De la modélisation formelle à la simulation à évènements discrets : application à la conception et à l'évaluation de protocoles sûrs et sécurisés pour les communications dans les transports

Emna Chebbi

▶ **To cite this version:**

Emna Chebbi. De la modélisation formelle à la simulation à évènements discrets : application à la conception et à l'évaluation de protocoles sûrs et sécurisés pour les communications dans les transports. Cryptographie et sécurité [cs.CR]. Université du Littoral Côte d'Opale, 2019. Français. NNT : 2019DUNK0538 . tel-02481559

HAL Id: tel-02481559
https://theses.hal.science/tel-02481559v1

Submitted on 17 Feb 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

École doctorale nº 072 : Sciences pour l'Ingénieur

Doctorat Lille Nord-de-France

THÈSE

pour obtenir le grade de docteur délivré par

l' Université du Littoral Côte d'Opale

Spécialité doctorale "Informatique"

présentée et soutenue publiquement par

Emna CHEBBI

le 18 octobre 2019

De la modélisation formelle à la simulation à évènements discrets : application à la conception et à l'évaluation de protocoles sûrs et sécurisés pour les communications dans les transports

Jury

M. Gregory ZACHAREWICZ,	Professeur, IMT d'Alès	Rapporteur
M. Mohamed GRAIET,	Professeur, ISIMM, Tunisie	Rapporteur
Mme Marion BERBINEAU,	Directrice de Recherche, IFSTTAR	Examinateur
Mme Martine WAHL,	Chargée de Recherche, IFSTTAR	Examinateur
M. Anis LAOUITI,	Maître de Conférences HDR, IMT Telecom-SudParis	Examinateur
M. Hassine MOUNGLA,	Maître de Conférences HDR, Paris 5 Descartes	Examinateur
Dr. Ouafae COHIN,	Chef de Projet Maturation, SATT Nord	Invitée
M. Eric RAMAT,	Professeur, ULCO	Diecteur de thèse
M. Patrick SONDI,	Maître de Conférences, ULCO	Encadrant

Université du Littoral Côte d'Opale
Maison de la Recherche Blaise Pascal, 50 Rue Ferdinand Buisson, 62228 Calais
Téléphone (+33) 3 21 46 36 00

Table des matières

Liste des figures

Liste des tableaux

Remerciements

Je ne peux pas commencer mes remerciements sans remercier Dieu.

Ce travail est dédié à l'âme de ma mère Fatma, qui m'a toujours motivée, encouragée et assurée que je continue mes études jusqu'au bout, ce qui m'a conduite à cette thèse de doctorat. La femme qui a sacrifié toute sa vie pour moi, pour être en même temps ma mère et mon père. J'espère que, du monde qui est sien maintenant, elle est fière de sa petite fille qui a toujours prié pour le salut de son âme. Puisse Dieu, le tout puissant, l'avoir en sa sainte miséricorde.

À mon encadrant, Patrick SONDI qui depuis mon stage à l'ULCO a cru à mes compétences et ma volonté envers la recherche scientifique. Je le remercie pour la confiance qu'il m'a accordée et pour son encadrement sérieux, qui m'ont beaucoup aidée et m'ont beaucoup appris dans mon domaine de recherche et en terme de culture générale. Le travail avec lui était très intéressant et j'ai profité de chaque conversation avec lui, même pendant nos pauses puisqu'il y a toujours quelque chose à apprendre de ce qu'il dit et de sa manière d'encadrement qui va me servir dans mon futur professionnel. Il a été quelqu'un de très humain et compréhensif durant mes mauvais moments à savoir le décès de ma mère et mes maladies. C'est une personne qui a laissé une trace positive dans mon parcours universitaire et je lui en suis très reconnaissante.

Je tiens tout particulièrement à remercier mon directeur Eric RAMAT qui m'a donné la chance d'effectuer cette thèse dans d'excellentes conditions. Sa disponibilité et ses conseils ont été indispensables à la concrétisation de cette thèse. Il m'a prodigué durant mon apprentissage de la simulation DEVS que j'ai découverte avec lui.

J'adresse mes plus respectueux remerciements à M. Gregory ZACHAREWICZ et M. Mohamed GRAIET d'avoir bien voulu accepter d'être les rapporteurs de mon mémoire et pour l'intérêt qu'ils ont porté à ce travail. Je tiens également à remercier tous les membres de mon jury pour m'avoir fait l'honneur d'assister à la soutenance publique de ma thèse de doctorat.

Une pensée affectueuse à mon mari Heni, qui a tout fait pour moi afin de me laisser libre pour mes études et mes travaux de thèse. Il était toujours à mes côtés, m'a beaucoup supportée et a pris la relève après le départ de ma maman.

A mon futur bébé Ahmatto, qui m'a accompagnée pendant les longues soirées de rédaction avec ses premiers mouvements.

À mes sœurs Mariem, Wela et Rim qui chacune d'entre elles ont eu une part très importante dans mon parcours universitaire pour leur soutien sans faille. Je remercie particulièrement Mariem pour son support et son amour sans limite.

J'adresse mes remerciements par une pensée exceptionnelle à la petite famille BELLAAJ, mes collègues et mes ami(e)s proches, qui se reconnaîtront sans que je les cite.

Je remercie la Région Hauts-de-France et l'ULCO pour le financement de ma thèse de doctorat.

Résumé

La conception de protocoles de communication repose généralement sur des modèles fonctionnels élaborés à partir des besoins du système. Dans les systèmes de transport intelligents (ITS), les fonctionnalités étudiées incluent l'auto- organisation, le routage, la fiabilité, la qualité de service et la sécurité. Les évaluations par simulation sur les protocoles dédiés aux ITS se focalisent sur les performances dans des scénarios spécifiques. Or, l'évolution des transports vers les véhicules autonomes nécessite des protocoles robustes offrant des garanties sur certaines de leurs propriétés. Les approches formelles permettent de fournir la preuve automatique de certaines propriétés, mais pour d'autres il est nécessaire de recourir à une preuve interactive impliquant le savoir d'un Expert. Les travaux menés dans cette thèse poursuivent l'objectif d'élaborer, dans le formalisme DEVS (Discrete Event System Specification), des modèles d'un ITS dont la simulation permettrait d'observer les propriétés, éventuellement vérifiées par une approche formelle, dans un scénario plus large et de générer sur les modèles des données susceptibles d'alimenter une boucle de preuve interactive au lieu d'un Expert. Prenant pour cible le protocole CBL-OLSR (Chain-Branch-Leaf in Optimized Link State Routing), cette thèse montre comment un modèle DEVS et un modèle formel Event-B équivalents peuvent être construits à partir de la même spécification fonctionnelle d'un réseau ad hoc où les noeuds utilisent ce protocole. Des propriétés relatives à la sûreté et à la sécurité sont introduites dans le modèle formel Event-B afin d'être vérifiées, puis une méthodologie est proposée afin de les transférer dans un modèle DEVS équivalent sous forme de contraintes, de choix ou d'observables selon des critères proposés. Enfin, cette thèse ouvre également les perspectives de l'automatisation de ce processus de conception, de l'intégration à la simulation DEVS de données réelles à la fois sur le trafic routier et sur les flux d'applications dédiées aux véhicules, et de l'interaction avec des simulateurs spécialisés pour les différents composants (par exemple MATLAB pour les modèles de propagation, OPNET ou NS3 pour les communications, SUMO pour les modèles de mobilité) ; le but étant une évaluation du protocole dans un contexte très réaliste du système.

Mots-clés

Ingénierie des protocoles, Simulation à événements discrets, Modélisation formelle, Communications sécurisées, Systèmes de transports intelligents, DEVS, Event-B, Réseaux ad hoc véhiculaires.

Astract

The design of communication protocols is generally based on functional models developed from the system needs. In Intelligent Transport Systems (ITS), the studied functionalities include self-organization, routing, reliability, quality of service and security. Simulation evaluations of ITS protocols mainly focus on performance in specific scenarios. However, the evolution of transportation towards autonomous vehicles requires robust protocols offering guarantees on some of their properties. Formal approaches make it possible to provide automatic proof of certain properties, but for others it is necessary to use interactive proof involving the knowledge of an Expert. The work carried out in this thesis aims to develop, in the DEVS formalism (Discrete Event System Specification), models of an ITS whose simulation would make it possible to observe the properties, possibly verified by a formal approach, in a broader scenario and to generate data on the models that could feed an interactive proof loop instead of an Expert. Targeting the CBL-OLSR (Chain-Branch-Leaf in Optimized Link State Routing) protocol, this thesis shows how a DEVS model and an equivalent formal Event-B model can be built from the same functional specification of an ad hoc network where nodes use this protocol. Safety and security properties are introduced into the formal Event-B model to be verified, and a methodology is proposed to transfer them to an equivalent DEVS model in the form of constraints, choices or observables according to pre-proposed criteria. Finally, this thesis also opens up the prospects for automating this design process, integrating real data on both road traffic and vehicle application flows into DEVS simulation, and interacting with specialized simulators for the various components (e. g. MATLAB for propagation models, OPNET or NS3 for communications, SUMO for mobility models) ; the aim being to evaluate the protocol in a very realistic system context.

Keywords

Protocol Engineering, Discrete Event Simulation, Formal Modelling, Network security, Intelligent Transportation Systems, DEVS, Event-B, Vehicular ad hoc networks.

Introduction générale

Contexte de la thèse

Aujourd'hui le véhicule connecté est devenu en réalité. En France, le ministère des transports a donné son feu vert pour une vaste campagne d'expérimentations de voitures autonomes. Elle devrait durer jusqu'en 2022 pour permettre de tester seize véhicules sans conducteur. Trois Français sur quatre s'y intéressent. Les communications se présentent d'emblée, au même titre que la localisation et la perception, comme une composante essentielle pour le développement de cette nouvelle forme de mobilité.

Dans la conception générale des systèmes de communication pour les transports intelligents (ITS), il existe deux tendances. Une catégorie qui consiste à déployer une infrastructure pour supporter les communications entre les véhicules et une autre qui favorise des communications qui se passeraient directement de véhicule à véhicule. Pour ce qui concerne l'infrastructure, l'organisation est assez simple. Il s'agit principalement de résoudre un problème de dimensionnement pour que l'infrastructure déployée permette de couvrir correctement l'ensemble des sections du trafic routier ciblé. En revanche, le choix d'un mode de communication ad hoc, de véhicule à véhicule (V2V) comme illustré dans la figure 1, pose un problème d'auto-organisation distribuée où des véhicules indépendant les uns des autres doivent coopérer pour que les communications soient performantes.

Le protocole de routage ad hoc joue un rôle déterminant dans la mise en place de cette auto-organisation. En effet, son fonctionnement implique que les véhicules échangent des informations et observent des règles de communication permettant à terme de réduire la part de consommation de ressources de communications dédiée à l'organisation du réseau et offrir la meilleure performance aux applications, comme si on avait une infrastructure qui opère.

Motivation et but de la thèse

La conception de protocole de communication est généralement basée sur les modèles fonctionnels développés en fonction des besoins du système. Dans le domaine des ITS, les caractéristiques étudiées concernant les réseaux ad hoc de véhicules (VANET) incluent l'auto-organisation,

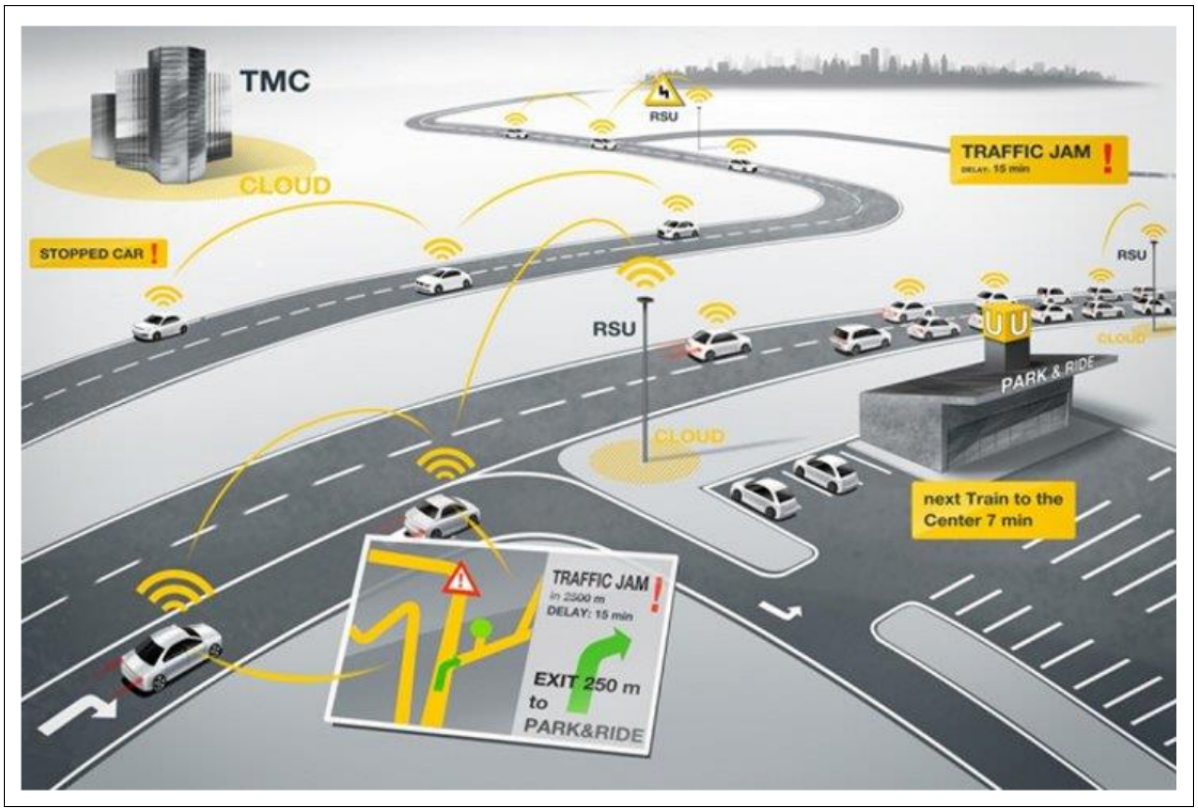

FIGURE 1 – Réseau Ad hoc, auto-organisation [Fro18]

le routage, la fiabilité, la qualité de service et la sécurité. Les études par simulation de ces protocoles de routage se focalisent habituellement sur leur performance pour les applications visées dans des scénarios spécifiques. Cependant, l'évolution des systèmes de transport vers des véhicules autonomes nécessite des protocoles robustes avec des propriétés prouvées ou tout au moins garanties. Les approches formelles fournissent des outils puissants offrant de mécanismes de preuves de propriétés pour la conception de système et de protocoles. Néanmoins, deux problèmes persistent en matière d'évaluation :

— Le premier concerne l'évaluation conjointe des composants modélisés formellement avec les autres composants de l'ITS qui ne se prêtent pas à la modélisation formelle. Des outils formels tels qu'Event-B permettent d'évaluer uniquement les composants modélisés formellement, tandis que les outils de simulation offrent la possibilité de connecter des modèles hétérogènes voire du matériel dans un seul processus d'évaluation ;

— Le second concerne l'exhaustivité et l'évolutivité : les outils formels permettent d'animer un nombre limité d'objets lors de la vérification du comportement d'un système en présence d'interactions. Cependant, il est difficile de les appliquer à l'évaluation d'un grand nombre d'objets connectés avec une forte dynamicité temporelle, ce qui est le cas des ITS. La simulation est particulièrement adaptée à ce type d'évaluations ;

Le but de cette thèse est de développer une approche combinant des outils formels qu'Event-

B et des outils basés sur le formalisme DEVS (Discrete Event System Specification) tel que VLE dans la conception et l'évaluation de protocoles de communication. Ainsi les vérifications et les preuves apportées par la modélisation formelle seront complétées par la capacité de la simulation à connecter des composants hétérogènes et tester à grande échelle, en terme de nombre de composants et de taille de déploiements, une variété de scénarios dynamiques du système.

Organisation du mémoire

Afin de présenter les travaux menés durant cette thèse, ce mémoire, tel qu'il est présenté, s'organise de la façon suivante :

— Un état de l'art succinct est proposé dans le premier chapitre sur les approches de conception et d'évaluation de protocoles de communication pour les réseaux dédiés aux transports, en particulier sur les réseaux ad hoc.

— Le second chapitre présente les modèles de différentes composantes d'un réseau ad hoc de véhicules (VANET) réalisé avec l'environnement de laboratoire virtuel (VLE). Il s'agit de montrer comment la simulation à événements discrets DEVS peut être utilisée pour modéliser et évaluer un protocole de communication proposé pour les VANET.

— Le troisième chapitre présente une modélisation formelle de ce même protocole à partir de sa spécification fonctionnelle, permettant ainsi de vérifier et de prouver un certain nombre de ses propriétés (notamment de sûreté et de sécurité) en recourant à l'outil formel Event-B basé sur la méthode B.

— Le quatrième chapitre présente notre tentative de construire une approche combinant les deux paradigmes, simulation à événement discrets DEVS et de modélisation formelle Event-B, pour la conception et l'évaluation du protocole de communication. Une illustration des concepts développés est faite sur des parties du modèle formel Event-B et du modèle DEVS présentés dans les chapitres 2 et 3.

Chapitre 1

Etat de l'art

Sommaire

1.1 Introduction

Historiquement, les protocoles de communication ont été développés de manière hétérogène en fonction des technologies de télécommunication utilisées et du domaine d'application ciblé. Alors que les protocoles de l'Internet étaient développés essentiellement pour utiliser au mieux les capacités du réseau et supporter des applications peu exigentes, des technologies et des protocoles spécifiques plus robustes étaient développés pour des domaines plus exigents tels que l'Industrie, les transports, la défense, etc. Les différentes approches développées pour l'ingénierie des protocoles ont suivi la même tendance. Sur le plan de la conception, les fonctionnalités du protocole de communication sont déterminées par les besoins fonctionnels du système dans lequel il sera déployé. Par exemple, dans un réseau ferroviaire opéré où l'organisation des communications est gérée via l'infrastructure dédiée, le protocole de communication n'intègre pas de mécanisme d'auto-organisation du réseau, alors que dans un réseau ad hoc de véhicules c'est l'une de ses fonctionnalités essentielles. Sur le plan de l'évaluation, les protocoles de communication pour des applications non critiques sont surtout étudiés sous l'angle de la performance, alors que ceux dédiés à des applications exigentes nécessitent des preuves ou des garanties sur certaines de leurs fonctionnalités.

Sans viser l'exhaustivité, l'objectif du succinct état de l'art présenté ici est surtout de montrer différentes approches et outils utilisés dans la conception et l'évaluation de protocoles de communication pour mettre en évidence le besoin qui a conduit à la proposition de cette thèse. Les références et exemples évoqués sont surtout choisis dans le domaine des transports terrestres (ferroviaire et routier), et en majorité dans les réseaux ad hoc de véhicules.

1.2 Conception et évaluation de protocoles par la simulation

La simulation est un outil intéressant pour l'ingénierie de protocoles. En phase de conception, elle permet de vérifier des hypothèses, optimiser les paramètres de fonctionnement et modifier des choix éventuels. En phase d'évaluation, elle permet d'observer le comportement du système dans tous les scénarios d'intérêt et analyser les performances. Si historiquement la simulation a parfois été critiquée pour les biais qu'elle pourrait introduire dans l'analyse des systèmes [Rob99], la puissance croissante des moyens informatiques permet d'en pallier la plupart. La prolifération d'outils basés sur la simulation pour la conception et d'évaluation des réseaux de communications en général [RPZW09] et des réseaux dédiés aux ITS en particulier [BHP17] témoigne du regain d'intérêt pour la simulation dans le domaine des réseaux.

Partons justement des principales sources de biais dans la simulation, à savoir la modélisation, les données et l'expérimentation, pour faire un point sur son apport actuellement dans la

conception et l'évaluation des protocoles de réseaux.

1.2.1 Modélisation et simulation des protocoles

Les principales erreurs liées à la modélisation proviennent soit de la faiblesse de concepts de modélisation, soit d'un défaut d'expertise du modélisateur, soit des erreurs d'implémentation. Plusieurs études récensent les différents approches de modélisation qui peuvent être utilisées pour la simulation des réseaux de communication, notamment pour évaluer les protocoles qui y sont proposés. Une première synthèse proposée par [Son04] recense plusieurs formalismes de modélisation de systèmes complexes et présente une illustration sur la simulation d'un réseau sans-fil. Ces formalismes offrent un cadre théorique permettant de garantir la correction et la cohérence des mécanismes propres à la simulation, indépendamment du contenu des modèles, permettant ainsi au modélisateur de se concentrer sur sa modélisation. Des exemples concrets de l'utilisation de ces formalismes dans le développement d'outils spécifiques pour la simulation des réseaux dédiés aux transports sont également présentés par [SCB11] et [BHP17].

Il ressort de cette revue de la littérature que de nombreux outils de modélisation libres ou commerciaux ont aujourd'hui atteint un niveau de maturité suffisamment avancé pour permettre une bonne modélisation de protocoles de communication. Les erreurs de modélisation dues à un défaut d'expertise sont quasiment exclues étant donné que le modélisateur est généralement le concepteur du protocole et qu'il n'a que son protocole à modéliser. En effet, la plupart d'outils de simulation offrent des bibliothèques assez fournies de modèles pour les autres composants du système qui échapperaient éventuellement à son expertise, par exemple un outil commercial comme l'OPNET Modeler de [Riv]. Les outils qui n'en disposent pas peuvent être complétés grâce aux bibliothèques d'outils libres équivalents à OPNET tels que NS2/NS3 ([ISI]), OMNET++ ([Opeb]), etc. Par la même occasion, en offrant au modélisateur une interface de modélisation de haut niveau pour la description des modèles et en prenant en charge l'implémentation informatique des modèles, les outils de simulation actuels réduisent les erreurs dues à l'implémentation.

Pour conclure sur ces aspects, il convient de noter que la plupart des simulateurs, y compris les plus complets, sont souvent très performants pour la modélisation de certains composants et un peu moins efficaces pour d'autres. Par exemple, si l'on souligne l'efficacité de MATLAB ([Mat]) sur la modélisation des mécanismes physiques telle que la propagation, il est admis qu'il est plus simple de simuler le traffic applicatif dans un réseau de véhicules à grande échelle avec OPNET ou la mobilité urbaine de véhicules avec SUMO ([noa16]). Dans ce contexte, des outils basés sur le couplage d'autres outils de simulations grâce à des mécanismes tel qu'HLA (High Level Architecture) [SZ00] [LLC03] [Zac06] permettent encore d'améliorer l'analyse des systèmes de transport par la simulation. L'outil VSimRTI présenté notamment dans [PSR17] réalise un couplage entre

SUMO pour la simulation de la mobilité et OMNET++/NS3 pour la simulation des communications afin d'offrir un cadre de simulation plus réaliste pour les réseaux véhiculaires. Une autre approche basée sur la cosimulation est proposée par [PS17] entre OPNET pour les communications et un simulateur de trafic ferroviaire pour pallier le fait que ce dernier ne supporte pas une intégration via HLA.

1.2.2 Données du domaine et simulation des protocoles

Les données du domaine constituent la seconde source de biais dans l'analyse par simulation des systèmes. Le biais peut provenir du fait que soit ces données n'existent pas, notamment parce que le système n'existe pas encore dans le cas d'une étude prospective, et que les données de substitution ne sont pas fiables, soit parce que les données utilisées sont erronées, soit parce que les données issues de la simulation sont mal interprétées.

Dans le domaine des transports, la disponibilité des données a longtemps posé problème pour une partie des composants du système, davantage parce qu'elles étaient difficement accessibles plutôt que du fait de leur inexistence. Depuis plusieurs annnées, cette tendance s'est inversée par la multiplication d'initiatives d'ouverture de données par les services publics, les industriels, les opérateurs et les chercheurs du domaine. Ces données comprennent pêle-mêle :

— des modèles d'équipements de réseaux mis à disposition des plateformes de simulation par des équipementiers (par exemple par 3COM dans OPNET) ou des éditeurs de logiciels (par exemple OpenRBC par [Mor]) ou des organismes de recherche (par exemple OpenAirInterface par [Eur]), etc ;

— des outils open source pour le test et la validation des applications de gestion de trafic (par exemple par un consortium européen d'acteurs du ferroviaire dans le cadre d'[Opea]) ;

— des données de couverture réseau et autres données administratives par les organismes publics (par exemple l'[ARC] dans le cadre de l'Open data)

— des données de mesures terrain des puissances reçues des équipements de télécommunication, des traces anonymisées de trajectoires de véhicules ou de communications entre véhicules et éléments d'infrastructure (par exemple par le site [Cra]). [SBKM13] proposent évaluation par simulation basée sur des traces réelles de trajectoires et de communications provenant d'équipementiers ou opérateurs du ferroviaire.

Toutes ces initiatives sont encourageantes pour l'avenir de la simulation comme outil de conception et d'évaluation des systèmes de transport intelligent ainsi que tous leurs composants, les protocoles de communication notamment.

1.2.3 Expérimentation et simulation des protocoles

Les biais liées à l'expérimentation sont souvent dus à un mauvais paramétrage de la simulation ou à une mauvaise analyse des résultats, notamment par un manque d'analyse de la sensibilité ou un mauvais choix des observables et des statistiques à collecter. C'est une vraie expertise et la plupart des environnements de modélisation et de simulation proposent aux modélisateurs un ensemble de choix prédéfinis par des experts. Dansle cadre des initiatives mentionnées dans la section précédente sur les données, il existe de nombreuses tentatives de formaliser ces choix et de standardiser leurs définitions ainsi que leur mode de collecte et de calcul pour permettre une meilleure comparaison de résultats obtenus à partir d'outils différents.

Une autre initiative intéressante qui va dans le même sens concerne la reproductibilité des expérimentations et des résultats scientifiques. Plusieurs éditeurs scientifiques encouragent les auteurs à publier l'ensemble de leurs paramètres d'expérimentation, la description détaillée de tous les scénarios considérés et tous les résultats obtenus en sus de ceux qu'ils ont intégrés à leur publication pour permettre aux lecteurs d'aller plus loin que ce que permet le nombre limité de pages de la publication.

Pour conclure sur cette section consacrée à la conception et l'évaluation par la simulation de protocoles de communication pour les réseaux dédiés aux transports, nous nous devons de mentionner les progrès sont importants à tel point qu'on évoque davantage la notion de laboratoire virtuel que simplement de simulation. En revanche, aujourd'hui, la convergence des technologies de télécommunication provoque des évolutions à la fois par l'intégration de tout type de données (texte, image, voix, videoconference) dans une même transmission, et par une mutualisation des infrastructures pour toutes les applications de la ville intelligente. Du fait de l'implication des vies humaines dans la plupart de ces domaines d'application, les évaluations destinées à valider les choix liés à toutes ces évolutions ne peuvent pas simplement porter sur la performance. En effet, dans des domaines avec une forte culture des systèmes critiques comme le transport, l'énergie, la défense, etc, il est nécessaire de distinguer clairement ce qui est de l'ordre du prouvé ou du garanti de ce qui est de l'ordre du toléré et avec des seuils clairs.

Or, d'une part, il n'est évidemment pas envisageable de pousser la modélisation jusqu'à son niveau le plus détaillé pour tous les composants d'un système de transport et de tenter de l'évaluer pour toutes les combinaisons de valeurs de paramètres et pour tous les scénarios d'intérêt imaginables afin d'en obtenir une connaissance exhaustive. Et d'autre part, même sous l'hypothèse que cela serait faisable, pour des systèmes en cours de conception et dont il n'existe aucun équivalent déjà implémenté, il peut s'avérer difficile pour le modélisateur de disposer d'une connaissance a priori de la liste exhaustive d'observables à considérer pour son analyse. Il s'avère donc nécessaire d'associer à la simulation des méthodes qui permettent plus efficacement de déterminer sur

une modélisation raisonnable ce qui est de l'ordre du vérifié, du prouvé ou de l'incertain dans le processus de conception et d'évaluation d'un protocole de communications.

1.3 Apport des outils formels à la conception de protocoles

Les méthodes et outils formels sont utilisés depuis longtemps dans la conception et la validation processus de pilotage des systèmes de transports, comme c'est le cas pour la plupart des domaines critiques. Ils offrent un cadre de modélisation qui permet d'exprimer, de vérifier et éventuellement de prouver un certain nombre de propriétés sur certains composants d'un système de transport intelligent. Plusieurs travaux ont déjà été entrepris dans ce sens, notamment concernant les protocoles de routage pour les réseaux ad hoc de véhicules. Une revue de la littérature sur ce dernier thème qui nous intéresse plus particulièrement est proposée par [FTM19].

Certains travaux s'intéressent à l'analyse des fonctionnalités des protocoles de routage pour les réseaux ad hoc avec la méthode B évenementielle (Event-B) [ABH+10], notamment le protocole DSR (Dynamic Source Routing) dans [MS11a], le protocole AODV (Ad hoc On-demand Distance Vector) dans [SS13] et le protocole OLSR (Optimized Link State Routing) dans [KP16].

D'autres travaux s'intéressent à des propriétés bien précises de mécanismes proposés pour des protocoles de routage pour les réseaux ad hoc. [Cav02] recense une série de propositions pour la modélisation et l'évaluation de la qualité de service dans les protocoles de communication avec des méthodes et langages formels. Des propositions similaires et plus récentes sont présentés pour la vérification de propriétés de sécurité dans les protocoles de communication, notamment par [Gye14] et par [DAR14].

1.4 Approches mixtes méthodes formelles et simulation

Deux problèmes persistent néanmoins aux approches formelles et plaident en faveur d'une approche combinée avec la simulation. Le premier concerne l'évaluation conjointe des composants qui s'adaptent à la modélisation formelle avec les autres composants de l'ITS qui ne s'y prêtent pas. Des outils tels qu'Event-B permettent d'évaluer uniquement des modèles formels, alors que les outils de simulation offrent la possibilité de connecter des modèles et des matériels hétérogènes dans un seul processus d'évaluation. Le second est lié au fait que, bien que permettant d'animer un nombre limité d'objets afin de vérifier le comportement d'un système en présence d' interactions, les outils formels n'offrent pas cette possibilité à grande échelle. Une première solution vient de [YHF16] qui proposent d'intégrer les mécanismes du formalisme DEVS dans l'outil de modélisation formelle Promela [RFP05]. Cette approche résout en partie les problèmes liés à l'animation des modèles et au passage à l'échelle, mais pas ceux concernant les in-

teractions entre les modèles formels et les modèles des composants de l'ITS qui ne se prêtent pas à la modélisation formelle.

L'idée de cette thèse s'inscrit dans cette dernière catégorie d'approche. Plutôt que d'insérer la simulation à événements discrets entre les phases du processus formel, nous défendons l'idée d'introduire la modélisation formelle dans un processus de multi-modélisation DEVS. De cette façon, les modèles formels entrent en interaction avec l'ensemble des autres modèles basés sur des formalismes différents et qui font partie du système de transport modélisé. Ce qui permettrait d'apporter une solution au problème susmentionné. La suite de ce manuscrit sera consacré à la description des travaux que nous avons effectués pour évoluer dans ce sens, avec une illustration au fil de l'eau sur la conception d'un protocole de routage pour un réseau véhiculaire ad hoc.

Chapitre 2

Simulation d'un protocole de communication

Sommaire

2.1 Introduction

La conception des protocoles pour les réseaux de communication repose généralement sur des modèles fonctionnels élaborés selon les besoins du système. Dans les systèmes de transport intelligents, les fonctionnalités étudiées incluent l'auto-organisation, le routage, la fiabilité, la qualité de service et la sécurité. La criticité des systèmes de transport, du fait qu'ils transportent des êtres humains, impose des exigences très précises sur les performances des technologies de communication dans différentes configurations du réseau de transport et de son trafic. Par exemple la validation du remplacement de la technologie de télécommunication filaire du système européen de gestion de trafic ferroviaire (ERTMS) par la technologie sans fil et mobile GSM a nécessité près de quatre années de tests dans les années 90 [Son12]. De nos jours, le développement et l'obsolescence rapides à la fois des technologies et des applications ne permettrait pas des tests aux délais et coûts aussi importants. Il est donc fait de plus en plus recourt à la simulation, notamment pour évaluer les technologies et protocoles dédiés aux ITS.

Ce chapitre montre comment le formalisme DEVS peut être appliqué à la modélisation d'un protocole de routage pour les réseaux ad hoc à partir de sa spécification fonctionnelle. L'objectif est de présenter une première approche de conception et d'évaluation de protocole basée sur la simulation avant de montrer comment elle pourrait être mise associée à une approche formelle pour aboutir au but de cette thèse.

Le principal résultat de ce chapitre est un modèle DEVS du protocole OLSR implémentant la méthode de clustering CBL [RWS+18] récemment proposé pour les réseaux ad hoc dans le cadre d'une collaboration.

2.2 Concepts pour la simulation et la modélisation

Cette partie définit les différentes notions de système et de modèle ainsi que le formalisme DEVS, ses modèles, ses extensions et sa hiérarchie. Elle est inspirée en grande partie de [Her16].

2.2.1 Notion de système

Un système est un ensemble d'entités ou d'éléments correspondant à l'interrogation du modélisateur concernant l'ensemble des éléments essentiels liés à un objet d'études. Par conséquent, pour comprendre la dynamique et le fonctionnement d'un système, il faut identifier les éléments qui constituent le sujet d'études. Cette étape n'est pas toujours simple et les chercheurs doivent souvent procéder par tâtonnements.

Il existe plusieurs types de systèmes selon :

— Le système complexe qui est un ensemble composé d'un grand nombre d'éléments en interaction locale et simultanée.

— Le système dynamique qui est un système où la dynamique est définie en fonction d'un état courant, une base de temps et une fonction d'évolution des états par rapport à des entrées du système.

— Le système spatio-temporel qui est un système dynamique où les processus et les éléments sont définis relativement à un espace.

2.2.2 Notion de modèle

Un modèle est une représentation abstraite, conceptuelle, graphique ou visuelle des systèmes ou des processus permettant d' analyser, de décrire, d'expliquer, simuler ces phénomènes ou processus. Un modèle permet de déterminer un résultat final à partir de certaines données d'entrée. On considère que la création d'un modèle est une partie essentielle de toute activité qui se base sur une simulation ou modélisation. La constitution d'un modèle, dans la mesure où elle peut être orientée de manière générique ou guidée à partir d'une théorie métaphysique, est toujours le résultat d'un contexte de test rigoureux, préparé de manière à ne pas être influencé par les attentes ou par l'interprétation. L'observation sur laquelle est fondée la formulation des modèles théoriques valables est invariante par rapport à l'observateur.

Pour créer un modèle, il est nécessaire de considérer une série d'hypothèses afin que le phénomène à représenter soit suffisamment reflété dans l'idéalisation, mais toujours assez simple pour pouvoir être manipulé et étudié. Un modèle peut être présenté donc comme le résultat de l'activité de modélisation, si on se restreint à l'activité de modélisation. Ce résultat implique un schéma expérimental, un ou des paradigmes et une méthodologie.

2.3 Le formalisme DEVS

Un formalisme est l'outil opérationnel du paradigme. Ce dernier est associé à un ou plusieurs formalismes. Le formalisme est l'outil permettant de spécifier l'ensemble des fonctions nécessaires à l'évolution du modèle. Il est l'outil d'expression des paradigmes. Le formalisme à événements discrets DEVS (Discrete Event System Specification) est présenté pour la première fois en 1976 [ZKP00].

DEVS est un formalisme mathématique qui sert de base à un cadre de modélisation et simulation. Ce formalisme de haut niveau est basé sur les événements discrets pour la simulation, la modélisation et l'analyse de systèmes complexes et continus. L'un de nombreux avantages de DEVS est qu'il permet la construction de modèles hiérarchiques et modulaires, le couplage de compo-

sants et même la prise en charge de la simulation continue de modèles d'événements discrets par approximation temporelle. Grâce à la plate-forme hiérarchique naturelle de DEVS, il permet le couplage modulaire des modèles qui existent afin de construire des systèmes plus grands et plus complexes. Comme le formalisme est fermé sous le couplage, un modèle couplé peut-être traité comme un composant DEVS de base. DEVS est notamment reconnu pour son universalité, son abstraction, son expressivité, et son indépendance de toute implémentation. Il intègre intrinsèquement la dimension temporelle qui est explicite et permet de manipuler conjointement les deux concepts d'état et d'évènement. Grâce à la durée de vie de ses états, DEVS introduit la possibilité d'évolution autonome du modèle. Depuis sa création, ce formalisme a été utilisé à de nombreuses reprises en biologie, en sociologie et en ingénierie notamment les systèmes de communications [DPR13].

Pour tous ces avantages, nous avons choisi d'utiliser le formalisme DEVS pour la simulation de notre réseau ad hoc de véhicules.

2.3.1 Les outils fondamentaux du formalisme DEVS

Le formalisme DEVS est composé de deux objets élémentaires tels que les modèles atomiques et les modèles couplés. Ces modèles permettent d'intégrer la dimension temporelle de façon explicite et de manipuler conjointement les deux concepts d'état et d'événement.

Modèles atomiques

Un modèle atomique est destiné pour gérer la dynamique du système à simuler. Il est défini comme un ensemble de ports d'entrée, de sortie, et un ensemble de fonctions. Ces données sont concaténées dans le sept paramètres ci-après :

$$M = \langle X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta \rangle \tag{2.1}$$

Où,

— X est l'ensemble des ports et des valeurs d'entrée.

— Y est l'ensemble des ports et des valeurs de sortie.

— S est l'ensemble des états partiels ou séquentiels contenant les variables d'états du système.

— $\delta_{int} : S \rightarrow S$ est la fonction de transition interne qui est conçue pour passer le système d'un état à un autre de manière autonome. Elle détermine les états futurs des états actifs, pourvu que leur durée de vie ne soit pas infinie.

— $\delta_{ext} : Q \times X \rightarrow S$ est la fonction de transition externe qui fait passer le système d'un état à un autre après l'apparition d'un événement externe perturbateur.

Où,

— Q = {(s, e)|s ∈ S, 0 ≤ e ≤ ta(s)} est l'ensemble total des états du système. "e" représente le temps écoulé dans l'état "s" dès la dernière transition. Le concept d'état total (s, e) permet de déterminer un état futur selon le temps écoulé dans l'état présent.

— $\lambda : S \rightarrow Y$ est la fonction de sortie. Elle n'est activée que si le temps écoulé dans un état donné est égal à sa durée de vie.

— $ta : S \rightarrow R_0^+ \cup +\infty$ est la fonction d'avancement du temps. Elle fournit la durée pendant laquelle le système reste dans un certain état.

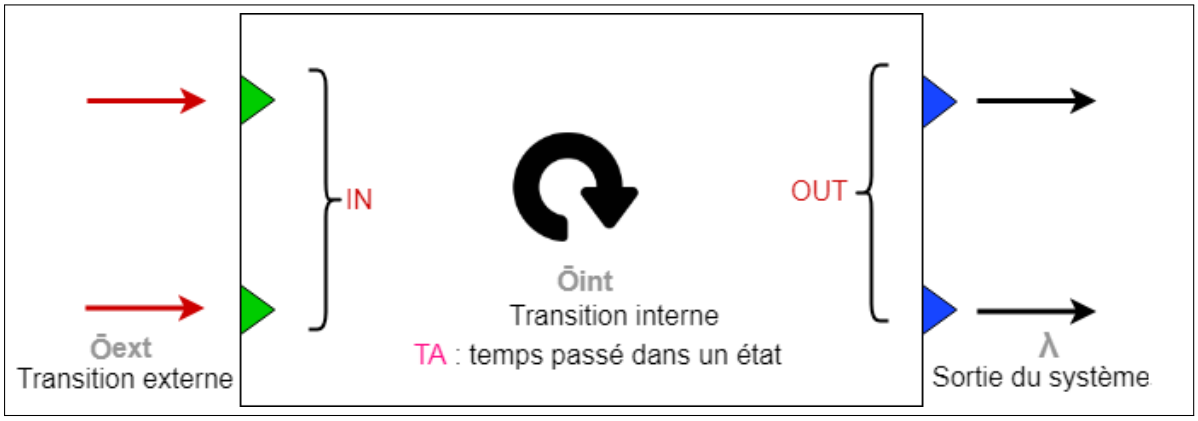

FIGURE 2.1 – Schématisation d'un modèle atomique

Modèles couplés

Le modèle couplé représente un système à événements discrets sous la forme d'un réseau de composants couplés. Le fait de coupler plusieurs modèles atomiques afin de former une superstructure, appelée modèle couplé. Un modèle couplé a la même structure et les mêmes propriétés qu'un modèle atomique. Chaque modèle couplé représente l'ensemble des connexions qui attachent les modèles qui le composent. C'est un modèle qui peut se composer à la fois de modèles atomiques et de modèles couplés. Ceux-ci peuvent eux-même se composer de modèles couplés et ainsi de suite. Cette composition fournit une hiérarchie de modèles propre au formalisme DEVS. Un modèle couplé est défini par :

$$N = \langle X_N, Y_N, D, \{M_d, d \in D\}, EIC, EOC, IC, Select \rangle \tag{2.2}$$

Où,

— X_N et Y_N possèdent la même définition que pour les modèles atomiques.

— D est l'ensemble des identifiants des modèles qui construisent le modèle couplé.

— M_d est l'ensemble des modèles (atomiques ou couplés) qui construisent le modèle couplé.

— EIC = {((N, a), (d, b)) | a ∈ IPorts, b IPorts d } définit la liste des ports d'entrée du modèle couplé connectés aux ports d'entrée des sous-modèles.

Où,

— IPorts est l'ensemble des ports d'entrées de N

— IP$orts_d$ est l'ensemble des ports d'entrées des sous-modèles d ∈ D

— EOC = {((N, a), (d, b)) | a ∈ OPorts, b ∈ OP$orts_d$ } est la liste des ports de sorties du modèle couplé connectés aux ports de sorties des sous-modèles.

Où,

— IPorts est l'ensemble des ports de sorties de N

— IP$orts_d$ est l'ensemble des ports de sorties des sous-modèles d ∈ D

— IC = ((i, a), (j, b)) | i, j D, i 6 = j, a OPorts i , b OPorts j : est la liste des connexions internes au modèle couplé (connexion entre les sous-modèles).

Une sortie d'un modèle ne peut pas être couplée à l'une de ses entrées.

— Select définit une priorité entre événements simultanés destinés à des composants différents

La figure 2.2 montre une représentation graphique d'un modèle nommé MODEL qui se compose lui-même de deux modèles couplés "model_i", "model_ii" et d'un modèle atomique "Atomic". Chaque modèle couplé se compose lui-même de plusieurs modèles atomiques. Afin d'apporter un exemple simple, le sous-modèle couplé model_ii est développé. Ses caractéristiques sont :

— X_N = {in} ainsi que l'ensemble des valeurs prises par in

— Y_N = {out} ainsi que l'ensemble des valeurs prises par out

— M = deux modèles atomiques

— D = D, E

— EIC = {((model_ii, in), (D, in d))}

— EOC = {((E, out e), (model_ii, out))}

— IC = {((D, out d), (E, in e))}

Les ensembles EIC, EOC et IC gèrent le transfert des événements entre les différents modèles. Cet exemple présente également la nature hiérarchique des modèles couplés.

2.3.2 Les extensions de DEVS

La modélisation DEVS que nous proposons, repose sur deux variantes du formalisme DEVS : P-DEVS [CZ94] [Cho96] et DS-DEVS (Dynamic Structure DEVS [Bar97].

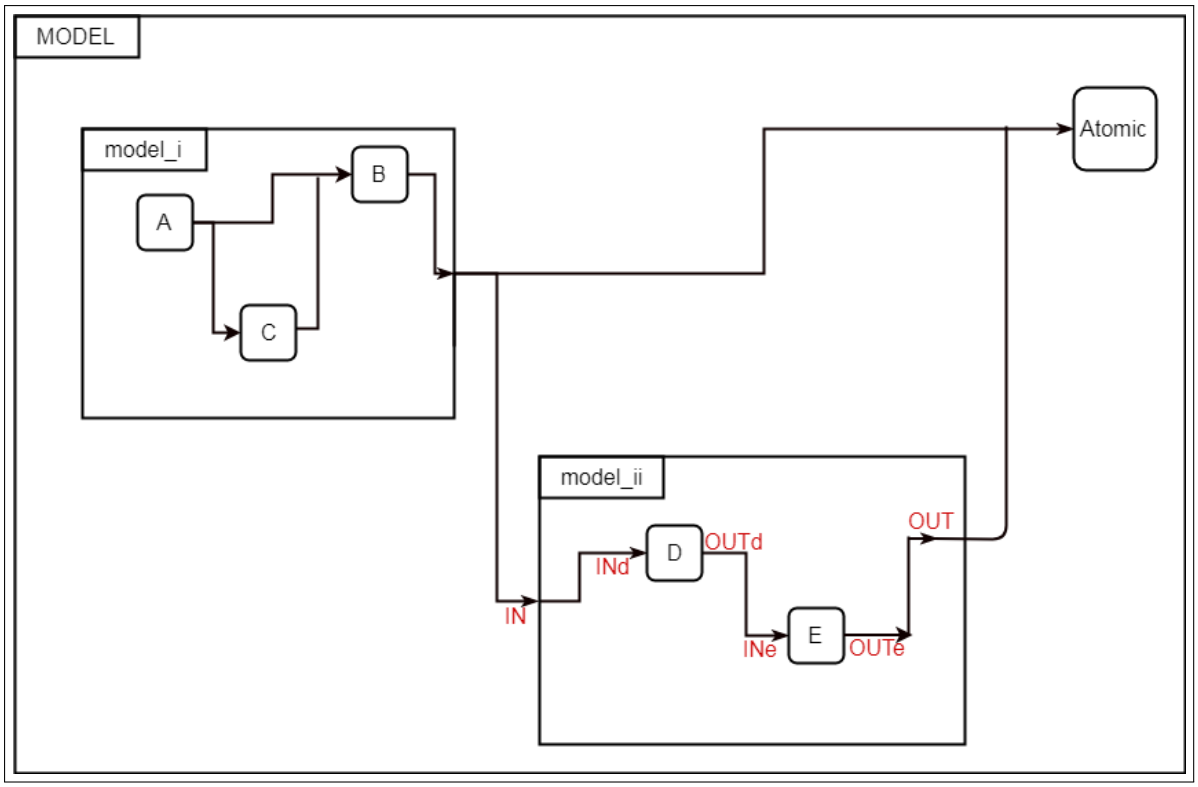

FIGURE 2.2 – Exemple et représentation d'un modèle couplé

Parallel-DEVS, est une extension du formalisme DEVS. Le but de cette extension est d'introduire le concept d'événements simultanés. Grâce à ce formalisme, il est devenu plus souple de gérer les priorités en les traitant à la fois au niveau du modèle couplé et au niveau du modèle atomique.

DS-DEVS est caractérisé par :

— offrir une structure de connexions dynamiques

— au cours du temps, le couplage entre les modèles qui composent le modèle global évolue

— les modèles peuvent apparaître et disparaître

— l'évolution est pilotée par un modèle spécial : le modèle exécutif

2.3.3 Hiérarchie structurée d'un modèle DEVS

B.P. Zeigler [ZKP00] a proposé une idée qui consiste à avoir un simulateur abstrait générique. Pour gérer le comportement d'un modèle, le simulateur abstrait met en œuvre ses fonctions sous forme d'une description algorithmique. Il en existe trois types :

— le coordinateur racine qui permet de gérer globalement la simulation en contrôlant l'horloge globale. Il a la possibilité de connaître la date du prochain événement en interrogeant ses modèles enfants.

— Le coordinateur qui est la représentation abstraite du modèle couplé. Il a la charge de gérer le routage des messages entre les modèles couplés. En plus, il assure la collection des dates de réveil de chacun de ses modèles enfants.

— le simulateur qui se définit comme la représentation abstraite du modèle atomique. Il simule le modèle atomique en utilisant les fonctions définies par DEVS.

DEVS représente opérationnellement un modèle couplé via le simulateur abstrait sous le nom d'arbre de simulation ou hiérarchie structurée de simulateurs abstraits dont le but est de surveiller les liens de dépendances entre les structures abstraites (simulateurs - coordinateurs, coordinateurs - coordinateurs). Mais, cette représentation ne permet pas de connaître les liens qui unissent les simulateurs entre eux.

2.3.4 VLE «Virtual Laboratory Environment»

[VLE] (Virtual Laboratory Environment) est un environnement multi-agents qui se repose sur le principe de la programmation orientée composants légers ou POC. Il est développé au LIL (Laboratoire d'Informatique du Littoral). VLE assure une description dynamique des modèles à l'aide de la réutilisation des fonctions pré-développées dans ses bibliothèques. C'est un logiciel Open Source développé en C++ basé sur le formalisme à événements discrets DEVS, ce qui lui assure une certaine rapidité d'exécution. Pour des raisons de portabilité, son interface graphique de conception et de visualisation d'expérimentation GVLE est écrite en Java.

VLE permet de prendre en charge les différentes étapes du cycle de modélisation-simulation :

— La modélisation via son interface graphique GVLE

— Les plans d'expériences

— La simulation via VLE

— L'analyse gérée par l'application EOV ou via l'outil AVLE basé sur la plateforme d'outils statistiques R

VLE est portable grâce à la bibliothèque glibmm. Les composants gnetmm et libxml++ assurent respectivement l'API portable de transfert d'information sur les réseaux IP et la manipulation des applications XML. Libutils, libvalue, libgeometry, libdata sont des outils de classes et fonctions supplémentaires. Les bibliothèques de VLE sont divisées entre des bibliothèques de base des différentes applications telles que libdevs, libgraph et libvpz ou des bibliothèques d'extensions tels que libextention et gtkmm.

Le package VLE contient quartes programmes principaux VLE, GVLE, AVLE et EOV et cinq types de composants légers : les greffons de simulation, les greffons flux de sortie, les Greffons de

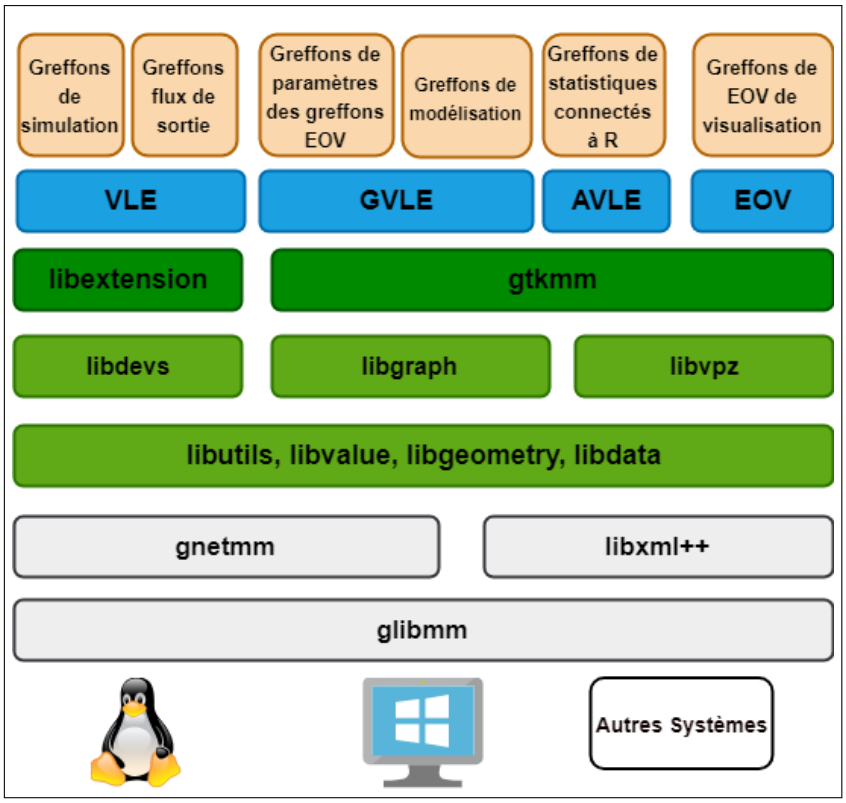

FIGURE 2.3 – Architecture générale de VLE

modélisation, les greffons de paramètres des greffons EOV, les greffons de statistiques connectés à R et les greffons EOV de visualisation.

2.4 Modélisation DEVS d'un système de transport : cas d'un réseau ad hoc de véhicules

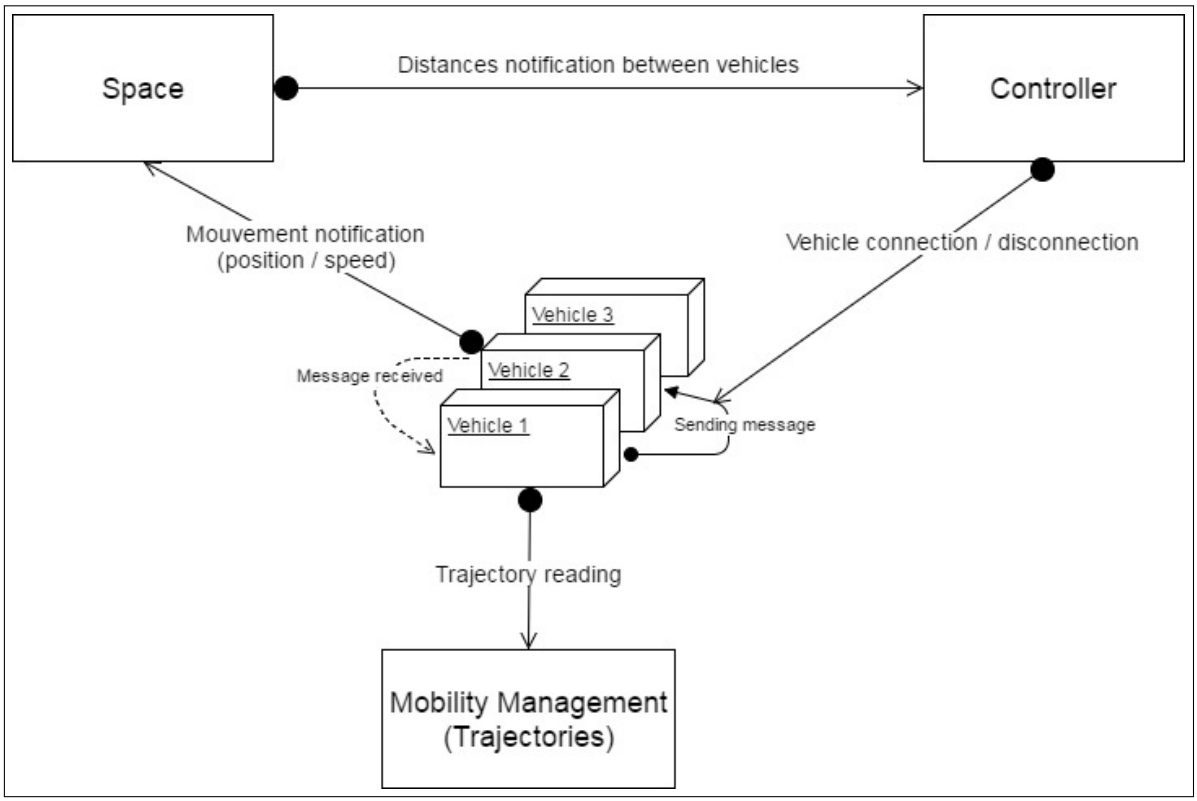

FIGURE 2.4 – Modèle DEVS d'un réseau ad hoc de véhicules

Un réseau ad hoc de véhicules (VANET) est une forme particulière de réseau mobile ad hoc (MANET) dans lequel les noeuds sont des véhicules qui échangent des informations afin d'améliorer la fluidité du traffic et la sécurité des usagers de la route. Dans un VANET, les véhicules se déplacent à des vitesses généralement plus élevées que dans les MANET initiaux, mais ils sont généralement plus limités spatialement du fait de la configuration des voies de circulation.

Il existe déjà des approches de réseaux exploitant DEVS, comme : [DRG02] et [Bou16].

Cette partie met en place une modélisation DEVS de l'ensemble des composants permettant de simuler un réseau ad hoc de véhicules. Il sera plus facile de vérifier que les modèles de simulation obtenus à partir des modèles formels sont conformes à la spécification du protocole si nous avons la certitude que l'environnement de simulation permet réellement une modélisation fiable du protocole. Les modèles DEVS présentés dans ce travail constituent une preuve de concept de la faisabilité d'une modélisation d'un protocole de communication pour les transports dans l'environnement de multi-modélisation DEVS offert par VLE.

La modélisation DEVS que nous proposons repose sur deux variantes du formalisme DEVS : P-DEVS (Parallel-DEVS [CZ94] et DS-DEVS (Dynamic Structure DEVS [Bar97]. La première variante permet de prendre en compte la simultanéité des événements externes et des transitions interne

par l'introduction de la fonction de conflit. La notion d'état transitoire est aussi présente dans cette variante avec l'introduction d'une durée de vie nulle. DS-DEVS et ses améliorations comme DS-DE [Bar98], quant à lui, introduit la possibilité de modifier le graphe des modèles au courant de la simulation. Par exemple, il est possible de créer et détruire des modèles atomiques ou couplés, ou de créer et détruire des connexions entre modèles.

En effet, un premier aspect est la nécessité de créér des instances du modèle atomique véhicule à chaque fois qu'un véhicule intègre le traffic simulé, et de générer à la volée des liens entre ces différentes instances lorsque les véhicules correspondant réunissent les conditions permettant des communications entre eux.

Le second aspect important est la gestion des déplacements des véhicules dans un espace 3D parfaitement connu (des voies de circulation) et continu. Plusieurs options s'ouvrent à nous :

— discrétiser l'espace avec la question du pas de discrétisation ;

— distribuer la définition de l'espace au sein de chaque modèle (de véhicule) ;

— centraliser la définition et la gestion de l'espace à un modèle spécialisé.

Nous avons choisi la troisième option. Le modèle prenant en charge l'espace se nomme «Space» et il collabore avec un modèle, le «Controller», qui possède un type spécial : c'est un exécutif au sens DS-DEVS. Un executif est un modèle atomique, unique au sein d'un modèle couplé, qui possède la faculté de modifier la structure du modèle couplé dans lequel il se trouve. Toutes ces opérations sont traitées par l'algorithme abstrait du coordinateur associé de manière à ce que l'on puisse garantir la causalité. Le couple «Space» - «Controller» a donc en charge la gestion de la localisation des véhicules, la détection et la création dynamique des connexions potentielles entre les véhicules, et l'apparition ainsi que la disparation des véhicules dans le tronçon de voie de circulation étudié.

Le modèle «Space» est averti par le «Controller» lorsqu'un véhicule pénètre dans le tronçon et lorsqu'il en sort. En fonction des changements de vitesse et de direction notifiés par le véhicule, le modèle «Space» recalcule les connexions. Ce calcul est réalisable assez facilement de manière événementielle. C'est notamment à ce niveau que le modélisateur peut gérer la prise en compte de différents modèles de propagations pour déterminer la possibilité d'établissement d'une connexion ou de réussite d'une transmission. À chaque nouvelle connexion, le «Controller» est averti et met à jour le graphe des connexions en conséquence. Un mécanisme analogue gère la suppression des connexions dans le graphe. En fonction de ses connexions, une instance du modèle «Vehicle» est en mesure de transmettre et recevoir des messages et réagir de manière adéquate, notamment par rapport au protocole de communications ou aux applications.

2.4.1 Description du modèle de l'espace : Space

Le modèle "Space" a pour rôle la gestion des positions, il est donc capable d'avoir une vision globale sur les positions des véhicules et d'implémenter les fonctions de gestion du temps. Notre modèle représente un espace à trois dimensions où les véhicules occupent une position à tout moment. Il peut déplacer les véhicules dans cet espace en fonction des mouvements notifiés par chaque instance du modèle «Vehicle, puis enclencher la création ou destruction de connexions entre les véhicules en conséquence.

Comme le montre la figure 2.5, en dehors des fonctions classiques des modèles atomiques DEVS, les fonctions additionnelles de Space sont :

— moveVehicles() : parcourt les différentes instances représentant les véhicules et met à jour leurs positions à partir des informations qui sont fournies par les véhicules sur leur mouvement.

— updateConnections() : met à jour les connections entre les véhicules en recalculant les distances actualisées entre les paires de véhicules.

Cette modélisation de l'espace s'est inspirée d'une modélisation proposée initialement pour un groupe d'oiseaux dans un espace 3D. En plus, dans le cadre de cette modélisation, nous avons utilisé un modèle de propagation simple qui est basé sur la distance. Lorsque deux véhicules sont à une distance inférieure à la portée de la technologie de communication, alors une liaison est créée, et elle sera détruite lorsque cette condition ne sera plus vérifiée. Toute prise en compte d'un modèle de propagation différent dans le réseau ad hoc de véhicules pourrait être réalisée au sein de ce modèle.

2.4.2 Le modèle exécutif : Controller

Le Controller a la capacité d'aller voir tout ce qui se passe lorsqu'un véhicule qui essaie d'envoyer un message à un autre véhicule. Le contrôleur intercepte le message et il est à mesure de consulter la table des connexions pour savoir s'il y a une connexion active à ce moment-là et appliquer des règles précises pour déterminer les conditions dans lesquelles l'émission ou la réception du message se font. On retrouve ici un mécanisme similaire à celui utilisé dans [Riv], un simulateur également basé sur DEVS, pour modéliser les erreurs de transmission des messages. La figure 2.7 montre la connexion entre les deux modèles Space et Controller ainsi que les ports de communication qui les relient.

Lorsqu'il reçoit des données en provenance de Space ou de Vehicle sur un de ses ports, le Controller exécutera une des fonctions suivantes (figure 2.6) :

```
class Space : public vle::devs::Dynamics
{
public:
    Space(const vle::devs::DynamicsInit& init,
          const vle::devs::InitEventList& events);


    virtual void output(const vle::devs::Time& time,
                        vle::devs::ExternalEventList& output) const;
    virtual vle::devs::Time timeAdvance() const;
    virtual vle::devs::Time init(const vle::devs::Time&);
    virtual void internalTransition(const vle::devs::Time& time);
    virtual void externalTransition(const vle::devs::ExternalEventList& event,
                                    const vle::devs::Time& time);
    virtual vle::value::Value* observation(
        const vle::devs::ObservationEvent&) const;
    virtual void confluentTransitions(
        const vle::devs::Time& time,
        const vle::devs::ExternalEventList& events)
    {
        externalTransition(events, time);
        internalTransition(time);
    }

private:
    void moveVehicles();
    void updateConnections();
}
```

FIGURE 2.5 – Interface de la classe du modèle atomique Space

— ConnectVehicles() : crée une liaison entre deux instances de Vehicle lorsque Space lui indique que les conditions pour le faire sont remplies ;

— createModel() : crée une instance de Vehicle dans la simulation lorsque le scénario implique l'apparition d'un véhicule dans le traffic routier simulé ;

— disconnectVehicles() : supprime une liaison entre deux instances de Vehicle lorsque Space indique que les conditions d'existence d'une liaison ne sont plus remplies.

2.4.3 Le modèle atomique du véhicule : Vehicle

Le modèle atomique prenant en charge les véhicules «Vehicle» est capable d'émettre et recevoir des messages. Le protocole de communication sera donc modélisé par une série de fonctions executées par le modèle atomique Vehicle lors de l'envoi et de la réception des messages de routage. Ainsi, il est possible d'observer chaque instance de Vehicle en particulier, mais également les effets au niveau du réseau global en se plaçant sur le Controller. Comme illustré sur la figure 2.8, les fonctions du modèle Vehicle sont entre autres :

— updateMovement() : cette fonction permet de mettre à jour les mouvements d'un véhicule à partir du modèle de mobilité ;

— calcul_AV_AR() : détermine si une autre instance de Vehicle dont on a reçu un message se

```
class Controler:public vle::devs::Executive
{
public:
    Controler(const vle::devs::ExecutiveInit& init,
              const vle::devs::InitEventList& events)
        : vle::devs::Executive(init, events),
        mVehicleNumber(vle::value::toInteger(events.get("vehicleNumber"))),
        mTrajectoryPath(vle::value::toString(events.get("trajectoryPath"))),
        mTrajectoryPrefix(vle::value::toString(events.get("trajectoryPrefix"))),
        mTrajectoryExt(vle::value::toString(events.get("trajectoryExt")))
    {
        std::valarray < double > values_min(0., 3);
        std::valarray < double > values_max(0., 3);

        values_min[0] = toDouble(toSet(events.get("boundaries_x"))[0]);
        values_min[1] = toDouble(toSet(events.get("boundaries_y"))[0]);
        values_min[2] = toDouble(toSet(events.get("boundaries_z"))[0]);
        values_max[0] = toDouble(toSet(events.get("boundaries_x"))[1]);
        values_max[1] = toDouble(toSet(events.get("boundaries_y"))[1]);
        values_max[2] = toDouble(toSet(events.get("boundaries_z"))[1]);
        mBoundaries.set(Point < 3, double >(values_min),
                        Point < 3, double >(values_max));
    }
    virtual ~Controler() { }
    virtual vle::devs::Time init(const vle::devs::Time&);
    virtual void output(const vle::devs::Time&,
                        vle::devs::ExternalEventList&) const;
    virtual vle::devs::Time timeAdvance() const;
    virtual void externalTransition(
        const vle::devs::ExternalEventList&,
        const vle::devs::Time&);
    virtual void internalTransition(const vle::devs::Time&);
    virtual vle::value::Value* observation(
        const vle::devs::ObservationEvent& event) const;

private:
    void connectVehicles(vle::devs::ExternalEvent* e);
    void createModel(int index, int vehicle_index);
    void disconnectVehicles(vle::devs::ExternalEvent* e);
}
```

FIGURE 2.6 – Interface de la Classe du modèle atomique Controller

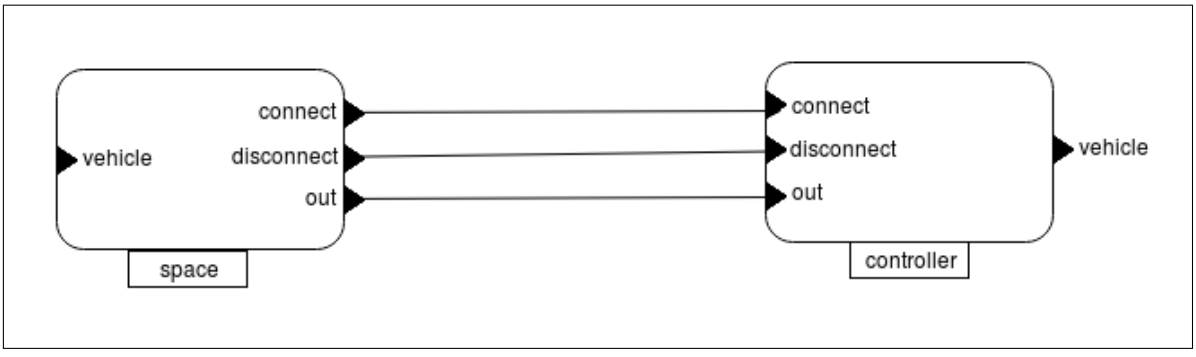

FIGURE 2.7 – Ports de communication entre les deux modèles Space et Controller

trouve à l'avant ou à l'arrière dans le sens de circulation ;

— calcul_direction() : calcule la direction du prochain mouvement d'une instance de Vehicle en fonction de sa position initiale et de sa prochaine position.

```
class Vehicle : public vd::Dynamics,
                public DynamicElement
{
private:
    vle::devs::ExternalEvent* cloneExternalEvent(
        const vle::devs::ExternalEvent* event) const

    void updateMovement(const vd::Time& time)
    void build_hello(const vd::Time& time)
    int calcul_number_veh_LOS(double neighbor_X,double neighbor_Y,
                              double neighbor_Z, std::string neighbor_adrr)

    bool calcul_AV_AR(double x1, double y1, double z1,  double steering_angle1,
                              double x2, double y2, double z2 )

    bool calcul_direction(double steering_angle1,  double steering_angle2 )
    int creation_chaine()
    static void calcul_stats_chaine()
    double calcul_link_duration(double x1, double y1, double z1, double speed1,
                                double steering_angle1, double x2, ndouble y2,
                                double z2, double speed2,double steering_angle2)

    bool comparerNoeudsBranches(double x1, double y1, double z1, double speed1,
                                double steering_angle1, double x2, double y2,
                                double z2, double speed2,double steering_angle2,
                                double x3, double y3, double z3,double speed3,
                                double steering_angle3, bool node_type3)

    bool comparerNoeudsFeuille(string originator_addr, double steering_angle1,
                                double steering_angle2, double speed2)

    bool comparerNoeudsChaine(string originator_addr,double x1, double y1,
                                double z1, double speed1,double steering_angle1,
                                double x2, double y2, double z2, double speed2,
                                double steering_angle2, bool apporte_voisin)

    void update_number_leaf_attached_stat()
    void process_hello(const vle::devs::ExternalEvent* ee, double time)


}
```

FIGURE 2.8 – Interface de la classe du modèle atomique Vehicle

2.4.4 Modèle de mobilité

Le déplacement d'une instance de Vehicle dans l'espace suit une trajectoire qui peut être calculée à partir d'un modèle de mobilité. Dans le cadre de cette modélisation et afin d'illustrer la possibilité d'intégrer des données du domaines dans une modélisation DEVS avec VLE, nous avons opté pour des trajectoires provenant de traces réelles de la circulation sur un tronçon de l'autoroute A27 en Hauts-de-France [CSR+18].

Par exemple, selon le fichier de trajectoire associé, au moment où le véhicule numéro X entre dans le tronçon du traffic simulé, le Controller crée une instance de Vehicle et lui transmet son fichier de trajectoire. L'instance X créée calcule sa direction à partir de sa position initiale et de sa

position suivante, récupère la durée de ce mouvement puis transmets toutes ces informations à Space. Il génère un évenement de mise à jour de son mouvement en fonction de la durée du précédent et entame l'envoi périodique ou à la demande des messages du protocole de communication ainsi que ceux des applications modélisées.

2.5 Modélisation d'un protocole de routage pour les réseaux ad hoc

Afin d'illustrer comment le protocole peut être évalué, nous allons modéliser le protocole Optimized Link State Routing (OLSR), décrit dans [CJ03], enrichi par une méthode de clustering particulière récemment proposée Chain-Branch-Leaf [RWS+18]. Le protocole va être modélisé à l'intérieur du modèle atomique Vehicle, mais l'effet de l'organisation produit par le protocole sera visible à différents niveaux dans les différents modèles atomiques ainsi qu'au niveau du modèle couplé représentant le système de transport, en l'occurence le réseau ad hoc.

2.5.1 Introduction aux protocoles de routage ad hoc

De nombreux travaux proposent des approches pour la conception et l'évaluation des protocoles de communication pour les réseaux ad hoc de véhicules. Ce type de réseau opère dans l'hypothèse de l'absence de toute infrastructure préexistante ou lorsque celle-ci est rendue inopérante (pannes, attaques, etc). Malgré un très grand nombre de protocoles de routage ad hoc recensés dans la littérature [LW07] [SAI14], ils peuvent être regroupés en un nombre restreint d'approches. Une première approche concerne les protocoles dits "réactifs" (DSR, AODV, etc) où les véhicules construisent les routes à la demande en fonction du trafic à envoyer [Wal13]. Ces protocoles ne maintiennent pas continuellement la topologie du réseau. Ils génèrent par conséquent moins de trafic de routage et subissent moins la charge liée aux opérations de routage. En revanche, il est très fréquent que le calcul de routes ne démarre qu'au moment où le trafic doit être envoyé, ce qui peut entraîner des délais importants, moins convenables à un environnement à forte mobilité et à des applications temps réel. Une autre famille de protocoles dits "proactifs" (OLSR, etc) maintiennent en permanence une structure dans le réseau de véhicules et calculent des routes prêtes à être utilisées dès que du trafic se présente pour une des destinations connues [SK14]. Cette catégorie est bien adaptée aux environnements nécessitant une forte réactivité, mais ils peuvent connaître des congestions dues au trafic de routage lorsque le réseau est dense. Ces protocoles implémentent donc en général une méthode de clustering permettant de réduire les effets des transmissions par diffusion (broadcast) et éviter les congestions. La dernière approche développée concerne les protocoles dits "géographiques" (GRP, etc) qui utilisent des informations sur la position géographique des véhicules pour organiser l'acheminement de données [MSHK11]. Ils

présupposent l'existence d'un service centralisé de localisation qui fournit à chaque véhicule les informations de localisation des autres véhicules, ce qui ne cadre pas tout à fait avec un fonctionnement en mode ad hoc pur où aucune infrastructure ne doit être présupposée pour garantir la continuité des fonctionnalités.

Nous avons choisi d'illustrer notre modélisation sur une variante du protocole OLSR, car son caractère proactif permet de créer une structure sur laquelle il est davantage possible de mettre en évidence une variété de protocoles en nombre et selon leurs caractéristiques. De nombreux travaux ayant déjà été proposé avec la méthode de clustering classique d'OLSR, la technique des relais multipoints, nous avons opté de mettre en évidence une proposition récente à laquelle nous participons, à savoir la méthode de clustering Chain-Branch-Leaf (CBL) [RWS+18].

2.5.2 Présentation de l'organisation structurelle de CBL dans le protocole OLSR

CBL est un algorithme distribué qui crée une hiérarchie entre les nœuds afin de construire des clusters tels que chaque nœud d'un cluster puisse communiquer directement avec la tête du cluster sans passer par un autre nœud relais. Il utilise les messages HELLO du protocole OLSR pour construire sa structure hiérarchique. Il définit deux types de nœuds : les nœuds Branch et les nœuds Leaf. Les deux types de nœuds émettent des messages HELLO périodiques afin de construire une structure, appelée chaîne, qui relie les nœuds dans chaque direction du trafic. Certaines définitions sont précisées comme suit :

— Un nœud Branch est un nœud de tête de cluster qui est élu par d'autres nœuds (Branch ou Leaf). Il émet des messages HELLO comme tout nœud, mais il est le seul autorisé à émettre des messages de contrôle de topologie (TC), à retransmettre des messages d'application et à participer à la construction d'une chaîne. Lors de la retransmission d'un message, selon la demande de l'application spécifiée dans les champs d'en-tête, un nœud Branch peut le transmettre à ses propres nœuds Leaf, ses nœuds Branch amont et aval, ainsi qu'aux noeuds Branch dans le trafic de la direction opposée.

— Un nœud Leaf est un nœud ordinaire qui essaie de se connecter au nœud de Branch le plus proche. Si aucun nœud Branch n'est détecté, le nœud Leaf élit le voisin se déplaçant à la vitesse la plus faible et dans la même direction de trafic en tant que son noeud Branch. Un nœud Leaf peut transmettre des messages HELLO et des messages d'application dont il est l'auteur.

— Une Chain est une dorsale virtuelle composée des nœuds Branch connectés. Idéalement, une Chain unique devrait être créée par sens de circulation. Dans un contexte routier longitudinal comme celui des autoroutes, les chaînes se comportent comme une épine dorsale virtuelle similaire à celle que l'on obtiendrait avec une infrastructure RSU déployée sur la

route. Elle offre à ses nœuds des routes pour transférer des messages d'application sur de longues distances en amont, en aval ou dans les deux directions.

— BranchChoice, l'adresse du noeud Branch choisi par un noeud Leaf. Le champ BranchChoice est vide si le nœud qui sélectionne est un nœud Branch.

— ChainUP, l'adresse du nœud Branch choisi pour relayer le trafic amont. Le champ ChainUP est vide si le nœud est un nœud Leaf. ChainDO, l'adresse du nœud Branch choisi pour relayer le trafic en aval. Le champ ChainDO est vide si le nœud est un nœud Leaf.

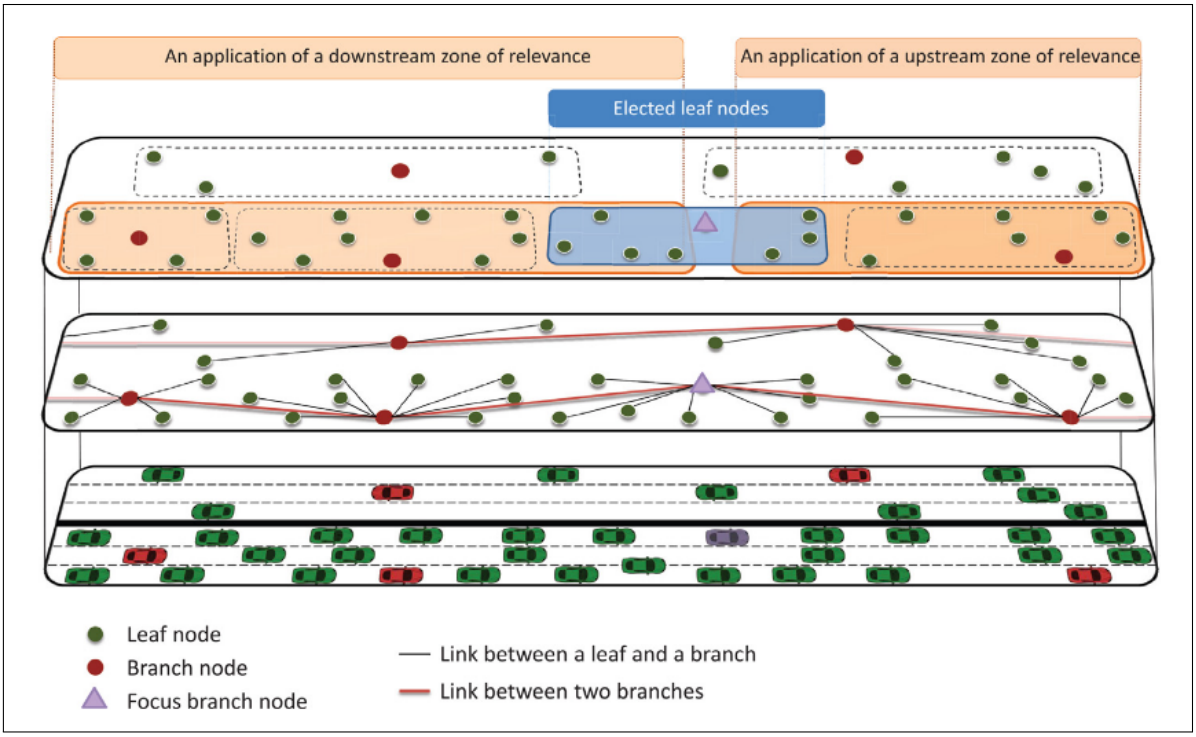

FIGURE 2.9 – Rôles et zones avec Chain-Branch-Leaf [RWS+18]

La figure 2.9 résume comment CBL considère les véhicules comme des nœuds (Leaf ou Branch) et comment il définit les zones amont et aval pertinentes des deux côtés d'un nœud Branch.

2.6 Validation des modèles

Les modèles réalisés ont été validés par une simulation élaborée de la manière suivante :

— Taille du réseau : le réseau comprend deux scénarios avec un total de 176 (faible densité) et 358 (forte densité) véhicules circulant sur l'A27 ;

— Mobilité : chaque véhicule est associé à l'une des trajectoires disponibles qui détermine à la fois sa date d'entrée sur le tronçon de route simulé, son mouvement matérialisé par des segments successifs à vitesses constantes hétérogènes (segments de longueurs différentes

et vitesses respectives différentes), ainsi que sa date de sortie du tronçon. Ces trajectoires sont issues de données réelles de circulation sur 6 km de l'A27 collectées durant 10 minutes ;

— Le protocole OLSR fonctionne selon les paramètres de [RWS+18]. Dans cette validation, seuls l'envoi et la réception des messages HELLO nécessaires à CBL sont modélisés ;

La figure 2.10 montre le nombre de véhicules qui entrent dans la simulation, ceux qui sortent et le nombre moyen de véhicules présents simultanément durant la simulation. On voit également le nombre total de connexions actives à chaque pas d'observation. Ces statistiques ont été définies pendant la modélisation et sont placés dans le modèle atomique adéquat parmi Vehicle (pour les statistiques locales), Space et Controller (pour les statistiques globales).

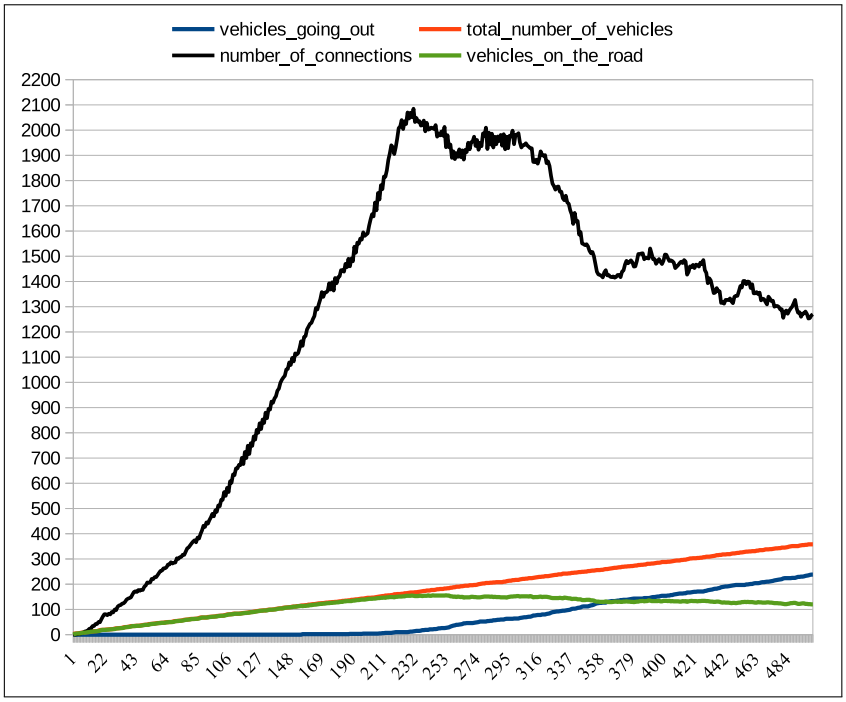

FIGURE 2.10 – Evolution du nombre de véhicules dans la simulation

La figure 2.11 montre que la valeur la plus élevée sur le nombre de connexions entre instances de Vehicle est atteinte lorsque le nombre maximum de véhicules simultanés est atteint (155), puis elle diminue et se stabilise autour de 1300 lorsque le nombre de véhicules se stabilise autour de 130. Le nombre total de nœuds Branch est de 15 % à 35 % du nombre total de nœuds. Par conséquent, environ 70% des véhicules sont des nœuds Leaf, ce qui correspond bien aux résultats de performance de CBL [RWS+18]. Le nombre de nœuds Branch par Chain indique qu'il y a 1 à 2 sous-ensembles connectés dans chaque sens de circulation, ce qui confirme que le réseau de véhicules est entièrement connecté, au moins dans le même sens de circulation.

La figure 2.12 précise que chaque véhicule a en moyenne 80 voisins dans l'ensemble du VANET, soit 40 dans le même sens de circulation. Ceux qui sont des nœuds Branch sont sélectionnés par 20% à 40% de leurs voisins directs qui s'y attachent comme des nœuds Leaf.

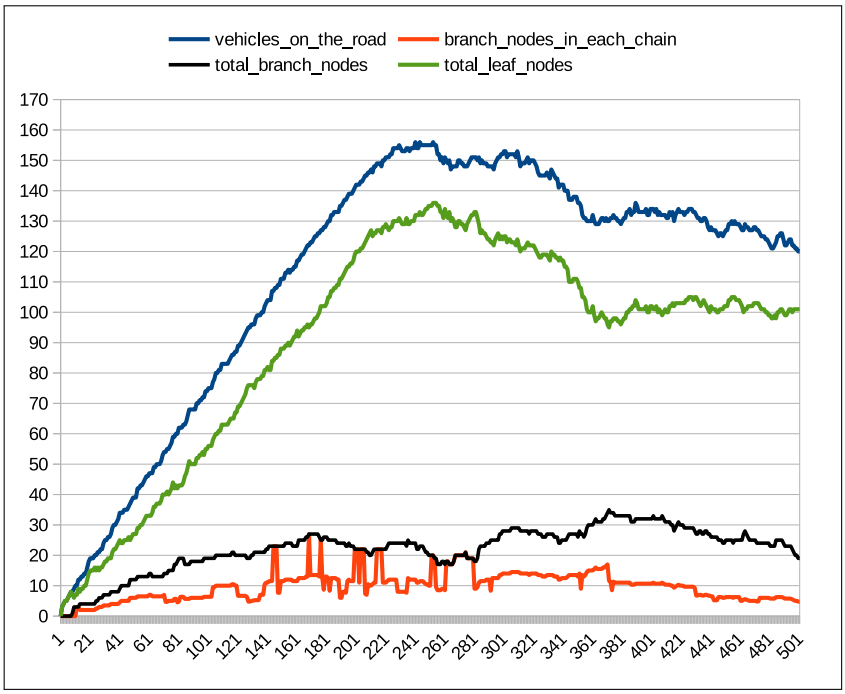

FIGURE 2.11 – Evolution du nombre des noeuds Branch et Leaf dans la simulation

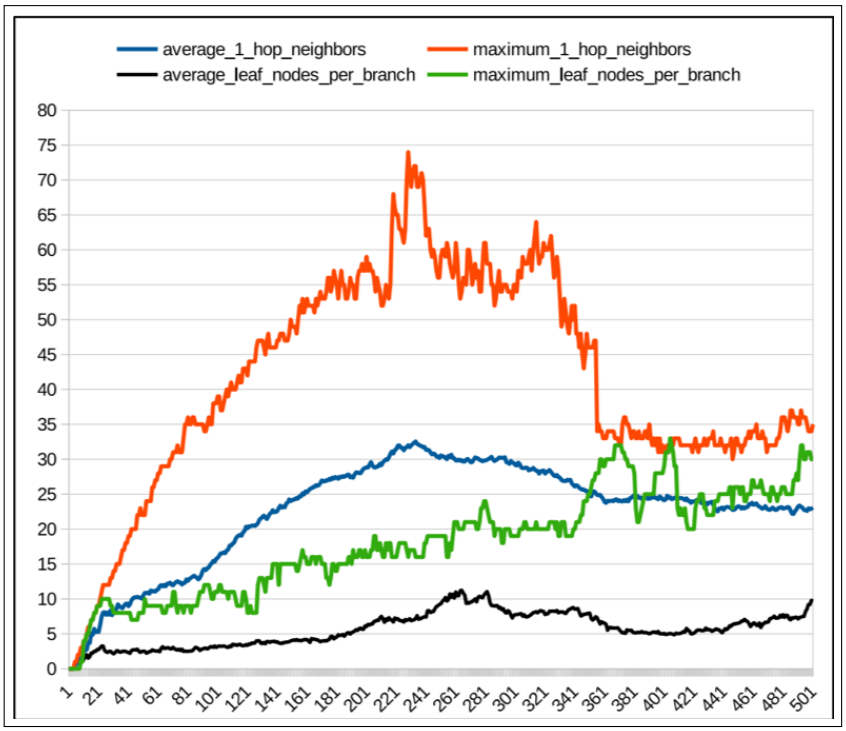

FIGURE 2.12 – Evolution du nombre des noeuds Leaf et des voisins par noeud Branch

La figure 2.13 montre que chacun des nœuds Leaf reste attaché à un nœud Branch 90% du temps, et qu'un nœud reste sans nœud Branch moins de 10% du temps. Ceci démontre que les nœuds isolés ne restent que peu de temps hors du réseau ad hoc véhiculaire.

Tous ces résultats montrent que ces modèles DEVS d'un VANET, implémentés avec VLE, permettent d'obtenir les mêmes performances pour OLSR avec le schéma CBL que celles obtenues avec des outils bien établis tels que MATLAB et OPNET[RWS+18]. Ils contribuent à valider les modèles conçus.

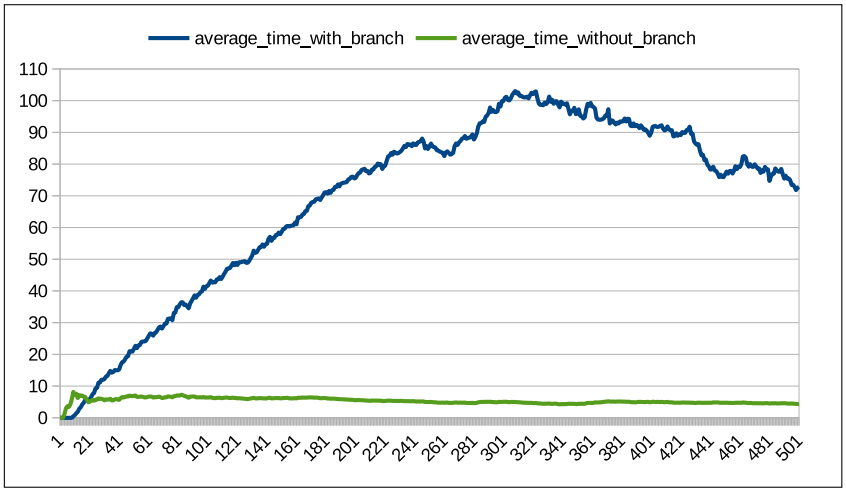

FIGURE 2.13 – La durée pendant laquelle un nœud Leaf est attaché ou non à un noeud Branch dans la simulation.

2.7 Conclusion

Ce chapitre a présenté les modèles DEVS de composants d'un réseau ad hoc et illustré comment les principales fonctionnalités y ont été modélisées. Il a également montré comment un protocole de communication, en l'occurence le protocole de routage OLSR avec la méthode de clustering CBL, peut être modélisé au niveau du modèle du véhicule.

VLE n'étant pas un outil traditionnellement utilisé pour la modélisation des réseaux de communications, il ne dispose pas d'une bibliothèque très fournie de modèles prêts à l'usage dans ce domaine. Néanmoins, il apparaît clairement qu'ils pourraient y être ajoutées à partir des solutions open source. Une démonstration a également été faite de la manière dont des données réelles du domaine peuvent être intégrées à une simulation avec VLE. La compatibilité de DEVS avec HLA laisse présager qu'il serait également assez simple d'intégrer des interactions avec d'autres simulateurs. Tous ces modèles ont été mis en œuvre avec VLE et validés par un scénario de réseau ad hoc de véhicules construit à partir de données réelles de trafic sur l'autoroute A27 dans les Hauts-de-France.

A travers les résultats de l'évaluation par simulation, on note que tous les observables dont

découlent les statistiques sont davantage assimilables à des indicateurs de performance qu'à des preuves de propriétés sur les modèles. Cet état de fait souligne la limite de la simulation seule et impose le recours à un autre paradigme de modélisation pour la conception et l'évaluation des protocoles de communication.

Chapitre 3

Modélisation formelle d'un protocoles de communication

Sommaire

3.1 Introduction

Ce chapitre présente une modélisation formelle réalisée avec Event-B du protocole de communication dont une modélisation DEVS a été proposée au chapitre précédent. Il y sera illustré comment certaines propriétés liées à la surêté et à la sécurité ont été introduites et vérifiées.

3.1.1 Généralités sur la modélisation formelle

Les méthodes formelles [CW96] [Win90] sont des techniques mathématiques pour la spécification, la conception et la vérification des systèmes logiciels. Ils utilisent la logique mathématique pour raisonner rigoureusement sur la construction du logiciel afin d'obtenir un système fiable et robuste. Ils permettent une bonne garantie concernant l'absence de "bug" dans le logiciel. Ils s'assurent que l'implémentation d'un logiciel est conforme à ses spécifications. Cependant, l'utilisation des méthodes formelles est généralement coûteuse et généralement réservée au développement de systèmes critiques.

Un système critique pour la sécurité est un système dont la défaillance ou le mauvais fonctionnement peut avoir des conséquences dramatiques. les conséquences, telles que la mort, les blessures graves, les dommages matériels et les atteintes à l'environnement. On peux mentionner par exemple les systèmes ferroviaires [But02], aérospatiaux [ABW10] et donc la route du futur avec son lot de véhicules autonomes.

La sûreté des systèmes critiques pour la sécurité peut être améliorée en utilisant des méthodes formelles pour développer leurs sous-systèmes critiques [Rus93]. Des preuves mathématiques permettent de s'assurer que les spécifications répondent aux exigences. Parfois, le code exécutable peut être généré directement à partir des spécifications formelles. Dans ce cas, les tests unitaires peuvent être retirés du processus de développement car la correction de la construction est garantie par des preuves mathématiques.

Il y a souvent des malentendus sur le rôle et l'application des méthodes formelles [BH95a] [BH95b] [BH06]. Ces idées ont surestimé l'importance de la formalisation complète d'une spécification ou d'une conception. Comme la formalisation complète d'un système est une tâche difficile et coûteuse. Diverses méthodes formelles légères [Jac01] [Bar05] proposées mettent l'accent sur des spécifications partielles et des applications ciblées. Les méthodes formelles peuvent être appliquées à différents stades du processus de développement, en particulier lors de la spécification, de la vérification et de la génération du code.

3.1.2 Présentation de B événementiel : Event-B

La méthode B [AA05] est une méthode formelle qui permet de raisonner sur des systèmes complexes et le développement logiciel. Les systèmes sont modélisés comme un ensemble de machines abstraites indépendantes, pour lesquelles une approche orientée objet est utilisée à toutes les étapes du développement. Contrairement à Classical-B et B-Sharp, Event-B est une évolution utilisant uniquement la notion d'événements pour décrire des actions et non plus des opérations. Ceci explique le succès d'Event-B dans la modélisation et l'analyse des protocoles de routage. Il est utilisé dans [Gye14] et [DAR14] pour vérifier les propriétés de sécurité des protocoles réseau, dans [SS13] pour la modélisation formelle du protocole AODV (Ad hoc On-demand Distance Vector), et dans [MS11a] pour la modélisation du protocole DSR (Dynamic Source Routing). Un modèle détaillé Event-B du protocole OLSR est également proposé dans [KP16]. Les modèles Event-B sont composés des deux constructions de base : les contextes et les machines. La figure 3.1 présente les différents composants d'un contexte et d'une machine.

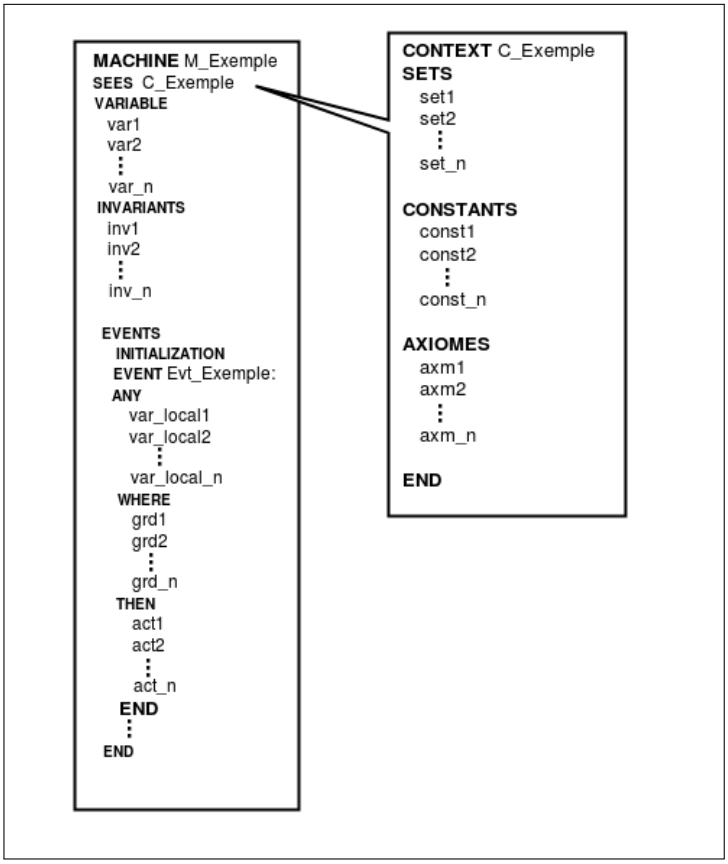

FIGURE 3.1 – Structure d'un modèle Event-B

Dans Event-B, il existe trois types de modèles :

— le premier type de modèle ne contient que des contextes, car il ne représente qu'une structure mathématique ;

— le second type de modèle ne contient que des machines et pas contexte. Le modèle ne

contient alors que des machines non paramétrées ;

— le dernier type de modèle contient des machines et des contextes. Les machines sont alors paramétrées par des contextes.

Un contexte Event-B contient les informations statiques du modèle. Il contient des ensembles, des constantes, des théorèmes et des axiomes. Les axiomes représentent les propriétés du modèle. Leur définition nécessite les ensembles et les constantes. Les théorèmes représentent les propriétés des axiomes.

Une machine contient la partie dynamique du système. Il spécifie les propriétés comportementales des modèles Event-B. Dans une machine, on peut trouver des variables, des invariants, des théorèmes et des événements et du modèle. Les variables décrivent l'état du modèle. Ils sont mis à jour à travers les événements. Les événements peuvent contenir des propriétés logiques reliant des constantes et des variables. Les invariants sont des propriétés qui doivent être vérifiées pour tout état du modèle.

Il existe de nombreuses relations liées aux machines et aux contextes. Une machine peut "**raffiner**" une autre machine. Un contexte peut '**extend**" un autre contexte. Le raffinement offre la possibilité de développer un modèle de plus en plus précis. Une modélisation Event-B peut donc contenir une séquence de modèles intégrés. Chacun de ces modèles embarqués est le raffinement du modèle précédent dans la séquence. Ainsi, chaque modèle obtenu par raffinement contiendra plus de variables que son abstraction.

La notion de transition dans Event-B est représentée par un événement dans une machine. Les événements sont composés de gardes et d'actions. Les gardes représentent les conditions à respecter pour qu'un événement se déclenche. Les actions représentent les modifications appliquées aux variables par l'événement. Les événements sont généralement composés de trois blocs : **When** (guard), **Then** (action) et **End**.

Nous avons choisi de modéliser formellement notre système avec Event-B pour plusieurs raisons. Il permet de séparer la machine du contexte, réduire la complexité, et aider à comprendre correctement les concepts d'abstraction et de raffinement. Event-B offre en plus la possibilité d'être démarré à partir du diagramme d'état. Dans la machine, un état de machine est représenté par un diagramme d'états, et la relation entre deux états est représentée par un événement. Ces deux états représentent respectivement les conditions préalables et postérieures de l'événement. La condition actuelle commence lorsque les gardes sont satisfaits, jusqu'à ce que les conditions suivantes soient remplies. Le modèle Event-B est plus facile à affiner et supporte le modèle formalisé de différents niveaux de raffinement, de sorte que chaque machine abstraite ait son échelle appropriée. Dans Event-B l'état est défini par une variable et garantit que l'invariant est vrai quelle que soit la valeur de la variable. La plupart des démonstrations théoriques peuvent être effectuées

automatiquement à l'aide d'outils pertinents, ce qui réduit les difficultés de vérification [SC].

3.1.3 L'outil Rodin

Il existe plusieurs outils qui servent à la modélisation formelle avec la méthode B, notamment ABTools, Atelier-B, Rodin, etc. Grâce à sa facilité d'utilisation et son extensibilité, nous avons choisi Rodin[ABHV08] [ABH+10] [JB14] pour présenter nos modèles formels Event-B. Cet outil est une plateforme d'extension d'Éclipse basée sur Java. Il contient le plugin ProB [BL09] pour la vérification et l'animation des contextes ainsi que la validation des modèles. Rodin supporte aussi le raffinement qui permet d'introduire progressivement des détails et de la complexité dans un modèle dans le processus de son amélioration. La figure 3.2 contient les principaux symboles utilisés dans Event-B.

SYMBOLE	EXPLICATION
$\wedge$	et logique
$\neg$	négation
$\vee$	ou logique
$\Rightarrow$	implication
$\Leftrightarrow$	équivalence
$\forall$	pour tout
$\exists$	il existe
$=$	n'est pas égal à
$\varnothing$	ensemble vide
$\times$	produit cartésien
P	ensemble des sous-ensembles
$\in$	appartient à
$\subseteq$	est inclus dans
$\subset$	est strictement inclus dans
$\cup$	union
$\cap$	intersection
$-$	différence d'ensembles
$\bigcup$	union quantifiée
$\bigcap$	intersection quantifiée
$\leftrightarrow$	relation entre deux ensembles
id	relation identité
-1	inverse d'une relation
$\otimes$	produit direct
$\parallel$	produit parallèle
$s \rightarrow t$	fonctions totales
Z	ensemble des entiers relatifs
N	ensemble des entiers positifs
$\leq$	inférieur ou égal
$<$	strictement inférieur
$\geq$	supérieur ou égal
$>$	strictement supérieur
card	nombre d'éléments d'un ensemble quelconque
seq(E)	séquences finies d'éléments de E

FIGURE 3.2 – Principaux symboles du language Event-B

3.2 Modélisation d'un protocole basique pour les réseaux ad hoc

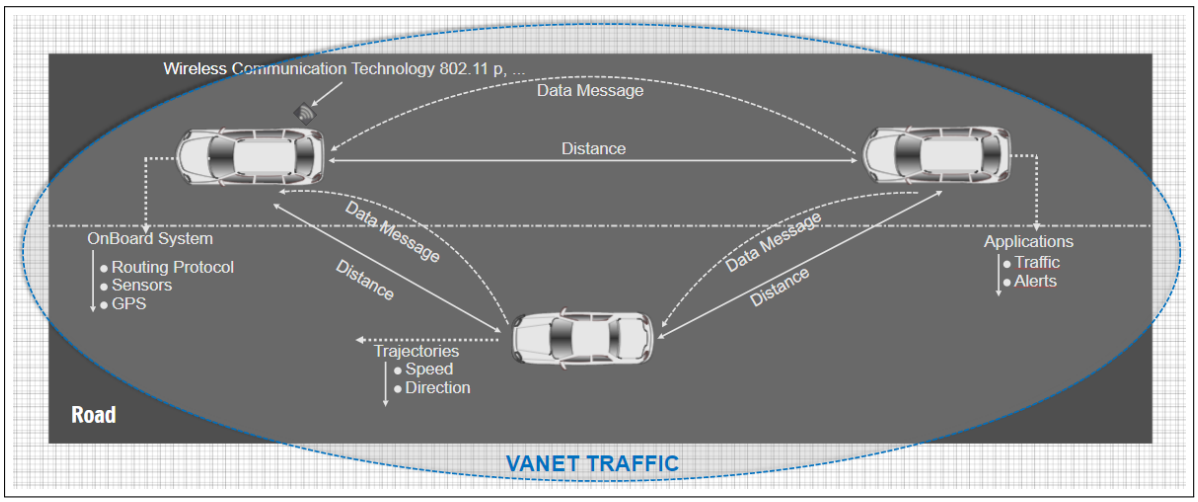

FIGURE 3.3 – Réseau ad hoc pour véhicules

Dans cette section, nous présentons un modèle de protocole de routage basique pour les réseaux ad hoc. Ce modèle implémente des concepts et des fonctionnalités simples quasi obligatoire pour tout protocole de routage qui aurait vocation à être utilisé dans un réseau ad hoc. Il est directement inspiré, à quelques détails mineurs près du modèle de protocole de routage basique proposé par [KP16].

3.2.1 La partie statique du modèle : Contexte C0

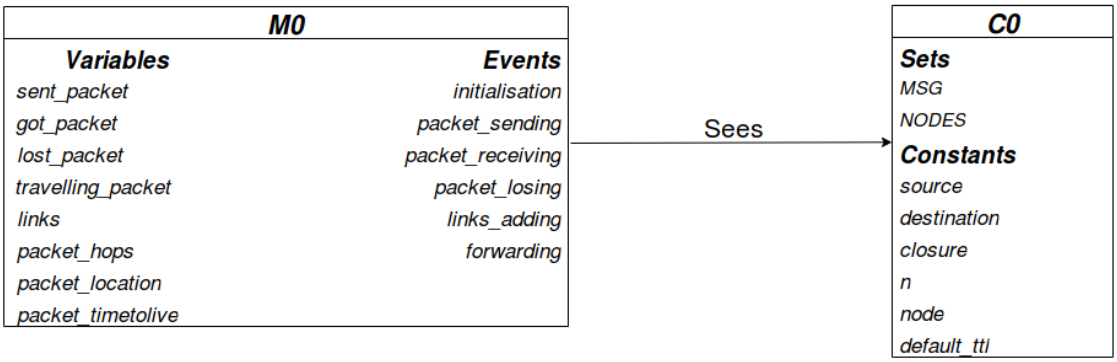

FIGURE 3.4 – Schéma du modèle d'un protocole basique

Dans le contexte C0, nous définissons deux ensembles, NODES et MSG, et cinq constantes : source, destination, closure, n et node. L'ensemble NODES modélise les nœuds du réseau. cet ensemble est fini et non vide, deux propriétés modélisées par les axiomes axm1 et axm2, respectivement. Le nombre d'éléments dans NODES est égal à n (axm3) qui est supérieur à 1 (axm4). Cette propriété fait référence à l'acception générale qu'il faut au moins deux terminaux pour considérer qu'il s'agit d'un réseau.

SETS

MSG

NODES

CONSTANTS

source

destination

closure

n

node

default_ttl

AXIOMS

axm1 : finite(NODES)

axm2 : NODES $\neq \emptyset$

axm3 : card(NODES) = n

axm4 : n > 1

axm5 : finite(MSG)

axm6 : MSG $\neq \emptyset$

axm7 : source $\in$ MSG $\rightarrow$ NODES

axm8 : destination $\in$ MSG $\rightarrow$ NODES

axm9 : closure $\in$ (NODES $\rightarrow$ NODES) $\leftrightarrow$ (NODES $\rightarrow$ NODES)

axm10 : $\forall$ r $\cdot$ r $\subseteq$ closure(r)

axm11 : $\forall$ r $\cdot$ closure(r) ; r $\subseteq$ closure(r)

axm12 : $\forall$ r, s $\cdot$ r $\subseteq$ s $\wedge$ s ; r $\subseteq$ s $\Rightarrow$ closure(r) $\subseteq$ s

axm13 : node $\notin$ NODES

axm14 : default_ttl = 255

END

Les axiomes axm5 et axm6 définissent l'ensemble des messages (MSG) comme un ensemble fini et non vide. Dans les axiomes axm7 et axm8, les constantes source et destination sont définies comme des applications au sens mathématique (static function au sens informatique) de MSG vers NODES. Ainsi, lorsqu'on applique source à un élément de MSG, on obient un élément de NODES qui est donc le noeud source du message considéré. La constante closure modélise la fermeture transitive des relations binaires entre les nœuds (NODES). Concrètement, elle permet de vérifier que des noeuds font partie d'une même composante connexe : il y a au moins une route reliant chaque paire de noeuds inclus la même closure. Les axiomes axm9 à axm12 modélisent cela. Le nœud constant node modélise un nœud abstrait qui représente tout noeud situé en dehors

du réseau modélisé (axm13). L'axiome (axm14) définit le TTL (Time To Live) par défaut dans le réseau. Ces ensembles, constantes et axiomes du contexte C0, sont "vus" par la machine M0.

3.2.2 La partie dynamique du modèle : la machine abstraite M0

INVARIANTS

inv1 : sent_packet ⊆ MSG

inv2 : lost_packet ⊆ sent_packet

inv3 : got_packet ⊆ sent_packet

inv4 : travelling_packet ⊆ sent_packet

inv5 : got_packet ∩ travelling_packet = ∅

inv6 : got_packet ∩ lost_packet = ∅

inv7 : lost_packet ∩ travelling_packet = ∅

inv8 : got_packet ∪ lost_packet ∪ travelling_packet = sent_packet

inv9 : links ∈ NODES → NODES

inv10 : (NODES ◁ id) ∩ links = ∅

inv11 : ∀ s, d · s ∈ NODES ∧ d ∈ NODES ∧ s ↦ d ∈ links ⇔ d ↦ s ∈ links

inv12 : packet_location ∈ sent_packet → NODES

inv13 : dom(packet_location) = sent_packet

inv14 : ∀ m · m ∈ MSG ∧ m ∉ sent_packet ⇒ (m ∉ got_packet ∧ (∀ s · s ∈ NODES ⇒ m ↦ s ∉ packet_location))

inv15 : ∀ m · m ∈ got_packet ⇒ m ↦ destination(m) ∈ packet_location

inv16 : ∀ m, s, r · m ↦ s ∈ packet_location ∧ m ↦ r ∈ packet_location ⇒ s = r

inv17 : packet_hops ∈ MSG ↔ N

inv18 : packet_timetolive ∈ MSG ↔ N

inv19 : ∀ m · m ∈ sent_packet \ (lost_packet ∪ got_packet) ⇔ m ∈ travelling_packet

inv20 : sent_packet \ lost_packet = got_packet ⇔ travelling_packet = ∅

Dans la machine M0, huit variables sont introduites pour modéliser les fonctionnalités d'un protocole de routage basique. Ces variables sont sent_packet, lost_packet, got_packet, travelling_packet, links, packet_hops, packet_location et packet_timetolive.

Lorsqu'un paquet est effectivement émis dans le réseau, il est placé dans sent_packet qui est donc déclarée comme un sous-ensemble de MSG à travers un invariant (inv1). La variable lost_packet contient les paquets perdus et got_packet les paquets reçus dans le réseau. Les invariants inv2, inv3 et inv4 expriment trois propriétés évidentes qui sont qu'un paquet ne peut être reçu, perdu ou en transit entre sa source et sa destination que s'il a été envoyé auparavant. Ainsi, Un paquet ne peut être que dans un seul des trois ensembles (inv5 à inv7) : got_packet, travelling_

packet ou lost_packet. De même l'invariant inv8 vérifie qu'un paquet envoyé (donc dans sent_-packet) doit obligatoirement se trouver dans l'un de ces trois ensembles (un paquet déjà émis est forcément soit reçu, soit en train d'être transféré, soit perdu).

Les invariants inv9 à inv11 modélisent les liens entre les différents nœuds du réseau. Par exemple, l'invariant inv10 précise qu'un noeud ne peut pas être connecté à lui-même. L'invariant inv12 définit packet_location comme une application qui associe à chaque paquet envoyé (donc élément de sent_packet) le noeud (de NODES) où il se trouve à un moment donné. Le domaine de définition de packet_location est donc bien sent_packet (inv13). Aucun paquet ne peut être reçu ni transféré dans le réseau s'il ne figure pas dans la liste des paquets envoyés (inv 14). Lorsqu'un nœud reçoit un paquet de données, packet_location l'associe naturellement à ce (inv15). L'invariant inv16 garantit qu'un paquet n'est pas localisé sur deux noeuds différents en même temps. Les applications packet_hops et packet_timetolive qui sont des applications de MSG vers N (ensemble des entiers naturels) associent un paquet respectivement au nombre de sauts qu'il a parcouru et au nombre sauts qu'il peut encore parcouru dans le réseau (inv17 et inv18). L'invariant inv19 garantit qu'un paquet envoyé qui n'est pas encore reçu et qui n'est pas perdu doit apparaître dans la liste des paquets en transit (travelling_packet). Enfin, l'invariant inv20 vérifie que si tous les paquets envoyés et non perdus sont bien reçus par leur destination, alors il n'y a plus de paquet en transit (travelling_packet est vide).

Dans ce modèle abstrait, il y a six événements dont l'événement INITIALISATION. Les événements packet_sending, packet_receiving et packet_losing sont communs à tout protocole de routage. L'événement INITIALISATION est le seul événement auto-créé par l'outil Rodin lorsque nous définissons une nouvelle machine, car chaque variable de la machine doit être initialisée d'une manière cohérente avec le modèle.

packet_sending

ANY

s, d, msg

WHERE

grd1 : $msg \in MSG \land msg \notin sent\_packet$

grd2 : $source(msg) = s \land destination(msg) = d$

grd3 : $s \neq d$

grd4 : $s \mapsto d \in closure(links)$

THEN

act1 : $sent\_packet := sent\_packet \cup \{msg\}$

act2 : $packet\_timetolive := packet\_timetolive \lhd \{msg \mapsto default\_ttl\}$

END

Les gardes grd1 à grd4 dans l'événement packet_sending assurent que le paquet à envoyer n'est pas déjà émis depuis son nœud source s vers le nœud destination d, ces deux derniers étant obligatoirement différents, et qu'il existe une route entre les deux.

L'événement packet_receiving survient lorsqu'un paquet, noté msg, arrive sur un noeud. Une vérification est faite qu'il ne s'agit pas d'une copie d'un paquet qui a déjà été traité (grd1) avant qu'il soit ajouté à la liste de paquets reçus (act1).

packet_receiving

ANY

msg

WHERE

grd1 : msg ∈ sent_packet \ (got_packet ∪ lost_packet)

THEN

act1 : got_packet := got_packet ∪ {msg}

END

Les paquets de données perdus sont traités par l'événement packet_losing où la garde (grd1) vérifie que le paquet (msg) n'a pas déjà été traité comme reçu ou perdu avant de l'ajouter à la liste de paquets perdus.

packet_losing

ANY

msg

WHERE

grd1 : msg ∈ sent_packet \ (got_packet ∪ lost_packet)

THEN

act1 : lost_packet := lost_packet ∪ {msg}

END

L'événement links_adding assure l'ajout de nouveaux liens dans le réseau. Les gardes (grd1-grd2) assurent qu'un lien entre la source et la destination n'existe pas déjà avant de procéder à sa création et à son ajout dans l'ensemble de liens. Dans ce niveau de modélisation, la liaison déclarée entre deux noeuds est forcément symétrique parce que les gardes vérifient que la liaison s vers d n'est pas déjà dans link et procède à l'ajout sans vérifier si la liaison inverse y est déjà. Dans le protocole OLSR, seules liaisons symétriques sont effectivement utilisées pour le calcul de routes et la transmission des traffics applicatifs, mais le protocole manipule des structures et des procédures pour détecter et mettre à jour tous les liens, y compris ceux asymétriques. Cette modélisation suppose donc que ce genre de traitement seront faits en amont par des routines

pré-traitement avant de déclencher les événements qui font effectivement l'objet de vérification formelle. En d'autres termes, tant que la liaison n'est pas symétrique, l'événement "links_adding" n'est jamais évoqué au niveau du modèle formel.

links_adding

ANY

s, d

WHERE

grd1 : $s \mapsto d \notin links$

grd2 : $s \neq d$

THEN

act1 : $links := links \cup \{s \mapsto d\}$

END

Nous avons introduit la variable TTL (time-to-live) pour chaque message, car nous pensons qu'il est obligatoire de limiter le transfert des messages à un nombre prédéfini de sauts dans le réseau véhiculaire, qui peut être très important en fonction du trafic routier. Par exemple, nous pouvons observer dans l'événement basic_forwarding qu'un paquet n'est transféré que si le TTL correspondant le permet (grd6). Dans OLSR, cette variable est fixée à 1 (TTL=1) pour les messages HELLO à leur création afin d'éviter leur retransmission.

basic_forwarding

ANY

d, a, b, msg

WHERE

grd1 : $msg \in travelling\_packet$

grd2 : $destination(msg) = d$

grd3 : $a \neq destination(msg) \wedge msg \mapsto a \in packet\_location \wedge msg \mapsto b \notin$

packet_location

grd4 : $a \mapsto d \in closure(links)$

grd5 : $a \mapsto b \in links$

grd6 : $packet\_timetolive(msg) > 1$

THEN

act1 : $packet\_location := (packet\_location \setminus \{msg \mapsto a\}) \cup \{msg \mapsto b\}$

act2 : $packet\_hops := packet\_hops \lhd \{msg \mapsto packet\_hop(msg) + 1\}$

act3 : $packet\_timetolive := packet\_timetolive \lhd \{msg \mapsto$

$packet\_timetolive(msg) - 1\}$

END

3.2.3 Synthèse

Ce premier modèle de protocole de base légèrement enrichi par rapport à sa version initiale [KP16] a été soumis à la vérification formelle. Toutes les obligations de preuves engendrées par le modèle ont été déchargées automatiquement par le prouveur 3.1. Il va donc nous servir de base pour développer un modèle du protocole OLSR implémentant la méthode de clustering Chain-Branch-Leaf (CBL).

TABLEAU 3.1 – Proof Statistics for a basic protocol.

Model	Number of Proof Obligations	Automatically Discharged	Interactively Discharged
C0	4	4	0
M0	62	62	0
Total	66	66	0

3.3 Modélisation formelle d'un protocole avancé dans le réseaux ad hoc

3.3.1 Présentation de l'organisation structurelle de CBL dans le protocole OLSR

Comme évoqué dans le chapitre précédent, les protocoles de routage proactifs offrent un plus large panel de possibilités du fait qu'ils maintiennent une topologie locale du réseau. Nous avons donc opté pour une variante du protocole OLSR implémentant l'algorithme de clustering Chain-Branch-Leaf (CBL). Comme décrit sur la figure 3.5, CBL construit une chaîne formée par des véhicules particulièrement stables dits "Branch" auxquels s'attachent les véhicules situés dans leur zone de couverture, appelés "Leaf". En pratique, CBL suit le trafic et forme donc tente donc de former une chaîne unique pour toutes les voies d'une route qui sont orientées vers la même direction. Ainsi, sur une autoroute à deux sens de circulation, CBL créera deux chaînes indépendantes, une pour chaque sens de circulation.

Grâce aux messages de routage d'OLSR, les véhicules échangent des informations qui permettent à chacun d'eux de décider de manière complètement distribuée s'il est Branch ou Leaf et, dans ce dernier cas, de désigner le nœud Branch auquel il se rattache. Comme dans OLSR, lorsqu'un message est diffusé à l'ensemble du réseau, il n'est retransmis que par les véhicules dits "Branch" (A, B et C dans l'exemple). En revanche, CBL réalise une optimisation supplémentaire en permettant d'indiquer dans quel sens le message doit être diffusé. Ainsi, un message peut être diffusé uniquement en amont du traffic (flèche verte sur la figure 3.5) : si l'émetteur est Branch C ou un des ses noeuds Leaf, en l'occurence 5 ou 6, le message est retransmis par le noeud Branch amont choisi par C, en l'occurence B, puis par le noeud Branch amont choisi par B, en l'occurence A, et ainsi de suite. De même, un message peut être diffusé uniquement en aval du traffic (flèche bleue sur la figure 3.5) : si l'émetteur est Branch A ou son noeud Leaf, en l'occurence 1, le message

est retransmis par le noeud Branch aval choisi par A, en l'occurence B, puis par le noeud Branch aval choisi par B, en l'occurence C, et ainsi de suite. Les messages destinés à l'ensemble du traffic sont diffusés en amont et en aval.

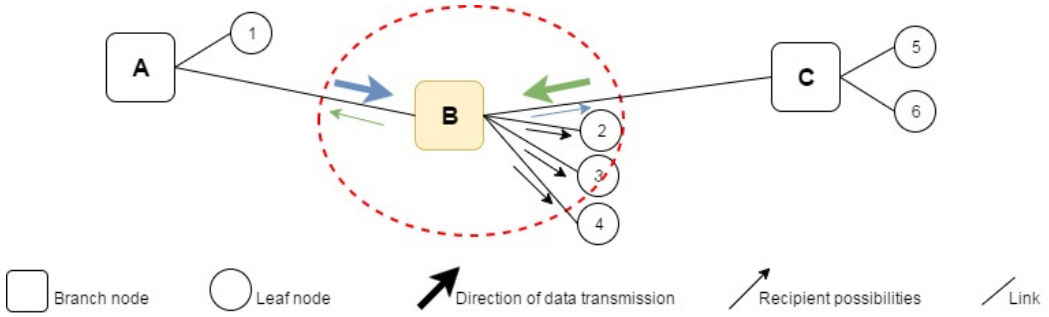

FIGURE 3.5 – Aperçu sur le fonctionnement de la méthode de clustering CBL

3.3.2 Premier raffinement : Contexte 1

Dans cette section, nous présentons le développement du protocole CBL-OLSR à deux différentes couches d'abstractions (Fig. 3.6). Le modèle comprend deux machines ainsi que deux contextes. Le premier contexte et la première machine sont ceux modélisant un protocole de base et que nous avons présentés dans la section précédente. Le nouveau contexte et la nouvelle machine sont également inspirés de [KP16] pour un fonctionnement classique d'OLSR, mais comporte de nombreuses modifications pour permettre de remplacer la technique des multipoints relais par CBL.

Nous étendons le contexte C0 au contexte C1 qui ajoute un ensemble (Traffic) et quatre nouvelles constantes : position, downstream, upstream et tc_ttl. Ils permettent d'introduire dans la modélisation formelle des mécanismes pour gérer les relations spatiales entre les véhicules (noeuds) du réseau. Ainsi, un élément de Traffic peut représenter toutes les voies correspondant au même sens de circulation sur une route. La fonction position permet d'introduire une numérotation des points successifs sur un élément de Traffic (axm1). En d'autres termes, deux véhicules qui roulent côte à côte (sur la même abscisse) sur deux voies contiguës de circulation dans la même direction sont sur le même point numéroté de ce Traffic. Il devient donc possible pour chaque véhicule de déterminer si un autre véhicule est derrière (upstream) ou devant (downstream) dans le sens de circulation du Traffic (axm2 à axm5). Cette représentation de l'espace est suffisante pour prouver formellement les propriétés relatives à CBL, tout autre traitement plus détaillée des positions réelles, notamment dans les calculs de distance seront prises en charge des routines de pré-traitement avant de remonter aux variables et événements du modèle formel.

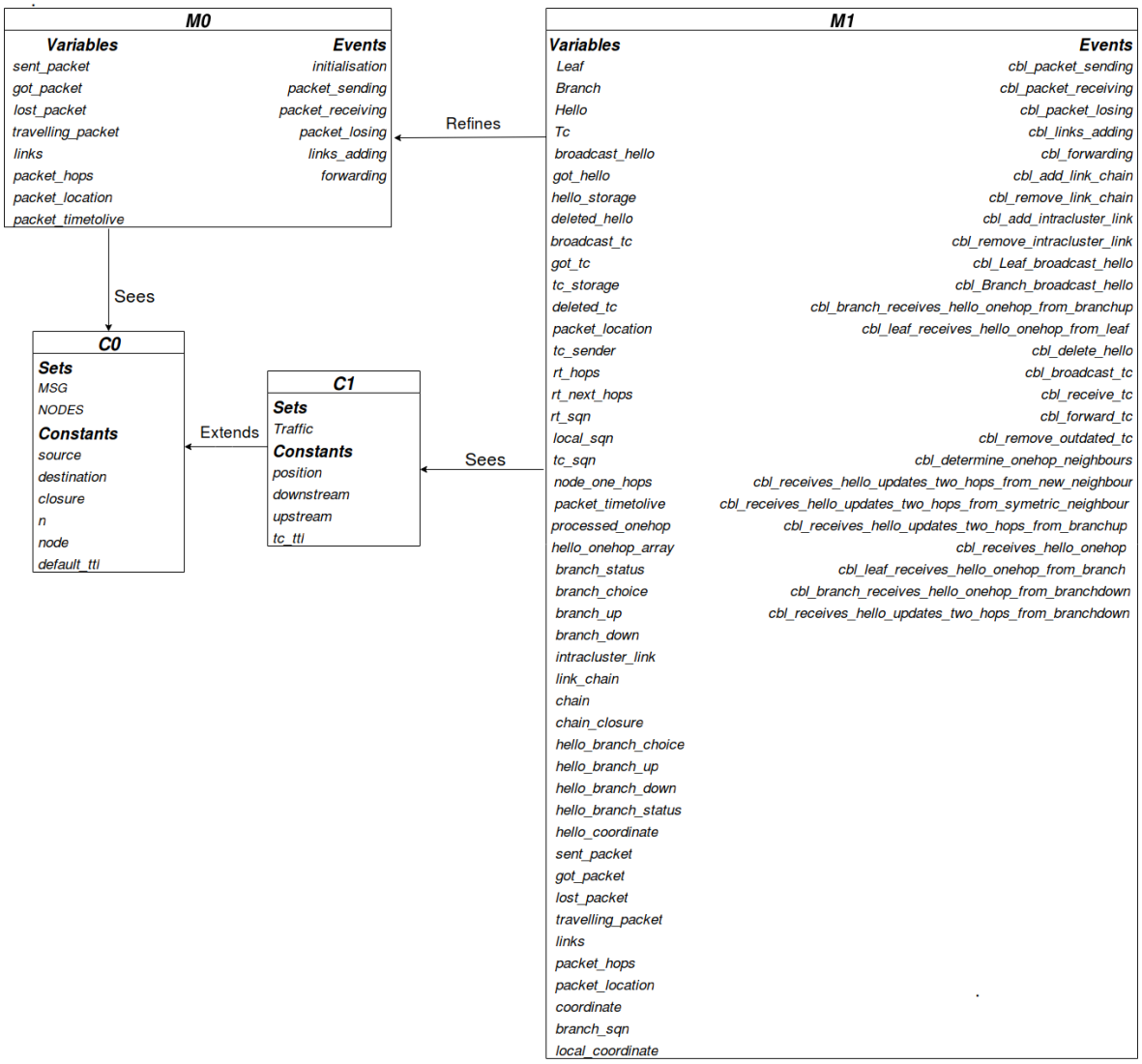

FIGURE 3.6 – Schéma du modèle formel de CBL-OLSR

AXIOMS

axm1 : position ∈ Traffic → N

axm2 : downstream ∈ Traffic → (Traffic → BOOL)

axm3 : upstream ∈ Traffic → (Traffic → BOOL)

axm4 : ∀ c, d · c ∈ Traffic ∧ d ∈ Traffic ∧ position(c) > position(d) ⇒ downstream(c)(d) = TRUE ∧ upstream(c)(d) = FALSE

axm5 : ∀ c, d · c ∈ Traffic ∧ d ∈ Traffic ∧ position(c) < position(d) ⇒ upstream(c)(d) = TRUE ∧ downstream(c)(d) = FALSE

axm6 : tc_ttl = 255

3.3.3 Premier raffinement : Machine 1

Par rapport au modèle d'OLSR proposé par [KP16], nous introduisons les sous-ensembles Branch et Leaf de NODES qui contiennent respectivement les noeuds Branch et les noeuds Leaf. Nous conservons les variables et événements du modèle initial pour le traitement des messages HELLO et TC, mais y ajoutons les spécificités de CBL. D'autres variables et événements spécifiques à CBL sont également ajoutés au modèle. La plupart des modifications susmentionnées sont assez intuitives à la lecture des modèles, nous allons donc nous concentrer sur l'explication des invariants qui sont la transcription des propriétés attendues du modèle, réseau ad hoc et protocole de communication inclus.

INVARIANTS

inv1 : Leaf ⊆ NODES

inv2 : Branch ⊆ NODES

inv3 : Branch ∩ Leaf = ∅

inv4 : Branch ∪ Leaf = NODES

Les invariants (inv1-inv4) précisent les relations des deux nouveaux sous-ensembles de NODES, Branch et Leaf, entre eux et avec leur ensemble père (NODES) dont ils forment une partition.

inv5 : Hello ⊆ MSG

inv6 : Tc ⊆ MSG

inv7 : broadcast_hello ⊆ Hello

inv8 : got_hello ⊆ broadcast_hello

inv9 : deleted_hello ⊆ broadcast_hello

inv10 : hello_storage ∈ NODES ↔ Hello

inv11 : broadcast_tc ⊆ Tc

inv12 : got_tc ⊆ broadcast_tc

inv13 : deleted_tc ⊆ broadcast_tc

inv14 : tc_storage ∈ NODES ↔ Tc

Les invariants inv5 à inv14 spécifient les ensembles précédemment définis pour les paquets en général dans le modèle d'un protocole de base pour les cas particuliers de messages HELLO et TC. On remarque l'analogie entre got_hello et got_tc avec got_packet, etc.

inv15 : tc_sender ∈ Tc → Branch

inv16 : rt_hops ∈ NODES → (NODES → N)

inv17 : rt_next_hops ∈ NODES → (NODES → NODES)

inv18 : ∃ msg, s · s = source(msg) ∧ msg ∈ got_packet ⇒ packet_hops(msg) > rt_hops(s)(destination(msg))

inv19 : ∀ a, tc · a ↦ tc ∈ tc_storage ∧ packet_hops(tc) = 0 ⇒ source(tc) = tc_sender(tc) ∧ rt_hops(a)(source(tc)) = 1

inv20 : got_hello ∪ ran(hello_storage) ∪ deleted_hello = broadcast_hello

inv21 : got_hello ∩ ran(hello_storage) = ∅

inv22 : ran(hello_storage) ∩ deleted_hello = ∅

inv23 : got_hello ∩ deleted_hello = ∅

inv24 : got_tc ∪ ran(tc_storage) ∪ deleted_tc = broadcast_tc

inv25 : got_tc ∩ ran(tc_storage) = ∅

inv26 : ran(tc_storage) ∩ deleted_tc = ∅

inv27 : got_tc ∩ deleted_tc = ∅

inv28 : ∀ tc · tc ∈ broadcast_tc ∧ tc ∉ ran(tc_storage) ∧ tc ∉ deleted_tc ⇒ tc ∈ got_tc

Les invariants inv15 à inv28 précisent les propriétés classiques liées aux messages topology control (TC) ainsi que les relations entre les différents ensembles de messages de contrôle (HELLO et TC), tels que proposés par [KP16].

inv29 : ∀ a, b · a ↦ b ∈ links ⇔ rt_next_hops(b)(a) = a

inv30 : ∀ s, m · s ↦ destination(m) ∈ closure(links) ⇔ rt_next_hops(s)(destination(m)) ≠ node

inv31 : ∀ s, m · m ↦ s ∈ packet_location ∧ rt_next_hops(s)(destination(m)) = node ⇒ m ∉ got_packet ∧ m ∈ lost_packet

inv32 : rt_sqn ∈ NODES → (NODES → N)

inv33 : local_sqn ∈ NODES → N

inv34 : branch_sqn ∈ NODES → N

inv35 : tc_sqn ∈ Tc → N

inv36 : ∀ tc · tc ∈ Tc ⇒ tc_sqn(tc) > branch_sqn(source(tc))

inv37 : ∀ tc, d · tc ∈ broadcast_tc ∧ d ↦ tc ∈ tc_storage ∧ tc_sqn(tc) ≤ rt_sqn(d)(source(tc)) ⇒ tc ∉ got_tc ∧ tc ∈ deleted_tc

Les invariants inv29 à inv37 introduisent les propriétés liées aux tables de routage, notamment la gestion des numéros de séquence des TC. Ils permettent également de restreindre les relations impliquant les messages TC aux seuls noeuds Branch.

inv38 : hello_onehop_array ∈ Hello → (NODES → BOOL)

inv39 : branch_status ∈ NODES ↔ BOOL

inv40 : node_one_hops ∈ NODES → (NODES → BOOL)

inv41 : processed_onehop ∈ NODES → BOOL

inv42 : ∀ s · s ∈ NODES ∧ branch_status(s) = TRUE ⇔ s ∈ Branch

inv43 : ∀ s · s ∈ NODES ∧ branch_status(s) = FALSE ⇔ s ∈ Leaf

inv44 : ∀ s · s ∈ NODES ∧ branch_status(s) = FALSE ⇔ (¬∃tc · tc ∈ broadcast_tc ∧ source(tc) = s)

inv45 : ∀ s · s ∈ NODES ∧ local_sqn(s) ≠ 0 ⇔ branch_status(s) = TRUE

inv46 : branch_choice ∈ Leaf → NODES

inv47 : ∀ s · s ∈ Leaf ∧ (∃x · x ∈ NODES ∧ node_one_hops(s)(x) = TRUE) ⇒ branch_choice(s) ≠ node

inv48 : ∀ s · s ∈ Leaf ∧ (∃x · x ∈ NODES ∧ node_one_hops(s)(x) = TRUE) ⇔ (∃y · y ∈ NODES ∧ node_one_hops(s)(y) = TRUE ∧ branch_choice(s) = y)

inv49 : ∀ s, d · s ∈ Leaf ∧ d ∈ NODES ∧ branch_choice(s) = d ⇒ d ∈ Branch

inv50 : ∀ s, a, b · s ∈ Leaf ∧ branch_choice(s) = a ∧ branch_choice(s) = b ⇒ a = b

inv51 : branch_up ∈ Branch → NODES

inv52 : branch_down ∈ Branch → NODES

inv53 : ∀ s · s ∈ Branch ∧ (∃x · x ∈ NODES ∧ node_one_hops(s)(x) = TRUE ∧ upstream(coordinate(x))(coordinate(s)) = TRUE) ⇒ branch_up(s) ≠ node

inv54 : ∀ s · s ∈ Branch ∧ (∃x · x ∈ NODES ∧ node_one_hops(s)(x) = TRUE ∧ downstream(coordinate(x))(coordinate(s)) = TRUE) ⇒ branch_down(s) ≠ node

inv55 : ∀ s · s ∈ Branch ∧ (∃x · x ∈ NODES ∧ node_one_hops(s)(x) = TRUE ∧ upstream(coordinate(x))(coordinate(s)) = TRUE) ⇔ (∃y · y ∈ NODES ∧ node_one_-hops(s)(y) = TRUE ∧ upstream(coordinate(y))(coordinate(s)) = TRUE ∧ branch_up(s) = y)

inv56 : ∀ s · s ∈ Branch ∧ (∃x · x ∈ NODES ∧ node_one_hops(s)(x) = TRUE ∧ downstream(coordinate(x))(coordinate(s)) = TRUE) ⇔ (∃y · y ∈ NODES ∧ node_one_-hops(s)(y) = TRUE ∧ downstream(coordinate(y))(coordinate(s)) = TRUE ∧ branch_down(s) = y)

inv57 : ∀ s, d · s ∈ Branch ∧ d ∈ NODES ∧ branch_up(s) = d ⇒ d ∈ Branch

inv58 : ∀ s, d · s ∈ Branch ∧ d ∈ NODES ∧ branch_down(s) = d ⇒ d ∈ Branch

inv59 : ∀ s, a, b · s ∈ Branch ∧ branch_up(s) = a ∧ branch_up(s) = b ⇒ a = b

inv60 : ∀ s, a, b · s ∈ Branch ∧ branch_down(s) = a ∧ branch_down(s) = b ⇒ a = b

Les invariants inv38 à inv50 fixent les règles à observer lors de la réception des messages HELLO pour que la détermination du statut de chaque noeud soit cohérent avec les spécifications de CBL. Par exemple, l'invariant inv48 stipule qu'un noeud X qui conserve le statut de noeud Leaf et qui a des voisins directs doit obligatoirement choisir un de ces voisins comme son noeud Branch (branch_choice(X)).

Les invariants inv51 à inv60 fixent les règles de construction de la chaîne par le mécanisme de choix de noeuds Branch amont et aval par les différents noeuds Branch qui sont dans le même sens de circulation (dans le même objet Traffic). La définitions des liens intra-cluster, inter-cluster ainsi que les règles de construction de routes au sein de la chaîne sont quand à elles matérialisées par les invariants inv61 à inv74.

inv61 : intracluster_link ⊆ links

inv62 : link_chain ⊆ links

inv63 : intracluster_link ∈ NODES ↔ Leaf

inv64 : link_chain ∈ Branch ↔ Branch

inv65 : chain ∈ NODES → (N → BOOL)

inv66 : ∀ s, t · s ∈ Branch ∧ t ∈ Branch ∧ (∃x · x ∈ N ∧ chain(s)(x) = TRUE ∧ chain(t)(x) = TRUE) ⇔ chain(s) = chain(t)

inv67 : ∀ s, x, y · s ∈ Branch ∧ x ∈ N ∧ y ∈ N ∧ chain(s)(x) = TRUE ∧ chain(s)(y) = TRUE ⇒ x = y

inv68 : ∀ s, b · s ∈ Leaf ∧ b ∈ Branch ∧ branch_choice(s) = b ⇒ chain(b) = chain(s)

inv69 : chain_closure ⊆ closure

inv70 : chain_closure ∈ (Branch ↔ Branch) → (Branch ↔ Branch)

inv71 : ∀ s, d · s ∈ Branch ∧ d ∈ Branch ∧ s ↦ d ∈ chain_closure(link_chain) ⇔ chain(s) = chain(d)

inv72 : ∀ s, d · s ∈ Branch ∧ d ∈ Branch ∧ s ↦ d ∉ chain_closure(link_chain) ⇔ chain(s) ≠ chain(d)

inv73 : ∀ s, x, y · s ∈ Branch ∧ x ∈ Branch ∧ y ∈ Branch ∧ s ↦ x ∈ chain_closure(link_chain) ∧ s ↦ y ∈ chain_closure(link_chain) ⇒ chain(x) = chain(y)

inv74 : ∀ s, d · s ↦ d ∈ intracluster_link ⇔ (s ∈ Leaf ∧ d ∈ Leaf ∧ branch_choice(s) = branch_choice(d)) ∨ (s ∈ Leaf ∧ d ∈ Branch ∧ branch_choice(s) = d) ∨ (s ∈ Branch ∧ d ∈ Leaf ∧ branch_choice(d) = s)

La mise en place de la structuration du réseau par CBL se fait essentiellement par l'échange des messages HELLO. En effet, à la réception d'un HELLO, un noeud Leaf sait s'il reste Leaf, il conserve donc son choix de noeud Branch avant le prochain choix, et un noeud Branch sait s'il reste noeud

Branch et conserve ses choix de noeuds Branch amont et aval. Par conséquent, le maintien de la chaîne s'opère donc uniquement par les HELLO. A ce stade de l'analyse du protocole, nous avons donc opté pour une modélisation complète du format du message HELLO uniquement. Les invariants inv75 à inv81 spécifient le format des données envoyées dans un message HELLO.

inv75 : hello_branch_choice ∈ Hello → NODES

inv76 : hello_branch_up ∈ Hello → NODES

inv77 : hello_branch_down ∈ Hello → NODES

inv78 : hello_branch_status ∈ Hello → BOOL

inv79 : hello_coordinate ∈ Hello → Traffic

inv80 : coordinate ∈ NODES → Traffic

inv81 : local_coordinates ∈ NODES → (NODES → Traffic)

Les événements du modèle initial proposé par [KP16] pour le protocole OLSR à partir du protocole de base (C0, M0) ont été profondément modifier afin de gérer la spécificité des comportements de noeuds Leaf et noeuds Branch dans CBL. Ces modifications s'étendent à tous les niveaux : l'envoi et la réception des messages, la construction et la destruction des liens, la mise à jour de la topologie et des tables de routage. L'événement cbl_forwarding décrit ci-après par exemple ajoute une garde pour vérifier que le noeud qui exécute le protocole a le droit de retransmettre, en d'autres termes qu'il est un noeud Branch (grd7).

cbl_forwarding

⊕ grd7 : (source(msg) ≠ a) ⇒ a ∈ Branch

Les événements cbl_add_link_chain et cbl_remove_link_chain permettent respectivement de créer et supprimer un lien entre deux noeuds Branch. Les événements cbl_add_intracluster_link et cbl_remove_intra_cluster_link permettent respectivement de créer et supprimer un lien entre un noeud Branch avec un de ses noeuds Leaf.

cbl_add_link_chain

ANY

x, y

WHERE

grd1 : x ↦ y ∉ link_chain

grd2 : x ∈ Branch ∧ y ∈ Branch

grd3 : (branch_up(x) = y ∧ branch_down(y) = x) ∨ (branch_down(x) = y ∧ branch_up(y) = x)

THEN

act1 : link_chain := link_chain ∪ {x ↦ y}

END

cbl_remove_link_chain

ANY

x, y

WHERE

grd1 : $x \mapsto y \in link\_chain$

THEN

act1 : $link\_chain := link\_chain \setminus \{x \mapsto y\}$

END

cbl_add_intracluster_link

ANY

x, y

WHERE

grd1 : $x \neq y \wedge x \mapsto y \notin intracluster\_link$

grd2 : $(x \in Leaf \wedge y \in Branch \wedge branch\_choice(x) = y)$

$\vee (x \in Branch \wedge y \in Leaf \wedge branch\_choice(y) = x) \vee (x \in Leaf \wedge y \in Leaf \wedge branch\_choice(x)$

$= branch\_choice(y))$

THEN

act1 : $intracluster\_link := intracluster\_link \cup \{x \mapsto y\}$

END

cbl_remove_intracluster_link

ANY

x, y

WHERE

grd1 : $x \mapsto y \in intracluster\_link$

THEN

act1 : $intracluster\_link := intracluster\_link \setminus \{x \mapsto y\}$

END

L'événement cbl_Leaf_broadcast_hello modélise la diffusion d'un message Hello de la part d'un noeud Leaf. On y note une vérification de la mise à jour des tables locales (grd6), ainsi que le transfert de l'information sur le choix de noeud Branch (act5). L'événement équivalent pour un émetteur noeud Branch, cbl_Branch_broadcast_hello, procède quant à lui à la notification des choix de noeud Branch amont et aval (act5 et act6) par l'émetteur.

Dans les deux cas, le processus démarre par une mise à jour des tables locales de voisins et

l'actualisation de la variable qui permet d'indiquer que l'opération a bien eu lieu : processed_-onehop.

cbl_determine_onehop_neighbours

ANY

s, d

WHERE

grd1 : s ∈ NODES ∧ d ∈ NODES ∧ rt_next_hops(s)(d) = d

grd2 : rt_hops(s)(d) = 1

grd3 : processed_onehop(s) = FALSE

THEN

act1 : node_one_hops(s) := node_one_hops(s) ⩤ {d ↦ TRUE}

act2 : processed_onehop := processed_onehop ⩤ {s ↦ TRUE}

END

cbl_Leaf_broadcast_hello

ANY

s, hello

WHERE

grd1 : s ∈ Leaf

grd2 : hello ∈ Hello

grd3 : hello ∉ broadcast_hello

grd4 : source(hello) = s

grd5 : s ↦ hello ∉ hello_storage

grd6 : processed_onehop(s) = TRUE

THEN

act1 : packet_timetolive := packet_timetolive ∪ {hello ↦ 1}

act2 : broadcast_hello := broadcast_hello ∪ {hello}

act3 : hello_storage := hello_storage ∪ {s ↦ hello}

act4 : hello_onehop_array(hello) := node_one_hops(s)

act5 : hello_branch_choice(hello) := branch_choice(s)

act6 : hello_branch_status(hello) := FALSE

act7 : hello_coordinate(hello) := coordinate(s)

act8 : processed_onehop := processed_onehop ⩤ {s ↦ FALSE}

END

cbl_Branch_broadcast_hello

ANY

s, hello

WHERE

grd1 : s ∈ Branch

grd2 : hello ∈ Hello

grd3 : hello ∉ broadcast_hello

grd4 : source(hello) = s

grd5 : s ↦ hello ∉ hello_storage

grd6 : processed_onehop(s) = TRUE

THEN

act1 : packet_timetolive := packet_timetolive ∪ {hello ↦ 1}

act2 : broadcast_hello := broadcast_hello ∪ {hello}

act3 : hello_storage := hello_storage ∪ {s ↦ hello}

act4 : hello_onehop_array(hello) := node_one_hops(s)

act5 : hello_branch_up(hello) := branch_up(s)

act6 : hello_branch_down(hello) := branch_down(s)

act7 : hello_branch_status(hello) := TRUE

act8 :hello_coordinate(hello) := coordinate(s)

act9 : processed_onehop := processed_onehop ⩤ {s ↦ FALSE}

END

De même que pour l'envoi, plusieurs événements sont implémentés pour modéliser la réception des messages HELLO selon les status (Branch ou Leaf) respectifs de l'émetteur et du récepteur. Le seul traitement commun à tous les cas s'agissant des messages HELLO concerne l'événement cbl_delete_hello qui procède à la suppression du message après son traitement.

cbl_delete_hello

ANY

d, hello

WHERE

grd1 : d ∈ NODES

grd2 : d ↦ hello ∈ hello_storage

grd3 : hello ∈ got_hello

THEN

act1 : deleted_hello := deleted_hello ∪ {hello}

act2 : got_hello := got_hello\{hello}

act3 : hello_storage := hello_storage\{d ↦ hello}

END

cbl_receives_ hello_onehop

ANY

s, d, hello, a

WHERE

grd1 : hello ∈ broadcast_hello\(got_hello ∪ deleted_hello) ∧ source(hello) = s

grd2 : s ≠ d

grd3 : d ↦ hello ∈ hello_storage

grd4 : a ∈ NODES

grd5 : hello_onehop_array(hello)(a) = FALSE

THEN

act1 : rt_hops(d) := rt_hops(d) ⩤ {s ↦ 1}

act2 : rt_next_hops(d) := rt_next_hops(d) ⩤ {s ↦ source(hello)}

act3 : got_hello := got_hello ∪ {hello}

act4 : local_coordinates(d) := local_coordinates(d) ⩤ {s ↦ hello_coordinate(hello)}

END

Le traitement le plus simple survient lorsqu'un noeud reçoit un message HELLO. Il actualise alors les informations sur la position de l'auteur contenues dans le message (cbl_receives_ hello_-onehop). S'il s'agit d'un nouveau voisin (cbl_receive_ hello_updates_two_hops_from_new_neighbor) qui n'est pas symétrique, aucune modification de structure n'est opérée.

cbl_receives_ hello_updates_two_hops_from_new_neighbor

ANY

s, d, hello, a

WHERE

grd1 : hello ∈ broadcast_hello\(got_hello ∪ deleted_hello) ∧ source(hello) = s

grd2 : s ≠ d

grd3 : d ↦ hello ∈ hello_storage

grd4 : a ∈ NODES

grd5 : hello_onehop_array(hello)(a) = TRUE

grd6 : a ≠ d

THEN

act1 : rt_hops(d) := rt_hops(d) ⩤ {s ↦ 1}

act2 : rt_next_hops(d) := rt_next_hops(d) ⩤ {s ↦ source(hello)}

act3 : got_hello := got_hello ∪ {hello}

END

En revanche, s'il s'agit d'un voisin symétrique qui est un noeud Branch et qui n'a ni Branch amont, ni Branch aval, alors le noeud recevant le Hello doit se rattacher à lui (act4).

cbl_receives_ hello_updates_two_hops_from_symetric_neighbor

ANY

s, d, hello, a

WHERE

grd1 : hello ∈ broadcast_hello \ (got_hello ∪ deleted_hello) ∧ source(hello) = s

grd2 : s ≠ d

grd3 : d ↦ hello ∈ hello_storage

grd4 : a ∈ NODES

grd5 : hello_onehop_array(hello)(a) = TRUE

grd6 : a = d ∧ ((s ∈ Branch ∧ branch_up(s) ≠ d) ∨ (s ∈ Branch ∧ branch_down(s) ≠ d))

THEN

act1 : rt_hops(d) := rt_hops(d) ⩤ {s ↦ 1}

act2 : rt_next_hops(d) := rt_next_hops(d) ⩤ {s ↦ source(hello)}

act3 : got_hello := got_hello ∪ {hello}

act4 : branch_choi(d) := s

END

cbl_leaf_receives_hello_onehop_from_leaf

ANY

s, d, hello, a

WHERE

grd1 : hello ∈ broadcast_hello\(got_hello ∪ deleted_hello) ∧ source(hello) = s

grd2 : s ≠ d ∧ s ∈ Leaf ∧ d ∈ Leaf

grd3 : d ↦ hello ∈ hello_storage

grd4 : a ∈ NODES

grd5 : hello_onehop_array(hello)(a) = FALSE

THEN

act1 : rt_hops(d) := rt_hops(d) ⩤ {s ↦ 1}

act2 : rt_next_hops(d) := rt_next_hops(d) ⩤ {s ↦ source(hello)}

act3 : got_hello := got_hello ∪ {hello}

act4 : local_coordinates(d) := local_coordinates(d) ⩤ {s ↦ hello_coordinate(hello)}

END

cbl_leaf_receives_hello_onehop_from_branch

ANY

s, d, hello, a

WHERE

grd1 : hello ∈ broadcast_hello\(got_hello ∪ deleted_hello) ∧ source(hello) = s

grd2 : s ≠ d ∧ s ∈ Branch ∧ d ∈ Leaf

grd3 : d ↦ hello ∈ hello_storage

grd4 : a ∈ NODES

grd5 : hello_onehop_array(hello)(a) = FALSE

THEN

act1 : rt_hops(d) := rt_hops(d) ⩤ {s ↦ 1}

act2 : rt_next_hops(d) := rt_next_hops(d) ⩤ {s ↦ source(hello)}

act3 : got_hello := got_hello ∪ {hello}

act4 : local_coordinates(d) := local_coordinates(d) ⩤ {s ↦ hello_coordinate(hello)}

act5 : branch_choice(d) := s

END

cbl_branch_receives_hello_onehop_from_branchup

ANY

s, d, hello, a

WHERE

grd1 : hello ∈ broadcast_hello\(got_hello ∪ deleted_hello) ∧ source(hello) = s

grd2 : s ≠ d ∧ s ∈ Branch ∧ d ∈ Branch

grd3 : d ↦ hello ∈ hello_storage

grd4 : upstream(coordinate(s))(coordinate(d)) = TRUE

grd5 : a ∈ NODES

grd6 : hello_onehop_array(hello)(a) = FALSE

THEN

act1 : rt_hops(d) := rt_hops(d) ⩤ {s ↦ 1}

act2 : rt_next_hops(d) := rt_next_hops(d) ⩤ {s ↦ source(hello)}

act3 : got_hello := got_hello ∪ {hello}

act4 : local_coordinates(d) := local_coordinates(d) ⩤ {s ↦ hello_coordinate(hello)}

act5 : branch_up(d) := s

act6 : branch_down(s) := d

END

Des événements spécifiques sont introduits pour traiter les messages Hello provenant d'un noeud Leaf par un noeud récepteur qui peut être respectivement Leaf ou Branch. Nous distinguons pour chacune de ces fonctions le cas où l'émetteur a déjà des voisins et le cas inverse.

Par exemple, lorsqu'un noeud Leaf reçoit un message Hello d'un noeud dans son voisinage qui est déjà Branch mais qui n'a pas de voisins, il doit tenter de s'y rattacher (cbl_leaf_receives_hello_-onehop_from_branch).

Lorsqu'un noeud Branch reçoit un message Hello provenant d'un noeud Branch situé en amont dans le trafic et qui n'a pas de voisin, il le choisit comme noeud Branch amont (cbl_branch_-receives_hello_onehop_from_branchup). Ce dernier doit donc logiquement avoir comme noeud Branch aval son unique voisin qui est le réceteur du HELLO.

Une procédure analogue permet de sélectionner comme noeud Branch aval un noeud Branch en aval sans voisin qui émettrait un Hello (cbl_branch_receives_hello_onehop_from_branchdown).

cbl_branch_receives_hello_onehop_from_branchdown

ANY

s, d, hello, a

WHERE

grd1 : hello ∈ broadcast_hello\(got_hello ∪ deleted_hello) ∧ source(hello) = s

grd2 : s ≠ d ∧ s ∈ Branch ∧ d ∈ Branch

grd3 : d ↦ hello ∈ hello_storage

grd4 : downstream(coordinate(s))(coordinate(d)) = TRUE

grd5 : a ∈ NODES

grd6 : hello_onehop_array(hello)(a) = FALSE

THEN

act1 : rt_hops(d) := rt_hops(d) ⩤ {s ↦ 1}

act2 : rt_next_hops(d) := rt_next_hops(d) ⩤ {s ↦ source(hello)}

act3 : got_hello := got_hello ∪ {hello}

act4 : local_coordinates(d) := local_coordinates(d) ⩤ {s ↦ hello_coordinate(hello)}

act5 : branch_down(d) := s

act6 : branch_up(s) := d

END

Des événements analogues aux précédents sont également introduits pour le cas d'un émetteur d'Hello qui a des voisins, dont éventuellement le noeud récepteur du HELLO.

Ainsi, lorsqu'un noeud Branch reçoit un message Hello provenant d'un noeud Branch situé en amont dans le trafic et qui a des voisins, il doit le choisir comme noeud Branch amont (cbl_-receive_ hello_updates_two_hops_from_branchup) pour tenter de conserver une cohérence dans la chaîne.

Une procédure analogue permet de sélectionner comme noeud Branch aval un noeud Branch en aval qui émettrait un Hello (cbl_receive_ hello_updates_two_hops_from_branchdown) dans lequel il aurait désigné le noeud récepteur actuel comme noeud Branch amont.

Il convient de noter que toutes les mises à jour, notamment en matière de choix de noeud Branch, noeud Branch amont, noeud Branch aval et de statut de noeud Branch effectués dans les événements sont des décisions locales des noeuds pour être en cohérence avec les propriétés attendues du protocole. L'application effective de ces choix s'opérerait une fois que tous les noeuds impliqués auront reçu les Hello les en notifiant et accepté les changements éventuels induits.

cbl_receives_ hello_updates_two_hops_from_branchup

ANY

s, d, hello, a

WHERE

grd1 : hello ∈ broadcast_hello\(got_hello ∪ deleted_hello) ∧ source(hello) = s

grd2 : s ≠ d

grd3 : d ↦ hello ∈ hello_storage

grd4 : a ∈ NODES

grd5 : hello_onehop_array(hello)(a) = TRUE

grd6 : a = d ∧ s ∈ Branch ∧ branch_down(s) = d

grd7 : upstream(coordinate(s))(coordinate(d)) = TRUE

THEN

act1 : rt_hops(d) := rt_hops(d) ⩤ {s ↦ 1}

act2 : rt_next_hops(d) := rt_next_hops(d) ⩤ {s ↦ source(hello)}

act3 : got_hello := got_hello ∪ {hello}

act4 : branch_up(d) := s

END

cbl_receives_ hello_updates_two_hops_from_branchdown

ANY

s, d, hello, a

WHERE

grd1 : hello ∈ broadcast_hello\(got_hello ∪ deleted_hello) ∧ source(hello) = s

grd2 : s ≠ d

grd3 : d ↦ hello ∈ hello_storage

grd4 : a ∈ NODES

grd5 : hello_onehop_array(hello)(a) = TRUE

grd6 : a = d ∧ s ∈ Branch ∧ branch_up(s) = d

grd7 : downstream(coordinate(s))(coordinate(d)) = TRUE

THEN

act1 : rt_hops(d) := rt_hops(d) ⩤ {s ↦ 1}

act2 : rt_next_hops(d) := rt_next_hops(d) ⩤ {s ↦ source(hello)}

act3 : got_hello := got_hello ∪ {hello}

act4 : branch_down(d) := s

END

cbl_broadcast_tc

ANY

s, tc

WHERE

grd1 : s ∈ Branch

grd2 : tc ∈ Tc

grd3 : tc ∉ broadcast_tc

grd4 : source(tc) = s

grd5 : s ↦ tc ∉ tc_storage

grd6 :branch_status(s) = TRUE

THEN

act1 : tc_sender := tc_sender $\lhd$ {tc $\mapsto$ s}

act2 : packet_timetolive := packet_timetolive $\cup$ {tc $\mapsto$ tc_ttl}

act3 :packet_hops := packet_hops $\cup$ {tc $\mapsto$ 0}

act4 : tc_storage := tc_storage $\cup$ {s $\mapsto$ tc}

act5 : broadcast_tc := broadcast_tc $\cup$ {tc}

act6 : tc_sqn(tc) := local_sqn(s)

act7 : local_sqn := local_sqn $\lhd$ {s $\mapsto$ local_sqn(s) + 1}

END

cbl_receive_tc

ANY

s, d, tc

WHERE

grd1 : tc $\in$ broadcast_tc\(got_tc $\cup$ deleted_tc)

grd2 : s $\in$ Branch

grd3 : source(tc) = s $\wedge$ source(tc) $\neq$ d

grd4 : s $\neq$ d

grd5 : d $\mapsto$ tc $\in$ tc_storage

grd6 : tc_sqn(tc) > (rt_sqn(d)(s))

THEN

act1 : rt_hops(d) := rt_hops(d) $\lhd$ {s $\mapsto$ packet_hops(tc) + 1}

act2 : rt_next_hops(d) := rt_next_hops(d) $\lhd$ {s $\mapsto$ tc_sender(tc)}

act3 : got_tc := got_tc $\cup$ {tc}

act4 : rt_sqn(d) := rt_sqn(d) $\lhd$ {s $\mapsto$ tc_sqn(tc)}

END

L'événement cbl_broadcast_tc matérialise la diffusion d'un message TC dans le réseau. Le nœud source est donc obligatoirement un noeud Branch (grd1). Afin de permettre la gestion de l'obsolescence des informations de topologie, comme dans OLSR, à chaque fois qu'un noeud émet un TC, il met à jour son local sequence number (act4).

Lors de la réception d'un message TC (cbl_receive_tc), le noeud récepteur vérifie que le TC n'est pas obsolète par rapport aux précédents TC reçus de la même source, puis rafraîchit sa table de routage. Lors de la retransmission d'un message TC, le noeud retransmetteur doit être un noeud Branch conformément aux règles de CBL. De plus, il faut que les possibilités de retransmission du paquet par rapport au réseau soient encore de mise (packet_timetolive(tc)>1). L'événement cbl_-remove_outdated_tc supprime les messages TC obsolètes au niveau de chaque noeuds.

cbl_forward_tc

ANY

a, b, tc

WHERE

grd1 : $tc \in got\_tc \land source(tc) \neq a$

grd2 : $b \in NODES \land a \in Branch \land a \neq b$

grd3 : $a \mapsto tc \in tc\_storage \land b \mapsto tc \notin tc\_storage$

grd4 : $packet\_timetolive(tc) > 1$

THEN

act1 : $tc\_storage := (tc\_storage \backslash a \mapsto tc) \cup \{b \mapsto tc\}$

act2 : $packet\_hops := packet\_hops \lhd \{tc \mapsto packet\_hops(tc) + 1\}$

act3 : $tc\_sender := tc\_sender \lhd \{tc \mapsto a\}$

act4 : $packet\_timetolive := packet\_timetolive \lhd \{tc \mapsto packet\_timetolive(tc)\text{-}1\}$

END

cbl_remove_outdated_tc

ANY

tc, a, s

WHERE

grd1 : $a \in NODES$

grd2 : $tc \in got\_tc$ not

grd3 : $a \mapsto tc \in tc\_storage$

grd4 : $tc\_sqn(tc) \leq rt\_sqn(a)(s) \lor packet\_timetolive(tc) < 1$

grd5 : s=source(tc)

THEN

act1 : $broadcast\_tc := broadcast\_tc \backslash \{tc\}$

act2 : $tc\_storage := tc\_storage \backslash \{a \mapsto tc\}$

act3 : $got\_tc := got\_tc \backslash \{tc\}$

END

3.4 Synthèse

Grâce au concept de raffinement offert par la modélisation formelle Event-B, nous avons raffiné le modèle d'un protocole de base pour obtenir le modèle spécifique du protocole OLSR implémentant la méthode de clustering CBL. Pour ce faire, nous avons ajouté à un modèle d'OLSR obtenu en synthétisant les différents raffinements proposés par [KP16] et en y ajoutant les proprié-

tés attendues de la méthode de clustering CBL. Tous les invariants introduits ont pu être déchargés automatiquement.

TABLEAU 3.2 – Proof statistics for CBL-OLSR

Model	Number of Proof Obligations	Automatically Discharged	Interactively Discharged
C0	4	4	0
M0	62	62	0
C1	2	2	0
M1	451	451	0
Total	519	519	0

3.5 Modélisation formelle de communications sécurisées

Nous allons maintenant nous intéresser à la modélisation de propriétés liées à la sécurité dans le protocole de communication. Pour évaluer la sécurité, on s'intéresse usuellement aux cinq principes présentés ci-après.

3.5.1 Rappel sur les principes de sécurité

La sécurité informatique est un terme qui désigne l'ensemble des moyens et technologies pour la protection des systèmes d'information. Le but de la sécurité informatique est d'assurer la disponibilité, la confidentialité et l'intégrité des biens et des actifs d'information. L' authentification et la non-répudiation s'ajoutent à ces trois principes.

Un synonyme souvent utilisé est cybersecurité. Ce terme représente une sous-classe de la sécurité informatique qui ne dépend que de la technique. Dans cette sous-classe , nous vérifions souvent les qualités de résilience, de robustesse et de réactivité qu'une technique ou un système d'information doit posséder pour faire face aux attaques(cyber attaques) visant à menacer son fonctionnement correct et ses performances.

Des éléments techniques, organisationnels, juridiques et humains sont impliqués dans la sécurité informatique. Les experts en sécurité informatique ont tendance à dire qu'il n'existe pas de système sûr à 100%. Selon les menaces, vous pouvez faire la distinction entre sécurité logique et sécurité physique. Pour évaluer la sécurité d'un système informatique, il est généralement nécessaire d'identifier les menaces et les vulnérabilités, étape nécessaire dans mise en place de mesures de protection face aux éventuelles attaques.

3.5.2 Définition des concepts de la sécurité

Comme présenté dans la Figure 3.7, les quatres concepts essentiels qui devront être évalués lors de la définition de la sécurité sont :

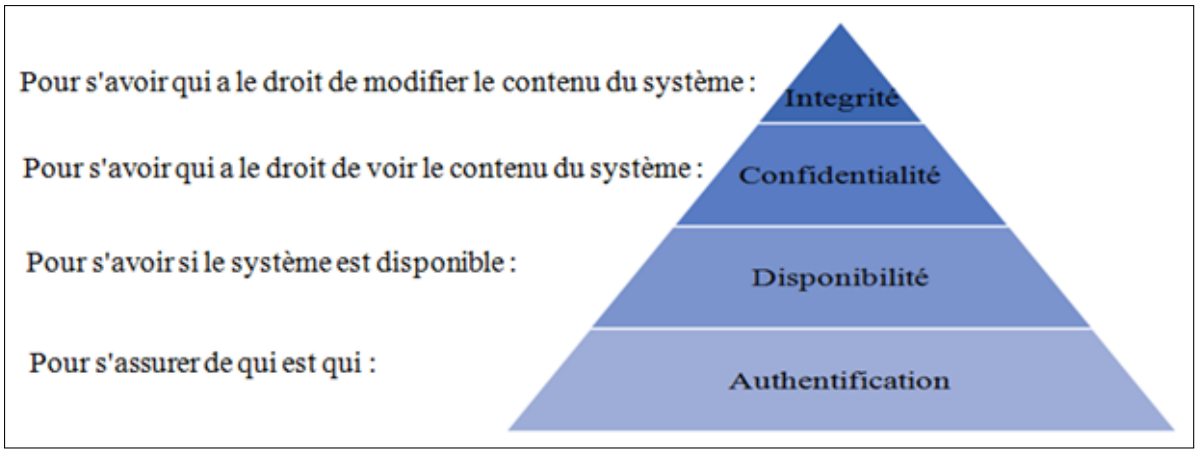

FIGURE 3.7 – Pyramide des proprietés de sécurité informatique

La disponibilité

La disponibilité d'un système reflète sa capacité à répondre à certaines exigences à un moment précis ou dans un délai convenu. Aucun équipement informatique ne peut fonctionner parfaitement en permanence. Donc la stabilité d'un système est mesuré par rapport à sa disponibilité. Plus la probabilité de disponibilité est élevée, plus le système est considéré comme stable. La disponibilité du SI est donc une notion basée sur des probabilités afin de calculer la fiabilité des divers éléments de l'infrastructure informatique.

La confidentialité

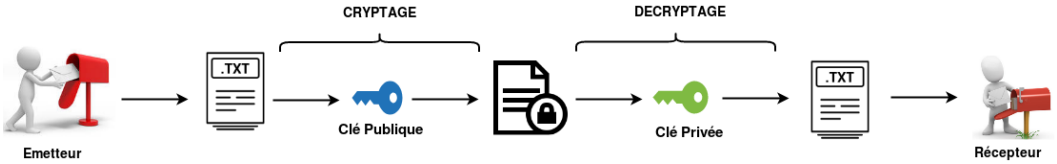

FIGURE 3.8 – Aperçu du modèle global

La confidentialité est la protection des données échangées entre un émetteur et un ou plusieurs destinataires contre des tiers non autorisés. La nécessité d'assurer la propriété de la confidentialité est particulièrement intéressante lorsque le système de communication utilisé est intrinsèquement peu sûr ou public comme internet.

Des mécanismes de cryptage des communications sont mis en place pour garantir la confidentialité. L' authentification de l'accès au système est un autre mécanisme permettant d'accroître la confidentialité, car il est potentiellement capable de supprimer l'accès des tiers non autorisés ou non authentifiés au système, et donc aux données qu'il héberge.

La figure 3.8 présente un exemple dans la cryptographie asymétrique, qui assure la confidentialité en permettant un cryptage avec la clé publique du destinataire et un décryptage avec sa clé privée.

L'intégrité

Le terme intégrité des données signifie la protection des données et des informations contre les modifications du contenu effectuées par un tiers. La notion d'intégrité comporte la possibilité de vérifier avec une certitude absolue si une donnée ou une information est restée intacte, c'est-à-dire qu'elle n'a pas changé de contenu pendant sa transmission ou son stockage. Dans un système garantissant l'intégrité, l'action d'un tiers modifiant le contenu des informations échangées entre l'expéditeur et le destinataire est alors détectée. Plus l'intégrité des données est grande, la possibilité d'une lecture ou écriture exacte augmente. L'intégrité des données échangées est très souvent associée à l'authentification. Il existe de nombreux protocoles, mécanismes et algorithmes de chiffrement qui sont en mesure d'assurer ces deux propriétés de sécurité informatique.

L'authentification

L'authentification est le processus de confirmation de l'identité d'une entité. Lors d'une connexion ou communication sur Internet ou dans un réseau ad hoc, il est important que l'utilisateur soit identifié de manière unique afin d'accéder à ses services dans le réseau. De la même manière, il est également essentiel de connaître l'identité de ceux qui se trouvent de l'autre côté de la ligne de communication et de s'assurer que l'interlocuteur avec lequel ils échangent des informations est bien celui qu'il prétend être et non un intrus. L'authentification est différente de l'identification (la détermination qu'un individu est connu ou non du système) et de l'autorisation (l'attribution à un utilisateur du droit d'accéder à des ressources spécifiques du système, en fonction de son identité). L'authentification peut être garantie à travers des protocoles de sécurité ou de cryptographie, notamment avec des outils tels que les certificats numériques. Un certificat numérique est un document électronique qui atteste de l'association unique entre une clé publique et l'identité d'un sujet (une personne, une entreprise, un ordinateur, etc.) qui prétend l'utiliser dans le cadre de procédures de chiffrement asymétrique ou de signature numérique.

3.5.3 Deuxième raffinement : Contexte 2

Dans cette section, nous présentons la mise en oeuvre de propriétés relatives à la sécurité dans le modèle formel du protocole CBL-OLSR modélisé précédement. Le modèle global ainsi obtenu est illustré dans la figure 3.9 où l'on retrouve les trois couches d'abstraction contenant les trois machines et les trois contextes correspondants.

Nous avons opté pour la mise en oeuvre d'une transaction dans laquelle se fait un transfert sécurisé, d'après les modèles proposés par [BM10]. Nous introduisons donc le Contexte 2 qui étend le Contexte 1 en y ajoutant un nouvel ensemble : Key, l'ensemble des clés.

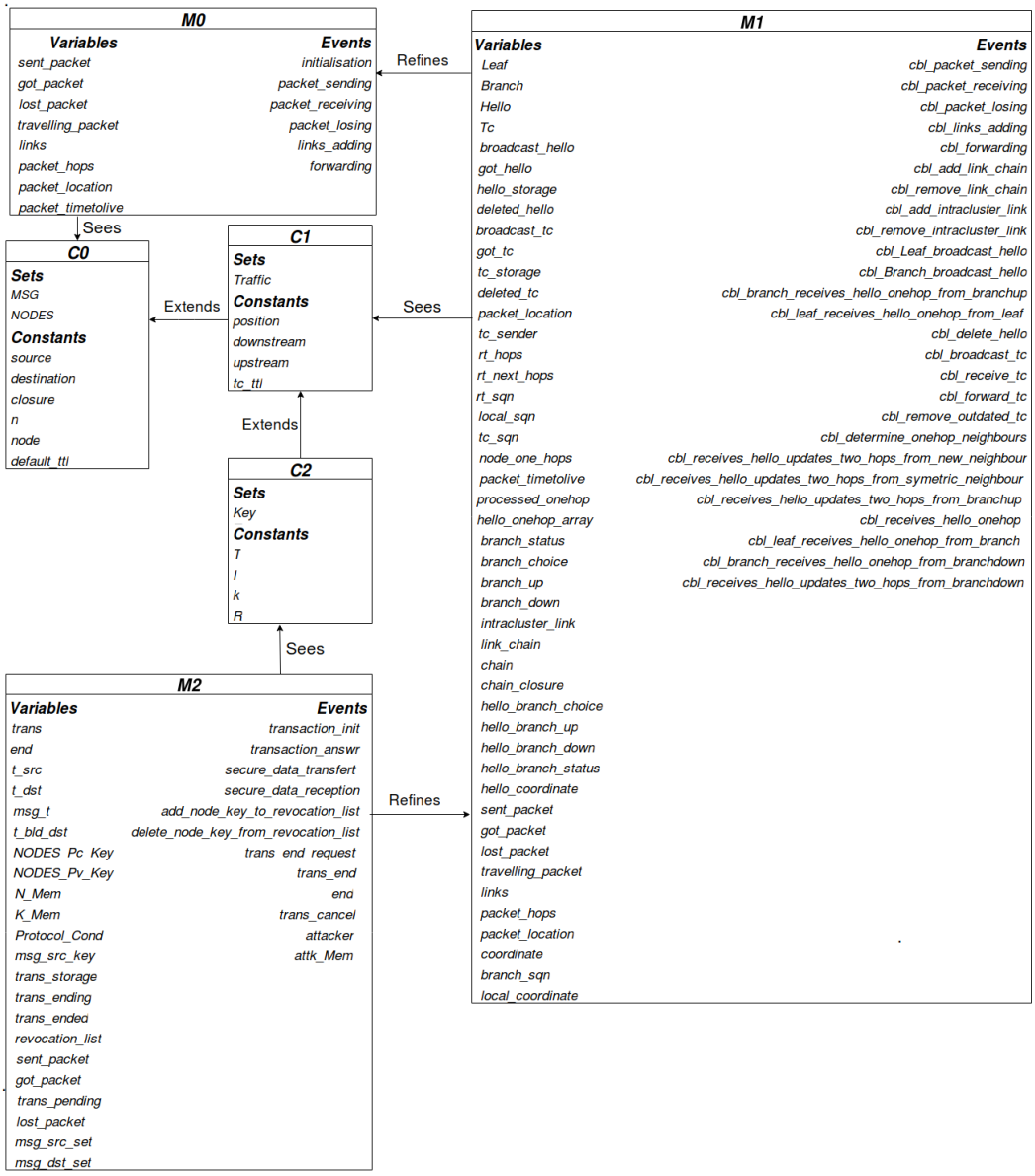

FIGURE 3.9 – Schéma du modèle de CBL-OLSR avec transactions et transferts sécurisés

axm1 : I ∈ NODES

axm2 : R ∈ NODES

axm3 : finite(Key)

axm4 : Key ≠

axm5 : card(Key) = k

axm6 : k > 1

axm7 : T ∈ NODES → NODES

3.5.4 Deuxième raffinement : Machine 3

Les différents échanges entre un noeud source et un noeud destinataire pour mettre en oeuvre une transaction et réaliser un transfert sécurisé sont décrits dans la figure 3.10. Le noeud source A

initie une transaction qu'il peut annuler après une durée TIMEOUT. Une fois la transaction confirmée par le destinataire B, le transfert sécurisé des données peut avoir lieu. Lorsqu'il n'y a plus de données à transférer, le noeud A peut notifier la fin de la transaction, et B peut confirmer.

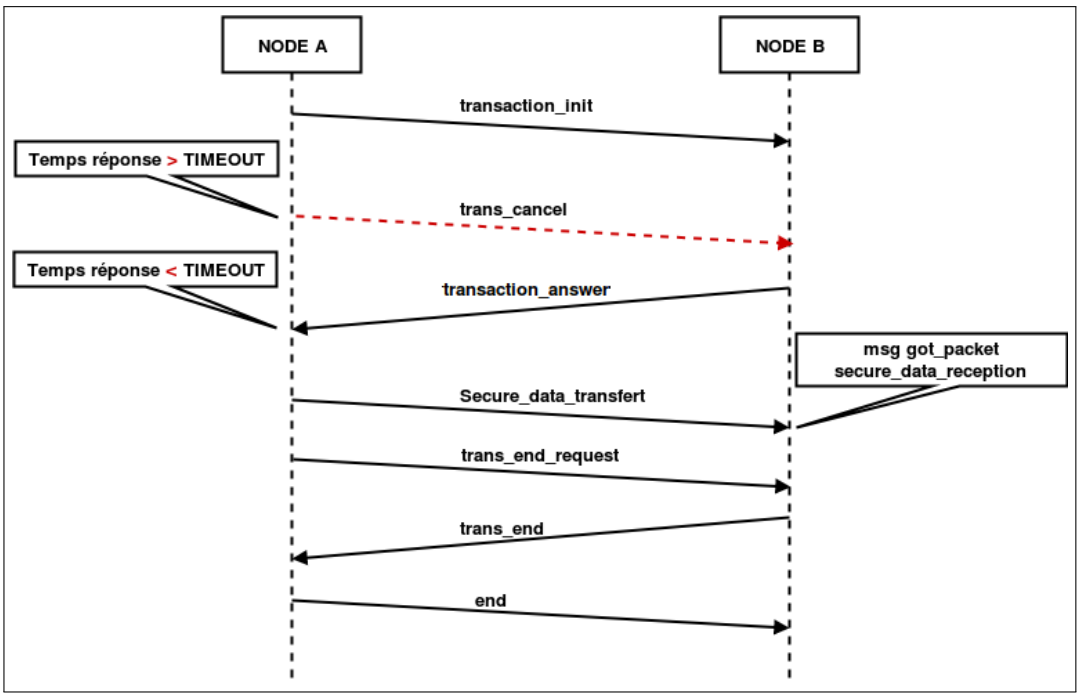

FIGURE 3.10 – Modélisation de transaction et transfert sécurisé

Les invariants inv1 à inv7 introduisent les définitions de transaction et les applications de l'ensemble de transations vers les ensembles de noeuds (NODES) et de messages (MSG via son sous-ensemble sent_packet). L'invariant inv8 établit qu'à la fin d'une transaction le destinataire effectif de la transaction doit être le destinataire de confiance (authentifié) pour cette transaction, autrement dit qu'un attaquant ne se soit pas substitué à lui dans l'entretemps. Les invariants inv9 et inv10 introduisent les ensembles de clés publiques et clés privées, respectivement. Les invariants inv11 et inv12 fixent les conditions qui permettent de vérifier l'intégrité des messages ainsi que de la transaction. L'invariant inv13 modélise une mémoire des associations entre noeuds et transactions qui pourrait être exploitée par un attaquant.

L'invariant inv14 modélise quant à lui une mémoire associant les noeuds avec leurs clés. Les invariants inv15 à inv20 modélisent les ensembles et applications qui permettent de suivre une transaction en cours.

inv1 : trans_pending ⊆ T

inv2 : trans ⊆ trans_pending

inv3 : end ⊆ trans

inv4 : t_src ∈ trans → NODES

inv5 : t_dst ∈ trans → NODES

inv6 : msg_t ∈ sent_packet → T

inv7 : t_bld_dst ∈ trans → NODES

inv8 : ∀ t·t ∈ end ⇒ t_dst(t) = t_bld_dst(t)

inv9 : NODES_Pc_Key ∈ NODES ⤖ Key

inv10 : NODES_Pv_Key ∈ NODES ⤖ Key

inv11 : Protocol_Cond ∈ MSG → BOOL

inv12 : ∀t, msg, s, d·t ∈ trans∧ s ∈ NODES∧ d ∈ NODES∧ msg ∈ sent_packet∧ t_src(t) = s ∧ t_bld_dst(t) = d∧ msg_t(msg) = t ∧ Protocol_Cond(msg) = TRUE ⇒ source(msg) = t_bld_dst(t)

inv13 : N_Mem ∈ NODES → (T)

inv14 : K_Mem ∈ NODES → (Key)

inv15 : trans_storage ∈ NODES ↔ trans

inv16 : trans_ending ⊆ trans

inv17 : trans_ended ⊆ trans

inv18 : revocation_list ∈ NODES ↔ (NODES ↔ Key)

inv19 : msg_src_key ∈ MSG ↔ Key

inv20 : msg_src_set ∈ MSG ↔ NODES

inv21 : msg_dst_set ∈ MSG ↔ NODES

Le noeud source de la transaction **transaction_init** initie la transaction en l'ajoutant à l'ensemble trans_pending et fixe les valeurs des variables t_src à lui-même et t_bld_dst au noeud avec lequel il veut communiquer. Il transmet sa clé publique dans le message. Il calcule une empreinte sur les données du message qu'il prend le soin de signer avec sa clé privée. Lorsque le destinataire recevra le message, il pourra utiliser la clé publique reçue pour vérifier la signature, puis comparer l'empreinte au message en clair pour authentifier par Challenge la source et plus tard pour envoyer à cette dernière des messages cryptés.

Dans l'événement transaction_answr, la source et la destination sont inversées par rapport à l'événement transaction_init, étant donné que le message dans cet événement a pour source (s) le destinataire du message dans l'événement précédent (donc destinataire de la transaction initiée). Avant d'accepter la transaction, le destinaire de la transaction (s) vérifie que la source (d) n'est

pas répertoriée dans la liste de révocations (grd8). Il vérifie également que la transaction a bien été initiée (grd2) puis accepte la transaction en l'ajoutant dans l'ensemble de transactions actives (act1). Il transmet sa clé publique dans le message. Il calcule une empreinte sur les données du message qu'il prend le soin de signer avec sa clé privée. Lorsque le destinataire (source de la transaction) recevra le message, il pourra utiliser la clé publique reçue pour vérifier la signature, puis comparer l'empreinte au message en clair pour authentifier par Challenge la source du message (destinataire de la transaction) et plus tard pour envoyer à cette dernière des messages cryptés.

Une fois la transaction établie, les noeuds sont authentifiés et peuvent procéder à des transferts sécurisés, avec la possibilité de crypter les données qui le nécessitent et de vérifier l'intégrité mes messages reçus via les mécanismes fournis par les paires de clés publique et privée. Pour renforcer la confiance, la transmission de la clé publique se fait via la transmission d'un certificat signé par une autorité de certification reputée, connue par tous les noeuds et dont la clé publique est accessible à tous (pour pouvoir vérifier sa signature dans les certificats).

transaction_init

ANY

t , s, d, msg, ks

WHERE

grd1 : $t \in T \setminus trans\_pending$

grd2 : $s \in NODES \wedge d \in NODES \wedge s \neq d$

grd3 : $ks \in Key \wedge NODES\_Pc\_Key(s)=ks$

grd4 : $msg \notin sent\_packet$

grd5 : source(msg) = s

grd6 : destination(msg) = d

THEN

act1 : msg_t(msg) := t

act2 : $trans\_pending := trans\_pending \cup \{t\}$

act3 : t_src(t) := s

act4 : t_bld_dst(t) := d

act5 : msg_src_key(msg) := ks

act6 : $sent\_packet := sent\_packet \cup \{msg\}$

END

transaction_answr

ANY

t, s, d, msg, kd, ks

WHERE

grd1 : s ∈ NODES ∧ d ∈ NODES

grd2 : t ∈ trans_pending \ trans

grd3 : ks ∈ Key ∧ NODES_Pc_Key(s) = ks

grd4 : kd ∈ Key ∧ NODES_Pc_Key(d) = kd

grd5 : source(msg) = s

grd6 : destination(msg) = d

grd7 : msg ∉ sent_packet

grd8 : d ↦ kd ∉ revocation_list(s)

THEN

act1 : trans := trans ∪ {t}

act2 : msg_t(msg) := t

act3 : msg_src_key(msg) := ks

act4 : sent_packet := sent_packet ∪ {msg}

END

secure_data_transfer

ANY

t, s, d, msg

WHERE

WHERE

grd1 : s ≠ d ∧ source(msg) = s ∧ destination(msg) = d ∧ msg_t(msg) = t

grd2 : t ∈ trans \ end

grd3 : msg ∈ MSG ∧ msg ∉ sent_packet

THEN

act1 : sent_packet := sent_packet ∪ {msg}

END

secure_data_reception

ANY

t, s, d, msg

WHERE

grd1 : $s \neq d \wedge source(msg) = s \wedge destination(msg) = d \wedge msg\_t(msg) = t$

grd2 : $t \in trans \setminus end$

grd3 : $msg \in sent\_packet \setminus (got\_packet \cup lost\_packet)$

THEN

act1 : $got\_packet := got\_packet \cup \{msg\}$

END

add_node_key_to_revocation_list

ANY

a, b, ka

WHERE

grd1 : $a \in NODES \wedge b \in NODES$

grd3 : $ka \in Key$

grd6 : $a \mapsto ka \notin revocation\_list(b)$

THEN

act1 : $revocation\_list(b) := revocation\_list(b) \cup \{a \mapsto ka\}$

END

delete_node_key_from_revocation_list

ANY

a, b, ka

WHERE

grd1 : $a \in NODES \wedge b \in NODES$

grd3 : $ka \in Key$

grd4 : $a \mapsto ka \in revocation\_list(b)$

THEN

act1 : $revocation\_list(b) := revocation\_list(b) \setminus \{a \mapsto ka\}$

END

Les événements **add_node_key_to_revocation_list** et **delete_node_key_from_revocation_-list** permettent respectivement d'ajouter un couple (NODES, Key) dans une liste de révocation selon que le noeud a été banni, que sa clé a été compromise ou que son certificat a expiré, etc, ou de supprimer ces informations de la liste de révocation pour des raisons diverses.

trans_end_request

ANY

t

WHERE

grd1 : $t \in trans \setminus (trans\_ending \cup trans\_ended)$

THEN

act1 : $trans\_ending := trans\_ending \cup \{t\}$

END

trans_end

ANY

t

WHERE

grd1 : $t \in trans \land t \in trans\_ending$

grd2 : $t \notin trans\_ended$

THEN

act1 : $trans\_ended := trans\_ended \cup \{t\}$

END

end

ANY

t, s, d, msg

WHERE

grd1 : $t \in trans\ \ end$

grd2 : $s \in NODES \land d \in NODES \land s \neq d$

grd3 : $t\_src(t) = s \land t\_dst(t) = d$

grd4 : $msg \notin sent\_packet$

grd5 : $msg\_t(msg) = t$

grd6 : $(source(msg) = s \land destination(msg) = d)\ \ (source(msg) = d \land destination(msg) = s)$

grd7 : $Protocol\_Cond(msg) = TRUE$

THEN

act1 : $end := end \cup \{t\}$

act2 : $sent\_packet := sent\_packet \cup \{msg\}$

END

La fermeture d'une transaction doit être une action unilatérale : elle peut partir aussi bien de la source de la transaction que de la destination de la transaction. Mais une fois déclenchée, celui

qui a décidé ferme la transaction et envoie un message pour en notifier son correspondant. Après confirmation de ce dernier, l'initiateur procède à la fermeture définitive de la transaction.

L'événement **trans_cancel** permet d'interrompre une transaction qui a été initiée, mais qui n'a pas encore été acceptée. Cela se fait automatiquement lorsque le timeout survient, mais aussi à l'initiative uniquement du noeud qui l'a initiée.

trans_cancel

ANY

t, s, d, msg

WHERE

grd1 : $t \in trans\_pending \setminus trans$

grd2 : $s \in NODES \land d \in NODES \land s \neq d$

grd3 : $t\_src(t) = s \land t\_dst(t) = d$

grd4 : $msg \notin sent\_packet$

grd5 : $msg\_t(msg) = t$

grd6 : $source(msg) = s \land destination(msg) = d$

THEN

act1 : $trans\_pending := trans\_pending \setminus \{t\}$

act2 : $sent\_packet := sent\_packet \cup \{msg\}$

END

attacker

ANY

t, s, msg

WHERE

grd1 : $t \in T$

grd2 : $s \in NODES \land d \in NODES \land s \neq d$

grd3 : $msg \in MSG \ sent\_packet$

grd4 : $s \neq I$

grd5 : $t \in N\_Mem(s)$

THEN

act5 : $msg\_src\_set(msg) := I$

act2 : $msg\_dst\_set(msg) := d$

act3 : $msg\_src\_key(msg) := NODES\_Pc\_Key(s)$

act4 : $msg\_t(msg) := t$

act1 : $sent\_packet := sent\_packet \cup \{msg\}$

END

Dans l'évement attacker, l'attaquant envoie des messages à des agents choisis au hasard pour essayer de les induire en erreur sur son identité. Il utilise une transaction qui a été créée par un noeud source s légitime. Donc la source du message est l'attaquant, mais il utilise la clé de s pour crypter le message afin de tromper la destination. La destination est censée vérifier que l'auteur du message ne correspond ni à la clé, ni à la transaction en cours.

Dans l'évenement attack_Mem : la source s a envoyé un message et on va ajouter dans la mémoire de s la transaction. C'est ce qui permet à l'attaquant s'il attaque la mémoire de s, d'aller récupérer les transactions que s utilisait.

attk_Mem

ANY

t , s , msg

WHERE

grd1 : s ∈ NODES

grd2 : NODES_Pv_Key(s) ∈ K_Mem(s)

grd3 : msg ∈ sent_packet

grd4 : msg_t(msg) = t

grd5 : msg_src_key(msg) = NODES_Pc_Key(s)

THEN

act1 : N_Mem(s) := N_Mem(s) ∪ {t}

END

3.6 Statistiques de preuve

Pour prouver la validité de certaines propriétés du protocole CBL, nous avons utilisé l'outil de la plate-forme Rodin pour générer des obligations de preuve pour les trois modèles. Le résumé des statistiques de preuve est présenté dans le Tableau 2.1, qui comprend le nombre d'obligations de preuve générées par la plate-forme Rodin, le nombre d'obligations de preuve déchargées automatiquement et le nombre d'obligations de preuve déchargées de manière interactive.

Comme le montre la Figure 3.9, la première couche décrit le comportement abstrait d'un protocole de routage. Le raffinement suivant améliore la machine abstraite en analysant les spécificités du protocole CBL-OLSR. Et le dernier raffinement augmente la sécurité des transferts de données dans le réseau. Nous pouvons remarquer que les propriétés de notre modèle sont toutes déchargées automatiquement.

Dans cette partie du chapitre, notre modèle formel nous a permis de décrire comment des propriétés de sécurité pouvaient être introduites dans CBL-OLSR. Le tableau suivant présente les statistiques du modèle global par différents niveaux de raffinement.

L'objectif de la thèse n'étant pas de proposer un protocole sécurisé, mais de montrer comment la modélisation formelle peut être utilisée pour concevoir des protocoles sécurisés dans une démarche de conception et d'évaluation où elle est associée à la modélisation et simulation DEVS, nous avons choisi d'appliquer le modèle proposé par [BM10]. Nous avons choisi d'illustrer simplement le mécanisme de transfert sécurisé entre deux véhicules tels qu'il pourrait être mis en place dans un protocole de communication. Notre choix a porté sur un mécanisme simple de transfert sécurisé qui implique une phase de mise en place d'une transaction, une phase de transfert de données et une phase de clôture de la transaction. Les modèles de l'attaquant et d'une attaque, proposés par [BM10] sont adaptés et repris pour illustrer.

TABLEAU 3.3 – Proof Statistics for secured CBL-OLSR.

Model	Number of Proof Obligations	Automatically Discharged	Interactively Discharged
C0	4	4	0
M0	62	62	0
C1	2	2	0
M1	451	451	0
C2	1	1	0
M2	84	84	0
Total	604	604	0

Chapitre 4

Projection de propriétés formelles vers la simulation DEVS

Sommaire

4.1 Introduction

Ce chapitre est une tentative de formalisation d'une approche basée sur la multi-modélisation combinant modélisation formelle et modélisation DEVS pour simuler des protocoles de communications pour les transports, avec une application aux réseaux ad hoc de véhicules. Dans le chapitre 2, nous avons montré une première modélisation et simulation à événements discrets DEVS du protocole OLSR implémentant la méthode de clustering CBL. Ensuite la modélisation formelle avec Event-B a permis de vérifier et de prouver des propriétés de sûreté et de sécurité de ce protocole au chapitre 3.

Dans ce chapitre, nous explorons et présentons une démarche pour exporter vers une modélisation DEVS non seulement le modèle du protocole et du réseau ad hoc, mais également toutes les connaissances que la modélisation formelle a permis de générer ces modèles afin de pouvoir les prendre en compte dans lors de la simulation.

4.1.1 Pourquoi projeter un modèle formel dans une modélisation DEVS?

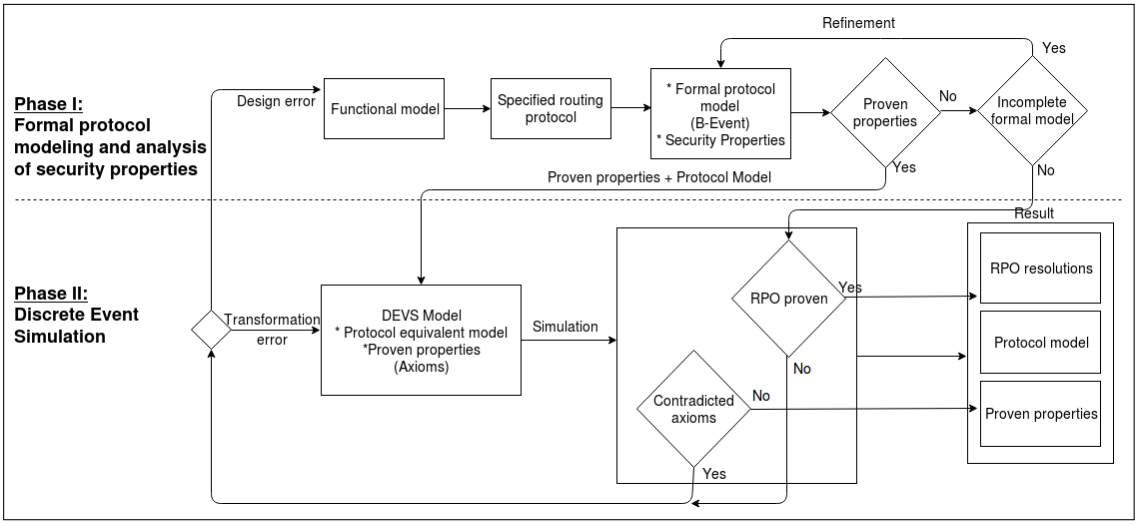

FIGURE 4.1 – Principales phases de projection de modèles Event-B vers DEVS[CSR17]

Deux problèmes persistent aux approches formelles et plaident en faveur d'une approche combinée avec la simulation. Le premier concerne l'évaluation conjointe des composants qui s'adaptent à la modélisation formelle avec les autres composants de l'ITS qui ne s'y prêtent pas. Des outils tels qu' Event-B permettent d'évaluer uniquement des modèles formels, alors que les outils de simulation offrent la possibilité de connecter des modèles et des matériels hétérogènes dans un seul processus d'évaluation. Le second est lié au fait que, bien que permettant d'animer un nombre limité d'objets afin de vérifier le comportement d'un système en présence d' interactions, les outils formels n'offrent pas cette possibilité à grande échelle.

4.1.2 Approche proposée

L'approche que nous proposons consiste à tirer partie des avantages à la fois de la simulation à événements discrets et des méthodes formelles pour augmenterdes composants qui s'adaptent à la modélisation formelle, ainsi que sur l'ensemble du système de transport intelligent. En l'occurrence, le composant considéré pour illustrer l'approche est une méthode de clustering pour les réseaux ad hoc, et donc également le protocole auquel elle est intégrée, et le système est un réseau ad hoc de véhicules. Comme illustré sur la Figure 4.1, cette approche se décline en deux phases. Une première phase consiste à modéliser le protocole de routage à l'aide d'un outil formel tel qu'Event-B et d'obtenir par raffinement un certain nombre de propriétés prouvées et une liste de propriétés dont la preuve dépend d'un processus interactif impliquant un expert (OPR - Obligations de preuve résiduelles). Le formalisme B offre une notation de modélisation basée sur la théorie des ensembles qui s'adapte très bien aux protocoles de réseaux. L'utilisation du raffinement dans B pour représenter les systèmes à différents niveaux d'abstraction et l'utilisation de preuves mathématiques pour vérifier la cohérence entre les niveaux de raffinement permettent une conception incrémentale maîtrisée. L'extension Event-B facilite la gestion des aspects dynamiques des réseaux de communication (mobilité, transmission des messages, etc). Néanmoins, le formalisme B et son extension Event-B se focalisent davantage sur les états du système et les conditions de changements dans ces états sans se préoccuper du temps. Le recours au formalisme DEVS qui offre de puissants outils pour la gestion du temps et dont l'approche événementielle semble intuitivement adaptée à une association avec Event-B s'inscrit dans ce cadre.

Ainsi, en seconde phase, le modèle formel, les propriétés prouvées et les OPRs sont transférés dans une modélisation DEVS intégrant les modèles des composants qui ne s'adaptent pas à la modélisation formelle et de procéder à l'évaluation de l'ensemble du système à travers une simulation à événements discrets. Cette approche permettrait à la fois d'obtenir des propriétés prouvées grâce aux outils formels et de résoudre les problèmes liés à la mise en interaction avec des modèles non formels ainsi que les problèmes de simulation d'instances à grande échelle. Néanmoins, la mise en œuvre de cette approche pose des verrous qu'il faudrait lever, notamment :

— une représentation équivalente en DEVS des modèles et des propriétés prouvées dans l'outil formel (Event-B)

— une méthodologie permettant d'extraire les connaissances nécessaires à la preuve des obligations de preuves résiduelles (non prouvées par l'outil formel.

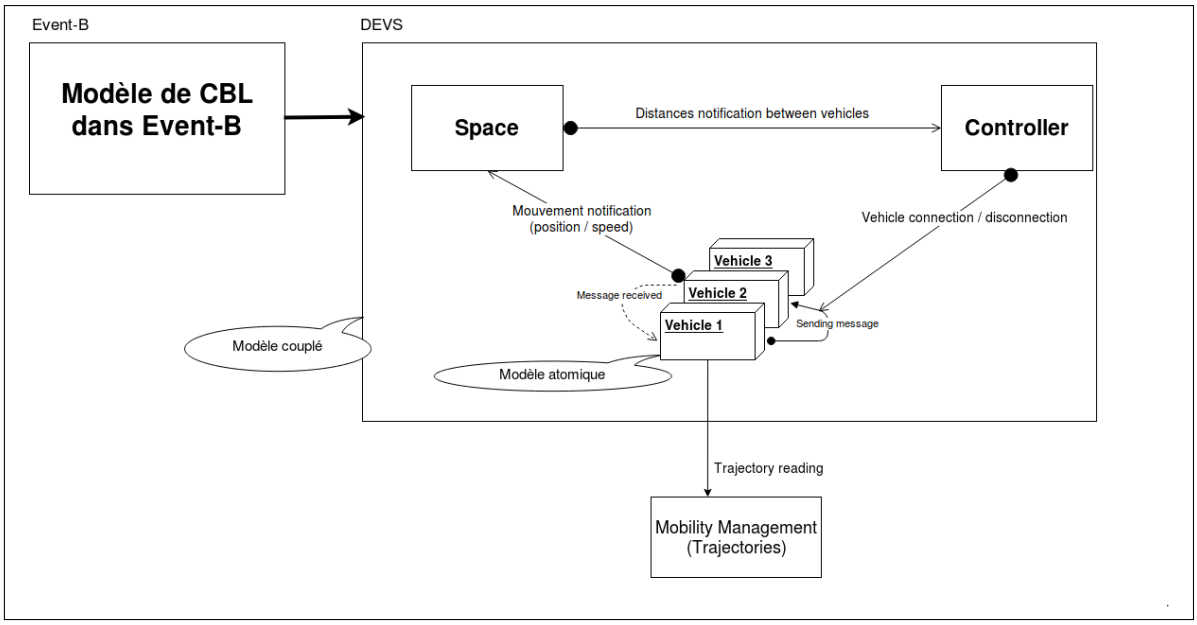

FIGURE 4.2 – Modèle DEVS du réseau ad hoc

4.2 Transformation des concepts formels Event-B en concepts de simulation DEVS

L'idée est de partir d'un point de vue global du modèle formel proposé, en citant les blocs des éléments qui vont être représentés. Puis mettre une correspondance les blocs d'éléments du modèle formel avec leur représentation en dans la modélisation DEVS.

Dans la simulation DEVS (fig. 4.2), un véhicule fait fonctionner le protocole CBL. Il est représenté dans le contexte formel par un noeud (élément de NODES), en tant qu'élément individuel de la modélisation. Dans la modélisation DEVS, le concept équivalent qui va le représenter, c'est «Vehicle». Un nœud peut être Branch ou Leaf. Dans la partie formelle, nous l'avons modélisé par deux ensembles "Branch" et "Leaf", et en même temps par la variable «branch_status». Dans le contexte formel, le fait d'appartenir à Branch ou à Leaf est global, ces ensembles sont donc des variables globales du réseau ad hoc, alors que branch_status(s) peut rendre compte à la fois au niveau local du véhicule ou au niveau global du réseau de l'appartenance de s à Branch ou Leaf. Toute une série d'invariants assurent la cohérence entre ces trois variables à tous les niveaux de la modélisation formelle. Les modèles présents dans la modélisation formelle, même lorsqu'ils concernent le protocole,peuvent donc être représentés à différents niveau du modèle DEVS : Controller, Vehicle, Space, etc.

Ainsi, les ensembles NODES, Branch, Leaf, etc, sont représentés au niveau du modèle qui a une vue globale des véhicules du réseau, à savoir le Controller. Lorsqu'une instance de Vehicle s met à jour sa variable locale branch_status, il doit notifier le Controller pour que ce dernier le déplace de l'ensemble Branch à l'ensemble Leaf ou inversément, et éventuellement qu'il mette à

jour dans la table globale branch_status(s) si elle a été modélisée également.

4.3 Répartition des composants du modèle Event-B dans les composants du système modélisé dans DEVS

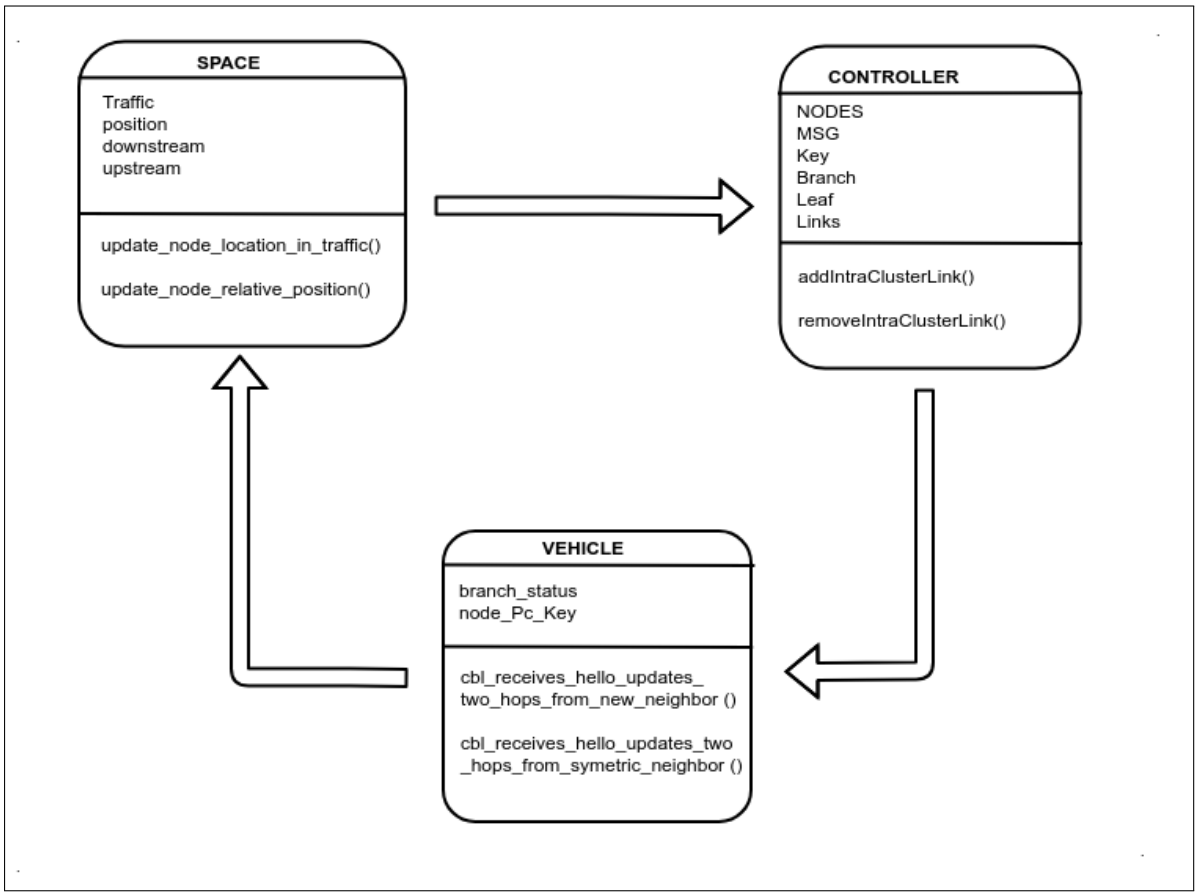

FIGURE 4.3 – Répartition des composants du modèle Event-B dans les composants du système

Afin de faire le passage de ma modélisation formelle avec Event-B vers DEVS, nous avons commencé par repartir nos variables, nos invariants et nos événements selon leur visibilité dans le composant du système correspondant dans la modélisation DEVS. Dans un premier temps, nous allons les classer dans les trois classes : Vehicle, Space et Controller.

La figure 4.3 présente une répartition de quelques attributs et événements du modèle Event-B après projection dans la modélisation à éventements discrets DEVS.

Les attributs globaux sont modélisés soit dans Space, soit dans Controller parce qu'ils ont la capacité de "voir" l'ensemble des instances de Vehicle dans le réseau ad hoc.

4.3.1 Controller

Plus précisément, le Controller peut par exemple créer des nouvelles liaisons entre deux nœuds après avoir reçu une information de la part de Space concernant leurs localisations géographiques.

```
class Controler:public vle::devs::Executive
{
public:
    Controler(const vle::devs::ExecutiveInit& init,
              const vle::devs::InitEventList& events)
        : vle::devs::Executive(init, events),
        mVehicleNumber(vle::value::toInteger(events.get("vehicleNumber"))),
        mTrajectoryPath(vle::value::toString(events.get("trajectoryPath"))),
        mTrajectoryPrefix(vle::value::toString(events.get("trajectoryPrefix"))),
        mTrajectoryExt(vle::value::toString(events.get("trajectoryExt")))
    {
        std::valarray < double > values_min(0., 3);
        std::valarray < double > values_max(0., 3);

        values_min[0] = toDouble(toSet(events.get("boundaries_x"))[0]);
        values_min[1] = toDouble(toSet(events.get("boundaries_y"))[0]);
        values_min[2] = toDouble(toSet(events.get("boundaries_z"))[0]);
        values_max[0] = toDouble(toSet(events.get("boundaries_x"))[1]);
        values_max[1] = toDouble(toSet(events.get("boundaries_y"))[1]);
        values_max[2] = toDouble(toSet(events.get("boundaries_z"))[1]);
        mBoundaries.set(Point < 3, double >(values_min),
                        Point < 3, double >(values_max));
    }
    virtual ~Controler() { }
    virtual vle::devs::Time init(const vle::devs::Time&);
    virtual void output(const vle::devs::Time&,
                        vle::devs::ExternalEventList&) const;
    virtual vle::devs::Time timeAdvance() const;
    virtual void externalTransition(
        const vle::devs::ExternalEventList&,
        const vle::devs::Time&);
    virtual void internalTransition(const vle::devs::Time&);
    virtual vle::value::Value* observation(
        const vle::devs::ObservationEvent& event) const;
private:
    NODE_set NODES
    MSG MSG
    Key Key
    NODE Branch
    NODE Leaf
    Link_set Links
private:
    void connectVehicles(vle::devs::ExternalEvent* e);
    void createModel(int index, int vehicle_index);
    void disconnectVehicles(vle::devs::ExternalEvent* e);
    void addIntraClusterLink();
    void removeIntraClusterLink();
}
```

FIGURE 4.4 – Le fichier en-tête de Controller après la répartition des attributs du modèle formel

Il peut également gérer l'affectation des instances de Vehicle à Branch ou Leaf au gré des notifications reçues par ces instances après qu'ils aient exécuté le protocole et sélectionné leur noeud Branch, Branch amont, Branch aval, etc.

La figure 4.4 présente le fichier en-tête de Controller après la répartition des attributs du modèle formel et l'ajout d'événements tels que :

— void addIntraClusterLink()

— void removeIntraClusterLink()

4.3.2 Space

```
class Space : public vle::devs::Dynamics
{
public:
    Space(const vle::devs::DynamicsInit& init,
          const vle::devs::InitEventList& events);


    virtual void output(const vle::devs::Time& time,
                        vle::devs::ExternalEventList& output) const;
    virtual vle::devs::Time timeAdvance() const;
    virtual vle::devs::Time init(const vle::devs::Time&);
    virtual void internalTransition(const vle::devs::Time& time);
    virtual void externalTransition(const vle::devs::ExternalEventList& event,
                                    const vle::devs::Time& time);
    virtual vle::value::Value* observation(
        const vle::devs::ObservationEvent&) const;
    virtual void confluentTransitions(
        const vle::devs::Time& time,
        const vle::devs::ExternalEventList& events)
    {
        externalTransition(events, time);
        internalTransition(time);
    }


private:
  Traffic traffic;
  Position position;
  Upstream upstream;
  Downstream downstream;
private:
    void moveVehicles();
    void updateConnections();
    void update_node_location_in_traffic ()
    void update_node_relative_position ()
}
```

FIGURE 4.5 – Le fichier en-tête de Space après la répartition des attributs du modèle formel

Le Space communique en temps réel avec les véhicules et récupère les données liées à leurs positions. Il est donc apte à mettre à jour la représentation de l'objet Traffic et les relations entre instances de Vehicle telles que downstream, upstream, etc.

La figure 4.5 présente le fichier entête du Space après la répartition des attributs du modèle

formel et l'ajout de fonctionnalités :

— void update_node_location_in_traffic ()

— void update_node_relative_position ()

4.3.3 Vehicle

```
class Vehicle : public vd::Dynamics,
                public DynamicElement
{
private:
    string branch_status;
    Key node_Pc_key
private:
    vle::devs::ExternalEvent* cloneExternalEvent(
        const vle::devs::ExternalEvent* event) const

    void updateMovement(const vd::Time& time)
    void build_hello(const vd::Time& time)
    int calcul_number_veh_LOS(double neighbor_X,double neighbor_Y,
                              double neighbor_Z, std::string neighbor_adrr)

    bool calcul_AV_AR(double x1, double y1, double z1,  double steering_angle1,
                              double x2, double y2, double z2 )

    bool calcul_direction(double steering_angle1,  double steering_angle2 )
    int creation_chaine()
    static void calcul_stats_chaine()
    double calcul_link_duration(double x1, double y1, double z1, double speed1,
                                double steering_angle1, double x2, ndouble y2,
                                double z2, double speed2,double steering_angle2)

    bool comparerNoeudsBranches(double x1, double y1, double z1, double speed1,
                                double steering_angle1, double x2, double y2,
                                double z2, double speed2,double steering_angle2,
                                double x3, double y3, double z3,double speed3,
                                double steering_angle3, bool node_type3)

    bool comparerNoeudsFeuille(string originator_addr, double steering_angle1,
                                double steering_angle2, double speed2)

    bool comparerNoeudsChaine(string originator_addr,double x1, double y1,
                                double z1, double speed1,double steering_angle1,
                                double x2, double y2, double z2, double speed2,
                                double steering_angle2, bool apporte_voisin)

    void update_number_leaf_attached_stat()
    void process_hello(const vle::devs::ExternalEvent* ee, double time)

    void transaction_init();
    void transaction_answr( node s, node d, mesg msg, KEY ks, KEY kd,
                                transaction t, transaction trans,
                                string trans_pending, int size_trans,
                                int size_trans_pending, string sent_packet,
                                int size_sent_packet);

}
```

FIGURE 4.6 – Le fichier en-tête de Vehicle après la répartition des attributs du modèle formel

Les attributs locaux sont orientés vers Vehicle parce qu'ils détaillent ce qui se passe au ni-

veau d'un véhicule en particulier. Notamment les variables branch_status, Nodes_Pc_Key, ainsi les procédure d'initialisation de transaction entre deux véhicules, de réponse aux transactions, de sélection de noeud Branch, etc, sont définies dans Vehicle.

La figure 4.6 présente le fichier entête du "vehicle" après la répartition des attributs du modele formel après modification et l'ajout des fonctionaitées

— void transaction_init()

— void transaction_answer(...)

— ...

4.3.4 De Event-B vers DEVS

Dans notre système nous avons des variables ou des invariants qui sont des axiomes. Un axiome peut être considéré comme une contrainte de fonctionnement puisqu'il est toujours vrai et qu'il n'y a aucun doute à son sujet.

D'autres invariants qui sont des propriétés que nous avons modélisées, ont toutes été vérifiées automatiquement. Mais elles sont cohérentes par rapport au niveau de détail ou de simplification que nous avons adopté dans notre modélisation. Il s'agit donc de s'assurer, grâce à la simulation, qu'elles restent bien toujours vraies dans tous les cas ou s'il existe de scénarios correspondant à un niveau plus fin où elles sont prises à défaut. Pour ce genre d'invariants, nous préconisons d'effectuer une vérification continue sur l'état de ces invariants après chaque exécution d'un événement qui est en relation avec eux ou avec les structures de données liées à ces invariants. Le but est de savoir si, à chaque exécution d'un événement, tous les invariants du modèle en relation avec cet événement sont toujours vérifiés ou si certains sont violés à cause d'un non-respect d'une hypothèse fonctionnelle qui a conduit à cet invariant.

Dans certains cas, pour des invariants qui correspondent à des propriétés sur lesquelles des cas de leur violation sont plausibles, nous préconisons d'introduire dans la modélisation à événement discret une statistique qui compte le nombre de fois que l'invariant correspondant n'est pas vérifié afin d'avoir des pourcentages sur le respect de l'invariant et pouvoir donner des intervalles de confiance au modélisateur.

Afin de convertir nos variables, constantes, axiomes, invariants et événements de Event-B vers la simulation DEVS, nous nous sommes basés sur le tableau presenté dans la figure 4.7

4.3.5 Déclaration de nouvelles structures de données

Pour définir les nouvelles structures de données dans VLE, nous avons introduit plusieurs types. Ci-dessous un exemple avec la structure représentant une transaction.

Event-B	C++	Détails
a ∈ B	B a	variable a de type B
a=b	if(a==b) {	clause conditionnelle
a < b	if(a<b) {	clause conditionnelle
a ≤ b	if(a<=b) {	clause conditionnelle
a > b	if(a>b) {	clause conditionnelle
a ≥ b	if(a>=b) {	clause conditionnelle
(a>b) ∧ (a≥c)	if ((a>b) && (a>=c) {	clause conditionnelle
(a>b) ∨ (a≥c)	if ((a>b) || (a>=c) {	clause conditionnelle
a := b + c	a = b + c;	affectation arithmétique
a := b - c	a = b - c;	affectation arithmétique
a := b * c	a = b * c;	affectation arithmétique
a := b ÷ c	a = b / c;	affectation arithmétique
a := F(b)	a = F(b);	affectation d'une fonction
a := a(b)	a = a(b);	affectation d'une fonction
a := b	a = b;	action scalaire
A⇒B	if(!A || B){	implication logique
A⇔B	if((!A || B) && (!B || A)){	équivalence logique
¬a<b	if(!(a<b)){	non logique
a ∈ N	unsigned long int a	a est un nombre naturel
a ∈ Z	signed long int a	a est un nombre entier
∀		quantificateur
∃		quantificateur
Sets	Supportée par C++	bibliothèque STL
Set1	set <type de données> Set1	bibliothèque STL
∪	set union(...)	bibliothèque STL
∩	set intersection(...)	bibliothèque STL
−	set difference(...)	bibliothèque STL
⊂	if (includes(...)){	bibliothèque STL
⊆	if (includes(...) || equal (...)){	bibliothèque STL
⊂	if (!(includes(...))){	bibliothèque STL
⊆	if (!(includes(...) || equal (...))){	bibliothèque STL

FIGURE 4.7 – Tableau récapitulatif des équivalents des fonctions Event-B dans une implémentation C/C++ qui pourrait être basée sur DEVS [MS11b]

```
typedef struct transaction {
      string name[1]={"trans1"};
      node source;
      node destination;
      KEY source_Pc_Key;
      KEY destination_Pc_Key;
      string transaction_State;
};
```

FIGURE 4.8 – Déclaration de la structure Transaction

Une transaction contient un nom, un nœud source, un nœud destination, une clé publique de la source, une clé publique de la destination et l'état de la transaction comme le montre la figure 4.8.

4.3.6 Déclaration des Invariants

```
bool inv2(std::string trans_pending[10], int size_trans_pending,  transaction trans , int size_trans)
{
    if(std::includes(trans_pending, trans_pending+size_trans_pending, trans.name, trans.name+size_trans)
    || std::equal (trans_pending, trans_pending+size_trans_pending, trans.name, trans.name+size_trans))
        return true;
    else
        return false;
}
```

FIGURE 4.9 – Déclaration d'un invariant

En se basant sur le tableau de la figure 4.7, nous avons déclaré les invariants de notre modèle de deux façons telles que le montre la figure 4.9. Si notre invariant est une simple déclaration de nouveau type, la traduction de l'invariant dans VLE sera une simple déclaration d'une variable ou structure de données. Si notre invariant est basé sur une inclusion, ou autre implication, nous le définissons sous forme d'une fonction qui retourne une valeur booléenne. La valeur est "true" si l'invariant est vérifié ou "false" si l'invariant est violé. La figure 4.9 nous montre l'invariant inv2 du modèle sécurisé (M2). L'invariant prend en entrée la liste des transactions dans l'état "trans_-pending" et une transaction trans. Le but de cet invariant est de vérifier si la transaction trans est inclut ou égale à trans_pending.

4.3.7 Développement d'un événement

La figure 4.10, présente l'événevement transaction_answer du modèle sécurisé. Les événements projetés dans VLE, sont composés de trois blocs de traitement :

— Premier bloc contient les gardes.

— Deuxième bloc présente les actions.

```
int transaction_answr( node s, node d, mesg msg, KEY ks, KEY kd, transaction t, transaction trans,
                       string trans_pending[10], int size_trans, int size_trans_pending,
                       string sent_packet[10], int size_sent_packet)
{
    std::cout << "execution de l'evenement transaction_answr" << endl<<endl;
    guard_result(gard2_transaction_answr(trans, trans_pending, size_trans_pending), 2);
    guard_result(gard3_transaction_answr(s, ks), 3);
    guard_result(gard4_transaction_answr(s, msg), 4);
    guard_result(gard5_transaction_answr(d, msg), 5);
    guard_result(gard6_transaction_answr(msg, sent_packet, size_sent_packet), 6);
    guard_result(gard7_transaction_answr(d, kd), 7);
        if(   gard2_transaction_answr(trans, trans_pending, size_trans_pending)
           && gard3_transaction_answr(s, ks)
           && gard4_transaction_answr(s, msg)
           && gard5_transaction_answr(d, msg)
           && gard6_transaction_answr(msg, sent_packet, size_sent_packet)
           && gard7_transaction_answr(d, kd))
           {
               trans=t;
               sent_packet[size_sent_packet+1]=msg.valeur;
               size_sent_packet++;
           }

    if(inv2(trans_pending,size_trans_pending,trans ,size_trans) == false)
        compteur_violation_inv2++;
     .
     .
     .

return 0;
}
```

FIGURE 4.10 – Implémentation de la fonction transaction_answer() dans VLE

— La dernière partie est introduite pour vérifier si les invariants en relation avec cet événement sont validés ou pas.

Transaction_answer contient 7 gardes et 4 actions et en relation avec une liste d'invariants.

4.3.8 Développement d'une garde

```
int gard2_transaction_answr(transaction trans,  string trans_pending[10] , int size_trans_pending){

    for (int i =0; i< size_trans_pending ; i++)
       if (trans_pending[i] == trans.name[1])
                return true;
        return false;
}
```

FIGURE 4.11 – Développement d'une garde

Chaque garde est défini dans une fonction qui retourne "true" si la garde est vérifiée ou "false" sinon, comme le montre la figure 4.11. On ne peut exécuter les actions que si toutes les gardes retournent "true".

Conclusion et perspectives

Ce mémoire de thèse porte sur la modélisation formelle et la simulation à événement discret appliquées à la conception et à l'évaluation des protocoles de communications pour les systèmes de transports intelligents (ITS), en particulier sur les réseaux ad hoc de véhicules (VANET).

Les travaux présentés dans cette thèse tentent d'introduire une nouvelle approche pour la modélisation, la vérification et la validation des ITS, en combinant la modélisation formelle et la simulation à événements discrets. L'idée de départ était de partir du modèle formel d'un protocole et d'engranger une liste des propriétés prouvées automatiquement sur ce protocole par l'outil formel et d'une liste d'obligations de preuves non prouvées automatiquement et qui nécessiteraient une preuve interactive par un expert. A partir de toutes ces informations, il s'agissait de construire une modélisation DEVS équivalente en reprenant les résultats du modèle formel soit comme des contraintes de fonctionnement (les axiomes), soit comme des observables déterminés (les propriétés prouvées), soit comme des alternatives (les OPR dont l'analyse s'averaient simples même par un modélisateur non expert) et comme des observables indéterminés (les OPR restants). Enfin, par une simulation à événements discrets qui permet d'évaluer de nombreux scénarios à large échelle, il s'agissait de déterminer si certains observables pouvaient permettre de détecter des défauts de modélisation (contradiction de propriétés prouvées suite à des observables déterminés aux valeurs non nulles en fin de simulation), si des alternatives n'étaient jamais explorées (réduction des choix sur les OPR associées) et si des observables indéterminées finissaient par se préciser, voire par être complètement déterminées (découverte de nouvelles propriétés voire connaissances sur le modèle). Afin d'illustrer cette démarche, la modélisation formelle a été consacrée à l'analyse de propriétés de sûreté et de sécurité d'un protocole de routage pour le réseau ad hoc de véhicules.

Pour réaliser ces objectifs, nous avons opté pour une démarche en trois phases. Dans un premier temps, ne disposant pas d'un modèle de protocole de routage ni de réseau ad hoc dans VLE, nous avons commencé par en réalisé un. En effet, ce modèle était essentiel pour l'analyse de faisabilite, le développement et la preuve de concept de toute notre approche. Le protocole choisi est une variante du protocole OLSR implémentant une méthode de clustering récemment proposée

pour les VANET, à savoir Chain-Branch-Leaf. Ce modèle a été présenté au chapitre 2 ainsi que les résultats d'analyse de performances réalisées sur le protocole en prenant en compte des données réelles de véhicules circulant sur un tronçon d'autoroute avec leur mobilité (positions, vitesses) réelle sur ce tronçon. Ces résultats se sont avérés conformes à ceux obtenus par une évaluation réalisée sur le même protocole avec les mêmes donnnées à l'aide d'un simulateur commercial reputé et également basé sur DEVS.

Nous avons ensuite proposé une modélisation formelle dudit protocole, CBL-OLSR, en partant d'un modèle existant d'OLSR disponible dans la littérature et en y introduisant les mécanismes spécifiques de CBL pour assurer les communications véhicules-à-véhicules et supporter les applications coopératives de sûreté. Poursuivant nos objectifs de mettre en évidence également des propriétés relatives à la sécurité, nous avons ensuite adapté un modèle de la litterature, initialement proposé pour analyser la sécurité de transactions dans les protocoles de communications, à la réalisation de transferts sécurisés de données dans un vANET qui utilise CBL-OLSR comme protocole de routage. Ces modèles, réalisés avec la méthode formelle B événementielle (Event-B) avec l'outil de modélisation Rodin méthode formelle Event-B et l'outil Rodin. Cette modélisation formelle présentée au chapitre 3 nous a permis de spécifier le comportement du système et de ses composants de manière à pouvoir définir des propriétés claires et à les vérifier dans Rodin.

Dans la dernière phase de cette thèse, nous avons amorcé la démarche de projection du modèle formel ainsi que des objects qui le composent (axiomes, variables, invariants, événements et propriétés) vers une modélisation DEVS. Nous avons montré au chapitre 4 comment nous mettons en correspondance les éléments du modèle formel présenté au chapitre 3 avec les composants de la modélisation DEVS présenté au chapitre 2. Nous illustrons d'une part l'approche globale visée et d'autre part les mécanismes de conversion des concepts utilisés dans la modélisation formelle lorsqu'ils sont projetés dans la modélisation DEVS. Dans l'état actuel de nos travaux, nous n'avons pas pu fournir l'intégralité du modèle résultant de cette projection, mais nous avons pu illustré la transformation d'au moins un objet de chacune des catégories d'objets de la modélisation. Le rythme et la durée de la thèse ne nous a cependant pas permis de faire la démonstration de l'augmentation de preuves de propriétés sur le modèle équivalent DEVS obtenu à partir du modèle formel.

Plusieurs perspectives s'ouvrent alors pour faire suite à nos travaux, à savoir :

— La première porterait sur une formalisation plus précise des critères pour déterminer les catégories d'objets du modèle formel et préciser leur mécanisme de projection vers une modélisation DEVS afin de les rendre plus génériques et adaptées à une automatisation.

— La modélisation proposée a été fortement dépendante de son application à un protocole de communications pour les réseaux ad hoc de véhicules. La démarche n'a pas été suffi-

samment généralisée pour rendre son application assez triviale pour d'autres protocoles de commmunications pour les transports, notamment dans le domaine ferroviaire. En effet, ce domaine est déjà très familiarisé avec l'utilisation de méthodes formelles notamment pour spécifier et évaluer la plupart de système de contrôles, l'ETCS entre autres, leur application à la modélisation du sous-système de télécommunication serait d'un intérêt immédiat et capital. Une généralisation de nos travaux afin de les rendre plus générique aux différents domaines de transport serait intéressant.

— Finalement, le dernier point d'amélioration de notre travail, est le développement de modules qui permettraient d'étendre un outil comme VLE pour assurer une conversion automatique d'un modèle Event-B vers une simulation DEVS. Ce programme détecterait automatiquement les variables déclarées dans le modèle Event-B et donnerait la possibilité via ses interfaces graphiques ou une bibliothèque de les préfixer en prenant en considération le composant modélisé dans VLE dans lequel elles seront représentées après la projection.

Bibliographie

[AA05] Jean-Raymond Abrial and Jean-Raymond Abrial. *The B-book : assigning programs to meanings*. Cambridge University Press, 2005. 38

[ABH+10] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, Thai Son Hoang, Farhad Mehta, and Laurent Voisin. Rodin : an open toolset for modelling and reasoning in event-b. *International journal on software tools for technology transfer*, 12(6) :447–466, 2010. 12, 40

[ABHV08] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, and Laurent Voisin. A roadmap for the rodin toolset. 2008. 40

[ABW10] Yamine Ait Ameur, Frédéric Boniol, and Virginie Wiels. Toward a wider use of formal methods for aerospace systems design and verification. *International Journal on Software Tools for Technology Transfer*, 12(1) :1–7, 2010. 37

[ARC] ARCEP. Open data. 10

[Bar97] Fernando J Barros. Modeling formalisms for dynamic structure systems. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 7(4) :501–515, 1997. 19, 23

[Bar98] Fernando J Barros. Abstract simulators for the DSDE formalism. In *Simulation Conference Proceedings, 1998. Winter*, volume 1, pages 407–412. IEEE, 1998. 24

[Bar05] Jörg Barner. *A Lightweight Formal Method for the Prediction of Non-functional System Properties*. Citeseer, 2005. 37

[BH95a] Jonathan P Bowen and Michael G Hinchey. Seven more myths of formal methods. *IEEE software*, 12(4) :34–41, 1995. 37

[BH95b] Jonathan P Bowen and Michael G Hinchey. Ten commandments of formal methods. *Computer*, 28(4) :56–63, 1995. 37

[BH06] Jonathan P Bowen and Michael G Hinchey. Ten commandments of formal methods...ten years later. *Computer*, 39(1) :40–48, 2006. 37

[BHP17] Alexey Vinel Benoit Hilt, Marion Berbineau and Alain Pirovano, editors. *Networking Simulation for Intelligent Transportation Systems : High Mobile Wireless Nodes*. ISTE, Wiley, 2017. 8, 9

[BL09] Jens Bendisposto and Michael Leuschel. Proof assisted model checking for b. In *International Conference on Formal Engineering Methods*, pages 504–520. Springer, 2009. 40

[BM10] Nazim Benaissa and Dominique Méry. Cryptographic protocols analysis in event b. In Amir Pnueli, Irina Virbitskaite, and Andrei Voronkov, editors, *Perspectives of Systems Informatics*, pages 282–293, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. 69, 79

[Bou16] Youssef Bouanan. *Contribution à une architecture de modélisation et de simulation à événements discrets : application à la propagation d'information dans les réseaux sociaux*. PhD thesis, Bordeaux, 2016. 23

[But02] Michael Butler. A system-based approach to the formal development of embedded controllers for a railway. *Design Automation for Embedded Systems*, 6(4) :355–366, 2002. 37

[Cav02] Ana Cavalli, editor. *Ingénierie des protocoles et qualité de service*. Eyrolles, 2002. 12

[Cho96] Alex Chung Hen Chow. Parallel devs : A parallel, hierarchical, modular modeling formalism and its distributed simulator. *TRANSACTIONS of the Society for Computer Simulation*, 13(2) :55–68, 1996. 19

[CJ03] T. Clausen and P. Jacquet. Optimized link state routing protocol (olsr), 2003. 29

[Cra] Crawdad. Crawdad datasets. 10

[CSR17] Emna Chebbi, Patrick Sondi, and Eric Ramat. Enhancing Formal Proofs of Network Protocols for Transport Systems using Discrete Event Simulation. In *The 9th Int. Conference on Advances in System Simulation*. IARIA, 2017. iii, 81

[CSR+18] Emna Chebbi, Patrick Sondi, Eric Ramat, Lucas Rivoirard, and Martine Wahl. Simulation of a clustering scheme for vehicular ad hoc networks using a devs-based virtual laboratory environment. In *ANT 2018, The 9th International Conference on Ambient Systems, Networks and Technologies*, volume 130, pages pp–344. Elsevier, 2018. 28

[CW96] Edmund M Clarke and Jeannette M Wing. Formal methods : State of the art and future directions. *ACM Computing Surveys (CSUR)*, 28(4) :626–643, 1996. 37

[CZ94] Alex Chung Hen Chow and Bernard P Zeigler. Parallel DEVS : A parallel, hierarchical, modular modeling formalism. In *Simulation Conference Proceedings, 1994. Winter*, pages 716–722. IEEE, 1994. 19, 23

[DAR14] Jesus Diaz, David Arroyo, and Francisco B Rodriguez. A formal methodology for integral security design and verification of network protocols. *Journal of Systems and Software*, 89 :87–98, 2014. 12, 38

[DPR13] Nicolas DUHAIL, Marc PARENTHOEN, and Pascal REDOU. Devs et ses extensions pour des simulations multi-mode les en interaction multi-e chelles. 2013. 17

[DRG02] Raphaël Duboz, Éric Ramat, and Norbert Giambiasi. Utilisation du formalisme devs pour la spécification de systèmes d'agents réactifs. In *JFSMA*, pages 99–102, 2002. 23

[Eur] Eurecom. Open air interface. 10

[Fro18] Adam Frost. Kapsch supplying equipment for australia's largest c-its connected vehicle pilot project. 2018. iii, 5

[FTM19] F. Fakhfakh, M. Tounsi, and M. Mosbah. An evaluative review of the formal verification for vanet protocols. In *2019 15th International Wireless Communications Mobile Computing Conference (IWCMC)*, pages 1209–1214, June 2019. 12

[Gye14] Lee Gyesik. How to formally model features of network security protocols. *International Journal of Security and Its Applications*, 8(1) :423–432, 2014. 12, 38

[Her16] Christopher Herbez. Parallélisation massive de dynamiques spatiales : contribution à la gestion durable du mildiou de la pomme de terre. Littoral, 2016. 15

[ISI] Information Science Institute ISI. Ns2 documentation. [En ligne], (consulté le 11 juin 2018). 9

[Jac01] Daniel Jackson. Lightweight formal methods. In *International Symposium of Formal Methods Europe*, pages 1–1. Springer, 2001. 37

[JB14] Michael Jastram and Prof Michael Butler. Rodin user's handbook : Covers rodin v. 2.8. 2014. 40

[KP16] M. Kamali and L. Petre. Modelling link state routing in event-b. In *2016 21st International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 207–210, Nov 2016. 12, 38, 42, 48, 49, 51, 52, 55, 66

[LLC03] Jong-keun Lee, Min-Woo Lee, and Sung-Do Chi. Devs/hla-based modeling and simulation for intelligent transportation systems. *Simulation*, 79(8) :423–439, 2003. 9

[LW07] Fan Li and Yu Wang. Routing in vehicular ad hoc networks : A survey. *IEEE Vehicular technology magazine*, 2(2), 2007. 29

[Mat] Mathworks. Matlab documentation. [En ligne], (consulté le 11 juin 2018). 9

[Mor] Moraves. Openrbc. 10

[MS11a] Dominique Méry and Neeraj Kumar Singh. Analysis of DSR protocol in Event-B. In *Symposium on Self-Stabilizing Systems*, pages 401–415. Springer, 2011. 12, 38

[MS11b] Dominique Méry and Neeraj Kumar Singh. Automatic code generation from event-b models. In *Proceedings of the second symposium on information and communication technology*, pages 179–188. ACM, 2011. iii, 89

[MSHK11] Atekeh Maghsoudlou, Marc St-Hilaire, and Thomas Kunz. A survey on geographic routing protocols for mobile ad hoc networks. *Carleton University, Systems and Computer Engineering, Technical Report SCE-11-03*, 2011. 29

[noa16] http ://sumo.dlr.de/wiki/Main_page, 2016. ://sumo.dlr./wiki/main$_page_2$016

[Opea] OpenETCS. Open etcs. 10

[Opeb] OpenSim. Omnet++ documentation. [En ligne], (consulté le 11 juin 2018). 9

[PS17] Marion Berbineau Patrick Sondi, Eric Ramat. *A Virtual Laboratory as an Assessment Tool for Wireless Technologies in Railway Systems*, pages 79–106. 04 2017. 10

[PSR17] Robert Protzmann, Bjoern Schuenemann, and Ilja Radusch. *Simulation of Convergent Networks for Intelligent Transport Systems with VSimRTI : High Mobile Wireless Nodes*, pages 1–28. 04 2017. 9

[RFP05] O. R. Ribeiro, J. M. Fernandes, and L. F. Pinto. Model checking embedded systems with promela. In *12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'05)*, pages 378–385, April 2005. 12

[Riv] Riverbed. Opnet documentation. [En ligne], (consulté le 11 juin 2018). 9, 25

[Rob99] Stewart Robinson. Three sources of simulation inaccuracy (and how to overcome them). volume 2, pages 1701 – 1708 vol.2, 02 1999. 8

[RPZW09] Muhammad Rahman, Algirdas Pakstas, and Frank Zhigang Wang. Network modelling and simulation tools. *Simulation Modelling Practice and Theory*, 17 :1011–1031, 07 2009. 8

[Rus93] John Rushby. *Formal methods and the certification of critical systems*. SRI International, Computer Science Laboratory, 1993. 37

[RWS+18] Lucas Rivoirard, Martine Wahl, Patrick Sondi, Marion Berbineau, and Dominique Gruyer. Chain-Branch-Leaf : A clustering scheme for vehicular networks using only V2V communications. *Ad Hoc Networks*, 68 :70–84, 2018. iii, 15, 29, 30, 31, 32, 34

[SAI14] Baraa T Sharef, Raed A Alsaqour, and Mahamod Ismail. Vehicular communication ad hoc routing protocols : A survey. *Journal of network and computer applications*, 40 :363–396, 2014. 29

[SBKM13] Patrick Sondi, Marion Berbineau, Mohamed Kassab, and Georges Mariano. Generating test scenarios based on real-world traces for ertms telecommunication subsystem evaluation. In Marion Berbineau, Magnus Jonsson, Jean-Marie Bonnin, Soumaya Cherkaoui, Marina Aguado, Cristina Rico-Garcia, Hassan Ghannoum, Rashid Mehmood, and Alexey Vinel, editors, *Communication Technologies for Vehicles*, pages 223–231, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. 10

[SC] Zou Shengrong and Wang Chen. Comparison and application of event-b and b method. 40

[SCB11] Razvan Stanic, Emmanuel Chaput, and André-Luc Beylot. Simulation of vehicular ad-hoc networks : Challenges, review of tools and recommendations. *Computer Networks*, 55(14) :3179 – 3188, 2011. Deploying vehicle-2-x communication. 9

[SK14] R Shenbagapriya and N Kumar. A survey on proactive routing protocols in manets. In *Science Engineering and Management Research (ICSEMR), 2014 International Conference on*, pages 1–7. IEEE, 2014. 29

[Son04] Michele Sonnessa. *Modelling and simulation of complex systems*. PhD thesis, Cultura e impresa, 2004. 9

[Son12] *Toward a Common Platform for Simulation-Based Evaluation of Both Functional and Telecommunication Sub-Systems of the ERTMS*, volume 2012 Joint Rail Conference of *ASME/IEEE Joint Rail Conference*, 07 2012. 15

[SS13] Ak Singh and Vk Singh. Formal Modeling of Distance Vector Routing Protocol using Event-B. *Electronic and Electric Engineering ISSN 2231-1297*, 3 :91–98, 2013. 12, 38

[SZ00] Hessam Sarjoughian and Bernard P Zeigler. Devs and hla : Complementary paradigms for modeling and simulation? *TRANSACTIONS of the Society for Computer Simulation*, 17(4) :187–197, 2000. 9

[VLE] VLE. Virtual laboratory environment. 21

[Wal13] Gurleen Kaur Walia. A survey on reactive routing protocols of the mobile ad hoc networks. *International Journal of Computer Applications*, 64(22), 2013. 29

[Win90] Jeannette M Wing. A specifier's introduction to formal methods. *Computer*, 23(9) :8–22, 1990. 37

[YHF16] Aznam Yacoub, Maâmar el-amine Hamri, and Claudia Frydman. Using dev-promela for modelling and verification of software. In *Proceedings of the 2016 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, SIGSIM-PADS '16, pages 245–253, New York, NY, USA, 2016. ACM. :2016 :UDM :2901378.2901388

[Zac06] Gregory Zacharewicz. *Un environnement G-DEVS/HLA : Application à la modélisation et simulation distribuée de workflow*. PhD thesis, 2006. 9

[ZKP00] Bernard P Zeigler, Tag Gon Kim, and Herbert Praehofer. *Theory of modeling and simulation*. Academic press, 2000. 16, 20