pour l'édition et l'analyse des modèles de RdP. Voici quelques avantages et qualités de CPNTools qui nous ont convaincus de son utilité :

– CPNTools permet de simuler les "timed CP nets" ou réseaux de Petri colorés temporisés (Gestion du temps et Modélisation des attentes).

– Il est dédié aux réseaux de Petri colorés et hiérarchiques (HCPN), ce qui correspond bien au concept de modularité que nous voulons exploiter.

– Il offre la possibilité de simuler les HCPNs et comme résultat de simulation, on obtient un rapport de la trace d'éxécution.

– Il offre la possibilité d'analyser les propriétés spécifiques des RdPs.

– les applications industrielles sont de plus en plus croissantes.

Simulation de la maquette "***SIMPLETramway***" (3.5) dans CPNTools

La simulation demeure un moyen efficace pour "*valider*" des systèmes. Les résultats de simulation permettent d'analyser des situations pouvant apparaître dans la pratique.

Considérons la maquette "***SIMPLETramway***". Nous allons créer un modèle de réseau de Petri coloré qui modélise le scénario suivant :

*Les trams se trouvent sur le réseau et circulent en boucle toute la journée. Chaque trams peut faire un ou plusieurs cycle dans un journée. Les cycles sont limités selon l'état du réseau. Le réseau comporte deux lignes de trams (la ligne 12 et la ligne 13). Chaque tram possède un attribut qui définit la ligne de trams à laquelle il appartient. Les transitions pour aller d'une place à une autre ont des délais approximatifs que nous modélisons par un intervalle de trois valeurs discrètes : On rajoute sur le réseau de l'indéterminisme pour le choix des délais.*

#### 3.3.1.1 Modélisation

Pour la modélisation, nous commençons par déclarer des couleurs et les variables qui modélisent les ressources de notre réseau. Ces déclarations sont illustrés par la figure 3.6. Ensuite, nous allons hiérarchiser le modèle de RdP équivalent à cette maquette. On utilise le toolbox "Hierarchy", le concept de substitutions de transitions et le concept de sous-pages disponibles dans CPNTools. Le résultat est un réseau de petri hiérarchique(HCPN) dont les pages sont illustrées par les figures 3.7, 3.8 et 3.9. Pour aller de la place STOCK à la place C22, de la place C22 à la place P12, de la place C22 à la place P13, de la place P13 à la place STOCK et de la place P12 à la place STOCK, on crée trois transitions avec des délais différents : Nous modélisons ainsi l'*indéterminisme* sur les délais. Ces délais seront additionnés aux heures de départ de chaque tram dans une place en amont et le résultat de cette addition sera l'heure d'arrivée du tram dans la place en aval. Pour le moment, nous avons modélisé le comportement du réseau avec des transitions *immédiates* (délai = 0)

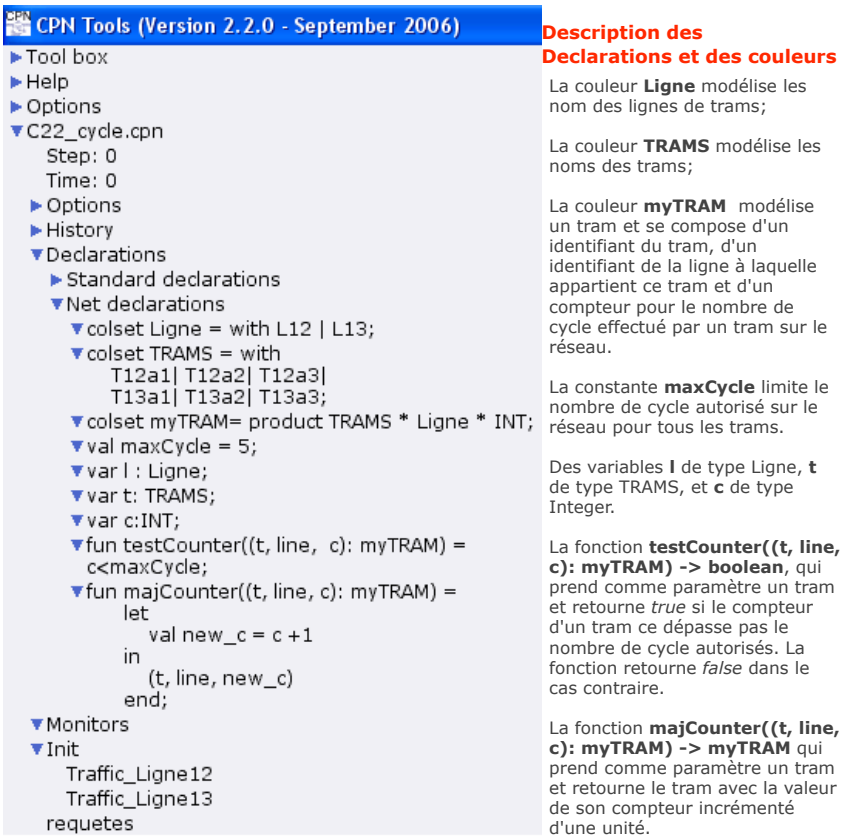



FIGURE 3.6 – "***SIMPLETramway***" dans CPNTools : Declaration des variables et des couleurs

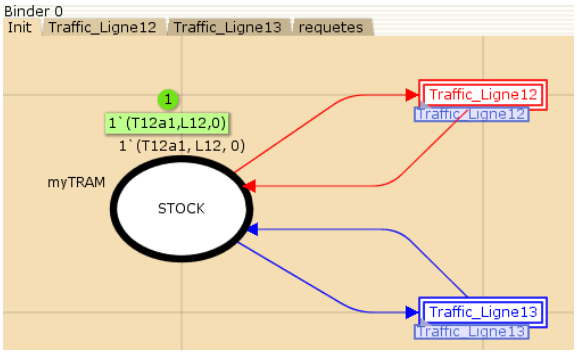



FIGURE 3.7 – "***SIMPLETramway***" dans CPNTools : La page "d'initialisation"

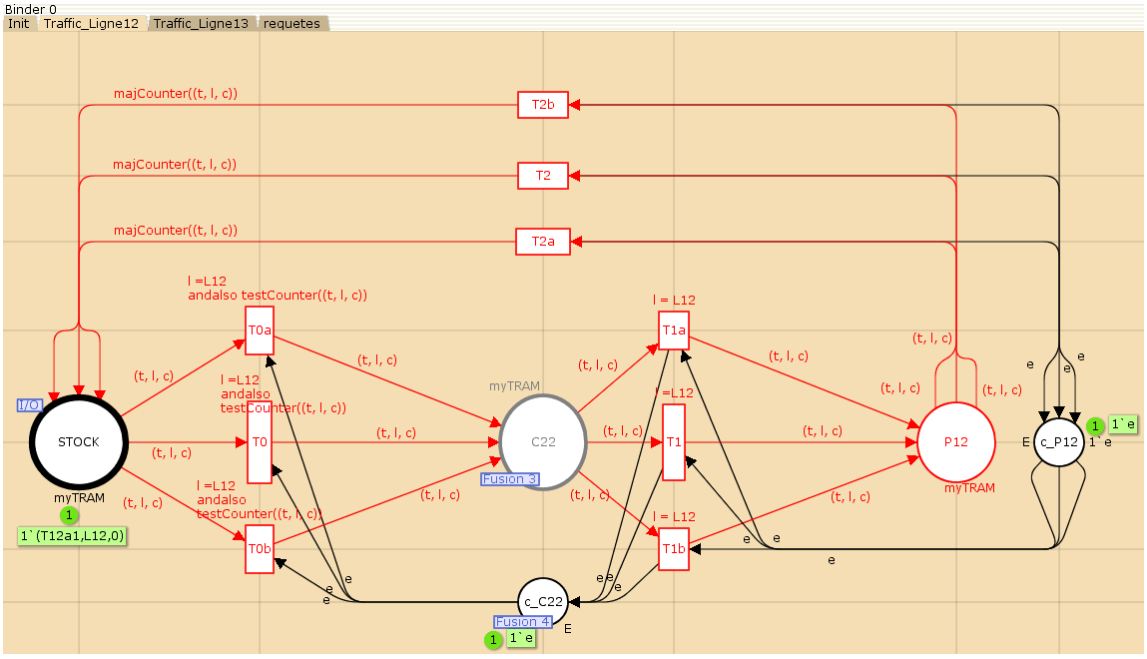



FIGURE 3.8 – "***SIMPLETramway***" dans CPNTools : La page du "traffic Ligne12"

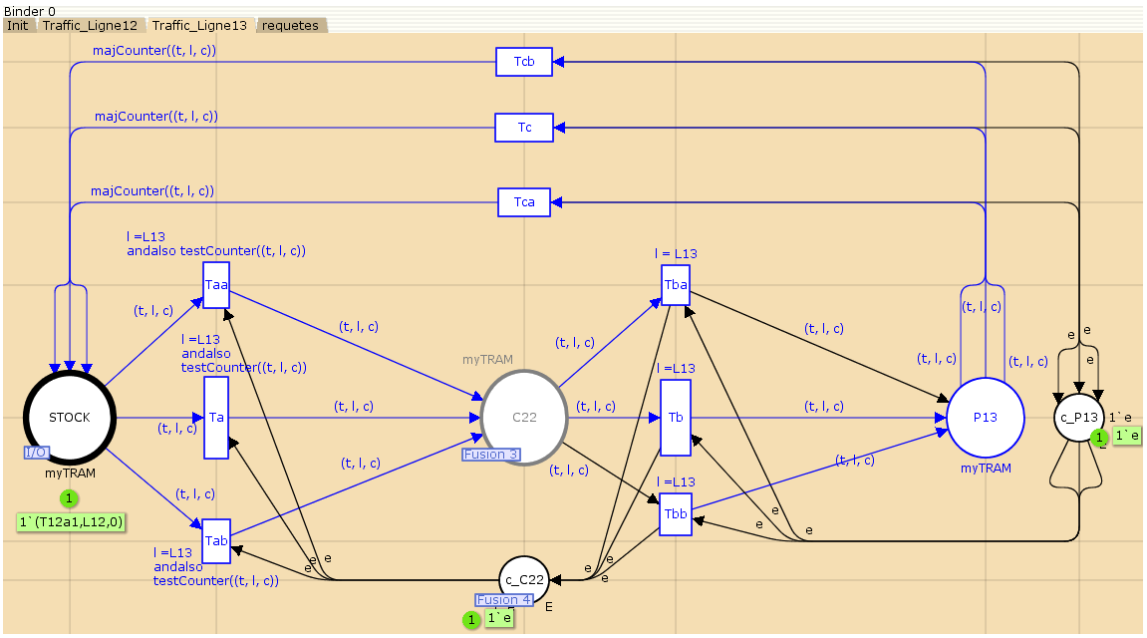



FIGURE 3.9 – "***SIMPLETramway***" dans CPNTools : La page du "traffic Ligne13"

#### 3.3.1.2 Expérimentation

Nous avons expérimenté l'exemple du réseau de Petri coloré ***SIMPLE-Tramway*** pour plusieurs ressources (trams) dans le réseau. Nous résumons dans le tableau 3.1 les données recueillies au cours de ces expérimentations.

| ***SIMPLETramway*** dans CPNTools | | | | |
|---|---|---|---|---|
| Nombre de cycles | Nombre de trams | Nombre d'états | Nombre d'arcs | **Status** |
| 1 | 1 | 4 | 9 | full |
| | 2 | 14 | 48 | full |
| | 3 | 50 | 246 | full |
| | 4 | 160 | 912 | full |
| | 5 | 472 | 2904 | full |
| 2 | 1 | 7 | 18 | full |
| | 2 | 41 | 156 | full |
| | 3 | 247 | 1350 | full |
| | 4 | 1281 | 7992 | full |
| | 5 | 5967 | 39690 | full |
| 3 | 1 | 10 | 27 | full |
| | 2 | 82 | 324 | full |
| | 3 | 694 | 3942 | full |
| | 4 | 4960 | 31968 | full |
| | 5 | *20630* | *134979* | PARTIAL |
| 4 | 1 | 13 | 36 | full |
| | 2 | 137 | 552 | full |
| | 3 | 1493 | 8652 | full |
| | 4 | 13585 | 89040 | full |
| | 5 | - | - | PARTIAL |
| 5 | 1 | 16 | 45 | full |
| | 2 | 206 | 840 | full |
| | 3 | 2746 | 16110 | full |
| | 4 | - | - | PARTIAL |
| | 5 | - | - | PARTIAL |

TABLE 3.1 – Résultat de simulation de SIMPLETramway dans CPNTools : Espace d'états

Les résultats de simulation dans le tableau 3.1 précédent indiquent que pour 5 ressources (trams) et pour 3 cycles dans le réseau, le rapport de l'espace d'états généré par l'outil mentionne le status *PARTIAL*. La colonne du tableau pour l'attribut (status = PARTIAL) indique qu'au delà de 13585 états atteignables du réseau, on ne peut plus faire de la validation du modèle pour une propriété donnée : L'espace d'état conservé en mémoire par l'outil est incomplet.

Pour l'analyse des propriétés sur le modèle, la figure 3.10 montre le résultat obtenu pour la validation de la propriété 1 ("Les trams ne se croisent jamais"). On utilise les fonctions "*PredAllNodes*" et "*size*" qui permettent de d'explorer les états du réseau. On cherche la liste des noeuds du réseau tel que les places C22, P12 et P13 ont plus d'une ressource(tram).

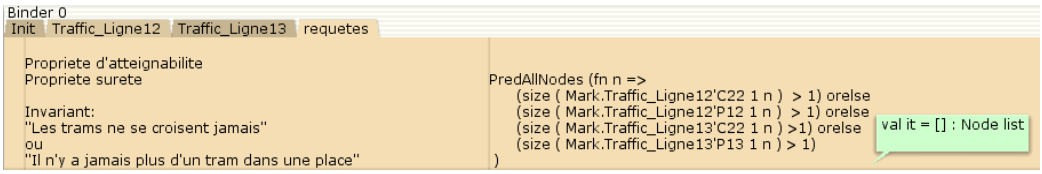



FIGURE 3.10 – "***SIMPLETramway***" : Résultat exécution des requêtes en ML sous CPNTools

La limitation de CPNTools due à un nombre, assez réduit, d'états conservés en mémoire après simulation du réseau, nous a amené à choisir un autre model-checker pour faire de la validation, car étant donné son comportement indéterministe, le réseau que nous sommes en train d'analyser peut potentiellement avoir un "*grand*" nombre d'états. La section suivante présente le modélisation du réseau SIMPLETramway avec l'outil HELENA.

### 3.3.2 Application du Model-checker HELENA

HELENA est un model-checker pour les réseaux de Petri colorés. C'est l'acronyme de Hihg LEvel Net Analyser. Il fournit à l'utilisateur un langage de description des réseaux, un langage de spécification de propriétés et divers algorithmes pour vérifier ces propriétés.

#### 3.3.2.1 Modélisation

**Langage de description de HELENA**

⊙ Les réseaux HELENA sont décrits à l'aide :

– de types de données de haut niveau (types structurés, types vecteurs...)

– de fonctions écrites dans un langage impératif.

⊙ Les propriétés sont spécifiées à l'aide d'itérateurs(p.ex : forall, exists ) qui permettent d'évaluer le contenu d'une place.

Le model-checker HELENA modélise les réseaux de Petri colorés sous format textuel, il ne possède pas d'interface graphique. La figure 3.12 représente le modèle graphique de RdP de la maquette ***SIMPLETramway*** (figure 3.5) que nous avons implémenté dans le format textuel qui respecte la syntyxe d'HELENA. Notre objectif ici est de comparer les deux model-checker HELENA et CPNTools, ainsi le RdP que nous présentons dans cette section est le même que celui que nous avons implémenté avec le model-checker CPN-Tools.

#### 3.3.2.2 Expérimentation

Nous avons expérimenté l'exemple du réseau de Petri coloré ***SIMPLE-Tramway*** pour plusieurs ressources (trams) dans le réseau. Nous avons résumé dans le tableau 3.2 les données recueillies au cours de ces expérimentations.

Une caractéristique importante du model-checker HELENA est la façon dont il représente l'espace d'états : Il utilise la méthode dite des "Δ_marquages" [Eva 04]. L'idée est de stocker un grand nombre d'états d'une manière non

Exemple

```
type tramID : enum (T12a1, T12a2, T12a3);

type Lines : enum (L12, L13);

subtype naturals : int range 0..100000;

// Une structure de données pour chaque tram : <( name, line, counter)>;
type tram :
        struct {
                tramID   name;
                Lines line;
                naturals counter; // Pour compter le nombre de cycles
        };

//une fonction qui incremente le compteur de tram

function majCounter(tram t) -> tram{
        return{t.name, t.line, t.counter + 1};
}

//les noeuds
place STOCK {
    dom: tram;
    init: <( {T12a1, L12, 0} )> ;
    capacity : 1;
}

transition T2 {
    in {
        P12 : <( t )>;
     }
     out {
        STOCK : <( majCounter(t) )>;
        c_P12 : epsilon ;
     }
 }

// Une propriété:
// Cherche l'état du système tel que la cardinalité de la place P12 > 1

  reject P12'card > 1

.....
```

FIGURE 3.11 – Le language de description d'Helena : Exemple

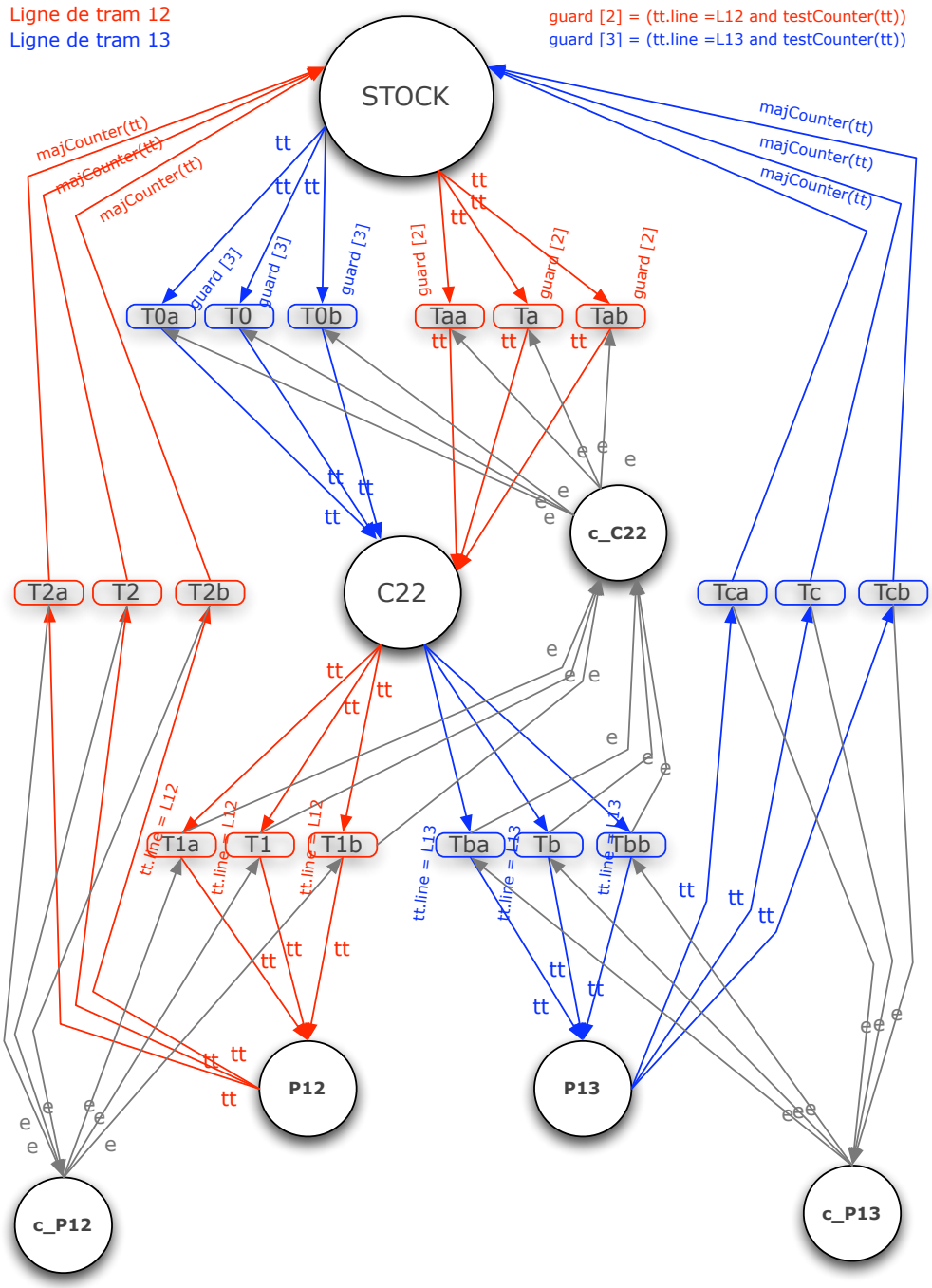



FIGURE 3.12 – "***SIMPLETramway***" dans HELENA

explicite en ne stockant que les références sur les autres états.

Nous avons expérimenté cette la manière dont HELENA stocke les états du réseau via les commandes ***–static-reductions=0*** et ***–partial-order=0*** disponible sous HELENA. Tout d'abord, des réductions statiques sont appliquées sur le réseau, ensuite, pendant la recherche, la méthode *partial-order* est utilisée pour éliminer l'exploration des chemins redondants. On remarque une nette réduction de la taille du graphe des marquages accessibles. Pour chaque exécution, avec et sans réduction partielle, nous donnons le tableau 3.2 du nombre de configurations possibles du réseau, le nombre d'arcs (ou transitions) entres ces états, et le temps nécessaire d'analyse de l'espace d'états (compilation du réseau + Recherche en profondeur).

### 3.3.3 CPNtools vs HELENA

Malgré les qualités de l'outil CPNTools, nous avons relevé deux inconvénients majeurs lors de notre apprentissage :

- Le Standard ML.
- La taille de l'espace d'états stockée en mémoire.

`Le Standard ML`

Les propriétés exprimées sous CPNTools sont spécifiées en langage ML, ce dernier est un langage fonctionnel. Les seuls objets manipulés dans les langages fonctionnels sont des fonctions ou des constantes : On ne fait pas d'assignation. Ces langages nécessitent un apprentissage relativement long.

`La taille de l'espace d'états stockée en mémoire`

Sous CPNTools, la taille de l'espace d'état atteignable stockée en mémoire est relativement "faible". Les résultats du tableau 3.1 indiquent que lorsqu'on initialise le RdP hiérarchique de la maquette "***SIMPLETramway***" avec un seul tram, CPNTools génère le graphe d'état total (*status = full*). Lorsqu'on augmente progressivement le nombre de ressources (trams) et le nombre maximum de cycle par ressource dans le réseau, CPNTools génère le graphe d'état partiel (*status = partial*). Avec ce graphe, on ne peut plus valider notre modèle pour une propriété prédéfinie.

| ***SIMPLETramway*** dans HELENA | | | | | | | |
|---|---|---|---|---|---|---|---|
| NO reduction | | | | | Reduction | | |
| Nombre de cycles | Nombre de trams | Nombre d'états | Nombre d'arcs | Temps | Nombre d'états | Nombre d'arcs | Temps |
| 1 | 1 | 4 | 9 | 17s | 3 | 12 | 29s |
| | 2 | 14 | 48 | 16s | 8 | 48 | 29s |
| | 3 | 50 | 246 | 16s | 20 | 144 | 29s |
| | 4 | 160 | 912 | 16s | 48 | 384 | 29s |
| | 5 | 472 | 2904 | 16s | 112 | 960 | 29s |
| 2 | 1 | 7 | 18 | 16s | 5 | 24 | 29s |
| | 2 | 41 | 156 | 16s | 21 | 144 | 29s |
| | 3 | 247 | 1350 | 16s | 81 | 648 | 29s |
| | 4 | 1281 | 7992 | 16s | 297 | 2592 | 29s |
| | 5 | 5967 | 39690 | 16s | 1053 | 9720 | 29s |
| 3 | 1 | 10 | 27 | 16s | 7 | 36 | 29s |
| | 2 | 82 | 324 | 16s | 40 | 288 | 29s |
| | 3 | 694 | 3942 | 16s | 208 | 1728 | 29s |
| | 4 | 4960 | 31968 | 16s | 1024 | 9216 | 29s |
| | 5 | 31456 | 215136 | 16s | 4864 | 46080 | 29s |
| 4 | 1 | 13 | 36 | 16s | 9 | 48 | 29s |
| | 2 | 137 | 552 | 16s | 65 | 480 | 29s |
| | 3 | 1493 | 8652 | 16s | 425 | 3600 | 29s |
| | 4 | 13585 | 89040 | 16s | 2625 | 24000 | 29s |
| | 5 | 108925 | 755700 | 18s | 15625 | 150000 | 30s |
| 5 | 1 | 16 | 45 | 16s | 11 | 60 | 29s |
| | 2 | 206 | 840 | 16s | 96 | 720 | 29s |
| | 3 | 2746 | 16110 | 16s | 756 | 6480 | 29s |
| | 4 | 30336 | 200880 | 16s | 5616 | 51840 | 29s |
| | 5 | 293976 | 2057400 | 25s | 40176 | 388800 | 31s |

TABLE 3.2 – Résultat de simulation de SIMPLETramway dans HELENA : Espace d'états

Pour le réseau de transport public par trams que nous sommes en train d'analyser, le model-checker HELENA s'avère être plus adapté pour gérer des espaces d'états relativement grands. HELENA peut gérer un espace d'états de $10^8$ états.

Cependant HELENA ne possède aucune sémantique temporelle. Pour palier à ce manque, nous avons modélisé **l'écoulement du temps** conforme à la simulation du temps sous CPNTools (figure 2.5). L'écoulement du temps sous CPNTools garanti un ordre croissant d'utilisation des jetons colorés par rapport à une horloge globale (temps du modèle). Dans CPNTools, l'information temporelle est introduite sur le jeton grâce au "timeStamp". Le jeton coloré qui possède le plus petit *timeStamp* dans l'ensemble des *timeStamp* est "prioritaire". Lorsque la valeur de tous les *timeStamp* est supérieure à l'horloge globale. Le simulateur incrémente son horloge d'une unité jusqu'à atteindre la valeur du minimum des *timeStamp*, ce qui permet d'éviter un deadlock du réseau.

Par exemple, pour modéliser l'écoulement du temps du RdP SIMPLE-Tramway avec HELENA, nous avons ajouté certains éléments au réseau :

- Une Place globale "**P_all_trams**" contenant le vecteur des tous les trams du réseau.

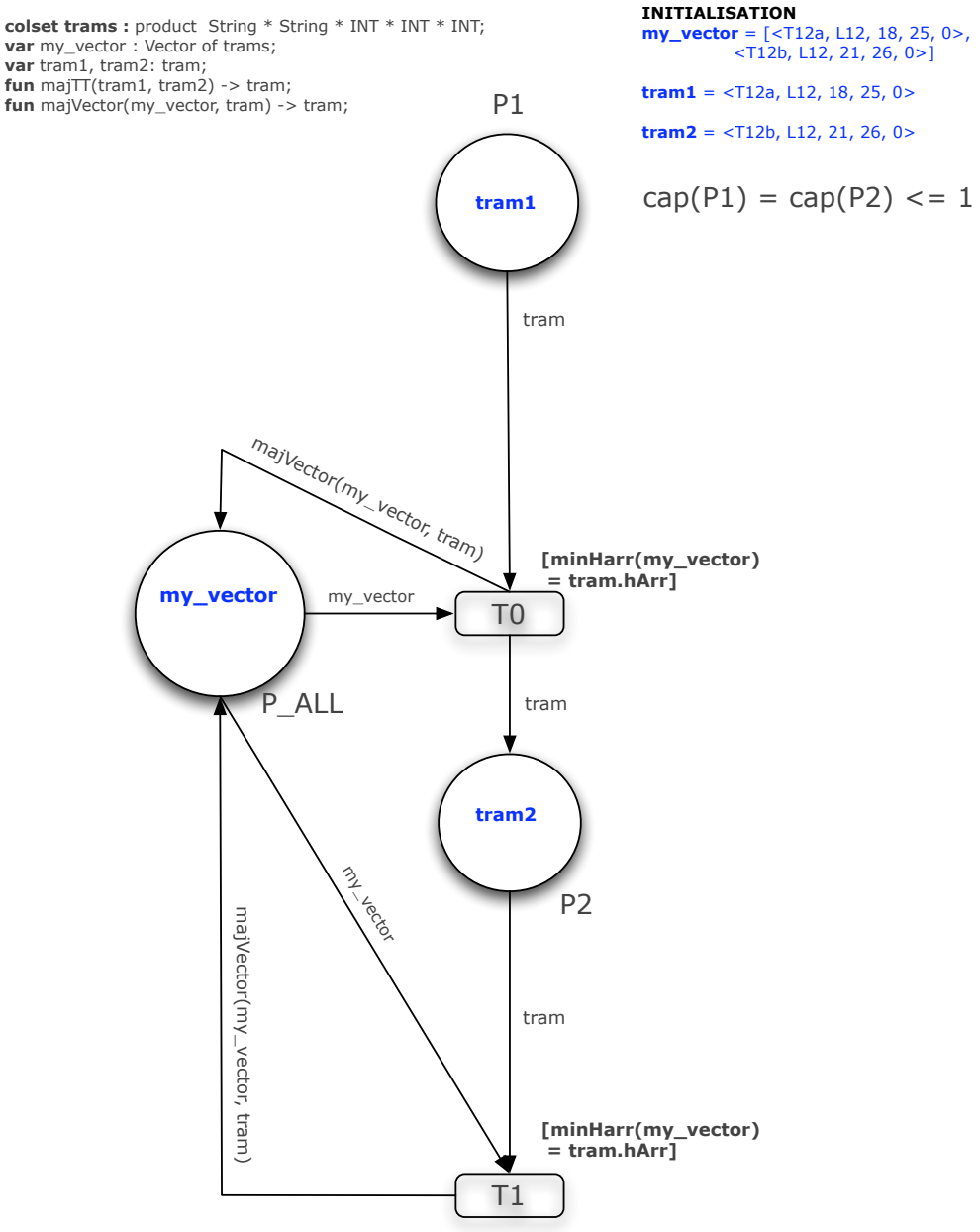
- Des attributs **hDep** et **hArr** pour un tram. Ces attributs sont ajoutés à la structures d'un tram et modélisent respectivement l'heure départ et d'arrivée du tram dans une place.

- Des transitions **Tx_wacthDOG** entre chaque place qui possède une condition de tir sur les heures d'arrivée des jetons colorés dans les places en amont et en aval. Ces transitions permettent d'éviter des *deadlock* sur le réseau.

- Des fonctions sur les arcs qui mettent à jour la valeur de l'heure d'arrivée dans les places.

Exemple d'exécution de l'écoulement du temps sous HELENA

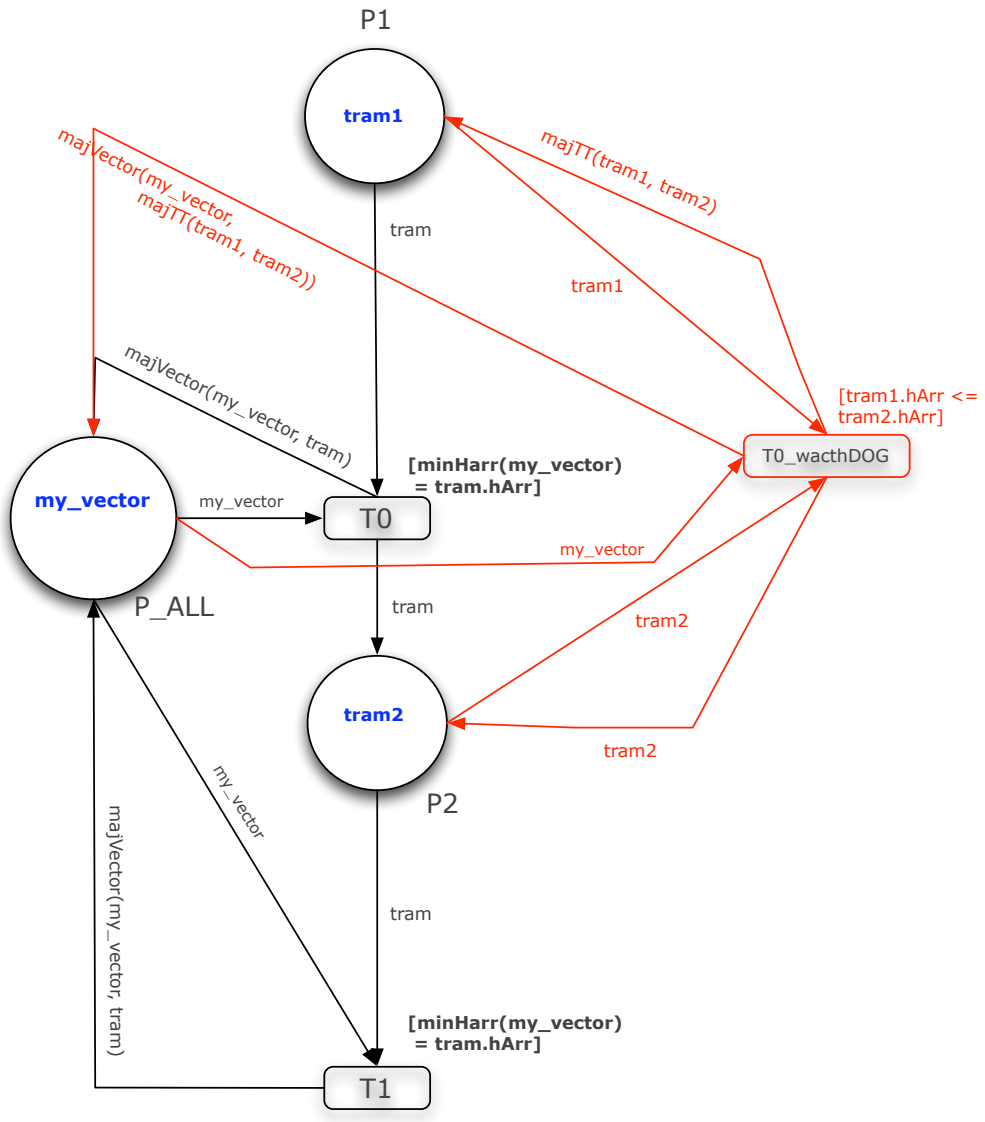
Les reseaux de Petri colorés des figures 3.13 3.14 illustrent deux scénarios, le premier avec deadlock et le deuxième sans deadlock.




T0 n'est pas tirable car cap(P2) <= 1, bien que (hArr1 = 25 ) < (hArr2 = 26 )

DEADLOCK

FIGURE 3.13 – Modélisation de l'écoulement du temps sous HELENA : Deadlock




T0 n'est pas tirable car cap(P2) <= 1, bien que (hArr1 = 25 ) < (hArr2 = 26 )

On tire T0_watchDOG, Résultat de la fonction majTT(tram1, tram2) :
hArr1 = hArr1 + (hArr2 - hArr1) + 1 = 27, le jeton dans la place P2 est "prêt" à être consommé (tir le la transition T1)

FIGURE 3.14 – Modélisation de l'écoulement du temps sous HELENA : Pas de Deadlock

Sur le RdP de la figure 3.13, on considère une place globale PALL, une transition T0 qui a une condition de tir [minHArr(Vector) = *tram1.hArr1*] (On vérifie que l'heure d'arrivée du tram1 soit la plus petite valeur des heures d'arrivée des trams du vecteurs), une transition T1 avec la même condition de tri, une place P1 en amont de T0 et une place P2 en aval de TO. Les places ont une capacité limitée à un et possèdent des jetons colorés de couleur *<( name, line, hDep, hArr, counter)>*. Même si la condition de tir de la transition T0 est respectée, elle n'est pas tirable car la capacité de la place P2 est limitée à un : *On a un deadlock sur réseau*. Le marquage du réseau à cet instant est :

$M_1(PALL)$ = [<T12a, L12, 18, 25, 0> + <T12b, L12, 21, 26, 0>]

$M_1(P1)$ = <T12a, L12, 18, 25, 0>

$M_1(P2)$ = <T12b, L12, 21, 26, 0>

Sur le RdP de la figure 3.14, on a le même réseau auquel on ajoute une transition **T_wacthDOG_0** qui a une condition de tir [*tram1.hArr1* $\leq$ *tram2.hArr2*] . Cette transition consomme les jetons colorés *tram1 = <( name1, line1, hDep1, hArr1, counter1)>* et *tram2 <( name2, line2, hDep2, hArr2, counter2)>* des places P1 et P2 respectivement. La fonction **majTT(tram1, tram2) -> new tram1** incrémente la valeur de l'heure d'arrivée hArr1 telle que hArr1 = hArr1 + (hArr2 - hArr1) + 1. La fonction **majVector(Vector, majTT(tram1, tram2)) -> new Vector** met à jour l'heure d'arrivée hArr1 dans le vecteur.

Grâce à ce mécanisme, on modélise l'ordonnacement des horaires de circulation des trams sur le réseau.

# Chapitre 4

# MDA et DSL

## 4.1 MDA : Etat de l'art

Dans le concept de MDA (Model Driven Architecture), ou encore Architecture dirigée par la modélisation en français, les modèles sont définis par des méta-modèles qui en spécifient la syntaxe et certaines propriétés structurelles, généralement sous la forme d'une syntaxe abstraite(Ecore, MOF, KM3, etc...), complétée de contraintes exprimées dans un langages de requêtes(ex. OCL). Les méta-modèles permettent de définir des langages de modélisation particulier à un domaine : Ce sont des langages dédiés ou DSL (Domain Specific Langage) en anglais. Ces DSLs ont l'avantage de permettre aux utilisateurs de se concentrer sur leur métier en manipulant un formalisme spécifique à leur activité. Ce chapitre fait un état de l'art des principes généraux de MDA et du processus de création des DSL.

L'OMG (Objet Management Group) [OMG] est à l'origine d'une initiative qui vise à promouvoir la **construction d'applications par modélisation plutôt que par codage**. Cette méthodologie s'appuie sur des standards regroupés au sein du MDA .

L'OMG propose une architecture permettant la définition de **modèles échangeables et transformables**. Ainsi quelque soit la technologie utilisée, nous pourrons facilement passer de cette technologie vers une autre, **à condition, bien sûr, d'avoir les outils de transformation nécessaires**.

### 4.1.1 Principe du MDA (d'après l'OMG)

Le principe clé de la méthode MDA consiste en l'utilisation de modèles aux différentes phases du cycle de développement d'une application :

- Le modèle d'exigences **CIM** (Computation Independent Model).
- Le modèle d'analyse et de conception **PIM** (Platform independent Model).
- Le modèle du code **PSM** (Platform Specific Model).

L'objectif visé par MDA est l'élaboration de modèles qui sont indépendants des détails techniques des plates-formes d'éxécution (J2EE, PHP, .NET, etc...), afin de permettre la génération automatique de la totalité du code des applications et d'obtenir un gain significatif de productivité.

Il existe plusieurs types de transformations de modèles préconisées par MDA :

- CIM $\longrightarrow$ PIM.
- PIM $\longrightarrow$ PSM.

La génération de code n'est pas considérée comme une transformation de modèles à part entière. MDA envisage aussi des transformations inverses dite de *Reverse Engineering*. Bien qu'il existe des techniques traditionnelles de *Reverse Engineering*, c'est une opération assez complexe et difficilement automatisable. :

- code source $\longrightarrow$ PSM
- PSM $\longrightarrow$ PIM.
- PIM $\longrightarrow$ CIM.

La démarche MDA supporte toutes les étapes de développement et standardise les passages de l'une à l'autre. La figure 4.1 illustre, en neuf points, toutes ces étapes. Les points 2, 4 et 6 peuvent en théorie être répétés un nombre indéterminé de fois :

1. La réalisation d'un modèle CIM. C'est le modèle métier ou modèle objet du domaine (MOD). Il saisit les types d'objets les plus important dans le contexte du système. Les objets du domaine représentent les "choses" qui existent ou les évènements qui se produisent dans l'environnement au sein duquel fonctionne le système.

2. l'enrichissement du modèle CIM. Cette étape est théoriquement réalisable de façon itérative, mais avec beaucoup de prudence. L'ajout exa-

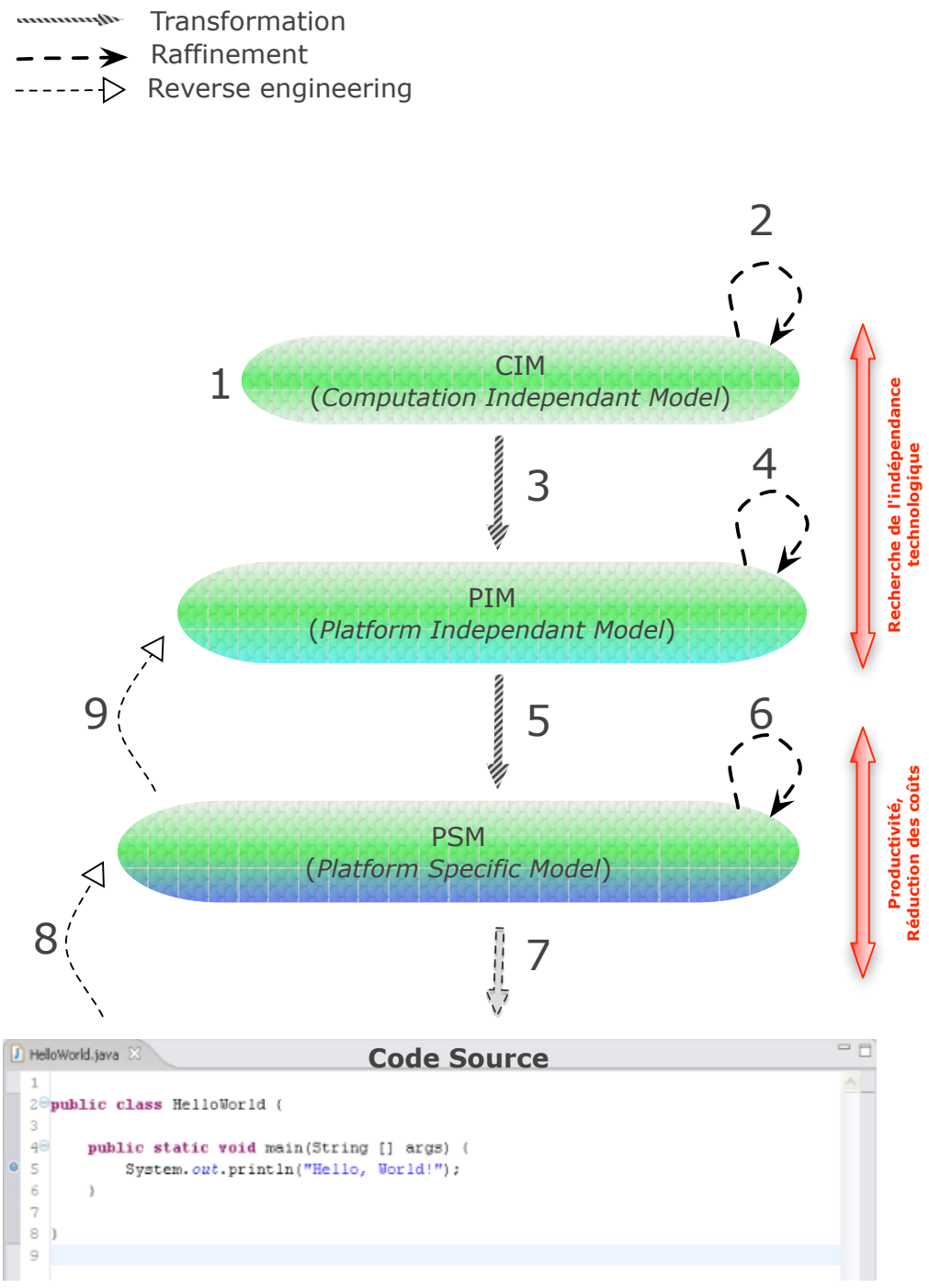



FIGURE 4.1 – Transformation de modèles du MDA

gérée des informations métiers crée parfois des redondances qui peuvent entraîner une modification considérable des évènements du système, et ensuite déstabiliser le processus de transformation des modèles.

3. La transformation du CIM vers un modèle d'analyse et de conception : un modèle métier (PIM). Ce modèle décrit le système, mais ne montre pas les détails de son utilisation sur la plate-forme. Le PIM doit être suffisament précis et contenir suffisament d'informations pour qu'une génération automatique de code soit envisageable. Par exemple, le diagramme de classe UML est un modèle PIM.

4. L'enrichissement et le raffinement du modèle PIM. Il s'agit de spécialiser, filtrer ou enrichir les informations des modèles sans rajouter aucune information liée à la plate-forme.

5. Le choix d'une plate-forme de mise en oeuvre et la génération du modèle spécifique correspondant (PSM).

6. L'enrichissement et le raffinement du modèle PSM jusqu'à obtention d'une implantation exécutable.

7. Génération du code source.

8. Reverse Engineering (Code source vers PSM).

9. Reverse Engineering (PSM vers PIM).

### 4.1.2 Architecture du MDA

Le logo sur la figure 4.3 représente l'architecture du MDA [MDA] et ses différentes couches de spécifications :

- Au coeur : Le standard UML, MOF (Meta Object Facility) et CWM (Common Warehouse Metamodel).

- Autour du coeur : Quelques-unes des plate-formes supportées.

– En surface : Les services systèmes.

– A l'extérieur : Les domaines pour lesquels des composants métiers doivent être définis (Domain Facilities).

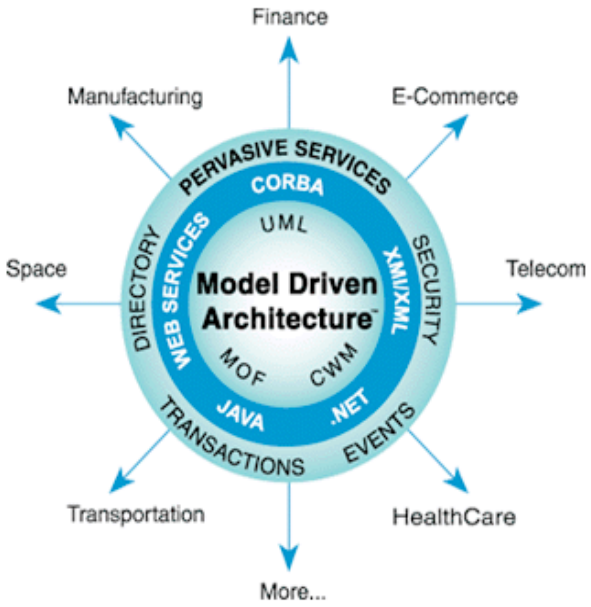



FIGURE 4.2 – Architecture du MDA

### 4.1.3 Standards de l'OMG

Nous avons détaillé les différents modèles du MDA (section 4.1.2) ainsi que le sens des transformations, mais nous n'avons pas décrit comment passer l'un modèle à un autre.

Dans cette section, nous allons examiner l'architecture du MDA puis les différents standards qui permettent ces transformations. La figure 4.3 illustre cette architecture qui est hiérarchisée en quatre niveaux. En partant du bas :

1. Le niveau **MO** correspond au monde réel. C'est une instance du modèle de **M1**.
2. Le niveau **M1** (ou modèle) contient et décrit l'information de **M0** . Les modèles UML, les PIM et les PSM appartiennent à ce niveau. Les

modèles **M1** sont les instances de méta-modèle de **M2**.

3. Le niveau **M2** (ou métamodèle) définit la grammaire de réprésentation des modèles de M1. Par exemple le métamodèle des Réseau de Petri défini dans le standard UML, et qui définit la structure interne des modèles de Réseau de Petri, fait partie de ce niveau.
4. Le niveau **M3** (ou métamétamodèle) est composé du MOF (Meta Object Facility) qui est un standard de métamodélisation et qui permet de décrire la structure des métamodèles, d'étandre ou de modifier des métamodèles existant. Le MOF est réflexif, il se décrit lui-même.

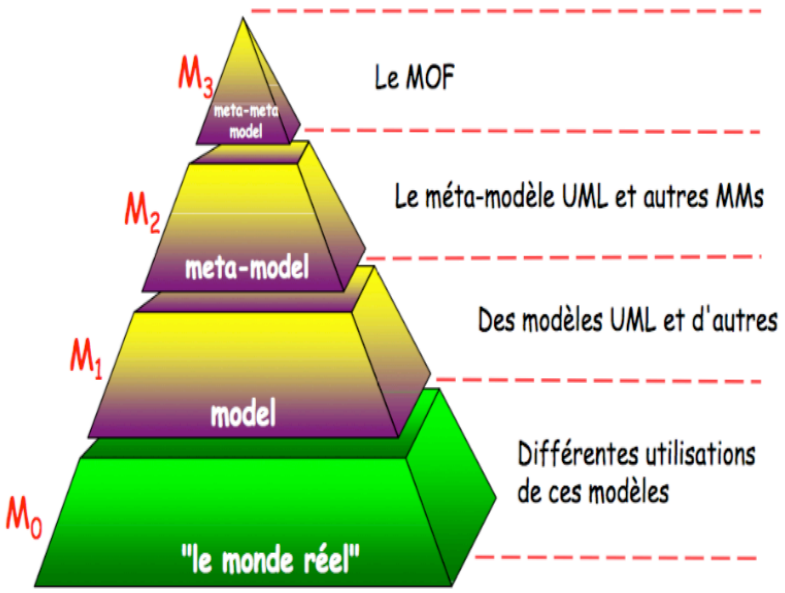



FIGURE 4.3 – Standard OMG : Les quatres couches de l'architecture MDA

## 4.2 DSL : Etat de l'art

L'ingénierie du logiciel s'oriente vers la méthode MDA qui propose une ingénierie dirigée par les modèles au sein de laquelle le système est vu non pas comme une suite de lignes de code mais comme un ensemble de modèles plus abstraits et décrivant chacun une vue (i.e une préoccupation) particulière sur le système. Cette approche favorise l'utilisation de petits langages dédiés (DSL) de plus en plus spécialisés en raison de la multiplication des

domaines d'application de l'informatique.

En informatique, on distingue deux types de langages : Les GPL [I] ou langages généralistes comme C, C++, Ada, Java, UML, etc... et les DSL [II] ou langage dédiés en français.

Il est communément [MAT 09] admis qu'au même titre qu'un GPL , un DSL peut être défini par :

- Sa *syntaxe abstraite* qui décrit la structure interne des constructions du langage.

- Sa *syntaxe concrète* qui décrit le formalisme (apparence externe) qui permet à l'utilisateur de manipuler les constructions du langage.

- Sa *sémantique* qui décrit le sens des constructions du langage.

### 4.2.1 Avantages des DSLs

• L'avantage [DSL] numéro 1 d'utiliser un DSL est de rapprocher l'implémentation de la conception, c'est à dire de faire en sorte que l'écriture du programme soit la plus simple possible, bref, que le code soit plus proche de la pensée qu'il ne l'est avec un langage universel.

• Les langages dédiés permettent d'exprimer des solutions avec les tournures idiomatiques au niveau d'abstraction du domaine traité. En conséquence, les experts du domaine eux-mêmes peuvent comprendre, valider, modifier, et souvent même développer des programmes en langages dédiés.

• Les langages dédiés facilitent la documentation du code.

• Les langages dédiées améliorent la qualité, la productivité, la fiabilité, la maintenabilité, la portabilité et les possibilités de réutilisation.

• Les langages dédiés permettent la validation au niveau du domaine. Aussi longtemps que les éléments du langage sont « sûrs », toute phrase

I. General Purpose Programming Language
II. Domain Specific Language

écrite avec ces éléments peut être considérée comme « sûre ».

### 4.2.2 Inconvénients des DSLs

• Le coût de conception[MAT 09], d'exécution et le maintien d'un DSL.

• Le coût de formation des utilisateurs du DSL. La contrainte pour les utilisateurs d'utiliser la terminologie définie.

• Il peut être difficile de faire collaborer différents DSLs.

### 4.2.3 Exemples de DSL

• Lex et Yacc (Analyseurs lexicaux et syntaxiques).

• Latex (Macro commandes pour faciliter l'utilisation du processeur de texte TeX)

• HTML (Hyper Text Markup Langage)

• Open GL (pour les graphiques 3D)

• Ant (outil de build multi-plateforme fait en Java. Permet d'automatiser les tâches répétitives durant le développement d'un projet)

• Make (compilation de projet en langage C)

etc...

# Chapitre 5

# Conception du DSL textuel (TramLang)

Ce chapitre résume toutes les détails de **conception** du langage dédié **TramLang**. Nous allons notamment décrire la syntaxe abstraite, la syntaxe concrète et la sémantique du langage.

## 5.1 Overview conception du DSL

Dans ce mémoire, nous nous intéressons à la `transformation d'un modèle de TramwayNET conforme au métamodèle` **TramwayMM** (figure 5.3) `en un fichier texte respectant le format d'entrée du model-checker` **HELENA**. L'exécution de ce fichier texte au format *.lna* représente la phase trois du processus de model-checking (3.1). Nous avons divisé le processus de transformation (figure 5.1) d'un modèle de TramwayNET en un modèle textuel de RdP en trois étapes :

- Etape 1 : Transformation ATL[I] d'un modèle de TramwayNET vers un modèle de *timedAPN* (réseau de Petri algébriques temporisés).

- Etape 2 : Transformation ATL d'un modèle de timedAPN vers un modèle de APN (Algebraic Petri Nets ou réseau de Petri algébriques en français).

- Etape 3 : Transformation JAVA d'un modèle de APN vers un modèle textuel de RdP. Le modèle textuel respecte le format d'entrée du model-checker **HELENA**.

I. Atlas Transformation Language

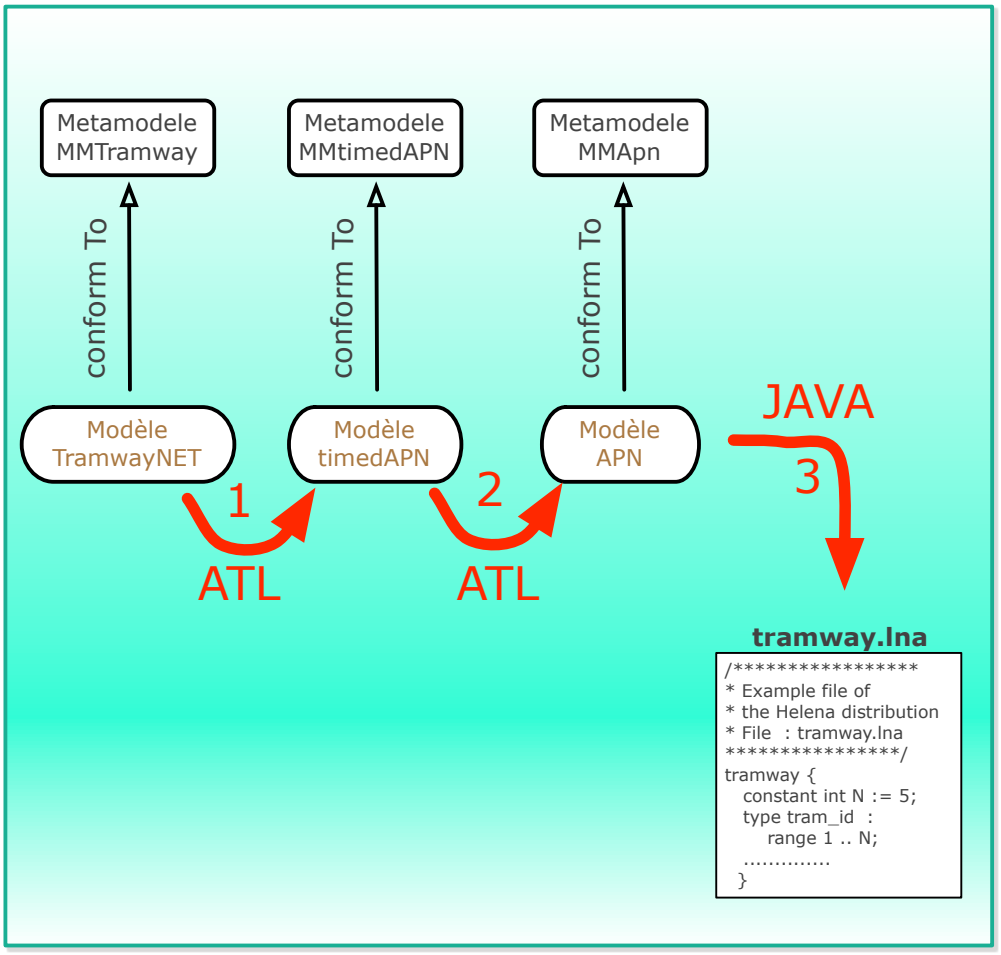



FIGURE 5.1 – Chaine des transformations sémantiques

Nous utilisons deux modèles intermédiaires, l'un (Modèle de timedAPN) conforme au métamodèle ("MMtimedAPN") des réseaux de Petri algébriques temporisés, l'autre (Modèle de APN) conforme au métamodèle des réseaux de Petri algébriques (APN). Des exemples graphiques de ces modèles sont présentés sur la figure 5.2. Ainsi, nous réalisons trois transformations, d'abord deux transformations ATL de modèle vers modèle, ensuite une transformation JAVA de modèle vers du texte (fichier d'entrée au format HELENA).

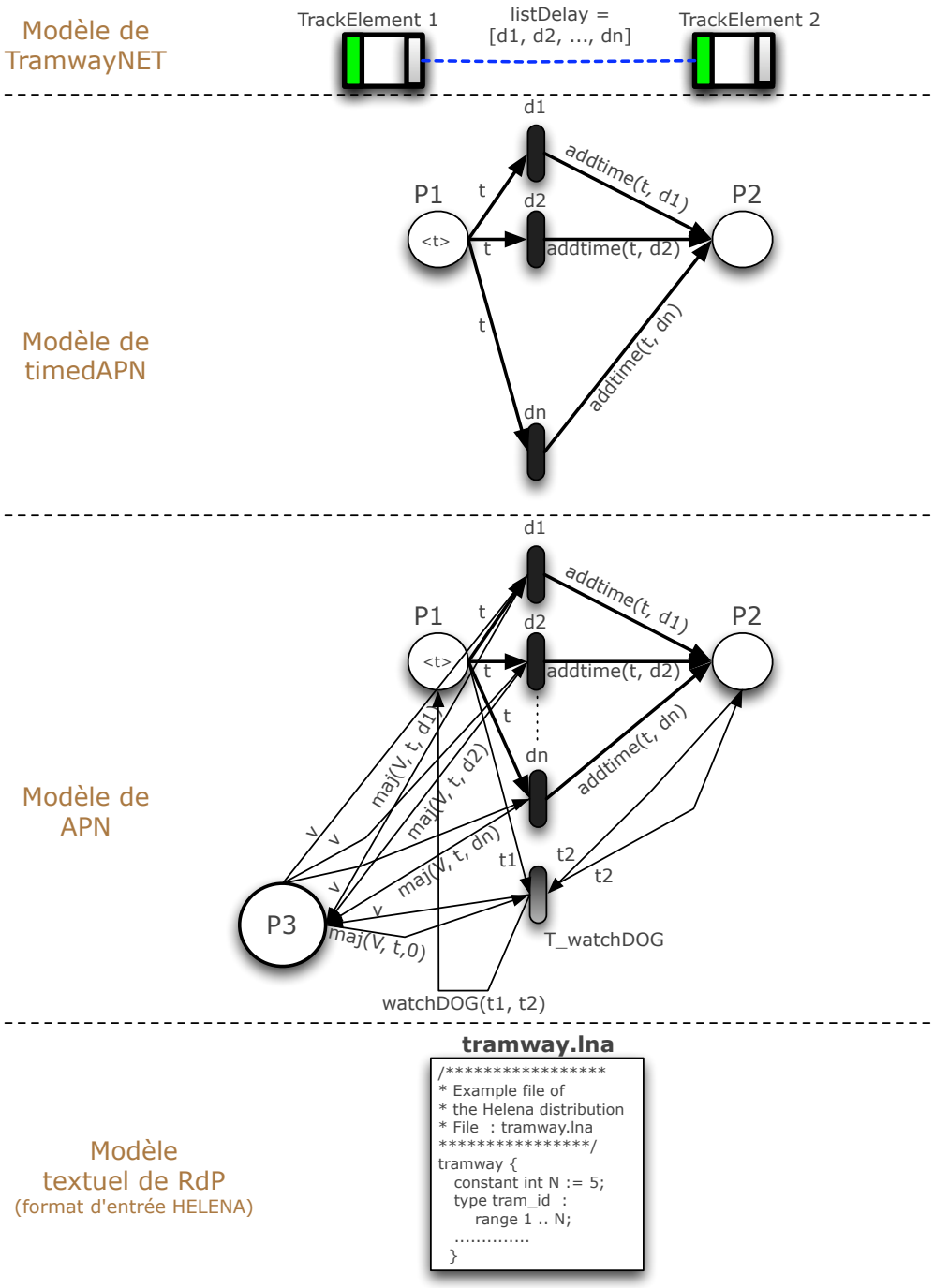



FIGURE 5.2 – Modèles utilisés pour la transformation de TramwayNET vers RdP au format HELENA

Le principal avantage d'utiliser des modèles intermédiaires (figure 5.2) pour construire notre DSL est un gain en **modularité** et **réutilisabilité**. Considérons le processus de transformation de la figure 5.1, on peut réutiliser la chaîne de transformation en remplaçant le modèle de TramwayNET par le nouveau système (réseaux d'aéroports, de trains, d'autoroutes avec des véhicules, etc...) qu'on aimerait modéliser et ensuite appliquer le worflow, à partir de l'étape 2.

## 5.2 Syntaxe abstraite du DSL

Dans cette section nous détaillons les métamodèles du processus de transformation (5.1). Un métamodèle ou syntaxe abstraite décrit la structure interne des constructions d'un langage.

### 5.2.1 métamodélisation : Etat de l'art

**MET(A)**
Elément, du grec meta, exprimant la succession, le changement, la participation, et en philosophie et dans les sciences humaines « ce qui dépasse, englobe » (un objet, une science) : métalangage, métamathématique.

METALANGAGE [metalangaz] n. m. - 1946 ; de méta- et langage ⟶ langage* (encadré) ; en polonais, Tarski, 1931. 1 • LOG. Langage formalisé supérieure qui décide de la vérité des propositions du langage-objet. 2 • LING. Langage(naturel ou formalisé) qui sert à décrire la langue naturelle (⇒ **métalinguistique**). « Le métalangage est un langage dont le signifié est un langage, un autre ou le même » (J. Rey-Debove) - On dit aussi METALANGUE n. f. 1958.

Dictionnaire "*Le PETIT ROBERT*".

Edition 2002 ISBN 2-85036-826-1

En se basant sur la définition du dictionnaire, on peut reconnaitre qu'être « *méta quelque chose* », c'est avant tout être « *quelque chose* ». Une métaconnaissance est une connaissance, une métalangue est une langue, une métaclasse est une classe, une métadonnée est une donnée, etc... **Un métamodèle est donc un modèle**.

Par définition [CAP, 08], la notion **MétaX** est une spécialisation de la notion **X**. Ainsi, une instance de modèle de **X** est conforme au modèle **MétaX**.

Nous allons décrire dans les sections qui suivent tous les métamodèles source, intermédiaires et destination qui ont été utilisés dans le processus 5.1 de transformation d'un modèle de TramwayNET en modèle de Rdp qui respecte le format d'entrée du model-checker HELENA.

### 5.2.2 "TramwayMM" : Metamodèle du réseau de trams

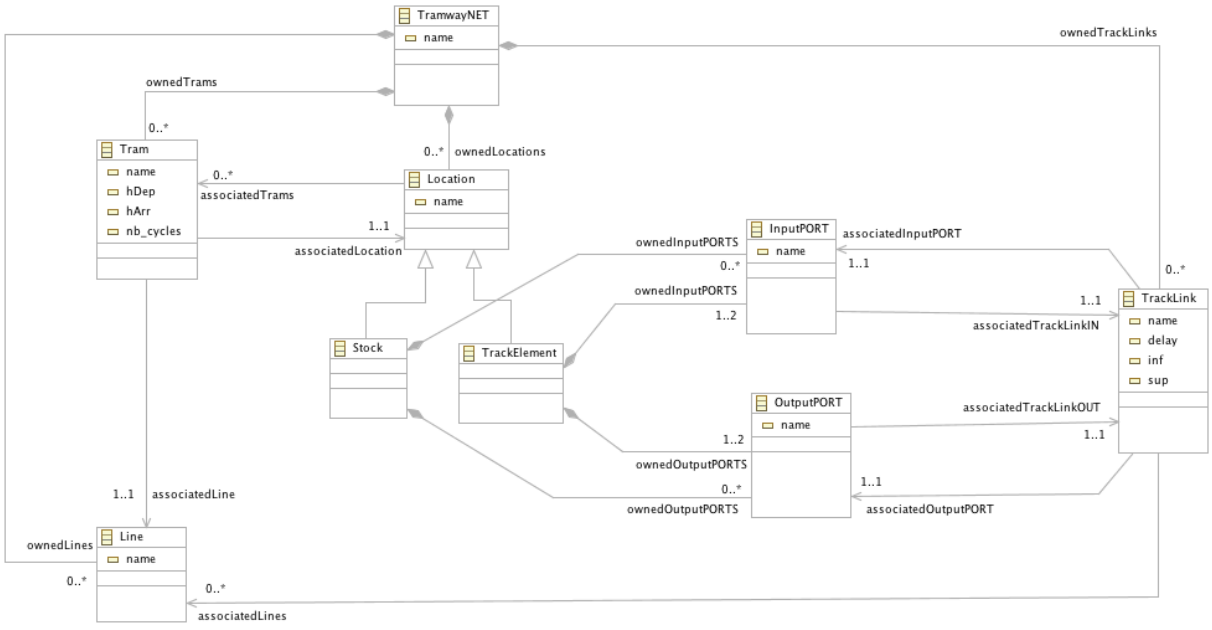



FIGURE 5.3 – Metamodèle de TramwayNET

Les modèles de TramwayNET que nous allons créer seront conformes au métamodèle "**TramwayMM**". La figure 5.4 illustre les métarelations entre un modèle de TramwayNET et le métamodèle "**TramwayMM**". Une méta-relation associe chaque élément du modèle à l'élément du métamodèle qu'il instancie

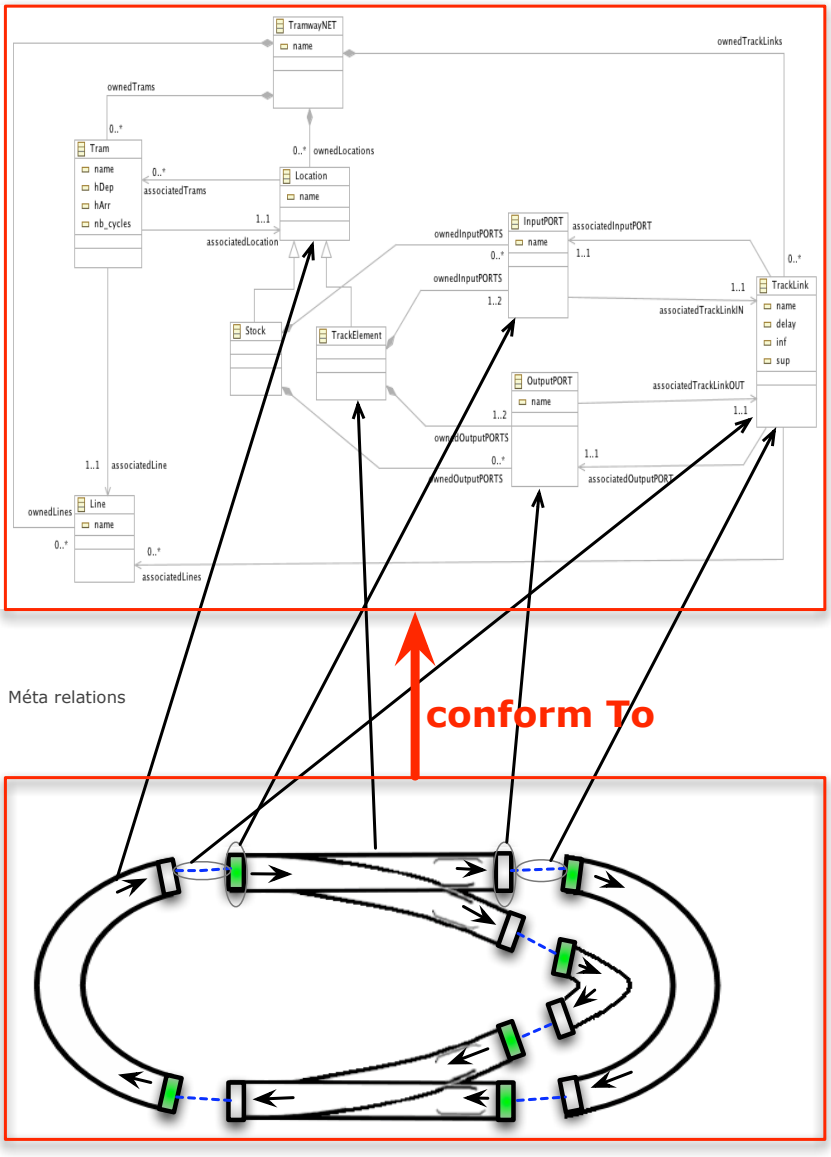



FIGURE 5.4 – Les métarelations entre un modèle de TramwayNET et son métamodèle

Pour le développement du DSL, nous allons considérer tous les éléments qui pourraient "*exister*" dans un réseau de transport public par trams. Après avoir analysé le TramwayNET de la ville de Genève, on considère le réseau physique avec des lignes de trams et ses trams. Le métamodèle de TramwayNET que nous allons implémenter et qui se trouve sur le diagramme UML de la figure 5.3 est composé par :

**Des métaclasses** qui représentent les entités d'un réseau de transport public par trams :

• **TramwayNET** qui représente le point d'entrée du modèle. Il a un attribut :

– `Name` : String

• **Line** pour modéliser une ligne de trams, son attribut :

– `Name` : String

• **Location** C'est une métaclasse abstraite pour modéliser les conteneurs de trams (**Stock**) et des blocs de paires de rails ou croisements sur lesquels circulent les trams. Ses croisements (**TrackElement**) ont été défini sur la figure 2.10. Son attribut :

– `Name` : String

• **Stock** hérite de la métaclasse **Location**. Il peut contenir plusieurs trams à la fois. Nous avons créer cette classe pour pouvoir faire de la simulation en insérant plusieurs trams dans le réseau.

• **TrackElement** hérite de la métaclasse **Location**. Il ne contient qu'un seul tram à la fois. Ils sont représentés sur la figure 2.10.

• **TrackLink** pour modéliser les liens entre les **Locations**, ses attributs :

– `Name` : String
– `delay` : Int, c'est le delai (durée) idéal pour aller d'une **Location** à une autre.
– `inf` : Int
– `sup` : Int

*inf* et *sup* sont les paramètres qui seront donnés en entrée par un utilisateur du langage **TramLang** pour calculer respectivement les délais minimum et maximum pour aller d'une **Location** à une autre.

• **Tram** qui représente une ressource (un tram) du réseau. Ses attributs :

- `Name` : String
- `hDep` : Int, C'est l'heure de départ du tram dans la Location ou il se trouve.
- `hArr` : Int, C'est l'heure d'arrivée du tram dans la Location ou il se trouve.
- `nb_cycles` : Int, c'est le nombre de cycles effectués par un tram sur le réseau.

• **InputPORT** cette métaclasse modélise le port d'entrée (figure 2.10)dans une **Location**. Son attribut :

- `Name` : String

• **OutputPORT** cette métaclasse modélise le port de sortie (figure 2.10) d'une **Location**. Son attribut :

- `Name` : String

**Des compositions** qui représentent des relations Composant-Composé entre les entités du réseau, La suppression du Composant entraîne la suppression du composé :

• **ownedTrackLinks** entre **TramwayNET** et **TrackLink**, avec comme cardinalité égale à 0 pour la limite inférieure et égale à * ($\infty$ [II]) pour la limite supérieure.

• **ownedLines** entre **TramwayNET** et **Line**, avec comme cardinalité égale à 0 pour la limite inférieure et égale à * ($\infty$) pour la limite supérieure.

• **ownedTrams** entre **TramwayNET** et **Tram**, avec comme cardinalité égale à 0 pour la limite inférieure et égale à * ($\infty$) pour la limite supérieure.

• **ownedLocations** entre **TramwayNET** et **Location**, avec comme car-

II. infini

dinalité égale à 0 pour la limite inférieure et égale à * ($\infty$) pour la limite supérieure.

• **ownedInputPORTS** entre **Stock** et **InputPORT**, avec comme cardinalité égale à 0 pour la limite inférieure et égale à * ($\infty$) pour la limite supérieure.

• **ownedOutputPORTS** entre **Stock** et **OutputPORT**, avec comme cardinalité égale à 0 pour la limite inférieure et égale à * ($\infty$) pour la limite supérieure.

• **ownedInputPORTS** entre **TrackElement** et **InputPORT**, avec comme cardinalité égale à 1 pour la limite inférieure et égale à 2 pour la limite supérieure.

• **ownedOutputPORTS** entre **TrackElement** et **OutputPORT**, avec comme cardinalité égale à 1 pour la limite inférieure et égale à 2 pour la limite supérieure.

**Des Associations** qui représentent des relations entre les entités du réseau :

• **associatedLine** entre **Tram** et **Line**, avec comme cardinalité égale à 1 pour la limite inférieure et égale à 1 pour la limite supérieure. (Un tram est associé à une ligne de tram)

• **associatedLines** entre **TrackLink** et **Line**, avec comme cardinalité égale à 0 pour la limite inférieure et égale à * ($\infty$) pour la limite supérieure. (Un **TrackLink** est associé à zéro ou plusieurs ligne de tram)

• **associatedLocation** entre **Tram** et **Location**, avec comme cardinalité égale à 1 pour la limite inférieure et égale à 1 pour la limite supérieure. (Un tram est associé à une **Location**)

• **associatedTrams** entre **Location** et **Tram**, avec comme cardinalité égale à 0 pour la limite inférieure et égale à * ($\infty$) pour la limite supérieure. (une **Location** est associée à zéro ou plusieurs trams)

• **associatedInputPORT** entre **TrackLink** et **InputPORT**, avec comme cardinalité égale à 1 pour la limite inférieure et égale à 1 pour la limite supérieure. (un **TrackLink** est associé à un **InputPORT**)

• **associatedOutputPORT** entre **TrackLink** et **OutputPORT**, avec comme cardinalité égale à 1 pour la limite inférieure et égale à 1 pour la limite supérieure. (un **TrackLink** est associé à un **OutputPORT**)

• **associatedTrackLinkIN** entre **InputPORT** et **TrackLink**, avec comme cardinalité égale à 1 pour la limite inférieure et égale à 1 pour la limite supérieure. (un **InputPORT** est associé à un **TrackLink**)

• **associatedTrackLinkOUT** entre **OutputPORT** et **TrackLink**, avec comme cardinalité égale à 1 pour la limite inférieure et égale à 1 pour la limite supérieure. (un **OutputPORT** est associé à un **TrackLink**)

Nous précisons que les associations sont **toujours** unidirectionnelles car EMF [III] ne gère pas les associations bidirectionnelles.

Expression des contraintes en OCL

Le métamodèle des tramways 5.3 que nous avons défini permet de construire des modèles "*incorrects*". Ces modèles sont néanmoins conformes au métamodèle des tramways. Pour les rendre corrects, du point de vue des spécifications du réseau, l'OMG préconise d'utiliser OCL (Objet Constraint Langage) [OCL, 06] qui est un langage formel de description de contraintes orienté-objet. Il est notamment utilisé pour combler les manques sur les diagrammes UML(Unified Modeling Language). Appliqué au niveau du métamodèle, il permet d'ajouter des propriétés principalement structurelles, qui n'ont pas pu être capturées par les concepts fournis par le métamétamodèle. Il s'agit donc d'un moyen de préciser la sémantique du métamodèle, en limitant les modèles conformes.

Par exemple, pour notre métamodèle des tramway, on peut définir les contraintes OCL suivantes :

III. Eclipse Modeling Framework pour créer des métamodèle (*.ecore)

**Context TrackElement**
**def** ListLinesINt = Self.ownedInputPORTS -> collect(in : InputPORT|in.associatedTrackLinkIN)
**def** ListLinesOUTt = Self.ownedOutputPORTS -> collect(out : OutputPORT|out.associatedTrackLinkOUT)

**Context Stock**
**def** ListLinesINs = Self.ownedInputPORTS -> collect(in : InputPORT|in.associatedTrackLinkIN)
**def** ListLinesOUTs = Self.ownedOutputPORTS -> collect(out : OutputPORT|out.associatedTrackLinkOUT)

↪ "*Pour chaque Location, le nombre de lignes en entrée est égal au nombre de lignes en sortie*"

**Context TrackElement**
**inv** : ListLinesINt -> size() = ListLinesOUTt -> size()

**Context Stock**
**inv** : ListLinesINs -> size() = ListLinesOUTs -> size()

↪ "*Pour chaque Location, Toutes les lignes en entrée doivent* être identiques en sortie" :

**Context TrackElement**
**inv** : ListLinesINt -> includeAll(ListLinesOUTt)

**Context Stock**
**inv** : ListLinesINs -> includeAll(ListLinesOUTs)

↪ "*le nombre d'objets Tram associés à un* TrackElement = à zéro ou 1"

**Context TrackElement**
**inv** : TrackElement.associatedTrams -> size() <= 1

### 5.2.3 "apnMM" : Metamodèle des réseaux de Petri algébriques

Dans un soucis de clarté dans ce mémoire et étant donnée la taille du métamodèle des APN qui a été développée au sein du groupe SMV (Software

Modeling Group) à l'université de Genève, nous présentons ici, une version simplifiée de ce métamodèle des APN. Cette version est tirée du mémoire de fin d'études de Master de M. alexis MARECHAL [ALE 08].

### 5.2.4 "timedAPNMM" : Metamodèle des réseaux de Petri algébriques temporisés

Nous construisons le métamodèle des timedAPN à partir du métamodèle des APN auquel nous ajoutons l'information temporelle. Le métamodèle des timedAPN correspond au métamodèle *étendu* pour la définition de la sémantique du temps formellement définie pour les TCPN [Jen 97].

Nous avons détaillé la sémantique temporelle des TCPN dans la section 2.2.3. Pour faire l'équivalence entre un TCPN et un timedAPN, nous considérons que :

- les couleurs (jetons) dans un TCPN sont des "*terms*" dans un timedAPN.

- Chaque "**TimeStamp**" associé à une couleur correspond à une valeur de temps $t \in \mathbb{N}^*$ associée à un "*term*" dans un timeAPN.

- Un délai attribué à une transition dans un TCPN correspond à un "*term*" dans un timedAPN. Dans ce cas les "*terms*" sont des inscriptions sur les transitions. Nous rappelons sur la figure 5.6 les inscriptions qui peuvent être associées à une transition de TCPN.

- Les inscriptions sur les arcs dans un TPCN correspondent à un multiset de "*termes*" dans un timedAPN.

Ainsi pour construire le métamodèle des timedAPN, nous avons rajouté au métamodèle des APN les composants suivants :

⊗ Ajout de l'attribut $\mathbf{r}_0$ de type *EInt* à la métaclasse "**PetriNet**". Cet attribut représente le "**Start Time**" ou "**Global Clock**" 2.2.3.

⊗ Ajout de l'attribut **delay** de type *EInt* à la métaclasse "**Transition**".

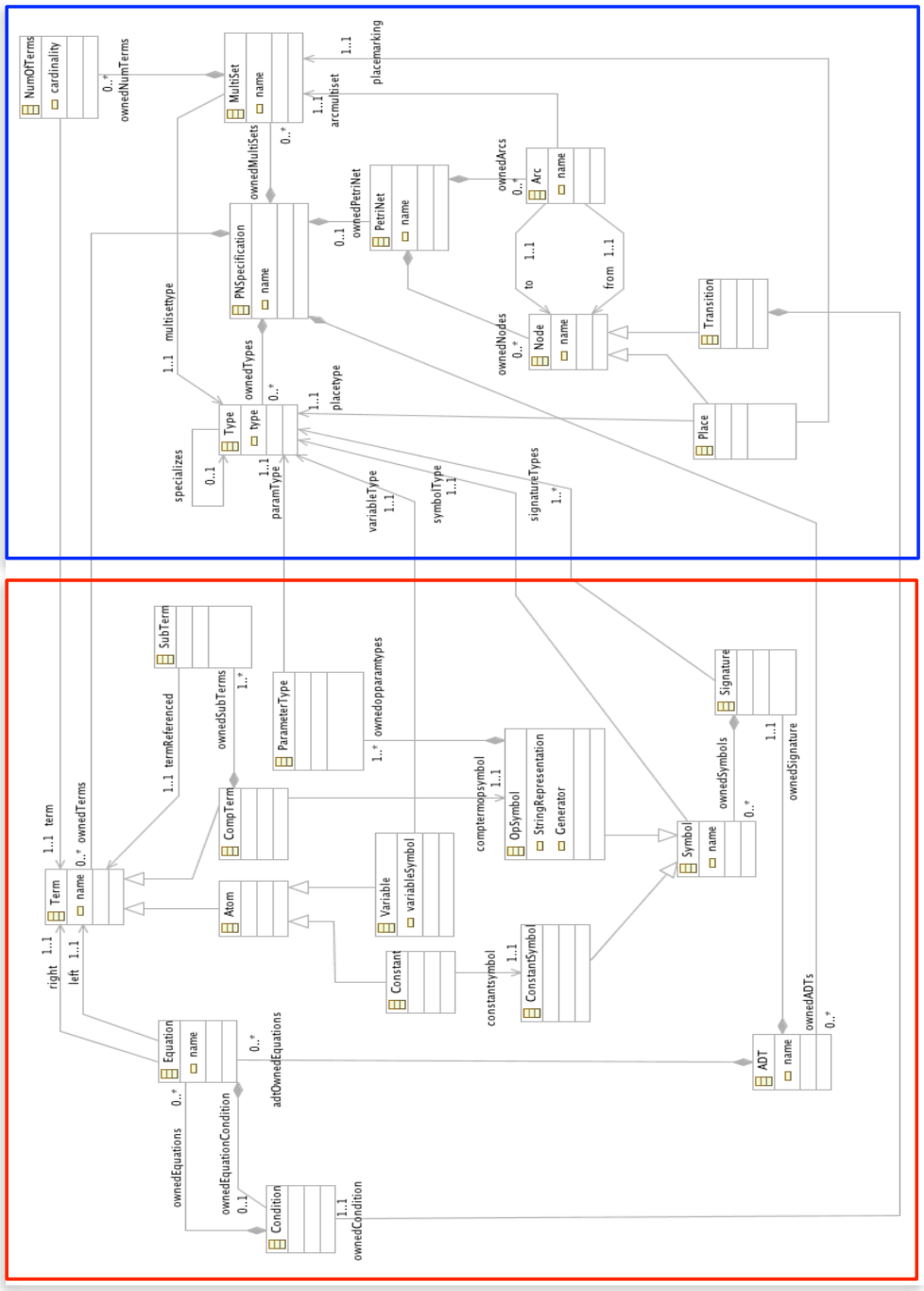
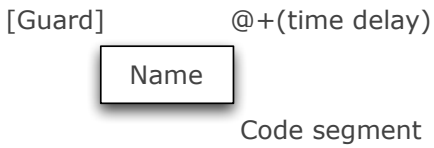



FIGURE 5.5 – Métamodèle pour les APN(carré bleu) et Métamodèle pour la représentation des ADT des APN (carré rouge)




**Name**: inscription pour le nom de la transition
**[Guard]**: inscription pour la garde de la transition
**@+(time delay)**: inscription pour le délai attribué à la transition
**Code segment**: inscription pour une fonction

FIGURE 5.6 – Les quatres inscriptions possibles sur une transition dans un TCPN

Cet attribut représente l'inscription du délai à une transition.

⊗ Ajout de la métaclasse "**timedMSElement**" avec son attribut $t$ de type *EInt*. Cet attribut représente le "**TimeStamp**" associé à un "*term*" (jeton).
⊗ Ajout des relations de composition entre les classes .....
la figure 5.7 illustre la version simplifiée du métamodèle des timedAPN.

## 5.3 Syntaxe concrète du DSL

Le développement des syntaxes abstraites et concrètes de manière indépendante, nous a posé beaucoup de problèmes en terme de cohérence lors des évolutions de la syntaxe abstraite.

La solution que nous proposons dans ce mémoire, est de définir la syntaxe concrète comme dépendante de la syntaxe concrète.

Notre première préoccupation est de définir un formalisme dédié aux experts métiers. Cela implique une contrainte de conservation de la terminologie utilisée. Pour définir ce formalisme, nous avons utilisé TMFxtext qui est un composant Eclipse,permettant d'associer une syntaxe textuelle à un métamodèle, afin d'obtenir ensuite automatiquement un éditeur dédié pour son propre langage. Le langage va fournir la colorisation et l'auto-complétion. Nous avons détaillé l'implémentation de la synataxe concrète de notre DSL TramLang dans le chapitre suivant 6

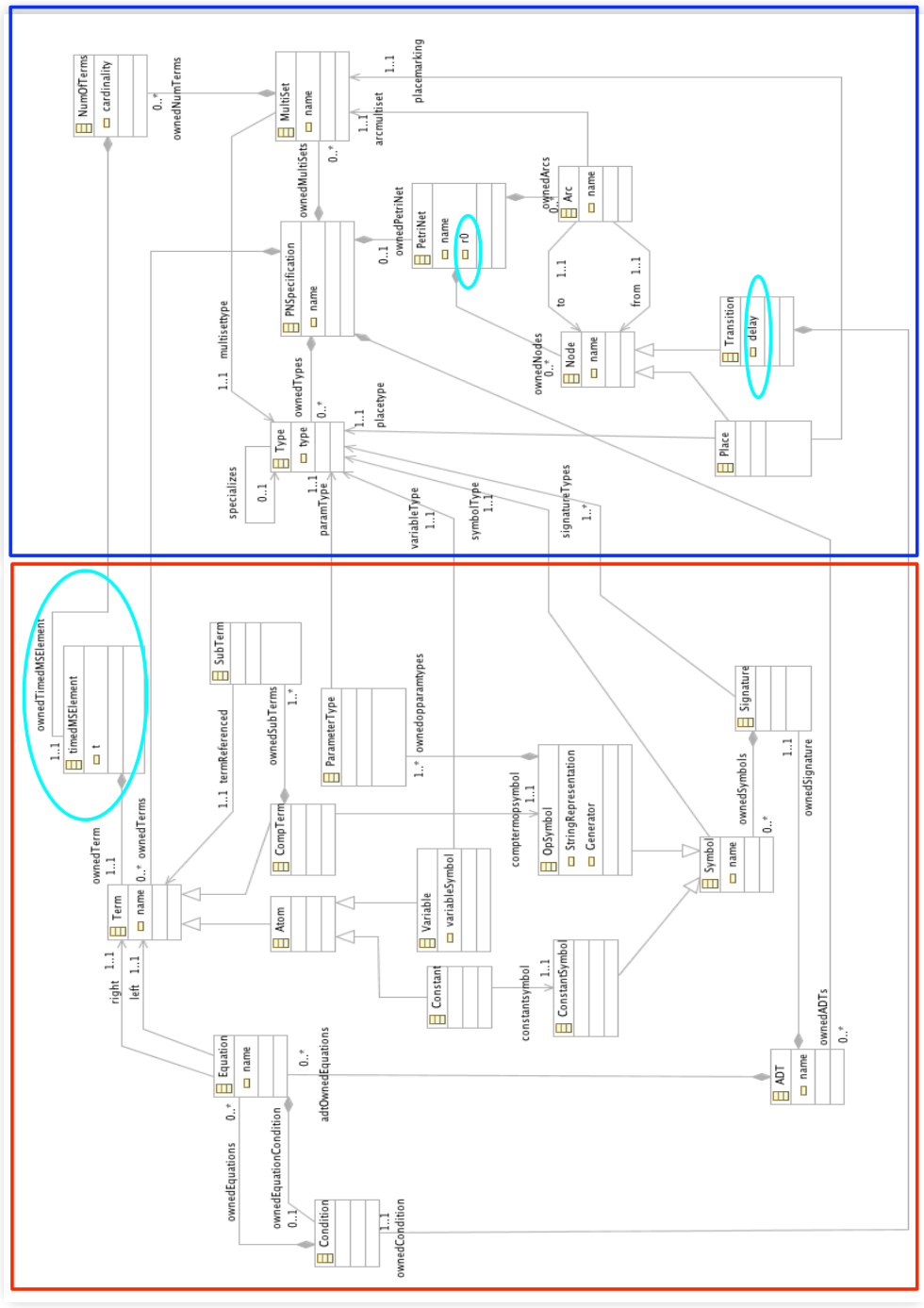



FIGURE 5.7 – Métamodèle pour les timedAPN(carré bleu) et Métamodèle pour la représentation des ADT des timedAPN (carré rouge)

## 5.4 Sémantique du DSL

La sémantique opérationnelle du langage TramLang est obtenue à partir du langage de transformations de modèles ATL (Atlas Transformation Language) ([Bez]).

Un modèle de transformation ATL peut transformer un ensemble de modèles sources en un ensemble de modèles cibles, à la condition que les métamodèles sources et cibles lui soient connus.

Nous reprenons ici les différentes étapes du processus 5.1 de transformation d'un TramwayNET en RdP dont nous présentons, point par point, tous les mappings qui ont été opérés.

### 5.4.1 Etape 1 : Transformation ATL de TramwayMM vers timedAPNMM

Dans cette étape, on transforme un modèle de **TramwayNET** en modèle **timedAPN**. Le modèle obtenu est un Rdp algébrique qui conserve l'*information temporelle*.

Mapping de métaclasse "**TramwayNET**" vers métaclasse "**PetriNET**"

Un programme de transformation écrit en ATL est composé de règles qui spécifient comment les éléments du modèles sont reconnus et parcourus pour créer et initialiser les éléments du modèle cible. Ces règles sont de la forme générale suivante (5.8) :

Sur la règle ATL de la figure 5.8 :

• main est le nom de la règle de transformation.

• t_net (resp. p_net) est le nom de la variable qui dans le corps de la règle va représenter l'élément source identifié (resp. l'élément cible crée).

```
module Tramway_2_timedAPN; -- Module Template

create OUT : timedAPNMM from  IN : TramwayMM;

-- Cette règle genère la structure d'un modèle de "timedAPN" à partir d'une classe "TramwayNET"

--règle principale
rule main { -- Matched rule

    from
        t_net : TramwayMM!TramwayNET

    to
        p_net : timedAPNMM!PetriNet(

            name <- ' This timed APN is conform to : ' + t_net.name
            ,
            ownedNodes <- t_net.ownedLocations
            ,
            ownedNodes <- t_net.ownedTrackLinks
            ,
            ownedALGSPEC <- algspec -- Un RdP contient une specification algébrique

        )

}

```

FIGURE 5.8 – Un exemple de règle ATL

• TramwayMM (resp. timedAPNMM) est le métamodèle auquel le modèle source identifié (resp. le modèle cible) de la transformation est conforme.

• TramwayNET désigne la métaclasse des éléments du modèle source auxquels cette règle va s'appliquer.

• PetriNet désigne la métaclasse à partir de laquelle la règle va instancier les éléments cibles.

• name est l'attribut de la métaclasse *PetriNet*.

• ownedNodes et ownedALGSPEC sont des relations de composition de la métaclasse *PetriNet*.

• La valeur de l'attribut *name* est initialisé à l'aide de la valeur de l'attribut t_net.name de la métaclasse *TramwayNET*.

• Les valeurs des relations de composition *ownedNodes* et *ownedALGSPEC* sont respectivement initialisés à l'aide des valeurs des relations de composition *t_net.ownedLocations* et *t_net.ownedTrackLinks* de la métaclasse *TramwayNET*.

• Le point d'exclamation permet de spécifier à quel métamodèle appartient une métaclasse en cas d'homonymie.

Voici comment on pourrait formuler la fonction de cette règle en langage naturel :

La règle *main*, pour chaque élément *t_net* de type *TramwayNET* qu'elle identifie dans le modèle source, crée dans un modèle cible un élément *p_net* de type *PetriNet*, et initialise les valeurs des attributs *name* de *p_net* avec la valeur de l'attribut *name* de *t_net*. Le même raisonnement est appliqué pour les relations de compositions *ownedNodes* et *ownedALGSPEC*.

Mapping métaclasse "**TrackLink**" vers métaclasse "**Transition**"

Dans le fichier de transformation ATL, on va commencer par le calcul de la liste des délais d'un TrackLink.

Prenons un exemple, on crée un TrackLink avec ses attributs delai = 2 , inf = 1 et sup = 1 alors la liste des delais attribués a ce TrackLink est égale à [delai-inf, delai+sup] = [1, 2, 3]. Nous avons défini la sémantique de transformation 2.4 d'un TramwayNET en RdP classique qui stipule qu'un **TrackLink** est transformé en **Transition**. Dans ce contexte, et en tenant compte de la liste des delais, on va créer, pour chaque delai de la liste des delais, une transition ayant comme attribut le délai correspondant.

Les autres mappings de métaclasse pour notre transformation ne seront pas détaillés dans ce document. Ce sont des règles ATL définies de la même manière que le mapping « Mapping de métaclasse "**TramwayNET**" vers métaclasse "**PetriNET**" » décrit ci-dessus. Voici la liste des autres mapping de métaclasse de notre transformation :

- Mapping de métaclasse "**TramwayNET**" vers métaclasse "**PetriNET**" et métaclasse "**AlgebraicSpec**".
- Mapping métaclasse "**Stock**" vers métaclasse "**Place**" et métaclasse "**Multiset**".
- Mapping métaclasse "**TrackElement**" vers métaclasse "**Place**" et mé-

taclasse "**Multiset**".

– Mapping métaclasse "**TrackLink**" vers métaclasse "**Transition**", métaclasse "**Arc**" et métaclasse "**Multiset**".

### 5.4.2 Etape 2 : Transformation ATL de timedAPNMM vers apnMM

Dans cette étape, on **réutilise** le modèle obtenu avec la première transformation (tramwayMM vers timedAPNMM). Ce modèle conserve toutes les informations (les délais, le nombre de trams, de places, de transition, d'arcs, les couleurs, les fonctions sur les arcs, les gardes sur les transitions, etc..) provenant du modèle de Tramway.

On va spécifier des règles ATL pour transformer un modèle de timedAPN en modèle d'APN. Le modèle d'APN obtenu conserve l'information temporelle dans la spécification algébrique du réseau via les *termes*. Le modèle obtenu permet de simuler l'écoulement du temps tel que nous l'avons décrit dans la section 3.3.3.

Nous précisons que les métamodèles timedAPNMM et APNMM sont très similaires car le premier a été généré a partir du second, les règles ATL de transformation sont plus facile à réaliser. Il s'agit de mappings entre les métaclasses ayant pour la plupart les mêmes attributs. On distingue :

– Mapping de métaclasse "**PetriNet**" vers métaclasse "**PetriNet**".

– Mapping métaclasse "**Transition**" vers métaclasse "**Transition**".

– Mapping métaclasse "**Place**" vers métaclasse "**Place**".

– Mapping métaclasse "**Arc**" vers métaclasse "**Arc**".

– Mapping métaclasse "**AlgebraicSpec**" vers métaclasse "**AlgebraicSpec**".

– Mapping métaclasse "**Multiset**" vers métaclasse "**Multiset**".

– etc...

L'information temporelle est représentée dans le métamodèle source timedAPNMM. L'objectif de cette transfomation est de pouvoir simuler l'écoulement du temps (figure 2.5). Ainsi le modèle d'APN généré va contenir une place supplémentaire qui modélise la variable globale. Cette variable globale est représentée dans le modèle d'APN, par une place qui contient un multiset avec plusieurs termes. L'ensemble des ressources (trams) du réseau est représenté par le multiset de termes composés.

### 5.4.3 Etape 3 : Transformation Java de apnMM vers HELENA

Dans cette étape, on va **réutiliser** le modèle obtenu avec la troisième transformation (timedAPNMM vers APNMM). On utilise le langage JAVA pour produire un modèle de RdP coloré, qui respecte le format (textuel) d'entrée du model-checker HELENA.

Ce modèle sera utilisé pour la validation des propriétés spécifiées sur le réseau initial (TramwayNET).

Dans cette transformation JAVA, on va charger le modèle d'APN obtenu de la transformation précédente et pour chaque objet qui nous intéresse, on va afficher ses attributs. L'affichage est formaté de façon à obtenir un fichier textuel, conforme à la syntaxe d'un fichier au format HELENA.

# Chapitre 6

# Implémentation du DSL textuel (TramLang)

L'utilisation de l'anglicisme « Implémentation » est courante [I] et désigne, en Ingénierie et plus particulièrement en informatique, la création d'un produit fini à partir d'un document de conception, d'un document de spécification, voire directement depuis un cahier des charges [wikitionnaire].

Dans ce chapitre, nous résumons les étapes techniques de création du DSL textuel "TramLang". Le développement du DSL tient compte des objectifs de conception du DSL énoncées dans le chapitre cinq. Parmi ces contraintes, on peut citer la modularité, la réutilisabilité des modèles, la compatibilité du code source avec les environnements de développement, etc...

## 6.1 Implémentation de la syntaxe abstraite

La syntaxe abstraite ou métamodèle désigne la structure interne des constructions du langage. Nous avons utilisé la notation EMF (qui ressemble à UML) pour créer nos métamodèles qui sont présentés dans le chapitre 5.2.

La génération de code se fait par la plate-forme Eclipse, à partir de modèle EMF. Le modèle est alors exprimé sous forme de Java annoté.

I. L'adoption du terme « Implémenter » par la commission générale de terminologie et de néologie a été publiée au journal officiel de la république Française le 20 Avril 2007.

## 6.2 Implémentation de la syntaxe concrète

Pour construire la syntaxe concrète du langage dédié **TramLang**, nous définissons des mots réservés qui seront incontournables par les utilisateurs du langage. Le choix de ses mots doit être porteur d'informations et suffisamment simple à utiliser. Les tests d'utilisabilité pourraient aider à concevoir de meilleurs langages. Voici les mots clés réservés du langage :

"Tramway net",
"Lines",
"Locations",
"TrackLinks",
"Trams",
"Stock",
"inputPORTS",
"outputPORTS",
"Tram",
"TrackElement",
"myLine",
"hDep",
"hArr",
"nd_cycles",
"associatedLocation",
"associatedLine",
"delay",
"inf",
"sup",
"TrackLink".

Définition de la grammaire

↪ La grammaire est définie selon le format semblable à BNF dans l'editeur TMFxtext.

↪ L'éditeur facilite la programmation avec la complétion de code et les contraintes définies sur la grammaire elle même.

↪ La grammaire est une collection de règles (figure 6.1). Une règle commence par son nom suivi des deux points (" :") et se termine avec le point virgule (" ;").

```
grammar ch.unige.cui.smv.TramwayDSL with org.eclipse.xtext.common.Terminals

import "platform:/resource/Tramway2BigTimedAPN_galileo/metamodels/Tramway.ecore" as tramwayMM

TramwayNET_rule returns tramwayMM::TramwayNET:

    //ID = terminals in org/eclipse/xtext/common/Terminal.xtext
    "Tramway net : " name = ID ":"

    "Lines : " (ownedLines += Line_rule ("," ownedLines += Line_rule)* )?
                                    // ? signifie optionnel
    "Locations : " (ownedLocations += Location_rule)*

    "TrackLinks : " (ownedTrackLinks += TrackLink_rule)*

    "Trams : " (ownedTrams += Tram_rule)*
;

Line_rule returns tramwayMM::Line :
    name = ID
;

Location_rule returns tramwayMM::Location :
    Stock_rule|TrackElement_rule
;

// [tramwayMM::Tram] est une référence vers un tram déjà existant
// CONTRAINTE: Un Stock est associé à 0 ou plusieurs Trams
Stock_rule returns tramwayMM::Stock :
    "Stock {"
        name = ID
        "inputPORTS : " ownedInputPORTS +=                          // contient 1..* InputPORT
                    InputPORT_rule ("," ownedInputPORTS += InputPORT_rule) *

        "outputPORTS : " ownedOutputPORTS +=                        // contient 1..* OutputPORT
                    OutputPORT_rule ("," ownedOutputPORTS += OutputPORT_rule) *


        "Tram : " (associatedTrams += [tramwayMM::Tram] ("," associatedTrams += [tramwayMM::Tram])* )?
    "}"
;
```

FIGURE 6.1 – Grammaire du DSL TramLang avec TMFxtext

## 6.3 Implémentation de la sémantique

Comme nous l'avons mentionné à la section 5.4, la sémantique opérationnelle du langage TramLang est obtenue par transformations successives. Les deux premières transformations ont été réalisées à l'aide du langage de transformations de modèles ATL.

Dans cette section nous présentons une fonctionnalité du langage ATL qui démontrent toute sa puissance .

L'operateur resolveTemp(<nom de règle>, <nom élément cible>).

L'opération resolveTemp D'ATL permet de retrouver un élément particulier du modèle cible crée dans une autre règle. Le code source ATL que nous présentons illustre l'intérêt de l'usage de cette fonction.

Considérons les deux règles de transformations en ATL suivantes, La règle « rule main » etla règle « rule TransitionIdeale_2_TransitionIdeale » :

```
-- The main rule
```

rule main

```
{
from
timed_p_net : timed_Merge_APNMM!PetriNet

to
p_net : Merge_APNMM!PetriNet (
name <- 'This is APN from timedAPN'
)

-- Pour un PetriNet, on crée une place "P_all_TRAMS"
-- qui sera notre var globale
,
p_all_TRAMS :  Merge_APNMM!Place(
name <- ' P_all_TRAMS'
```

```
)

-- The Transition to Transition rule
```
rule TransitionIdeale_2_TransitionIdeale
```
{
from
t1_Ideale : timed_Merge_APNMM!TransitionIdeale
(
--thisModule.transition_Ideale_Set -> includes(t1_Ideale)
t1_Ideale.oclIsTypeOf(timed_Merge_APNMM!TransitionIdeale)
)

to
t2_Ideale : Merge_APNMM!TransitionIdeale(
name <- t1_Ideale.name
,
delay <- t1_Ideale.delay
)
,
arcIN_ideal : Merge_APNMM!Arc (

from <- t2_Ideale

,-- Lien vers la classe qui a crée la place 'p_all_TRAMS'
to <- thisModule.resolveTemp(t1_Ideale.refImmediateComposite(),
'p_all_TRAMS') -- to Place
)
```

Dans la règle « rule main », on crée une place (P_all_TRAMS) dans notre réseau. Ensuite dans la règle « rule TransitionIdeale_2_TransitionIdeale », on va créer des transitions est les arcs qui sont rattachés à ces transitions.

Considérons l'instruction suivante :

```
thisModule.resolveTemp(t1_Ideale.refImmediateComposite(), 'p_all_TRAMS')
```

Le résultat de son éxécution est une référence vers la place P_all_TRAMS qui a été créée dans la première règle (« rule main »).

## 6.4 DSL et Support d'implémentation

Dans cette section, La figure 6.2 résume l'ensemble des outils techniques et les formats de données qui ont été utilisés pour réaliser ce DSL :

- Eclipse (version Ganymede et Galileo) Plate-forme de développement d'applications.

- EMF(Eclipse Modeling Framework) permet de créer des modèles, de les enrichir, et de créer des programmes exécutables à partir de ces modèles. Un des avantages de cette approche est que lorsqu'on enrichie un modèle, le code généré est automatiquement conforme aux nouvelles modifications.

- ATL(Atlas Transformation Language), disponible sous forme de plug-in Eclipse, qui permet de réaliser des transformations quelconques sur des modèles crées avec EMF.

- TMFxtext permet de créer un editeur/parseur qui permet de produire des instances de métamodèle crée avec EMF.

- Ecore, c'est l'implémentation de MOF (Meta Object Facility) par EMF.

- XMI (XML Metadata Interchange), format standard défini par l'OMG, tous les modèles sont disponibles au format XMI, mais la représentation sous forme de fichier texte fait que, pour la manipulation, les formats graphiques UML et les modèles Ecore d'EMF sont préférés.

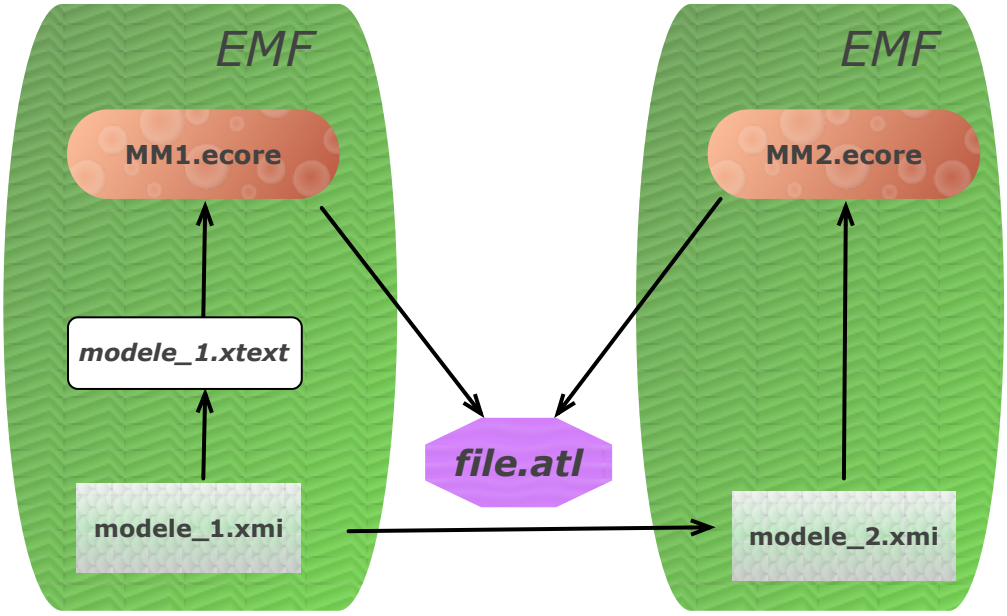



FIGURE 6.2 – Liens entre les outils et les formats de données : EMF, ATL , XMI, Ecore, TMFXtext

# Chapitre 7

# Conclusion et Perspectives

Nous avons présenté dans ce mémoire de Master, les principes et les méthodes de conception, qui nous ont permis de construire un DSL (Domain Specific Language). Ce DSL est spécifique aux réseaux de transport public par trams et est transposable à d'autres réseaux de transport public par voie ferrée. L'utilisation de la méthode MDA et la transformation des modèles *via* leur métamodèle ont ainsi conduit à la réalisation de ce DSL. Les modèles crées grâce au DSL sont transformés, via une sémantique clairement définie, en réseaux de Petri. Le modèle de réseau de Petri obtenu et une propriété spécifique aux réseaux de transport public par tram, sont fournis en entrée à un model-checker qui va explorer algorithmiquement toutes les configurations possibles du réseau, pour, soit le *valider* pour cette propriété, soit fournir un contre-exemple.

Le travail décrit dans ce document n'a pas la prétention d'être exhaustif et laisse de nombreuses pistes de réflexions et d'études sur les trois principaux thèmes traités dans ce mémoire :

- Thème 1 : Conception et Implémentation d'un DSL.
- Thème 2 : Transformation M2M (modèle vers modèle) et transformation M2T (modèle vers texte).
- Thème 3 : Model-checking.

Ainsi pour le Thème 1, il est important de souligner que le design d'un métamodèle (la syntaxe abstraite du DSL) est une tâche délicate et compliquée, car tous les composants du système et les liens entre eux doivent

être *clairement* énoncés. Les métamodèles de grands systèmes peuvent très vite devenir illisibles et incompréhensibles, c'est pourquoi il faut travailler au niveau de la modularité des métamodèles.

Pour le Thème 2, un certain nombre de points concernant l'implémentation des règles de transformation avec le langage ATL (Atlas Transformation Langage) restent à traiter, notamment au niveau de la mise en oeuvre technique.

Pour le Thème 3, selon un critère de comparaison basé sur l'espace d'états, nous avons analysé deux model-checkers (CPNTools et HELENA). Les résultats obtenus nous ont amené à choisir le model-checker HELENA car la taille de l'espace d'état qu'il peut stocker en mémoire est de l'ordre de $10^8$. L'analyse des deux model-checkers selon leur espace d'états dépend bien évidemment de l'application. Dans notre cas, il était important de pouvoir analyser un grand nombre d'états, ce qui est l'une des caractéristiques des systèmes concurrents tels que les réseaux de transport public par trams. Nous avons spécifié deux propriétés d'atteignabilité pour notre réseau de trams, il serait également intéressant d'entrevoir de valider nos modèles, pour des propriétés exprimées en logique CTL(Computational Tree Logic) ou LTL (Logique Temporelle linéaire).

Enfin, une partie des efforts futurs pourraient être dédiés au développement d'un DSL graphique spécifique aux réseaux de transport public par voie ferrée. Ce type de DSL améliore la communication et le code généré est adapté (si le DSL est bon!!!) Une grande partie du travail serait consacrée au développement de l'interface graphique qui pourrait considérablement être amélioré par des tests d'utilisabilité.

# Bibliographie

[ALPINA] an Algebraic Petri Net Analyser http ://alpina.unige.ch/

[ALE 08] A. MARECHAL, Validation de modèles pour réseaux de Petri Algébriques. Diplôme de Master, Université de Genève, septembre 2008.

[BUC 08] Didier BUCHS, Pascal RACLOZ, Réseaux de Petri :Formalisme, Université de Genève, septembre 2008.

[BUC 09] Didier BUCHS, Algébraic Petri Nets, Université de Genève, 30 Avril 2009.

[MAT 09] M. Risoldi, Domain Specific Languages, Software Modeling and Verification course, Univesity of Geneva, 2009.

[TPG 09] URL http ://www.tpg.ch/

[Petri, 62] Carl ADAM PETRI, "Communication avec des Automates", Thèse de doctorat, Université de Darmstadt, BONN, 1962

[CPNTools] URL http ://wiki.daimi.au.dk/cpntools/_home.wiki

[Model-checking Tools] URL http ://en.wikipedia.org/wiki/Model-checking

[Jen 97] Kurt Jensen. "Colored Petri Nets". Basis Concepts, Analysis Methods and Practical Use. Vol 1 and Vol 2. EATCS monographs on Theoretical Computer Science. Springer-Verlag. Berlin, 1997

[RAM, 74] Ramchandani, Chander. (1974). Analysis of Asynchronous Concurrent Systems by Timed Petri Nets. PhD thesis, MIT, 1974. Laboratory for Computer Science, Cmabridge, MA.

[MER, 74] P. Merlin. A study of the recoverability of computer system. PhD thesis, Université de Californie à Irvine, 1974.

[SIF, 77] Use of Petri nets for performance Evaluation, Proceedings of the Third International Symposium on Measuring, Modelling and Evaluating compure Systeme, 1977.

[CAP, 08] Modèle et Métamodèles, ISBN 978-2-88074-749-7, Presses Polytechniques et Universitaires romandes, 2008.

[OCL, 06] OMG : OCL(Object Constraint Language) 2.0 Specification. Chapter 7. October 2003.

[MDA] OMG Model-Driven Architecture Home Page. http ://www.omg.org/mda

[OMG] OMG Object Management Group Home Page. http ://www.omg.org

[Bez] "Using ATL for checking models" GraMoT, 2005.

[wikitionnaire] URL http ://en.wikipedia.org/wiki/Implementation

[BON 01] Patrice BONHOMME, "Reseaux de Petri P-Temporels : Contributions à la commande robuste", Thèse de doctorat, Université de SAVOIE, 12 Juillet 2001.

[DSL] URL http ://en.wikipedia.org/wiki/ Domain-specific_language#Advantages_and_disadvantages

[APN] URL http ://en.wikipedia.org/wiki/Algebraic_Petri_nets#cite_note-3

[WOL 91] Wolfgang Reisig : Petri nets and Algebraic Specifications. Theor. Comput. Sci, 1991.

[VAU 85] Jacques Vautherin. Un modèle algébrique, basé sur les réseaux de Petri, pour l'étude des systèmes parallèles. Thèse de Doctorat, Univ de Paris-Sud.

[Eva 04] Evangelista S. and Pradat-Peyre J.F. Efficient state space storage in explicit model checking. Technical Report 682, Cedric, CNAM, http ://-cedric.cnam.fr/, 2004.

# *Annexe*

**CUI** Centre Universitaire d'Informatique
**TPG** Transports Publics Genevois
**ALPINA** Algebraic Petri Net Analyser
**RdP** Reseau de Petri
**RdPC** Reseau de Petri coloré
**RdPT** Reseau de Petri temporisé
**TCPN** Timed Colored Petri Net ou Reseau de Petri Temporisé et Coloré
**APN** Algebraic Petri Net ou Reseau de Petri Algebrique
**timed APN** Réseaux de Petri Algébriques temporisés
**ADT** Algebraic Data Type ou Type Abstrait algebrique
**EMF** Eclipse Modeling Framework
**MOF** Meta Object Facility
**CWM** Common Warehouse Metamodel
**OMG** Object Management Group
**MOD** Modèle Objet du Domain
**UML** Unified Modeling Language
**OCL** Object Constraint Language
**MDA** Model Driven Architecture
**CIM** Computation Independent Model
**PIM** Platform Independent Model
**PSM** Platform Specific Model
**GPL** General Programming Language
**ATL** Atlas Transformation Language
**DSL** Domain Specific language
**[C/1, 1]** Croisement 1 entrée et 1 sortie
**[C/1, 2]** Croisement 1 entrée et 2 sorties
**[C/2, 1]** Croisement 2 entrées et 1 sortie
**[C/2, 2]** Croisement 2 entrées et 2 sorties
**[C/n, m]** Croisement n entrées et m sorties