B événementiel pour la modélisation du domaine : application au transport

Atif Mashkoor, Jean-Pierre Jacquot et Jeanine Souquières

LORIA – Nancy Université
Campus Scientifique, BP 239,
F-54506, Vandœuvre lès Nancy, France
{Prénom.Nom}@loria.fr

Résumé Ce papier rapporte sur l'utilisation de B événementiel pour la modélisation du domaine dans le cas particulier des transports[1]. Le développement de nouveaux systèmes de transports urbains, tels que les véhicules autonomes en libre-service, pose des problèmes difficiles d'homologation, d'acceptabilité, de sécurisation et de certification. La spécification formelle de ces systèmes comme de leur environnement (ou domaine) est incontournable. Nous rapportons les premières observations réalisées lors de la modélisation d'un domaine complexe en utilisant un langage formel non spécifiquement conçu pour cet usage. Au delà des observations techniques sur l'outillage, Rodin en l'occurrence, nous mettons en évidence des points d'ordre méthodologique, en particulier la nécessité de distinguer la notion de raffinement de celle de niveaux d'observation.
Mots-clés : B événementiel, domaine du transport, propriétés, vérification.

1 Introduction

La bonne compréhension du domaine d'une application donnée est un prérequis de base pour le développement du logiciel [1]. Le modèle d'un domaine désigne un modèle conceptuel qui décrit diverses entités, phénomènes et leurs relations, ainsi qu'un ensemble de propriétés statiques et dynamiques importantes. Il peut être exprimé sous forme de besoins, de spécifications ou de références architecturales. Généralement, un tel modèle est utilisé pour vérifier et valider la compréhension du domaine d'un problème donné par différents utilisateurs [2, 3].

Pour faciliter leur utilisation et guider le développement de logiciels, les modèles de domaine devraient être capturés et spécifiés de façon systématique [4], en utilisant des méthodes formelles. Bien que les méthodes formelles demandent des efforts importants dans leur mise en œuvre et qu'elles soient consommatrices de temps, des études récentes ont montré tout l'intérêt de leur utilisation dans les phases amont du développement de logiciels quant à la réduction des erreurs [5].

Dans [6], Jackson présente les besoins d'un système en termes de relations entre les phénomènes du domaine et introduit la nécessité d'exprimer les propriétés du domaine. Avec les schémas de problèmes [7], il introduit explicitement les domaines de l'application et les domaines de la machine à réaliser.

Cette idée d'introduire explicitement la notion de domaine a été reprise par Bjørner [8, 9] : la correction du logiciel est fortement dépendante du modèle

[1] Ce travail est réalisé dans le cadre des projets TACOS, ANR-06-SETI-017 (http ://tacos.loria.fr) et CRISTAL, pôle de compétitivité Alsace/Franche-Comté (http ://www.projet-cristal.net)

de domaine et de l'expression des besoins du logiciel. La sûreté est également l'un des facteurs principaux qui ne doit pas être négligé lors de la conception de systèmes critiques complexes. Le développement de systèmes sûrs et corrects est une tâche difficile ; les méthodes traditionnelles sont sources d'erreurs. L'utilisation de méthodes formelles est une aide importante pour le développement de logiciels, permettant d'assurer la correction et la structuration du développement, depuis la modélisation du domaine jusqu'à son implantation. Comme le montre l'étude de cas des autoroutes automatisées [10, 11, 12], la spécification de tels systèmes cirtiques est d'une grande complexité.

Nous proposons ici l'utilisation de B événementiel pour modéliser le domaine. Bien qu'il n'ait pas été conçu pour cette utilisation, nous pensons que la technique d'expression des systèmes basée sur un modèle abstrait et son raffinement en utilisant des événements est appropriée pour la modélisation de domaines. Nous travaillons dans la perspective du développement de systèmes de transport urbain basé sur des véhicules autonomes. B événementiel a aussi été utilisé pour spécifier le développement de systèmes multi-agents situés avec application au problème du *platooning* ou accrochage immatériel de véhicules [13]. L'utilisation d'un même langage d'expression pour le domaine et pour un sous-système facilitera la confrontation de l'un à l'autre.

Cet article est organisé comme suit. La section 2 présente une vue d'ensemble de l'approche B événementiel. La section 3 décrit le domaine des transports et les principales entités nécessaires à sa modélisation. La section 4 présente la spécification du domaine en termes de raffinement. La section 5 introduit la prise en compte de la notion de temps de trajet dans le modèle précédent. La hiérarchie du modèle B evénementiel obtenue avec ses trois niveaux d'abstraction et l'introduction du temps est décrite dans la section 6. La section 7 présente l'expérience acquise avec cette modélisation formelle du domaine de transport en distinguant les observations sur les outils, le langage et le processus de formalisation utilisés. Une conclusion et les suites de ce travail terminent ce papier.

2 B événementiel

B [14] est une méthode formelle fondée sur la théorie des ensembles, permettant un développement incrémental de logiciels séquentiels grâce au raffinement. Elle a été appliquée avec succès dans le développement d'applications réelles complexes, notamment dans le domaine du transport avec le projet METEOR [15] et le métro Val [16]. Elle s'appuie sur des outils robustes [17, 18]. B événementiel [19] est une évolution de la méthode B pour la modélisation globale d'un système avec observation de son comportement.

Spécifications abstraites. Une spécification écrite en B événementiel contient les données dynamiques du modèle. Elle est encapsulée dans une **MACHINE** identifiée par un nom unique. Les variables sont données dans la clause **VARIABLES** et initialisées dans la clause **INITIALISATION**. Un **INVARIANT** définit l'espace d'état des variables et les propriétés du système.

Chaque événement de la clause **EVENTS** est une substitution ; sa sémantique est celle du calcul des plus faibles préconditions de Dijkstra [20]. Un événement est constitué d'une garde et d'un corps. Quand la garde est satisfaite, l'événement peut être activé. Lorsque les gardes de plusieurs événements sont satisfaites en

même temps, le choix de l'événement à activer est indéterministe. Un exemple de spécification abstraite est donné figure 2 avec la machine Movement.

Une machine peut voir au moins un contexte introduit par la clause **SEES**. Un **CONTEXT** permet de spécifier des données statiques. Il se compose d'ensembles, de constantes avec leurs axiomes et éventuellement de théorèmes. Deux exemples de contextes sont proposés dans la partie droite de la figure 2 avec StartState et Location, le premier étant défini par extension du second.

Des obligations de preuves sont générées par Rodin[2] pour vérifier la consistence du modèle, c'est-à-dire la préservation de l'invariant par les différents événements.

Raffinement. Comme en B classique, le concept de raffinement est la caractéristique principale de B événementiel. Un modèle abstrait est transformé progressivement en un modèle plus concret et plus élaboré. De nouvelles variables peuvent être introduites et les anciennes raffinées en des variables plus concrètes. La clause **WITH** permet d'exprimer le lien entre les paramètres d'un événement abstrait et sa concrétisation, les paramètres abstraits pouvant être supprimés de l'événement raffiné. De nouveaux événements peuvent aussi être introduits, ils ne doivent pas empêcher les anciens d'être activés. Cette contrainte est satisfaite par l'introduction d'un **VARIANT** ; il s'agit d'une expression entière décrémentée par chaque nouvel événement.

Un exemple de raffinement est présenté figure 4 avec la machine Movement1 dans laquelle un nouveau contexte StartState1 est introduit.

Les obligations de preuves permettent de vérifier que le modèle raffiné est consistent, c'est-à-dire que son **INVARIANT** est vérifié et que son **VARIANT** est décrémenté par les nouveaux événements. Elles permettent aussi de vérifier que le raffinement est correct, c'est-à-dire que chaque événement concret ne contredit pas l'événement abstrait correspondant.

3 Présentation du domaine

Notre travail se déroule dans le cadre des projets CRISTAL (pôle de compétitivité d'Alsace/Franche-Compté) et TACOS (ANR). L'objectif de ces projets est de concevoir de nouveaux systèmes de transport urbain utilisant des véhicules électriques autonomes en libre-service, les `CyCab` [21]. Les problèmes à résoudre sont nombreux : utilisabilité, insertion dans le tissu urbain, gestion du trafic, etc. Les plus importants concernent toutefois l'homologation et la certification de tels systèmes pour lesquels aucune norme n'existe.

Le terme « transport » se rapporte au mouvement de personnes et de marchandises en utilisant des véhicules pour aller d'un endroit à un autre [9]. De cette définition simple, on peut déjà déduire qu'il faut spécifier des notions telles que les véhicules, les lieux, la localisation, ainsi que l'événement correspondant à un mouvement.

Dans cet article, nous avons choisi de considérer le transport sur un réseau. Celui-ci est modélisé comme un graphe orienté dont nous appellerons les sommets hubs et les arêtes paths. Nous ne considérons pour l'instant qu'un seul

[2] La plate-forme Rodin, support pour la spécification et la vérification de spécifications B événementielles, est développée dans le cadre du projet de recherche Européen UE IST511599 (`http://rodin-b-sharp.sourceforge.net`

réseau. Les hubs peuvent être de deux catégories : des junctions qui modèlisent les intersections ou des stations qui modèlisent les lieux où les passagers et les marchandises peuvent entrer et sortir des véhicules. Un déplacement correspond à une action par laquelle un véhicule initialement arrêté à une station en rejoint une autre et s'y arrête. Le déplacement est contraint par la topologie du réseau, il est donc nécessaire d'introduire une notion de route qui modélise un itinéraire. La figure 1 montre un réseau routier.

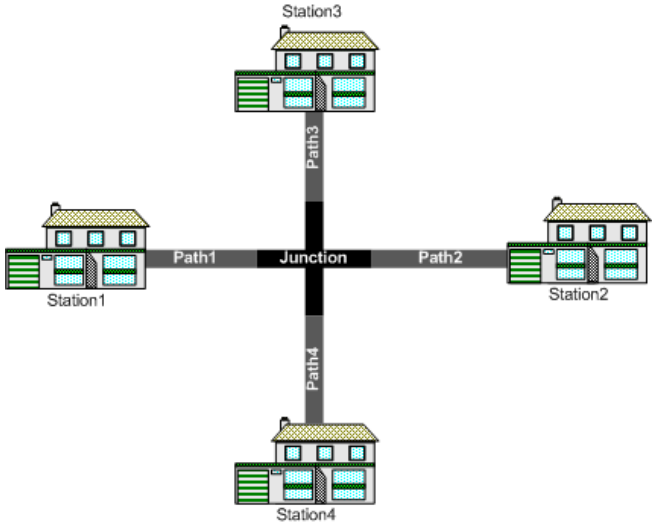


Fig. 1. Un réseau routier

L'apparente simplicité de la description intuitive du domaine ne doit pas masquer la complexité qui s'y cache. En particulier, plusieurs véhicules circulent simultanément et leurs interactions conditionnent de fait les propriétés du système global. Nous nous intéressons ici à deux propriétés : l'évitement des collisions aux intersections et la détermination des temps de trajet.

4 Spécification du domaine

La complexité du domaine nous a incité à décomposer notre analyse en plusieurs niveaux d'observation, dont trois sont présentés ici :

- Le premier niveau a pour rôle de fixer la topologie et les concepts importants du domaine. Techniquement, ce sont les contextes qui y sont importants, l'événement travel est simple.
- Le deuxième niveau introduit l'idée qu'un déplacement est réalisé par une suite d'actions contrôlées par la topologie du réseau. Deux événements, crossHub et traversePath, sont introduits pour décomposer travel.
- Le troisième niveau introduit la notion d'interactions entre les véhicules. Nous ne traitons ici que le cas des interactions aux hubs. crossHub est ainsi décomposé en trois événements : enterHub, leaveHub et wait.

Il faut noter que niveau d'observation et raffinement sont deux notions distinctes et indépendantes. Le niveau d'observation est une notion qui permet d'organiser et de structurer la description du domaine, en particulier celle des différentes propriétés d'intérêt. Il peut être constitué de plusieurs raffinements successifs. Le raffinement est une notion mathématique qui permet d'exprimer petit à petit des propriétés complexes en simplifiant ainsi leur vérification formelle.

4.1 Modèle initial

Le modèle initial sert principalement à fixer le vocabulaire et les définitions sur lesquels la spécification générale est construite. Il est constitué d'une machine qui comporte un seul événement simpliste, travel et un ensemble de contextes :

- Net décrit la topologie du réseau modélisé.
- Location spécifie la notion de localisation dans le réseau.
- Vehicle spécifie les propriétés des véhicules qui circulent dans le réseau.
- StartState spécifie l'état initial du réseau, en particulier, la localisation des véhicules.

Comme présenté figure 2, la machine Movement est extrêmement simple. Son seul événement indique qu'un véhicule se déplace d'un point à un autre.

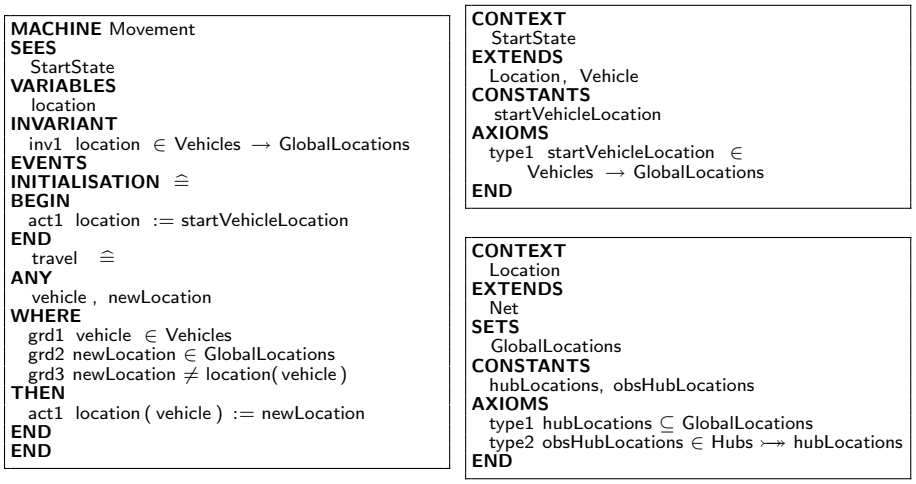


Fig. 2. Modèle initial

L'essentiel de l'effort de spécification porte sur les contextes. Le contexte Net présenté figure 3 décrit le réseau comme un graphe augmenté de quelques propriétés spécifiques à un réseau de transport. Par exemple, nous demandons que tous les hubs d'un réseau soient connectés (prop1) ; nous interdisons qu'un connexion soit réflexive (prop3). Même si les concepts de déplacements sur plusieurs réseaux ou d'interconnexions ne sont pas développés dans ce papier, la spécification du contexte prépare ce développement. Nous ne montrons ici que les

axiomes pertinents pour la discussion (la spécification complète avec les axiomes relatifs aux graphes orientés est disponible à l'adresse internet suivante `http ://www.loria.fr/~mashkooa/eventb.htm`).
A titre d'exemples de propriétés exprimées par les axiomes :

- type1 exprime qu'un hub peut appartenir à plusieurs réseaux,
- type2 exprime qu'une connection n'appartient qu'à un seul réseau,
- prop4 exprime qu'un réseau n'existe que s'il y au moins deux hubs et une connection.

Comme on peut le voir figure 3, nous avons choisi de classer les axiomes en utilisant un mécanisme de préfixe pour construire les noms : tech_n référence un axiome technique nécessaire lors des preuves, type_n référence un axiome de typage et prop_n une propriété invariante des constantes. La troisième catégorie contient les propriétés les plus intéressantes, alors que la première catégorie ne donne que des trivialités.

CONTEXT
Net
SETS
Nets, Hubs, Connections
CONSTANTS
obsNetHubs, obsNetConnections, connectionOrigin, connectionDestination
AXIOMS

tech1	finite (Nets)
tech2	finite (Hubs)
tech3	finite (Connections)
tech4	finite (obsNetConnections)
tech5	finite (obsNetHubs)
type1	$obsNetHubs \in Hubs \leftrightarrow Nets$
type2	$obsNetConnections \in Connections \rightarrow Nets$
type3	$connectionOrigin \in Connections \rightarrow Hubs$
type4	$connectionDestination \in Connections \rightarrow Hubs$
prop1	$dom(obsNetHubs) = Hubs \land ran(obsNetHubs) = Nets$
prop2	$ran(obsNetConnections) = Nets$
prop3	$\forall c \cdot c \in Connections \Rightarrow connectionOrigin(c) \neq connectionDestination(c)$
prop4	$\forall n \cdot n \in Nets \Rightarrow card(obsNetHubs^{-1}[\{n\}]) \geq 2 \land card(obsNetConnections^{-1}[\{n\}]) \geq 1$

END

Fig. 3. Le contexte Net

Les contextes Vehicle et Location sont quasi vides. Ils sont néanmoins essentiels pour structurer la démarche. Intuitivement, nous savons qu'il faut considérer des véhicules et qu'il faudra affiner leurs caractéristiques. De même, nous savons qu'il faut un mécanisme de localisation.

Le contexte StartState n'est présent à ce niveau que par un souci d'homogénéité et de régularité de la spécification. Au cours des raffinements, il est apparu que nous avions un problème technique pour spécifier les initialisations. Intuitivement, certaines valeurs dépendent de la position initiale des véhicules dans le réseau qui doivent donc être définies avant. Ceci n'est pas spécifiable dans une clause **THEN** puisque toutes les actions sont supposées être réalisées en même temps. La spécification de l'état initial, en particulier de la position initiale des véhicules, comme un contexte permet de résoudre le souci technique.

4.2 Premier raffinement

Le premier raffinement consiste à préciser que la localisation et les mouvements sont centrés sur les hubs, et plus précisemment sur les stations. Ainsi, les points de départ et d'arrivée de travel sont maintenant des stations. Cette idée est matérialisée par l'introduction de deux nouvelles stations origin et destination correspondant respectivement au hub de départ et au hub d'arrivée, et par la modification de la garde grd2 en grd2' qui précise le hub d'arrivée, voir figure 4.

Le contexte Net évolue par extension[3] en Net1 par le partitionnement de l'ensemble Hubs en deux : stations et junctions. Les axiomes définissent la partition. StartState évolue en StartState1.

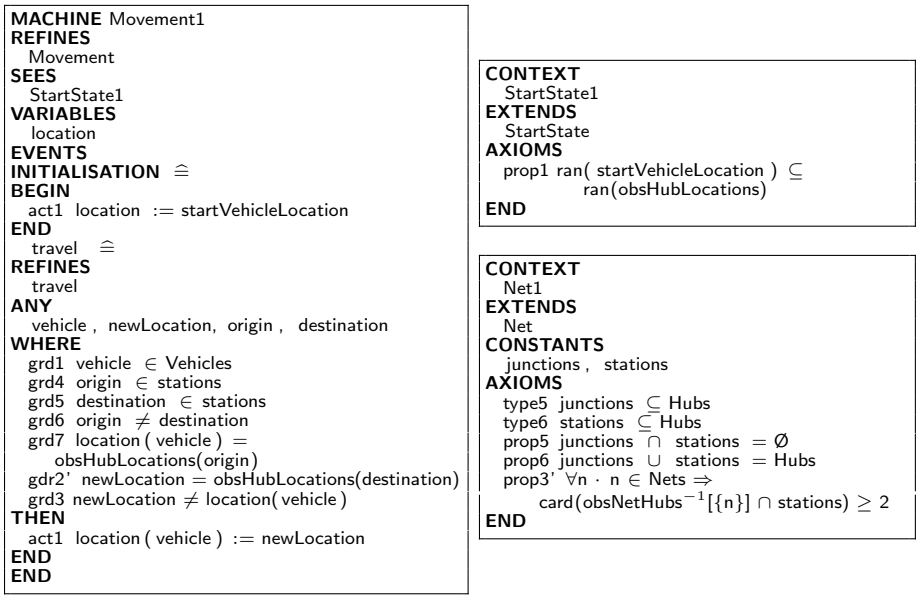


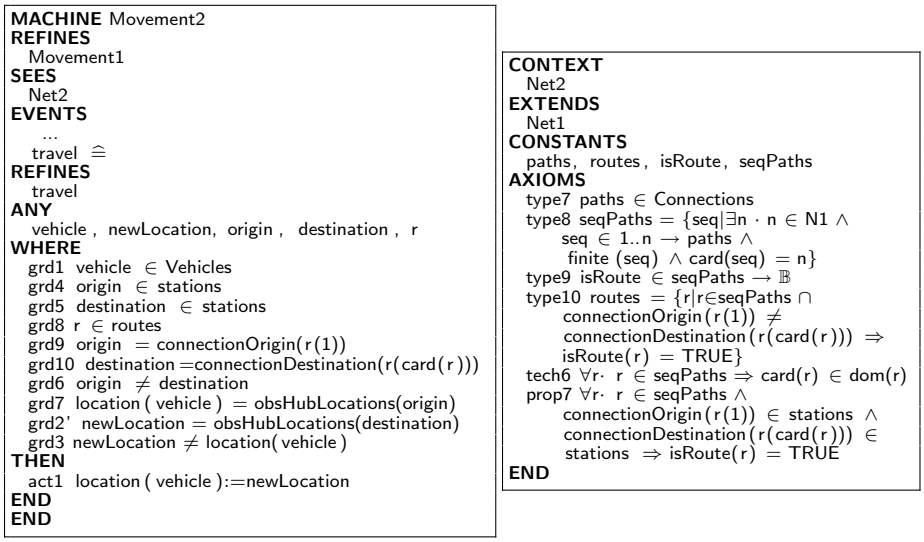
Fig. 4. Premier raffinement

4.3 Deuxième raffinement

Ce raffinement consiste à préciser la topologie du réseau. La notion de paths est introduite pour représenter une voie qui relie directement deux hubs adjacents (c'est une arête dans le graphe). Les connexions deviennent alors des routes : une suite de paths. Ce raffinement est très technique du fait de l'absence de la notion de suites en B événementiel. Il faut définir celles-ci explicitement. La figure 5 montre le raffinement de travel indiquant simplement que départ et arrivée sont

[3] En B événementiel, les machines évoluent par *raffinement*, alors que les contextes évoluent par *extention*. D'un point de vue méthodologique, il s'agit de constructions similaires.

connectés par une route. Net1 est étendu en Net2 avec introduction de la définition des séquences ainsi que du prédicat caractéristique des routes (suite de paths adjacents, dont le premier et le dernier hub sont des stations).



```
MACHINE Movement2
REFINES
  Movement1
SEES
  Net2
EVENTS
    ...
  travel ≙
REFINES
  travel
ANY
  vehicle , newLocation, origin , destination , r
WHERE
  grd1 vehicle ∈ Vehicles
  grd4 origin ∈ stations
  grd5 destination ∈ stations
  grd8 r ∈ routes
  grd9 origin = connectionOrigin(r(1))
  grd10 destination =connectionDestination(r(card(r)))
  grd6 origin ≠ destination
  grd7 location ( vehicle ) = obsHubLocations(origin)
  grd2' newLocation = obsHubLocations(destination)
  grd3 newLocation ≠ location( vehicle )
THEN
  act1 location ( vehicle ):=newLocation
END
END
```

```
CONTEXT
  Net2
EXTENDS
  Net1
CONSTANTS
  paths, routes, isRoute, seqPaths
AXIOMS
  type7 paths ∈ Connections
  type8 seqPaths = {seq|∃n · n ∈ N1 ∧
       seq ∈ 1..n → paths ∧
        finite (seq) ∧ card(seq) = n}
  type9 isRoute ∈ seqPaths → ℙ
  type10 routes = {r|r∈seqPaths ∩
       connectionOrigin(r(1)) ≠
       connectionDestination(r(card(r))) ⇒
       isRoute(r) = TRUE}
  tech6 ∀r· r ∈ seqPaths ⇒ card(r) ∈ dom(r)
  prop7 ∀r· r ∈ seqPaths ∧
       connectionOrigin(r(1)) ∈ stations ∧
       connectionDestination(r(card(r))) ∈
        stations ⇒ isRoute(r) = TRUE
END
```

Fig. 5. Deuxième raffinement

4.4 Troisième raffinement

La définition générale d'un voyage étant posée, cette étape consiste à observer comment il est réalisé. Le raffinement consiste à exprimer un voyage comme une suite de déplacements plus précis, en prenant en compte la traversée des paths et le franchissement des hubs. Nous observons donc le déplacement à un niveau plus fin.

En B événementiel, il n'est pas possible d'exprimer directement qu'un événement est décomposé en une séquence ordonnée d'événements « plus petits ». La technique utilisée consiste à introduire de nouveaux événements indépendants de l'événement décomposé (c'est-à-dire qui ne soient pas dans une relation de raffinement) et à déterminer les gardes de sorte que le déclenchement du dernier petit événement provoque le déclenchement de l'événement décomposé. Une difficulté suplémentaire, propre à notre modèle, est que l'ordre des événements est important.

Le problème est réglé par l'introduction de trois événements (crossHub, traverse Path et startTravel) et de deux ensembles de contrôle (pathsToTraverse et hubsToCross). Ces deux ensembles sont construits par l'événement startTravel qui introduit l'ensemble de paths et de hubs que le véhicule devra parcourir. Les événements traversePath et crossHub sont respectivement associés à un path et à un hub ; une action importante de ces événements consiste à retirer le path

(respectivement le hub) de l'ensemble adéquat. Il est ainsi possible de contrôler finement la progression du véhicule. La figure 6 présente ces deux événements. Le schéma de décomposition exprimant la dynamique de ces événements peut être décrit à l'aide de l'expression régulière suivante :

$$travel \ \equiv \ (startTravel;\ (crossHub;\ traversePath)+)$$

Un point important à noter est qu'il a fallu redéfinir la variable location correspondant à la localisation des véhicules. La variable position est utilisée dans traversePath et crossHub ; elle joue le même rôle et possède le même profil que la variable location dans l'événement travel . En fait, ces deux variables représentent bien la même notion intuitive et il est tentant de n'en utiliser qu'une seule. C'est malheureusement impossible si on souhaite conserver la définition de travel comme un déplacement d'un point à un autre. En effet le dernier traversePath amène le véhicule à la station finale et il n'est alors plus possible d'identifier un point de départ différent du point d'arrivée pour travel .

MACHINE Movement3
REFINES
Movement2
...
VARIABLES
pathsToTraverse, hubsToCross, position
INVARIANT
inv1 pathsToTraverse $\subseteq$ Vehicles $\times$ Connections
inv2 hubsToCross $\subseteq$ Vehicles $\times$ Hubs
inv3 position $\in$ Vehicles $\rightarrow$ GlobalLocations
EVENTS
...
traversePath $\widehat{=}$
ANY
vehicle , r , p, newPosition
WHERE
grd1 vehicle $\in$ Vehicles
grd2 r $\in$ routes
grd3 p $\in$ paths $\wedge$ p $\in$ ran(r)
grd4 position (vehicle) $\in$ obsHubLocations(connectionOrigin(p))
grd5 newPosition $\in$ obsHubLocations(connectionDestination(p))
grd6 newPosition $\neq$ position(vehicle)
grd7 vehicle$\mapsto$p $\in$ pathsToTraverse
grd8 vehicle$\mapsto$connectionOrigin(p) $\in$ hubsToCross
grd9 vehicle$\mapsto$connectionDestination(p) $\in$ hubsToCross
THEN
act1 position (vehicle) := newPosition
act2 pathsToTraverse := pathsToTraverse $\setminus$ {vehicle$\mapsto$p}
END
crossHub $\widehat{=}$
ANY
vehicle , hub
WHERE
grd1 vehicle $\in$ Vehicles
grd2 hub $\in$ Hubs
grd3 position (vehicle) $\in$ obsHubLocations(hub)
grd4 vehicle$\mapsto$hub $\in$ hubsToCross
THEN
act1 hubsToCross := hubsToCross $\setminus$ {vehicle$\mapsto$hub}
END
END

Fig. 6. Introduction des événements traversePath et crossHub

4.5 Quatrième raffinement

Les développements précédents concernent le déplacement d'un seul véhicule. Ici, nous abordons les interactions entre véhicules qui circulent sur un réseau. Une propriété éminemment souhaitable de notre modèle est l'absence de collision entre véhicules. Nous étudions ici une version simplifiée de cette propriété :

- l'étude est restreinte aux hubs,
- avec la définition abstraite suivante : « il y a collision s'il y a plus de véhicules dans un hub que celui-ci ne peut en contenir. »

```
MACHINE Movement4
REFINES
  Movement3
SEES
  Net3
  ...
VARIABLES
  hubload
INVARIANT
  inv1 hubload ∈ Hubs → N
  inv2 ∀h · h ∈ Hubs⇒hubLoad(h) ≤ hubCapacity(h)
EVENTS
  enterHub ≙
ANY
  vehicle , hub
WHERE
  grd1 vehicle ∈ Vehicles
  grd2 hub ∈ Hubs
  grd3 position ( vehicle ) = obsHubLocations(hub)
  grd4 vehicle ↦ hub ∈ hubsToCross
  grd5 hubLoad(hub)<hubCapacity(hub)
THEN
  act1 position ( vehicle ) := position(vehicle)
  act2 hubLoad(hub) := hubLoad(hub) + 1
END
  leaveHub ≙
ANY
  vehicle , hub
WHERE
  grd1 vehicle ∈ Vehicles
  grd2 hub ∈ Hubs
  grd3 position ( vehicle ) ∈ obsHubLocations(hub)
  grd4 vehicle↦hub ∈ hubsToCross
THEN
  act1 position ( vehicle ) := position(vehicle)
  act2 hubLoad(hub) := hubLoad(hub)−1
END
  wait ≙
ANY
  vehicle , hub
WHERE
  grd1 vehicle ∈ Vehicles
  grd2 hub ∈ Hubs
  grd3 position ( vehicle ) = obsHubLocations(hub)
  grd4 vehicle↦hub ∈ hubsToCross
  grd5 hubLoad(hub) ≥ hubCapacity(hub)
THEN
  act1 position ( vehicle ) := position(vehicle)
END
END
```

Fig. 7. Quatrième raffinement

```
MACHINE Movement5
REFINES
  Movement4
  ...
VARIABLES
  travelTime, time
INVARIANT
  inv1 travelTime ∈ Vehicles → N
  inv2 time ∈ N
EVENTS
  travel ≙
REFINES
  travel
ANY
    ...
  startTime
WHERE
    ....
  grd9 startTime ∈ N
  grd10 startTime < time
THEN
    ...
  act2 travelTime( vehicle ) := time − startTime
END
  ticTac ≙
ANY
  t
WHERE
  grd1 t ∈ N
  grd2 t > time
THEN
  act1 time := t
END
END
```

Fig. 8. La notion de temps

Nous adoptons la même approche que dans le raffinement précédent : nous passons à un niveau d'observation plus fin en décomposant l'événement crossHub.

La décomposition implique trois nouveaux événements : enterHub, leaveHub et wait. Ce dernier événement modélise l'idée qu'un véhicule ne peut entrer dans un hub que s'il en a la place. enterHub et wait sont deux événements exclusifs, leurs gardes sont dans un rapport de stricte négation, comme le montre la figure 7 (voir grd5). Le schéma de décomposition exprimant la dynamique de ces événements peut être décrit à l'aide de l'expression régulière suivante :

$$crossHub \equiv (wait*;\ enterHub;\ leaveHub)$$

Le contrôle d'entrée se fait par observation de l'état du hub, c'est-à-dire du nombre de véhicules qui l'occupent. Ceci est modélisé par une variable hubLoad incrémentée et décrémentée par les évévements enterHub et leaveHub. Le contexte Net2 est étendu pour introduire la notion de capacité d'un hub (hubCapacity) qui indique la charge maximale admissible d'un hub.

L'expression de la propriété d'absence de collision s'exprime naturellement par l'**INVARIANT** de la machine Movement4.

5 Introduction d'une nouvelle propriété : le temps

La notion de temps joue un rôle très important dans le domaine des transports. Par exemple :

- C'est par rapport au temps dont il dispose qu'un utilisateur choisit son itinéraire.
- Un système de transport doit chercher à optimiser les temps de trajet ou à les garantir.

Nous présentons ici la façon dont nous prenons en considération la notion de temps de trajet dans le modèle. La modélisation est effectuée en deux étapes. Nous donnons d'abord une définition du temps de trajet. Ensuite, nous modélisons la manière de déterminer ce temps. Pour cela, nous utilisons la technique de contraintes temporelles développée par Cansell et al. [22, 23] et en nous appuyant sur la structure de la spécification existante.

5.1 Première étape

Conceptuellement, la définition du temps de trajet relève du premier niveau d'observation. La machine Movement est enrichie par les éléments suivants :

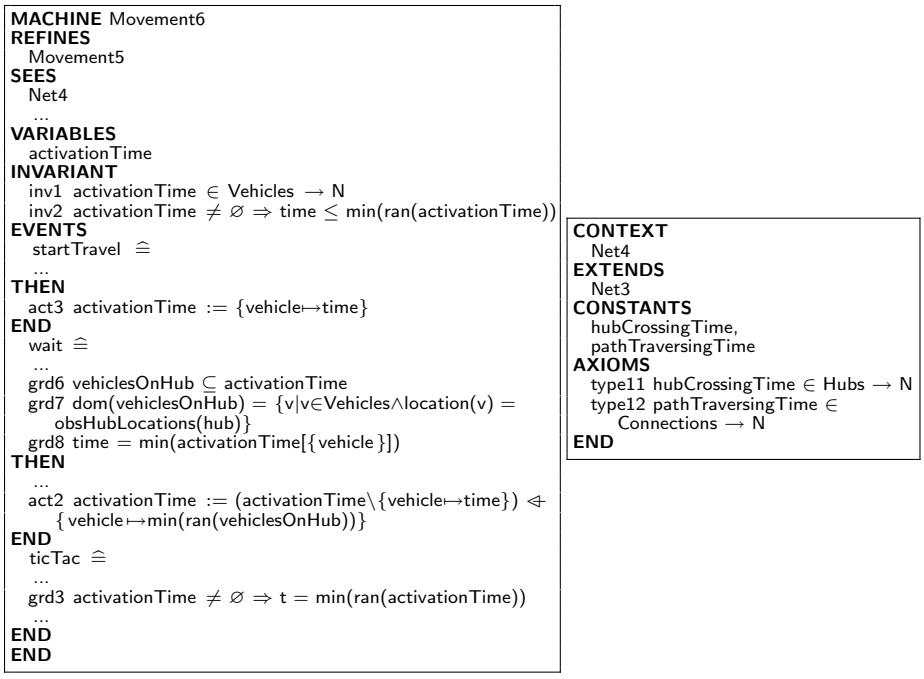
- Deux nouvelles variables, travelTime et time, sont introduites. La première est représentée par une fonction qui enregistre le temps de déplacement des véhicules. La seconde modélise l'horloge.
- Une nouvelle action est introduite dans l'événement travel. Elle calcule travelTime(vehicle) comme la différence entre le temps courant (time) et le temps de départ (startTime). Ce dernier est simplement supposé exister ; il est inférieur au temps courant.
- Un nouvel événement, ticTac, fait avancer le temps courant.

La figure 8 montre les évolutions dans la machine. Techniquement, comme Rodin ne permet pas de gérer des « raffinements parallèles », les modifications sont apportées sous la forme d'un raffinement de la machine Movement4.

5.2 Deuxième étape

Les définitions précédentes doivent maintenant être complétées en répondant à deux questions :

- Comment le temps est-il associé aux événements ?
- Comment l'horloge évolue-t-elle ?



MACHINE Movement6
REFINES
Movement5
SEES
Net4
...
VARIABLES
activationTime
INVARIANT
inv1 activationTime $\in$ Vehicles $\rightarrow$ N
inv2 activationTime $\neq \varnothing \Rightarrow$ time $\leq$ min(ran(activationTime))
EVENTS
startTravel $\widehat{=}$
...
THEN
act3 activationTime := {vehicle$\mapsto$time}
END
wait $\widehat{=}$
...
grd6 vehiclesOnHub $\subseteq$ activationTime
grd7 dom(vehiclesOnHub) = {v|v$\in$Vehicles$\wedge$location(v) = obsHubLocations(hub)}
grd8 time = min(activationTime[{vehicle}])
THEN
...
act2 activationTime := (activationTime$\setminus${vehicle$\mapsto$time}) $\lhd$ {vehicle$\mapsto$min(ran(vehiclesOnHub))}
END
ticTac $\widehat{=}$
...
grd3 activationTime $\neq \varnothing \Rightarrow$ t = min(ran(activationTime))
...
END
END

CONTEXT
Net4
EXTENDS
Net3
CONSTANTS
hubCrossingTime,
pathTraversingTime
AXIOMS
type11 hubCrossingTime $\in$ Hubs $\rightarrow$ N
type12 pathTraversingTime $\in$ Connections $\rightarrow$ N
END

Fig. 9. Raffinement du temps

Pour cela, nous avons repris les principes mis en œuvre dans les techniques de simulation de systèmes à files d'attente. L'élément central est un échéancier (activationTime) qui contient, pour chaque véhicule en mouvement, la date du prochain événement qui le concerne. Le déclenchement d'un événement daté pour un véhicule donné est alors contraint : il faut que le couple vehicle $\mapsto$ time soit dans l'échéancier. Les événements datés ont une action sur l'échéancier : ils y introduisent la date du prochain événement pour le véhicule concerné. La détermination de la date future nécessite que l'on possède une opération de calcul de la durée de chaque événement. L'échéancier pilote également l'horloge : celle-ci avance jusqu'à la date du prochain événement.

Conceptuellement, nous avons choisi de travailler au troisième niveau d'observation. C'est en effet à ce niveau qu'il est possible de déterminer la durée des événements. C'est également à ce niveau que l'événement wait, le plus intéressant du point de vue de l'observation du temps de trajet, est défini. L'évolution de la spécification découle directement de l'analyse précédente :

- Le contexte Net3 est étendu avec la définition de deux constantes, hubCrossing−Time et pathTraversingTime, qui donnent pour chaque véhicule et chaque hub (respectivement path), le temps mis à le franchir.
- La variable activationTime qui modélise l'échéancier est introduite.
- L'événement ticTac est raffiné pour que l'incrément de l'horloge soit calculé à partir de l'échéancier.
- Pour chacun des événements du troisième niveau d'observation, wait, enter−Hub et leaveHub, les modifications suivantes sont introduites, comme présenté sur l'événement wait de la figure 9 :
  - Une garde est introduite pour contraindre l'événement à ne se déclencher qu'à la bonne date (voir grd6, grd7 et grd8).
  - Une action est ajoutée pour modifier l'échéancier en calculant la date de l'événement suivant (voir act2).
- L'événement technique startTravel est raffiné par l'introduction d'une action initialisant l'échéancier relativement au véhicule qui démarre le voyage.

La figure 9 montre la réalisation en B événementiel, techniquement écrite comme un raffinement de la machine Movement5.

6 Présentation de la hiérarchie du modèle

La hiérarchie B événementielle du modèle de domaine de transport obtenue par le processus de développement décrit dans les sections 4 et 5 est présentée figure 10. Elle contient les différents contextes et machines spécifiques au modèle. Nous la présentons sous la forme de quatre colonnes :

- La deuxième concerne la construction principale du modèle, avec la machine Movement et ses différents raffinements.
- La troisième concerne la description de la localisation des véhicules avec les différents raffinements. Les contextes StartState sont dédiés à l'initialisation des variables.
- La quatrième concerne la description de la topologie du réseau avec le contexte Net et l'ensemble de ses différents raffinements.
- La première concerne l'introduction du temps en deux étapes.

Dans la hiérarchie, apparaissent aussi les différents niveaux d'observation pour les colonnes :

- Les quatre premières rangées de la hiérarchie représentent le premier niveau d'observation, fixant la topologie du réseau et les concepts du domaine.
- La cinquième rangée correspond au deuxième niveau d'observation, dans lequel un déplacement est réalisé par une suite d'actions contrôlées par la topologie du réseau.
- La sixième rangée correspond au troisième niveau d'observation, avec l'introduction des interactions entre véhicules.

Les sauts de niveau d'observation sont matérialisés par des lignes en pointillé.

7 Observations

La modélisation de domaines diffère de la spécification de systèmes sur plusieurs aspects : les propriétés du monde réel doivent être exprimées au plus haut niveau, l'affinage n'est pas dirigé vers la concrétisation d'une implantation mais

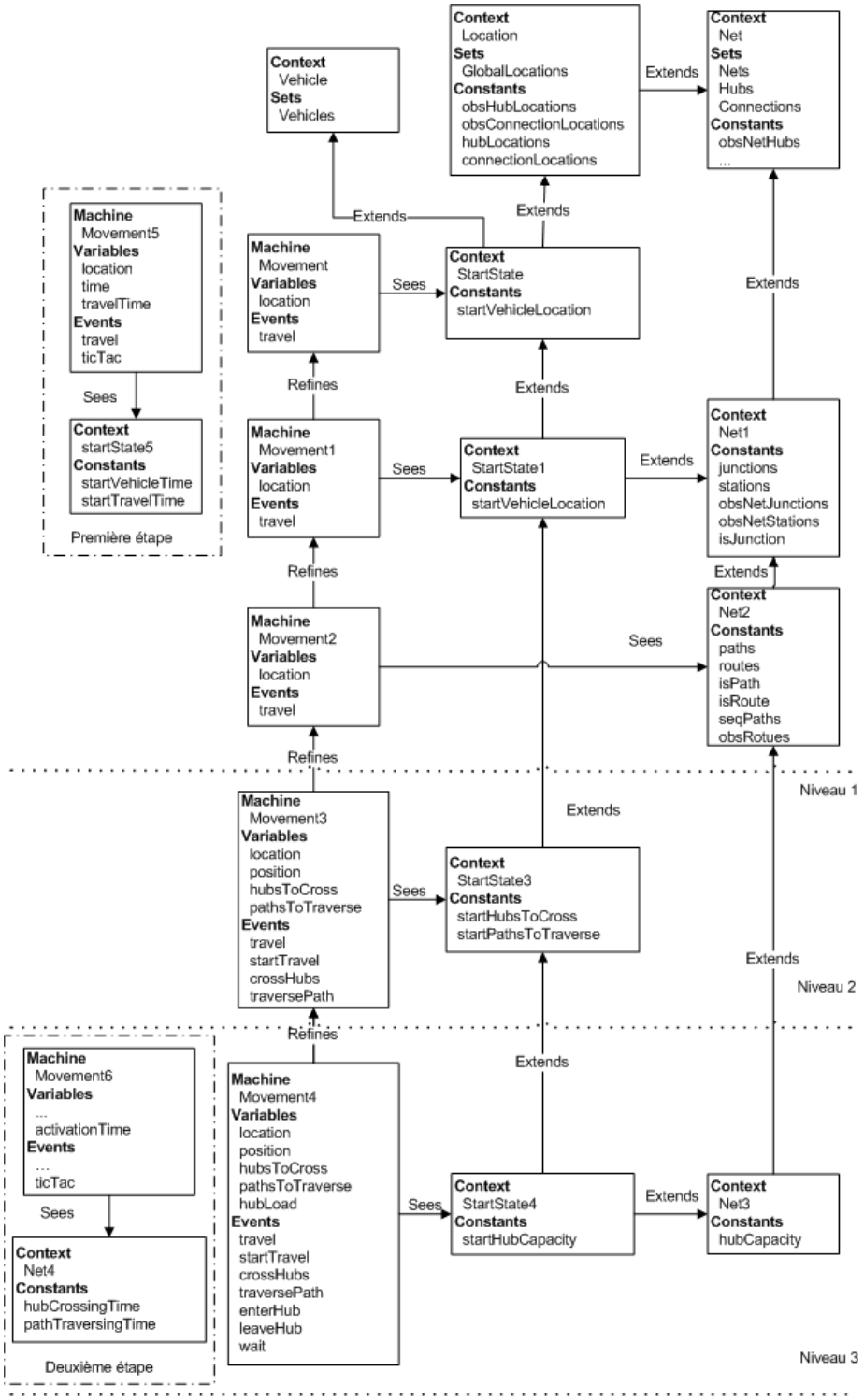


Fig. 10. Hiérarchie du modèle en B événementiel

vers une analyse fine des propriétés et l'arrêt de l'affinage obéit à des critères différents.

B événementiel et sa plateforme support Rodin n'ont pas été conçus pour la spécification de domaines. Il était donc prévisible que nous rencontrions quelques difficultés qui mettent en évidence des points intéressants. Bien que notre travail n'en soit qu'à ses débuts, nous avons collecté un ensemble d'observations intéressantes sur les outils, le langage et le processus de formalisation.

7.1 Les outils

Un outillage puissant est indispensable pour exploiter le potentiel des méthodes formelles. Son rôle est moins d'aider à réaliser des tâches fastidieuses que de révéler les erreurs : l'échec d'une preuve signe une erreur de spécification. De ce fait, nous utilisons Rodin et nous nous reposons beaucoup sur lui.

La version courante de Rodin souffre de quelques problèmes mineurs qui irritent mais ne mettent pas en cause l'utilité de l'outil. Par exemple, trop de preuves « évidentes » nécessitent une intervention manuelle. Les deux ou trois clicks nécessaires pour terminer la preuve consomment du temps et de la concentration au détriment des preuves plus complexes. De plus, un *crash* (heureusement peu fréquent) ou l'essai de variantes oblige à recommencer l'exercice.

La fonction de Rodin qui nous a le plus manqué concerne la `détection` des axiomes logiquement inconsistants. L'utilisateur n'est pas informé d'une telle situation, sinon par le fait que les preuves deviennent soudain étrangement faciles à conclure. Il est possible que la détection automatique de la preuve de $\bot$ soit difficile, mais cette fonction nous paraît indispensable pour la modélisation de domaines. Notre implantation temporaire de cette fonction consiste à inclure le théorème `TRUE = FALSE` dans les contextes. Prouver ce théorème prouve sans ambiguïté que la spécification est inconsistante[4]. L'incohérence logique est une question cruciale pour la modélisation de domaines. Les contextes modélisent les structures et les propriétés élémentaires du monde réel. De fait, ils contiennent un ensemble de formules logiques riches et étroitement imbriquées. La spécification présentée figure 3 montre que l'idée simple qu'un réseau de transport est un graphe orienté munis de quelques propriétés naïves se traduit pas plus de vingt axiomes en B événementiel[5]. La probabiblité d'introduire une incohérence subtile est élevée.

Les générateurs d'obligations de preuves et les démonstrateurs inclus dans Rodin permettent de vérifier la spécification, ils ne permettent pas de la valider. Or, dans notre cas, la validation est indispensable. Nous avons commencé à travailler avec Brama, un animateur de spécifications développé par Clearsy, mais nous n'avons pas encore de résultats probants. Comme nous le verrons un peu plus loin, les limitations actuelles de cet outil imposent de mettre en place des techniques de contournement qu'il nous reste à découvrir.

7.2 Le langage

En ce qui concerne le langage, la représentation des axiomes dans une longue liste unique n'est pas pratique. Justifiée pour des raisons mathématiques, cette

[4] Il convient de retirer ce théorème du contexte par la suite, sinon, même s'il n'est pas prouvé, il introduit de l'incohérence dans les spécifications qui utilisent le contexte.

[5] Dans les spécifications présentées dans cet article, nous n'avons pas donné les axiomes relatifs aux propriétés des graphes orientés.

présentation n'est pas la plus facile à utiliser. Les axiomes peuvent être classés en trois catégories liées à leur fonction dans le modèle : (1) axiomes techniques nécessaires à la preuve (fini, non-vide, etc.), (2) expression du typage et (3) propriétés spécifiques au domaine. Le spécifieur travaille surtout sur cette dernière catégorie, souvent dans un esprit de « mise au point » de la formule modélisant la propriété. Isoler visuellement chaque catégorie d'axiomes apporterait une aide non négligeable.

L'absence de la structure de liste, et, plus généralement, des structures algorithmiques usuelles est pénalisante. Il faut redéfinir les axiomes à chaque fois que nous voulons utiliser une telle structure, donc augmenter encore la longueur de la liste des axiomes. De façon plus gênante, l'absence des listes bloque l'utilisation de Brama. En effet, l'outil est incapable de générer un itérateur à partir de la définition mathématique des listes. Il faut alors « tricher » avec la spécification, ce qui introduit un risque d'erreur non négligeable.

7.3 Le processus de modélisation

Dans le processus de modélisation, nous avons utilisé deux techniques différentes pour faire évoluer la spécification : le raffinement défini dans le langage et le passage à un « niveau d'observation » différent. Cette dernière technique peut être vue comme la décomposition d'un événement macroscopique en une séquence d'événements microscopiques. Par exemple, `travel` est décomposé en une suite de `crossHub` et de `traversePath`. Techniquement, il n'existe pas de lien de raffinement entre ces événements. Conceptuellement, le changement de niveau d'observation nous permet d'introduire une structure hiérarchique dans le modèle. Nous décrivons quels petits évéments réalisent un gros événement. Cette vision hiérarchique soulève plusieurs questions :

- Une notion manipulée à deux niveaux d'observation doit être formellement `redéfinie`. Par exemple, la position d'un véhicule dans le réseau est la même notion à tous les niveaux, pourtant, il faut définir autant de variables de profil identique (location et position par exemple) et ayant la même sémantique intuitive, qu'il y a de niveaux. Bien que les obligations de preuves mettent assez vite en évidence un éventuel oubli de redéfinition, il serait agréable qu'un outil d'aide à l'édition nous rappelle cette contrainte technique.
- Comme décrit lors de l'introduction de la notion de temps dans le modèle, la structuration en niveaux d'observation est un guide intéressant. En particulier, elle nous a permis d'identifier rapidement quels événements étaient concernés soit par la définition de la nouvelle notion, soit par sa modélisation précise.
- Vérifier que la `décomposition est correcte` est une tâche difficile. Le problème vient de ce que les liens entre les événements situés à différents niveaux n'ont pas d'expression dans le langage. Il n'y a donc pas d'obligations de preuves générées automatiquement, et donc pas de preuves :
  - Les notions de variant et d'événement convergents permettent de s'assurer que la décomposition ne « boucle » pas, mais elle ne permet pas de vérifier que les micro-événements seront tous déclenchés, et dans le bon ordre !
  - Elles ne permettent pas non plus de prouver que le dernier micro-événement, qui doit déclencher le macro-événement, sera bien activé. Il faut

de plus noter que le domaine étudié impose deux événements qui ne peuvent pas être convergents par nature, startTravel et wait. Le premier correspond à l'initialisation d'un voyage, nécessaire pour des raisons techniques. Le second, wait, est plus intéressant car lié directement aux propriétés du domaine.
- Le temps de transport est une notion essentielle liée à l'interaction entre les différents véhicules. Notre modèle prévoit qu'un véhicule doit attendre pour entrer dans un hub qu'il y ait de la place ; par définition de l'attente, l'événement n'a pas d'action (on ne fait rien quand on attend) et ne peut donc pas faire décroître un variant. Le fait que le système évolue tout de même parce que l'attente est causée par un autre événement qui a lieu simultatnément (un autre véhicle est en train de traverser le hub) n'est pas exprimable. Nous aurions alors de grandes difficultés pour exprimer et prouver une propriété telle que le temps d'attente est borné.

La dernière phrase du paragraphe précédent soulève une interrogation plus générale : est-il raisonable d'exprimer toutes les propriétés d'un domaine comme un théorème ? Vouloir exprimer et prouver que le temps d'attente est fini peut introduire subrepticement une contrainte plus forte que celle que nous imaginons. En l'occurrence, cette propriété naïve implique l'existence d'un contrôleur ayant une vue globale et complète du domaine. Certains systèmes de transport posent ce contrôleur comme préalable, le ferroviaire par exemple. Dans d'autres systèmes, l'aérien par exemple, un contrôle centralisé est une nécessité logique impliquée par le temps d'attente en vol très court dont dispose un avion. Dans le cas d'un système routier, un contrôleur global n'a pas de sens ; de fait, une caractéristique fondamentale (non modélisée ici) est que chaque véhicule agit de façon autonome en fonction de sa perception de l'état de l'environnement local. Le point important à souligner est qu'il faut être vigilant quant à la nature des propriétés à exprimer :
- Pour certaines, les techniques formelles implantées dans B événementiel sont parfaites et l'absence de collision relève de cette catégorie.
- Pour d'autres, des techniques probabilistes ou des processus d'optimisation peuvent être plus appropriées.

8 Perspectives

En choisissant les transports comme étude de cas pour la modélisation de domaines en B événementiel, nous avons débuté un travail de longue haleine et l'état actuel du modèle est un point de départ pour plusieurs études.

La première concerne la `validation du modèle`. Pour cela, nous devons mettre en œuvre des techniques d'animation compatibles avec les outils existants. Ceci nous permettra de contrôler la conformité de notre modèle sur deux plans :
- Sur le plan technique, l'animation permettra de vérifier que la décomposition des événements conduit bien au comportement souhaité. Si les relectures attentives donnent une bonne confiance dans notre modèle, nous ne sommes pas à l'abri d'une surprise dans des cas particuliers vicieux.
- Sur un plan général, nous devons nous assurer que le comportement global du système est un modèle raisonable des systèmes réels.

La deuxième concerne `l'utilité du modèle`. Là encore, il est bon de distinguer le point de vue général du point de vue technique :

– Sur le plan général, l'élaboration du modèle est un excellent exercice de compréhension du domaine étudié.
– Sur le plan technique, il permet de donner une définition à des propriétés non-fonctionnelles souvent implicites.

Dans les deux cas, cette compréhension est indispensable à la conception et à l'homologation de nouveaux systèmes de transport basés sur des véhicules autonomes ou des véhicules circulant en convoi (*platooning*). Dans le cadre du projet ANR TACOS, nous étudions comment un modèle de buts développé en KAOS [24] peut se formaliser en utisant des éléments de notre modèle [25]. Dans le projet CRISTAL, nous devrons étudier comment des réalisations concrètes peuvent être confrontées à notre modèle.

La troisième concerne l'aspect `méthodologique`. L'idée d'introduire des niveaux d'observation en tant que guide doit être validée avec la prise en compte de nouvelles propriétés. Il sera intéressant de voir si l'introduction de notions telles que la consommation d'énergie ou la disponibilité des véhicules par exemple, relève d'une démarche similaire à l'introduction du temps. Cette étude pourra conduire à la proposition de patrons de développement pour guider la prise en compte progressive de nouvelles propriétés.

Enfin, il va de soi que l'affinement des comportements, en particulier sur les `paths`, que nous avons ignorés ici, fait partie des suites immédiates à ce travail.

Références

[1] Curtis, B., Krasner, H., Iscoe, N. : A field study of the software design process for large systems. Communications of the ACM **31**(11) (1988) 1268–1287

[2] Neighbors, J.M. : Software construction using components. PhD thesis (1980)

[3] Debaud, J.M., Rugaber, S. : A software re-engineering method using domain models. In : 11th IEEE International Conference on Software Maintenance (ICSM). (1995)

[4] Arango, G. : A brief introduction to domain analysis. In : Proceedings of the ACM symposium on Applied computing (SAC). (1994) 42–46

[5] Pressman, R.S. : Software Engineering : A Practitioner's Approach. McGraw-Hill Inc (2005)

[6] Jackson, M. : Problems and requirements (software development). In : Second IEEE International Symposium on Requirement Engineering, IEEE Computer Society (1995) 2–9

[7] Jackson, M. : Problem Frames : Analyzing and Structuring Software Development Problems. Addison-Wesley (2001)

[8] Bjørner, D. : Software Engineering 3 : Domains, Requirements, and Software Design (Texts in Theoretical Computer Science, an EATCS Series). Springer-Verlag New York, Inc. (2006)

[9] Bjørner, D. : Development of transportation systems. In : International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISOLA). (2007)

[10] Hitchcock, A. : A Specification Of An Automated Freeway With Vehicle-Borne Intelligence. Research Report UCB-ITS-PRR-92-18, California Partners for Advanced Transit and Highways (PATH) (1992)

[11] Hitchcock, A. : Methods Of Analysis Of IVHS Safety. Research Report UCB-ITS-PRR-92-14, California Partners for Advanced Transit and Highways (PATH) (1992)

[12] Tsao, H.S.J., Hall, R.W., Shladover, S.E. : Design Options for Operating Automated Highway Systems. In : Proc. IEEE-IEE Vehicle Navigation & Information System Conference. (1993) 494–500

[13] Lanoix, A. : Event-B specification of a situated multi-agent system : Study of a platoon of vehicles. In : 2nd IFIP/IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE), IEEE Computer Society (2008) 297–304

[14] Abrial, J.R. : The B Book. Cambridge University Press (1996)

[15] Behm, P., Benoit, P., Meynadier, J.M. : METEOR : A successful application of B in a large project. In : Integrated Formal Methods. Volume 1708 of LNCS., Springer Verlag (1999) 369–387

[16] Badeau, F., Amelot, A. : Using B as a high level programming language in an industrial project : Roissy VAL. In : Formal Specification and Development in Z and B. Volume 3455 of LNCS., Springer-Verlag (2005) 334–354

[17] Steria : Obligations de preuve : Manuel de référence, version 3.0. Steria – Technologies de l'information. (1998)

[18] Clearsy : B4free. website, `http://www.clearsy.com/` (2008)

[19] Abrial, J.R., Hallerstede, S. : Refinement, Decomposition, and Instantiation of Discrete Models : Application to Event-B. Fundamenta Informaticae **77**(1-2) (2007) 1–28

[20] Dijkstra, E.W. : A Discipline of Programming. Prentice Hall (1976)

[21] Baille, G., Garnier, P., Mathieu, H., Roger, P.G. : Le cycab de l'INRIA rhônes-alpes. Technical Report RT-0229, INRIA – Rhônes-Alpes (1999)

[22] Rehm, J., Cansell, D. : Proved Development of the Real-Time Properties of the IEEE 1394 Root Contention Protocol with the Event B Method. In : ISoLA. (2007) 179–190

[23] Cansell, D., Mery, D., Rehm, J. : Time Constraint Patterns for Event B Development. In Julliand, J., Kouchnarenko, O., eds. : 7th International Conference of B Users. Volume 4355 of LNCS., Springer Verlag (2007) 140–154

[24] Lamsweerde, A.v. : From System Goals to Software Architecture. In Bernardo, M., Inverardi, P., S.V., eds. : Formal Methods for Software Architectures. Volume 2804 of LNCS. (2003)

[25] Matoussi, A., Gervais, F., Laleau, R. : A First Attempt to Express KAOS Refinement Patterns with Event B. In : Proc. of the Int. Conf. on ASM, B and Z (ABZ). Lecture Notes in Computer Science, Springer-Verlag (2008) 12–14