Analyse formelle de spécifications hybrides à partir de modèles SysML pour la validation fonctionnelle des systèmes embarqués

Slim Medimegh

université PARIS-SACLAY

CentraleSupélec

NNT : 2018SACLC093

Analyse formelle de spécifications hybrides à partir de modèles SysML pour la validation fonctionnelle des systèmes embarqués

Thèse de doctorat de l'Université Paris-Saclay

Préparée à CentraleSupélec

École doctorale n°580
Sciences et technologies de l'information et de la communication (STIC)

Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Palaiseau, le 20 Décembre 2018, par :

Slim Medimegh

Composition du Jury :

Philippe Dague Professeur, Paris-Sud (LRI)	Président
Louise Travé-Massuyès Directrice de recherche, CNRS (LAAS)	Rapporteur
Jean-Philippe Babau Professeur, Université de Bretagne (STICC)	Rapporteur
Marie-Agnès Peraldi-Frati Maître de Conférence, Université de Sophia Antipolis (I3S)	Examinatrice
Frédéric Boulanger Professeur, CentraleSupélec (LRI)	Directeur de thèse
Jean-Yves Pierron Ingénieur chercheur, CEA LIST (LIDEO)	Co-encadrant

Thèse de doctorat

Dédicaces

En hommage au grand Homme à la grande Classe qu'était mon Père,
A la mémoire de ma mère, que Dieu l'accueille dans son vaste paradis,
A celle qui m'a inspiré le désir de vivre autrement, Ma femme,
A celle qui demeure pour moi ma seconde maman, ma sœur Meriam,
A celui qui est pour moi mon fils, mon frère Hamza,
A toute ma famille,
A tous mes Amis

Remerciements

Mes premiers remerciements s'adressent à Mesdames et Messieurs les membres du jury de la soutenance de cette thèse pour l'intérêt porté à mon travail.

Merci à mon directeur de thèse, professeur *Frédéric Boulanger*, pour l'orientation de mon sujet de doctorat, son encadrement minutieux et attentif et ses réflexions précises et pertinentes. Il m'a guidé dans l'épineux sentier de cette thèse, grâce à sa disponibilité et ses précieux conseils. Il a su orienter ma thématique de recherche avec mes goûts scientifiques.

Je tiens aussi à exprimer ma gratitude envers mon encadrant, ingénieur chercheur *Jean-Yves Pierron* pour sa gentillesse, pour les conseils qu'il a su me prodiguer, le temps qu'il a bien voulu me consacrer tout au long de mes travaux. Sa réflexion sur les différentes problématiques que j'ai abordées dans cette thèse a amélioré la qualité, la cohérence et la profondeur de mon travail.

Je remercie également les chercheurs de l'équipe *LIDEO*, le laboratoire dans lequel j'ai passé ces trente-six mois de ma thèse, pour l'intérêt qu'ils ont porté à mon travail. Je remercie particulièrement : *Asma Smaoui, Vincent Lorenzo, Nicolas Fauvergue, Arnault Lapitre* pour les moments d'échanges très enrichissants, scientifiques et personnels. Leur collaboration dans l'utilisation des techniques d'ingénierie dirigée par les modèles, a été un apport considérable dans l'avancement des travaux de ma thèse.

Table des matières

Table des figures

Chapitre 1

Introduction

Les systèmes embarqués sont devenus essentiels dans la pluspart des secteurs industriels : énergie, transport, télécommunications, santé... L'omniprésence de l'informatique dans nos sociétés impose la nécessité de la fiabilité des logiciels : les conséquences économiques, juridiques ou même humaines d'une défaillance logicielle peuvent être catastrophiques pour leur concepteur et/ou distributeur, ce qui rend les procédés industriels de plus en plus complexes.

Afin de garantir leur bon fonctionnement, il est nécessaire de prendre en compte les aspects continus et événementiels de leur dynamique. D'une façon générale, les systèmes dynamiques faisant intervenir explicitement et simultanément des phénomènes ou des modèles de type dynamique continus et événementiels sont appelés systèmes dynamiques hybrides **SDH**. Ces systèmes sont classiquement constitués de processus continus interagissant avec des processus discrets. La modélisation cherche à formaliser des modèles précis pouvant décrire le comportement riche et complexe des SDH. Ces systèmes sont classiquement représentés par le formalisme des automates hybrides, définis par des ensembles d'états et de variables discrètes et continues.

La simulation de ces systèmes hybrides nécessite des données précises et une puissance de calcul pour détecter les changements des valeurs continues et les synchroniser avec les transitions discrètes. Mais, durant les premières phases de conception, les valeurs exactes de quelques paramètres ne sont pas encore connues, alors qu'il est nécessaire de vérifier les comportements possibles du système pour prendre des décisions de conception.

Pour les variables continues des systèmes hybrides, les lois d'évolution sont souvent décrites par des équations différentielles. Les variables continues évoluent ainsi en suivant ces lois sur les états temporisés du système. Souvent ces équations sont complexes ou incomplètes. Dans ces conditions, la simulation qualitative peut être une alternative à la simulation numérique pour ce type de modèle. Son principe est la discrétisation des domaines de variation des variables continues et de leurs dérivées. Elle mène à une description qualitative de l'évolution des variables continues : *positive, negative, null, increasing, decreasing, constant...* De cette manière, on peut obtenir un arbre des comportements abstraits du système. Chaque nœud décrit l'évolution des variables du système pendant une phase du comportement. Combiné avec un modèle de la partie discrète du système, on obtient un modèle global discret du comportement du système à qui des techniques formelles peuvent être appliquées.

Parfois les équations différentielles ne sont pas disponibles ou quelques paramètres ne sont pas encore connus. Dans cette situation, nous pouvons utiliser un modèle abstrait des lois d'évolution des variables continues pour effectuer une simulation

qualitative. En effet, nous pouvons représenter les variations de vitesse des variables continues (dérivées premières non explicitées) et établir des liens de causalité entre elles. Par exemple, les valeurs précises et les équations différentielles ne sont pas nécessaires pour prédire qu'un objet laissé dans un champ gravitationnel touchera le sol. Ce type de modèle qualitatif peut être modélisé par un automate hybride.

Pour assurer un haut niveau de fiabilité, il est essentiel d'effectuer dans les premières phases de conception, une analyse du comportement du système quand il intéragit avec son environnement. Cet environnement implique généralement diverses connaissances métiers (électronique, hydraulique, mécanique, automatique, informatique...) décrites dans un formalisme approprié. Le langage SysML [2] tend à devenir un standard dans la modélisation de spécifications multi-domaines. Il est utilisé pour spécifier le système, analyser sa structure et ses fonctionnalités. Il est aussi utilisé pour vérifier des propriétés du système avant sa réalisation. Ce langage graphique offre des possibilités d'extension par la définition de profils. Cette activité est facilitée par l'utilisation de plateformes évoluées de modélisation telle que Papyrus, développée au CEA LIST.

Diversity est une plateforme logicielle, développée par l'équipe du laboratoire LISE du CEA LIST dans laquelle j'ai fait ma thèse. Cet outil est dédié à l'analyse formelle de modèles de systèmes embarqués. Il met en œuvre des techniques de simulation symbolique pour générer un arbre des comportements du système à partir duquel des vérifications peuvent être effectuées afin d'améliorer la fiabilité et la sûreté d'une spécification. Les technologies mises en œuvre dans cette plateforme sont tout à fait adaptées pour la simulation qualitative des systèmes hybrides.

1.1 Contributions de la thèse

La contribution de nos travaux consiste à proposer une méthodologie d'analyse outillée dans le périmètre des systèmes hybrides. Notre méthodologie nous permettra l'analyse fonctionnelle des systèmes hybrides dans les premières phases de conception. Nous avons mis en place des modèles d'exécution qui permettent d'effectuer la simulation qualitative sans équations différentielles au sens du modèle mathématique continu. Notre approche fournit un langage de modélisation dédié à la simulation qualitative sans équations différentielles. Ce langage est basé sur le support fourni par le langage SysML. Afin de rendre notre méthodologie utilisable par les ingénieurs, nous avons implémenté les modèles d'exécution dans une chaîne d'outils qui permet

de filtrer les comportements du système hybride et d'aider l'utilisateur à comprendre les comportements qualitatifs obtenus.

1.2 Structure de la thèse

Notre contribution, et un parcours de l'ensemble des disciplines y ayant trait, sont présentés dans le présent manuscrit comme suit :

— **chapitre 1, Introduction**. Ce chapitre introduit le contexte pluridisciplinaire de notre thèse ;

— **chapitre 2, Contexte scientifique**. Ce chapitre détaille le contexte scientifique de notre thèse en présentant les différents domaines liés à nos travaux ;

— **chapitre 3, État de l'art**. Ce chapitre présente les différentes approches existantes dans la littérature pour la simulation qualitative ;

— **chapitre 4, Modèles d'exécution pour la simulation qualitative**. Ce chapitre décrit les trois modèles d'exécution que nous avons mis en place dans notre approche pour effectuer la simulation qualitative sans équations différentielles ;

— **chapitre 5, Langage pour la simulation qualitative**. Ce chapitre présente notre langage conçu dans le but de rendre notre approche utilisable par les concepteurs dans les premières phases de conception ;

— **chapitre 6, Implémentation : Chaîne d'outils pour la validation des spécifications hybrides**. Ce chapitre présente la chaîne d'outils qui implémente nos modèles d'exécution pour la validation fonctionnelle des systèmes hybrides ;

— **chapitre 7, Cas d'étude : Plateforme de gestion thermique d'un véhicule hybride**. Ce chapitre présente le cas d'étude industriel sur lequel nous avons appliqué notre méthodologie ;

— **chapitre 8, Conclusion**. Ce chapitre clôture notre manuscrit. Il rappelle alors notre contribution, met le point sur les limites du travail et indique comment elles pourraient être dépassées.

Chapitre 2

Contexte scientifique

Dans ce chapitre, nous allons présenter le contexte scientifique autour duquel se déroulent les travaux de recherche de cette thèse : nous présenterons l'ingénierie dirigée par les modèles, les problématiques que nous voulons résoudre dans nos travaux, le raisonnement qualitatif, la simulation qualitative, les systèmes hybrides, l'exécution symbolique et finalement les outils et les défis pour la simulation qualitative.

2.1 Ingénierie dirigée par les modèles

2.1.1 Motivation

De nos jours les technologies logicielles ne cessent d'évoluer sans fin. Tous les jours de nouveaux paradigmes et de nouvelles technologies apparaissent : technologies WEB (HTML, CSS) ; technologies Microsoft comme C♯, .Net. La question qui se pose, c'est quelle est la meilleure technologie. La réponse à cette question est celle à venir. Dans le but de profiter de la dernière technologie, il est nécessaire d'adapter une application déjà développée. Le coût d'adaptation est très élevé. En effet, il est obligatoire de réécrire presque entièrement l'application puisque dans cette dernière il y a un mélange du code métier et du code technique. Prenons comme exemple une application écrite en *C* qui fait des calculs scientifiques distribués sur un réseau de machines. Si nous voulons prendre cette application et passer du langage *C* vers *Java*, il est impossible de reprendre le code existant bien que les algorithmes de distribution des calculs et de répartition des charges sur les machines soient indépendants de la technologie mise en œuvre. Il est clairement nécessaire de découpler la logique métier et la mise en œuvre technologique [17]. C'est pour ces raisons que l'Ingénierie Dirigée par les Modèles (*IDM*) a vu le jour fin *2000*.

2.1.2 MDA

L'OMG (Object Management Group) avait rendu publique son initiative *MDA Model Driven Architecture* [4] qui visait à la définition d'un cadre normatif pour l'IDM. Le but principal de cette approche est de séparer les parties métier de leur mise en œuvre technologique et ainsi garantir l'interopérabilité des modèles fonctionnels pour différents choix d'implémentation [48]. Pour réaliser cette séparation, MDA se base sur des technologies et standards de l'OMG comme le *langage de modélisation UML, le langage de contraintes OCL, le langage de transformation de modèles QVT etc.* Le MDA propose trois principaux niveaux de modèles [48] :

— *CIM, Computation Independent Model* correspondant à la spécification du système de point de vue extérieur de l'utilisateur ;

— *PIM, Platform Independent Model* correspondant à la spécification de la partie métier d'une application indépendamment de la technologie de mise en œuvre ;

— *PSM, Platform Specific Model* correspondant à la spécification d'une application après projection sur une plate-forme technologique donnée.

2.1.3 Principes de l'IDM

L'Ingénierie Dirigée par les Modèles est une discipline récente du génie logiciel qui promeut les modèles en entités de première classe dans le développement logiciel [10]. C'est un paradigme rassemblant de nombreux principes autour de la notion centrale de *modèle*. L'IDM repose sur les principes suivants [48] :

— *Capitalisation*, les modèles doivent être réutilisables ;

— *Abstraction*, les modèles doivent être indépendants des technologies dans le but d'adapter une logique métier à un contexte et faire évoluer les applications vers de nouvelles technologies ;

— *Modélisation*, faite suivant une vision bien définie afin de pouvoir générer le code final du logiciel pour une plateforme donnée ;

— *Séparation des préoccupations*, l'IDM se base sur deux principales préoccupations, le métier (le cœur de l'application) et la plateforme de sa mise en œuvre.

Le but de l'IDM est de passer d'une vision plutôt contemplative des modèles qui vise la documentation, spécification et la communication, à une vision productive qui permet de générer le code final du logiciel pour une technologie de mise en œuvre donnée. Pour que les modèles soient productifs, il est nécessaire qu'ils soient bien définis. Ceci est assuré par la notion de *méta-modèle*. Par conséquent, les modèles productifs peuvent être manipulés et interprétés via des outils qui permettent [48] :

— la définition de méta-modèles, de langages *DSL : Domain Specific Language*, de transformations et leur exécution ;

— la génération de code ;

— la composition de modèles ;

— la génération d'environnements graphiques de modélisation, la vérification de conformité de modèles, la spécification de contraintes...

L'IDM est donc une forme d'ingénierie générative, par laquelle tout ou partie d'une application informatique est engendrée à partir de modèles. Dans cette nouvelle vision, les modèles occupent une place importante parmi les artefacts de développement des systèmes. Ils doivent en revanche être précis et riches afin de pouvoir être interprétés ou transformés par des machines. Le processus de développement des systèmes peut alors être vu comme une séquence de transformations de modèles, chaque transformation prend un ou des modèles en entrée et produit un ou des modèles en sortie, jusqu'à l'obtention d'artefacts exécutables [25].

2.1.4 SysML

Le langage **SysML** (Systems Modeling Language) [2] tend à devenir un standard dans la modélisation de spécifications multi-domaines. SysML a vu le jour en tant qu'extension du langage orienté-objet **UML** (Unified Modeling Language) pour couvrir toutes les étapes de conception de systèmes complexes et hétérogènes. Il est basé sur UML et remplace la modélisation de classes et d'objets par la modélisation de blocs pour un vocabulaire plus adapté à l'ingénierie Système. Un bloc englobe tout concept logiciel, matériel, données, processus et même la gestion des personnes. Ce langage graphique offre des possibilités d'extension par la définition de profils. Il existe différents types de diagrammes dans SysML [3]. Le premier type est le *diagramme de structures* qui peut être :

— un *diagramme de définition de blocs*, représentant le bloc principal et la hiérarchie des blocs qui le composent, qu'ils soient logiciels ou matériels ;

— un *diagramme de blocs interne*, décrivant la vue interne d'un bloc et les différentes communications entre ses sous-blocs ;

— un *diagramme paramétrique*, une nouveauté SysML, permettant d'intégrer des analyses système (performance, fiabilité, etc.) avec des blocs de contrainte. Un bloc de contrainte représente une expression mathématique dont les paramètres peuvent faire référence à des éléments du système.

Le deuxième type est le *diagramme de comportements*. Ce dernier peut être :

— un *diagramme d'états*, utilisé avec SysML de la même manière qu'avec UML2 et modélisant tous les états possibles d'un bloc ;

— un *diagramme de séquence*, représentant les interactions entre un acteur et le système d'un point de vue « boîte noire » ;

— un *diagramme d'activité*, utilisé pour représenter les étapes d'un traitement ;

— un *diagramme de cas d'études*, basé sur les interactions acteurs/système pour identifier les acteurs et les cas d'utilisation d'un point de vue utilisation du système.

2.2 Problématiques

Les systèmes industriels deviennent de plus en plus complexes. Pour faire face à cette complexité, les ingénieurs utilisent la simulation numérique par des outils comme *Matalb* ou *Modelica*. Ce type de simulation fournit des résultats précis des comportements des systèmes hybrides. En effet, lorsque tous les paramètres du système sont connus dans la phase de conception, ces outils permettent de simuler le comportement du système d'une manière précise et satisfaisante pour l'utilisateur. Cependant en phase de préconception, la simulation numérique des systèmes hybrides est un peu limitée. Durant cette phase du cycle de développement, l'ingénieur ne dispose pas de tous les paramètres numériques de son système hybride, mais il veut avoir une idée du comportement global de son système avant sa phase de réalisation. Une possibilité est de faire de la validation partielle en utilisant les simulateurs numériques : l'ingénieur fait des expériences avec plusieurs valeurs de paramètres de son système. Les résultats obtenus sont des scénarios préétablis qui ne donnent pas une idée globale sur tous les comportements possibles du système. Par conséquent, en optant pour cette approche, l'ingénieur peut rater des scénarios critiques de son système. En effet, il y a un manque d'exhaustivité des comportements des systèmes hybrides. Aussi dans la phase de préconception, les ingénieurs n'ont pas forcément le simulateur numérique qui a un coût élevé en termes de temps de calcul.

Ainsi, afin de garantir un haut niveau de fiabilité des comportements des systèmes hybrides et aider les utilisateurs à prendre des décisions pour mieux concevoir leur systèmes, il est indispensable d'effectuer au plus tôt dans son cycle de développement une analyse de ce que le système doit faire vis-à-vis d'un environnement avec lequel il interagit.

Le langage SysML sert à spécifier les systèmes, analyser leurs structures et leurs fonctionnements. Il permet de décrire et concevoir des systèmes composés de sous-systèmes. Cependant, dans SysML, nous ne disposons pas d'éléments de langage pour décrire le comportement continu des états hybrides. En effet, il n'y a que le diagramme paramétrique pour la modélisation du comportement des variables continues. Les *constraint blocks* du diagramme paramétrique sont utilisés pour modéliser les équations différentielles qui décrivent le comportement dynamique et continu des variables.

Dans la phase de préconception des systèmes hybrides, ces équations ne sont pas toujours disponibles ou sont incomplètes. D'où la nécessité d'avoir un langage SysML qui permet la modélisation des systèmes hybrides sans équations différentielles au sens du modèle mathématique continu.

2.3 Raisonnement qualitatif

Le raisonnement qualitatif est un domaine de l'intelligence artificielle, en effet un programme d'IA doit être capable de résoudre un problème avec des informations quantitatives restreintes ou absentes nécessaires pour la simulation numérique. Le raisonnement qualitatif produit :

— la prédiction des états qualitatifs du système ;

— l'explication des comportements qualitatifs obtenus, en se basant sur la description qualitative du système fournie par l'utilisateur [45].

2.3.1 Motivation du raisonnement qualitatif

L'intérêt pour le raisonnement qualitatif a de diverses raisons :

— Dans les problèmes physiques, il y a souvent un manque de données numériques, ainsi la résolution de tels problèmes est coûteuse en terme de temps de calcul ;

— On ne dispose pas souvent d'un modèle quantitatif complet. En effet les relations reliant les différents paramètres du système ne peuvent pas être liées par des formules mathématiques. La représentation qualitative peut être une solution pour ce genre de problème par des liens de causalité entre les paramètres.

2.3.2 Application du raisonnement qualitatif

Le raisonnement qualitatif est utilisé dans plusieurs domaines [50]. Parmi ces domaines :

— l'Explication, à partir de la structure d'un système, le raisonnement qualitatif fournit une explication du comportement de ce système [23] ;

— la Prédiction, à partir de la description du système, le raisonnement qualitatif donne une prédiction de l'ensemble des états futurs du système [31] ;

— le Diagnostic, partant d'un état ou comportement inconsistant avec la structure d'un système, il consiste à détecter l'élément de la structure à l'origine du comportement observé [50] ;

— la Commande d'un système se résume dans la description à un certain niveau permettant d'agir. Ainsi, l'utilisation des représentations qualitatives peut être d'un apport non négligeable ;

— la Planification, partant d'un état initial du système et d'un état cible pour le système, elle doit être capable de décrire les transformations permettant de passer de l'état initial du système vers l'état cible ;

— la Conception, connaissant la description du comportement souhaité du système, elle consiste à produire une structure du système qui aura le comportement souhaité.

2.3.3 Techniques du raisonnement qualitatif

2.3.3.1 Raisonnement causal

La causalité est un des concepts essentiels pour raisonner sur les systèmes physiques. Ainsi, la physique qualitative s'est intéressée à la notion de causalité et à ses différentes formes opérationnelles [21]. La causalité est considérée comme une réponse à la question "comment le système fonctionne" [21] . En effet, elle fournit une connexion entre la structure et le fonctionnement permettant d'expliquer quels changements fonctionnels vont résulter de quels changements structurels. Cette connexion est très importante dans le domaine du diagnostic qui est devenu très tôt un domaine d'application très privilégié du raisonnement causal. Différentes méthodes ont été mises au point dans le cadre du raisonnement causal qualitatif. Nous allons citer les principales méthodes évoquées dans [21] :

— La causalité mythique de De Kleer et Brown [24] est définie comme étant un comportement décrit d'une manière causale comme un diagramme d'états. Ce diagramme contient les états qualitatifs du système et les transitions entre ces états. Il est obtenu par la résolution des équations différentielles qualitatives du système et se base sur les notions classiques de causalité : *les effets ont des causes uniques (nécessité), la cause est structurellement proche de l'effet (localité) et la cause précède l'effet (temporalité)* [21] ;

— L'ordonnancement causal de Williams, Iwasaki et Simon [38] [24] [39] est une méthode qui déduit l'ordre causal d'une analyse purement structurelle des équations du système. Leur approche est l'opposé de la causalité mythique dans laquelle il est nécessaire de faire la résolution des équations différentielles. L'ordonnancement causal est introduit comme une relation asymétrique au sein des

variables et équations d'un système sans établir un lien avec les constituants physiques sous-jacents [21] ;

— Les bond graphs (les graphes de liaisons) sont proposés pour représenter les systèmes physiques comme un intermédiaire entre la représentation physique et le modèle mathématique [21]. Ils s'appuient sur une représentation des transferts d'énergie au sein du système et sur un formalisme de modélisation en termes de variables d'effort et de flux (la puissance est le produit d'un effort par un flux). Les bond graphs constituent un modèle graphique à partir duquel on peut d'une part décrire des modèles mathématiques qui sous-tendent le comportement physique du système et d'autre part exprimer des relations de cause à effet entre les différentes variables du système. C'est pratiquement pour la dernière raison que plusieurs chercheurs du raisonnement qualitatif se sont intéressés aux Bond Graphs et à leur modélisation.

2.3.3.2 Théorie des processus qualitatifs

Forbus [30] parlait de changements en se posant les questions suivantes : "quels changements se produisent et comment ?", mais aussi "pourquoi un changement a eu lieu et quelles sont les causes de ce changement ?" [30]. La théorie des processus apporte des réponses à ces questions, en effet ce sont les processus qui causent le changement. Les phénomènes de changement d'état physique comme l'ébullition, la solidification, la liquéfaction, le mouvement ou la pression sont des exemples de processus. Un processus se base sur cinq éléments : les individus, les pré-conditions, les conditions sur les valeurs, les relations et les influences [30].

2.3.3.3 Envisionnement

Cette technique est introduite pour la première fois par De Kleer [22] dans le but de résoudre un système mécanique. Elle consiste à prédire le comportement d'un système physique à partir d'une description qualitative. Elle produit comme sortie, un arbre d'états qualitatifs décrivant ce qui pourrait arriver. Il s'agit d'un arbre de comportements qui peut être considéré comme un graphe de transitions dont les chemins présentent les comportements qualitatifs possibles du système. L'envisionnement était la base de la simulation qualitative.

2.4 Simulation qualitative

La simulation qualitative est une autre technique du raisonnement qualitatif. Elle est utilisée pour l'analyse qualitative des systèmes continus. Le but de ce genre de simulation est de générer les comportements qualitatifs du système à partir d'une description qualitative de ce dernier souvent sous forme d'équations différentielles qualitatives et d'un état qualitatif initial. Les équations différentielles qualitatives sont une abstraction des équations différentielles ordinaires. La simulation qualitative se base sur des connaissances imprécises, voire incomplètes du système étudié. Ainsi, elle nécéssite [50] :

— des représentations de structures du système ;

— des algorithmes pour calculer les comportements qualitatifs du système à partir des équations différentielles et des conditions initiales.

Les dimensions qui caractérisent les algorithmes de la simulation qualitative sont :

— les états initiaux ;

— les conditions de filtrage des états et des transitions ;

— la génération de nouvelles bornes ;

— l'espace de quantités (domaine de variation d'une variable) caractérisé par des valeurs remarquables $l_1 < l_2 \ldots < l_n$.

Le but de la simulation qualitative est de raisonner à propos du comportement d'une variable continue sans calculer sa valeur numérique exacte. La Figure 2.1 montre le genre de changements qualitatifs qui nous intéressent.

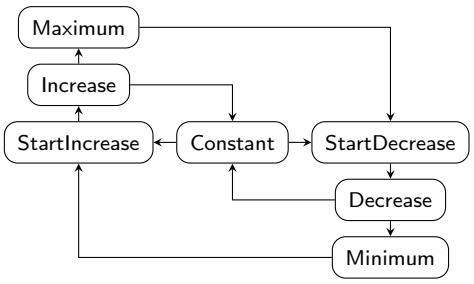

FIGURE 2.1 – Modèle qualitatif des lois d'évolution des variables continues

Pour les systèmes hybrides, la partie discrète n'est pas influencée par le processus d'abstraction. Les variables continues et leurs dérivées sont discrétisées dans le but de ne considérer que leurs changements qualitatifs. Ainsi, les transitions continues deviennent discrètes et le système résultant est entièrement discrétisé et peut être traité par des techniques traditionnelles de vérification des systèmes discrets.

2.4.1 Principe général de la simulation qualitative

La simulation qualitative se base sur le principe de la discrétisation par partitionnement des domaines de variation des variables continues du système et de ses dérivées. Le but est de calculer leur états qualitatifs (*increasing, decreasing, constant...*). Ce principe peut être étendu jusqu'à la dérivée $n^{ième}$ pour distinguer plus d'états qualitatifs. Une fois les états qualitatifs correspondant à ce partitionnement qualitatif créés, on construit les transitions possibles entre ces états en prenant en considération les contraintes de continuité. Par exemple, chaque variable ou dérivée ne peut passer de négatif à positif sans être nulle entre les deux. Aussi une variable peut passer de négatif à zéro et de zéro à positif si sa dérivée est positive etc. Ceci limite la possibilité d'évolution et réduit par conséquent la taille de l'automate correspondant. Finalement, les équations différentielles du système sont abstraites dans un graphe discret. Les états de ce graphe sont basés sur le partitionnement du changement des valeurs des variables continues et de dérivées. Les transitions de ce graphe sont les évolutions physiquement possibles entre ces états. Le résultat de la simulation qualitative est une abstraction des solutions des équations différentielles.

2.4.2 Extension pour les automates hybrides

Dans les systèmes hybrides, les équations différentielles qui décrivent la partie continue du système interagissent avec un système discret. Pour la simulation qualitative, la partie discrète du système n'est pas influencée par le processus de discrétisation décrit ci-dessus dans la sous-section 2.4.1. Mais ce comportement discret réagit au changement de comportement continu discrétisé et lui ajoute des contraintes. La Figure 2.2 de la balle rebondissante, montre que le contrôle discret réagit au changement qualitatif de la hauteur de la balle et force un changement qualitatif dans la vitesse de la balle pour la faire rebondir quand elle touche le sol.

En combinant la partie discrète inchangée avec le modèle qualitatif discrétisé de la partie continue du système, on obtient à la fin un modèle totalement discret qui peut être analysé par des outils pour les systèmes discrets.

2.4.3 Approches de la simulation qualitative

Il existe deux stratégies pour la simulation qualitative [35] :

— avec équations différentielles ;

— sans équations différentielles.

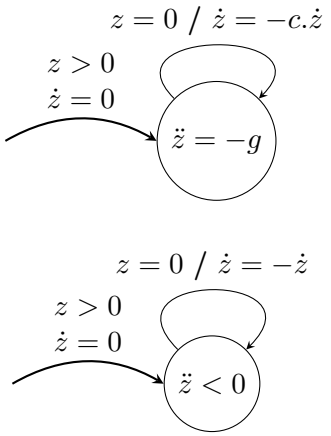

FIGURE 2.2 – Automate hybride (haut) et qualitatif (bas) de la balle rebondissante

Dans la première stratégie, les équations différentielles sont connues et des outils comme QEPCAD [15] sont utilisés pour déterminer les conditions nécessaires aux changements qualitatifs. Quand un changement est possible, la branche correspondante est étiquetée par la condition du changement. Dans l'autre cas, la branche correspondante dans l'arbre d'exécution symbolique est coupée.

Dans la deuxième stratégie, les équations différentielles sont absentes ou incomplètes et parfois complexes. Dans ce cas, on construit un modèle qualitatif de ces equations en considérant les relations entre les variables, leur dérivées premières et secondes. Les résultats de ce genre de simulation sont évidemment moins précis que la première approche. On peut utiliser ce genre de simulation qualitative dans les premières phases du développement d'un système. Durant les travaux de cette thèse, nous nous sommes intéressés à la deuxième approche.

2.5 Systèmes hybrides

2.5.1 Définition

Les systèmes hybrides sont des systèmes dynamiques faisant intervenir des évènements discrets et des comportements continus. Ils font interagir un mode opératoire continu avec un mode opératoire discret à travers une interface [54]. Ce type de système provient de l'organisation hiérarchique des systèmes complexes de surveillance et de contrôle, ou de l'interaction entre les algorithmes de planifications discrètes et les algorithmes de contrôles continus.

2.5.2 Modélisation des systèmes dynamiques hybrides

Dans la littérature, il existe plusieurs formalismes de modélisation de ce genre de systèmes. Parmi ces formalismes, on trouve *les automates hybrides* et *les bonds graph à communications*.

2.5.2.1 Automates hybrides

Un automate hybride est défini par un ensemble de variables continues, d'états et de transitions discrètes contenant des conditions de franchissement des transitions et des affectations de ces variables. L'évolution des variables continues est décrite par différents types d'équations différentielles :

— *les automates hybrides polyédriques*, si les équations différentielles, les gardes et affectations sont linéaires ;

— *les automates hybrides rectangulaires*, si les équations différentielles sont constantes et les conditions sont comparées aux constantes ;

— *les automates hybrides temporisés*, si les équations différentielles et les conditions incluent le temps. Ce type d'automates peut être manipulé par des outils comme UPPAAL [7], KRONOS [13], TIAMO [12]...

Il existe d'autres outils qui traitent d'autres types de systèmes hybrides : HYTECH [36] pour traiter des automates hybrides rectangulaires avec des conditions linéaires sur les transitions et CHECKMATE [18] pour manipuler des évolutions continues non linéaires avec des conditions linéaires.

2.5.2.2 Les bond graphs à communications

Le bond graph est une technique graphique utilisée pour modéliser les systèmes avec un langage unifié pour tous les domaines des sciences physiques [11]. On peut associer des sous-modèles de différents types de systèmes tels que les systèmes électriques, mécaniques, hydrauliques, thermiques en un seul bond graph, ce qui permet une visualisation graphique des relations de cause à effet, et assure la conservation de la puissance.

Principe de base Dans le formalisme des bond graphs, il existe deux variables généralisées permettant d'assurer une analogie entre les différents principaux domaines de la physique ; la variable d'effort e et la variable de flux f. Lorsqu'on intègre l'effort par rapport au temps, on obtient le moment p. En intégrant le flux par rapport au

temps, on obtient le déplacement q. Le transfert de puissance entre deux sous systèmes est représenté par une demi-flèche qui correspond au "bond" (liaison) du graphe. Le sens de la demi-flèche correspond au sens de transfert positif de la puissance. La liaison porte les variables d'effort et de flux. Le produit de l'effort et du flux est égal à la puissance $P = e * f$. Le flux est représenté du coté de la demi-flèche [54]. La Figure 2.3 illustre la représentation d'une liaison d'un bond graph.

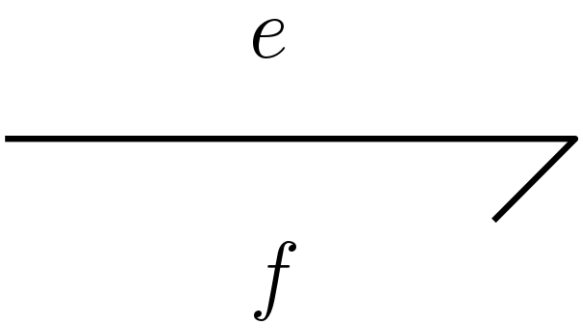

FIGURE 2.3 – Représentation d'une liaison de bond graph

La Figure 2.4 présente la signification des variables généralisées pour les principaux domaines de la physique.

Domaine	Effort e	Flux f	Moment généralisé ρ	Déplacement généralisé q
Electrotechnique	Tension u	Courant i	Flux magnétique λ	Charge q
Mécanique de translation	Force F	Vitesse v	Quantité de mouvement ρ	Déplacement x
Mécanique de rotation	Couple C	Taux de rotation ω	Moment cinétique σ	Angle θ
Hydraulique & pneumatique	Pression P	Débit volumique q_v	Impulsion ρ	Volume V
Thermique	Température T	Flux d'entropie q_s		Entropie S
Chimie	Potentiel chimique μ	Flux molaire q_m		Nombre de moles N

FIGURE 2.4 – Variables généralisées pour différents domaines de la physique

Expression de la causalité Le modèle par graphe de liaisons permet de faire apparaître les relations de cause à effet. Prenons l'exemple de deux sous-systèmes A et B qui sont couplés et échangent de la puissance. Deux situations se présentent :

— A exerce un effort e sur B, B réagit en envoyant un flux f à A en fonction de e ;

— A envoie un flux f à B, B réagit par un effort e en fonction de f.

Pour modéliser graphiquement la notion de cause à effet, un trait causal est placé perpendiculairement à chaque liaison. Le trait causal est placé du côté de l'élément sur lequel l'effort est appliqué. La Figure 2.5 présente la notion de causalité entre deux sous-systèmes A et B.

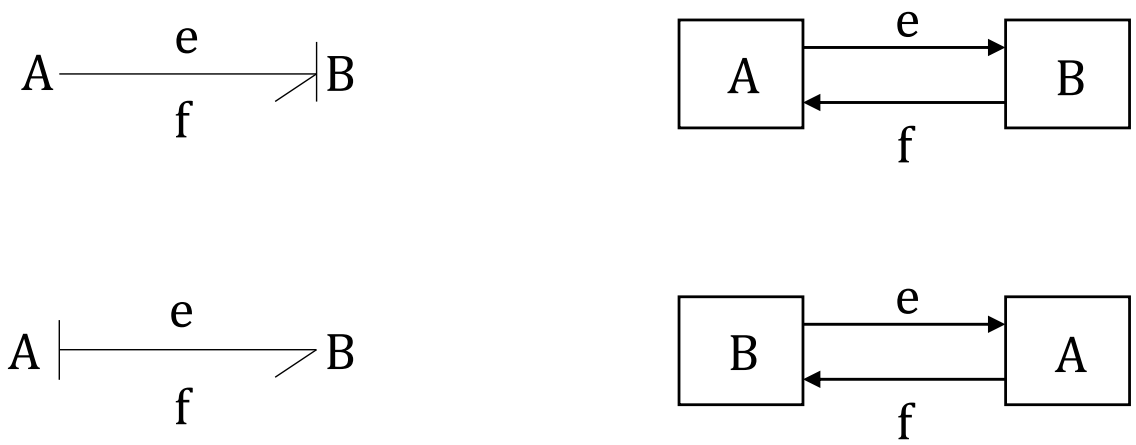

FIGURE 2.5 – Causalité dans les bond graphs

2.6 Exécution symbolique

L'exécution symbolique a été proposée dans [43] et [19] dans le but de construire des tests structurels pour des programmes séquentiels. Le principe de l'exécution symbolique se base sur l'utilisation de symboles comme données d'entrée à la place des valeurs numériques. Dans la littérature, il existe différents outils utilisant l'exécution symbolique, parmi ces outils on trouve *Diversity*. Diversity est un moteur d'exécution symbolique développé au sein du CEA LIST dans le but de produire des scénarios symboliques correspondant à des classes de comportements du système. Des propriétés peuvent être prouvées sur cet ensemble de scénarios et des scénarios numériques concrets peuvent être générés à partir des scénarios symboliques. Dans le but de limiter le nombre de cas de tests, la taille et le nombre de comportements peuvent être bornés et la détection de comportements redondants peut être utilisée. La Figure 2.6 montre les principales fonctionnalités de Diversity.

Diversity utilise une adaptation de l'exécution symbolique pour générer des tests à partir de spécifications sous forme d'automates. Le langage d'entrée de Diversity est basé sur le graphe d'exécution symbolique avec affectations (STGA) [51]. Ce langage

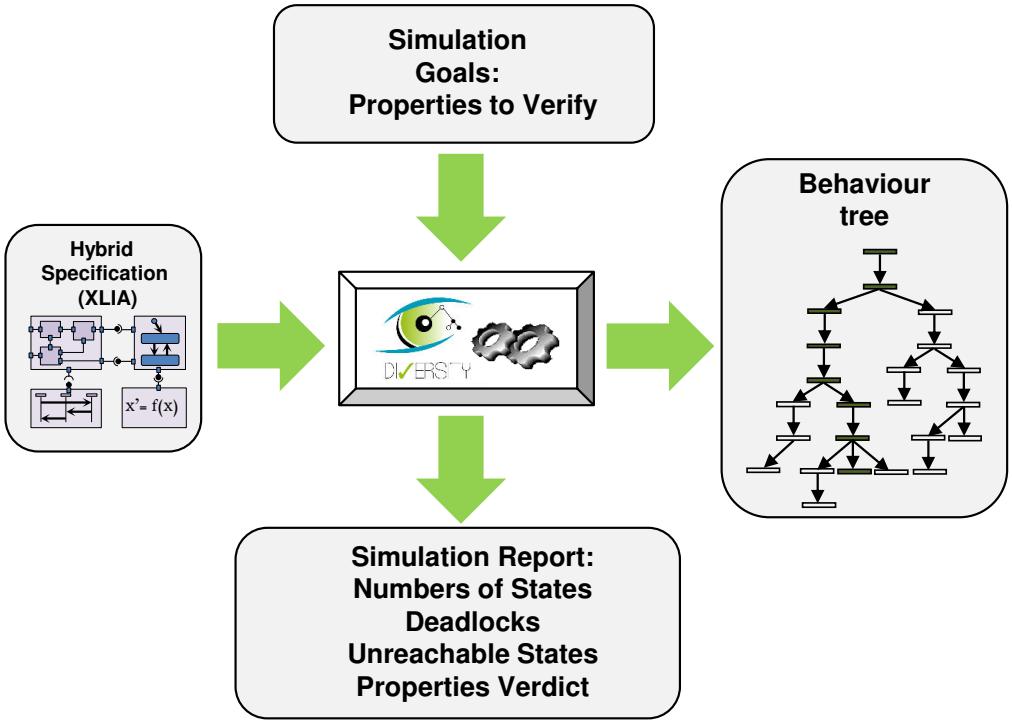

FIGURE 2.6 – Diversity

permet la représentation de tous les comportements des spécifications d'une manière abstraite. Les transitions représentent des événements qui permettent ou proviennent de l'évolution du système. Elles ont des conditions de franchissement avec des expressions logiques des variables du système pour dire quand la transition peut être tirée. Le moteur d'exécution Diversity [49] applique l'exécution symbolique pour les STGA en utilisant l'approche définie dans [37]. Ainsi, il peut simuler le comportement d'une spécification de STGA en affectant des valeurs symboliques aux variables au lieu de valeurs numériques. On obtient ainsi des états symboliques nommés "contextes d'exécution". Comme le montre la Figure 2.7, un contexte d'exécution (EC) inclut :

— un état de contrôle (CS) ;

— une condition de chemin (PC), la condition nécessaire pour atteindre l'état symbolique depuis l'état initial ;

— une mémoire symbolique qui associe à chaque variable une expression basée sur les symboles d'entrée.

Le résultat de l'exécution symbolique est un arbre de contextes d'exécution dans lequel chaque chemin représente une évolution symbolique des variables. La condition de chemin est une conjonction de tous les contextes d'exécution. La notation ♯ est

$$\mathsf{EC} = \begin{cases} \mathsf{CS} : \mathsf{Null\_der} \\ \mathsf{PC} : \dot{x}\sharp 1 = 0 \\ \ddot{x}_{\text{-}1} = \ddot{x}_{\text{-}1}\sharp 0 \\ \dot{x} = \dot{x}\sharp 1 \end{cases}$$

FIGURE 2.7 – Contexte d'exécution

utilisée pour identifier les valeurs symboliques successives des variables. La Figure 2.8 montre deux transitions t_1 et t_2 depuis l'état source Null_der, qui est l'état de contrôle CS du contexte d'exécution EC de la Figure 2.7.

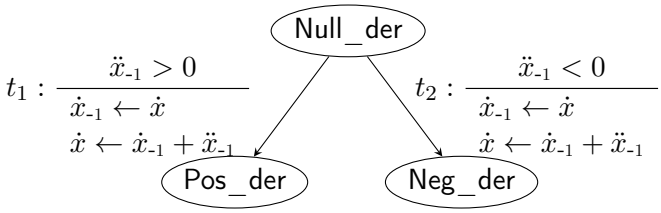

FIGURE 2.8 – Transititon

L'exécution symbolique de t_1 et t_2 produit deux contextes d'exécution décrits dans la Figure 2.9, qui correspondent à deux comportements possibles.

$$\mathsf{EC_1} = \begin{cases} \mathsf{CS} : \mathsf{Pos\_der} \\ \mathsf{PC} : \dot{x}\sharp 1 = 0 \wedge \ddot{x}_{\text{-}1}\sharp 0 > 0 \\ \dot{x}_{\text{-}1} = \dot{x}\sharp 1 \\ \dot{x} = \dot{x}\sharp 1 + \ddot{x}_{\text{-}1}\sharp 0 \end{cases}$$

$$\mathsf{EC_2} = \begin{cases} \mathsf{CS} : \mathsf{Neg\_der} \\ \mathsf{PC} : \dot{x}\sharp 1 = 0 \wedge \ddot{x}_{\text{-}1}\sharp 0 < 0 \\ \dot{x}_{\text{-}1} = \dot{x}\sharp 1 \\ \dot{x} = \dot{x}\sharp 1 + \ddot{x}_{\text{-}1}\sharp 0 \end{cases}$$

FIGURE 2.9 – Exécution Symbolique

2.7 Conclusion

Nous avons présenté le contexte général autour duquel se déroulent les travaux de recherche de notre thèse. L'ingénierie dirigée par les modèles en général et le langage SysML en particulier, sont utilisés de plus en plus dans le milieu industriel surtout dans les premières phases de développement. En effet, ils permettent de découpler la logique métier et la mise en œuvre technologique. La simulation qualitative, qui

est une technique du raisonnement qualitatif, permet de résoudre un problème avec des informations quantitatives restreintes ou absentes nécessaires pour la simulation numérique. Les automates hybrides et les bond graphs sont les formalismes les mieux adaptés pour la modélisation des systèmes hybrides. En effet, la notion de causalité est une notion importante pour comprendre le fonctionnement des systèmes hybrides. Cette notion est présente dans le formalisme des bond graphs. L'exécution symbolique peut être utilisée comme technique de simulation qualitative puisqu'elle se base sur des symboles comme données d'entrée à la place des valeurs numériques.

Chapitre 3

Etat de l'art

Dans ce chapitre, nous allons présenter les différents travaux et les différents outils qui ont été développés dans la littérature autour de la simulation qualitative.

3.1 QSIM

L'algorithme QSIM de Kuipers [44] est l'algorithme le plus populaire effectuant la simulation qualitative. L'objectif est de générer à partir d'une équation différentielle qualitative (une abstraction d'une équation différentielle ordinaire) et d'un état qualitatif initial incomplet, les comportements qualitatifs qui représentent des abstractions de solutions correspondant à l'équation différentielle qualitative [21].

3.1.1 Valeurs, états et comportements qualitatifs

La valeur qualitative d'une variable v à l'instant t est exprimée en termes de valeurs remarquables $l_1 < l_2 \ldots < l_n$ de son espace de quantités et de la tendance de son évolution. Ainsi la valeur qualitative de $v(t)$ $QV(v,t)$ par rapport à l'espace des quantités $l_1 < l_2 \ldots < l_n$, est le couple $< qmag, qdir >$, avec $qmag$, la magnitude qualitative (la grandeur d'une quantité est décrite qualitativement en fonction de ses valeurs critiques) et $qdir$, la direction qualitative (la variable augmente, diminue ou reste constante) de v à l'instant t [21].

Un état qualitatif à un instant caractéristique ou dans un intervalle de temps compris entre deux instants caractéristiques adjacents est un m-uplet de valeurs qualitatives, chaque variable dans v ayant une valeur : $QS(v,t_i) = < QV(v_1,t_i), \ldots, QV(v_m,t_i) >$ $QS(v,t_i,t_{i+1}) = < QV(v_1,t_i,t_{i+1}), \ldots, QV(v_m,t_i,t_{i+1}) >$, avec t_i et t_{i+1}, des instants caractéristiques pour au moins une variable v_j, $1 \leq j \leq m$.

Un comportement qualitatif au cours de l'intervalle de temps $[a,b]$ d'un système dynamique avec des variables v est une séquence d'états qualitatifs : $QB(v) =< QS(v,t_0), QS(v,t_0,t_1), QS(v,t_1), \ldots, QS(v,t_{p-1},t_p), QS(v,t_p) >$ avec $t_0 = a$ et $t_p = b$.

Cet algorithme se base sur l'algèbre de signes (négatif, positif, zéro) et consiste à résoudre les contraintes des équations différentielles qualitatives.

3.1.2 Contraintes sur les valeurs qualitatives

Le calcul des états qualitatifs et les transitions entre les états qualitatifs est déterminé par trois types de contraintes : les contraintes d'état, les contraintes de transition et les contraintes globales que nous allons expliquer par la suite.

3.1.2.1 Contraintes d'état

Les contraintes d'état mettent des restrictions sur les valeurs qualitatives dans un état qualitatif. Elles comportent une équation différentielle qualitative. Partant d'un état incomplètement spécifié, les contraintes d'état précisent comment on peut compléter les valeurs qualitatives manquantes. Prenons l'exemple de la contrainte d'état $ADD(x, y, z)$. Cette contrainte est une abstraction de l'équation de base $z(t) = x(t) + y(t)$. $ADD(x, y, z)$ détermine quelles sont les valeurs qualitatives $< qmag_x, qdir_x >, < qmag_y, qdir_y >, < qmag_z, qdir_z >$ possibles de x, y et z. Sous l'hypothèse que $]-\infty, 0[$ et $]0, +\infty[$ représentent respectivement $-$ et $+$, si par exemple $qval_x =< +, inc >$ et $qval_y =< 0, std >$ alors nécessairement $qval_z =< +, inc >$.

3.1.2.2 Contraintes de transition

Le deuxième type de contraintes impose des restrictions sur les transitions d'un état à un instant $QS(t_i)$ vers des états dans un intervalle successeur $QS(t_i, t_{i+1})_1, \ldots, QS(t_i, t_{i+1})_s$, ou d'un état dans un intervalle $QS(t_i, t_{i+1})_1$ vers des états à un instant successeur $QS(t_{i+1})_1, \ldots, QS(t_{i+1})_s$.Ces contraintes sont basées sur le principe de continuité des variables du système et de leurs dérivées. En effet, le sens de variation d'une variable ne peut pas passer de *inc* à *dec* sans passer par *std*. La Figure 3.1 présente les transitions d'un instant vers un intervalle alors que la Figure 3.2 présente les transitions d'un intervalle vers un instant.

	Qval(f, ti)	**Qval(f,(ti,ti+1)**
P1	(lj ,std)	(lj ,std)
P2	(lj ,std)	((lj,lj+1),inc)
P3	(lj ,std)	((lj-1,lj),dec)
P4	(lj ,inc)	((lj,lj+1),inc)
P5	((lj,lj+1),inc)	((lj,lj+1),inc)
P6	(lj,dec)	((lj,lj+1),dec)
P7	((lj,lj+1),dec)	((lj-1,lj),dec)

FIGURE 3.1 – P-transitions [44]

3.1.2.3 Contraintes globales

Les contraintes précédentes sont locales. En effet, elles concernent un seul état qualitatif ou une paire d'états qualitatifs successifs. QSIM peut produire une séquence d'états qualitatifs qui ne sont pas valides étant donnée la séquence d'états précédents.

	Qval(f,(ti,ti+1)	Qval(f, ti)
I1	(lj,std)	(lj, std)
I2	((lj,lj+1),inc)	(lj+1,std)
I3	((lj,lj+1),inc)	(lj+1,inc)
I4	((lj,lj+1),inc)	((lj,lj+1),inc)
I5	((lj,lj+1),dec)	(lj,std)
I6	((lj,lj+1),dec)	(lj,dec)
I7	((lj,lj+1),dec)	((lj,lj+1),dec)
I8	((lj,lj+1),inc)	(l*,std)
I9	((l j,l j+1),dec)	(l*,std)

FIGURE 3.2 – l-transitions [44]

Cette séquence d'états invalides est appelée *comportement factice*. Dans la littérature, plusieurs chercheurs se sont intéressés à corriger ce problème de génération de comportement factices et ont proposé des extensions à l'algorithme de QSIM [47] par des méthodes pour changer le niveau de description afin d'ajouter des informations supplémentaires pour éliminer les comportements qualitatifs sans distinction qualitative. [47] et [23] ont proposé de raisonner sur des dérivées d'ordre supérieur afin d'avoir les contraintes de courbure. [32] ont proposé d'ajouter des contraintes d'énergie en décomposant le système global en une partie conservatrice et une autre non conservatrice.

3.1.3 Déroulement de QSIM

L'algorithme consiste à prédire tous les successeurs immédiats de chaque état qualitatif. Il se déroule en 4 étapes :

- d'abord, compléter l'état initial ;
- ensuite, QSIM détermine pour chaque état complet ses successeurs possibles en utilisant ses contraintes de transitions ;
- puis, relier les états successeurs à l'état initial à condition qu'ils soient cohérents avec les contraintes d'état et les contraintes globales ;
- enfin, filtrer les états successeurs pour ne garder que les états successeurs éligibles, un état successeur est dit éligible s'il n'est pas un état :
  - *quiescent*, c'est-à-dire un état dans lequel toutes les variables ont une direction qualitative égale à *std* ;
  - *de transition*, c'est-à-dire un état inclus dans la frontière de la région opérationnelle et sortant ;

— *identique* à un état précédent, indiquant un comportement qualitatif cyclique ;

— atteignable à t = ∞ seulement.

3.1.4 Discussion

QSIM est la principale contribution dans le domaine de la simulation qualitative. La principale limitation de QSIM est la génération des comportements factices et le problème de l'explosion combinatoire. Plusieurs extensions ont été apportées à QSIM dans le but de résoudre ces problèmes : soit par l'ajout des contraintes d'énergie dans la modélisation du système, soit par l'ajout d'une description d'ordre supérieur du système. QSIM est utilisé en phase de conception, c'est-à-dire qu'on dispose déja des équations différentielles et qu'on essaye de fournir les comportements du système à partir d'une description qualitative de ces équations. Nous nous plaçons en amont, dans les premières phases du cycle du développement : en phase de spécification. En effet, nous ne disposons pas forcément des équations différentielles mais nous voulons fournir les comportements qualitatifs du système. QSIM utilise l'algèbre des signes induisant le problème de l'explosion combinatoire quand on dispose de deux quantités de signes opposés. Aussi, l'utilisateur a besoin d'introduire d'autres valeurs que *+,-,0* dans la modélisation qualitative de son système. QSIM génère les comportements qualitatifs du système sans fournir d'explications sur la cause d'obtension de ces comportements.

3.2 Ordres de grandeurs absolus

3.2.1 Motivation

Les principaux algorithmes de la simulation qualitative tels que QSIM de Kuipers [44] ou les extensions qui ont été apportées à QSIM [47] [23], se basent sur l'algèbre des signes. La principale limitation de cette approche provient du fait que le résultat d'une opération est lié à l'ordre de grandeur des paramètres. Ainsi, la connaissance des signes des paramètres mène souvent à des indéterminismes, c'est-à-dire $(+) + (-) =?$. Cet indéterminisme contribue à l'explosion combinatoire des comportements générés.

3.2.2 Construction du modèle formel

Le principe est de construire une structure algébrique plus sophistiquée, à base d'une partition de l'axe des réels plus fine que $+, -, 0$ [65] [61]. Cette nouvelle répartition permet de distinguer des quantités positives et négatives et de préciser si elles sont petites, moyennes ou grandes. La Figure 3.3 illustre cette nouvelle répartition avec NG : négatif grand, NM : négatif moyen, NP : négatif petit, PP : positif petit, PM : positif moyen, PG : positif grand. Chacun des symboles correspond à un intervalle de l'axe des réels et sont ordonnés : $NG < NM < NP < 0 < PP < PM < PG$.

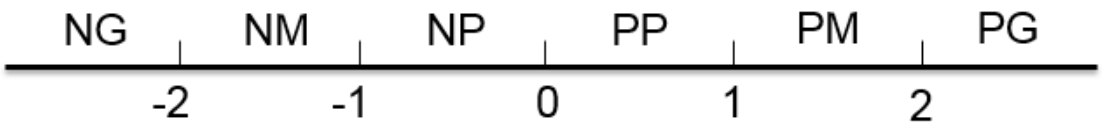

FIGURE 3.3 – Partition selon l'ordre des grandeurs

Les ordres de grandeurs absolus résolvent en partie le problème d'indéterminisme dans le calcul qualitatif. En effet, $NG + NP = NG$, $NG + NM = NG$, par contre la somme de $NP + PP$ peut avoir comme résultat $NP, PP, 0$.

3.2.3 Discussion

Les ordres de grandeurs absolus sont introduits dans le but de distinguer la grandeur des quantités positives et négatives. Ils sont utilisés pour affiner le calcul de l'algèbre des signes dans le but d'enlever l'ambiguïté de l'addition de deux quantités de signes opposés en connaissant leur ordre de grandeur. Cette approche a permis de résoudre une partie de ce problème. En effet, $PG + NP = PG$ ou PM. En première phase du cycle de développement, les gens n'ont pas forcément une idée sur l'ordre de grandeur des variables de leur système. Ainsi cette approche ne peut pas être utilisée dans notre cas. Cependant, la répartition des quantités symboliques sur l'axe des réels d'une manière ordonnée nous a inspirée dans notre méthodologie en considérant les seuils remarquables des variables du système de cette façon. Notre objectif est de fournir à l'utilisateur un langage qui lui permet de décrire les différentes relations entre les variable du système d'une manière qualitative. Les ordres de grandeurs absolus ne permettent pas de décrire des relations entre les variables du système.

3.3 Ordres de grandeurs relatifs

3.3.1 Motivation

Une autre façon de raisonner et de considérer les ordres de grandeurs relatifs. Au lieu de positionner les quantités sur une échelle qualitative en faisant un partitionnement de l'axe des réels, l'idée est de positionner les quantités les unes par rapport aux autres. C'est-à-dire, une quantité est petite ou grande par rapport à une autre. Ce type de raisonnement est souvent utilisé par les chimistes qui veulent déterminer le pH d'une solution : souvent, ils négligent certains pH par rapport à d'autres en fonction des conditions initiales et des valeurs d'équilibre des réactions [21].

3.3.2 Système FOG

La formalisation de ce mode de raisonnement a été faite pour la première fois par le système FOG. En effet, ce système définit trois opérateurs :

— *Ne*, pour dire qu'une quantité A est *négligeable* par rapport à une quantité B ($A\ Ne\ B$) ;

— *Vo*, pour dire qu'une quantité A est proche ou *voisine* en valeur absolue par rapport à une quantité B ($A\ Vo\ B$) ;

— *Co*, pour dire qu'une quantité A est *comparable* à une quantité B ($A\ Co\ B$), A et B ont le même ordre de grandeur.

Une trentaine de règles sont définies dans [63] permettant de passer d'un opérateur à un autre.

3.3.3 Système O(M)

Dans la littérature, des améliorations ont été apportées au système FOG : [55] et [56] ont proposé le système $O(M)$ dans le but de faire le couplage de système purement symbolique FOG avec des données numériques pour certaines quantités. Dans $O(M)$ comme dans FOG, les quantités sont soit, des variables symboliques de valeurs inconnues, soit des valeurs remarquables de valeurs connues. Les ordres de grandeurs dans $O(M)$ sont considérés entre les valeurs absolues des quantités. Le système $O(M)$ introduit 7 relations de base :

— *beaucoup plus petit que*, notée $<<$;

— *modérément plus petit que*, notée $-<$;

— *légèrement plus petit que*, notée $<$;

— *identique à*, notée $==$;

— *légèrement plus grand que*, notée $>$;

— *modérément plus grand que*, notée $>-$;

— *beaucoup plus grand que*, notée $>>$.

3.3.4 Les systèmes Rom(K) et Rom($\mathbb{R}$)

Dans la littérature, des améliorations ont été apportées au système $O(M)$ et Fog pour combler le manque de pouvoir expressif, tant pour la modélisation que pour le raisonnement symbolique. Le système $O(M)$ se caractérise par une impossibilité d'exprimer un changement graduel d'un ordre de grandeur à un autre. Par exemple, pour décrire dans Fog les ordres de grandeurs successifs de deux quantités positives A et B avec $A < B$ qui s'éloignent l'une de l'autre, on ne dispose que des relations suivantes :

— $A\ Vo\ B$;

— $\neg\ (A\ Vo\ B) \wedge A\ Co\ B$;

— $A\ Ne\ B$.

Ces relations ne modélisent pas le caractère graduel de l'éloignement entre A et B. Le système formel Rom(K) [20] est ainsi défini par l'introduction d'une quatrième relation qui va être définie en termes de Co. Cette relation Di "distant", modélise l'idée que deux quantités positives A et B avec $A > B$ peuvent être considérées comme distantes à partir du moment où leur écart $A - B$ est comparable à A :
$A\ Di\ B \iff ((A\ -\ B)\ Co\ A) \vee ((B\ -A)\ Co\ B)$. Quatre relations binaires Di, Ne, Co et Vo sont données sur un corps commutatif totalement ordonné (K). Le système Rom(K) se base sur quinze axiomes qui sont définis dans [20]. Cependant, le manque d'interface numérique-symbolique et la difficulté de contrôler la validité du processus d'inférence symbolique restent présents dans Rom(K) comme dans Fog et $O(M)$. Ceci est dû à l'absence du modèle relationnel de Rom(K) dans $\mathbb{R}$. Le système formel Rom($\mathbb{R}$) [20] est ainsi défini par les relations naturelles suivantes, paramétrées par un réel positif k :

— *proche à l'ordre k*, P_k, $A\ P_k\ B \iff |A - B| \leq k.Max\ (|A|\ ,\ |B|)$, c'est-à-dire pour $k < 1$, $(1 - k) \leq A/B \leq 1/(1 - k)) \vee (A = B = 0)$;

— *distant à l'ordre k*, D_k, $A\ D_k\ B \iff |A - B| \geq k.Max\ (|A|\ ,\ |B|)$, c'est-à-dire pour $k < 1$, $(A/B \leq 1 - k) \vee (A/B \geq 1\ /\ (1 - k)) \vee (B = 0)$;

— *négligable à l'ordre k*, N_k, $A\ N_k\ B \iff |A| \leq k.|B|$, c'est-à-dire $(-k \leq A/B \leq k) \vee (A = B = 0)$.

P_k est utlisée pour modéliser Vo et Co, D_k pour modéliser Di et N_k pour modéliser Ne en associant un ordre k à chaque relation.

3.3.5 Discussion

Les ordres des grandeurs relatifs permettent de positionner les quantités les unes par rapport aux autres. Ce type de raisonnement ne permet pas de modéliser comment les variables évoluent les unes par rapport aux autres. En effet, en utilisant les notions de *négligence, voisinage, comparaison*, nous ne pouvons pas décrire le fonctionnement d'un système dynamique d'un point de vue qualitatif. Cependant, ce type d'approche peut être utilisée dans le calcul qualitatif. En effet, disposant d'un système d'équations différentielles qualitatives et connaissant les relations de *négligence, voisinage, comparaison* entre les variables des équations, on peut déduire des solutions qualitatives.

3.4 Introduction de l'information temporelle dans la simulation qualitative

3.4.1 Motivation

Dans des situations réelles même simples, la simulation qualitative est confrontée au problème de l'explosion combinatoire avec un grand nombre de comportements générés. Plus le système est complexe plus la prolifération des comportements augmente. Par exemple, en prenant un simple système oscillatoire, le deuxième extremum trouvé ne peut pas être comparé au premier et dans cette situation trois cas de figures se présentent : le nouvel extremum peut être supérieur, inférieur ou égal au précédent. Dans QSIM, le temps est représenté d'une manière symbolique. Ainsi le comportement qualitatif généré est une séquence d'états qualitatifs indépendamment des durées des états. En prenant l'exemple d'un réservoir qui est en train de se vider, on ne peut pas prédire par QSIM dans combien de temps le réservoir devient complètement vide. La considération de l'information temporelle dans la simulation qualitative comme étant un filtre global pour filtrer les comportements qualitatifs peut apporter des réponses au problème de l'explosion combinatoire.

3.4.2 Filtre temporel pour le calcul de la durée des états

Dans la littérature, des chercheurs [46] [9] [64] ont commencé à s'intéresser à l'introduction de l'information temporelle dans la simulation qualitative pour calculer la durée d'évaluation des états en introduisant des informations numériques incomplètes. Le temps ne peut pas être traité d'une manière symbolique comme dans QSIM. Il est nécessaire de faire une intégration mathématique des équations différentielles. [60] s'est inspiré des formules de *Taylor* pour rendre le temps explicite. En effet, si on prend $x(t)$ une variable du système continûement différentiable en fonction du temps et en considérant deux points de temps $t1$ et $t2$, alors la formule de *Taylor-Lagrange* appliquée au premier ordre montre qu'il existe t entre $t1$ et $t2$ avec : $x(t2) = x(t1) + (t2 - t1) * \dot{x}(t)$. Ainsi deux limites de la dérivée conduisent à deux limites de la durée. La durée qualitative Δt correspondant à l'état dans lequel x évolue entre deux valeurs remarquables $x1$ et $x2$ est évaluée par la formule $\Delta t \approx (x2 - x1)/\dot{x}$ avec $\dot{x}$ la valeur qualitative de la dérivée entre $t1$ et $t2$ et $\approx$ l'égalité qualitative [65].

3.4.3 Discussion

L'ajout de l'information temporelle a enrichi la simulation qualitative en ajoutant de l'information sur la durée des états qualitatifs. Ceci a permis d'éliminer des comportements incorrects dans la prédiction des comportements qualitatifs. [60] s'est inspiré de la formule de *Taylor* pour le calcul de la durée qualitative des états. Dans notre étude, nous nous plaçons en haut du cycle de développement, le temps n'est pas critique dans les phases de spécifications. En effet, l'ajout d'un filtre temporel à ce stade de développement n'est pas d'un apport important. Cependant l'utilisation des formules de *Taylor* nous a inspiré pour mettre en place un modèle de calcul symbolique avec une intégration qualitative (en prenant un pas unitaire) des différentes valeurs symboliques de la dérivée seconde et première.

3.5 Les différents outils existants dédiés à la simulation qualitative

Les outils de raisonnement qualitatif sont très utiles pour résoudre des problèmes dans lesquels les informations sont incomplètes ou complètement absentes. Les résultats de la simulation qualitative vont être en conséquence pauvres. Ainsi les simulateurs qualitatifs sont utilisés quand il y a un manque de connaissances du système. La plupart des simulateurs qualitatifs fonctionnent en deux étapes :

— génération de toutes les transitions possibles ;

— analyse des états qualitatifs obtenus pour éliminer ceux qui ne respectent pas les contraintes.

Parmi les simulateurs qualitatifs on trouve :

3.5.1 QSIM

Nous avons présenté en détail son principe de fonctionnement dans la section 3.1 page 23. L'algorithme de QSIM se base sur des valeurs symboliques pour prédire tous les comportements possibles du système. Beaucoup de filtres ont été apportés à QSIM pour corriger le problème de l'explosion combinatoire. QSIM n'est pas complet à cause de la génération des comportements factices. Ceci est dû au fait qu'il ne raisonne pas sur le signe des dérivées supérieures. En effet, QSIM ne prend en considération que le signe de la dérivée première pour chaque variable.

3.5.2 PA

PA(Predictive Algorithm) [68] est un algorithme qualitatif qui fait partie de la même famille que QSIM. PA a le même problème que QSIM : il produit trop de transitions impossibles dans la phase de génération de transitions et certaines d'entre elles ne peuvent pas être éliminées dans la phase d'analyse qualitative [21]. Lors de cette phase, si une transition ne vérifie pas les contraintes, elle sera éliminée. Cependant, si une transition vérifie toutes les contraintes, les transitions après ne seront pas examinées. Ainsi, des transitions possibles peuvent être ignorées.

3.5.3 Garp3

Garp3 [14] est le simulateur le plus populaire pour le raisonnement qualitatif. Il simule des modèles de fragments. Ces derniers décrivent une partie de la structure et du comportement du système d'une manière générale. Ils représentent des modèles partiels composés de plusieurs :

— entités modélisant les objets physiques composant le système ;

— agents modélisant les entités de l'environnement extérieur du système ;

— hypothèses étant utilisées pour restreindre le comportement généré par le modèle ;

— configurations modélisant les relations entre les entités et les agents.

Les entités dans Garp3 sont caractérisées par deux propriétés :

— *ordre de grandeur* : zéro, plus, petit, moyen, large ;

— *dérivabilité* : $0, +, -$.

Garp3 produit en sortie un graphe d'états contenant toutes les transitions possibles. La description qualitative dans ce simulateur se fait au travers des relations existantes entre les différentes entités décrites dans les fragments de modèle. Il fournit un langage qui se base sur deux opérateurs :

— *proportionnalité*, elle concerne les dérivées de la variable. Par exemple étant données deux entités $Q1$ et $Q2$, *P+(Q2,Q1)* modélise le fait que si $Q1$ augmente alors $Q2$ augmente et si $Q1$ diminue alors $Q2$ diminue ;

— *influence directe*, elle concerne les valeurs et les dérivées. Par exemple étant données deux entités $Q1$ et $Q2$, *I-(Q2,Q1)* modélise le fait que si $Q1 > 0$ alors $Q2$ diminue et si $Q1 < 0$ alors $Q2$ augmente.

Dans Garp3, il y a moyen de raisonner sur la dérivée seconde dans le cas où deux relations opposées sont appliquées sur la même entité comme illustré dans la Figure 3.4. Dans cette figure, on voit clairement que les entités B et C ont un ordre de grandeur positif modélisé par *Plus*. B impose une *influence directe négative* sur A, alors que C impose une *influence directe positive* sur A.

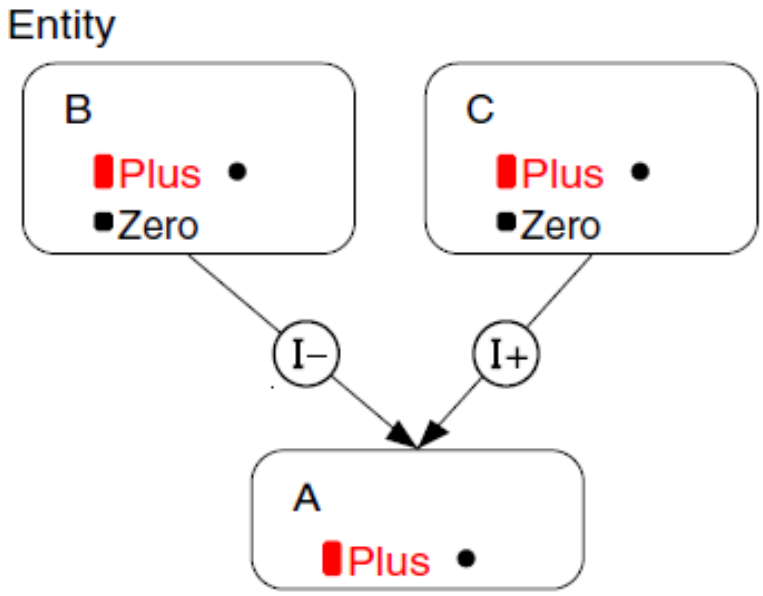

FIGURE 3.4 – Situation ambiguë par l'application de deux influences directes opposées [14]

Dans ce cas, trois états sont générés comme le montre la Figure 3.5 si :

— $B > C$ alors A diminue ;

— $B = C$ alors A reste constante ;

— $B < C$ alors A augmente.

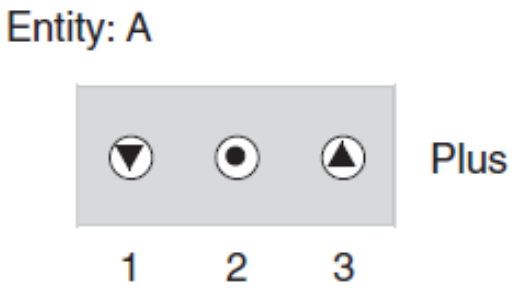

FIGURE 3.5 – Historique de valeurs de la situation expliquée dans la Figure 3.4 [14]

En ajoutant l'information sur la dérivée seconde, le sens de variation de A peut être précisé (en supposant que B et C ont le même ordre de grandeur). En effet, si la dérivée seconde de B est $-$ et celle de C est 0 alors A diminue. Par contre, si la dérivée seconde de B est 0 et celle de C est $+$ alors A augmente.

Dans ce simulateur, la variation de la dérivée seconde est visible dans l'historique de valeurs de la simulation sous forme de petites flèches à côté des symboles de la dérivée première.

Il existe d'autres simulateurs dans la littérature dédiés à la simulation qualitative et qui incluent une certaine connaissance numérique dans le but d'améliorer la qualité des comportements obtenus en terme de nombre et de précision comme :

Q2 [46] est une extension de QSIM qui permet d'associer des informations numériques aux contraintes qualitatives. En plus de l'algorithme de QSIM utilisé, Q2 utilise un algorithme de propagation d'intervalles pour réduire les comportements possibles. Les comportements générés sont plus précis que ceux de QSIM [21].

Q3 [9] est une extension de Q2. Q3 se base sur le principe de la simulation quantitative : si le temps d'échantillonnage est plus petit, les résultats sont plus précis. En effet, Q3 utilise ce principe de simulation quantitative dans la simulation qualitative en insérant des points de temps intermédiaires. Q3 augmente le nombre d'états, le nombre de contraintes et la précision des comportements générés [21].

SQSIM (semi qualitative Simulator) [41] est un simulateur incluant une combinaison de plusieurs simulateurs : QSIM, Q2, Q3 et NSIM[42] (un simulateur numérique utilisant des méthodes d'intervalles). SQSIM fait l'intersection des différents résultats trouvés par chacun d'eux. En effet, SQSIM produit une enveloppe correcte moins surdimensionnée issue de l'intersection de toutes les enveloppes des autres simulateurs [21].

QuaMo-QuaSi [53] [52] se base sur le principe des automates finis. Le système est un automate contenant les probabilités de chaque transition. En effet, il produit les états qualitatifs avec leur probabilité [21].

3.5.4 Discussion

Il existe deux types de simulateurs qualitatifs :

— Simulateur purement qualitatif, dans lequel on n'exploite pas l'information numérique comme QSIM, PA et Garp3 ;

— Simulateur semi qualitatif, dans lequel on utilise l'information numérique comme Q2, Q3, SQSIM et QuaMo-QuaSi.

Nous nous plaçons en haut du cycle de développement. Ainsi, en phase de spécification, l'information numérique n'est pas importante pour donner une idée sur le comportement global du système. Les valeurs remarquables des variables du système vont être des valeurs symboliques ordonnées. Le résultat principal des simulateurs qualitatifs est la prédiction des comportements qualitatifs du système. QSIM utilise des filtres globaux après la génération des comportements qualitatifs dans le but de réduire au maximum les comportements qualitatifs incorrects. Ceci nous a incité dans notre méthodologie à inclure des filtres mathématiques et relationnels pour éliminer les comportements qualitatifs incorrects. Cependant ce qui manque à QSIM et Garp3 est de fournir une explication pour l'obtention des comportements qualitatifs. Il est nécessaire de fournir ce genre d'explications en permettant d'observer d'une manière incrémentale les différentes variables du système pour comprendre l'influence des variables les unes sur les autres. Garp3 fournit un langage dédié au raisonnement qualitatif en se basant sur deux opérateurs : la proportionnalité et l'influence. Ces deux opérateurs, nous ont inspiré dans notre méthodologie dans la proposition d'éléments de langages qui incluent la notion de proportionnalité et d'influence. Ces deux opérateurs permettent de lier deux dérivées premières de différentes entités ensemble ainsi qu'une valeur et une dérivée première de deux entités différentes. Cependant, il

existe un manque d'expressivité dans ce langage : si nous prenons un objet de masse m, nous ne pouvons pas exprimer son poids $p = m * g$ avec $g =$ *accélération gravitationnelle* car dans Garp3, nous ne pouvons pas lier une dérivée seconde avec une valeur directement. Aussi, les valeurs de comparaison des opérateurs de Garp3 sont égales à zéro. L'utilisateur a besoin de comparer les paramètres à d'autres valeurs remarquables autres que 0. Par conséquent, il est nécessaire de disposer d'un langage qualitatif qui permet d'intégrer d'autres seuils symboliques de comparaison.

Aussi Garp3, fournit la variation de la dérivée seconde sous forme de petites flèches à coté des symboles de la dérivée. Du point de vue utilisateur, il n'est pas facile d'analyser les comportements générés par ce simulateur. D'où le besoin de calculer les différents états qualitatifs en prenant en considération les dérivées premières et secondes. La notion de causalité est une notion essentielle dans le raisonnement qualitatif. Cette notion est implicite dans Garp3 pour les opérateurs de *proportionnalité* et *d'influences*. Elle est explicite dans les bond graphs par le biais d'un trait perpendiculaire du côté de l'élément sur lequel l'effort est appliqué. Rendre cette notion explicite par un langage qualitatif causal, facilitera la modélisation du système du point de vue utilisateur. L'utilisation de Garp3 reste restreinte. En effet, le modèle qualitatif du système par ce simulateur n'est pas utilisé par d'autres simulateurs pour améliorer la qualité de l'analyse sur les comportements qualitatifs générés. Faire un langage qualitatif qui se base sur un standard de modélisation que les industriels utilisent en phase de préconception, améliorera l'utilisation de l'approche qualitative dans les premières phases de développement. Ainsi, une communication entre le modèle qualitatif standardisé et d'autres outils de simulation rendra l'analyse du système global plus fiable.

Dans ce chapitre, nous avons présenté les différents travaux existants autour de la simulation qualitative et les différents outils dédiés à la simulation qualitative. Nous avons montré les différents avantages et inconvénients de chaque approche et notre positionnement par rapport à ces différents travaux. Aussi, nous avons mentionné les différentes méthodes qui nous ont inspiré dans notre contribution.

Chapitre 4

Modèles d'exécution pour la simulation qualitative

Dans ce chapitre, nous allons présenter les 3 modèles de calcul élaborés dans la thèse pour améliorer la simulation qualitative sans équations différentielles au sens du modèle mathématique continu : (1) le modèle d'exécution énuméré, (2) le modèle exécution symbolique avec intégration d'Euler, (3) le modèle d'exécution avec contraintes qualitatives.

4.1 Préliminaire

Nous commençons le chapitre par la définition de *valeur qualitative, variable d'état et comportement qualitatif* que nous utilisons dans notre méthodologie.

4.1.1 Valeur qualitative

La valeur qualitative d'une variable v est exprimée en termes de valeurs remarquables $l_1 < l_2 \ldots < l_n$ de son espace de quantités. Ainsi la valeur qualitative de v $QV(v)$ par rapport à l'espace des quantités $l_1 < l_2 \ldots < l_n$, est l'ensemble de valeurs dans lequel $v < l_1$, $v > l_2$, $l_2 < v < l_3 \ldots$

4.1.2 Variable d'état

Une variable d'état est caractérisée par six valeurs : la valeur courante (x) et la valeur (x_{-1}) précédente de la variable, la valeur courante $(\dot{x})$ et précédente $(\dot{x}_{-1})$ de la dérivée première et la valeur courante $(\ddot{x})$ et précédente $(\ddot{x}_{-1})$ de la dérivée seconde.

4.1.3 Comportement qualitatif

Un comportement qualitatif est une succession de contextes d'exécution ; chaque contexte d'exécution est un ensemble de variables d'états associées à une valeur qualitative.

4.2 Modèle d'exécution énuméré

Dans ce modèle d'exécution [59], on ne considère que les changements continus des variables d'états et de leurs dérivées premières. La dérivée seconde, qui est le résultat des forces d'entrée du système, peut ne pas être continue. Par exemple, la valeur d'une variable d'état ne peut pas changer de *Négative* à *Positive* sans passer par *Nulle*. Aussi, la valeur ne peut pas changer de *Négative* à *Nulle* à moins que la dérivée

première ne soit *Positive*. Ces contraintes de continuité et de dérivabilité peuvent être modélisées sous forme de machines à états comme illustré dans la Figure 4.1 qui modélise l'évolution de la valeur de la variable d'état.

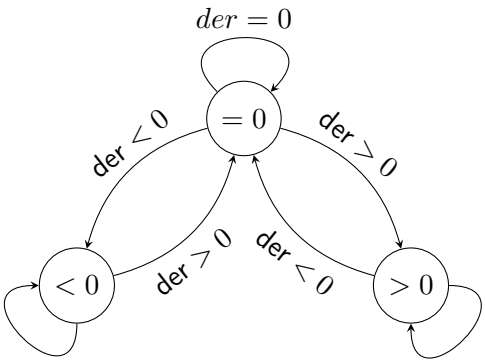

FIGURE 4.1 – Évolution qualitative d'une variable d'état

Un automate similaire contrôle les changements de la dérivée première en suivant le même principe évoqué précédemment mais en prenant en considération la dérivée seconde. La dérivée première ne peut pas changer de *Positive* à *Nulle* à moins que la dérivée seconde ne soit *Négative* comme le montre la Figure 4.2.

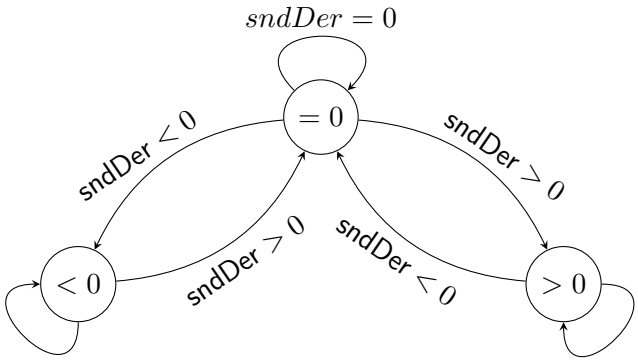

FIGURE 4.2 – Évolution qualitative de la dérivée première d'une variable d'état

4.2.1 Calcul du comportement qualitatif

Pour calculer le comportement qualitatif d'une variable d'état, on observe les changements des valeurs de la variable. Par exemple, quand la dérivée seconde est *Négative* et la dérivée première est *Nulle*, on détecte un *Maximum*. La valeur courante de la variable d'état et la valeur courante de sa dérivée première ne sont pas suffisantes pour faire la différence entre un *Maximum* et *Start Increase* ou *Start Decrease* après

un état *Constant*. C'est pourquoi, on modélise chaque variable d'état continue par cinq valeurs : la valeur courante (x) et la valeur (x_{-1}) précédente de la variable, la valeur courante ($\dot{x}$) et précédente ($\dot{x}_{-1}$) de la dérivée première et la valeur courante ($\ddot{x}$) de la dérivée seconde. En prenant en considération $\dot{x}_{-1}$, $\dot{x}$ et $\ddot{x}$, on identifie 13 états qualitatifs :

Constant quand $\dot{x}_{-1}$, $\dot{x}$ et $\ddot{x}$ sont nulles ;

FlexStartIncrease si $\dot{x}_{-1} = 0$, $\dot{x} = 0$ et $\ddot{x} > 0$. La dérivée première n'est pas encore positive, mais la dynamique est en transition pour augmenter ;

FlexStartDecrease quand $\dot{x}_{-1} = 0$, $\dot{x} = 0$ et $\ddot{x} < 0$. La dérivée première n'est pas encore négative, mais la dynamique est en transition pour diminuer ;

StartIncrease quand $\dot{x}_{-1} = 0$ et $\dot{x} > 0$. La dérivée première vient juste de devenir positive. Il n'y a pas de conditions sur la dérivée seconde qui peut redevenir nulle ;

StartDecrease quand $\dot{x}_{-1} = 0$ et $\dot{x} < 0$. La dérivée première vient juste de devenir négative. Il n'y a pas de conditions sur la dérivée seconde qui peut redevenir nulle ;

Increase quand $\dot{x}_{-1} > 0$ et $\dot{x} > 0$. La dérivée première est positive ;

Decrease , quand $\dot{x}_{-1} < 0$ et $\dot{x} < 0$. La dérivée première est négative ;

Maximum quand $\dot{x}_{-1} > 0$, $\dot{x} = 0$ et $\ddot{x} < 0$. Ces trois conditions sont nécessaires pour distinguer cet état parmi d'autres états qualitatifs comme les points d'inflexion ;

Minimum quand $\dot{x}_{-1} < 0$, $\dot{x} = 0$ et $\ddot{x} > 0$. Ces trois conditions sont nécessaires pour distinguer cet état parmi d'autres états qualitatifs comme les points d'inflexion ;

FlexIncrease quand $\dot{x}_{-1} > 0$, $\dot{x} = 0$ et $\ddot{x} > 0$. Point d'inflexion pendant une phase d'augmentation ;

FlexDecrease quand $\dot{x}_{-1} < 0$, $\dot{x} = 0$ et $\ddot{x} < 0$. Point d'inflexion pendant une phase de diminution ;

StopIncrease quand $\dot{x}_{-1} > 0$, $\dot{x} = 0$ et $\ddot{x} = 0$. La variable d'état atteint un plateau à la fin de la phase d'augmentation ;

StopDecrease quand $\dot{x}_{-1} < 0$, $\dot{x} = 0$ et $\ddot{x} = 0$. La variable d'état atteint un plateau à la fin de la phase de diminution.

En prenant en considération x_{-1} et x, il est possible de distinguer des sous états de ces états qualitatifs, par exemple atteindre un maximum positif quand $x_{-1} = 0$ et $x > 0$ ou atteindre un minimum nul quand $x_{-1} > 0$ and $x = 0$. Il existe des

cas impossibles qualitativement. Deux sont dus à la continuité de la variable : la conjonction de $x_{-1} < 0$ and $x > 0$ et la conjonction de $x_{-1} > 0$ and $x < 0$ sont impossibles. Deux autres sont dus à la continuité de la dérivée première (pas de points angulaires). Dix autres sont dus au fait que chaque variable doit être consistante avec la valeur précédente de la dérivée première. Par exemple, avec $x_{-1} = 0$ et $\dot{x}_{-1} = 0$, on a nécessairement $x = 0$. Partant de 0 avec une dérivée première nulle, la variable ne peut pas changer.

4.2.2 Exemple illustratif : la balle rebondissante

Cette technique a été appliquée sur un exemple type de système hybride : la balle rebondissante. Dans ce système, on a une seule variable d'état, la hauteur de la balle au dessus du sol, noté z. La manière habituelle de modéliser la balle rebondissante est de considérer qu'elle a un seul état dans lequel la balle est toujours en chute libre avec $\ddot{z} = -g$ et $g = 9.81m.s^{-2}$. Le rebond est modélisé par une seule transition déclenchée quand la balle atteint le sol ($z = 0$). La vitesse de la balle est inversée par un facteur d'amortissement $0 < c \leq 1$. Ce modèle et son abstraction qualitative sont montrés dans la Figure 4.3.

FIGURE 4.3 – Automates hybride (à gauche) et qualitatif (à droite) de la balle rebondissante

Le comportement numérique de ce modèle de la balle, obtenu avec l'environnement Ptolemy II [29] est illustré par la Figure 4.4.

Ce type de modèle est une abstraction de la réalité physique. Si la vitesse de la balle change instantanément de signe, il va y avoir un échange d'une quantité d'énergie finie pendant un laps de temps infiniment court entre la balle et son environnement. Ce qui correspond à une puissance infinie et est donc impossible dans la réalité.

4.2.3 Simulation qualitative du modèle hybride brut

En effectuant la simulation qualitative de ce modèle qui n'est pas physique en utilisant notre modèle d'exécution, Diversity trouve un dead-lock comme le montre la Figure 4.5 :

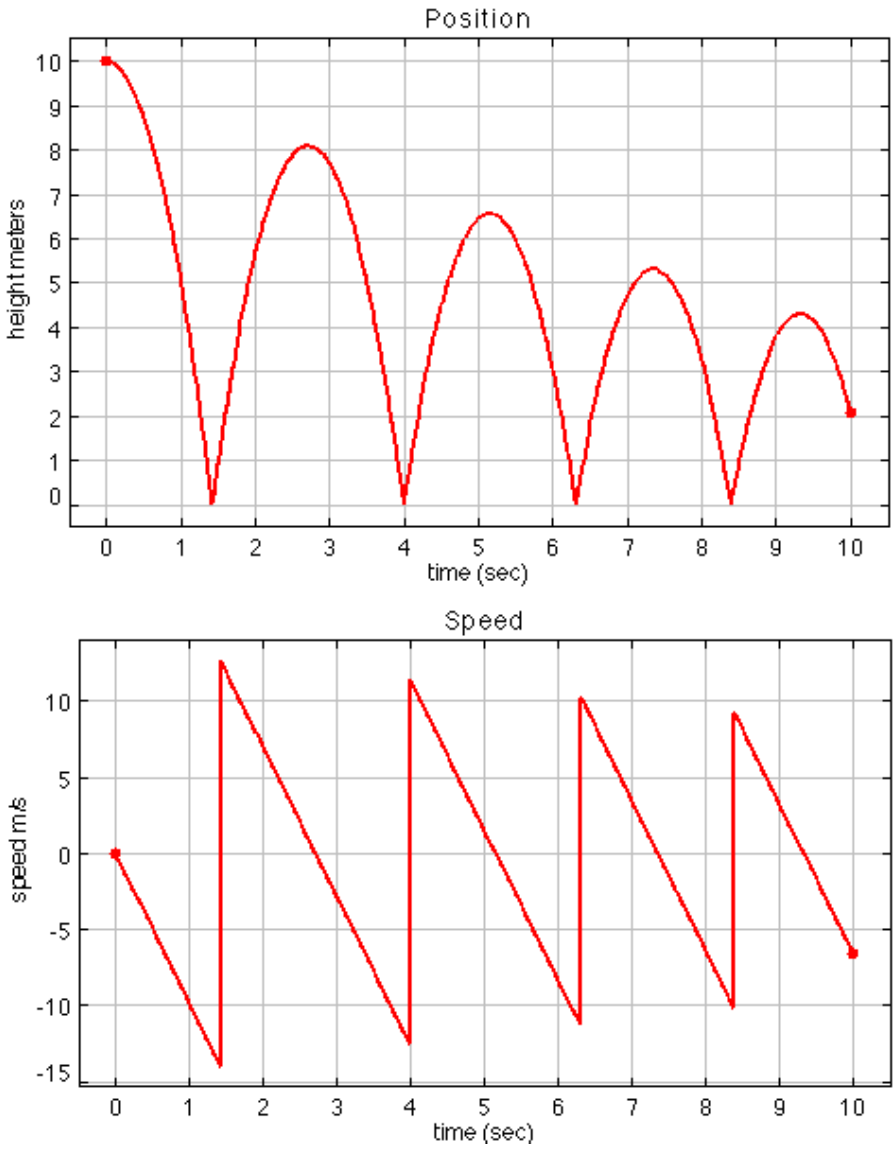

FIGURE 4.4 – Comportement quantitatif de la balle rebondissante

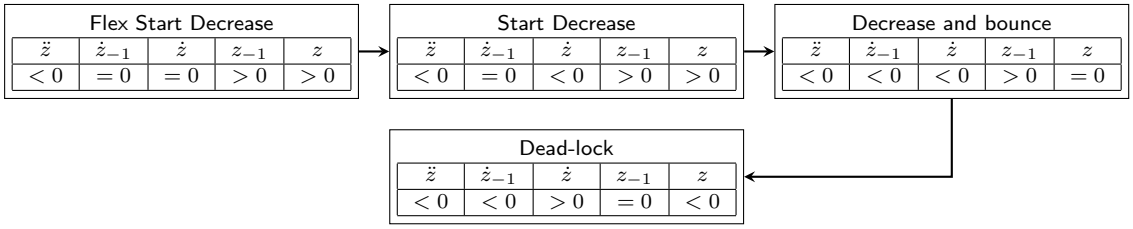

FIGURE 4.5 – Simulation qualitative du modèle brut

Ce dead-lock apparaît parce que le modèle de la balle force la dérivée première à changer brutalement de négative à positive sans passer par zéro. Dans ce cas, la ma-

chine à états modélisant la dérivée première va être bloquée dans son état “négative”. Notre modèle d'exécution force la continuité et la dérivabilité des états physiques, mais le modèle de la balle ne respecte pas ces propriétés. Pour gérer cette discontinuité, nous proposons un ajustement de notre modèle d'exécution

4.2.4 Ajuster le Modèle d'exécution

La première solution est d'ajuster notre modèle d'exécution pour autoriser ce genre de transition qui ne sont pas physiques. Avec ce modèle d'exécution ajusté, l'état qualitatif change à “Increase” après le rebond, puis “Maximum”, puis à “Start Decrease” avant de reboucler à l'état qualitatif “Decrease” comme le montre la Figure 4.6. Ceci permet à la balle de rebondir et de détecter le maximum de sa hauteur, mais son minimum de hauteur atteint pendant le rebond n'est pas détecté.

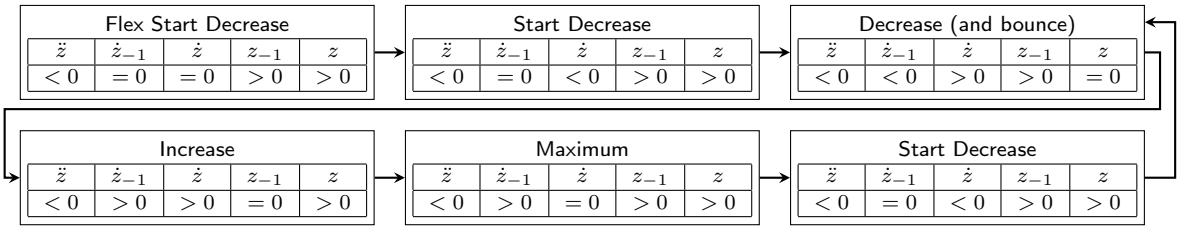

FIGURE 4.6 – Simulation qualitative avec discontinuités

Le problème ici est que la détection des états qualitatifs se base sur la continuité et la dérivabilité de la variable d'état. Le changement discontinu de la vitesse de la balle empêche la détection du minimum.

4.2.5 Ajuster le modèle

Une autre solution est d'ajuster le modèle de la balle. Dans ce cas le modèle de la balle rebondissante peut être automatiquement ajusté en remplaçant la transition de rebond par une série de transitions. La méthode qui permet cet ajustement est la suivante :

— changer la dérivée de < 0 à > 0 est illégal. L'unique chemin légal est d'aller de < 0 à $= 0$ puis de $= 0$ à > 0, donc on remplace la transition illégale par deux transitions ;

— changer la dérivée de < 0 à $= 0$ puis de $= 0$ à > 0 demande que la dérivée seconde soit positive, c'est pourquoi on met la dérivée seconde positive pendant le rebond (la dérivée seconde peut être discontinue).

La Figure 4.7 montre l'automate qui modélise cette méthode, les états ajoutés sont en gris. Le premier état ajouté, quand on atteint $z = 0$, met la dérivée seconde à "Positive" pour changer la dérivée de "Négative" à "Nulle". Quand la dérivée première devient nulle, on atteint le deuxième état ajouté et la dérivée première devient positive comme demandé par la transition initiale qui n'est pas physique. Puis on repart de nouveau vers l'état de chute libre de la balle avec une vitesse inversée.

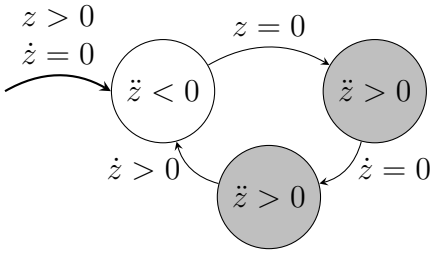

FIGURE 4.7 – Modèle de la balle rebondissante ajusté

Avec ce modèle ajusté de la balle rebondissante et le modèle physique de la simulation qualitative, on obtient le comportement qualitatif montré dans la Figure 4.8.

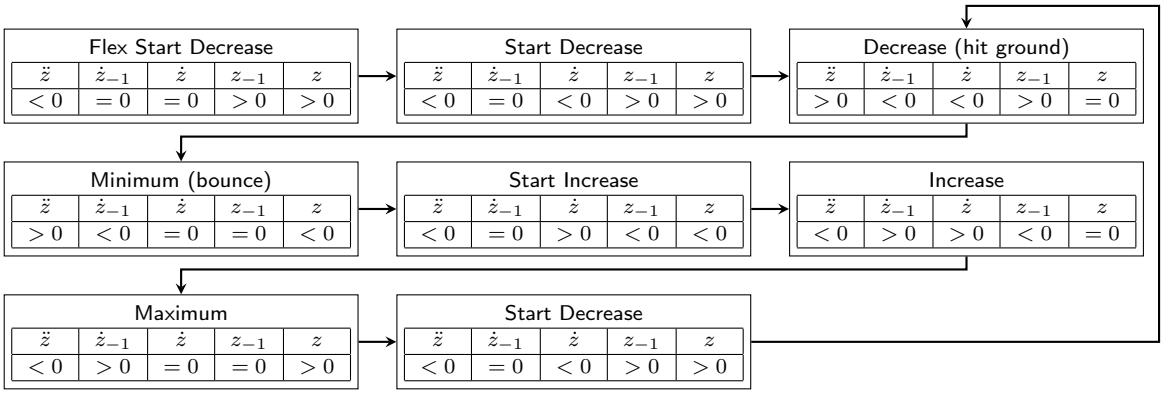

FIGURE 4.8 – Simulation qualitative du modèle ajusté

La simulation qualitative détecte maintenant le minimum de la balle quand elle atteint le sol et rebondit, et le maximum quand la balle atteint sa haute position.

4.2.6 Discussion sur le modèle d'exécution énuméré

Dans cette simulation qualitative, on observe un artefact inattendu : la hauteur de la balle au dessus du sol devient négative pendant la phase de rebond. Ceci est dû au fait que la dérivée première reste négative pendant trois états de descente. $\dot{z}$ devient nulle quand la balle atteint le sol. Ainsi, la hauteur passe de nulle à négative.

La condition $z = 0$ pour déclencher le rebond, correspond à la surface de la balle qui heurte le sol. Par conséquent, la variable d'état z représente la hauteur du centre de la balle au dessus du sol. z devient nulle au moment où le sol commence à pousser la balle vers le haut. L'action de mettre $\ddot{z} > 0$ représente le résultat de cette réaction. Ceci rend la balle plus lente et finit par l'arrêter ($\dot{z}$ passe de négative à nulle). $\dot{z}$ est encore négative, z est en cours de descente, c'est pourquoi elle passe de nulle à négative. Pour le système physique, ceci correspond à la déformation de la balle qui convertit l'énergie cinétique en énergie élastique (et en chaleur si la collision n'est pas élastique). Quand la balle est déformée contre le sol, son centre d'inertie est en dessous de sa position quand elle a atteint le sol, ce qui rend z négative. En illustrant notre approche avec l'exemple de la balle rebondissante, on montre qu'un seul comportement qualitatif est trouvé. Parmi les autres comportements trouvés, il y a ceux qui sont physiques. Par exemple, après le rebond la balle part vers le haut indéfiniment sans atteindre un maximum. Ce comportement est possible si la balle atteint une vitesse très grande pour s'échapper. Dans notre cas, on ne connaît pas la valeur exacte de la vitesse de la balle, c'est pourquoi on ne peut pas dire en utilisant la simulation qualitative si la balle va atteindre un maximum ou va rebondir vers l'infini. Un autre comportement plus troublant : la balle peut descendre indéfiniment sans atteindre le sol. Ce comportement est possible dans notre modèle d'exécution qualitatif parce qu'il est de premier ordre. Quand la balle est en train de descendre on sait que la hauteur de la balle est en phase de chute, mais on n'a pas l'information que sa vitesse est en train d'augmenter en terme d'ordre de grandeur. Sans cette information, il est possible que la balle aille de plus en plus lentement vers le sol sans le toucher. Pour éliminer les comportements qui ne reflètent pas la réalité physique de la simulation qualitative, on va proposer un autre modèle de calcul de second ordre.

4.3 Modèle d'exécution symbolique inspiré de la méthode d'Euler

Nous présentons un nouveau modèle d'exécution symbolique [58] dans lequel nous intégrons les dérivées des variables d'états d'une manière symbolique. Nous utilisons une simple intégration d'Euler puisque nous ne calculons pas des valeurs numériques exactes. Nous modélisons chaque variable d'état continue du système par six valeurs : les valeurs courante (x) et précédente ($x_{\text{-}1}$) de la variable, les valeurs courante ($\dot{x}$) et précédente ($\dot{x}_{\text{-}1}$) de la première dérivée et la valeur courante de la dérivée seconde ($\ddot{x}$). Dans cette version de modèle de calcul, nous avons ajouté la valeur précédente

de la dérivée seconde ($\ddot{x}_{\text{-}1}$). Nous avons besoin de ($\ddot{x}_{\text{-}1}$) pour intégrer la dérivée première. L'intégration symbolique avec la méthode d'Euler, en prenant un pas unitaire d'intégration, donne $x = x_{\text{-}1} + \dot{x}_{\text{-}1}$ et $\dot{x} = \dot{x}_{\text{-}1} + \ddot{x}_{\text{-}1}$. Les contraintes de continuité et de dérivabilité qui reflètent la physique du phénomène sont modélisées sous forme de machine à états comme le montre la Figure 4.9 pour le contrôle d'une variable d'état.

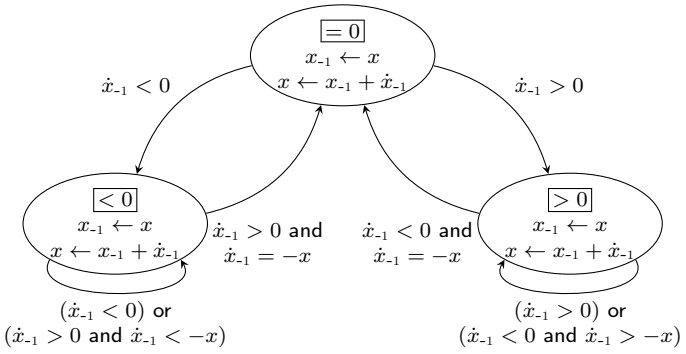

FIGURE 4.9 – Évolution qualitative avec intégration symbolique

Contrairement au modèle d'exécution énuméré qui était non déterministe (lorsqu'on est dans un état > 0, si la dérivée première est *Négative*, on peut soit aller vers un état $= 0$, soit rester dans l'état courant), ce modèle d'exécution symbolique est déterministe : si on est dans un état > 0 de x, si $\dot{x}_{\text{-}1} < 0$ et $\dot{x}_{\text{-}1} = \text{-}x$, on va dans un état $= 0$ de x, si $\dot{x}_{\text{-}1} > 0$ ou $\dot{x}_{\text{-}1} < 0$ et $\text{-}\dot{x}_{\text{-}1} < x$, on reste dans l'état courant de x.

4.3.1 Implémentation du modèle d'exécution symbolique dans Diversity

Ce modèle d'exécution symbolique se base sur cinq automates (voir l'Annexe A page 121 pour les automates de la balle rebondissante) : le premier automate dans lequel l'utilisateur modélise le système et spécifie les différentes valeurs de la dérivée seconde ; le deuxième automate suit les changements de la dérivée seconde ($\ddot{x}$) spécifiés par l'automate du système ; le troisième automate suit les changements de la dérivée première ($\dot{x}$) intégrée symboliquement par la valeur précédente de la dérivée première et la valeur précédente de la dérivée seconde ; le quatrième automate suit les valeurs qualitatives de la variable d'état (x), qui est symboliquement intégrée par sa valeur précédente et la valeur précédente de la dérivée première ; finalement le cinquième automate calcule la variation qualitative de la variable en observant les automates de la valeur de la variable, ses dérivées et les valeurs précédentes de ses dérivées.

L'automate du système s'exécute en premier. Ceci peut changer les valeurs symboliques calculées par les automates d'intégration de $\dot{x}$ et x. En effet, partant d'un

état positif de x avec une valeur symbolique $x\sharp 1$, la présence d'une transition dans l'automate du système avec une garde $x = 0$, changera la valeur de x de $x\sharp 1$ à 0. L'automate du système doit réagir à un changement de valeur de x et ne pas déclencher un tel changement. La valeur de x ne doit être changée que par l'automate qui intègre symboliquement $\dot{x}$. Le rôle de l'automate du système est de spécifier la valeur de $\ddot{x}$ pour que les autres automates puissent calculer $\dot{x}$ et x. Pour renforcer le signe des valeurs symboliques, nous ajoutons une garde après l'intégration symbolique à l'entrée des états comme le montre la Figure 4.10. Ainsi, l'automate du système respecte les valeurs symboliques calculées par les autres automates de la dérivée première et de la valeur des variables d'état. Il est un stimulateur pour les autres automates en fournissant la valeur courante de la dérivée seconde, qui déclenche le calcul des valeurs symboliques, mais aussi est un observateur des différentes valeurs symboliques calculées par le mécanisme de l'intégration symbolique implémenté dans les autres automates.

$$x_{\text{-}1} = x$$
$$x = x_{\text{-}1} + \dot{x}_{\text{-}1}$$
$$\text{guard } (\ x > 0 \ )$$

FIGURE 4.10 – Affectation de valeurs symboliques à l'aide de gardes dans Diversity

4.3.2 Calcul des comportements qualitatifs de second ordre

Dans ce nouveau modèle d'exécution, on a ajouté plusieurs états qualitatifs pour faciliter l'interprétation par l'utilisateur des différents comportements qualitatifs. Ces nouveaux états qualitatifs correspondent à une variation qualitative d'ordre supérieur vu qu'elle qualifie la variation de la dérivée première. On identifie quatre nouveaux états qualitatifs :

Increasingly_ Increase, si $\dot{x}_{\text{-}1} > 0$ et $\ddot{x}_{\text{-}1} > 0$;

Decreasingly_ Increase, si $\dot{x}_{\text{-}1} > 0$ et $\ddot{x}_{\text{-}1} < 0$;

Increasingly_ Decrease, si $\dot{x}_{\text{-}1} < 0$ et $\ddot{x}_{\text{-}1} < 0$;

Decreasingly_ Decrease, si $\dot{x}_{\text{-}1} < 0$ et $\ddot{x}_{\text{-}1} > 0$.

4.3.3 Exemple illustratif : la balle rebondissante

Cette technique a été appliquée sur l'exemple de la balle rebondissante présenté dans la sous-section 4.2.2 page 41. Nous obtenons le comportement qualitatif illustré par la Figure 4.11. Dans cette figure, nous voyons clairement que la vitesse de la

balle, lorsqu'elle est en phase de descente, est en train d'augmenter. Ceci est modélisé par l'état *Increasingly_ Decrease*. Nous voyons aussi que sa vitesse est en train de diminuer en phase de rebond, ce qui est modélisé par l'état *Decreasingly_ Increase*.

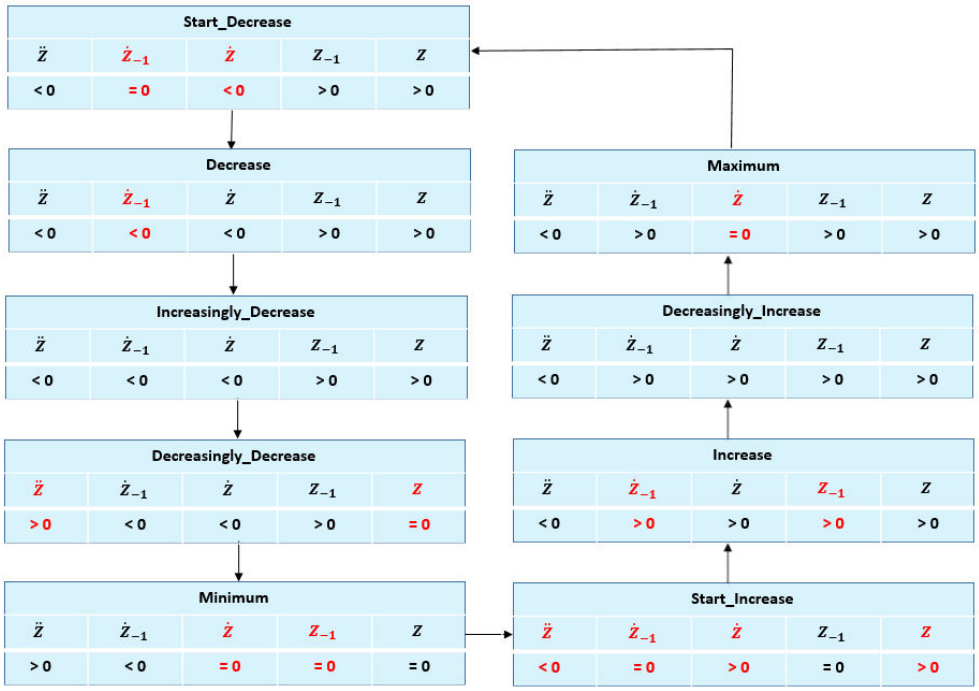

FIGURE 4.11 – Comportement qualitatif de la balle rebondissante

4.3.4 Exemple illustratif : Le circuit RC

Cette technique a été appliquée sur l'exemple du circuit RC expliqué dans la Figure 4.12. Dans le circuit RC, on a deux variables couplées : la tension U_r aux bornes de la résistance R et la tension U_c aux bornes du condensateur C.

4.3.4.1 Calcul des comportements qualitatifs de premier ordre

Dans ce système, nous avons besoin de considérer uniquement la dérivée première de la tension, qui est proportionnelle à l'intensité du courant à cause du condensateur. Le changement de phase entre la charge et la décharge force le courant à être positif pendant la charge et négatif pendant la décharge. Nous ne disposons pas d'informations sur la dérivée seconde des tensions. Notre approche peut s'adapter à ce genre de situation en ne prenant en considération que la dérivée première. Quelques états qualitatifs comme **StartIncrease**, **StartDecrease**, **Increase** et **Decrease** sont inchangés puisqu'ils ne dépendent pas de la dérivée seconde, alors que

Phase de charge

$$\begin{aligned} E &= U_r + U_c \\ U_r(t) &= E.e^{\frac{-t}{RC}} \\ U_c(t) &= E(1 - e^{\frac{-t}{RC}}) \end{aligned}$$

Phase de décharge

$$\begin{aligned} 0 &= U_r + U_c \\ U_r(t) &= -E.e^{\frac{-t}{RC}} \\ U_c(t) &= E.e^{\frac{-t}{RC}} \end{aligned}$$

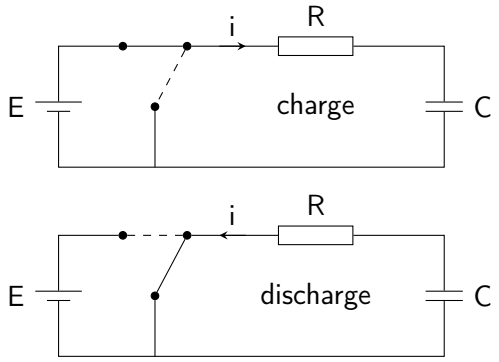

FIGURE 4.12 – Phases de charge et de décharge du condensateur du circuit RC

d'autres comme **StopIncrease** et **StopDecrease** ne vont pas être identifiés puisqu'ils ont besoin d'une dérivée seconde nulle. Dans ce modèle de premier ordre, on a ajouté deux états qualitatifs qui modélisent la discontinuité de la dérivée première : **Angular_Increase**, quand $\dot{x}_{\text{-1}} < 0$, $\dot{x} > 0$, et **Angular_Decrease**, quand $\dot{x}_{\text{-1}} > 0$, $\dot{x} < 0$.

Dans cet exemple, la tension U_c est continue alors que la tension U_r est discontinue. Dans la phase de charge, U_r diminue de E à 0 et U_c augmente de 0 à E. Dans la phase de décharge, U_r augmente de $-E$ à 0 et la tension U_c diminue de E à 0. Puisqu'on n'est pas intéressé par les valeurs exactes dans la simulation qualitative, $+E$ et $-E$ vont être considérés respectivement comme > 0 et < 0 symboliquement.

Comme nous avons fait avec le modèle de la balle rebondissante, le circuit RC peut être automatiquement ajusté pour assurer la continuité en remplaçant la transition de la phase de charge vers la phase de décharge et vice versa :

— changer la valeur de U_r de > 0 à < 0 est illégal. L'unique chemin légal est de passer de > 0 à $= 0$ et puis de $= 0$ à < 0. Donc nous remplaçons la transition illégale par ces deux transitions ;

— changer la valeur de > 0 à $= 0$ et de $= 0$ à < 0 exige une dérivée première négative, c'est pourquoi nous mettons la dérivée première négative pendant le changement de phase.

On fait la même chose pour changer la valeur de U_r de < 0 à > 0 quand on passe de la phase de décharge vers la phase de charge, mais avec une dérivée première positive.

4.3.4.2 Machine à états du circuit RC

Les équations du circuit RC sont modélisées par une machine à états avec quatre états et cinq transitions comme le montre la Figure 4.13 :

— la transition t_0 met les valeurs initiales des variables d'états U_c à $= 0$ et U_r à > 0, et met la dérivée première de U_c à > 0 et U_r à < 0 ;

— la transition t_1 déclenche le changement pour la phase de décharge et met la dérivée première de U_r à < 0 et celle de U_c à < 0 ;

— la transition t_2 déclenche le changement pour la phase de décharge et met la dérivée première de U_r à > 0 et celle de U_c à < 0 ;

— la transition t_3 déclenche le changement pour la phase de charge et met la dérivée première de U_r à > 0 et celle de U_c à > 0 ;

— la transition t_4 déclenche le retour à la phase de charge et met la dérivée première de U_r à < 0 et celle de U_c à > 0.

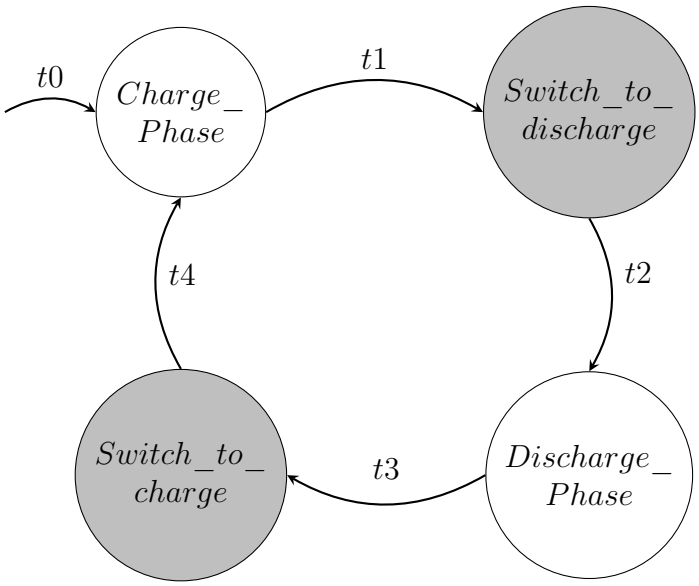

FIGURE 4.13 – Automate du circuit RC

4.3.4.3 Comportement qualitatif calculé par Diversity

Nous voyons clairement dans la Figure 4.14 que les tensions U_r au borne de la résistance R et U_c au borne du condensateur C varient d'une manière opposée :

— dans la *Charge_phase*, U_r est dans un *Decreasing_state* alors que U_c est dans un *Increasing_state* ;

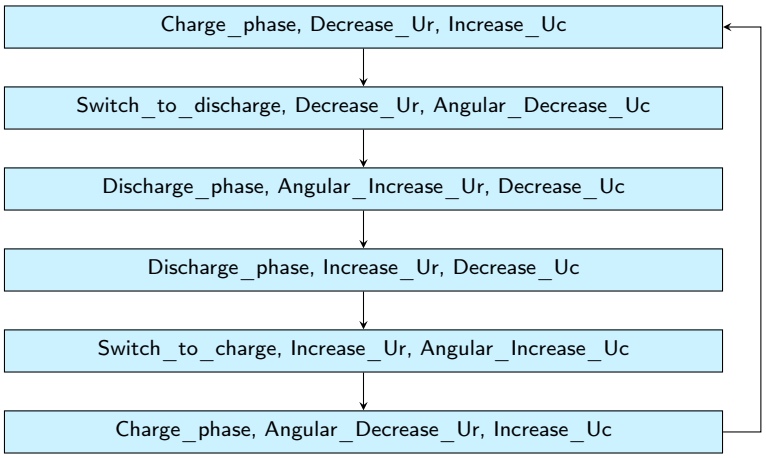

FIGURE 4.14 – Comportement qualitatif du circuit RC

— dans le *Switch_to_discharge*, U_r est encore dans un *Decreasing_state* pour atteindre $-E$ alors que U_c est dans un *Angular_Decreasing_state* parce que le condensateur commence à se décharger ;

— dans la *Discharge_phase*, U_r est dans un *Increasing_state* alors que U_c est dans un *Decreasing_state* ;

— dans le *Switch_to_charge*, U_r est encore dans un *Increasing_state* pour atteindre $+E$ alors que U_c est dans un *Angular_Increasing_state* parce que le condensateur commence à se charger.

4.3.5 Discussion sur le modèle d'exécution symbolique inspiré de la méthode d'Euler

Parmi les comportements qualitatifs trouvés par Diversity pour la balle rebondissante, on n'a présenté que le comportement qui correspond au modèle physique de la balle. D'autres comportements ont été trouvés (voir l'arbre de comportements qui se trouve dans l'Annexe A page 120). Dans la version du modèle d'exécution présenté dans la sous-section 4.2 page 38, on a décrit deux comportements qui étaient impossibles et devraient être éliminés. Grâce à ce nouveau modèle d'exécution symbolique qui calcule les variations qualitatives de la dérivée première comme *Increasingly_Increase*, on a réussi à les éliminer. Par exemple, quand la balle est en chute, sa vitesse est négative et décroissante (parce que la dérivée seconde est négative), donc la vitesse est négative et croissante en terme d'ordre de grandeur. Le nouveau modèle d'exécution identifie la variation qualitative de la hauteur de la balle avec *Increasingly_Increase*. Commençant à partir d'une position positive, la hauteur de la balle

devient donc nulle à un certain moment. Ceci élimine le comportement de la balle dans lequel elle chute indéfiniment sans toucher le sol. Il reste encore des comportements trouvées par Diversity qui diffèrent par quelques séquences d'états. Ceci est dû à la manière dont Diversity détecte la *Redondance*. Pendant l'exécution symbolique d'un modèle, Diversity construit un arbre, chaque branche correspond à un choix de valeurs symboliques des variables. Quand Diversity trouve un contexte d'exécution déjà rencontré dans l'arbre, il arrête l'exécution de cette branche et marque l'état déjà rencontré avant. Ceci transforme l'arbre en un graphe d'exécution qui permet d'identifier des comportements infinis dans une structure finie. La redondance peut être configurée suivant les variables sur lesquelles nous voulons détecter des comportements similaires. Ceci crée plusieurs chemins d'exécution pour le même comportement physique. Ces chemins d'exécution rendent les résultats plus difficiles à analyser. C'est pourquoi, nous avons mis dans notre méthodologie, un module qui permet de filtrer les résultats de Diversity dans le but de garder un seul chemin pour chaque comportement physique des systèmes hybrides.

4.4 Modèle d'exécution symbolique avec contraintes qualitatives

Dans les premières phases de spécification, les équations différentielles ne sont pas toujours présentes ou sont incomplètes. Dans le but de proposer à l'utilisateur un moyen d'exprimer un comportement physique sans équations différentielles au sens du modèle mathématique continu, nous présentons un nouveau modèle d'exécution [57] dans lequel nous lions les dérivées et les valeurs de deux variables d'états. Ce modèle d'exécution se base aussi sur les automates définis dans la sous-section 4.3.1 page 46. Nous avons défini quatre opérateurs inspirés de [14] et de la notion de causalité notamment présente dans le formalisme des bond graphs :

— *causally proportional*, **CPROP** ;

— *proportional*, **PROP** ;

— *causally inversely proportional*, **CIPROP** ;

— *inversely proportional*, **IPROP**.

On utilisera **thres** pour spécifier les seuils de la variable d'état.

4.4.1 CPROP

Cet opérateur modélise la proportionnalité causale entre les valeurs de deux variables d'état différentes. Par exemple, x **thres** T_c **is CPROP** y **thres** T_s signifie que le signe de $x - T_c$ doit être le même que le signe de $y - T_s$, mais une correction de x ne sera faite que si on détecte un changement de signe de $y - T_s$. Nous présenterons cette correction en détail dans la sous-section 4.4.6 page 55. Donc si $y_{\text{-}1} = T_s$, $y > T_s$ et $x = T_c$, on doit ajuster la valeur de x à $> T_c$. Mais si x devient $< T_c$ alors que y reste $> T_s$, on ne fait pas une correction sur x. Dans la version du modèle d'exécution expliquée dans la section 4.3 page 45, nous avions $x = x_{\text{-}1} + \dot{x}_{\text{-}1}$. La relation ajoute un terme correctif ${x_{Cor}}^y$ pour corriger la valeur de x quand $y - T_s$ change de signe. Par conséquent, on aura $x = x_{\text{-}1} + \dot{x}_{\text{-}1} + {x_{Cor}}^y$.

4.4.2 PROP

Cet opérateur modélise la proportionnalité de deux variables d'état différentes. Par exemple, $\dot{x}$ **thres** 0 **is PROP** $\dot{y}$ **thres** 0 signifie que $\dot{x}$ et $\dot{y}$ doivent être de même signe. Contrairement à **CPROP**, un changement qualitatif de $\dot{y}$ n'est pas nécessaire pour assurer la proportionnalité. Ici, $\dot{x}$ doit toujours être corrigé pour être de même signe que $\dot{y}$. Dans la version du modèle d'exécution expliquée dans la section 4.3 page 45, nous avions $\dot{x} = \dot{x}_{\text{-}1} + \ddot{x}_{\text{-}1}$. La relation ajoute un terme correctif ${\dot{x}_{Cor}}^y$ à $\dot{x}$ pour corriger la valeur de $\dot{x}$ quand $\dot{x}$ n'a pas le même signe que $\dot{y}$. Par conséquent, on aura $\dot{x} = \dot{x}_{\text{-}1} + \ddot{x}_{\text{-}1} + {\dot{x}_{Cor}}^y$.

4.4.3 CIPROP

Cet opérateur modélise la proportionnalité causale inverse entre les valeurs de deux variables d'état. Par exemple, $\ddot{x}$ **thres** 0 **is CIPROP** y **thres** T_s signifie que $\ddot{x}$ et $y - T_s$ doivent être de signe opposé. Une correction n'est nécessaire que si nous détectons un changement qualitatif dans $y - T_s$. Si $y_{\text{-}1} = T_s$, $y > T_s$ et $\ddot{x} \geq 0$, on doit corriger la valeur de $\ddot{x}$ pour la rendre négative. Dans la version du modèle d'exécution expliquée dans la section 4.3 page 45, nous avions $\ddot{x} = \ddot{x}_{\text{-}1}$. La relation ajoute un terme correctif ${\ddot{x}_{Cor}}^y$ à $\ddot{x}$ dans le but de corriger la valeur de $\ddot{x}$ quand $y - T_s$ change de signe. Par conséquent, on aura $\ddot{x} = \ddot{x}_{\text{-}1} + {\ddot{x}_{Cor}}^y$.

4.4.4 IPROP

Cet opérateur modélise la proportionnalité inverse entre les valeurs de deux variables d'état différentes. Par exemple, $\dot{x}$ **thres** 0 **is IPROP** $\dot{y}$ **thres** 0 signifie

que $\dot{x}$ et $\dot{y}$ doivent être de signe opposé et la correction de $\dot{x}$ doit être faite même sans le changement qualitatif de $\dot{y}$. Si $\dot{y} < 0$ et $\dot{x} \geq 0$, on doit corriger la valeur de $\dot{x}$ pour la rendre négative. On utilisera le terme correctif ${\dot{x}_{Cor}}^{y}$ pour corriger la valeur de $\dot{x}$ quand $\dot{x}$ est de même signe que $\dot{y}$.

4.4.5 Évolution qualitative d'une variable d'état

Avec ces règles, la valeur qualitative d'une variable d'état est contrôlée par une machine à états. Si on considère par exemple x **thres** 0 **is CPROP** y **thres** 0, le terme correctif ${x_{Cor}}^{y}$ va être ajouté à l'intégration symbolique de x comme le montre la Figure 4.15. Dans l'état $= 0$ de x, si $x_{\text{-}1} + \dot{x}_{\text{-}1} + {x_{Cor}}^{y} > 0$, l'état suivant sera > 0, si $x_{\text{-}1} + \dot{x}_{\text{-}1} + {x_{Cor}}^{y} < 0$, l'état suivant sera < 0.

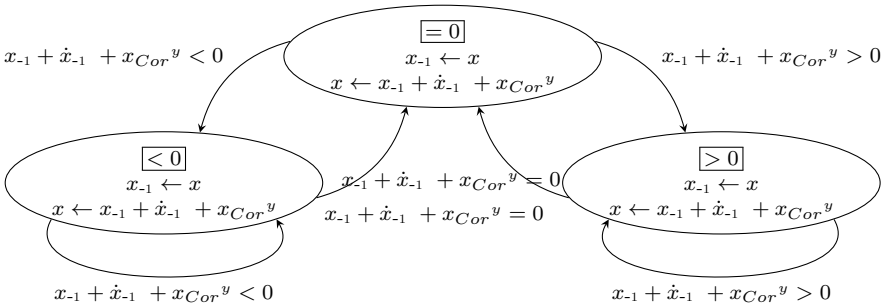

FIGURE 4.15 – Évolution qualitative d'une variable d'état avec contraintes qualitatives

Un automate similaire contrôle les changements de la dérivée première en considérant sa valeur précédente ($\dot{x}_{\text{-}1}$), la valeur précédente de la dérivée seconde ($\ddot{x}_{\text{-}1}$) et les termes correctifs issus des différentes relations liées à la dérivée première. Si on considère par exemple $\dot{x}$ **thres** 0 **is CPROP** y **thres** 0, le terme correctif ${\dot{x}_{Cor}}^{y}$ s'ajoute à l'intégration symbolique de $\dot{x}$ comme le montre la Figure 4.16. Dans l'état $= 0$ de $\dot{x}$, si $\dot{x}_{\text{-}1} + \ddot{x}_{\text{-}1} + {\dot{x}_{Cor}}^{y} > 0$, l'état suivant sera > 0, si $\dot{x}_{\text{-}1} + \ddot{x}_{\text{-}1} + {\dot{x}_{Cor}}^{y} < 0$, l'état suivant sera < 0.

Un autre automate similaire contrôle les changements de la dérivée seconde en considérant sa valeur précédente ($\ddot{x}_{\text{-}1}$) et les termes correctifs issus des différentes relations liées à la dérivée seconde. Si on considère par exemple $\ddot{x}$ **thres** 0 **is CPROP** y **thres** 0, le terme correctif ${\ddot{x}_{Cor}}^{y}$ s'ajoute à l'intégration symbolique de $\ddot{x}$ comme le montre la Figure 4.17. Normalement l'intégration symbolique de $\ddot{x}$ est égale à $\ddot{x}_{\text{-}1} + {\ddot{x}_{Cor}}^{y}$. Mais nous nous limitons au second ordre, nous n'utilisons pas $\dddot{x}$ parce que généralement les ingénieurs se limitent au second ordre dans leur

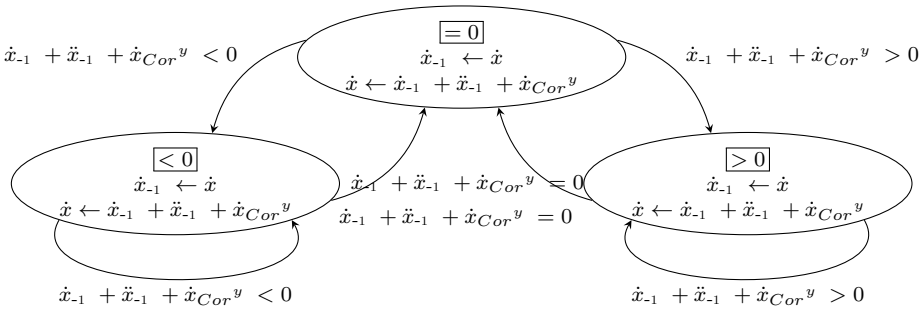

FIGURE 4.16 – Évolution qualitative de la dérivée première avec contraintes qualitatives

modélisation. Dans l'état $= 0$ de $\ddot{x}$, si $\ddot{x}_{\text{-}1} + \ddot{x}_{Cor}{}^{y} > 0$, l'état suivant sera > 0, si $\ddot{x}_{\text{-}1} + \ddot{x}_{Cor}{}^{y} < 0$, l'état suivant sera < 0.

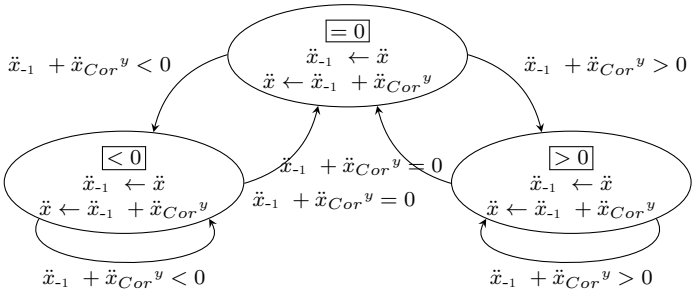

FIGURE 4.17 – Évolution qualitative de la dérivée seconde avec contraintes qualitatives

4.4.6 Mise en œuvre de la génération du terme correctif

La génération du terme correctif est assurée par une machine à états. L'ajout du terme correctif permet de changer la variable à gauche de la relation (variable d'*effet*) de manière à rendre la relation vraie. Pour les relations causales, le terme correctif est utilisé quand la relation n'est pas vérifiée à cause d'un changement dans la variable se trouvant à droite de la relation (variable de *cause*). En prenant cet exemple : x **thres** 0 **is CPROP** y **thres** 0, cette relation est modélisée par un automate comme le montre la Figure 4.18. Comme illustré dans cette figure, on détecte le changement qualitatif de la variable de cause y pour injecter une nouvelle valeur qualitative pour le terme correctif. Quand y devient > 0 et $x \leq 0$, nous mettons une nouvelle valeur qualitative > 0 pour le terme correctif dans le but de faire évoluer la valeur symbolique de x vers une valeur positive. Évidemment, quand les valeurs de x et y sont de nouveau proportionnelles, nous mettons la valeur qualitative du terme correctif à 0.

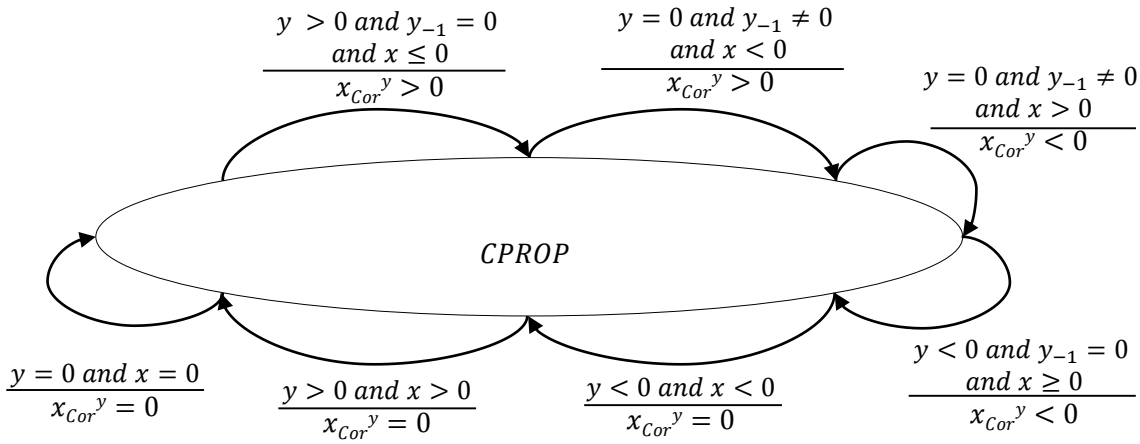

FIGURE 4.18 – Automate de l'opérateur qualitatif CPROP

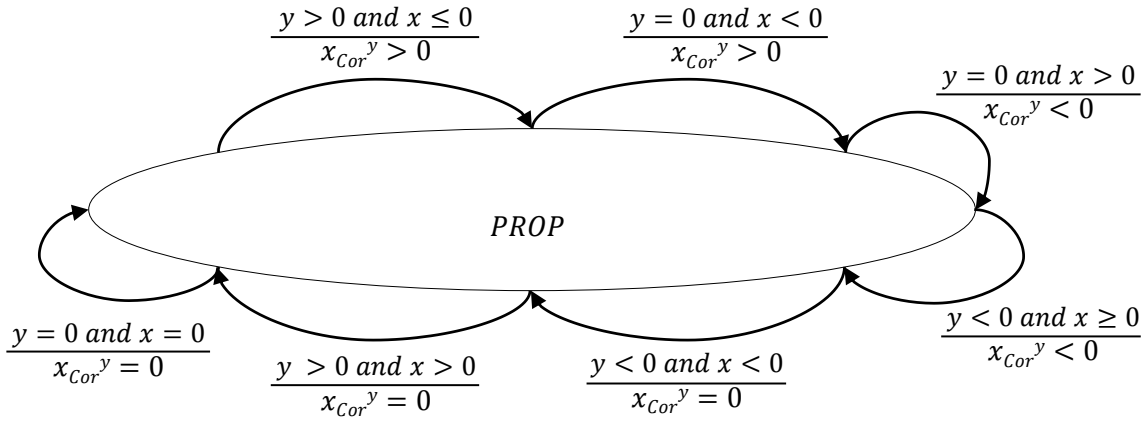

FIGURE 4.19 – Automate de l'opérateur qualitatif PROP

Considérons maintenant x **thres** 0 **is PROP** y **thres** 0. Cette relation est modélisée par un automate illustré par la Figure 4.19. Comme montré dans cette figure, on ne cherche pas à détecter le changement qualitatif de la variable de cause de la relation (y) dans le but de mettre une nouvelle valeur pour le terme correctif. Quand y est > 0 et $x \leq 0$, nous injectons une nouvelle valeur qualitative > 0 du terme correctif dans le but de faire évoluer la valeur symbolique de x vers une valeur positive. Évidemment, quand les valeurs de x et y sont de nouveau proportionnelles, nous ajustons la valeur qualitative du terme correctif à 0.

Pour les deux autres opérateurs **CIPROP** comme le montre la Figure 4.20 et **IPROP** illustré dans la Figure 4.21, l'automate qui injecte les nouvelles valeurs qualitatives du terme correctif est similaire aux automates de **CPROP** et **PROP**. Par contre, il influence les valeurs qualitatives de x et y d'une manière opposée :

— x **thres** 0 **is CIPROP** y **thres** 0 : Quand y devient < 0 et $x \leq 0$, nous

injectons une nouvelle valeur qualitative > 0 pour le terme correctif dans le but de faire évoluer la valeur symbolique de x vers une valeur positive. Évidemment, quand les valeurs de x et y sont de nouveau inversement proportionnelles, nous mettons la valeur qualitative du terme correctif à 0 ;

— x **thres** 0 **is IPROP** y **thres** 0 : Quand y est < 0 et $x \leq 0$, nous injectons une nouvelle valeur qualitative > 0 pour le terme correctif dans le but de faire évoluer la valeur symbolique de x vers une valeur positive. Évidemment, quand les valeurs de x et y sont de nouveau inversement proportionnelles, nous ajustons la valeur qualitative du terme correctif à 0.

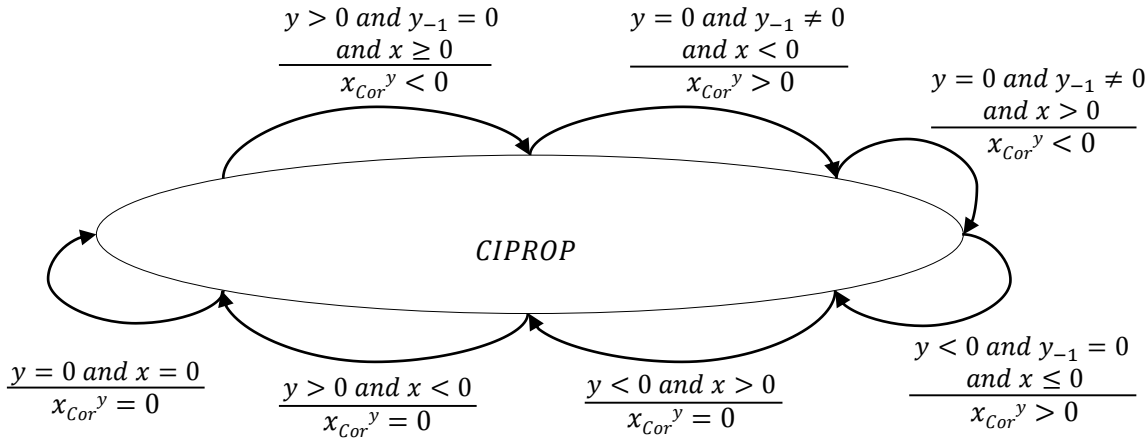

FIGURE 4.20 – Automate de l'opérateur qualitatif CIPROP

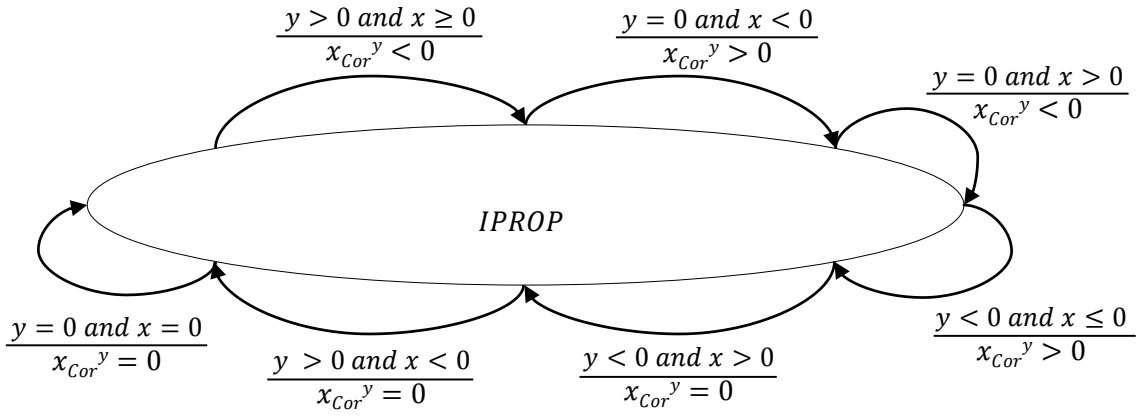

FIGURE 4.21 – Automate de l'opérateur qualitatif IPROP

4.4.7 Vérification des relations

Notre modèle d'exécution de contraintes qualitatives se base sur une intégration symbolique. Quand nous injectons une nouvelle valeur dans le modèle d'exécution pour ramener la variable d'effet d'une relation à suivre la variable de cause, la relation n'est pas forcément vérifiée à la fin de l'exécution. En effet, le terme correctif influence la variable d'effet pour suivre la variation de la variable de cause dans le même sens ou un sens opposé. Cette influence physique peut être aboutie ou pas. Par exemple, si on considère la relation suivante : x **thres** 0 **is PROP** y **thres** 0. Quand $y > 0$ et $x < 0$, le terme correctif est > 0 et $x = x_{\text{-}1} + \dot{x}_{\text{-}1} + {x_{Cor}}^{y}$. Selon que $x_{\text{-}1} + \dot{x}_{\text{-}1}$ est en valeur absolue inférieur à ${x_{Cor}}^{y}$ ou non, x peut devenir $= 0$ puis > 0 ou rester ≤ 0. Dans l'exécution symbolique, on va avoir les deux chemins : un chemin dans lequel l'influence physique n'est pas atteinte et un autre dans lequel l'influence physique est aboutie. Dans le but de discriminer le chemin dans lequel $x > 0$, nous utilisons un automate qui observe les changements qualitatifs de x en respectant la relation. Cet automate marque le chemin dans lequel la relation est vérifiée comme illustré dans la Figure 4.22.

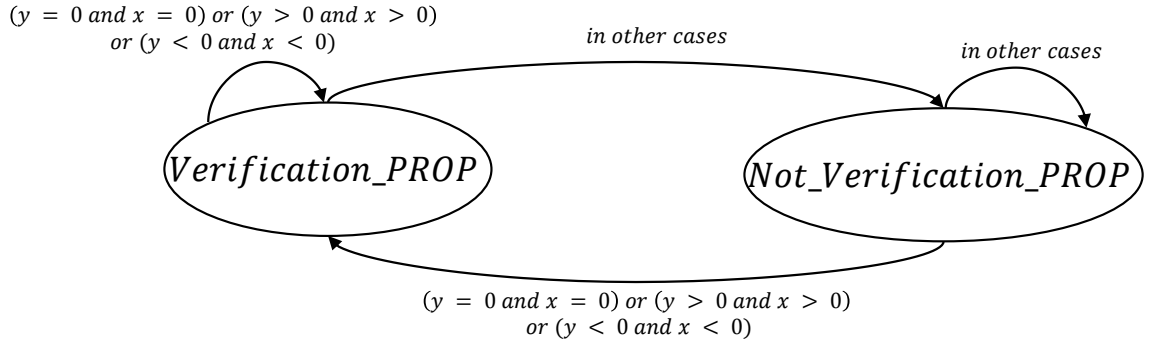

FIGURE 4.22 – Automate vérificateur de l'opérateur PROP

Dans cette figure, on voit que si les valeurs de x et y sont proportionnelles (ils sont tous les deux $= 0$, > 0 ou < 0), on est dans un état dans lequel la relation **PROP** est vérifiée. Dans tous les autres cas, on est dans un état dans lequel la relation **PROP** n'est pas vérifiée. Un automate similaire contrôle la vérification de l'opérateur **CPROP** mais dans ce cas, on prend en considération le changement qualitatif de la variable de cause de la relation. La Figure 4.23 modélise la vérification de la relation : x **thres** 0 **is CPROP** y **thres** 0.

Dans cette figure, on voit clairement qu'on observe le changement de la variable cause y pour dire que la relation est vérifiée ou pas : si $y_{\text{-}1} = 0$, $y > 0$ et $x > 0$ alors la

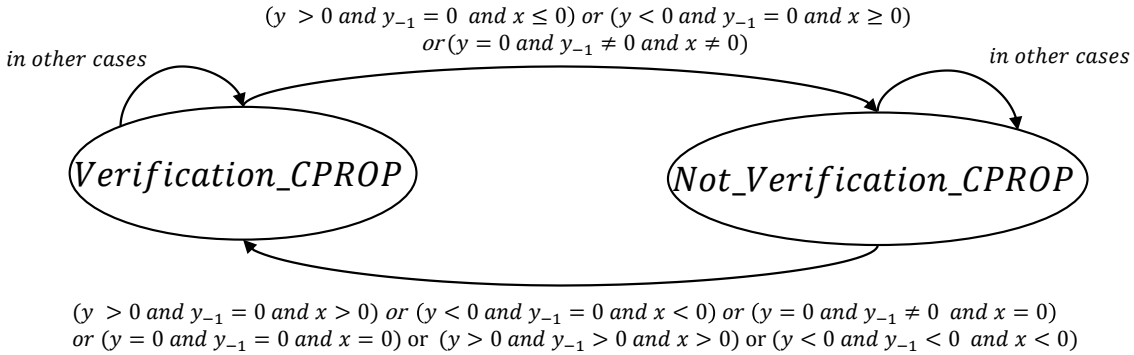

FIGURE 4.23 – Automate vérificateur de l'opérateur CPROP

relation est vérifiée instantanément mais aussi si $y_{\text{-}1} > 0$, $y > 0$ et $x > 0$, la relation est aussi vérifiée. Ceci est dû au fait que le terme correctif injecté dans le modèle d'exécution peut prendre un certain temps avant de modifier la valeur qualitative de $x = x_{\text{-}1} + \dot{x}_{\text{-}1} + {x_{Cor}}^{y}$. Supposons que $x_{\text{-}1} + \dot{x}_{\text{-}1} < 0$ et ${x_{Cor}}^{y} > 0$, la somme $x_{\text{-}1} + \dot{x}_{\text{-}1} + {x_{Cor}}^{y}$ peut rester < 0. D'après le principe de l'intégration symbolique $x_{\text{-}1} \leftarrow x$, $x_{\text{-}1} + \dot{x}_{\text{-}1} + {x_{Cor}}^{y}$ peut devenir $= 0$ pour être par la suite > 0 (on aura une autre valeur qualitative du terme correctif ${x_{Cor}}^{y} > 0$). Pendant ce temps, $y_{\text{-}1} \leftarrow y$ et ainsi $y_{\text{-}1} > 0$ et $y > 0$. C'est pourquoi on doit tenir compte de ce phénomène de propagation de l'intégration symbolique pour la vérification des relations causales.

Pour les deux autres opérateurs **IPROP** et **CIPROP**, on a deux autres automates pour vérifier que les variables liées par les relations sont inversement proportionnelles comme le montrent les deux Figures 4.24 et 4.25. Ces deux autres automates se basent sur le même principe expliqué ci-dessus pour les opérateurs **PROP** et **CPROP**.

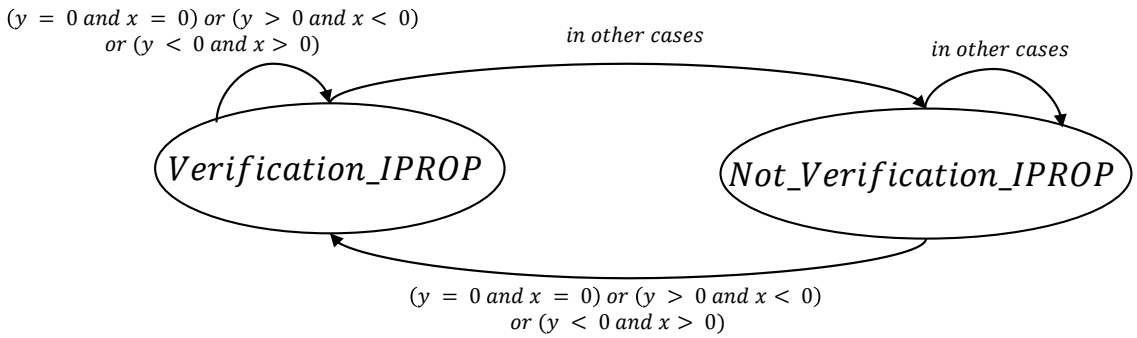

FIGURE 4.24 – Automate vérificateur de l'opérateur IPROP

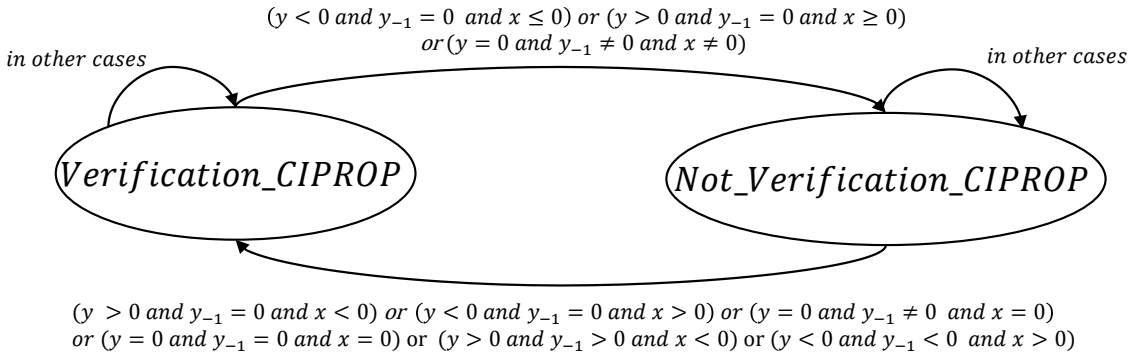

FIGURE 4.25 – Automate vérificateur de l'opérateur CIPROP

4.4.8 Implémentation du modèle de contraintes qualitatives dans Diversity

Ce modèle d'exécution avec contraintes qualitatives se compose de plusieurs automates. Le premier automate est l'automate du scénario dans lequel l'utilisateur spécifie le comportement de l'entrée du système et les différentes valeurs de la dérivée seconde courante. Le deuxième automate calcule les valeurs des différents termes correctifs des relations qui modélisent les interactions entre les différentes variables d'état du système. On a en effet autant d'automates calculant les termes correctifs que de nombre de relations. Le troisième automate suit les changements qualitatifs de la dérivée seconde ($\ddot{x}$) en l'intégrant symboliquement avec la valeur précédente de la dérivée seconde et les termes correctifs liées à $\ddot{x}$. Le quatrième automate calcule les valeurs qualitatives de la dérivée première ($\dot{x}$) qui est symboliquement intégrée avec la valeur précédente de la dérivée première, la dérivée seconde et les termes correctifs liées à $\dot{x}$. Le cinquième automate calcule la valeur qualitative de la variable d'état (x) qui est intégrée symboliquement avec la valeur précédente, la dérivée première et les termes correctifs liées à x. Le sixième automate calcule les variations qualitatives de la variable en observant les différents automates de la valeur qualitative de la variable, ses dérivées et ses valeurs précédentes pour détecter les différentes variations qualitatives des variables d'états (**Constant**, **Increase**...). Le septième automate calcule la vérification des relations après l'injection des termes correctifs dans le modèle d'exécution. Le huitième automate modélise le contrôleur spécifié par l'utilisateur. Ce dernier automate observe les sorties du modèle d'exécution de contraintes et ajuste les variables d'état concernées par la régulation du système. L'ordre dans lequel ces automates s'exécutent est important. Au début, l'automate du scénario est exécuté en séquence avec le bloc de tous les autres automates (lorsqu'on a des automates de

même nature, on utilisera la sémantique parallèle pour les exécuter). Ce bloc exécute les automates des relations en parallèle, puis les automates de la dérivée seconde en parallèle, ensuite les automates de la dérivée première en parallèle, après les automates de la valeur de la variable d'état en parallèle, puis les automates du comportement qualitatif en parallèle, ensuite les automates de vérification des relations en parallèle et finalement les automates du contrôleur en parallèle. L'automate du scénario fournit l'entrée du système et inilisalise les automates de la dérivée seconde, dérivée première et la valeur. C'est pourquoi, il doit être exécuté en premier lieu. Pour faire l'intégration symbolique, nous avons besoin de l'information du terme correctif. La génération du terme correctif est assurée par les automates de relations. D'où le besoin d'exécuter les automates de relations juste après l'automate du scénario. Le calcul de la dérivée première dépend de la dérivée seconde et celui de la valeur dépend de la dérivée première. Ainsi, l'automate de la dérivée seconde s'exécute avant celui de la dérivée première qui s'exécute avant celui de la valeur. La vérification des relations dépend des valeurs de la dérivée seconde, première et la valeur, ainsi les automates des relations vont s'exécuter par la suite. Aussi, les automates de régulation observant les différentes valeurs de la variable d'état vont s'exécuter par la suite.

4.4.9 Analyse des comportements qualitatifs

L'exécution symbolique nous permet d'explorer les comportements possibles du système. En effet, Diversity génère des traces qui incluent tous les chemins du modèle d'exécution des contraintes qualitatives. Parmi ces traces, il y a certains chemins d'exécution dans lesquels les influences physiques ne sont pas atteintes. Ainsi, ces traces doivent être éliminées parce qu'elles ne respectent pas la modélisation du système et les liens entre les différentes variables d'états du système. Ceci nous pousse encore plus à envisager de mettre dans notre chaîne d'outils, présentée dans le chapitre 6, un module qui va filtrer les traces générées par Diversity et produire les comportements qualitatifs réels du système en se basant sur les variables qu'on veut observer dans la simulation.

4.4.10 Exemple illustratif : Système de refroidissement

4.4.10.1 Principe général du système de refroidissement

Nous avons appliqué notre méthode sur un système de refroidissement de température montré dans la Figure 4.26. Ce type de système est utilisé dans les centrales nucléaires pour refroidir les circuits auxiliaires. Le but de ce circuit est de refroidir

différentes sources chaudes à travers une source froide contrôlée par des échangeurs. Le système de refroidissement contient des parties logiques et analogiques. Dans ce système, on a un flux de chaleur **Heat** qui entre à travers la source chaude et essaye de se refroidir quand il passe par la source froide. La source chaude produit un flux de température chaude **Thot** alors que la source froide produit un flux de température froide **Tcold**.

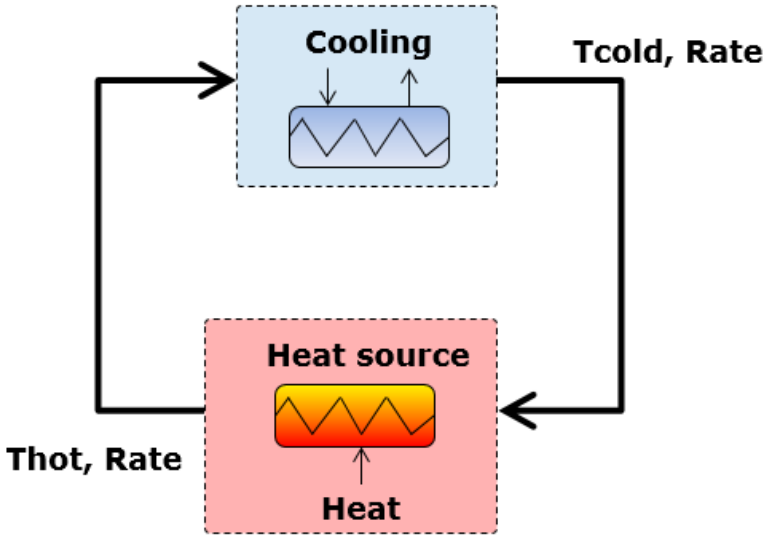

FIGURE 4.26 – Principe général du système de refroidissement

Le but principal du système de refroidissement est de réguler la température **Tcold** pour qu'elle soit égale à une température de consigne **Tc**. Pour faire ceci, on doit réguler le débit **Rate** de l'eau froide. Quatre variables d'états dans le système de refroidissement de température font le sujet principal de la simulation : le flux de chaleur *Heat* (entrée du système), la température chaude du circuit *Thot* (entrée de l'échangeur), la température froide *Tcold* (sortie de l'échangeur) et le débit *Rate* dans les échangeurs. Tous les autres paramètres du système sont constants. Dans ce système, la dynamique est décrite par les équations suivantes :

— $\dot{T}hot = \alpha\ \dot{H}eat$, avec $\alpha \in \mathbb{R}^+$;

— $\dot{T}cold = \beta\ \dot{T}hot$, avec $\beta \in \mathbb{R}^+$;

— $\dot{T}hot = \gamma\ \dot{T}cold$, avec $\gamma \in \mathbb{R}^+$;

— $\ddot{T}cold = -\theta\ \dot{R}ate$, avec $\theta \in \mathbb{R}^+$.

Ces équations peuvent être modélisées qualitativement par notre langage du modèle d'exécution de contraintes qualitatives sous forme de relations :

— **relation 1** $= \dot{T}hot$ **thres** 0 **is CPROP** $\dot{H}eat$ **thres** 0 ;

— **relation 2** = $\dot{T}cold$ **thres** 0 **is PROP** $\dot{T}hot$ **thres** 0 ;

— **relation 3** = $\dot{T}hot$ **thres** 0 **is PROP** $\dot{T}cold$ **thres** 0 ;

— **relation 4** = $\ddot{T}cold$ **thres** 0 **is CIPROP** $\dot{R}ate$ **thres** 0.

La **relation 1** modélise le fait que la variation de la chaleur $Heat$ influence la variation de la température chaude $Thot$ d'une manière proportionnelle. On ne veut corriger $Thot$ que si $Heat$ change de valeur qualitative, on utilise donc l'opérateur **CPROP**. Les **relations 2** et **3** modélisent le fait que la température chaude $Thot$ et la température froide $Tcold$ varient proportionnellement. On veut maintenir cette relation tout au long de l'exécution, on utilise alors l'opérateur **PROP**. La température chaude $Thot$ et la température froide $Tcold$ s'influencent d'une façon acausale. On modélise cette influence mutuelle par les **relations 2** et **3**. Dans la **relation 2**, la variation de $Thot$ impacte la variation de $Tcold$ et dans la **relation 3**, la variation de $Tcold$ impacte la variation de $Thot$. La **relation 4** modélise le fait que le changement qualitatif du débit $Rate$ fait varier la dérivée seconde de la température froide d'une manière opposée. Quand le débit $Rate$ de l'eau froide augmente, la température froide $Tcold$ a tendance à diminuer, donc la dérivée seconde $\ddot{T}cold$ est négative. Quand le débit $Rate$ de l'eau froide diminue, la température froide $Tcold$ a tendance à augmenter, donc la dérivée seconde $\ddot{T}cold$ est positive.

4.4.10.2 Scénario du système de refroidissement de température

On a choisi un scénario pour le système de refroidissement qui peut être modélisé par une machine à états comme le montre la Figure 4.27 avec 3 états et 4 transitions :

— la transition $t0$ initialise la variation de la chaleur $\dot{H}eat$ à 0 ;

— la transition $t1$ augmente la variation de la chaleur en mettant $\dot{H}eat > 0$;

— la transition $t2$ stabilise la variation de la chaleur en mettant $\dot{H}eat$ égale à 0 ;

— la transition $t3$ renforce la stabilité de la variation de la chaleur en mettant $\dot{H}eat$ égale à 0.

4.4.10.3 Régulation du système de refroidissement de température

La régulation du système de refroidissement de température est modélisée par une machine à états comme le montre la Figure 4.28 avec 3 états et 7 transitions :

— la transition $t0$ initialise la valeur de $Tcold$ à $= Tc$;

— la transition $t1$ diminue la variation du débit en déclenchant le mode Decrease_Rate si la valeur de $Tcold < Tc$;

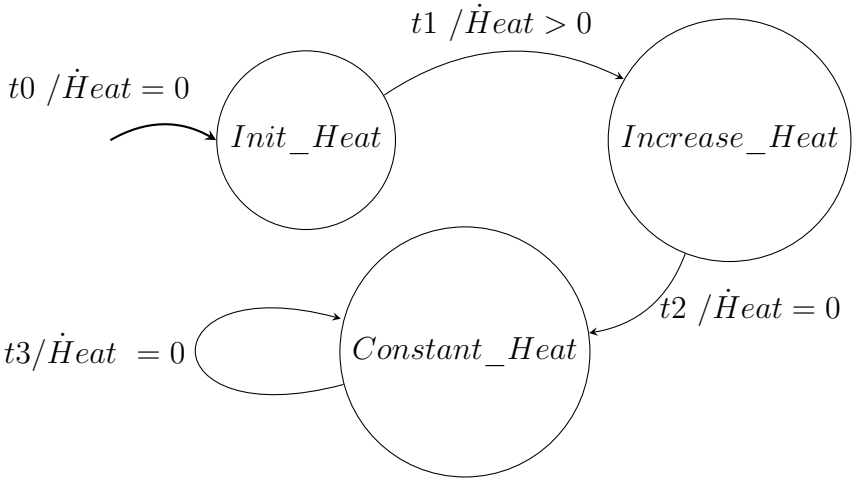

FIGURE 4.27 – Scénario du système de refroidissement

— la transition $t3$ rend la variation du débit constante en étant dans un mode Constant_Rate si la valeur de $Tcold = Tc$;

— la transition $t5$ diminue la variation du débit en déclenchant le mode Decrease_Rate si la valeur de $Tcold$ est encore $< Tc$;

— la transition $t2$ augmente la variation du débit en déclenchant le mode Increase_Rate si la valeur de $Tcold > Tc$;

— la transition $t6$ augmente la variation du débit en déclenchant le mode Increase_Rate si la valeur de $Tcold$ est encore $> Tc$.

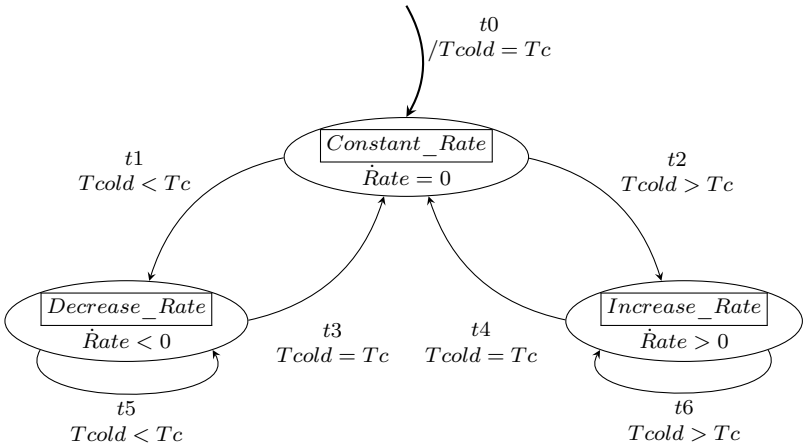

FIGURE 4.28 – Automate de régulation du système de refroidissement

4.4.10.4 Comportement qualitatif calculé par Diversity

Diversity trouve plusieurs comportements du système de refroidissement de température. Nous présenterons ces comportements dans la sous-section 4.4.11 page 66. On voit clairement dans la Figure 4.29 que les températures $Thot$ et $Tcold$ varient d'une manière proportionnelle. Lorsque $Thot$ est dans un état $Increase\_Thot$, $Tcold$ est aussi dans un état $Increase\_Tcold$. Lorsque $Tcold$ est dans un Max, $Thot$ arrête d'augmenter et elle se trouve dans l'état $Stop\_Increase$. On dispose de l'information sur $\ddot{T}cold$, c'est pourquoi on arrive à détecter les maxima et les minima. Par contre, on n'a pas l'information sur $\ddot{T}hot$, donc les maxima et les minima vont être remplacés par des Stop_Increase et Stop_Decrease. On remarque aussi que la variation de la chaleur $Heat$ a engendré une variation de Thot. Le débit $Rate$ varie suivant le seuil de la valeur de $Tcold$. Lorsque $Tcold > Tc$, $Rate$ augmente et lorsque $Tcold < Tc$, $Rate$ diminue, ce qui est modélisé par l'automate de la régulation du système de refroidissement qui se trouve dans la Figure 4.28.

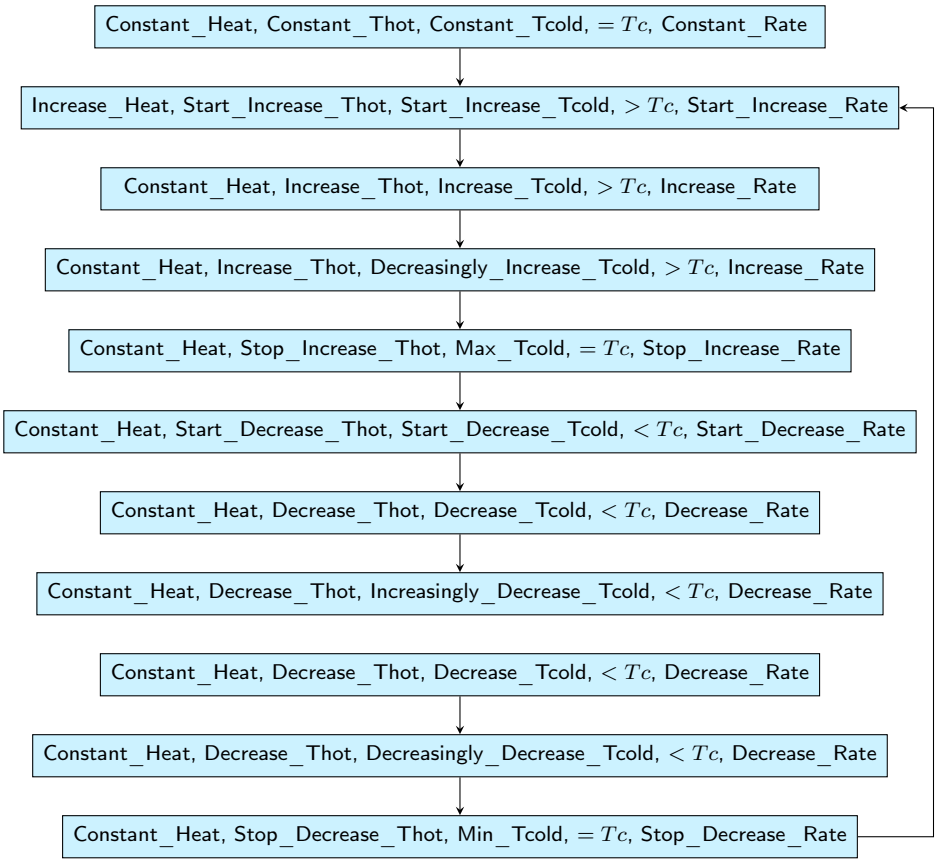

FIGURE 4.29 – Comportement qualitatif du système de refroidissement

4.4.11 Discussion sur le modèle d'exécution symbolique avec contraintes qualitatives

Parmi les comportements qualitatifs trouvés par Diversity pour le système de refroidissement, nous avons choisi un comportement oscillatoire du système dans lequel la température oscille autour de la température d'équilibre Tc. Il y a d'autres comportements trouvés par Diversity. On a un comportement dans lequel la température froide $Tcold$ reste toujours au dessus de la température d'équilibre Tc. Ce comportement est possible physiquement : on peut ne pas réguler le système suffisamment pour le ramener à sa température de consigne. Le débit injecté ne permet pas de faire descendre la température. Un autre comportement est que la température $Tcold$ reste toujours en dessous de sa température d'équilibre Tc. Ce comportement est possible aussi physiquement : on peut continuer à injecter du débit dans le système pour que la température reste toujours $< Tc$. Il reste encore des comportements trouvés par Diversity dans lesquels les relations ne sont jamais vérifiées. Nous avons mis alors dans notre méthodologie, un module qui permet de filtrer les résultats de Diversity dans le but de garder un seul chemin pour chaque comportement physique des systèmes hybrides et d'éliminer les comportements qui ne reflètent pas la dynamique du système, dans lesquels les relations ne sont pas vérifiées.

4.5 Discussion sur les trois modèles d'exécution

Le premier modèle de calcul présenté dans ce chapitre, se base sur le formalisme des automates discrétisés. Ce modèle est inspiré de l'algèbre de signes et se base sur *+,-,0*. Il nous a permis de mettre en évidence les limites de l'algèbre de signes : l'indéterminisme et le problème de l'explosion combinatoire. La différence de ce modèle d'exécution par rapport aux autres approches existantes qui se basent sur l'algèbre de signes, est que ce modèle énuméré est construit à base de discrétisation d'automates qui est le formalisme le mieux adapté pour la modélisation des systèmes hybrides et non pas sur des contraintes qualitatives comme dans QSIM. Cette première version de modèle énuméré observe les différentes valeurs de la dérivée seconde, dérivée première et la valeur pour déterminer les différents états qualitatifs de la variable comme : *Increase, Decrease...* Afin de résoudre le problème d'indéterminisme, nous avons construit le deuxième modèle d'exécution symbolique. Ce modèle s'inspire de la méthode d'Euler pour le calcul de la dérivée première en fonction de la dérivée seconde ou pour le calcul de la valeur en fonction de la dérivée première en prenant un pas d'intégration unitaire. Ce modèle compare symboliquement les valeurs de la

dérivée seconde, dérivée première et la valeur de la variable d'état pour connaître les transitions possibles : si $\dot{x} > 0$ et $\dot{x}_{\text{-}1} + \ddot{x}_{\text{-}1} > 0$ alors $\dot{x}$ reste dans le même état > 0, alors que si $\dot{x}_{\text{-}1} + \ddot{x}_{\text{-}1} = 0$, dans ce cas $\dot{x}$ change d'état qualitatif et devient $= 0$. Dans ce modèle d'exécution symbolique, nous avons ajouté des états qualitatifs qui modélisent un ordre supérieur de variation qualitative en s'inspirant de [47] et [23], comme *Increasingly_ Increase, Decreasingly_ Increase.*... Dans cette version de modèle de calcul, on ne dispose pas vraiment d'éléments de langage qui nous permettent de lier des variables du système. Dans l'exemple du circuit Rc, nous avons modélisé l'interaction entre la tension U_r et U_c dans l'automate du scénario. En effet, nous avons décrit ce qui se passe en phase de charge et de décharge. C'est pour ces raisons que nous avons construit le troisième modèle d'exécution à base de contraintes qualitatives. Dans cette version de modèle, nous avons défini des opérateurs qui permettent de lier différents niveaux des variables d'états. Nous nous sommes inspirés des opérateurs définis dans Garp3. Dans le but d'éliminer les comportements qui ne reflètent pas la dynamique du système, nous avons opté pour une solution algorithmique dans notre chaîne d'outils présenté dans le chapitre 5.

4.6 Conclusion

Dans ce chapitre, nous avons présenté les 3 modèles de calcul élaborés dans la thèse pour améliorer la simulation qualitative sans équations différentielles au sens du modèle mathématique continu :

- modèle d'exécution énuméré, dans ce modèle de calcul on se base sur l'algèbre des signes. La dérivée seconde, la dérivée première et la valeur sont modélisées sous forme d'automates ;
- modèle d'exécution symbolique avec intégration d'Euler, dans ce modèle de calcul, on prend un pas unitaire d'intégration ce qui donne $x = x_{\text{-}1} + \dot{x}_{\text{-}1}$ et $\dot{x} = \dot{x}_{\text{-}1} + \ddot{x}_{\text{-}1}$;
- modèle d'exécution avec contraintes qualitatives, dans ce modèle de calcul, on lie les dérivées et les valeurs de deux variables d'états. Dans le but de proposer à l'utilisateur un moyen de modéliser le comportement physique des systèmes hybrides sans équations différentielles au sens du modèle mathématique continu, nous avons défini 4 opérateurs :
  - *causally proportional*, **CPROP** ;
  - *proportional*, **PROP** ;

— *causally inversely proportional*, **CIPROP** ;

— *inversely proportional*, **IPROP**.

L'intégration symbolique sera ainsi $x = x_{\text{-}1} + \dot{x}_{\text{-}1} + {x_{Cor}}^{y}$, $\dot{x} = \dot{x}_{\text{-}1} + \ddot{x}_{\text{-}1} + {\dot{x}_{Cor}}^{y}$, $\ddot{x} = \ddot{x}_{\text{-}1} + {\ddot{x}_{Cor}}^{y}$, avec ${x_{Cor}}^{y}$, ${\dot{x}_{Cor}}^{y}$ et ${\ddot{x}_{Cor}}^{y}$, les termes correctifs liées respectivement à x, $\dot{x}$ et $\ddot{x}$.

Nous avons montré les limites des deux premières versions de modèles d'exécution notamment le problème de l'explosion combinatoire du modèle énuméré et le manque d'éléments de langage dans le modèle d'exécution symbolique pour lier deux variables d'état. Dans le troisième modèle, nous avons ajouté des automates de filtrage pour distinguer les chemins dans lesquels l'influence physique des relations est aboutie.

Chapitre 5

Langage pour la simulation qualitative

Dans ce chapitre, nous présenterons HyDiv qui est un langage de modélisation spécifique au domaine des systèmes hybrides que nous avons conçu. Aussi, nous présenterons le profil SysML dédié à la simulation qualitative sans équations différentielles au sens du modèle mathématique continu que nous avons mis au point.

5.1 Travaux existants dans la modélisation UML des systèmes hybrides

Il existe différents travaux qui se sont intéressés à la modélisation des systèmes hybrides en utilisant UML :

— *HybridUML* [8] est un profil UML basé sur CHARON [6] pour décrire les automates hybrides du système. Dans CHARON, un langage textuel, le comportement est représenté par des modes. Des transitions discrètes relient les différents modes. Des modes atomiques modélisent le comportement continu sous forme de contraintes algébriques : des contraintes différentielles et des invariants. HybridUML modélise l'aspect continu par des équations différentielles algébriques (DAE) ;

— *Vanderperren et al.* [67] ont proposé un outil pour établir une communication entre la conception du modèle par UML/SysML et la simulation du modèle avec Matlab/Simulink ;

— *Turki et Soriano* [66] ont proposé un diagramme d'activité basé sur les bond graphs présentés dans la sous-section 2.5.2.2 page 16 pour représenter la dynamique continue des systèmes mécatroniques ;

— *ModelicaML* [62] est un profil UML permettant aux ingénieurs de spécifier les besoins et les comportements. Ce profil modifie les diagrammes BDD et IBD. Il crée deux autres types de diagrammes, le diagramme d'équations pour décrire le comportement des classes et le diagramme de simulation pour faire varier les paramètres ;

— *UML^h* [34] est un profil UML qui décrit la description graphique et la modélisation des systèmes hybrides dans Modelica. Ce profil change les diagrammes de classes, d'états et de collaboration pour représenter les constructeurs textuels de Modelica [33] ;

— *Johnson et al.* [40] ont proposé un mapping entre SysML et Modelica pour la modélisation des systèmes dynamiques. Ils ont étendu la sémantique de SysML par la création des stéréotypes en fonction des composants de Modelica ;

— *Yue CAO Yusheng LIU et J.J. PAREDIS* [16] considèrent que le comportement discret est modélisé par le diagramme de séquence, d'activité et d'états de SysML. Ainsi, ils ont proposé une extension du diagramme paramétrique : *Sequence Parametric Diagram (SPD).*

Dans ces approches à l'exception de [67], la modélisation des systèmes hybrides se fait par la création d'un profil. SysML est un langage graphique et évolutif. En effet, il offre la possibilité d'extension de langage par la création des DSL (Domain Specific Language). Ceci nous a incité à créer un profil SysML pour la modélisation qualitative des systèmes hybrides. Ce profil donne un moyen de modélisation qualitative à une communauté d'ingénieurs système dans les premières phases de spécifications.

5.2 Méta modèle HyDiv

HyDiv est un langage textuel de modélisation spécifique au domaine des systèmes hybrides. Il est implémenté avec Xtext [26]. Il fournit une représentation compacte du système découplée du formalisme d'entrée (SysML) et du modèle d'exécution utilisé dans Diversity. La Figure 5.1 montre le métamodèle du langage HyDiv :

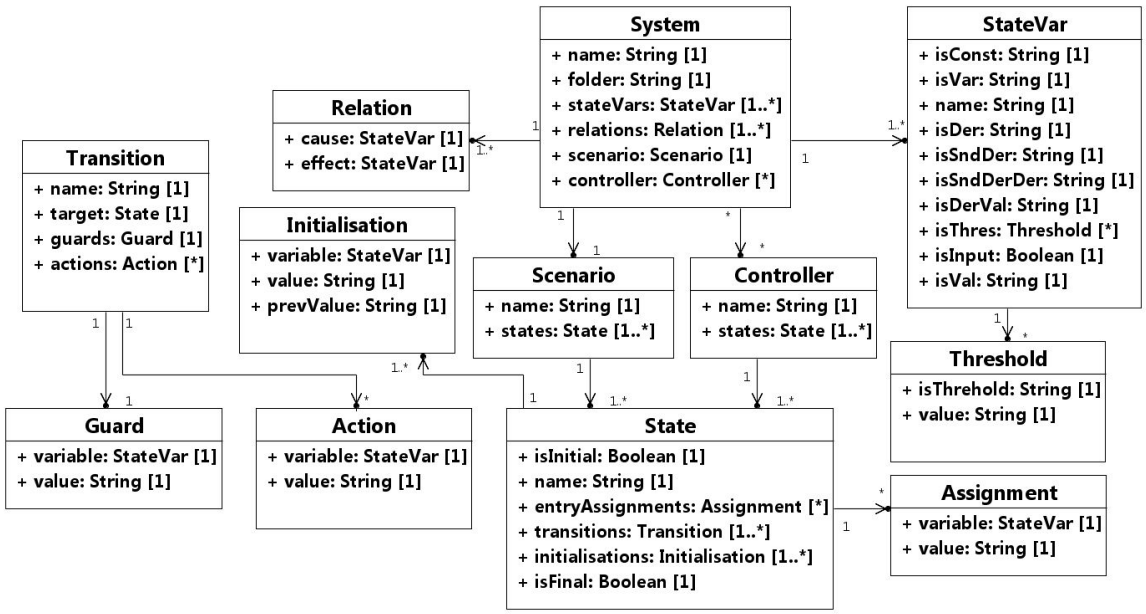

FIGURE 5.1 – Métamodèle HyDiv

un système hybride contient différentes propriétés telles que *name* et *folder* pour indiquer son emplacement, *statesVars* pour indiquer ses variables d'états, *relations* pour spécifier les relations qui modélisent les interactions entre les variables d'états

du système, *scenario* pour modéliser la variation des variables d'entrée du système et *controller* pour modéliser le régulateur du système qui réagit à la variation des différentes variables d'état du système.

Pour modéliser le fait qu'on peut utiliser les trois degrés de dérivées de chaque variable d'état, chaque *StateVar* contient : *isVar* pour préciser qu'on peut utiliser la dérivée seconde, la dérivée première et la valeur d'une variable d'état, *isSndDer* pour spécifier qu'on s'intéresse seulement à la dérivée seconde, *isDer* pour préciser qu'on s'intéresse seulement à la dérivée première de la variable d'état, *isVal* pour spécifier qu'on s'intéresse seulement à la valeur de la variable d'état, *isSnDerDer* pour dire qu'on peut utiliser la dérivée seconde et la dérivée première d'une variable d'état, *isDerVal* pour préciser qu'on s'intéresse à la dérivée première et la valeur d'une variable d'état. Le *StateVar* contient aussi *isConst* pour préciser s'il s'agit d'une constante, *isInput* pour préciser s'il s'agit d'une variable d'entrée pour le système et *isThres* pour préciser les seuils de la variable d'état.

Chaque *Relation* contient deux attributs : *cause* pour préciser la variable d'état cause et *effect* pour spécifier la variable d'état effet de la relation.

Chaque *controller* ou *scenario* contient : *name*, *states* pour modéliser les états qui composent l'automate du contrôleur ou du scénario.

Chaque *State* contient : *isInitial* pour préciser si c'est un état initial ou non, *isFinal* pour préciser si c'est un état final ou pas, *entryAssignments* pour modéliser les affectations des variables à l'entrée des états, *transitions* pour préciser les transitons de chaque état et *intialisations* pour préciser la valeur initiale de chaque variable d'état.

Chaque *Transition* contient : *name*, *target* pour préciser l'état cible de la transition, *guards* pour modéliser les conditions de franchissement de la transition et *actions* pour préciser les actions qui peuvent se faire après le franchissement de la transition.

5.3 Profil pour la simulation qualitative

Pour rendre notre approche utilisable par les concepteurs de systèmes et pour l'isoler des changements de notre modèle d'exécution et du format d'entrée, nous avons mis au point un profil pour SysML comme langage d'entrée pour la simulation qualitative sans équations différentielles au sens du modèle mathématique continu. Nous avons choisi SysML parce qu'il permet la modélisation des spécifications multi domaines. Il est aussi utilisé par une large communauté d'ingénieurs pour modéliser le système dans les premières phases de développement.

5.3.1 Extension des Blocks

Pour distinguer les différents composants du système qu'on veut modéliser d'une manière qualitative, on a créé trois stéréotypes : **QualitativeSystem** pour étiqueter le système global, **QualitativeBlock** pour étiqueter les différents composants du système et **Scenario** pour étiqueter l'entrée du système comme le montre la Figure 5.2. Un *QualitativeSystem* est composé de différents *QualitativeBlock* et d'un *Scenario*.

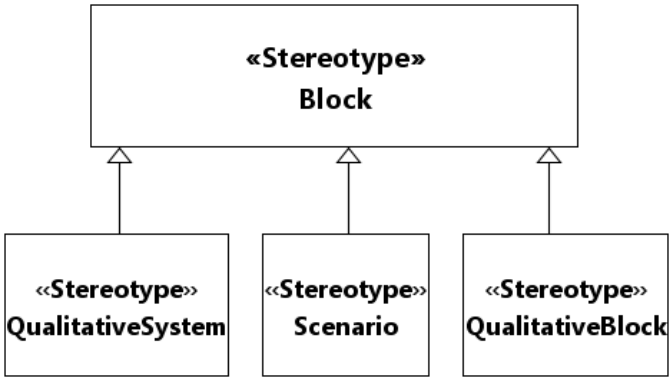

FIGURE 5.2 – Extensions de Block pour les blocks qualitatifs

5.3.2 Extension des machines à états

Pour distinguer les différentes machines à états qui modélisent le comportement du système et l'entrée du système, on a créé deux stéréotypes : **ScenarioStateMachine** pour étiqueter l'automate qui modélise le comportement du block de Scénario et **SystemStateMachine** pour étiqueter l'automate du comportement d'un système qualitatif comme le montre la Figure 5.3.

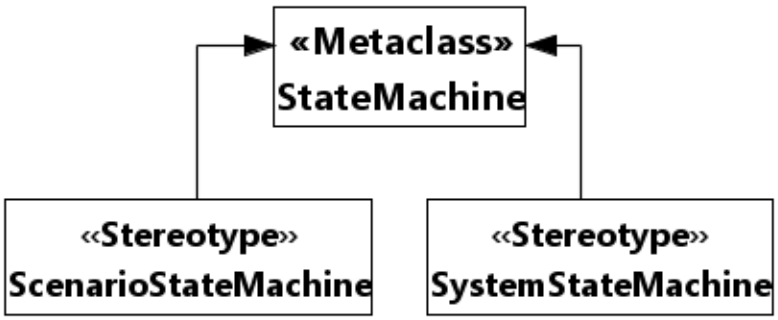

FIGURE 5.3 – Extension de Machine à états

Le *Classifier Behavior* d'un système qualitatif est une machine à états qui sera étiquetée par le stéréotype SystemStateMachine. On fait la même chose pour le Scénario mais dans ce cas le Classifier Behavior est une machine à états étiquetée par le stéréotype ScenarioStateMachine.

5.3.3 Extension des propriétés

Dans le but de distinguer les différentes variables d'états d'un block qualitatif, on a créé le stéréotype **QualitativeStateVariable** qui est composé de trois attributs comme le montre la Figure 5.4 :

— **order**, pour connaître l'ordre de la variable d'état. L'ordre peut être soit **first** si on n'utilise que la dérivée première, soit **second** si on utilise la dérivée seconde de la variable d'état ;

— **threshold**, pour connaître les différents seuils de la valeur de la variable d'état ;

— **isContinuous**, pour déterminer si la valeur spécifiée dans la propriété ordre est continue ou pas.

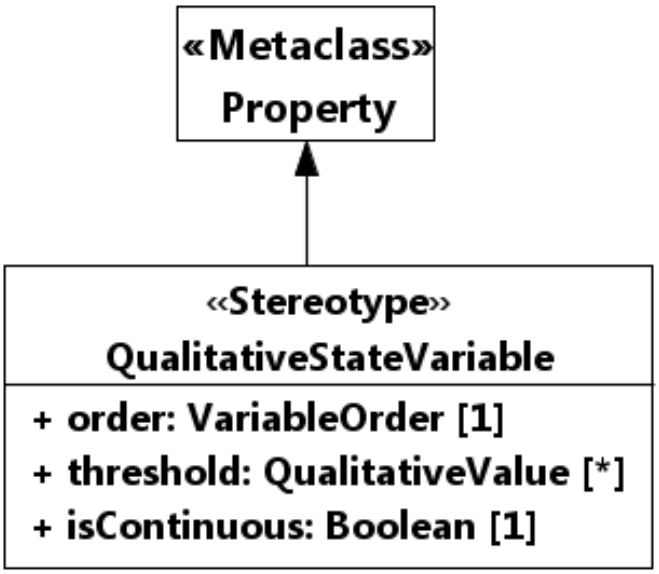

FIGURE 5.4 – Extension de *Property* pour les états qualitatifs

Un bloc qualitatif contient différents attributs étiquetés par le stéréotype **QualitativeStateVariable** pour spécifier les propriétés dont notre modèle d'exécution avec contraintes qualitatives a besoin. Ces propriétés vont être typées par **QualitativeType** pour pouvoir utiliser les différents niveaux de description d'une variable d'état comme illustré dans la Figure 5.5.

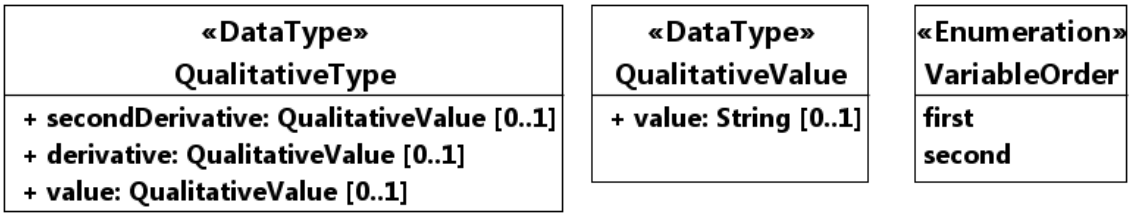

FIGURE 5.5 – Les différents types qualitatifs

5.3.4 Extension des Constraint Blocks

Pour exprimer le comportement continu des systèmes hybrides dans SysML, les concepteurs utilisent le diagramme paramétrique. Ce type de diagramme est utilisé pour modéliser une équation par le biais du composant SysML **ConstraintBlock**. Dans notre cas, nous ne disposons pas des équations différentielles numériques, nous avons un modèle qualitatif des équations qui peuvent être décrites par des relations basées sur les différents opérateurs de notre modèle d'exécution avec contraintes qualitatives décrit dans la section 4.4 page 52. Pour rendre la modélisation qualitative sans équations différentielles au sens du modèle mathématique continu, plus expressive dans SysML, nous avons créé le stéréotype **QualitativeConstraintProperty** pour nous permettre de saisir les seuils utilisés des variables d'état comme le montre la Figure 5.6.

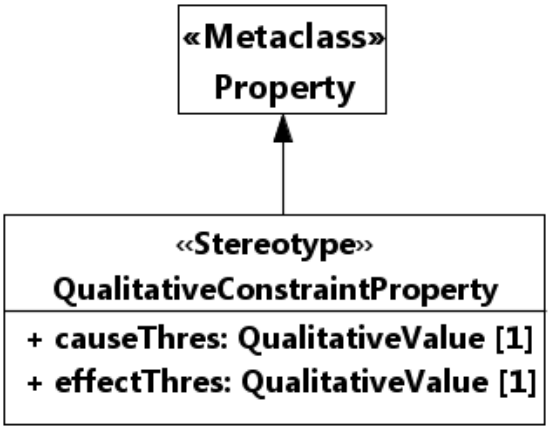

FIGURE 5.6 – Extension de *Property* pour les contraintes qualitatives

La **QualitativeConstraintProperty** a deux propriétés : **causeThres** qui modélise le seuil de la variable cause de la relation et **effectThres** qui modélise le seuil de la variable effet de la relation. Pour pouvoir utiliser les différents opérateurs définis dans la section 4.4, nous avons construit une **Qualitative Constraints Library**

qui contient les 4 opérateurs de notre modèle d'exécution avec contraintes qualitatives : **CPROP**, **PROP**, **CIPROP** et **IPROP**. Chaque opérateur a deux attributs : **cause** et **effect** pour spécifier les différentes variables d'une relation comme le montre la Figure 5.7.

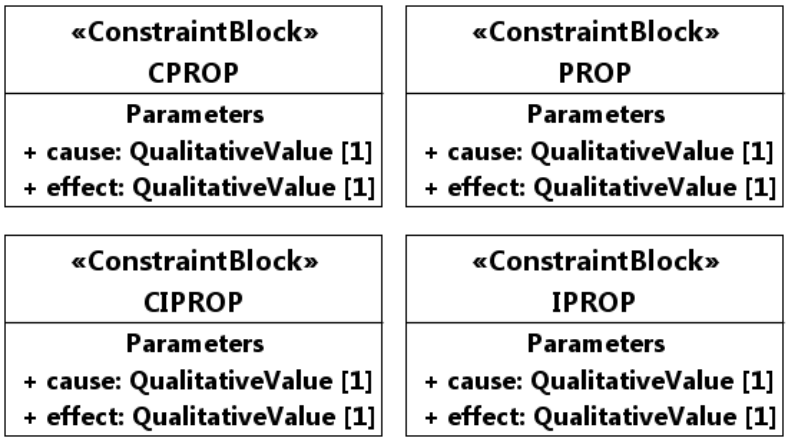

FIGURE 5.7 – La bibliothèque des contraintes qualitatives

5.3.5 Conclusion

Dans ce chapitre, nous avons présenté HyDiv qui est un langage textuel de modélisation spécifique au domaine de systèmes hybrides. Ce langage fournit une représentation compacte du système découplée du formalisme d'entrée (SysML) et du modèle d'exécution utilisé dans Diversity. Nous avons aussi présenté le profil SysML que nous avons mis en place pour la simulation qualitative sans équations différentielles. Ce profil a été mis au point dans le but de rendre la modélisation qualitative dans SysML plus expressive. SysML est un langage évolutif. En effet, il nous donne la possibilité de créer un DSL (Domain Specific Language) graphique pour un domaine spécifique. Nous apportons une dimension qualitative à SysML. Nous proposons aux ingénieurs système un moyen de faire de la simulation qualitative en amont du cycle de développement.

Chapitre 6

Implémentation : Chaîne d'outils pour la validation des spécifications hybrides

Dans ce chapitre nous présentons en détail une chaîne d'outils que nous avons mise en place pour la validation fonctionnelle des systèmes hybrides. Nous présentons les modules de notre chaîne d'outils et les technologies de l'ingénierie dirigée par les modèles utilisées.

Nous modélisons un système hybride avec un modèle SysML, sur lequel nous avons appliqué notre profil de simulation qualitative, en utilisant le plug-in Papyrus d'Eclipse. Ensuite nous transformons le modèle SysML en utilisant "QVT operational" [28] qui est un langage permettant de décrire une transformation du modèle source SysML vers un modèle cible HyDiv. HyDiv est un langage pivot que nous avons conçu pour faire une représentation de haut niveau du système hybride, indépendamment du modèle source. Nous utilisons ensuite une transformation Acceleo [27] qui transforme le modèle HyDiv en Xlia, le langage d'entrée de Diversity. Dans le but de générer des comportements qualitatifs, nous avons ajouté un module qui prend en entrée les traces brutes produites par Diversity et qui produit en sortie des comportements qualitatifs en appliquant quelques filtres pour éliminer les problèmes expliqués dans le chapitre 4. La Figure 6.1 montre le principe général de la chaîne d'outils.

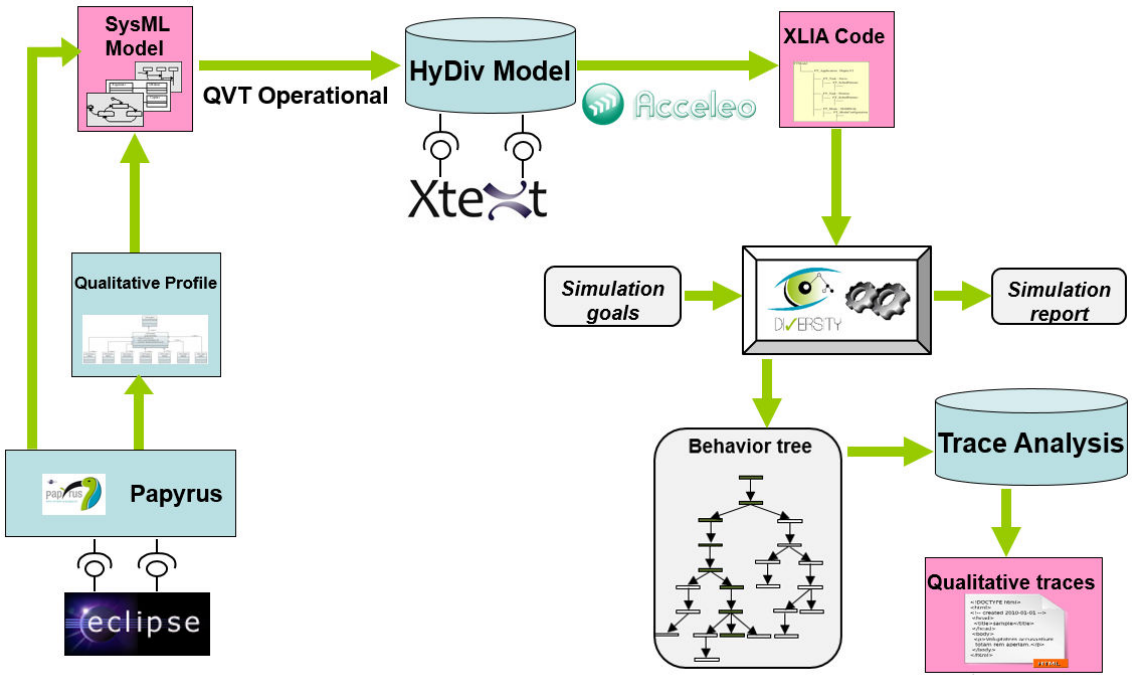

FIGURE 6.1 – La chaîne d'outils

6.1 Transformation de modèles

6.1.1 Transformation de modèle en modèle

6.1.1.1 Règles de la transformation de modèle en modèle

La première étape pour effectuer la simulation qualitative est de transformer le modèle SysML du système, sur lequel est appliqué le profil de simulation qualitative, en un modèle HyDiv. La transformation QVT opérationnelle depuis SysML vers HyDiv se fait par les mapping suivants :

— *SysML Block* avec le stéréotype *QualitativeSystem* est traduit en *System* ;

— *UML Property* avec le stéréotype *QualitativeStateVariable* est traduite en *StateVar* ;

— *ConstraintBlock* avec le stéréotype *QualitativeConstraintProperty* est traduite en *Relation* ;

— *StateMachine*, attachée au Classifier Behavior d'un Block avec le stéréotype *ScenarioStateMachine*, est traduite en *Scenario* ;

— *StateMachine*, attachée au Classifier Behavior d'un Block avec le stéréotype *SystemStateMachine*, est traduite en *Controller* ;

— *State* d'une machine à états avec le stéréotype *ScenarioStateMachine* ou *SystemStateMachine*, est traduit en *State* ;

— *Transition* d'une machine à états avec le stéréotype *ScenarioStateMachine* ou *SystemStateMachine*, est traduite en *Transition* ;

— *Effect* d'une transition est traduit en *Action* ;

— *Entry* d'un état est traduit en *Assignment* ;

— *Guard* d'une transition est traduite en *Guard*.

6.1.1.2 Algorithme de la transformation de modèle en modèle

La transformation de modèle en modèle se déroule en plusieurs étapes comme le montre la Figure 6.2 :

— Parcourir les blocks SysML du modèle pour détecter celui sur lequel le stéréotype *QualitativeSystem* est appliqué et le mapper en *System* (1) ;

— Sélectionner les blocks qui composent le block trouvé dans (1) et sur lesquels le stéréotype *QualitativeBlock* est appliqué (2) ;

— Pour chaque block trouvé dans (2), sélectionner les attributs sur lesquels le stéréotype *QualitativeStateVariable* est appliqué et les mapper en *StateVar* ;

— Pour chaque block trouvé dans (2), sélectionner la machine à états, qui est dans le Classifier Behavior du block, sur lequel le stéréotype *SystemStateMachine* est appliqué et la mapper en *Controler* ;

— Pour chaque block trouvé dans (2), sélectionner la machine à états, qui est dans le Classifier Behavior du block, sur lequel le stéréotype *ScenarioStateMachine* est appliqué et la mapper en *Scenario* ;

— Pour le block trouvé dans (1), trouver les attributs dont le type est un des 4 opérateurs *CPROP, PROP, CIPROP, IPROP* et les mapper en *Relation*.

```
var modelBlock : Set(UML::Class) -> sélectionner les classes du modèle uml ;
var systemBlock : Set(SysML::Block) -> sélectionner parmi modelBlock les blocks avec
                                       le stéréotype «QualitativeSystem» appliqué ;
systemBlock.map SystemBlock2HYDModel() ;

mapping uml::Class :: SystemBlock2HYDModel() : HyDiv::System {
        var qualitativeBlock : Set(SysML::Block) -> sélectionner les blocks qui composent systemBlock avec
                                                    le stéréotype «QualitativeBlock» appliqué ;
        var scenarioBlock : Set(SysML::Block) -> sélectionner parmi modelBlock les blocks avec
                                                 le stéréotype «Scenario» appliqué ;
        Pour chaque qualitativeBlock faire :
                1) var properties : Set(HyDiv::StateVar) -> sélectionner les attributs sur lesquels
                                                            le stéréotype «QualitativeStateVariable» est appliqué ;
                2) mapper properties en HyDiv:: StateVar ;

                3) var stateMachineSystem : (UML::StateMachine) -> sélectionner la machine à états avec le stéréotype
                                                                   «SystemStateMachine» qui se trouve dans le
                                                                   classifier behavior de qualitativeBlock ;

                4) mapper stateMachineSystem en HyDiv:: Controller ;

        var stateMachineScenario : (UML::StateMachine) -> sélectionner la machine à états avec le stéréotype
                                                          «ScenarioStateMachine» qui se trouve dans le classifier
                                                          behavior de scenarioBlock ;

        mapper stateMachineScenario en HyDiv:: Scenario ;
        var attributes : Set(HyDiv::Relation) -> sélectionner les attributs de systemBlock
                                                 dont le type est soit: CPROP, PROP, CIPROP, IPROP ;
        mapper attributes en HyDiv:: Relation ;
}
```

FIGURE 6.2 – Pseudo code de l'algorithme de la transformation de modèle en modèle

6.1.2 Transformation de modèle en texte

La transformation Acceleo du modèle d'entrée HyDiv pour générer le code Xlia de Diversity se déroule en plusieurs étapes :

```
@xfsp <system, 1.0 >:
[comment]Générer le nom du système[/comment]
system <and> [aSystem.name +'_with_relationships'/] {
[comment]Parcourrir les variables d'états du système[/comment]
        [for (svar: StateVar | aSystem.stateVars)]
            [if (svar.isInput->notEmpty()) and (svar.isDer->notEmpty())]
[comment]Générer les variables d'entrée du système[/comment]
        var Input_Qualitative_State_without_Value [svar.name/] ;
            [/if]
[comment]Générer les variables d'états sans valeur du système[/comment]
            [if (svar.isVar->notEmpty()) and (svar.isDer->notEmpty())]
        var Qualitative_State_without_Value [svar.name/] ;
            [/if]
[comment]Générer les variables d'états avec valeur du système[/comment]
            [if (svar.isVar->notEmpty()) and (svar.isDer->isEmpty()
                and (svar.isSndDer->isEmpty()) and (svar.isSndDerDer->isEmpty())
                and (svar.isDerVal->isEmpty())) ]
        var Qualitative_State [svar.name/] ;
            [/if]
[comment]Générer les constantes du système[/comment]
            [if (svar.isVar->notEmpty())
                and (svar.isDer->isEmpty() and (svar.isSndDer->isEmpty())
                and (svar.isSndDerDer->isEmpty()) and (svar.isDerVal->isEmpty())) ]
        const var int [svar.isSeuil.valueVal->at(1)/] ;
            [/if]
        [/for]
```

FIGURE 6.3 – Code de la transformation Acceleo pour générer le nom du système et les variables qualitatives

— Parcourir le modèle textuel de HyDiv comme le montre la Figure 6.3 pour générer le nom du système et les variables qualitatives ;

— Parcourir le modèle textuel de HyDiv comme illustré par les Figures 6.4 et 6.5 pour générer l'automate du scénario en prenant en compte les actions des transitions et les affectations dans les états ;

— Parcourir le modèle textuel de HyDiv pour générer l'automate du système en prenant en compte les actions des transitions et les affectations dans les états ;

— Parcourir le modèle textuel de HyDiv pour générer les automates de relations qualitatives et les automates de vérification en prenant en compte les 4 opérateurs : *CPROP, PROP, CIPROP, IPROP* et les variables d'état liées avec leur seuils ;

— Générer le modèle d'exécution des différents automates présents dans le système comme illustré par les Figures 6.6 et 6.7 suivant ce que nous avons expliqué dans la sous-section 4.3.5 page 51.

```
        //State machine for the Scenario
        statemachine <or> Scenario {
          @region:
[comment]Générer l'état initial de l'automate du scénario[/comment]
          state <initial> InitSys_scenario {
            transition Init2[aSystem.initStateScenario().name/] --> [aSystem.initStateScenario().name/] {
            }
          }
[comment]Parcourrir les états de l'automate du scénario[/comment]
        [for (state: State | aSystem.scenario.states)]
[comment]Générer les états de l'automate du scénario[/comment]
          state [if (state.isFinal->notEmpty())]<final>[/if][state.name/] {
[comment]Générer les affectations des états de l'automate du scénario[/comment]
              [if (state.entryAssignments->notEmpty())]
                @enable {
                    [for (assignment:Assignment | state.entryAssignments)]
                    [assignment.assignment.variable.genVar()/] [assignment.assignment.operation.op/]
                        [assignment.assignment.operation.value/];
                    [/for]
                }
              [/if]
[comment]Tester l'état final[/comment]
              [if (state.isFinal->notEmpty())]
                @enable {
                    exit;
                }
              [/if]
```

FIGURE 6.4 – Code de la transformation Acceleo pour générer l'automate du scénario (1)

```
        [comment]Parcourrir les transitions des états de l'automate du scénario[/comment]
                  [for (trans: Transition | state.transitions)]
        [comment]Générer les transitions des états de l'automate du scénario[/comment]
                  transition [trans.name/] --> [trans.target.name/] {
                                [if (trans.actions->notEmpty())]
        [comment]Parcourrir les actions des transitions de l'automate du scénario[/comment]
                                    [for (action: Action | trans.actions)]
        [comment]Générer les actions des transitions de l'automate du scénario[/comment]
                    newfresh( [action.variable1.genVar()/] ) ;
                                    [/for]
                                [/if]
                    [if (trans.guards->isEmpty())][else]
        [comment]Générer les gardes des transitions de l'automate du scénario[/comment]
                    guard([for (guard: Guard | trans.guards) separator (' && ')]
                          ([guard.expression.variable.genVar()/] [guard.expression.op/]
                          [guard.expression.value/])[/for][for (action: Action | trans.actions)
                          before (' && ') separator (' && ')]([action.variable.genVar()/]
                          [action.operation.op/] [action.operation.value/])[/for]);
                    [/if]
                    [if (trans.guards->notEmpty())][else]
                    guard([for (guard: Guard | trans.guards) separator (' && ')]
                          ([guard.expression.variable.genVar()/] [guard.expression.op/]
                          [guard.expression.value/])[/for][for (action: Action | trans.actions)
                          separator (' && ')]([action.variable.genVar()/] [action.operation.op/]
                          [action.operation.value/])[/for]) ;
                    [/if]
                  }
                 [/for]
                }
               [/for]
               }
```

FIGURE 6.5 – Code de la transformation Acceleo pour générer l'automate du scénario (2)

```
// Model of concurrency
@moe:
        @schedule{ |;|
[comment]Générer la sémantique d'exécution de l'automate du scénario[/comment]
            run Scenario ;
          [if ((aSystem.relations->notEmpty()))]
[comment]Générer la sémantique d'exécution des relations du système[/comment]
               { |,|
[comment]Parcourrir les relations du système[/comment]
              [for (rela: Relation | aSystem.relations)]
                    run Relation[rela.name/] ;
              [/for]
               }
          [/if]
[comment]Parcourrir variables d'états du système[/comment]
              [for (svar: StateVar| aSystem.stateVars)]
                [if ((svar.isVar->notEmpty()) and (svar.isDer->isEmpty())
                       and (svar.isDer->isEmpty()) and (svar.isSndDer->isEmpty())
                       and (svar.isSndDerDer->isEmpty()) and (svar.isDerVal->isEmpty()) )]
              { |;|
              [for (svar: StateVar| aSystem.stateVars)]
                   [if ((svar.isSndDer->notEmpty()))]
[comment]Générer la sémantique d'exécution des automates de la dérivée seconde[/comment]
                { |,|
                   run [svar.name/]_SndDerivative ;
                }
                   [/if]
              [/for]
                [/if]
              [/for]
```

FIGURE 6.6 – Code de la transformation Acceleo pour générér le modèle d'exécution des différents automates (1)

```
[comment]Générer la sémantique d'exécution des automates de la dérivée première[/comment]
               { |,|
             [for (svar: StateVar| aSystem.stateVars)]
               [if ((svar.isVar->notEmpty()) or (svar.isInput->notEmpty()))]
                  run [svar.name/]_Derivative ;
               [/if]
             [/for]
               }
[comment]Générer la sémantique d'exécution des automates de la valeur[/comment]
               { |,|
             [for (svar: StateVar| aSystem.stateVars)]
               [if ((svar.isSeuil.valueVal->notEmpty()))]
                  run [svar.name/]_Value ;
               [/if]
             [/for]
               }
             }
             [for (svar: StateVar| aSystem.stateVars)]
[comment]Générer la sémantique d'exécution des automates de comportement qualitatif[/comment]
               [if ((svar.isInput->isEmpty()))]
                  run [svar.name/]_QualVar ;
               [/if]
             [/for]
              [if ((aSystem.relations->notEmpty()))]
[comment]Générer la sémantique d'exécution des automates de vérification des relations[/comment]
             [for (rela: Relation | aSystem.relations)]
                  run Verification[rela.name/] ;
             [/for]
          [/if]
[comment]Générer la sémantique d'exécution de l'automate du contrôleur[/comment]
                  run Controller ;
       }
```

FIGURE 6.7 – Code de la transformation Acceleo pour générér le modèle d'exécution des différents automates (2)

6.2 Analyseur de traces

6.2.1 Présentation de l'analyseur de traces

L'analyseur de traces est une application Java qui prend en entrée les traces brutes de la simulation symbolique générées par Diversity et produit en sortie les comportements qualitatifs du système hybride en utilisant plusieurs filtres :

Filtre mathématique Dans les traces brutes d'exécution symbolique de Diversity, il existe certaines qui finissent par une variable d'état x restant au-dessus de son seuil avec sa dérivée première $\dot{x} < 0$ et une dérivée seconde $\ddot{x} < 0$. Symboliquement ce comportement est possible. En effet, $x = x_{-1} + \dot{x}_{-1}$ avec une valeur symbolique $x\sharp 1 > 0$, $\dot{x} = \dot{x}_{-1} + \ddot{x}_{-1}$ avec une valeur symbolique $\dot{x}\sharp 2 < 0$ et $\ddot{x}$ avec une valeur symbolique $\ddot{x}\sharp 3 < 0$. Ainsi $x = x\sharp 1 + \dot{x}\sharp 2 + \ddot{x}\sharp 3$. Si $x\sharp 1 > |\dot{x}\sharp 2 + \ddot{x}\sharp 3|$ alors la variable d'état reste au-dessus de son seuil.

D'autres traces finissent par une variable d'état x restant en-dessous de son seuil avec sa dérivée première $\dot{x} > 0$ et sa dérivée seconde $\ddot{x} > 0$. Dans le cas $x\sharp 1 < 0$ $\dot{x}\sharp 2 > 0$ et $\ddot{x}\sharp 3 > 0$, si $|x\sharp 1| > \dot{x}\sharp 2 + \ddot{x}\sharp 3$ alors la variable d'état reste en-dessous de son seuil.

Ce filtre élimine les chemins qui finissent par une variable d'état restant toujours au dessus de son seuil alors que ses dérivées premières et secondes sont négatives. Dans ce cas, la variable est dans un état *Increasingly_ Decrease* mais elle reste indéfiniment au dessus de son seuil fini, ce qui est mathématiquement impossible. Ce filtre élimine aussi l'autre cas quand la variable est dans un état *Decreasingly_ Increase*. Elle reste toujours en dessous de son seuil fini avec une dérivée première et seconde positives.

Filtre de validation élimine les chemins qui marquent le non aboutissement d'une influence physique d'une relation. En effet, dans le dernier contexte d'exécution, si la tendance physique de la relation n'a pas abouti alors ce chemin est éliminé des traces de simulation.

Filtre des variables observées fusionne les comportements similaires quand on cache les variables que l'utilisateur ne sélectionne pas pour l'observation. En choisissant certaines variables, l'utilisateur obtient des comportements abstraits n'incluant que les changements qualitatifs des variables sélectionnées. Ces comportements abstraits peuvent être raffinés à différents niveaux en augmentant le nombre de variables choisies pour aider l'utilisateur à comprendre les rai-

sons d'obtention de tels comportements abstraits. Les comportements détaillés incluant toutes les variables peuvent être inspectés en cas de besoin.

6.2.2 L'algorithme de l'analyseur de traces

Le module de l'analyse des traces brutes fournies par Diversity se déroule en plusieurs étapes :

- (1) lecture des traces textuelles brutes de l'exécution symbolique ;
- (2) transformation des traces textuelles en une structure de données qualitatives comportant :
  - un identificateur pour distinguer les différents contextes les uns des autres ;
  - une liste de valeurs qualitatives ;
  - un successeur ;
  - un champ de redondance pour indiquer si le contexte en question est un contexte déjà rencontré dans la trace d'exécution.
- (3) extraction des comportements qualitatifs à partir des traces brutes de Diversity transformées dans (2). Diversity fournit un arbre de simulation symbolique dans lequel l'information de la redondance peut être présente dans tout l'arbre d'exécution et pas forcément sur la branche courante. Dans cette étape de l'algorithme, nous construisons des comportements qualitatifs dans lesquels l'information de la redondance est dans le même chemin de l'exécution ;
- (4) appliquer le *filtre mathématique* sur les comportements trouvés dans (3) en enlevant les comportements qui sont incorrects mathématiquement ;
- (5) appliquer le *filtre de validation* qui élimine les comportements trouvés dans (4) qui ne respectent pas les relations décrivant la dynamique du système ;
- (6) appliquer le *Filtre des variables observées*. A cette étape de l'algorithme, nous ne générons que les comportements qualitatifs des variables que l'utilisateur veut observer. Ainsi, nous parcourrons les comportements qualitatifs valides trouvés dans (5) en prenant en considération les variables observables. Lorsque deux contextes d'exécution ont les mêmes valeurs qualitatives des variables observées, nous les fusionnons dans un même contexte ;
- (7) rechercher les comportements qualitatifs complets (dans lesquels toutes les variables du système sont présentes) depuis lesquels nous avons extrait les comportements trouvés dans (6) ;

— associer par des liens hypertextes les comportements trouvés dans (6) et (7) ;

— (8) mettre les comportements trouvés dans (6) et (7) sous forme de *Html* ;

— (9) diviser le fichier généré en (8) en sous fichiers de *100* comportements qualitatifs chacun pour des raisons de lisibilité.

6.3 Exemple illustratif : Système de refroidissement de température

6.3.1 Architecture du système de refroidissement de température

L'architecture du système de refroidissement est modélisée dans un Block Definition Diagram dans SysML comme le montre la Figure 6.8.

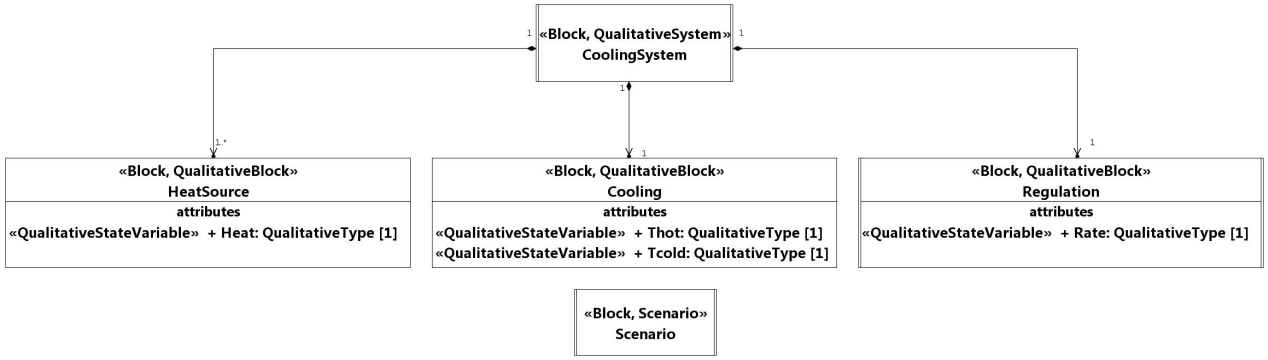

FIGURE 6.8 – Diagramme de définition de blocks du système de refroidissement

Le système de refroidissement qui est un **QualitativeSystem** est composé de trois **QualitativeBlock** :

— le block de *HeatSource* contenant une **QualitativeStateVariable** : *Heat* ;

— le block de *Cooling* contenant deux **QualitativeStateVariable** : *Thot* et *Tcold* ;

— le block de *Regulation* contenant une **QualitativeStateVariable** : *Rate* ;

— le block de *Scenario* contenant une **ScenarioStateMachine** dans laquelle l'utilisateur spécifie la variation de la chaleur qui est l'entrée du système.

6.3.2 Scénario du système de refroidissement

Nous avons choisi un scénario pour le système de refroidissement qui peut être modélisé par une machine à états comme le montre la Figure 6.9 avec 2 états, 1 pseudo-état et 3 transitions :

— la transition $t0$ initialise la variation de la chaleur $\dot{H}eat$ à > 0 ;

— la transition $t1$ stabilise la variation de la chaleur en mettant $\dot{H}eat$ égale à 0 ;

— la transition $t2$ renforce la stabilité de la variation de la chaleur en mettant $\dot{H}eat$ égale à 0.

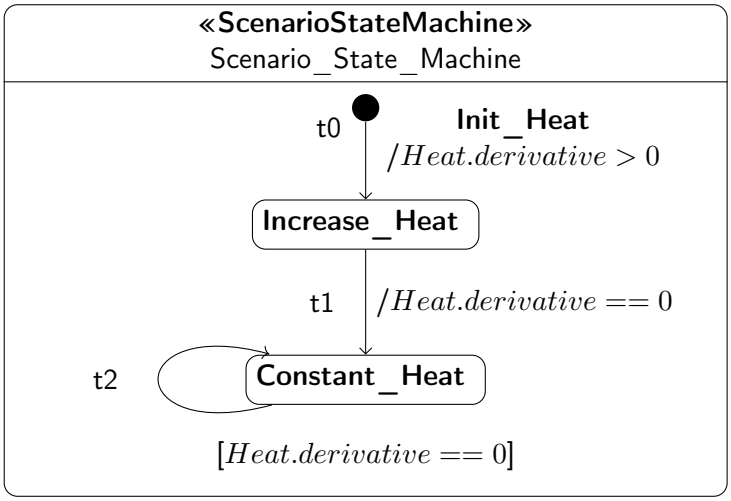

FIGURE 6.9 – Automate du scénario de simulation du système de refroidissement

6.3.3 Régulation du système de refroidissement

La régulation du système de refroidissement est modélisée par une machine à états comme le montre la Figure 6.10 avec 3 états et 7 transitions :

— la transition $t0$ initialise la valeur de $Tcold$ à $= Tc$;

— la transition $t1$ diminue la variation du débit en déclenchant le mode Decrease_Rate si la valeur de $Tcold < Tc$;

— la transition $t2$ rend la variation du débit constante en étant dans un mode Constant_Rate si la valeur de $Tcold = Tc$;

— la transition $t3$ diminue la variation du débit en déclenchant le mode Decrease_Rate si la valeur de $Tcold$ est encore $< Tc$;

— la transition $t4$ augmente la variation du débit en déclenchant le mode Increase_Rate si la valeur de $Tcold > Tc$;

— la transition $t6$ augmente la variation du débit en déclenchant le mode Increase_Rate si la valeur de $Tcold$ est encore $> Tc$.

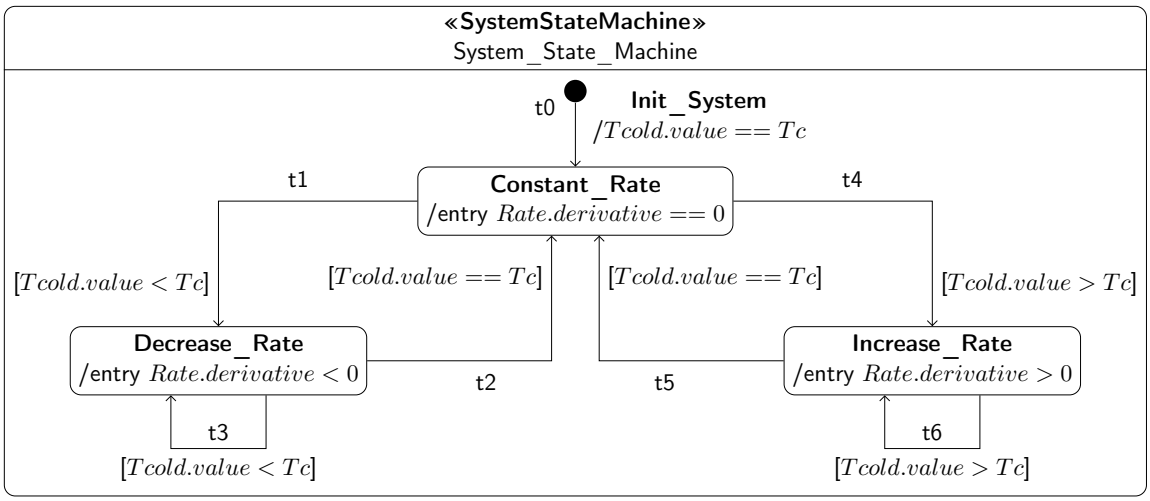

FIGURE 6.10 – Automate de régulation du système de refroidissement

6.3.4 Comportement du système de refroidissement

Comme expliqué dans la section 4.4.9 page 61, le comportement continu du système de refroidissement est modélisé par 4 relations :

— **Relation_1** = $\dot{T}hot$ **thres** 0 **is CPROP** $\dot{H}eat$ **thres** 0 ;

— **Relation_2** = $\dot{T}cold$ **thres** 0 **is PROP** $\dot{T}hot$ **thres** 0 ;

— **Relation_3** = $\dot{T}hot$ **thres** 0 **is PROP** $\dot{T}cold$ **thres** 0 ;

— **Relation_4** = $\ddot{T}cold$ **thres** 0 **is CIPROP** $\dot{R}ate$ **thres** 0.

Ces relations peuvent être modélisées dans un diagramme paramétrique comme le montre la Figure 6.11. La **Relation_1** a deux propriétés : la propriété **cause** est connectée à la dérivée première de **Heat** qui appartient au block **HeatSource**, la propriété **effect** est connectée à la dérivée première de **Thot** qui appartient au block **Cooling**. Les seuils des relations sont spécifiés dans la propriété du stéréotype **QualitativeConstraintProperty** appliqué à l'opérateur comme le montre la Figure 6.12.

6.3.5 Modèle HyDiv du système de refroidissement

La transformation QVT opérationnelle transforme le modèle SysML du système de refroidissement en un modèle HyDiv comme le montre la Figure 6.13.

6.3.6 Modèle Diversity du système de refroidissement

La transformation Acceleo produit à partir du modèle HyDiv du système de refroidissement le code source Xlia du modèle Diversity du système. En effet, elle génère

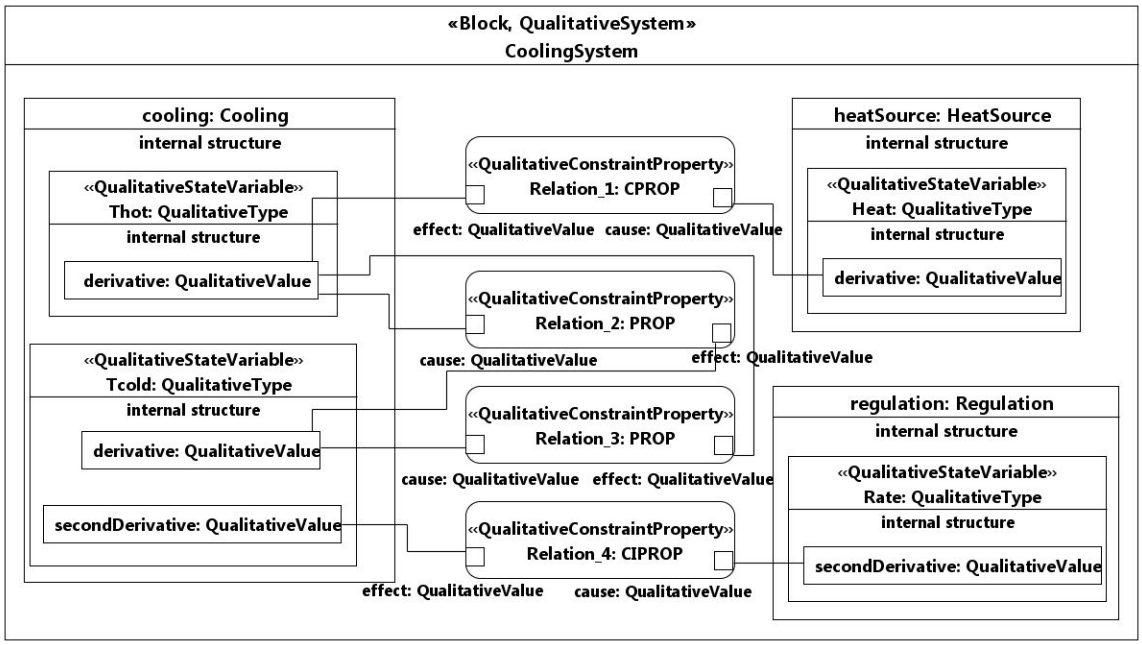

FIGURE 6.11 – Relations qualitatives du système de refroidissement

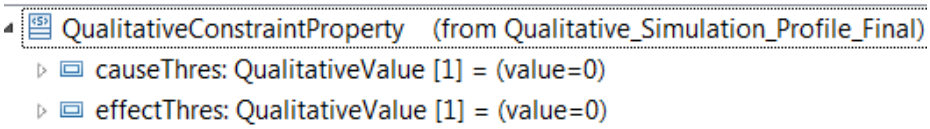

FIGURE 6.12 – Propriétés de l'opérateur

les différents automates du modèle d'exécution : l'automate du scénario, les 4 automates des relations, les automates de la dérivée seconde de $Tcold$, les automates de la dérivée première de $Heat, Thot, Tcold, Rate$, l'automate de la valeur de $Tcold$, les automates de validation des 4 relations et à la fin l'automate de régulation du système de refroidissement.

6.3.7 Comportements qualitatifs obtenus

En exécutant le module d'analyse des traces brutes générées par Diversity, on obtient 7 comportements qualitatifs pour le système de refroidissement en n'observant que la variable $Tcold$. La Figure 6.14 montre un comportement oscillatoire. La couleur rouge montre un changement de valeur de la variable, la couleur verte montre le

contexte d'exécution (EC) vers lequel le dernier EC reboucle. $Tcold$ est $= Tc$ dans $EC3$, puis $Tcold$ devient $> Tc$ dans $EC7$, la température froide devient de nouveau $= Tc$ dans $EC27$, elle devient $< Tc$ dans $EC29$ et après elle reboucle de nouveau vers $EC3$ où elle devient $= Tc$. Si on veut voir l'impact de la variation de $Tcold$ sur

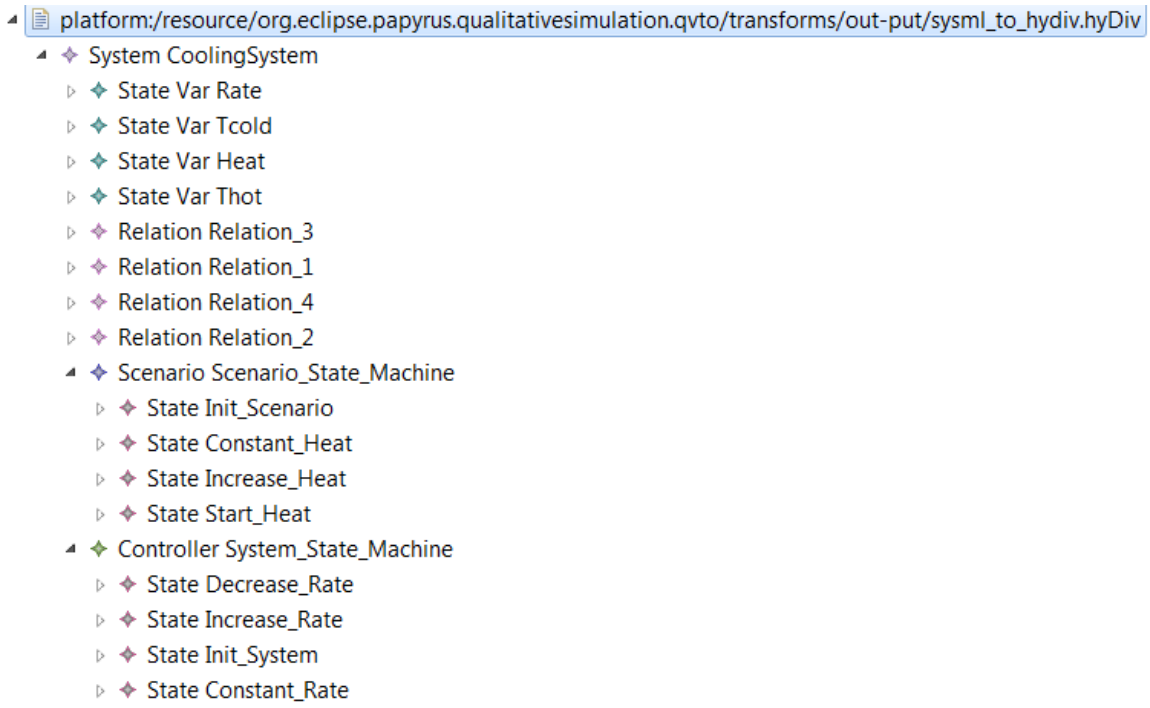

FIGURE 6.13 – Modèle HyDiv du système de refroidissement

QUALITATIVE BEHAVIOR 2

behavior		complete
EC / QV	Tcold	
EC 3	= Tc	complete_behavior 1.2.1
EC 7	> Tc	complete_behavior 1.2.2
EC 27	= Tc	complete_behavior 1.2.3
EC 29	< Tc	

FIGURE 6.14 – Comportement oscillatoire de Tcold

les autres variables, on peut ajouter la variation de la dérivée première du Débit *Rate* comme le montre la Figure 6.15.

Dans cette figure, on voit clairement que quand $Tcold > Tc$ dans $EC7$, *Rate* commence à augmenter (la dérivée première de *Rate* est > 0 dans $EC9$). Quand

QUALITATIVE BEHAVIOR 2

behavior			complete
EC / QV	Tcold	Rate '	
EC 3	= Tc	= 0	
EC 7	> Tc	= 0	
EC 9	> Tc	> 0	complete_behavior 1.2.1
EC 27	= Tc	> 0	complete_behavior 1.2.2
EC 29	< Tc	= 0	complete_behavior 1.2.3
EC 31	< Tc	< 0	
EC 42	= Tc	< 0	

FIGURE 6.15 – Comportement oscillatoire de Tcold et Rate

$Tcold = Tc$ dans $EC27$, le $Rate$ arrête d'augmenter ($\dot{R}ate = 0$ dans $EC29$). Quand $Tcold < Tc$ dans $EC29$, le $Rate$ commence à diminuer ($\dot{R}ate < 0$ dans $EC31$). Il y a toujours un décalage entre la variation de $Tcold$ et $Rate$. Ceci est dû au fait que l'automate qui contrôle la variation de $Rate$ est exécuté en dernier. Cet automate observe les sorties de notre modèle d'exécution de contraintes qualitatives et change par la suite les variables que nous voulons réguler. La colonne complète montrée dans les Figures 6.14 et 6.15 est utilisée pour stocker des liens hypertextes des fichiers de traces qui incluent toutes les variables d'états du système. Les comportements qualitatifs montrés dans le champ *behavior field* sont extraits à partir de ces traces. La Figure 6.16 montre un exemple d'une trace qualitative complète. Dans cette figure, on voit toutes les variables d'états du système : $Heat$, $Thot$, $Tcold$, $Rate$ et aussi la vérification des relations dans la colonne de vérification. Le EC vers lequel on reboucle diffère sur les Figures 6.14 ($EC3$), 6.15 ($EC7$) et 6.16 ($EC29$), bien que ces trois figures représentent le même comportement qualitatif.

Le comportement de la Figure 6.14 est raffiné par celui de la Figure 6.15 et le comportement de la Figure 6.15 est raffiné par celui de la Figure 6.16. La détection du EC vers lequel on reboucle dépend des variables observées.

COMPLETE BEHAVIOR 3

EC / QV	Heat'	Thot'	Thot_QualVar	Tcold''	Tcold'	Tcold	Tcold_QualVar	Rate'	Rate_QualVar	Verif_1	Verif_2	Verif_3	Verif_4
EC 3	= 0	= 0	Constant_Thot	= 0	= 0	= Tc	Constant_Tcold	= 0	Constant_Rate	true	true	true	true
EC 4	> 0	> 0	Start_Increase_Thot	= 0	= 0	= Tc	Constant_Tcold	= 0	Constant_Rate	true	false	false	true
EC 5	= 0	> 0	Increase_Thot	= 0	> 0	= Tc	Start_Increase_Tcold	= 0	Constant_Rate	false	true	true	true
EC 7	= 0	> 0	Increase_Thot	= 0	> 0	> Tc	Increase_Tcold	= 0	Constant_Rate	false	true	true	true
EC 9	= 0	> 0	Increase_Thot	< 0	> 0	> Tc	Dec_Increase_Tcold	> 0	Start_Increase_Rate	false	true	true	true
EC 11	= 0	> 0	Increase_Thot	< 0	> 0	> Tc	Dec_Increase_Tcold	> 0	Increase_Rate	false	true	true	true
EC 16	= 0	> 0	Increase_Thot	< 0	= 0	> Tc	Max_Tcold	> 0	Increase_Rate	false	false	false	true
EC 19	= 0	> 0	Increase_Thot	< 0	< 0	> Tc	Start_Decrease_Tcold	> 0	Increase_Rate	false	false	false	true
EC 21	= 0	> 0	Increase_Thot	< 0	< 0	> Tc	Decrease_Tcold	> 0	Increase_Rate	false	false	false	true
EC 23	= 0	> 0	Increase_Thot	< 0	< 0	> Tc	Inc_Decrease_Tcold	> 0	Increase_Rate	false	false	false	true
EC 26	= 0	= 0	Stop_Increase_Thot	< 0	< 0	> Tc	Inc_Decrease_Tcold	> 0	Increase_Rate	true	false	false	true
EC 27	= 0	< 0	Start_Decrease_Thot	< 0	< 0	= Tc	Inc_Decrease_Tcold	> 0	Increase_Rate	true	true	true	true
EC 29	= 0	< 0	Decrease_Thot	= 0	< 0	< Tc	Decrease_Tcold	= 0	Stop_Increase_Rate	true	true	true	true
EC 31	= 0	< 0	Decrease_Thot	> 0	< 0	< Tc	Dec_Decrease_Tcold	< 0	Start_Decrease_Rate	true	true	true	true
EC 32	= 0	< 0	Decrease_Thot	> 0	< 0	< Tc	Dec_Decrease_Tcold	< 0	Decrease_Rate	true	true	true	true
EC 35	= 0	< 0	Decrease_Thot	> 0	= 0	< Tc	Min_Tcold	< 0	Decrease_Rate	true	false	false	true
EC 36	= 0	< 0	Decrease_Thot	> 0	> 0	< Tc	Start_Increase_Tcold	< 0	Decrease_Rate	true	false	false	true
EC 38	= 0	< 0	Decrease_Thot	> 0	> 0	< Tc	Increase_Tcold	< 0	Decrease_Rate	true	false	false	true
EC 40	= 0	< 0	Decrease_Thot	> 0	> 0	< Tc	Inc_Increase_Tcold	< 0	Decrease_Rate	true	false	false	true
EC 42	= 0	< 0	Decrease_Thot	> 0	> 0	= Tc	Inc_Increase_Tcold	< 0	Decrease_Rate	true	false	false	true
EC 46	= 0	< 0	Decrease_Thot	= 0	> 0	> Tc	Increase_Tcold	= 0	Stop_Decrease_Rate	true	false	false	true
EC 50	= 0	< 0	Decrease_Thot	< 0	> 0	> Tc	Dec_Increase_Tcold	> 0	Start_Increase_Rate	true	false	false	true
EC 52	= 0	< 0	Decrease_Thot	< 0	> 0	> Tc	Dec_Increase_Tcold	> 0	Increase_Rate	true	false	false	true
EC 57	= 0	< 0	Decrease_Thot	< 0	= 0	> Tc	Max_Tcold	> 0	Increase_Rate	true	false	false	true
EC 61	= 0	= 0	Stop_Decrease_Thot	< 0	< 0	> Tc	Start_Decrease_Tcold	> 0	Increase_Rate	true	false	false	true
EC 80	= 0	< 0	Start_Decrease_Thot	< 0	< 0	= Tc	Decrease_Tcold	> 0	Increase_Rate	true	true	true	true

FIGURE 6.16 – Trace qualitative complète

6.4 Conclusion

Dans ce chapitre, nous avons présenté notre chaîne d'outils pour la validation fonctionnelle des systèmes hybrides. Cette chaîne d'outils a été mise en place pour automatiser notre méthodologie d'analyse qualitative des systèmes hybrides. Elle se base sur des techniques de l'ingénierie dirigée par les modèles : une transformation M2M (Model to Model) depuis le modèle SysML avec le profil de la simulation qualitative appliqué vers le langage pivot HyDiv et une transformation M2T (Model to Text) depuis ce langage pivot vers Diversity. Nous avons ajouté un module d'analyse de traces pour filtrer les traces symboliques qui correspondent au même comportement physique dans l'arbre d'exécution généré par Diversity. Ce module sert aussi à filtrer les traces qui ne respectent pas l'influence physique des relations. Nous avons appliqué notre méthodologie sur le système de refroidissement de température. Nous avons montré que les comportements qualitatifs du système peuvent être raffinés à différents niveaux d'abstraction. La génération des comportements qualitatifs est basée sur les variables choisies par l'utilisateur. Pour comprendre les raisons d'obtention de tels comportements abstraits, un raffinement est possible en augmentant le nombre de variables choisies.

Chapitre 7

Cas d'étude : Plateforme de gestion thermique d'un véhicule hybride

Dans ce chapitre, nous présentons le cas d'étude industriel : Plateforme de gestion thermique d'un véhicule hybride de la société *Sherpa* [5]. Nous détaillons les différentes étapes de l'application de notre méthodologie sur ce cas d'étude et présentons les résultats obtenus.

7.1 Présentation de la plateforme de gestion thermique d'un véhicule hybride

La plateforme de gestion thermique d'un véhicule hybride comme le montre la Figure 7.1 se compose principalement :

— d'une *cabine* représentant le confort thermique d'un passager ;

— d'*équipements de conditionnement thermique* représentant les équipements qui demandent un conditionnement thermique ;

— d'*entités de production et distribution d'énergie thermique* produisant et distribuant l'énergie thermique ;

— d'une *fonction de contrôle* gérant les fonctions du système.

La cabine et les équipements envoient à l'entité de production et de distribution d'énergie leur besoin en énergie et cette dernière fonction fournit l'énergie demandée par ces deux consommateurs.

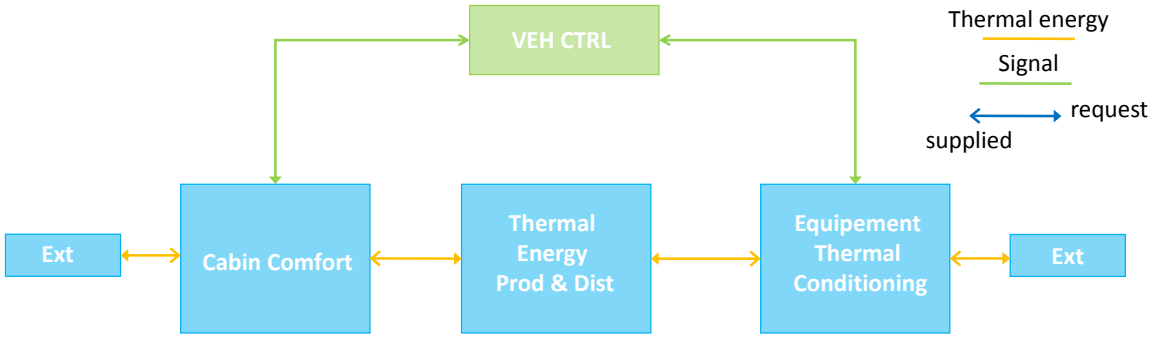

FIGURE 7.1 – L'architecture fonctionnelle globale de la plateforme de gestion thermique

En détaillant chaque fonction comme illustré par la Figure 7.2, nous arrivons à 3 niveaux de besoin en énergie thermique :

— conditionnement thermique pour la batterie à 40 $^{\circ}C$, assuré par le *système de refroidissement 1* ;

— conditionnement thermique des machines thermiques à 100 $^{\circ}C$, assuré par le *système de refroidissement 2* ;

— conditionnement thermique de la cabine suivant le conducteur (20 $^{\circ}C$), assuré par le *système de réfrigération* et *le système de conditionnement d'air*.

7.2 Modélisation qualitative de la plateforme de gestion thermique d'un véhicule hybride

Dans notre approche pour la modélisation qualitative de la plateforme de gestion thermique, nous avons opté pour une décomposition architecturale du système comme le montre la Figure 7.2. En effet, la cas d'étude est composé principalement :

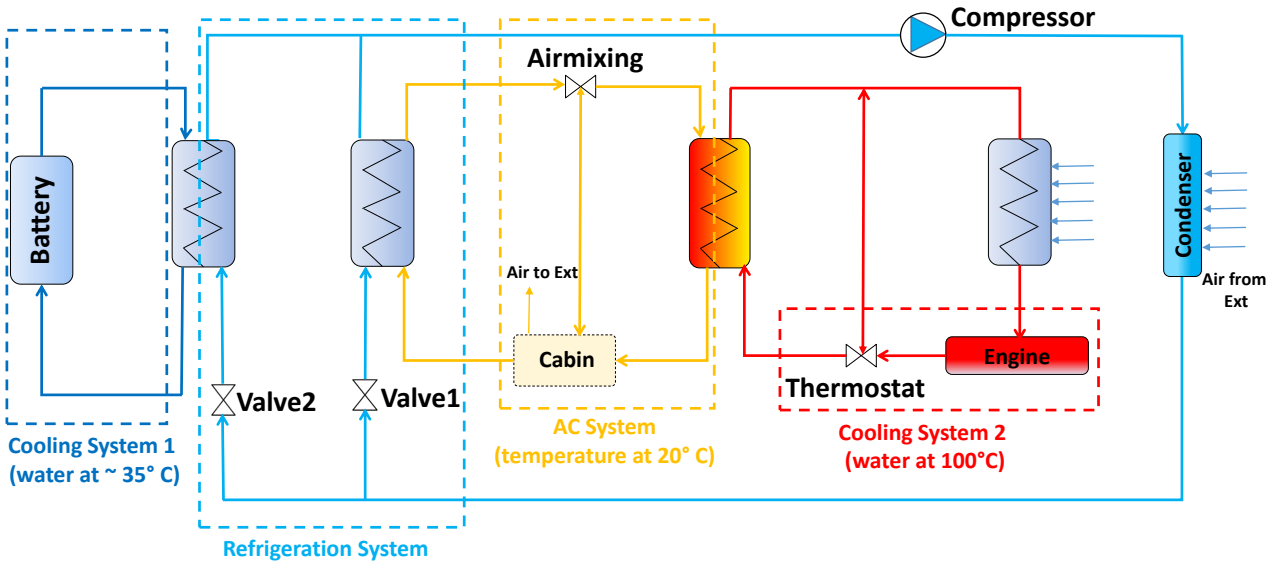

FIGURE 7.2 – L'architecture physique de la plateforme de gestion thermique

— *circuit batterie*, la température de l'eau dans le circuit de refroidissement à la sortie de la batterie doit être autour de 35 $^{\circ}C$;

— *système de réfrigération*, ce système contient les deux vannes de régulation $Valve1$, $Valve2$, un compresseur et un condenseur ;

— *cabine*, la température de la cabine doit être autour de 20 $^{\circ}C$;

— *circuit de moteur*, la température à la sortie du moteur doit être autour de 100 $^{\circ}C$.

Ce système est composé de sept variables d'états :

— *Airmixing*, la vanne qui régule la température de la cabine $Tcab$;

— $Tcab$, la température de la cabine qui doit être égale à son seuil Tc (numériquement 20 $^{\circ}C$) ;

— *Tengine*, la température à la sortie du moteur qui doit être égale à son seuil Te $(100\ ^{\circ}C)$;

— *Thermostat*, la vanne qui régule la température à la sortie du moteur $Tengine$;

— *Valve1*, la vanne qui régule la température de la cabine $Tcab$;

— *Valve2*, la vanne qui régule la température à la sortie de la batterie $Tbat$;

— *Tbat*, la température à la sortie de la batterie qui doit être égale à son seuil Tb $(35\ ^{\circ}C)$.

Le but du système est de maintenir les températures des variables $Tcab$, $Tbat$ et $Tengine$ égales à leur températures de consignes respectivement Tc, Tb et Te. En effet, la régulation de $Tbat$ se fait par l'ouverture et la fermeture de la vanne $Valve2$. Celle de la cabine $Tcab$ se fait par l'ouverture et la fermeture des vannes $Valve1$ et *Airmixing*. Celle du moteur $Tengine$ se fait par l'ouverture et la fermeture de la vanne du $Thermostat$. Nous détaillerons par la suite le processus de régulation de chaque variable.

Dans cette plateforme de gestion thermique, nous ne disposons pas des équations différentielles qui décrivent son fonctionnement. Par le language que nous avons proposé, nous pouvons établir un modèle qualitatif de relations qui décrit la dynamique du système :

— **relation 1** = $\dot{T}engine$ **thres** 0 **is CIPROP** $Thermostat$ **thres** T ;

— **relation 2** = $\dot{T}cab$ **thres** 0 **is CPROP** $\dot{T}engine$ **thres** 0 ;

— **relation 3** = $\dot{T}cab$ **thres** 0 **is CPROP** $Airmixing$ **thres** A ;

— **relation 4** = $\dot{T}cab$ **thres** 0 **is CIPROP** $Valve1$ **thres** $V1$;

— **relation 5** = $\dot{T}bat$ **thres** 0 **is CIPROP** $Valve2$ **thres** $V2$.

La **relation 1** modélise le fait que la variation du *Thermostat* influence la variation de la température du moteur *Tengine* d'une manière inversement proportionnelle. Nous voulons réguler *Tengine* uniquement si le *Thermostat* change de valeur qualitative. Nous avons choisi alors l'opérateur **CIPROP**. Les **relations 2** et **3** modélisent la température de la cabine *Tcab* variant d'une manière proportionnelle par rapport à la température du moteur *Tengine* et la valeur de la vanne de *Airmixing*. En effet, si nous ouvrons de plus en plus la vanne de *Airmixing*, la température de la cabine *Tcab* augmente. Aussi, si la température du moteur *Tengine* diminue alors *Tcab* subit une variation dans le même sens. La **relation 4** modélise la variation de la température de cabine *Tcab* étant inversement proportionnelle à celle de la valeur de la vanne *Valve1*. La vanne *Valve1* contient de l'eau froide. Après évaporation, si nous injectons

de plus en plus d'air froid, la température de la cabine *Tcab* diminue de plus en plus. La **relation 5** modélise la température de la batterie *Tbat* variant d'une manière inversement proportionnelle par rapport à la vanne *Valve2*. La vanne *Valve2* contient de l'eau froide. Si nous injectons de plus en plus d'eau froide, la température de la batterie *Tbat* diminue de plus en plus.

7.3 Architecture SysML de la plateforme de gestion thermique d'un véhicule hybride

L'architecture de la plateforme de gestion thermique est modélisée par un diagramme paramétrique comme illustré par la Figure 7.3. Ce système est stéréotypé par **QualitativeSystem**. Il se compose de cinq **QualitativeBlock** :

— le bloc de la cabine qui contient deux **QualitativeStateVariable**, Airmixing et Tcab ;

— le bloc du circuit moteur qui contient deux **QualitativeStateVariable**, Tengine et Thermostat ;

— le bloc du système de réfrigération qui contient deux **QualitativeStateVariable**, Valve1 et Valve2 ;

— le bloc du circuit de batterie qui contient une **QualitativeStateVariable** Tbat ;

— le bloc du scénario **ScenarioStateMachine** qui spécifie le comportement de l'entrée du système.

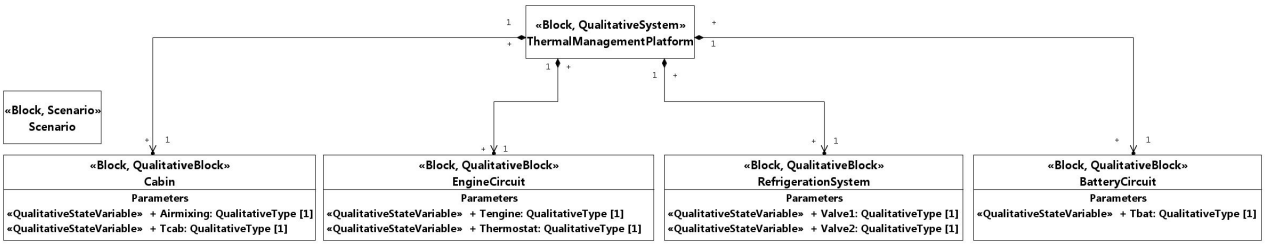

FIGURE 7.3 – Diagramme de définition de blocs de la plateforme de gestion thermique d'un véhicule hybride

7.4 Mode de fonctionnement de la plateforme de gestion thermique d'un véhicule hybride

Nous distinguons trois modes de fonctionnement de la plateforme de gestion thermique d'un véhicule hybride :

— *Mode électrique*, dans lequel la plateforme fonctionne entièrement grâce à son circuit batterie et par conséquent nous négligeons la partie du circuit moteur ;

— *Mode thermique*, dans lequel la plateforme fonctionne entièrement grâce à son circuit moteur et par conséquent nous négligeons la partie du circuit batterie ;

— *Mode hybride*, dans lequel la plateforme fonctionne grâce à son circuit moteur et son circuit batterie.

7.4.1 Mode électrique

Lorsque la plateforme de gestion thermique fonctionne en mode électrique, nous ne considérons que les 3 dernières relations :

— **relation 3** $= \dot{T}cab$ **thres** 0 **is CPROP** $Airmixing$ **thres** A ;

— **relation 4** $= \dot{T}cab$ **thres** 0 **is CIPROP** $Valve1$ **thres** $V1$;

— **relation 5** $= \dot{T}bat$ **thres** 0 **is CIPROP** $Valve2$ **thres** $V2$.

7.4.1.1 Régulation de la plateforme en mode électrique

La régulation de la vanne de $Airmixing$ se fait par un automate avec 3 états et 7 transitions comme illustré par la Figure 7.4 :

— la transition $t0$ initialise la valeur de $Tcab$ à $= Tc$;

— la transition $t1$ diminue l'ouverture de la vanne $Airmixing$ en déclenchant le mode Decrease_Airmixing si la valeur de $Tcab < Tc$;

— la transition $t2$ rend l'ouverture de la vanne $Airmixing$ constante en étant dans un mode Constant_Airmixing si la valeur de $Tcab = Tc$;

— la transition $t3$ diminue l'ouverture de la vanne $Airmixing$ en déclenchant le mode Decrease_Airmixing si la valeur de $Tcab$ est encore $< Tc$;

— la transition $t4$ augmente l'ouverture de la vanne $Airmixing$ en déclenchant le mode Increase_Airmixing si la valeur de $Tcab > Tc$;

— la transition $t6$ augmente l'ouverture de la vanne $Airmixing$ en déclenchant le mode Increase_Airmixing si la valeur de $Tcab$ est encore $> Tc$.

La régulation de la vanne de $Valve2$ se fait par un automate avec 3 états et 7 transitions comme illustré par la Figure 7.5 :

— la transition $t0$ initialise la valeur de $Tbat$ à $= Tb$;

— la transition $t1$ diminue l'ouverture de la vanne $Valve2$ en déclenchant le mode Decrease_Valve2 si la valeur de $Tbat < Tb$ ou $Valve1 > Valve1_{-1}$;

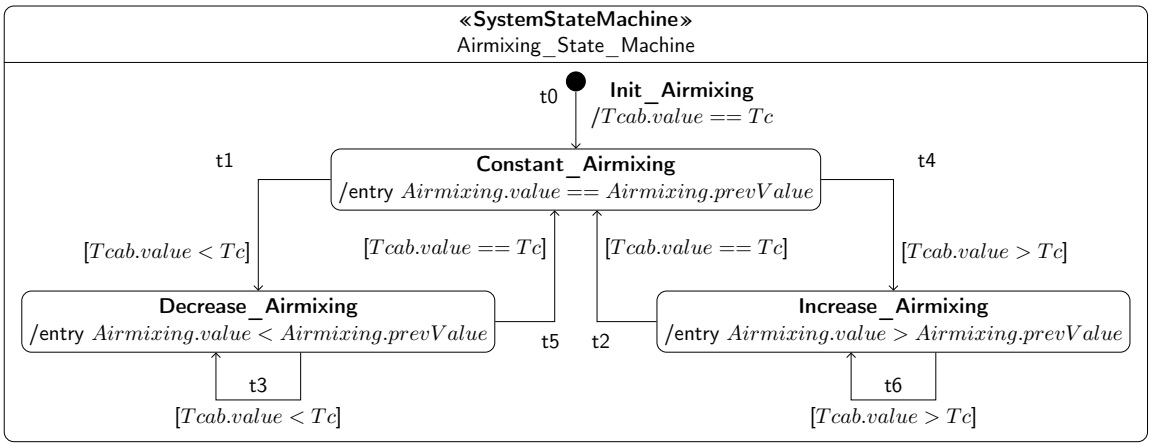

FIGURE 7.4 – Automate de régulation de Airmixing

— la transition $t2$ rend l'ouverture de la vanne $Valve2$ constante en étant dans un mode Constant_Valve2 si la valeur de $Tbat = Tb$ ou $Valve1 == Valve1_{-1}$;

— la transition $t3$ diminue l'ouverture de la vanne $Valve2$ en déclenchant le mode Decrease_Valve2 si la valeur de $Tbat$ est encore $< Tb$ ou $Valve1 > Valve1_{-1}$;

— la transition $t4$ augmente l'ouverture de la vanne $Valve2$ en déclenchant le mode Increase_Valve2 si la valeur de $Tbat > Tb$ ou $Valve1 < Valve1_{-1}$;

— la transition $t6$ augmente l'ouverture de la vanne $Valve2$ en déclenchant le mode Increase_Valve2 si la valeur de $Tbat$ est encore $> Tb$ ou $Valve1 < Valve1_{-1}$.

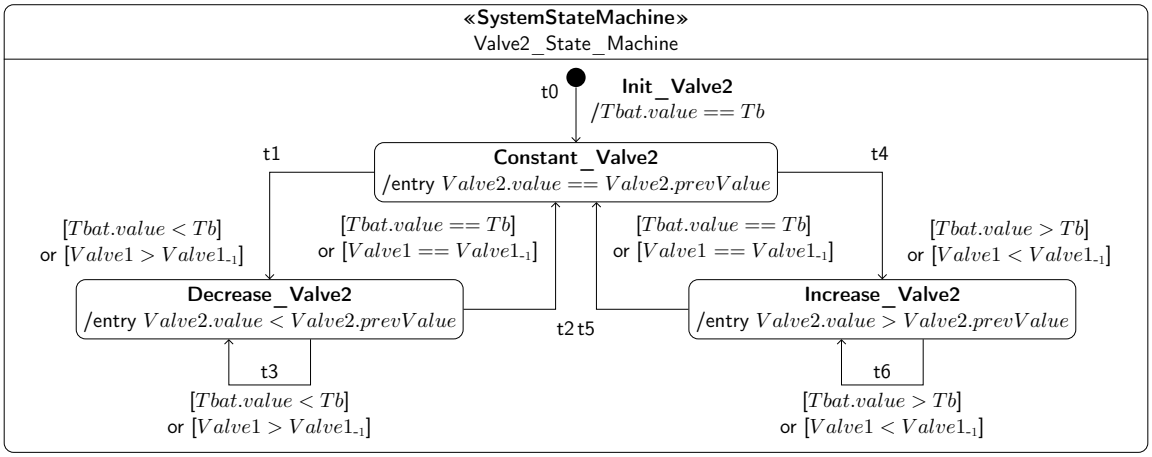

FIGURE 7.5 – Automate de régulation de la vanne Valve2

La régulation de la vanne de $Valve1$ se fait par un automate avec 3 états et 7 transitions comme illustré par la Figure 7.6 :

— la transition $t0$ initialise la valeur de $Tcab$ à $= Tc$;

— la transition $t1$ diminue l'ouverture de la vanne $Valve1$ en déclenchant le mode Decrease_Valve1 si la valeur de $Tcab < Tc$ ou $Valve2 > Valve2_{-1}$;

— la transition $t2$ rend l'ouverture de la vanne $Valve1$ constante en étant dans un mode Constant_Valve1 si la valeur de $Tcab = Tc$ ou $Valve2 == Valve2_{-1}$;

— la transition $t3$ diminue l'ouverture de la vanne $Valve1$ en déclenchant le mode Decrease_Valve1 si la valeur de $Tcab$ est encore $< Tc$ ou $Valve2 > Valve2_{-1}$;

— la transition $t4$ augmente l'ouverture de la vanne $Valve1$ en déclenchant le mode Increase_Valve1 si la valeur de $Tcab > Tc$ ou $Valve2 < Valve2_{-1}$;

— la transition $t6$ augmente l'ouverture de la vanne $Valve1$ en déclenchant le mode Increase_Valve1 si la valeur de $Tcab$ est encore $> Tc$ ou $Valve2 < Valve2_{-1}$.

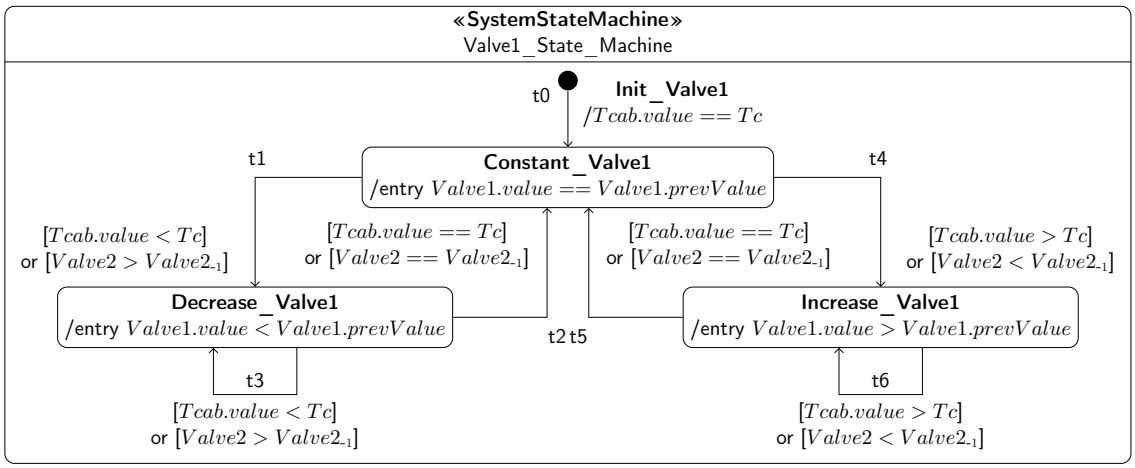

FIGURE 7.6 – Automate de régulation de la vanne Valve1

7.4.1.2 Scénario de la plateforme en mode électrique

Nous avons choisi un scénario pour la plateforme de gestion thermique en mode électrique qui peut être modélisé par une machine à états comme le montre la Figure 7.7 avec 2 états, 1 pseudo-état et 4 transitions :

— la transition $t0$ initialise la valeur de la température de cabine $Tcab$ à $> Tc$;

— la transition $t1$ stabilise la valeur de la température de cabine $Tcab$ à $= Tc$;

— la transition $t2$ renforce la stabilité de la température de cabine $Tcab$ à $= Tc$;

— la transition $t3$ renforce la valeur de la température de cabine $Tcab$ à $> Tc$.

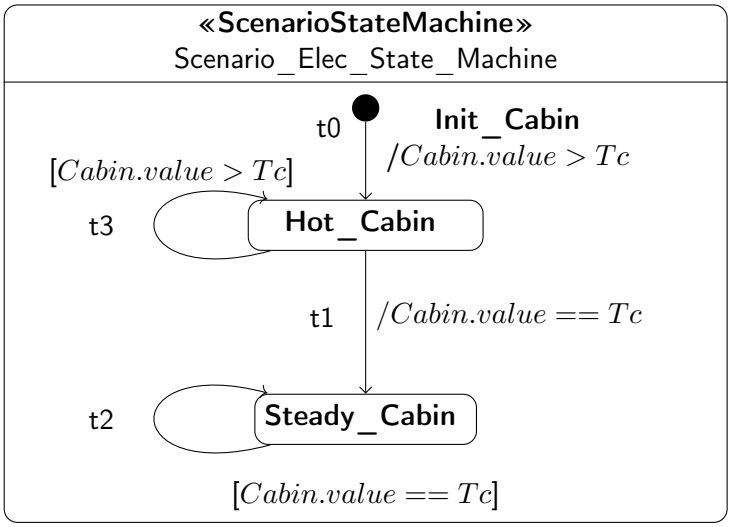

FIGURE 7.7 – Automate du scénario de simulation de la plateforme en mode électrique

7.4.1.3 Comportements de la plateforme en mode électrique

Le comportement continu de la plateforme de gestion thermique en mode électrique est modélisé par un diagramme paramétrique comme illustré par la Figure 7.8. Prenons l'exemple de la **relation 4**, la propriété **cause** est liée à la valeur de $Valve1$ dans le système de réfrigération et la propriété **effect** est liée à la dérivée première de $Tcab$ dans la cabine. Les seuils des relations sont spécifiés dans les propriétés du stéréotype *QualitativeConstraintProperty* de l'opérateur comme montré par la Figure 7.9.

7.4.1.4 Analyse des comportements de la plateforme en mode électrique

Nous analysons les traces brutes de simulation symboliques à l'aide du module d'analyse de notre chaîne d'outils. Par l'observation de la valeur de la température de cabine $Tcab$, nous obtenons deux comportements abstraits : le premier indique que la cabine est chaude (sa température reste $> Tc$) et le deuxième indique que la cabine est stable (sa température reste $= Tc$) comme illustré par la Figure 7.10.

Pour voir l'influence de ce comportement sur les autres variables, nous pouvons observer la variation de la vanne $Valve1$ qui régule la température de la cabine. Nous obtenons deux comportements illustrés par les Figures 7.11 et 7.12. Lorsque la température de la cabine devient $> Tc$ dans le contexte d'exécution ($EC4$), la vanne $Valve1$ s'ouvre plus et sa valeur devient $> V1$ dans ($EC5$). $Valve1$ reste $> V1$ dans ($EC36$) et ($EC182$) bien que la température de la cabine $Tcab$ soit $= Tc$.

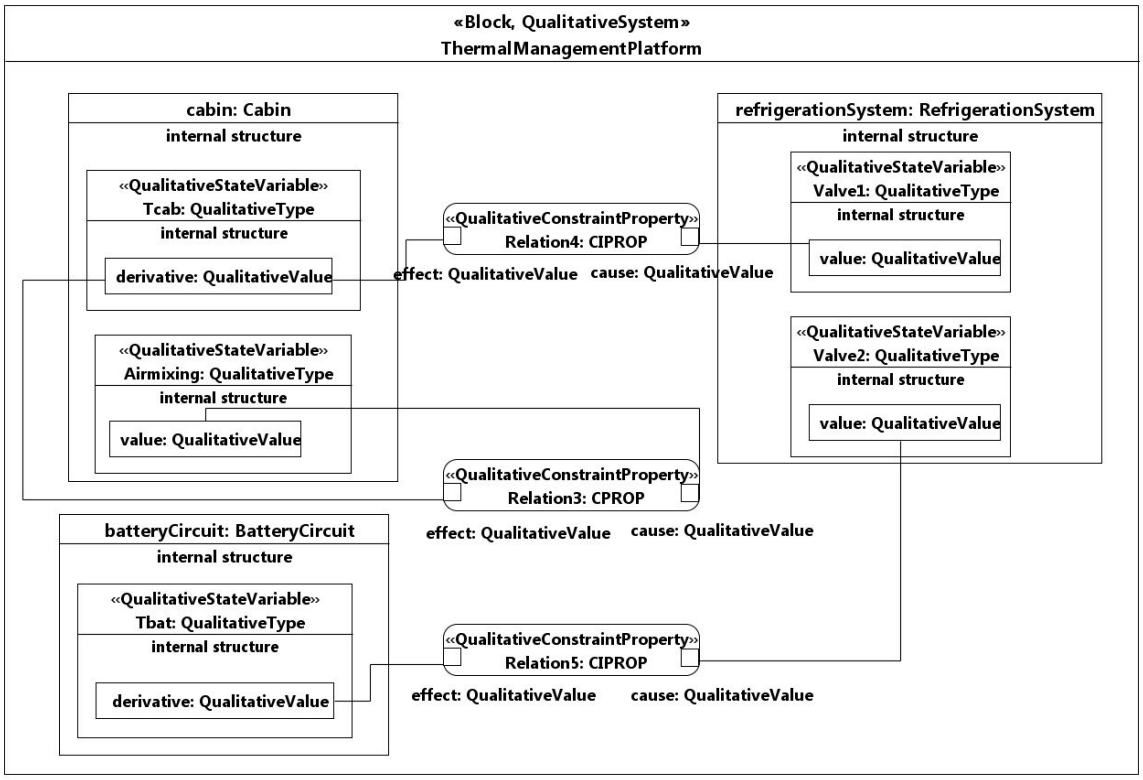

FIGURE 7.8 – Relations qualitatives de la plateforme de gestion thermique en mode électrique

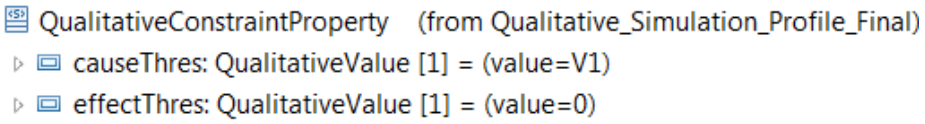

FIGURE 7.9 – Propriétés de la relation 4

Pour comprendre ce phénomène, nous pouvons raffiner les comportements de la Figure 7.11 et 7.12. Par l'observation de la vanne $Valve2$, nous obtenons 4 comportements illustrés par les Figures 7.13, 7.14 7.15 et 7.16.

Dans ces figures, quand la vanne $Valve1$ s'ouvre de plus en plus dans $(EC5)$, la vanne $Valve2$ diminue sa valeur de plus en plus en devenant $< V2$ dans $EC7$. En effet, les deux vannes fonctionnent d'une manière inversement proportionnelle. La valeur de la température $Tbat$ devient $> Tb$ dans $(EC13)$. Ainsi, la $Valve2$ passe par $= V2$ dans $(EC17)$ pour devenir par la suite dans $(EC23) > V2$ afin de réguler la température de la batterie. Quand la température de la cabine $Tcab$ se stabilise dans $(EC36)$, la valeur de la vanne $Valve1$ devient $= V1$ dans $(EC38)$. Elle devient $< V1$ dans $(EC46)$ parce que la valeur de la $valve2$ est $> V2$ dans le contexte

QUALITATIVE BEHAVIOR 2

behavior	complete
EC / QV | Tcab EC 3 | = Tc EC 4 | > Tc EC 99 | = Tc	complete_behavior 1.2.1 complete_behavior 1.2.2 complete_behavior 1.2.3 complete_behavior 1.2.4 complete_behavior 1.2.5 complete_behavior 1.2.6 complete_behavior 1.2.7 complete_behavior 1.2.8 complete_behavior 1.2.9 complete_behavior 1.2.10 complete_behavior 1.2.11 complete_behavior 1.2.12 complete_behavior 1.2.13 complete_behavior 1.2.14 complete_behavior 1.2.15 complete_behavior 1.2.16 complete_behavior 1.2.17 complete_behavior 1.2.18 complete_behavior 1.2.19 complete_behavior 1.2.20 complete_behavior 1.2.21 complete_behavior 1.2.22 complete_behavior 1.2.23 complete_behavior 1.2.24 complete_behavior 1.2.25 complete_behavior 1.2.26 complete_behavior 1.2.27 complete_behavior 1.2.28 complete_behavior 1.2.29 complete_behavior 1.2.30

FIGURE 7.10 – Stabilisation de la cabine

QUALITATIVE BEHAVIOR 6

behavior			complete
EC / QV	Tcab	Valve1	
EC 3	= Tc	= V1	complete_behavior 1.6.1
EC 4	> Tc	= V1	
EC 5	> Tc	> V1	
EC 36	= Tc	> V1	

FIGURE 7.11 – Raffinement du comportement de la cabine stable par la vanne 1 (1)

QUALITATIVE BEHAVIOR 9

behavior			complete
EC / QV	Tcab	Valve1	
EC 3	= Tc	= V1	complete_behavior 1.9.1
EC 4	> Tc	= V1	
EC 182	= Tc	> V1	

FIGURE 7.12 – Raffinement du comportement de la cabine stable par la vanne 1 (2)

QUALITATIVE BEHAVIOR 3

behavior					complete
EC / QV	Tcab	Valve1	Valve2	Tbat	
EC 3	= Tc	= V1	= V2	= Tb	complete_behavior 1.3.1 complete_behavior 1.3.2 complete_behavior 1.3.3
EC 4	> Tc	= V1	= V2	= Tb	
EC 5	> Tc	> V1	= V2	= Tb	
EC 7	> Tc	> V1	< V2	= Tb	
EC 13	> Tc	> V1	< V2	> Tb	
EC 17	> Tc	> V1	= V2	> Tb	
EC 23	> Tc	> V1	> V2	> Tb	
EC 36	= Tc	> V1	> V2	> Tb	
EC 38	= Tc	= V1	> V2	> Tb	
EC 46	= Tc	< V1	> V2	> Tb	
EC 52	= Tc	< V1	> V2	= Tb	
EC 64	= Tc	< V1	> V2	< Tb	
EC 66	= Tc	< V1	= V2	< Tb	
EC 157	= Tc	< V1	< V2	< Tb	

FIGURE 7.13 – Raffinement du comportement de la cabine stable par les vannes 1, 2 et la température du circuit batterie (1)

QUALITATIVE BEHAVIOR 7

behavior					complete
EC / QV	Tcab	Valve1	Valve2	Tbat	
EC 3	= Tc	= V1	= V2	= Tb	complete_behavior 1.7.1
EC 4	> Tc	= V1	= V2	= Tb	
EC 5	> Tc	> V1	= V2	= Tb	
EC 7	> Tc	> V1	< V2	= Tb	
EC 13	> Tc	> V1	< V2	> Tb	
EC 17	> Tc	> V1	= V2	> Tb	
EC 23	> Tc	> V1	> V2	> Tb	
EC 36	= Tc	> V1	> V2	> Tb	
EC 38	= Tc	= V1	> V2	> Tb	
EC 46	= Tc	< V1	> V2	> Tb	
EC 52	= Tc	< V1	> V2	= Tb	
EC 64	= Tc	< V1	> V2	< Tb	
EC 67	= Tc	< V1	= V2	< Tb	
EC 171	= Tc	= V1	< V2	< Tb	
EC 177	= Tc	= V1	< V2	= Tb	
EC 181	= Tc	> V1	< V2	> Tb	

FIGURE 7.14 – Raffinement du comportement de la cabine stable par les vannes 1, 2 et la température du circuit batterie (2)

d'exécution d'avant. Dans la Figure 7.16, nous observons dans le dernier contexte d'exécution que $Valve2$ reste $> V2$. Physiquement, ce comportement est possible. En effet, l'ouverture de la vanne n'est pas suffisante pour réguler la température de la batterie. Par conséquent, la vanne $Valve1$ reste $< V1$. Dans les Figure 7.14 et 7.15, la $Valve2$ reste $< V2$ bien que la température de $Tbat$ soit $> Tb$. Ceci est dû au fait que la $Valve1$ est $< V1$. Dans cette situation, la variation de la $Valve1$ est dominante par rapport à la variation de $Tbat$. Dans la Figure 7.13, $Tbat$ reste $< Tb$. Physiquement, ce comportement est possible. En effet, la fermeture de la vanne $Valve2$ n'est pas suffisante pour ramener la température de la batterie à sa température d'équilibre.

7.4.2 Mode thermique

Lorsque la plateforme de gestion thermique fonctionne en mode thermique, nous ne considérons que les 3 premières relations :

— **relation 1** $= \dot{T}engine$ **thres** 0 **is CIPROP** $Thermostat$ **thres** T ;

— **relation 2** $= \dot{T}cab$ **thres** 0 **is CPROP** $\dot{T}engine$ **thres** 0 ;

— **relation 3** $= \dot{T}cab$ **thres** 0 **is CPROP** $Airmixing$ **thres** A.

QUALITATIVE BEHAVIOR 11

behavior					complete
EC / QV	Tcab	Valve1	Valve2	Tbat	
EC 3	= Tc	= V1	= V2	= Tb	
EC 4	> Tc	= V1	= V2	= Tb	
EC 5	> Tc	> V1	= V2	= Tb	
EC 7	> Tc	> V1	< V2	= Tb	
EC 13	> Tc	> V1	< V2	> Tb	
EC 17	> Tc	> V1	= V2	> Tb	
EC 23	> Tc	> V1	> V2	> Tb	
EC 36	= Tc	> V1	> V2	> Tb	
EC 38	= Tc	= V1	> V2	> Tb	
EC 46	= Tc	< V1	> V2	> Tb	
EC 52	= Tc	< V1	> V2	= Tb	
EC 920	= Tc	< V1	> V2	< Tb	
					complete_behavior 1.11.1

FIGURE 7.15 – Raffinement du comportement de la cabine stable par les vannes 1, 2 et la température du circuit batterie (3)

QUALITATIVE BEHAVIOR 15

behavior					complete
EC / QV	Tcab	Valve1	Valve2	Tbat	
EC 3	= Tc	= V1	= V2	= Tb	
EC 4	> Tc	= V1	= V2	= Tb	
EC 5	> Tc	> V1	= V2	= Tb	
EC 7	> Tc	> V1	< V2	= Tb	
EC 13	> Tc	> V1	< V2	> Tb	
EC 17	> Tc	> V1	= V2	> Tb	
EC 23	> Tc	> V1	> V2	> Tb	
EC 36	= Tc	> V1	> V2	> Tb	
EC 38	= Tc	= V1	> V2	> Tb	
EC 46	= Tc	< V1	> V2	> Tb	
EC 52	= Tc	< V1	> V2	= Tb	
EC 64	= Tc	< V1	> V2	< Tb	
EC 67	= Tc	< V1	= V2	< Tb	
EC 171	= Tc	= V1	< V2	< Tb	
EC 177	= Tc	= V1	< V2	= Tb	
EC 181	= Tc	> V1	< V2	> Tb	
EC 341	= Tc	= V1	= V2	> Tb	
EC 198	= Tc	= V1	> V2	> Tb	
EC 229	= Tc	< V1	> V2	> Tb	
					complete_behavior 1.15.1

FIGURE 7.16 – Raffinement du comportement de la cabine stable par les vannes 1, 2 et la température du circuit batterie (4)

7.4.2.1 Régulation de la plateforme de gestion thermique en mode thermique

La régulation de la vanne de $Tengine$ se fait par un automate avec 3 états et 7 transitions comme illustré par la Figure 7.17 :

- la transition $t0$ initialise la valeur de $Tengine$ à $= Te$;
- la transition $t1$ diminue l'ouverture de la vanne $Tengine$ en déclenchant le mode Decrease_Tengine si la valeur de $Tengine < Te$;
- la transition $t2$ rend l'ouverture de la vanne $Tengine$ constante en étant dans un mode Constant_Tengine si la valeur de $Tengine = Te$;
- la transition $t3$ diminue l'ouverture de la vanne $Tengine$ en déclenchant le mode Decrease_Tengine si la valeur de $Tengine$ est encore $< Te$;
- la transition $t4$ augmente l'ouverture de la vanne $Tengine$ en déclenchant le mode Increase_Tengine si la valeur de $Tengine > Te$;
- la transition $t6$ augmente l'ouverture de la vanne $Tengine$ en déclenchant le mode Increase_Tengine si la valeur de $Tengine$ est encore $> Te$.

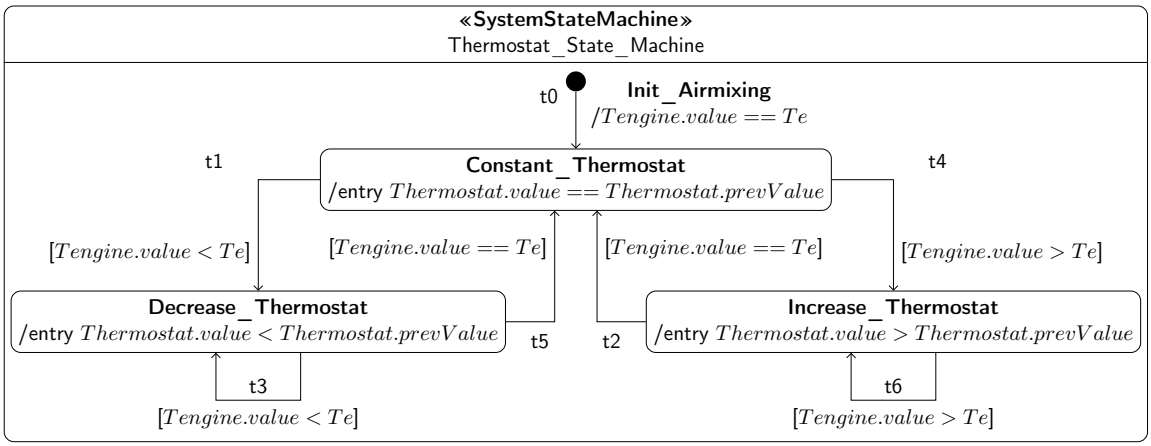

FIGURE 7.17 – Automate de régulation du Thermostat

La régulation de la vanne *Airmixing* se fait de la même manière que nous avons décrite dans la sous-section 7.4.1.1 page 99.

7.4.2.2 Scénario de la plateforme en mode thermique

Nous avons choisi un scénario pour la plateforme en mode thermique qui peut être modélisé par une machine à états comme le montre la Figure 7.18 avec 2 états, 1 pseudo-état et 4 transitions :

- la transition $t0$ initialise la valeur de la température du circuit moteur $Tengine$ à $> Te$;
- la transition $t1$ stabilise la valeur de la température du circuit moteur $Tengine$ à $= Te$;
- la transition $t2$ renforce la stabilité de la température du circuit moteur $Tengine$ à $= Te$;
- la transition $t3$ renforce la valeur de la température du circuit moteur $Tengine$ à $> Te$.

7.4.2.3 Comportements de la plateforme de gestion thermique en mode thermique

Le comportement continu de la plateforme en mode thermique est modélisé par un diagramme paramétrique comme illustré par la Figure 7.19.

Prenons l'exemple de la **relation 2**, la propriété **cause** est liée à la dérivée première de $Tengine$ dans le circuit moteur et la propriété **effect** est liée à la dérivée

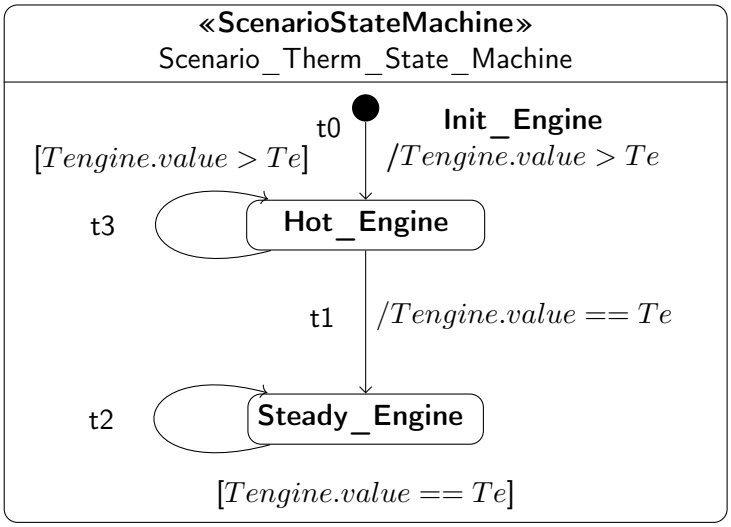

FIGURE 7.18 – Automate du scénario de simulation de la plateforme en mode thermique

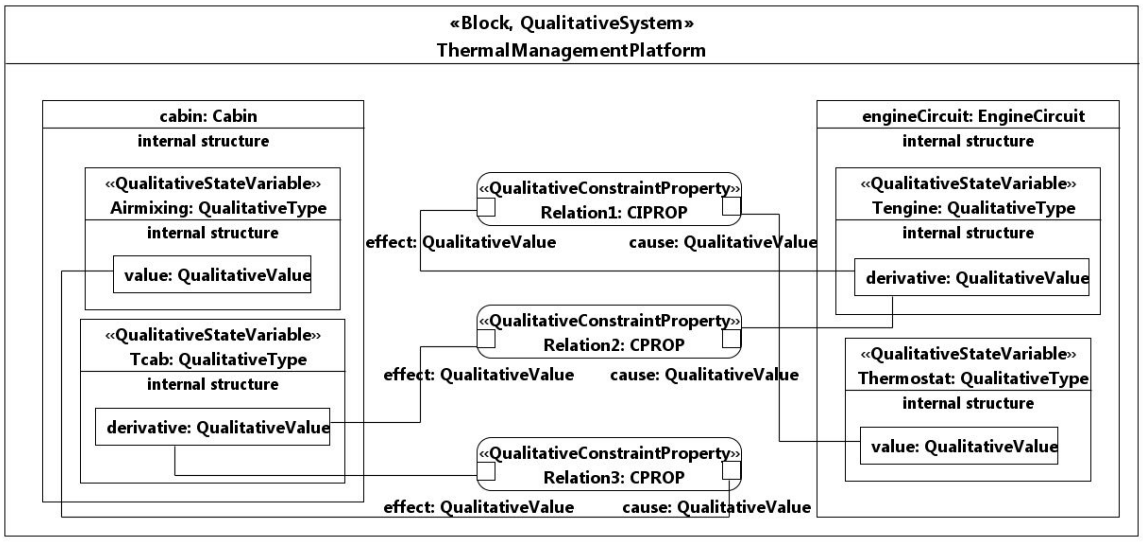

FIGURE 7.19 – Relations qualitatives de la plateforme de gestion thermique en mode thermique

première de $Tcab$ dans la cabine. Les seuils des relations sont spécifiés dans les propriétés du stéréotype *QualitativeConstraintProperty* de l'opérateur comme montré par la Figure 7.20.

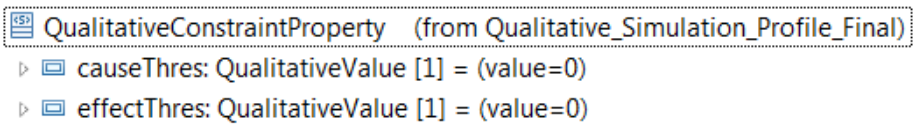

FIGURE 7.20 – Propriétés de la relation 2

7.4.2.4 Analyse des comportements de la plateforme en mode thermique

Nous analysons les traces brutes de simulation symbolique à l'aide du module d'analyse de notre chaine d'outils. Si nous n'observons que la variation de la température cabine $Tcab$, nous n'obtenons qu'un seul comportement abstrait : la température de cabine augmente puis se stabilise et par la suite diminue comme illustré par la Figure 7.21.

QUALITATIVE BEHAVIOR 1

behavior		complete
EC / QV	Tcab '	
EC 3	= 0	complete_behavior 1.1.1
EC 7	> 0	
EC 16	= 0	
EC 21	< 0	

FIGURE 7.21 – Diminution de la variation de la température cabine

Pour comprendre l'obtention de ce comportement, nous pouvons raffiner par l'observation des variables $\dot{T}engine$ *et* $Airmixing$. Ces deux variables sont liées à $\dot{T}cab$ par les **relations 2** et **3**. Nous obtenons deux comportements raffinés comme le montre les Figures 7.22 et 7.23.

QUALITATIVE BEHAVIOR 1

behavior				complete
EC / QV	Tcab '	Tengine '	Airmixing	
EC 3	= 0	= 0	= A	complete_behavior 1.1.1
EC 5	= 0	> 0	= A	
EC 7	> 0	> 0	= A	
EC 11	> 0	> 0	< A	
EC 16	= 0	> 0	< A	
EC 21	< 0	> 0	< A	

FIGURE 7.22 – Raffinement de la diminution de la variation de la température cabine (1)

QUALITATIVE BEHAVIOR 2

behavior				complete
EC / QV	Tcab '	Tengine '	Airmixing	
EC 3	= 0	= 0	= A	complete_behavior 1.2.1
EC 5	= 0	> 0	= A	
EC 7	> 0	> 0	= A	
EC 16	= 0	> 0	< A	
EC 21	< 0	> 0	< A	

FIGURE 7.23 – Raffinement de la diminution de la variation de la température cabine (2)

Dans ces deux figures, nous voyons que la température du circuit moteur $Tengine$ commence à augmenter dans $(EC5)$. La température de cabine augmente juste aprés

dans ($EC7$) grâce à la **relation 2** de causalité proportionnelle. Quand la valeur de la vanne $Airmixing$ devient $< A$, la température de cabine suit cette variation dans le même sens. Par conséquent, $\dot{T}cab$ devient $= 0$ dans ($EC16$) puis < 0 dans ($EC21$).

7.4.3 Mode hybride

Lorsque la plateforme fonctionne en mode hybride, nous considérons toutes les relations :

— **relation 1** $= \dot{T}engine$ **thres** 0 **is CIPROP** $Thermostat$ **thres** T ;
— **relation 2** $= \dot{T}cab$ **thres** 0 **is CPROP** $\dot{T}engine$ **thres** 0 ;
— **relation 3** $= \dot{T}cab$ **thres** 0 **is CPROP** $Airmixing$ **thres** A ;
— **relation 4** $= \dot{T}cab$ **thres** 0 **is CIPROP** $Valve1$ **thres** $V1$;
— **relation 5** $= \dot{T}bat$ **thres** 0 **is CIPROP** $Valve2$ **thres** $V2$.

7.4.3.1 Scénario de la plateforme en mode hybride

Nous avons choisi un scénario pour la plateforme de gestion thermique en mode hybride qui peut être modélisé par une machine à états comme le montre la Figure 7.24 avec 4 états, 1 pseudo-état et 7 transitions :

— la transition $t0$ initialise la valeur de la température du circuit moteur $Tengine$ à $< Te$, celle du circuit batterie $Tbat$ à $> Tb$ et celle de la cabine $Tcab$ à $> Tc$;
— la transition $t1$ renforce le mode électrique en mettant la valeur de la température du circuit moteur $Tengine$ à $< Te$, celle du circuit batterie $Tbat$ à $> Tb$ et celle de la cabine $Tcab$ à $> Tc$;
— la transition $t2$ stabilise le mode électrique en mettant la valeur de la température du circuit engine $Tengine$ à $= Te$, celle du circuit batterie $Tbat$ à $= Tb$ et celle de la cabine $Tcab$ à $= Tc$;
— la transition $t3$ renforce la stabilité du mode électrique ;
— la transition $t4$ déclenche le mode thermique en mettant la valeur de la température du circuit moteur $Tengine$ à $> Te$, celle du circuit batterie $Tbat$ à $< Tb$ et celle de la cabine $Tcab$ à $> Tc$;
— la transition $t5$ renforce le mode thermique ;
— la transition $t6$ stabilise le mode thermique en mettant la valeur de la température du circuit moteur $Tengine$ à $= Te$, celle du circuit batterie $Tbat$ à $= Tb$ et celle de la cabine $Tcab$ à $= Tc$;
— la transition $t7$ renforce la stabilité du mode thermique.

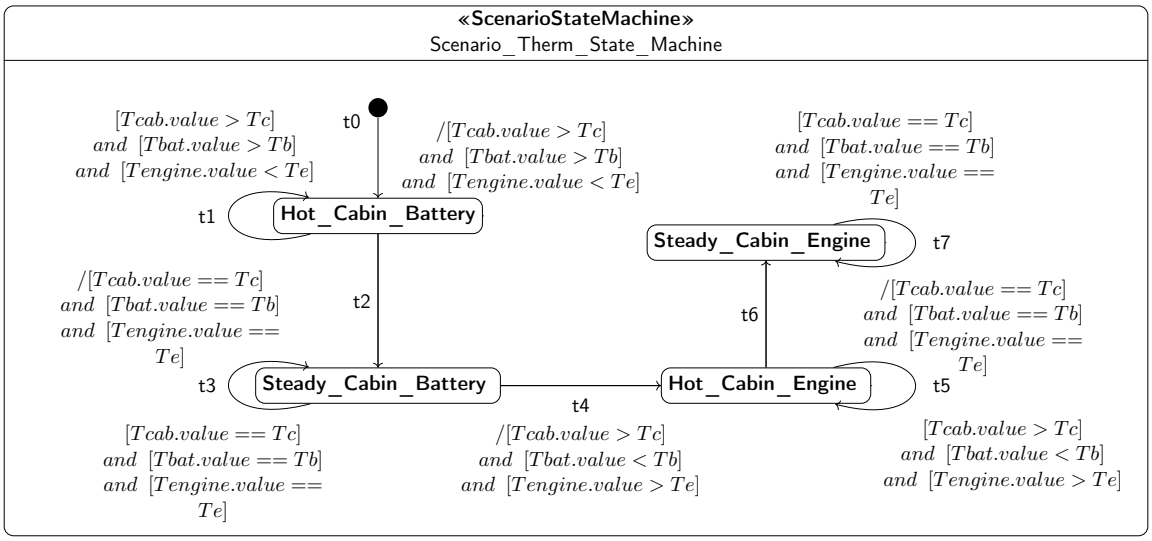

FIGURE 7.24 – Automate du scénario de simulation de la plateforme en mode hybride

7.4.3.2 Comportements de la plateforme en mode hybride

Le comportement continu de la plateforme de gestion thermique en mode hybride est modélisé par un diagramme paramétrique comme illustré par la Figure 7.25. Dans cette figure, nous trouvons les relations du mode électrique ainsi que celles du mode thermique.

7.4.3.3 Analyse des comportements de la plateforme en mode hybride

Nous analysons les traces brutes de simulation symboliques par le module d'analyse de notre chaîne d'outils. En n'observant que la variation de la température du circuit batterie $\dot{B}at$, nous obtenons plusieurs comportements abstraits. Parmi ceux-ci, nous obtenons un comportement dans lequel la variation de $Tbat$ reste constante aprés avoir diminué comme le montre la Figure 7.26.

Pour comprendre ce comportement. Nous regardons la variation de la température du circuit moteur $Tengine$ ainsi que la valeur de la vanne du $Thermostat$. Ce comportement stable de $\dot{T}bat$ est raffiné par plusieurs comportements comme illustré par les Figures 7.27, 7.28, 7.29 et 7.30.

Dans ces figures, nous remarquons que la température du circuit moteur $Tengine$ diminue quand la vanne $Thermostat$ devient $> T$. Ceci est modélisé par la **relation 1** avec l'opérateur de causalité inversement proportionnelle. Dans la Figure 7.29, la température $Tengine$ reste en phase de croissance et la vanne $Thermostat$ est $< T$ dans l'exécution de contexte ($EC1488$). Ceci correspond parfaitement à la régulation assurée par la **relation 1**. Dans la Figure 7.27, c'est l'inverse : la température $Tengine$

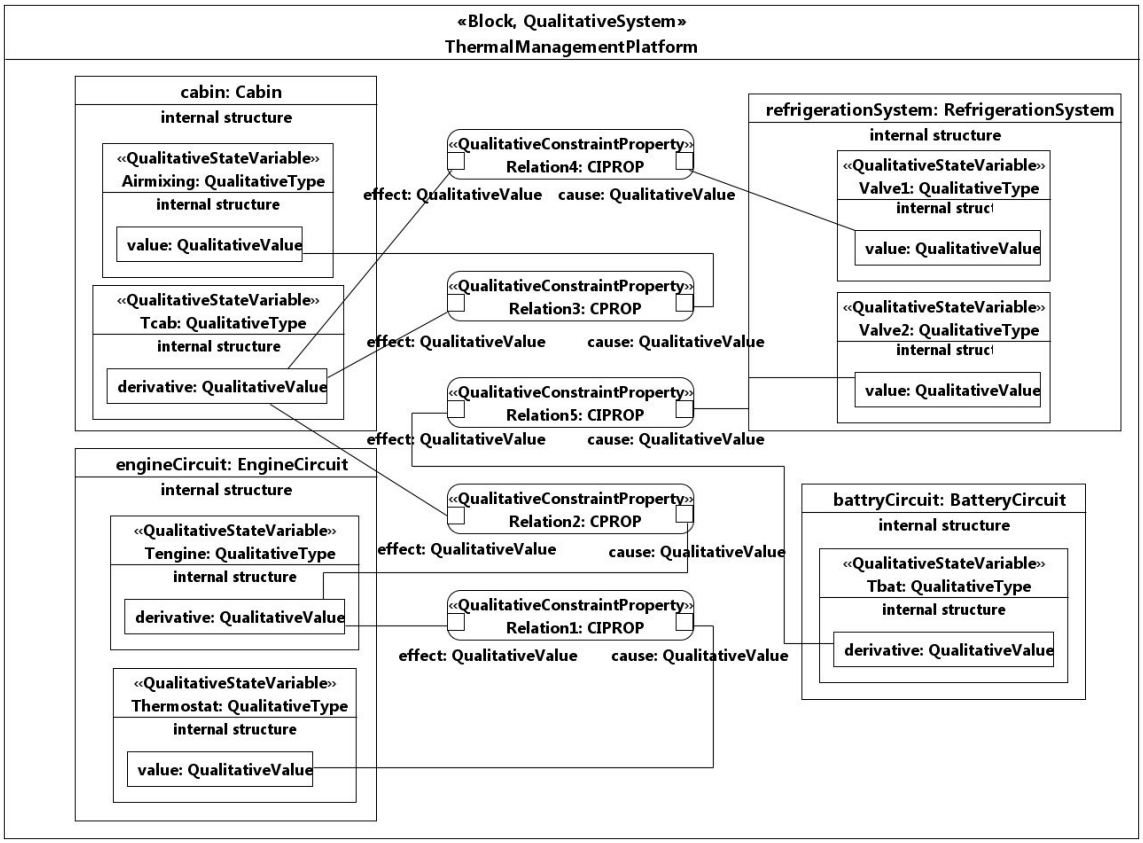

FIGURE 7.25 – Relations qualitatives de la plateforme de gestion thermique en mode hybride

QUALITATIVE BEHAVIOR 3

behavior		complete
EC / QV	Tbat '	
EC 3	= 0	complete_behavior 1.3.1 complete_behavior 1.3.2 complete_behavior 1.3.3 complete_behavior 1.3.4
EC 5	< 0	
EC 1488	= 0	

FIGURE 7.26 – Stabilisation de la variation de la température du circuit batterie $Tbat$

reste en phase de diminution et la vanne $Thermostat$ est $> T$ dans l'exécution de contexte ($EC2358$). Dans les Figures 7.28 et 7.30, nous avons une régulation complète de la variation de $Tengine$ dans le contexte d'exécution ($EC2480$). Dans ce cas, les températures des circuits batterie et moteur sont en phase d'équilibre.

QUALITATIVE BEHAVIOR 9

behavior				complete
EC / QV	Tbat '	Tengine '	Thermostat	
EC 3	= 0	= 0	= T	complete_behavior 1.9.1
EC 5	< 0	< 0	> T	
EC 2358	= 0	< 0	> T	

FIGURE 7.27 – Raffinement du comportement du circuit batterie $Tbat$ par $\dot{T}engine$ et $Thermostat$ (1)

QUALITATIVE BEHAVIOR 10

behavior				complete
EC / QV	Tbat '	Tengine '	Thermostat	
EC 3	= 0	= 0	= T	complete_behavior 1.10.1
EC 5	< 0	< 0	> T	
EC 35	< 0	= 0	= T	
EC 2480	= 0	= 0	= T	

FIGURE 7.28 – Raffinement du comportement du circuit batterie $Tbat$ par $\dot{T}engine$ et $Thermostat$ (2)

QUALITATIVE BEHAVIOR 43

behavior				complete
EC / QV	Tbat '	Tengine '	Thermostat	
EC 3	= 0	= 0	= T	complete_behavior 1.43.1
EC 5	< 0	< 0	> T	
EC 14	< 0	= 0	= T	
EC 1488	= 0	> 0	< T	

FIGURE 7.29 – Raffinement du comportement du circuit batterie $Tbat$ par $\dot{T}engine$ et $Thermostat$ (3)

QUALITATIVE BEHAVIOR 59

behavior				complete
EC / QV	Tbat '	Tengine '	Thermostat	
EC 3	= 0	= 0	= T	complete_behavior 1.59.1
EC 5	< 0	< 0	> T	
EC 39	< 0	< 0	= T	
EC 2480	= 0	= 0	= T	

FIGURE 7.30 – Raffinement du comportement du circuit batterie $Tbat$ par $\dot{T}engine$ et $Thermostat$ (4)

7.5 Conclusion

Dans ce chapitre, nous avons présenté le cas d'étude insdustriel *plateforme de gestion thermique d'un véhicule hybride* sur lequel nous avons appliqué notre méthodologie. Dans ce système, le concepteur ne dispose pas d'équations différentielles qui modélisent la dynamique du système. D'où l'intérêt de ce cas d'étude pour la validation de notre approche. Nous avons distingué trois modes de fonctionnement de ce système :

— mode électrique, dans lequel nous avons négligé la boucle de régulation du circuit moteur ;

— mode thermique, dans lequel nous avons négligé la boucle de régulation du circuit batterie ;

— mode hybride, dans lequel nous avons lié les trois boucles de régulation.

Nous avons analysé les comportements obtenus pour chaque mode. Nous avons montré que notre méthodologie permet d'aider le concepteur à analyser les comportements obtenus d'une manière incrémentale en observant à chaque fois les variables concernées du système, partant d'un comportement abstrait qui peut être raffiné. Cet

aspect incrémental de notre approche permet à l'utilisateur de comprendre les causes d'obtention de tels comportements qualitatifs. L'explication des comportements qualitatifs obtenus est absente dans des simulateurs qualitatifs comme QSIM [44]. Elle n'est pas explicite dans Garp3 [14]. Le diagramme de valeurs dans Garp3 permet de suivre l'évolution des variables dans les différents états du système. Cependant cette approche n'est pas incrémentale, les chemins du graphe d'états sont fixes et ne peuvent être raffinés.

Chapitre 8

Conclusion

8.1 Contribution

Définir une méthodologie pour l'analyse des systèmes hybrides dans les premières phases de conception est la principale contribution de notre thèse.

Nous avons mené notre thèse, dans toutes ses étapes, avec une progression sans cesse mise à épreuve à travers nos publications dans les actes de conférences et workshops internationaux, que nous avons jointes à la liste des références bibliographiques à la fin de ce manuscrit. Nous avons mis en place une méthodologie à base d'un langage dédié à la simulation qualitative sans équations différentielles fourni dans le périmètre de *SysML*.

Notre approche se base sur des techniques de l'Ingénierie dirigée par les modèles comme les transformations de modèle en modèle M2M et de modèle en texte M2T, dans le but de rendre notre méthode utilisée par les concepteurs de systèmes, les isoler des changements de notre modèle d'exécution symbolique et du format d'entrée de notre chaîne d'outils.

Durant notre thèse, nous avons mis en place un modèle d'exécution symbolique qui permet la liaison des différents niveaux de deux variables d'état. Ce modèle d'exécution fournit des résultats de simulation symboliques bruts. Un module dans notre chaîne d'outils analyse ces traces symboliques pour fournir les comportements qualitatifs physiques du système. Nous avons opté pour une approche incrémentale des comportements qualitatifs fournis à l'utilisateur pour l'aider à comprendre l'obtention de tels comportements du système.

La mise en place du modèle d'exécution symbolique est faite en 3 phases :

— dans la première phase, nous avons conçu un modèle d'exécution énuméré. Ce modèle nous a permis de nous familiariser avec les automates hybrides. Il s'inspire de l'algèbre de signes *+, -, 0* et met en évidence les limites de cet algèbre : l'indéterminisme et le problème de l'explosion combinatoire. Cette première version de modèle énuméré observe les différentes valeurs de la dérivée seconde, dérivée première et la valeur pour déterminer les différents états qualitatifs de la variable comme *Increase, Decrease...* ;

— dans la deuxième phase, nous avons construit le deuxième modèle d'exécution symbolique. Ce modèle s'inspire de la méthode d'Euler pour le calcul de la dérivée première en fonction de la dérivée seconde ou pour le calcul de la valeur en fonction de la dérivée première en prenant un pas d'intégration unitaire. Ce modèle compare symboliquement les valeurs de la dérivée seconde, dérivée première et la valeur de la variable d'état pour connaître les transitions possibles.

Dans ce modèle d'exécution symbolique, nous avons ajouté des états qualitatifs qui modélisent un ordre supérieur de variation qualitative en s'inspirant de [47] et [23] comme *Increasingly_Increase, Decreasingly_Increase...*. L'application de ce modèle d'exécution sur le circuit Rc nous a fait remarquer que nous manquions d'éléments de langage pour modéliser la liaison entre deux variables d'état ;

— dans la troisième phase, nous avons construit le troisième modèle d'exécution à base de contraintes qualitatives. Dans cette version de modèle, nous avons défini des opérateurs, inspirés de Garp3 [14], qui permettent de lier à différents niveaux des variables d'états.

La liaison que nous avons faite entre le langage SysML et la simulation qualitative, met en valeur la facilité d'utiliser notre méthodologie par les concepteurs de système dans les premières phases de conception. Cette liaison est faite par le profil que nous avons conçu pour la simulation qualitative sans équations différentielles. SysML, le standard de modélisation des spécifications multi-domaines, est fait pour analyser la structure et les fonctionnalités des systèmes avant leur réalisation, c'est-à-dire dans les premières phases de conception. C'est à ce stade que les ingénieurs ne disposent pas de toute l'information numérique sur leur système. La simulation qualitative fournit des réponses sur les comportements des systèmes en l'absence de paramètres numériques.
Notre méthodologie a été appliquée sur un cas d'étude industriel : *Plateforme de gestion thermique d'un véhicule hybride*. Nous avons montré que nous pouvions générer les comportements qualitatifs de ce système en absence d'informations numériques. Ces comportements qualitatifs pourraient aider le concepteur du système à mieux spécifier son système.

8.2 Limites et perspectives

Il y a nécessairement des limitations à l'analyse des systèmes hybrides par la simulation qualitative sans équations différentielles au sens du modèle mathématique continu. Les comportements qui dépendent des valeurs numériques précises ne peuvent pas être fournis à l'utilisateur. Les systèmes traités par notre méthodologie sont des systèmes linéaires. Les opérateurs que nous avons définis dans notre approche, sont linéaires. En effet, *proportionnel, inversement proportionnel, causalement proportionnel, causalement inversement proportionnel* ne peuvent modéliser que deux variables d'état qui sont :

— en phase ou en opposition de phase en suivant un seuil ;

— proportionnelles ou inversement proportionnelles suivant un seuil.

Des systèmes non linéaires comme le brusselator [1] ne peuvent pas être modélisés par notre approche. Dans le futur, il sera nécessaire d'enrichir notre langage par des opérateurs qui permettront de modéliser la dynamique non linéaire des systèmes hybrides.

Dans les sytèmes hybrides, il y a souvent des variables qui dominent d'autres. Cette *dominance* peut être globale, dans tout le processus de simulation, ou partielle dans certaines conditions. Il serait intéressant d'enrichir notre langage qualitatif par cette notion et voir son influence sur les comportements qualitatifs générés.

Une co-simulation entre le modèle qualitatif du système et l'environnement dans lequel le système interagit est envisageable pour améliorer la qualité de l'analyse de notre méthodologie pour déterminer des scénarios dépendant des paramètres numériques. La co-simulation nécessite des simulateurs numériques qui demandent un temps de calcul important pour simuler un scénario spécifique. Ceci réduit la portée d'exploration. En contrepartie, la simulation qualitative fournit une abstraction de tous les comportements du système. Ainsi, la simulation qualitative, avec l'intégration d'informations numériques [46] [9], pourrait produire une exploration riche des comportements du système dans un temps de calcul raisonnable.

Annexe A

Exécution symbolique de la balle rebondissante par Diversity

Nous présentons l'arbre d'exécution symbolique généré par Diversity pour la balle rebondissante illustré par la Figure A.1. Dans cette Figure, la couleur vert clair modélise un contexte d'exécution vers lequel on reboucle et la couleur vert foncé modélise la détection d'une redondance. Cet arbre est obtenu par le modèle d'exécution symbolique inspiré de la méthode d'Euler, expliqué dans la section 4.3 page 45.

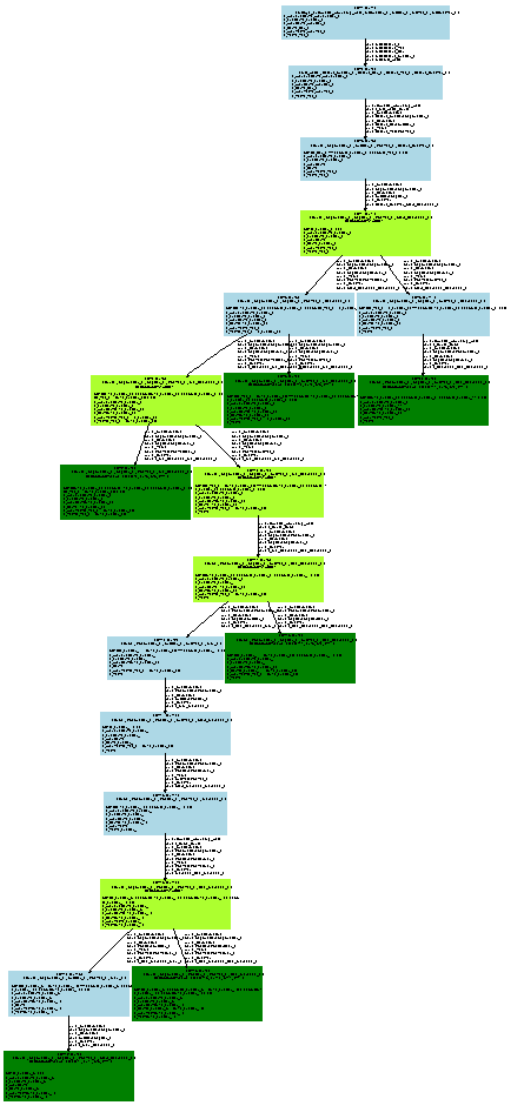

FIGURE A.1 – Arbre de comportements symboliques généré par Diversity

Cet arbre d'exécution symbolique est composé de 21 états, sa hauteur maximale est 12, sa largeur maximale est 4, le nombre d'états redondants est 6 comme illustré par la Figure A.2.

```
REPORT
 STOP CRITERIA  PROCESSOR
  The CONTEXT  count : 21
  The RUN EVAL count : 14
  The RUN STEP count : 15

  The Max HEIGHT reaching : 12
  The Max WIDTH  reaching : 4
 REDUNDANCY PROCESSOR
  Comparer predicate: NONE
  The redundancy count: 6 for 6 tests !
 BASIC TRACE GENERATOR
  The TRACE count : 6
  DONE !
```

FIGURE A.2 – Rapport de simulation du modèle de la balle rebondissante

Nous prenons un chemin de l'arbre de la Figure A.1 qui modélise le comportement de la balle rebondissante présenté dans la Figure 4.11 du chapitre 4. Ce chemin est illustré par les Figures A.3, A.4 et A.5. Dans ces Figures, nous voyons clairement les différents contextes d'exécution constitués des :

— noms des états ;

— conditions de noeuds (NC) permettant le passage d'un contexte d'exécution à un autre ;

— différentes affectations symboliques de la variable d'état Z suivant le modèle d'intégration symbolique inspiré de la méthode d'Euler.

Les cinq automates qui permettent l'obtention de l'arbre de la Figure A.1 sont implémentés dans Diversity. La Figure A.6 présente l'automate du système. La Figure A.7 modélise l'automate de la variation qualitative de la dérivée seconde. La Figure A.8 présente l'automate de l'évolution qualitative avec intégration symbolique de la dérivée première $\dot{Z}$. La Figure A.9 modélise l'automate de l'évolution qualitative avec intégration symbolique de la valeur Z. La Figure A.10 présente l'automate qui calcule les différents comportements qualitatifs de la balle rebondissante.

FIGURE A.3 – Chemin symbolique de l'arbre généré par Diversity (1)

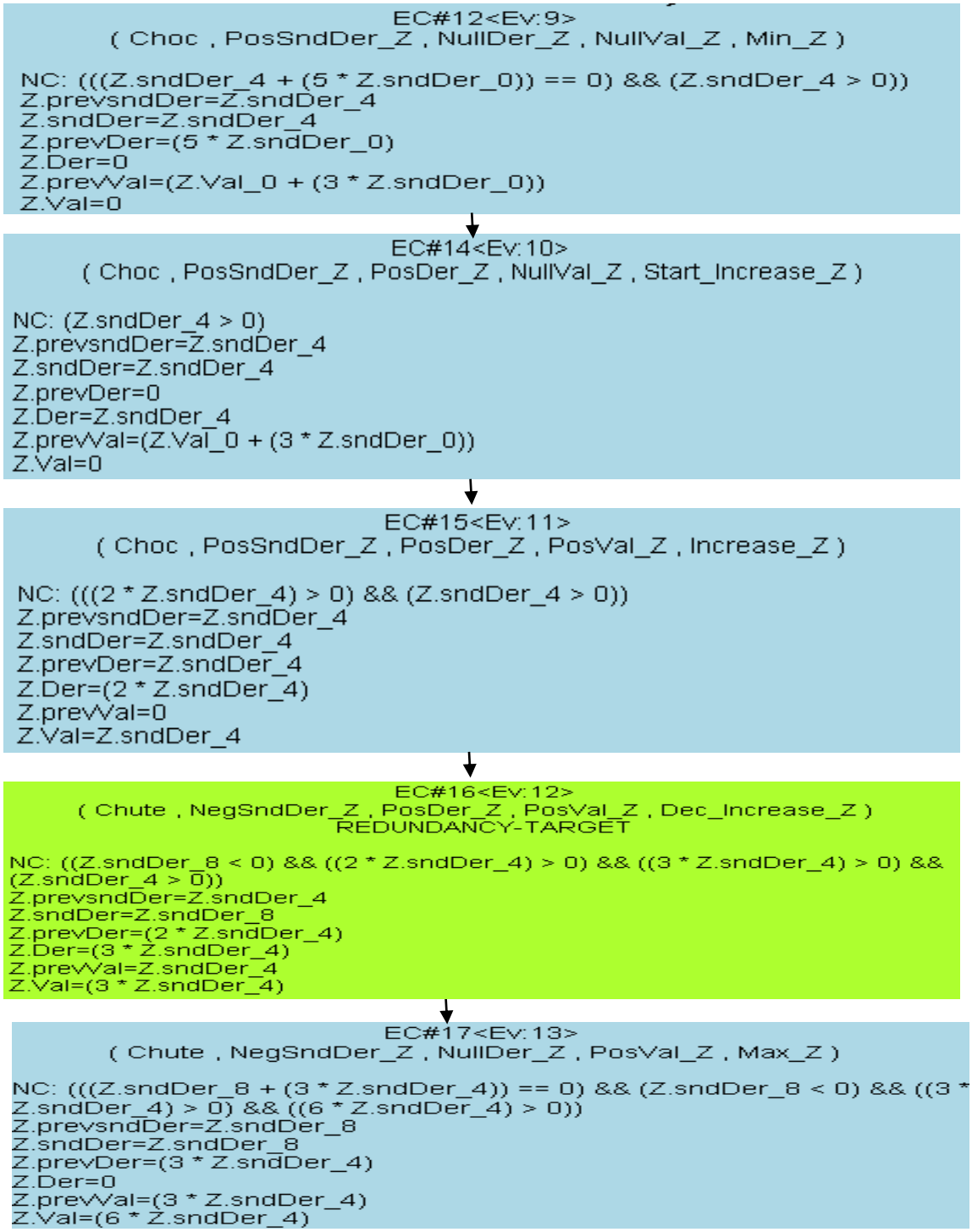

FIGURE A.4 – Chemin symbolique de l'arbre généré par Diversity (2)

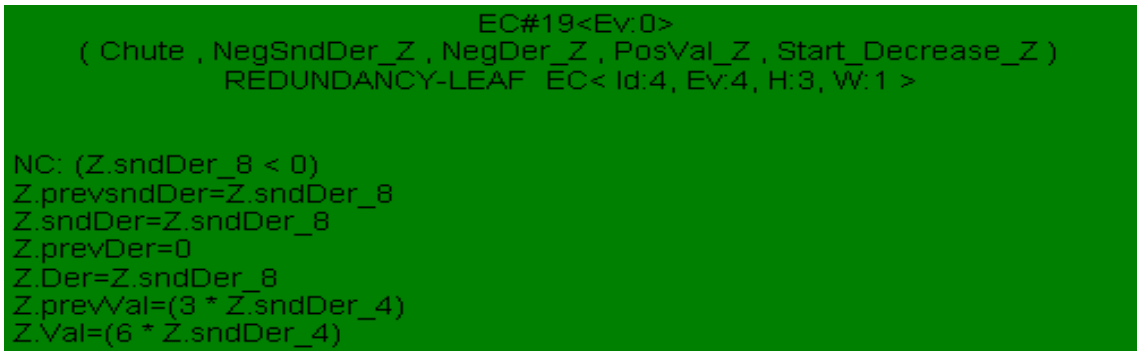

FIGURE A.5 – Chemin symbolique de l'arbre généré par Diversity (3)

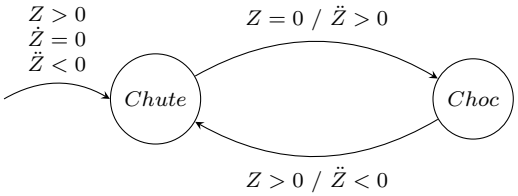

FIGURE A.6 – Machine à états de la balle rebondissante

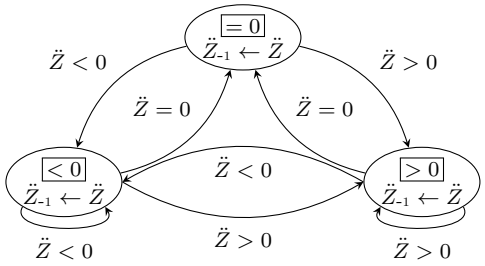

FIGURE A.7 – Évolution qualitative de la dérivée seconde $\ddot{Z}$

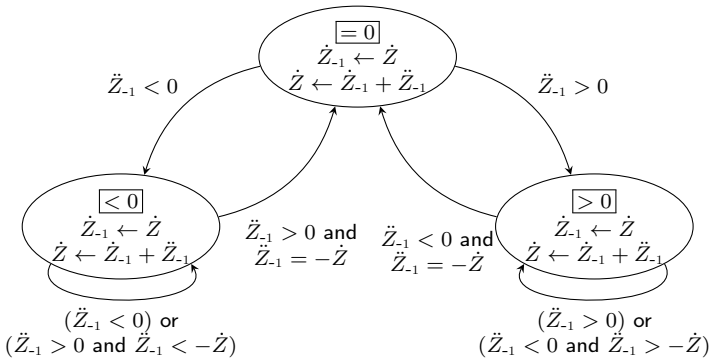

FIGURE A.8 – Évolution qualitative avec intégration symbolique de la dérivée première $\dot{Z}$

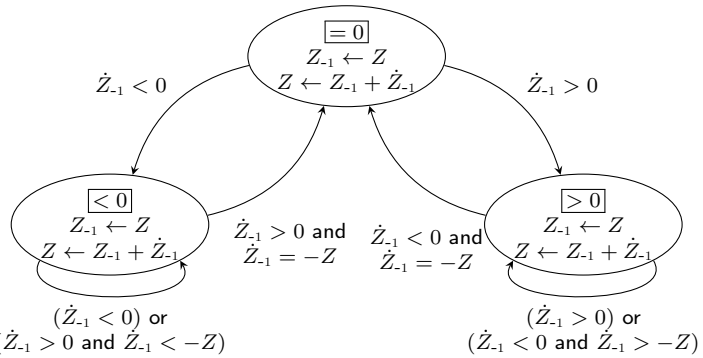

FIGURE A.9 – Évolution qualitative avec intégration symbolique de la valeur Z

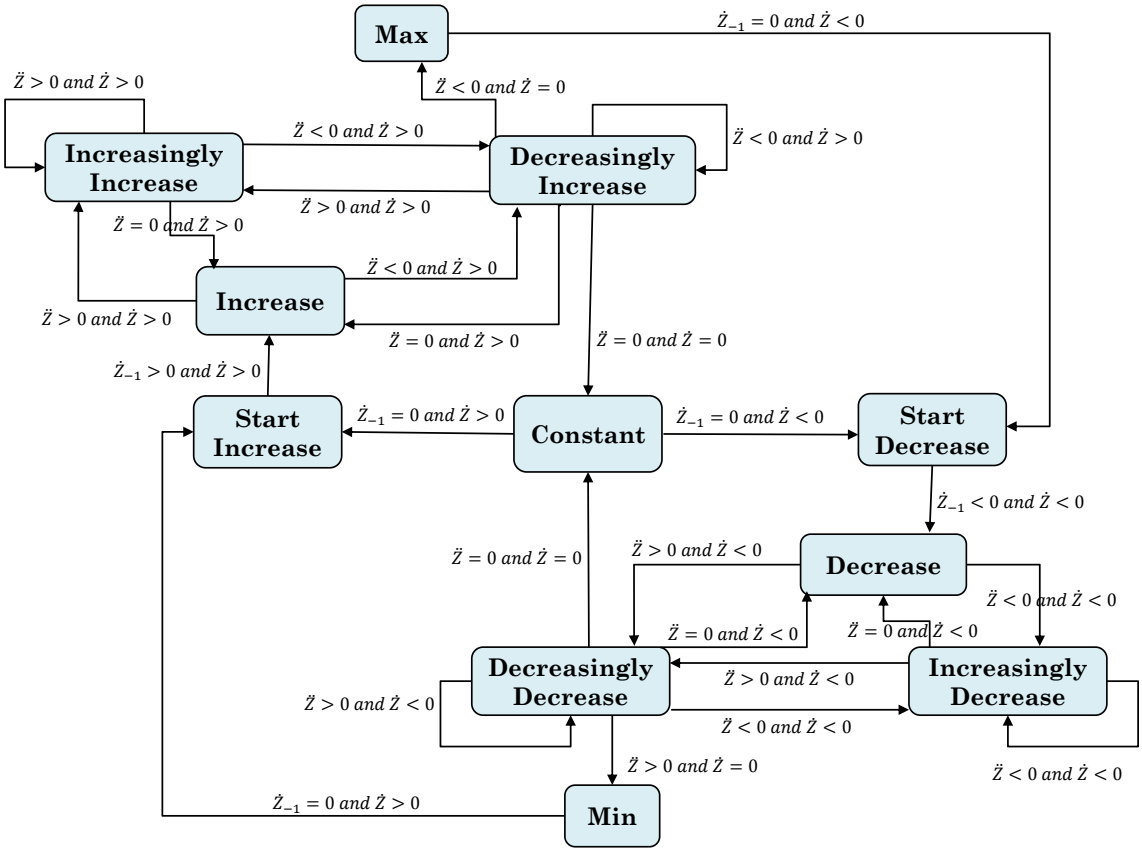

FIGURE A.10 – Machine à états des comportements qualitatifs de la balle rebondissante

Bibliographie

[1] Brusselator. https://www-dimat.unipv.it/boffi/teaching/download/Brusselator.pdf.

[2] La modélisation sysml. http://www.uml-sysml.org/sysml/la-modelisation-sysml.

[3] Langage sysml. http://projet.eu.org/pedago/sin/term/3-SysML.pdf.

[4] Model driven architecture. https://www.omg.org/mda/specs.htm.

[5] Sherpa engineering. https://www.sherpa-eng.com.

[6] Rajeev Alur, Thao Dang, Joel Esposito, Yerang Hur, Franjo Ivancic, Vijay Kumar, P Mishra, GJ Pappas, and Oleg Sokolsky. Hierarchical modeling and analysis of embedded systems. *Proceedings of the IEEE*, 91(1) :11–28, 2003.

[7] Johan Bengtsson, WO David Griffioen, Kåre J Kristoffersen, Kim G Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Automated verification of an audio-control protocol using uppaal. *The Journal of Logic and Algebraic Programming*, 52 :163–181, 2002.

[8] Kirsten Berkenkötter, Stefan Bisanz, Ulrich Hannemann, and Jan Peleska. The hybriduml profile for uml 2.0. *International Journal on Software Tools for Technology Transfer*, 8(2) :167–176, 2006.

[9] Daniel Berleant and Benjamin Kuipers. Qualitative-numeric simulation with q3. *Recent advances in qualitative physics*, 98 :285–313, 1992.

[10] Jean Bézivin and Jean-Pierre Briot. Sur les principes de base de l'ingénierie des modèles. *L'OBJET*, 10(4) :145–157, 2004.

[11] Wolfgang Borutzky. *Bond graph modelling of engineering systems*. Springer, 2011.

[12] Patricia Bouyer, Maximilien Colange, and Nicolas Markey. Symbolic optimal reachability in weighted timed automata. In Swarat Chaudhuri and Azadeh Farzan, editors, *Computer Aided Verification : 28th International Conference,*

CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I, pages 513–530. Springer International Publishing, 2016.

[13] Marius Bozga, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. Kronos : A model-checking tool for real-time systems. In Anders P. Ravn and Hans Rischel, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems : 5th International Symposium, FTRTFT'98 Lyngby, Denmark, September 14–18, 1998 Proceedings*, pages 298–302, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[14] Bert Bredeweg, Floris Linnebank, Anders Bouwer, and Jochem Liem. Garp3 — workbench for qualitative modelling and simulation. *Ecological informatics*, 4(5) :263–281, 2009.

[15] Christopher W Brown. Qepcad b : a program for computing with semi-algebraic sets using cads. *ACM SIGSAM Bulletin*, 37(4) :97–108, 2003.

[16] Yue Cao, Yusheng Liu, and Christiaan JJ Paredis. System-level model integration of design and simulation for mechatronic systems based on sysml. *Mechatronics*, 21(6) :1063–1075, 2011.

[17] Eric Cariou. Ingénierie des modèles introduction générale. `http://ecariou.perso.univ-pau.fr/cours/idm-old/cours-intro.pdf`.

[18] Alongkrit Chutinan and Bruce H. Krogh. Computational techniques for hybrid system verification. *IEEE Trans. Automat. Contr.*, 48 :64–75, 2003.

[19] Lori A. Clarke. A system to generate test data and symbolically execute programs. *IEEE Trans. on software engineering*, (3) :215–222, 1976.

[20] Philippe Dague. Numeric reasoning with relative orders of magnitude. In *AAAI*, pages 541–547, 1993.

[21] Philippe Dague and Louise Travé-Massuyès. *Modèles et raisonnements qualitatifs*. hermes Science, 2003.

[22] Johan De Kleer. Multiple representations of knowledge in a mechanics problem-solver. In *Readings in qualitative reasoning about physical systems*, pages 40–45. Elsevier, 1990.

[23] Johan De Kleer and John Seely Brown. A qualitative physics based on confluences. *Artificial intelligence*, 24(1-3) :7–83, 1984.

[24] Johan De Kleer and John Seely Brown. Theories of causal ordering. In *Readings in qualitative reasoning about physical systems*, pages 646–660. Elsevier, 1990.

[25] Samba Diaw, Rédouane Lbath, and Bernard Coulette. Etat de l'art sur le développement logiciel dirigé par les modèles. *Technique et Science Informatique TSI*, 29(505-536) :71, 2008.

[26] Eclipse Modeling Project. xtext. `https://www.eclipse.org/Xtext/documentation/308_emf_integration.html`.

[27] Eclipse Modeling Project. Acceleo. `https://projects.eclipse.org/projects/modeling.m2t.acceleo`, 2017.

[28] Eclipse Modeling Project. Qvt. `https://projects.eclipse.org/projects/modeling.mmt.qvt-oml`, 2017.

[29] Johan Eker, Jorn W Janneck, Edward A. Lee, Jie Liu, Xiaojun Liu, Jozsef Ludvig, Sonia Sachs, Yuhong Xiong, and Stephen Neuendorffer. Taming heterogeneity - the ptolemy approach. *Proceedings of the IEEE*, 91(1) :127–144, 2003.

[30] Brian A Kyckelhahn Kenneth D Forbus. Jitter in self-explanatory simulation. `http://www.qrg.northwestern.edu/papers/Files/QR04_Jitter_final.pdf`.

[31] Kenneth D Forbus. Qualitative process theory. *Artificial intelligence*, 24(1-3) :85–168, 1984.

[32] Pierre Fouché and Benjamin J Kuipers. Reasoning about energy in qualitative simulation. *IEEE Trans. on Systems, Man, and Cybernetics*, 22(1) :47–63, 1992.

[33] Peter Fritzson. *Principles of object-oriented modeling and simulation with Modelica 2.1.* John Wiley & Sons, 2010.

[34] Peter Fritzson, David Broman, François Cellier, and Christoph Nytsch-Geusen. Equation-based object-oriented languages and tools report on the workshop eoolt 2007 at ecoop 2007. In *European Conference on Object-Oriented Programming*, pages 27–39. Springer, 2007.

[35] Jean-Pierre Gallois and Jean-Yves Pierron. Qualitative simulation and validation of complex hybrid systems. In *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, 2016.

[36] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. Hytech : A model checker for hybrid systems. In Orna Grumberg, editor, *Computer Aided Verification : 9th International Conference, CAV'97 Haifa, Israel, June 22–25, 1997 Proceedings*, pages 460–463, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[37] Robert M Hierons. Testing from a z specification. *Software Testing, Verification and Reliability*, 7(1) :19–33, 1997.

[38] Yumi Iwasaki and Herbert A Simon. Causality in device behavior. In *Readings in qualitative reasoning about physical systems*, pages 631–645. Elsevier, 1990.

[39] Yumi Iwasaki and Herbert A Simon. Causality and model abstraction. *Artificial intelligence*, 67(1) :143–194, 1994.

[40] Thomas A Johnson, Jonathan M Jobe, Christiaan JJ Paredis, and Roger Burkhart. Modeling continuous system dynamics in sysml. In *ASME 2007 International Mechanical Engineering Congress and Exposition*, pages 197–205. American Society of Mechanical Engineers, 2007.

[41] Herbert Kay. *Refining imprecise models and their behaviors*. PhD thesis, Citeseer, 1996.

[42] Herbert Kay and Benjamin Kuipers. Numerical behavior envelopes for qualitative models. In *AAAI*, pages 606–613, 1993.

[43] James C King. Symbolic execution and program testing. *Com. of the ACM*, 19(7) :385–394, 1976.

[44] Benjamin Kuipers. Qualitative simulation. *Artificial intelligence*, 29(3) :289–338, 1986.

[45] Benjamin Kuipers. *Qualitative reasoning : modeling and simulation with incomplete knowledge*. MIT press, 1994.

[46] Benjamin Kuipers and Daniel Berleant. Using incomplete quantitative knowledge in qualitative reasoning. In *AAAI*, volume 88, pages 324–329. Saint-Paul, MN, 1988.

[47] Benjamin Kuipers and Charles Chiu. Taming intractible branching in qualitative simulation. *Readings in qualitative reasoning about physical systems*, 1987.

[48] Pierre Laforcade, Vincent Barré, and Boubekeur Zendagui. Scénarisation pédagogique et ingénierie dirigé par les modèles, 2007.

[49] Arnault Lapitre. *Procédures de réduction pour les systèmes à base d'automates communicants : formalisation et mise en oeuvre*. PhD thesis, Paris 11, Orsay, 2002.

[50] Mohamed El Habib Laraba. *Sur le raisonnement qualitatif ; une approche à base d'agents dialogiques pour l'explication de la simulation qualitative*. PhD thesis, Université Mentouri de Constantine, 2007.

[51] Huimin Lin. Symbolic transition graph with assignment. *CONCUR'96 : Concurrency Theory*, pages 50–65, 1996.

[52] J Lunze and J Schröder. Application of qualitative observation and prediction to a neutralisation process. *IFAC Proceedings Volumes*, 32(2) :4277–4282, 1999.

[53] Jan Lunze. Qualitative modelling of linear dynamical systems with quantized state measurements. *automatica*, 30(3) :417–431, 1994.

[54] Nancy Lynch, Roberto Segala, and Frits Vaandrager. Hybrid i/o automata. *Information and computation*, 185(1) :105–157, 2003.

[55] Michael L Mavrovouniotis and George Stephanopoulos. Reasoning with orders of magnitude and approximate relations. In *AAAI*, pages 626–630, 1987.

[56] Michael L Mavrovouniotis and George Stephanopoulos. Formal order-of-magnitude reasoning in process engineering. *Computers & Chemical Engineering*, 12(9-10) :867–880, 1988.

[57] Slim Medimegh, Jean-Yves Pierron, and Frédéric Boulanger. A new qualitative language for qualitative simulation. In *2nd International Symposium on Computer Science and Intelligent Control*, number 1731, 2018.

[58] Slim Medimegh, Jean-Yves Pierron, and Frédéric Boulanger. Qualitative simulation of hybrid systems with an application to sysml models. In *6th International Conference on Model-Driven Engineering and Software Development*. SCITEPRESS-Science and Technology Publications, 2018.

[59] Slim Medimegh, Jean-Yves Pierron, Jean-Pierre Gallois, and Frédéric Boulanger. A new approach of qualitative simulation for the validation of hybrid systems. In *GEMOC International Workshop on The Globalization of Modeling Languages at MODELS 2016*, number 1731, 2016.

[60] Antoine Missier and L Travé-Massuyès. Temporal information in qualitative simulation. In *AI, Simulation and Planning in High Autonomy Systems, 1991. Integrating Qualitative and Quantitative System Knowledge, Proceedings of the Second Annual Conference on*, pages 298–305. IEEE, 1991.

[61] N Piera and L Travé. About qualitative equality : Axioms and properties. In *9th International Workshop on Expert System and their Applications*. Avignon, 1989.

[62] Adrian Pop, David Akhvlediani, and Peter Fritzson. Towards unified system modeling with the modelicaml uml profile. In *Proceedings of the 1st International Workshop on Equation-Based Object-Oriented Languages and Tools : Berlin ; Germany ; July 30 ; 2007 ; conjunction with ECOOP*, number 024. Linköping University Electronic Press, 2007.

[63] Olivier Raiman. Order of magnitude reasoning. In *Readings in qualitative reasoning about physical systems*, pages 318–322. Elsevier, 1990.

[64] Qiang Shen and Roy Leitch. Integrating common-sense and qualitative simulation by the use of fuzzy sets. In *Proceedings of the Fourth International Workshop on Qualitative Physics*, pages 220–232. Citeseer, 1990.

[65] Louise Trave-Massuyes and Nuria Piera. The orders of magnitude models as qualitative algebras. In *IJCAI*, pages 1261–1266. Citeseer, 1989.

[66] Skander Turki and Thierry Soriano. A sysml extension for bond graphs support. In *Proc. of the International Conference on Technology and Automation (ICTA), Greece*. Citeseer, 2005.

[67] Yves Vanderperren and Wim Dehaene. From uml/sysml to matlab/simulink : current state and future perspectives. In *Proceedings of the conference on Design, automation and test in Europe : Proceedings*, pages 93–93. European Design and Automation Association, 2006.

[68] Mark Eric Wiegand. *Constructive qualitative simulation of continuous dynamic systems*. PhD thesis, Heriot-Watt University, 1991.

université PARIS-SACLAY | ÉCOLE DOCTORALE Sciences et technologies de l'information et de la communication (STIC)

Titre : Analyse formelle de spécifications hybrides à partir de modèles SysML pour la validation fonctionnelle des systèmes embarqués

Mots clés: systèmes hybrides, simulation qualitative, exécution symbolique, transformation de modèle, SysML, comportement qualitatif.

Résumé: Le logiciel embarqué est devenu aujourd'hui incontournable dans la plupart des secteurs industriels. Ce dernier fait appel en général à des connaissances métier différentes. L'ensemble du système (le logiciel et son environnement) est ainsi spécifié d'une manière hétérogène, avec des parties discrètes et d'autres continues. La simulation de ces systèmes hybrides nécessite des données précises et une synchronisation des changements continus avec les transitions discrètes. Mais, dans les premières phases de conception, l'absence des informations empêche de simuler le système numériquement. Dans notre thèse, nous présentons un nouveau langage qualitatif dédié à la simulation qualitative des systèmes hybrides. Ce nouveau langage consiste à modéliser les relations entre les variables du système. Il est implémenté dans Diversity, un moteur d'exécution symbolique, pour construire les traces du système. Nous avons appliqué cette approche à l'analyse des modèles SysML, en utilisant une transformation M2M à partir de SysML vers un langage pivot, une transformation M2T à partir de ce langage vers Diversity. Nous avons aussi analysé les traces brutes de l'exécution symbolique de Diversity pour construire les comportements qualitatifs du système.

Title: Formal analysis of hybrid specifications from SysML models for functional validation of embedded systems specifications

Keywords: hybrid systems, qualitative simulation, symbolic execution, model transformation, SysML, qualitative behavior.

Abstract: Embedded software has become essential in most industrial sectors. The latter usually involves various business knowledge. The whole system (the software and its environment) is specified in a heterogeneous form, with discrete and continuous parts. Simulating these hybrid systems requires precise data and synchronization of continuous changes and discrete transitions. However, in the first design steps, missing information forbids numerical simulation. We present in our thesis a new qualitative language for qualitative simulation of hybrid systems, which consists in computing the relationships between the system variables. This language is implemented in the Diversity symbolic execution engine to build the traces of the system. We apply this approach to the analysis of SysML models, using an M2M transformation from SysML to a pivot language, an M2T transformation from this language to Diversity. We also analyze the brutal symbolic traces obtained by Diversity to build the real qualitative behaviors of the system.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France