Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

وزارة التعليم العالي والبحث العلمي

BADJI MOKHTAR – ANNABA UNIVERSITY
UNIVERSITE BADJI MOKHTAR – ANNABA
Faculté des Sciences de l'Ingéniorat
Département d'Informatique

جامعة باجي مختار –عنابة
كليـة علـوم الهنـدسة
قسـم الإعلام الآلــي

Année : 2019/2020

THÈSE

présentée pour obtenir le grade

universitaire de Docteur

Spécialité : Informatique Embarquée

Formalisation des spécifications UML MARTE par les réseaux de Petri temporellement temporisés en vue de la vérification formelle des systèmes temps-réel embarqués

---

Par : Nadia CHABBAT

DEVANT LE JURY

Nom et prénom	Qualité	Grade	Etablissement
Mohamed Tahar KIMOUR	Président	Professeur	U. Annaba
Salim Ghanemi	Rapporteur	Professeur	U. Annaba
Abdelkarim AMIRAT	Examinateur	Professeur	U. Souk Ahras
Allaoua CHAOUI	Examinateur	Professeur	U. Constantine.2
Toufik Messaoud MAAROUK	Examinateur	M.C.A	U. Khenchla
Nora BOUNOUR	Examinateur	M.C.A	U. Annaba
Djamel Eddine SAIDOUNI	Invité	Professeur	U. Constantine.2

Dédicace

En témoignage de ma gratitude et de mon grand amour, je dédie ce modeste travail

A ma chère mère Fatma, source d'amour, d'affection et de générosité, symbole de tendresse et d'amour, qui m'a appris beaucoup de choses dans la vie et qui n'a jamais arrêté de m'aider, de m'encourager et de me soutenir durant les heures les plus critiques de ma vie, cette réussite scolaire est pour elle,

A mon cher père Said, qui m'a toujours aidé et qui ne m'a jamais privé de conseils de ses orientations de poursuivre mes études,

A mes très sœurs et frères, en particulier ma sœur Sabah, a qui je souhaite beaucoup de succès dans leur vie,

A mon marie Redouan et sa famille,

A mes chères collègues et amies qui m'ont aidé et encouragé.

Lydia . C

(18 février 2021)

Remerciements

En premier, je remercie "الله " de m'avoir donné la force et la persévérance pour faire aboutir ce travail « الحمد لله».

Je tiens à remercier Monsieur Tahar Mohamed Kimour, Professeur à l'Université Badji Mokhtar –Annaba, pour m'avoir fait l'honneur de présider de la soutenance de ma thèse de Doctorat en Sciences.

Toute ma gratitude va à mes encadreurs Monsieur Salim Ghanemi, Professeur à l'Université Badji Mokhtar –Annaba et Monsieur Djamel Eddine Saidouni, Professeur à l'Université Abdelhamid Mehri – Constantine 2. Monsieur Salim Ghanemi qui m'a encadré tout au long de cette thèse. Ses conseils et ses encouragements ont permis à ce travail d'aboutir. Monsieur Djamel Eddine Saidouni qui a été toujours disponible pour répondre aux questions que je lui posais, que ce soit théorique, techniques ou LATEXniques. Je le remercie aussi pour m'avoir donné toute ces idées, ainsi que pour toutes les riches séances de travail, sans quoi je ne serai jamais arrivé à bout de cette thèse. Je ne vous remercierai jamais assez pour m'avoir fait profiter de votre expérience et de m'avoir témoigné tant de bienveillance.

Je tiens à remercier tous les membres du jury, Abdelkarim AMIRAT, Professeur à l'Université Mohamed-Chérif Messaadia –Souk ahras, Allaoua Chaoui, Professeur à l'Université Abdelhamid Mehri –Constantine 2, Toufik Messaoud Maarouk, Maitre de conférences à l'Université de khenchla et Nora Bounour, Maitre de conférences à l'Université Badji Mokhtar –Annaba, pour le temps qu'ils ont accordé pour examiner ce modeste travail.

J'adresse également mes sincères remerciements à ma famille, parents, frères et sœurs, particulièrement ma sœur Sabah, de m'avoir aidé à surmonter tous les obstacles et à me forger à travers les difficultés vécues durant toute cette période de travail.

Je dois aussi exprimer toute ma gratitude et ma reconnaissance à Madame Radja Boukharou, son marié Monsieur Ahmed Chawki chaouche, ainsi que Madame Souad Ghellati, Enseignants à l'Université Abdelhamid Mehri – Constantine 2, qui m'ont aidé et soutenu dans la réalisation de cette thèse.

Résumé

Les systèmes temps-réel embarqués (STRE) sont souvent complexes et critiques, et nécessitent un développement rigoureux pour affirmer leur correction fonctionnelle sous des contraintes temporelles. Le profil UML MARTE (Modeling and Analysis of Real Time Embedded systems) est imposé comme un standard spécifié par l'OMG pour la modélisation et l'analyse des systèmes temps-réel embarqués. Il offre un Framework général de modélisation pour concevoir et analyser les STRE. Cependant, les spécifications en modèles UML MARTE manquent de fondements formels requis pour accomplir une vérification complète de ces modèles. De ce fait, les modèles UML MARTE ne peuvent pas être exploités à des fins de vérification et validation des systèmes. Une combinaison des modèles UML MARTE et des méthodes formelles est une alternative pour surmonter un tel problème.

Dans ce contexte, la vérification formelle basée modèle (*model-checking*) constitue un outil important du fait qu'elle est dotée de procédure permettant la vérification automatique et exhaustive des propriétés attendues du système en étude. Malgré son importance, la vérification basée modèle se heurte au problème de l'explosion combinatoire de l'espace d'états, en particulier si le nombre des horloges est important.

Le travail de cette thèse s'inscrit dans le cade de la spécification et la vérification formelle des systèmes temps-réel embarqués. Il consiste à tirer profit des avantages offerts par les modèles formels temporellement temporisés et basés sur les sémantiques du vrai parallélisme. Ainsi, nous présentons dans un premier temps, une approche orientée modèle pour spécifier et vérifier formellement les systèmes temps-réel embarqués décrits par les modèles UML MARTE. Par ailleurs, nous formalisons la sémantique des diagrammes de séquence UML2 annotés par des contraintes temporelles en utilisant le profil MARTE. Ainsi, nous proposons une méthode opérationnelle de translation de ces diagrammes vers les réseaux de Petri temporellement temporisés (DTPNs). Les règles graphiques et formelles de la translation (MARTESDtoDTPN) sont développées et appliquées sur des exemples illustratifs. La vérification formelle exploite les structures sémantiques associées aux spécifications décrites en DTPN, à savoir les automates temporisés avec durées actions (daTAs). En effet, l'espace d'états d'un daTA est exprimé par un graphe de régions ou par un graphe de zones basé sur la maximalité, et la vérification peut se faire par des outils model-checker tels que UPPAAL ou KRONOS. Cette contribution est validée sur un cas d'étude d'un système temps-réel embarqué, en l'occurrence le système de contrôle d'ascenseurs (SCA).

Mots-clés : Systèmes temps-réel embarqués, UML MARTE, réseaux de Petri temporellement temporisés, DTPN, automate temporisé, automate temporisé avec durées d'actions, daTA, spécification formelle, vérification formelle.

Abstract

Real-time Embedded systems (RTES) are often complex and critical, and require rigorous development to assert their functional correctness under time constraints. The UML MARTE profile (Modeling and Analysis of Real Time Embedded systems) is imposed as a standard specified by the OMG for the modeling and analysis of real-time embedded systems. It offers a general modeling framework to design and analyze STREs. However, the specifications in UML MARTE models lack the formal foundations required to accomplish a complete verification of these models. Therefore, the UML MARTE models cannot be used for systems verification and validation purposes. A combination of UML MARTE models and formal methods is an alternative to overcome such problem.

In this context, formal model-based verification (model-checking) is an important tool because it is equipped with a procedure allowing the automatic and exhaustive verification of the expected properties of the system under study. However, its importance model-based verification comes up against the state-space explosion problem, particularly if the number of clocks is large.

The research work of this thesis is situated in the field of the specification and the formal verification of embedded real-time systems. It consists in taking advantage of the advantages offered by the formal models temporally timed and based on semantics true parallelism. Thus, we present first, a model-oriented approach to formally specify and verify the real-time on-board systems described by the UML MARTE models. In addition, we formalize the semantics of UML2 sequence diagrams annotated by time constraints using the MARTE profile. Thus, we offer an operational method of translating these diagrams into temporally delayed Petri nets (DTPNs). The graphic and formal frozen translation (MARTESDtoDTPN) are developed and applied on illustrative examples. The formal verification uses the semantic structures associated with the specifications described in DTPN, namely timed automata with duration of actions (daTAs). Indeed, the state space of a daTA is expressed by a regions graph or by a zones graph based on maximality, and the verification can be done by tools at UPPAAL or KRONOS. This contribution is validated on a case study of a real-time embedded system, in this case the elevator control system (ECS).

Key Words: Real-time embedded Systems, MARTE UML, durational timed Petri nets (DTPN), timed automata, durational action timed automata (daTA), formal specification, formal verification.

ملخص

غالبًا ما تكون أنظمة الوقت الفعلي المضمنة (STRE) معقدة وحاسمة، وتتطلب تطويرًا صارمًا لتأكيد صحتها الوظيفية في ظل قيود الوقت. يتم فرض ملف تعريف UML MARTE (نمذجة وتحليل الأنظمة المضمنة في الوقت الفعلي) كمعيار محدد بواسطة OMG لنمذجة وتحليل الأنظمة المضمنة في الوقت الفعلي. يقدم إطار عمل نمذجة عام للتصميم وتحليل .STREs ومع ذلك، فإن المواصفات في نماذج UML MARTE تفتقر إلى الأسس الرسمية المطلوبة لإنجاز التحقق الكامل من هذه النماذج .لذلك، لا يمكن استخدام نماذج UML MARTE لأغراض التحقق من الأنظمة والتحقق من صحتها. مزيج من نماذج UML MARTE والأساليب الرسمية وبديل للتغلب على مثل هذه المشكلة.

في هذا السياق، التحقق الرسمي القائم على النموذج (فحص النموذج) أداة مهمة لأنها لديه إجراء للتحقق التلقائي وشاملة للخصائص المتوقعة للنظام قيد الدراسة. على الرغم من أهمية التحقق المستند إلى النموذج، فإنه يواجه مشكلة الانفجار الاندماجي لفضاء الحالة، خاصة إذا كان عدد الساعات كبيرًا.

يقع عمل هذه الأطروحة ضمن نطاق المواصفات والتحقق الرسمي لأنظمة الوقت الحقيقي المضمنة. وهي تتمثل في الاستفادة من المزايا التي توفرها النماذج الرسمية الموقوتة زمنياً بناءً على دلالات التوازي الحقيقي. وبالتالي، نقدم أولاً نهجًا موجهًا نحو النموذج لتحديد أنظمة الوقت الحقيقي المضمنة التي تم وصفها بواسطة نماذج UML MARTE والتحقق منها رسميًا. بالإضافة إلى ذلك، قمنا بإضفاء الطابع الرسمي على دلالات مخططات تسلسل UML2 المشروحة بواسطة قيود الوقت باستخدام ملف تعريف .MARTE وبالتالي، نقترح طريقة تشغيلية لترجمة هذه المخططات إلى شبكات بتري (DTPNs) مؤجلة مؤقتًا. تم تطوير وتطبيق التجميد الرسومي والرسمي للترجمة (DSMARTEtoDTPN)على أمثلة توضيحية. يستخدم التحقق الرسمي الهياكل الدلالية المرتبطة بالمواصفات الموضحة فيDTPN وهي الأوتومات الموقوتة مع مدة الإجراءات .(daTAs) في الواقع، يتم التعبير عن مساحة الولاية في daTA من خلال رسم بياني للمناطق أو رسم بياني للمناطق بناءً على الحد الأقصى، ويمكن إجراء التحقق بواسطة أدوات مثل UPPAAL أو .KRONOS يتم التحقق من صحة هذه المساهمة في دراسة حالة لنظام الوقت الفعلي، في هذه الحالة نظام التحكم في المصعد (SCA).

كلمات مفتاحيه: أنظمة الوقت الحقيقي المشحونة، UML MARTE، شبكات بتري المتأخرة زمنيا (DTPN)، التشغيل الآلي الموقوت، التشغيل الآلي مؤقت مع مدة الإجراءات (daTA)، المواصفات الرسمية، التحقق الرسمي.

Table Des Matières

Table des figures

Liste des Tableaux

1. Introduction Générale

Sommaire

« Si nous ne changeons pas notre façon de penser, nous ne serons pas capables de résoudre les problèmes que nous créons avec nos modes actuels de pensée».

Albert Einstein

Chapitre 1

Introduction générale

1.1 Introduction

Aujourd'hui, les systèmes temps-réel embarqués sont de plus en plus présents dans notre vie quotidienne et fonctionnelle. Il est désormais difficile de trouver un domaine démuni de ces systèmes. Les applications les plus connues des systèmes temps-réel embarqués sont les systèmes de transport (voiture, avion, train) et les systèmes mobiles autonomes (robot, fusé, satellite). De même, les systèmes liés à la gestion d'un périphérique (imprimante, souris sans fil), à la mesure (acquisition en temps-réel) et les systèmes domotiques (électroménager) sont des systèmes embarqués pouvant posséder des contraintes temps-réel liées aux capteurs ou actionneurs utilisés. La notion d'embarqué peut être étendue aux objets portables grand public (carte à puce, assistant personnel, téléphone mobile, lecteur, vidéo, consoles de jeu). Le point commun de tous ces systèmes porte la spécificité de leurs contraintes [Js05]. Ces systèmes sont souvent critiques dotés d'un comportement qui est contraint par le temps, dans le sens où ils doivent interagir correctement avec l'environnement non seulement au regard des informations échangées, mais également au regard des instants auxquels ces interactions se produisent.

Actuellement, les systèmes temps-réel embarqués, souvent cachés aux utilisateurs, sont complexes, le risque de les mal concevoir est un problème croissant. Par conséquent, la présence d'un comportement indésirable dans un module ou un logiciel critique peut mener à des conséquences dramatiques, il existe de nombreux exemples de notre vie courante de dysfonctionnements qui avaient entrainé des pertes de vies humaines et des dégâts matériels ou financiers [Pri03, Her01]. Ces dysfonctionnements ont toutes un point en commun : la disproportion entre le coût des dégâts qu'elles ont engendrés et la simplicité des erreurs à la source du problème.

La conception de tels systèmes exige des méthodes et des outils permettant de prendre en compte dans ses phases amont, non seulement les exigences fonctionnelles concernant la correction du calcul mais également les exigences extra-fonctionnelles relatives à l'utilisation

optimale de ressources tels que le temps, la mémoire et l'énergie. A ces exigences s'ajoutent celles de l'autonomie, la réactivité et la robustesse. L'objectif d'outils de conception des systèmes temps-réel embarqués est d'aider le concepteur à spécifier, à vérifier rapidement, générer automatiquement des exécutifs conformes à la spécification, et assurer une grande maîtrise de la sûreté de fonctionnement. De ce fait, l'étude de systèmes temps-réel embarqués fait appel à des modèles et des méthodes formelles temporelles et/ou temporisées permettant de répondre aux exigences auxquelles sont soumises de tels systèmes. Ainsi, diverses techniques de spécification et de vérification ont été développées et étendues, offrant donc un ensemble d'outils permettant de garantir le bon fonctionnement du système spécifié.

1.2 Problématique

Afin d'offrir des mécanismes de modélisation avancés, plusieurs études ont proposé des approches d'ingénierie basées sur les modèles, et l'utilisation des notations semi-formelles tels que UML (Unified Modeling Language) [RHCF05], enrichie avec les notations formelles. Par exemple, le profil UML MARTE est le premier profil OMG UML dédié à la modélisation et à l'analyse des systèmes embarqués temps-réel [Gér07, OMG08]. Ce profil vise à exprimer des contraintes temporelles sur des modèles UML. Cependant, les modèles qui sont construits ne peuvent pas être analysés formellement pour la vérification des exigences temporelles définies par le concepteur. Pour répondre à ces exigences, des approches analytiques formelles ont été développées afin d'intégrer ou d'étendre les modèles formels et les méthodes formelles dans le processus de conception de systèmes temps-réel embarqués développés avec le profil UML MARTE.

Il existe peu d'approches qui intègrent la vérification formelle aux modèles UML MARTE dans la littérature. Dans [YYSQ12], une approche intéressante a été proposée. Elle interprète les activités parallèles, modélisées dans le diagramme de séquence avec des annotations UML MARTE, par des transitions parallèles dans un réseau de Petri comme spécification. Plus précisément, la spécification est écrite dans le formalisme des réseaux de Petri colorés temporisés avec arcs inhibiteurs (TCPNIA en anglais timed colored Petri net inhibitor arcs) [NH10]. Dans ce travail, la durée d'une activité est intégrée comme un intervalle de contrainte associé à la transition correspondante du réseau de Petri. Pour vérifier certaines propriétés, la spécification TCPNIA est translatée en automates temporisés (Timed Automata) [AD90, AD94] afin d'utiliser certains outils de vérification de modèle existants comme *SMV*. Cependant, cette approche n'exprime que la durée de l'activité sans prendre en considération la spécification de temps de *latence* (*latency*), de *retard* (*delay*) et de *congestion*. Etant donné que le modèle des automates temporisés est basé sur la sémantique d'entrelacement, il n'y a aucun moyen d'exprimer l'exécution parallèle de deux activités. Pour surmonter cette limitation, chaque activité ayant une durée non nulle est interprétée par deux transitions séquentielles modélisant les événements de début et de fin de l'activité. Bien que cette solution soit correcte, elle présente de sérieux inconvénients. En fait, un plus grand nombre de transitions dans la spécification du réseau de Petri entraîne une augmentation significative du nombre d'horloges dans les automates temporisés associés. En effet, le nombre de zones, respectivement de régions, dans le graphe de zones, respectivement de régions, est exponentiel au nombre d'horloges [AD94]. Ainsi, cela conduit au problème de l'explosion combinatoire de l'espace d'états [CY92, GRR06, BFLM18]. Pour remédier à ce problème, on peut utiliser les modèles du vrai parallélisme basés sur

la sémantique de maximalité qui permettent d'éviter ces difficultés et de conserver la nature parallèle des systèmes concurrents (parallèles). De plus, nous mettons l'accent sur la nécessité de spécifier, à la fois les contraintes temporelles et les durées explicites des actions.

Dans cette thèse, nous nous intéressons aux deux modèles basés sur la sémantique de maximalité, appelés respectivement Time Petri Nets with action Duration (DTPN) et durational action Timed Automata (daTA). Ces deux modèles se présentent comme une alternative intéressante aux modèles des diagrammes de séquence UML2 annotés par le profil MARTE du point de vu spécification et vérification formelle par *modèle-checking*. Le premier modèle, DTPN est destiné principalement à la tâche de spécification des systèmes, où nous nous intéressons beaucoup plus aux actions non atomiques (la non-atomicité temporelle et structurelle des actions), aux contraintes temporelles et aux durées explicites des actions. Le deuxième modèle basé sur la sémantique de vrai parallélisme, à savoir la sémantique de maximalité, daTA est destiné à la vérification formelle avec la possibilité de spécifier aussi bien les durées explicites des actions que les contraintes temporelles et les contraintes d'urgence agissant sur le comportement du système temps-réel à analyser.

1.3 Contributions

Dans ce travail nous nous sommes intéressés à la spécification et la vérification formelle des systèmes temps-réel embarqués en utilisant le profil MARTE, et particulièrement les diagrammes de séquence annotés par des contraintes temporelles. Nos contributions peuvent être résumées comme suit :

— Etude d'un sous-ensemble représentatif des langages de modélisation conçue spécialement pour l'analyse et la conception de systèmes temps-réel embarqués, chacun mettant un accent particulier plus sur certains aspects du système que sur d'autres. Une comparaison ensuite est réalisée sur la base de critère *temps réel*, en s'appuyant sur un ensemble de propriétés qui sert de fil conducteur. Cette étude permet aux concepteurs de systèmes temps-réel embarqués de mieux choisir le (ou les) langage (s) à utiliser selon leurs attentes.

— Formalisation des diagrammes de séquence UML2 avec des annotations de profil MARTE pour répondre aux exigences des systèmes temps-réel.

— Proposition d'une sémantique opérationnelle pour les diagrammes de séquence UML2 annotés par des contraintes temporelles en utilisant le profil UML MARTE. Le modèle sémantique sous-jacent est un réseau de Petri temporellement temporisé. Basé sur la sémantique de maximalité, la sémantique des spécifications DTPNs sont définis en termes d'automates temporisée avec durées actions (daTAs).

— Application de l'approche proposée pour la spécification et la vérification formelle des STRE en utilisant le profil UML MARTE sur une étude de cas réel, en l'occurrence le système de contrôle de deux ascenseurs installés dans un immeuble couvrant plusieurs étages, afin de montrer les gains de profil UML MARTE, le modèle formel de haut niveau DTPN, et le modèle de bas niveau daTA, qui ont été adoptés dans ce travail.

1.4 Plan du manuscrit

Ce manuscrit est organisé en deux parties distinctes. La première partie est consacrée à un état de l'art sur les différents langages, modèles et techniques de la vérification formelle des systèmes temps-réel et embarqués afin de permettre une bonne compréhension de notre contribution. Nous l'avons subdivisé en trois chapitres :

— Le chapitre 2 traite en première partie, les différentes techniques offertes par l'ingénierie dirigée par les modèles et qui peuvent être utilisées pour améliorer le processus de développement des systèmes temps-réel embarqués. La deuxième partie est consacrée à la littérature qui s'intéresse aux langages de modélisation dédiés aux systèmes temps-réel et embarqués. Entre autre les langages UML, UML-RT, MARTE, SysML, et AAD qui sont non seulement les plus traités dans la littérature du domaine, mais aussi les plus récents et répondus. Les caractéristiques et les concepts fondamentaux de ces langages ou profils sont étudiés et discutés afin de discerner leurs limites. L'utilisation d'un tel langage ou profil pour un développement reste difficile sans au préalable définir des critères qui permettent de faire le choix en fonction des besoins du concepteur. Pour l'aspect temps-réel, on y déterminé certaines propriétés destinées à guider les développeurs des systèmes temps-réel et embarqués dans le choix de (ou des) langage ou profil (s) de modélisation le(s) plus adapté(s) pour leur conception.

— Le chapitre 3 introduit quelques modèles formels utilisés dans le contexte temps-réel, à savoir les algèbres de processus, les réseaux de Petri et les automates temporisés. Il présente les différentes extensions temporelles et temporisées de chaque modèle utilisé pour la spécification, en particulier le comportement contraint par le temps d'un système temps-réel. Il détaille le modèle des réseaux de Petri temporels avec durée d'action, l'une des extensions temporelles de réseau de Petri temporel, ainsi que le modèle des automates temporisés avec durée d'action qui sont utilisés dans cette thèse.

— Le chapitre 4 met l'accent sur les techniques de vérification formelle par model-checking, principe, méthodes, outils et limites. Le model-checking constitue dans notre contexte de travail un très important outil permettant la vérification formelle automatique et exhaustive par des model-checker de propriétés attendues sur une spécification du système. Ces dernières sont écrites dans une logique temporelle, à savoir la logique temporelle linéaire LTL, la logique temporelle arborescente CTL et la logique temporelle arborescente temporisée TCTL, qui sont les logiques que nous considérons dans cette thèse.

La deuxième partie dédiée aux contributions de cette recherche. Elle est structurée en trois chapitres (chapitres 5,6 et 7) :

— Le chapitre 5 donne en premier un aperçu général du processus proposé pour la vérification formelle des spécifications semi-formelles UML MARTE. Ensuite, il définit la sémantique formelle proposée pour les diagrammes de séquence UML2 avec des annotations UML MARTE afin de décrire les contraintes temporelles. Puis, il développe la méthode opérationnelle de la translation des diagrammes de séquence UML2 avec des annotations temporelles UML MARTE en réseaux de réseau de Petri temporellement temporisée (DTPN). Nous explorons les concepts de base (ligne de vie, objet, message), ainsi que les

concepts avancés (fragments combinés seq, strict, par, alt, opt, loop) des diagrammes de séquence UML2 annotés par le profil UML MARTE. Par la suite, chaque concept est appliqué par un exemple illustratif.

— Chapitre 6 illustre les profits de l'approche proposée en l'appliquant sur un exemple d'étude d'un système temps-réel embarqué, en l'occurrence le système de contrôle de deux ascenseurs installés dans un immeuble couvrant plusieurs étages.

— Chapitre 7 présente et discute quelques travaux ont été réalisés par d'autres auteurs sur la translation des modèles UML MARTE en langages ou modèles formels pour la vérification formelle, et particulièrement les diagrammes de séquence UML2. Ainsi, les gains du travail proposé ont été mis en évidence.

Nous terminons par le chapitre 8, qui reprend la conclusion et énoncé quelques pistes de recherche future, qui s'inscrivent en partie dans la continuité de la thèse.

Première partie : État d'art

Modèles & Outils

" L'efficacité d'un projet informatique augmente

si toutes les préoccupations de nature différentes

sont bien modularisées "

— *Demeter, [LoD.1987]*

2. Modélisation des systèmes temps-réel embarqués

Sommaire

Chapitre 2

Modélisation des systèmes temps-réel embarqués

2.1 Introduction

Le développement de systèmes consiste à étudier, concevoir, construire, transformer, mettre au point et maintenir des logiciels. Ces différentes activités qui se situent à plusieurs étapes du cycle de développement, sont plus ou moins facilement réalisées en fonction de méthodes et/ou de langages de modélisation utilisés. Ces derniers permettent notamment de prendre connaissance des attentes de l'usager, créer un modèle « théorique du logiciel », qui servira de plan de construction, puis construire le logiciel, contrôler son bon fonctionnement et son adéquation au besoin. Ceci est aussi valable pour les systèmes embarqués et temps-réel. Cependant, ces derniers ont d'autres caractéristiques et contraintes [Gom84] que l'on ne retrouve pas dans les systèmes logiciels classiques, interaction avec leur environnement, contraintes temps-réel (temps d'exécution, période, etc.), contrôle temps-réel, traitements concurrents, ressources limitées, etc., nécessitent aussi dans leur développement des activités d'analyse des modèles produits avant leur mise en œuvre. Il faut donc utiliser pour leur développement des langages différents et des formalismes variés prenant en compte ces spécificités ou chacune étant adaptée à un métier donné [EL03].

La spécification sert de point de départ à la conception du logiciel. Elle doit donc décrire à un haut niveau d'abstraction les algorithmes, l'architecture matérielle et les contraintes temporelles d'exécution des algorithmes sur l'architecture [KSdeV00]. L'objectif primordial de cette spécification réalisée à l'aide d'un langage de modélisation de haut niveau est d'obtenir un modèle simulable et représentatif de l'implémentation finale de système. Il existe dans la littérature plusieurs langages de modélisation qui ont été développés pour l'analyse et la conception de systèmes temps-réel embarqués, chacun mettant un accent particulier plus sur certains aspects du système que sur d'autres. Le présent chapitre est consacré à l'étude de quelques langages de modélisation, UML, UML-RT, MARTE, AADL et SysML, et nous présentons en détail MARTE, le langage de modélisation ou le profil UML qui est notre choix dans ce travail. Ces derniers langages sont non seulement les plus traités dans la littérature du domaine mais aussi récents et répandus dans le domaine de STRE.

2.2 Génie logiciel et ingénierie dirigée par les modèles

Le principal problème du génie logiciel (GL) ne se pose plus en termes de « *donnez-moi une spécification immuable et je produis une implantation de qualité* », mais plutôt en «*comment réaliser une implantation de qualité avec des spécifications continuellement mouvantes*» [JGB06]. Quel qu'il soit, un processus de développement logiciel englobe un certain

nombre d'activités (comme l'expression des besoins, l'analyse, la conception, l'implantation ou encore la validation) qui chacune produit un ou plusieurs artefacts (documentation, diagrammes, codes sources, fichiers de configuration, fiches de tests, rapports de qualification, etc.). Ces artefacts donnent de multiples points de vue sur le logiciel en cours de développement et, en pratique, sont souvent indépendants les uns des autres. Un problème ici, est d'être capable d'assurer une cohérence entre ces vues, ou au minimum une traçabilité entre les éléments des différents artefacts. L'ingénierie dirigée par les modèles (IDM) y apporte des éléments de réponse.

L'IDM a pour principal objectif de relever un certain nombre de défis du génie logiciel (qualité, productivité, séparation des préoccupations, portabilité, traçabilité, réutilisabilité, maintenabilité, cohérence entre modèles et code, coût, etc.) en suivant une approche à base de modèles dite générative. En focalisant le raisonnement sur les modèles, l'IDM permet de travailler à un niveau d'abstraction élevé, et par la suite de vérifier sur des modèles du système, un ensemble de propriétés que l'on devait vérifier auparavant sur le système final. Avec cette approche, l'IDM permet de lever le verrou consistant à ne pouvoir tester un système que tardivement dans le cycle de vie [DLC10]. L'IDM va également permettre de capitaliser le savoir-faire au niveau des modèles et non pas uniquement au niveau du code source, favorisant ainsi la réutilisabilité, la productivité et la traçabilité. Quelle que soit la technique mise en œuvre pour passer du modèle d'un système à une application logicielle exécutable, il est nécessaire que la sémantique du modèle à exécuter soit clairement définie et surtout non ambigüe. La complétude d'un modèle va influer directement le niveau d'exécution atteignable, par l'une ou l'autre des techniques [JGB06].

2.3 Ingénierie dirigée par les modèles pour la conception de systèmes temps-réel embarqués

L'ingénierie du logiciel a considérablement évolué. On a ainsi vu apparaître au fil du temps de nouveaux paradigmes de programmation. On peut notamment citer la programmation fonctionnelle, le paradigme objet et plus récemment les composants et les aspects. Chacune de ces approches vient avec ses avantages et ses inconvénients selon les utilisations, et induit une vision différente de la programmation. Par ailleurs, avec l'augmentation des besoins encouragée par l'amélioration des performances des ordinateurs, la taille des programmes n'a eu de cesse d'augmenter, et le besoin d'outils pour appréhender des grandes quantités de code s'est fait ressentir. Ainsi, des langages avec un plus haut niveau d'abstraction sont apparus, parmi lesquels UML [OMG03]. Le langage UML présente de nombreux diagrammes pour représenter différentes facettes d'un système pendant son développement. Mais il s'est avéré limité, car ne pouvant être totalement en adéquation avec tous les domaines, par exemple l'ingénierie système, les systèmes temps-réel, la robotique, etc. L'Ingénierie Dirigée par les Modèles (IDM) est née de ce besoin de pouvoir créer des langages de modélisation adaptés à chaque situation.

L'IDM ou MDE (Model Driven Engineering) [BB04, Com08], est une discipline qui a pour vocation l'automatisation et la sûreté du développement des systèmes logiciels complexes particulièrement les systèmes embarqués, en fournissant des outils et des langages permettant la transformation de modèles, d'un niveau d'abstraction à un autre ou d'un espace technologique à un autre. C'est un paradigme assez récent de l'ingénierie du logiciel. Son principe essentiel consiste à

considérer les modèles comme entités de base dans le développement d'un système, et non plus l'objet comme c'est le cas dans le paradigme objet. De la même manière que le principe *tout est objet* a permis de faire avancer la technologie objet, le principe *tout est modèle* est aussi essentiel pour l'ingénierie dirigée par les modèles. Il s'agit de considérer à priori que toute entité manipulée par un système informatique est un modèle. Avec cette technologie les modèles, dont l'utilisation a longtemps été limitée à la documentation des logiciels, font désormais partie de la définition de ces derniers. Ils sont utilisés pour décrire à la fois le problème posé et sa solution. Il existe plusieurs implémentations de l'IDM telles que le MDA (Model Driven Architecture) [Sol00] de l'OMG (Object Management Group), le MIC (Model Integrated Computing) [SK97], les usines à logiciel (SoftwareFactories) [GS04], SNets (Semantic Network) [Béz05a], etc.

2.3.1 Modélisation et méta-modélisation

Un modèle est une abstraction, une simplification d'un système qui est modélisée sous la forme d'un ensemble de faits construits dans une intention particulière. Une définition plus complète est donnée par B. Selic dans [Sel03]. Un modèle doit pouvoir être utilisé pour répondre à des questions sur le système modélisé [Com08]. Les modèles permettent de contenir toute l'information. Il faut que le modèle puisse être représente graphiquement, c'est ce qui le rend facilement accessible et c'est notamment ainsi qu'il peut aider à gérer un système complexe (par rapport à une représentation textuelle).

Un méta-modèle est un modèle qui définit le langage d'expression d'un modèle, c'est à dire le langage de modélisation [Poe06]. Il doit être accompagné d'une documentation, écrite en langage naturel. Avec cette documentation, tout ce que l'on peut vouloir exprimer est représentable à l'aide d'un modèle conforme au méta-modèle.

2.3.2 Transformation des modèles

Les deux principaux artefacts de l'ingénierie des modèles étant les modèles et les transformations de modèles. La transformation de modèle est également utilisée dans la définition des langages de modélisation pour établir les « mappings » et des traductions entre différents langages. Nous exprimons les principes de transformation de modèle à travers les trois définitions suivantes :

- **Transformation :** Une transformation est la génération automatique d'un (de) modèle(s) cible(s) à partir d'un (de) modèle(s) source(s), suivant une définition de transformation [Béz05b].

- **Définition de transformation** : Une définition de transformation est un ensemble de règles de transformation qui spécifient comment un modèle décrit dans un langage source, peut être transformé en un modèle décrit dans le langage cible [Béz05b].

- **Règle de transformation** : Une règle de transformation décrit comment une ou plusieurs constructions en langage source peuvent être transformées en une ou plusieurs constructions en langage cible [KWWB03].

La figure 2.1 schématise le principe de transformation. Selon le schéma de la figure 2.1, les règles de transformation sont spécifiées en s'appuyant sur les langages (méta-modèles) sources

et cibles. Le moteur de transformation prends en entrée un modèle source, lit les règles de transformation, et produit un modèle cible, qui est conforme au méta modèle cible.

On peut utiliser la transformation de modèle pour générer du code dans un langage de programmation cible (Java, C++, etc.) à partir de modèles sources. Dans ce cas on parle de la transformation de modèles à texte (M2T). Il s'agit d'un cas particulier de transformation de modèles à modèles dans lequel le méta modèle cible s'identifie à la grammaire textuelle du langage de programmation.

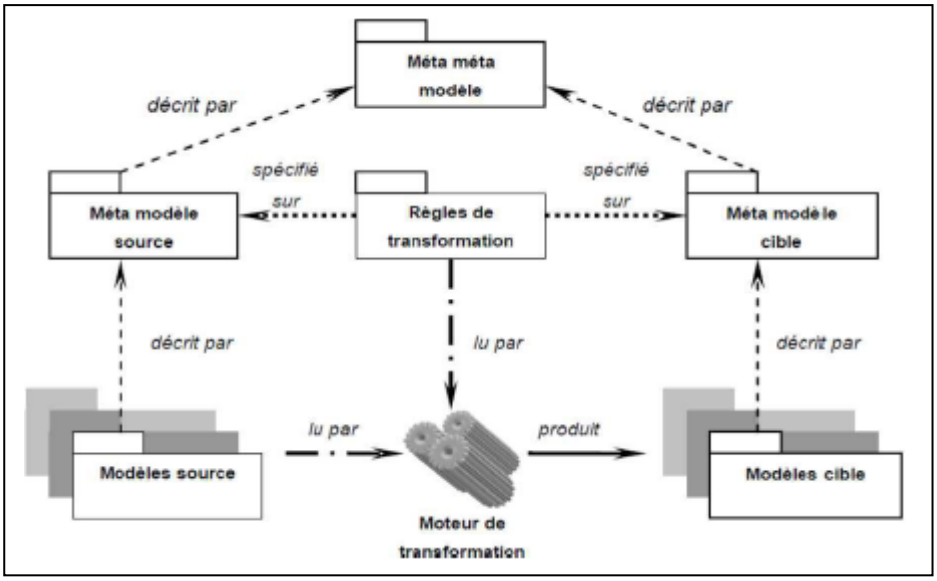


FIGURE 2.1– Principe d'une transformation de modèles [Lab10]

2.3.3 Classification des transformations

Dans une transformation de modèles, les modèles sources et cibles peuvent :

- être conforme à des méta-modèles identiques ou non ;

- appartenir à un même niveau d'abstraction ou à des niveaux d'abstractions différents.

En fonction de ces points de variation, on classe les transformations en quatre catégories.

Selon [MVG06], une transformation est :

— ***Endogène*** : si les modèles source et cible sont conformes au même méta-modèle ;

— ***Exogène*** : si les modèles source et cible sont conformes à des méta-modèles différents ;

— ***Horizontale*** : si les méta-modèles se situent au même niveau d'abstraction ;

— ***Verticale*** : si les méta-modèles se situent à des niveaux d'abstraction différents.

***Remarque* :** *Actuellement, il n'y a pas de consensus sur les outils ou les méta-modèles à utiliser pour écrire et exécuter une transformation de modèles. Il existe de nombreuses propositions mais aucune n'est validée par la communauté. Citons quand même QVT* [OMG11] *(Query View Transform) qui a été récemment standardisé. Il existe aussi EMF [EMP14] (Eclipse Modeling Framework) qui est une bibliothèque Java pour créer et modifier des modèles.*

2.3.4 Approches pour la transformation des modèles

Il existe plusieurs approches pour la transformation de modèles [CH03, Gar13] qui sont classées en deux grandes familles : les approches de modèles à modèles (M2M), et les approches de modèle à texte (M2T). Selon l'IDM, une approche M2T ne devrait pas se distinguer d'une approche M2M, le code étant assimilé à un modèle. D'ailleurs, une transformation vers du code implique théoriquement que l'on fournisse comme cible le méta-modèle d'un langage de programmation. Dans la pratique, il est courant de s'affranchir de cette contrainte technique. Il est dans bien des cas plus simple et plus rapide de travailler sur un squelette de code composé de parties fixes et variables, que de spécifier des règles en fonction d'un méta-modèle cible. Très souvent donc, en M2T le code est tout simplement généré sous forme de texte, dans un fichier, à partir d'un modèle, d'un méta-modèle source et de règles de transformation. Ce fichier de code généré est ensuite directement utilisable par un compilateur ou un interpréteur. C'est pourquoi ce genre de transformation est appelée « Model To Text ». Nous présentons ci-dessous quelques approches et outils de chaque famille. Comme approche de transformations de modèles à texte, nous pouvons citer :

- ***La génération de code par parcours de modèle (visitor-based)* :** elle consiste à parcourir la représentation interne du modèle donné en entrée et à écrire du texte (code) sur un flux de sortie, ceci à l'aide de visiteur. On retrouve cette approche dans le Framework Jamda par exemple ;

- ***La génération de code par template (template-based)* :** C'est la solution préconisée par l'OMG. Il s'agit de l'approche M2T la plus employée. Elle consiste à utiliser comme modèle cible un texte fixe dans lequel certaines parties sont variables : elles sont renseignées en fonction des informations récupérées dans le modèle source. Comme le membre droit d'un template concerne du texte, il n'y a pas de notion de typage, ce qui peut présenter un inconvénient (erreurs de typage lors de l'interprétation,..etc). L'avantage d'une telle approche découle de cet inconvénient : l'approche par template est extrêmement souple et facilite la génération de n'importe quel artefact textuel.

La spécification de l'OMG nommée MOFM2 (MOF Model To Text Transformation Language) décrit un standard aligné avec UML 2.0, MOF 2.0 et OCL 2.0 permettant de générer du code à partir de templates. Cette spécification a été implémentée au sein de divers outils IDM, une majorité d'entre eux supporte désormais la génération de code par template. La plupart de ces implémentations sont disponibles sous forme de plugins sur la plateforme EMF, et font partie du projet Eclipse M2T. Parmi les plus utilisées actuellement, citons : XSLT, JET, Xpand, Acceleo. Comme approche de transformations de modèles à modèles, nous pouvons citer les :

- ***Approches par manipulation directe* :** elles offrent une représentation interne d'un modèle et des API (Application Programming Interface) pour les manipuler. Il incombe aux utilisateurs d'implémenter eux-mêmes leurs règles, en utilisant un langage de programmation classique.

Jamda utilise cette approche (qui est similaire dans son principe à l'approche M2T par parcours de modèle) ;

- ***Approches opérationnelles* :** variantes de la manipulation directe, mais située à un niveau d'abstraction un peu plus élevé. Une solution ici est d'étendre le méta formalisme avec un langage d'expression de requêtes, par exemple OCL, pour spécifier une sémantique opérationnelle. Cette approche se retrouve dans les langages tels que MTL (Model Transformation Language) [VJ04] et QVT Operationnal ;

- ***Approches relationnelles* :** elles utilisent des relations mathématiques pour lier les éléments source et cibles. Puis on spécifie les contraintes des liens entre la source et la cible d'une relation. Nous pouvons citer comme exemple : MOFQVT, MTL, KMTF (KentModel Transformation Language) et l'outil XMF-Mosaic.

- ***Approches basées sur les transformations de graphes* :** elles se basent sur la théorie de transformation de graphes. *Une règle de transformation = graphe source + graphe cible*. Parmi ces approches nous pouvons citer : AGG (Attributed Graph Grammar), l'environnement VMTS (Visual Modeling and Transformation System), BOTL (Bidirectional Object-oriented Transformation Language), la suite d'outils Fujaba (From UML to Java And Back Again) ;

- ***Approches dirigées par la structure* :** il y a une phase de création de la structure du modèle cible, puis une phase de renseignement des attributs et des références. OptimalJ est un exemple d'environnement implémentant cette approche ;

- ***Approches hybrides* :** Elles permettent, selon le contexte, de spécifier des règles de manière impérative, déclarative, ou bien de mélanger les deux.

Nous avons vu que dans une démarche IDM, les spécifications du logiciel sont décrites grâces à des modèles, qui sont les principaux artefacts du système. Dans ce monde où *tout est modèle*, il existe aujourd'hui un outillage conséquent permettant de spécifier ces modèles. Parmi ces outils nous avons UML qui est l'un des langages de description de modèles (un méta-modèle) le plus connu et le plus utilisé. Ce dernier fait partir des méta-modèles très généraux, qui ne sont pas liés à un domaine particulier. Mais l'IDM permet, à travers les techniques de méta-modélisation de définir des langages spécifiques à un domaine donné. La section suivante aborde cette notion.

2.4 Langages de modélisation spécifiques aux systèmes temps-réel ou embarqués

2.4.1 UML

Unified Modeling Language (UML) [RHCF05, Spe07] est un langage de modélisation orienté objet standardisé par l'OMG pour spécifier, construire, documenter et visualiser graphiquement les aspects d'un système logiciel. Il peut être considéré comme une combinaison de notations qui existent dans les méthodes comme (OMT et OOSE), etc. Le mot *unifié* implique une unification de différentes modélisations. L'ambition d'UML est de rassembler en une seule notation les meilleures caractéristiques des différents langages de modélisation à objets. Cette unification a aussi pour effet de donner une masse critique à UML. L'UML est un outil de modélisation qui guide la création et la notation de nombreux types de diagrammes, y compris les diagrammes comportementaux,

les diagrammes d'interaction et les diagrammes de structure. Il permet de voir le système sous deux (2) angles différents appelés vues : une vue statique et une vue comportementale. Dans sa version 2, UML offre 14 diagrammes, les détails sur ces derniers sont disponibles dans [OMG15]. Les diagrammes et les notations utilisés sont informels, mais il est possible d'utiliser le langage OCL (Object Constraint Language) pour formaliser les diagrammes utilisés. Bien que l'UML n'étant pas une méthode, il peut être combiné avec presque toute méthode de développement de systèmes temps-réel et/ou embarqués (Ex, ROOM, HOOD, YSM, MASCOT).

Formalisme de représentation

UML offre un ensemble de notations graphiques regroupées en diagrammes spécifiques à chaque aspect d'un système. Aux graphiques sont associés des textes qui expliquent leur contenu. Dans ce travail nous adoptons les diagrammes de séquence qui permettent de décrire le comportement dynamique de système et de mettre l'accent sur l'aspect temporel (ordre des messages). Ils sont particulièrement utiles pour spécifier des systèmes avec des fonctions dépendant du temps telles que des applications en temps-réel, et pour modéliser des scénarios complexes où la dépendance temporelle joue un rôle important.

Un diagramme séquence UML est un type de diagramme d'interaction. Il décrit une interaction spécifique en termes d'ensemble d'objets participants et une séquence de messages qu'ils échangent au fur et à mesure de leur déroulement afin de réaliser une activité souhaitée. La figure 2.2 représente un sous-ensemble du méta modèle de diagrammes de séquence UML 2.0. Les principaux fragments du diagramme de séquence sont représentés par les éléments de modèle ligne de vie message et fragments Combinés. Une ligne de vie est une ligne verticale qui montre l'existence d'un objet sur une période de temps donnée, l'ordre des événements placés sur la ligne de vie est significatif, indiquant en général l'ordre dans lequel ces événements se produiront. Les messages sont affichés sous forme de flèches horizontales d'une ligne de vie d'un objet à une ligne de vie d'un autre. Les messages sont représentés sous forme de flèches horizontales entre la ligne de vie d'une instance et la ligne de vie d'une autre. Un message est une communication entre deux instances d'objet qui peut provoquer l'invocation d'une opération, déclencher d'un signal, créer ou à détruire un objet. Un message spécifie le type de communication entre les objets (synchcall, asynchcall, reply, create, delete, etc). Dans ce travaille nous allons prendre en considération l'ordre et le type des messages, synchrone, asynchrone et réponse. Ainsi que les contraintes temporelles imposées sur les messages échangés entre les objets participant dans l'interaction.

Un diagramme de séquence décrit uniquement un fragment du comportement du système et le comportement complet du système peut être exprimé par un ensemble de diagrammes de séquence pour spécifier toutes les interactions possibles pendant le cycle de vie de l'objet. Une interaction est une unité de comportement qui se base sur les transmissions d'information observables dans temps entre les objets connectés. Chaque interaction peut être provoquée par des actions exécutées par les objets communiqués. Comme défini dans [MW08], *an action takes a set of inputs and converts them into a set of outputs, though either or both sets may be empty.* Par exemple, une action peut être une opération d'appel, événement d'envoi ou de recevoir, envoyer ou

recevoir signal, écrire ou lire une variable. L'exécution des actions peut conduire par un événement comme un appel à une opération ou une émission d'un signal.

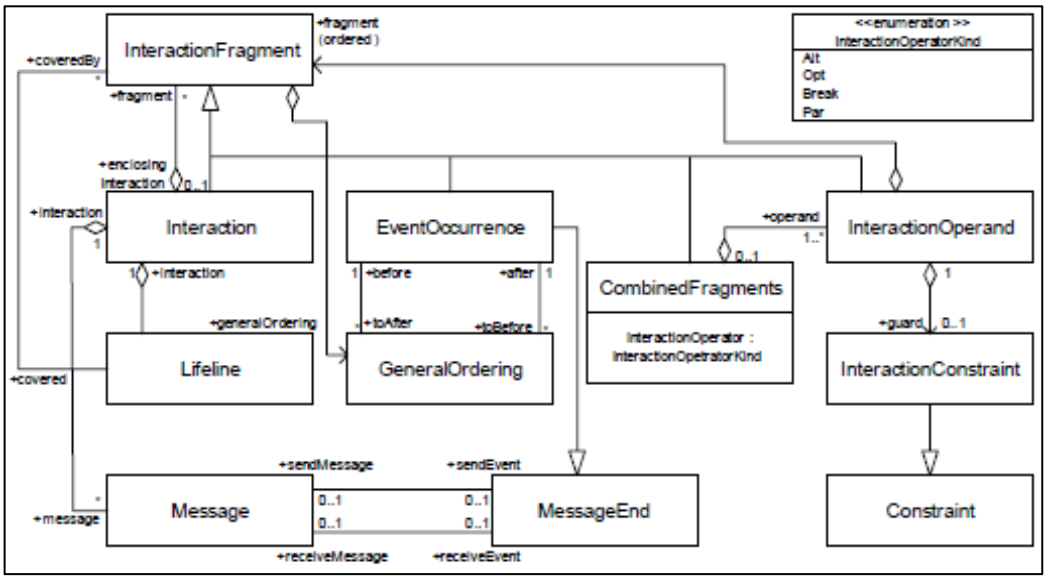


FIGURE 2.2– Meta modèle de diagramme de séquence [YYSQ12]

Pour obtenir des interactions plus complexes on peut utiliser la technique de fragments combinés. Un fragment combiné se compose d'un opérateur et d'un ensemble d'opérandes. L'opérateur conditionne la signification du fragment combiné et détermine le nombre d'opérandes de fragment. Il existe treize (13) d'opérateurs définis dans la notation UML2 : *alt*, *seq*, *weak*, *par*, *op*, *loop*, *ref*, *assert*, *neg*, *critical*, *break*, *ignore*, *consider*. Dans le cas de *neg*, *assert*, *loop* et *ref* le fragment a exactement un opérande, tandis que pour la plupart d'autre des opérateurs ils ont plusieurs. Les fragments et leurs opérateurs peuvent être inductivement combinés pour décrire des interactions plus complexes. La figure 2.3 montre un exemple d'un diagramme de séquence en utilisant des constructions de UML2.

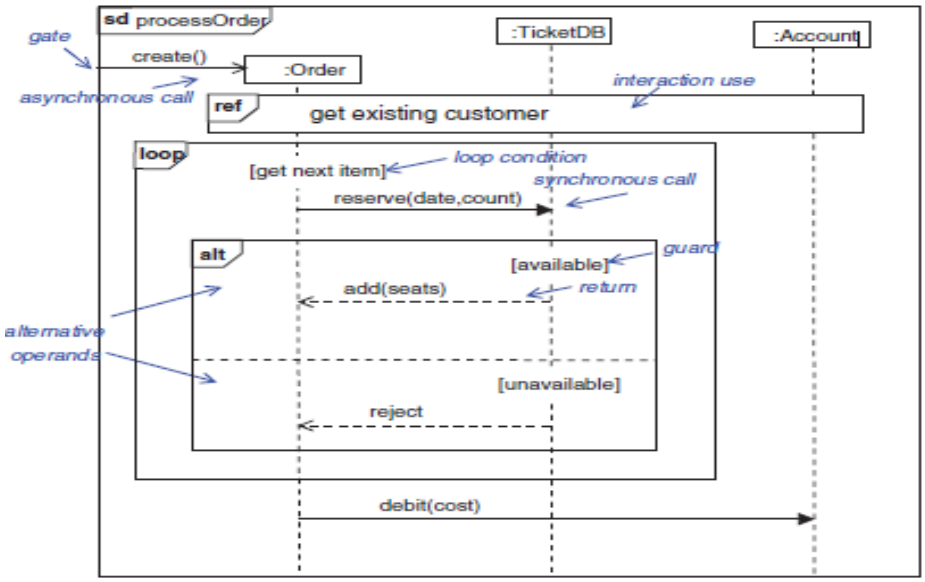


FIGURE 2.3– Diagramme de séquence [BM10]

Extensibilité d'UML

Quelques modèles peuvent être directement étudiés par le langage UML, d'autres nécessitent une spécialisation liée à leur classe d'application. Cette nécessité a conduit vers l'introduction des extensions sans avoir à modifier le langage UML et rendues possibles via la notion du profil [VSB+13]. Un profil correspond au regroupement d'extensions et de spécialisation du langage UML du point de vue notation et sémantique. Ceci est principalement réalisé via le concept de stéréotypes [Gli09] qui représentent des fonctions d'annotation standard pour exprimer les propriétés fonctionnelles et extra-fonctionnelles, ainsi que des moyens pour en introduire de nouvelles. Un stéréotype ajoute une sémantique à l'élément UML (classe, association, attribut, etc) sur lequel il est placé. Dans le domaine de systèmes temps-réel et embarqués, plusieurs profils ont été proposés pour permettre la prise en compte des caractéristiques de base de ces systèmes dans un modèle UML, aussi bien sur le plan modélisation et analyse que sur le plan validation et génération du code exécutable.

2.4.2 UML-RT

UML-RT (Unified Modeling Language Real Time) [Ant01, FOW01] est une extension d'UML pour la modélisation des systèmes logiciels embarqués réactifs et temps-réel. C'est un profil UML proposé comme langage de modélisation de tels systèmes. Il se focalise particulièrement sur la description du système au niveau architectural, et la définition de la structure générale du système. Comme UML, UML-RT offre une notation graphique, mais définit de nouveaux concepts par rapport à UML. Nous avons :

— *Les Capsules* : décrivent les composants complexes du système qui peuvent interagir avec leur environnement. Leur interaction avec l'environnement est gérée en leurs rattachant des ports, qui sont le seul moyen d'interaction avec l'environnement. Ils sont souvent associés à des protocoles qui régulent le flux d'informations passant par un port. Un port peut en outre être public ou privé. Les ports publics sont situés à la frontière d'une capsule. Les connecteurs sont utilisés pour interconnecter deux ou plusieurs ports de capsules, et décrivent ainsi les relations de communication entre les capsules.

— *Le Protocole* : c'est un ensemble de signaux envoyés et reçus le long des connecteurs.

La représentation explicite des ports, des protocoles et des connecteurs permet la construction de modèles architecturaux à partir d'une collection de capsules. Pour la représentation de l'architecture du système, un diagramme de collaboration des capsules est défini et décrit les composants du système et la façon dont ils sont connectés. Pour décrire comment les capsules réagissent aux messages reçus ou envoyés via leurs ports, UML-RT reprend le diagramme d'état transition d'UML.

Formalisme de représentation

Dans UML-RT, les différents concepts sont représentés pour :

— La capsule par un box labellisé libellé avec le nom de la capsule ;

— Le port par un petit carré vide ou plein situé sur la bordure de la capsule ;

— Le connecteur par une ligne entre deux (2) ports.

La figure 2.4 donne une représentation de ces différents concepts. Sur cette figure, nous avons deux capsules portant chacune un port public, les deux reliés par un connecteur.

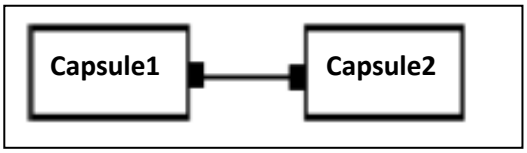


FIGURE 2.4– Notations utilisées par UML-RT

2.4.3 MARTE

MARTE (Modeling and Analysis of Real Time and Embedded systems) [Gér07, OMG08] est un profil UML ayant pour objectif l'utilisation d'une approche dirigée par les modèles pour le développement des systèmes embarqués temps-réel. Ce profil est destiné à remplacer le profil UML SPT (ordonnancement, performances et temps) [OMG05], qui n'est pas conforme avec les normes UML2 et MDA. Il consiste à définir des fondations pour la description à base de modèles des applications à temps-réel et systèmes embarqués. MARTE fournit des concepts de base et avancés de modélisation de temps, tels que les stéréotypes permettant la prise en compte des comportements temporels des systèmes. Ces concepts sont ensuite raffinés à la fois pour la modélisation détaillée et l'analyse des différents aspects du système. La figure 2.4 présente l'architecture globale du profil MARTE selon une décomposition en paquetages. MARTE est structuré selon deux directions, la modélisation des concepts des systèmes embarqués temps-réel et l'annotation des modèles d'applications pour supporter l'analyse des propriétés de systèmes. L'organisation du profil reflète cette structure, par la séparation des deux paquetages MARTE *design model* et MARTE *analysis model*. Comme illustré en figure 2.5, la spécification de MARTE est composée de 14 sous profils regroupés en quatre paquetages principaux traitant chacun un aspect du système :

— *MARTE Fondations* : c'est le paquetage de base qui définit des notions fondamentales dans le domaine de l'embarqué. Il fournit des modèles de construction pour spécifier des propriétés non fonctionnelles (NFPs en anglais Non-Functional Properties,). Une NFP peut être soit élémentaire ou complexe, qualitative ou quantitative. Il introduit des notions de temps pour modéliser des aspects temporels de systèmes (Time), de ressources abstraites (GRM en anglais Generic Resource Modeling) nécessaires à la modélisation de plateformes générales dédiées à l'exécution des systèmes embarqués temps-réel. Le sous paquetage GCM (Generic Component Model) rassemble les concepts nécessaires pour une modélisation à base de composants. Enfin, le paquetage Allocation modeling (Alloc) permet de décrire des éléments d'association matériel et logiciel ;

— *MARTE Design model* : des concepts nécessaires pour modéliser des modèles de calculs et de communication sont introduits dans ce paquetage. En plus, des plateformes d'exécution logicielles

(SRM) et matérielles (HRM) sont définies dans le paquetage Detailed Modeling Resource (DRM). Les sous paquetages SRM et HRM sont un raffinement du paquetage GRM ;

— *MARTE Analysis model* : ce paquetage introduit les éléments communs qui peuvent être utilisés pour contribuer à des nombreux types d'analyse. Il contient le paquetage Generic Quantitative Analysis Modeling (GQAM) et ses sous paquetages Performance Analysis Modeling (PAM) et Scheduling Analysis Modeling (SAM). Il offre des analyses quantitatives de modèles, des ordonnancements et des analyses de performances ;

— *MARTE Annexes* : en annexe, Marte contient, entre autre, un langage textuel pour la spécification de valeurs (VSL en anglais Value Specification Language). Il contient aussi des mécanismes pour la modélisation de structures répétitives (RSM en anglais Repetitive Structure Modeling). La bibliothèque MARTE_Library définit des types de primitives. Ces primitives comprennent des opérations prédéfinies, couramment utilisées dans des systèmes temps-réel.

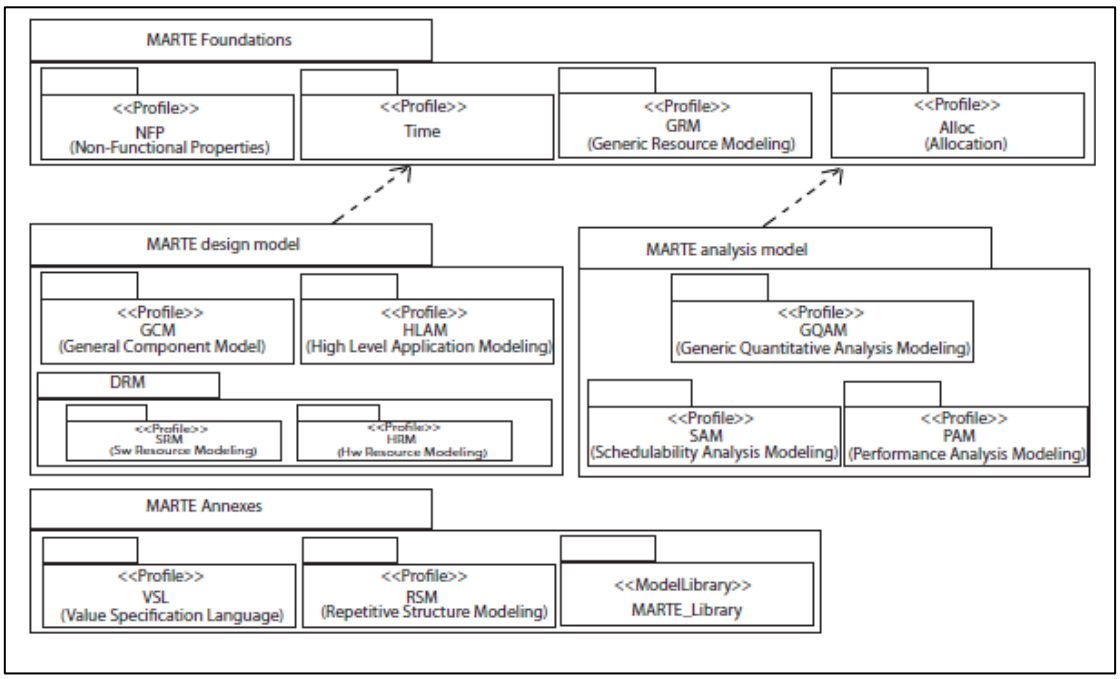


FIGURE 2.5– Structure de l'organisation du standard MARTE [Bou11]

Formalisme de représentation

En tant que profil UML, MARTE est basé sur la notion de stéréotype pour représenter les différents éléments d'un modèle. Parmi ses 14 sous profils le sous profil Time du paquetage MARTE Fondations est celui qui s'occupe de la modélisation de concepts de temps du système (durés, période, échéance, etc.). Il permet d'enrichir le modèle de base par des informations temporelles, valeur et contraintes temporelles précises permettant ainsi une interprétation temporelle de modèles UML. C'est donc ce sous profil qui nous intéresse dans cette thèse. La figure 2.5 décrit une interaction sous forme de diagramme de séquence annoté par des stéréotypes MARTE. Nous présentons sur le diagramme de séquence les diffèrent concepts utilisés pour spécifier le temps et les contraintes temporelles. Ces éléments sont définis dans le paquetage

SimpleTime de CommonBehaviors. TimeObservation est une référence à une instance de temps (observation temporelle), tant que DurationObservation est une référence à une durée d'exécution d'une action. TimeConstraints peut être sous forme de durée, ainsi que certains instants de temps. Le diagramme de séquence supporte les deux. DurationConstraint définit une contrainte qui fait référence à un intervalle de durée, qui définit la gamme entre deux durées. Une durée définit une spécification de valeur qui spécifie la distance temporelle entre deux instants de temps. Les contraintes temporelles sont exprimées entre une paire d'accolades. Les annotations en couleur ne font pas partie du modèle, elles sont utilisées pour spécifier les éléments du modèle.

D'après la figure 2.6, la séquence est activée par le message *start*. Le contrôleur reçoit ce message à l'instant *t0*. Cet instant est contraint de deux façons par la contrainte *constr1* donnée dans la partie supérieure du cadre et par la contrainte de *gigue* écrite à côté du message. Il en résulte que les réceptions de *start* sont périodiques de période *100 ms* avec une gigue inférieure à *5 ms*. La propagation du message acquière a une durée *d1* qui est inférieure ou égale à *1 ms*. La durée de traitement entre la réception de la demande d'acquisition et l'émission de la valeur acquise (message sendData) est contrainte à l'intervalle **[***d1..3*d1]* dont la valeur dépend de celle de *d1*. Les contraintes temporelles sont exprimées entre une paire d'accolades.

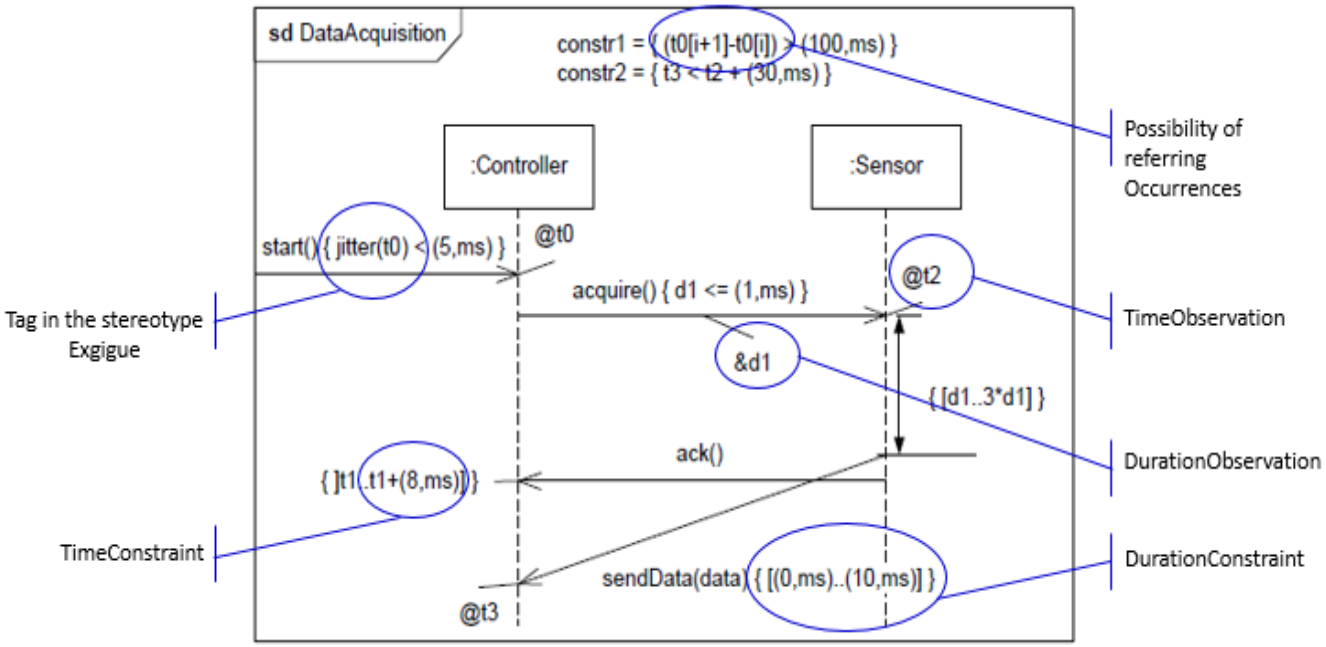


FIGURE 2.6– Diagramme de séquence annoté par le temps et les contraintes temporelles [And11]

2.4.4 SysML

SysML (Systems Modeling Language) [FMS06] est un langage de modélisation graphique développé par l'OMG en réponse à la demande de proposition d'UML pour l'ingénierie des systèmes. Il prend en charge la spécification, l'analyse, la conception, la vérification et la validation de systèmes(le matériel, les logiciels, les données, le personnel et les procédures). C'est un langage de modélisation visuelle qui fournit une sémantique et une notation. Ce n'est pas une méthode ou un outil, et donc est indépendant de toutes méthodologies et outils. SysML est basé sur UML

et remplace la modélisation de classes par des blocs pour un vocabulaire plus adapté à l'ingénierie des systèmes. Un bloc englobe tout concept logiciel, matériel, données, processus, et même gestion des personnes. SysML réutilise une partie d'UML2, et apporte également ses propres définitions. Il n'utilise donc pas les diagrammes d'UML2, mais propose un ensemble de diagrammes adaptés à l'ingénierie des systèmes (voir figure 2.7). Pour la modélisation d'un système, SysML propose un ensemble de diagrammes répartis en trois (3) catégories (vues du système) :

— *Les diagrammes structurels* : qui permettent de décrire les composants du produit et son environnement. Parmi ces derniers l'on a en particulier le diagramme paramétrique, qui est une extension SysML pour l'analyse de paramètres critiques du système ;

— *Les diagrammes dynamiques* : qui modélisent le comportement du système en réaction à son environnement ;

— *Le diagramme des exigences* : qui est une extension SysML.

L'ordre de présentation de ces diagrammes ne définit en aucun cas la méthodologie à suivre pour modéliser un système avec SysML.

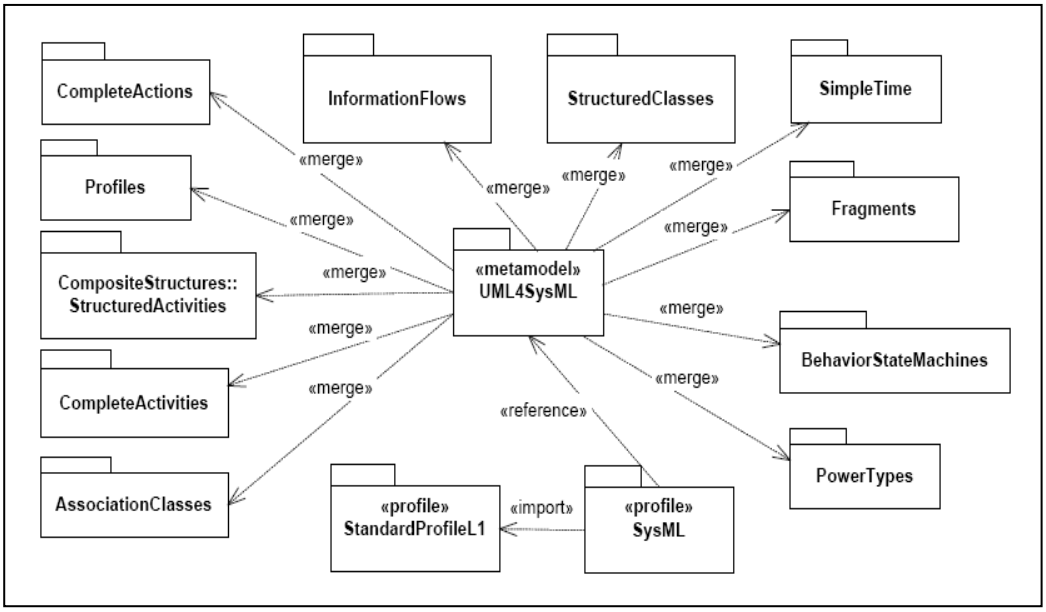


FIGURE 2.7– Architecture du SysML [Tea06]

Formalisme de représentation

Une des spécificités de SysML par rapport à UML étant son diagramme des exigences, nous présentons le formalisme de représentation utilisé par ce dernier. Les autres diagrammes de SysML peuvent être consultés dans [FMS06]. Dans le diagramme des exigences, on spécifie, hiérarchise et documente les exigences, c'est-à-dire les attentes portant sur le système ou sur son comportement. On va donc y mettre un texte qui définit toutes sortes de caractéristiques chiffrées, des choix technologiques imposés, le respect des normes, etc. La figure 2.8 donne la représentation

graphique d'une exigence dans SysML. Sur cette figure, on représente une exigence appelée "*Exigence*", qui a pour identifiant "01". L'attribut *text* donne une description de l'exigence.

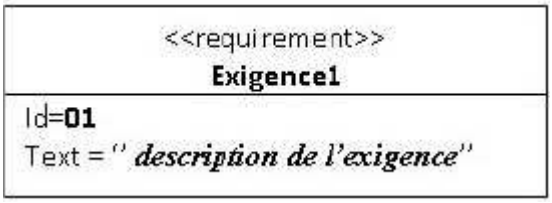


FIGURE 2.8– Notations graphiques d'une exigence SysML

2.4.5 AADL

AADL (Architecture Analysis and Design Language) [FLV06, FSW12] est un langage de description d'architecture développé depuis 2001 par Peter H. Feiler, Bruce A. Lewis et Steve Vestal. Il a été conçu pour la modélisation de STRE, et permet de décrire les composants matériels et logiciels d'une architecture ainsi que leurs interactions. AADL se base sur le langage MetaH. Il s'appuie sur le paradigme composant/connecteur/architecture, tout en intégrant de nouveaux aspects permettant un fort enrichissement de la description de l'architecture d'une application. Il définit l'architecture du système comme un ensemble de composants interconnectés. La modélisation de composants AADL consiste à décrire les interfaces, les implémentations et les propriétés des composants. AADL est extensible : il est possible d'utiliser d'autres langages pour définir des annexes AADL. De plus, le standard fournit un certain nombre d'annexes permettant la spécification du comportement d'une application, la modélisation de données, la modélisation d'erreurs et des directives pour la génération de code.

Formalisme de représentation

La figure 2.9 donne les symboles de la notation graphique de AADL. On y retrouve :

— Les composants logiciels : ligne (1) sur la figure ;

— Les composants de la plateforme d'exécution : ligne (2) ;

— Le composant système : ligne (3).

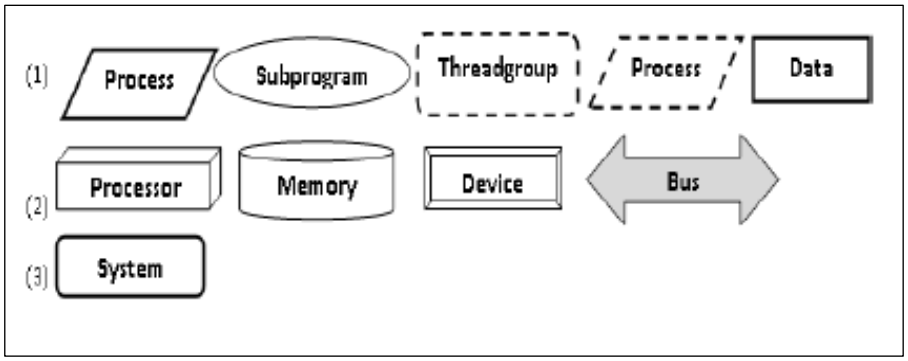


FIGURE 2.9– Notations graphiques des composants utilisés par AADL

Chacun des symboles présentés à la figure 2.8 possède une notation textuelle équivalente. La figure 2.10 présente par exemple un composant *device* nommé *brakePedal,* et l'équivalent textuel de sa définition.

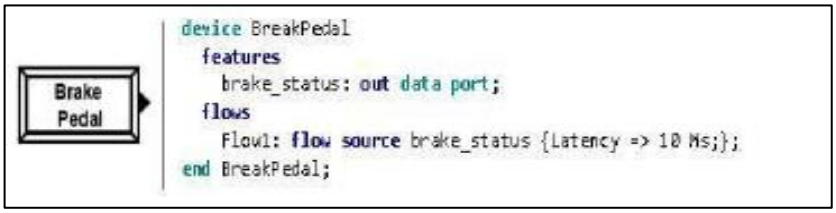


FIGURE 2.10– Un composant device AADL et son équivalent textuel

Sur le schéma de la figure 2.10, le petit triangle représente le port de donnée, qui est en fait l'interface du composant avec le monde extérieur. Dans cet exemple, le port de données est un port de sortie. Le nom de la variable de ce port est brake_status. Nous signalons également que ce dispositif fait partie d'une spécification de flux par le mot réservé flows, qu'il est une source de flux (flow source), et que le nom de flux est Flow1.Afin d'effectuer l'analyse de latence à travers un flux, la propriété définissant le temps de latence doit être ajouté à l'attribut de flux (par exemple, 10 ms). Pour un périphérique (composant), cette valeur représente le temps de latence entre le moment où la pédale est enfoncée et le moment où le signal associé est disponible sur le port de sortie.

2.5 Discussion

Les langages de modélisation ou les profils sont des outils offrant la possibilité aux concepteurs de formaliser leurs pensées et conceptualiser la réalité sous forme explicite pouvant être textuel ou graphique [BCW17]. Il existe dans littérature plusieurs travaux sur la comparaison de langages basés sur UML pour la modélisation des processus logiciels [Cle96, MT00, BJGB10, Shi13]. Les auteurs ont défini un ensemble de critères que doit satisfaire un processus de modélisation de systèmes tel que la richesse sémantique, la modularité, l'excitabilité des modèles, la conformité au standard UML et la formalité. Dans cette section, pour chaque profil, UML, UML-RT, MARTE, SysML et AADL, nous donnons : les facilités d'utilisation qu'il offre (prise en main, disponibilité des outils la supportant, concepts et notations offerts pour la représentation des systèmes) ; les moyens qu'elle fournit pour composer le système à partir des éléments structurels ; les éléments fournis pour les aspects temps-réel du système ; les mécanismes de sûreté de fonctionnement qu'elle offre. Ainsi, nous essayons de mettre en évidence les forces et les faiblesses de chaque profil. Une comparaison objective entre les différents profils de conception des systèmes temps-réel et embarqués, selon les critères de l'aspect temps définis dans [MNKT13] est résumée dans le tableau1.1.

2.5.1 UML

- **Utilisabilité**

UML offre un ensemble de notations graphiques auxquelles sont associés des textes. La génération du code peut être faite à partir de quelques diagrammes, tels que le diagramme de classes. UML est supporté par des dizaines d'outils. Ces d*erniers proposent des éditeurs graphiques pour la construction des diagrammes, mais offrent également pour la plupart, une fonctionnalité de génération de code à partir des diagrammes (comme par exemple le diagramme de classes). Certains de ces outils sont libres, comme Acceleo, ArgoUML, BOUML, Eclipse UML2 Tools, etc. et d'autres ne le sont pas, comme IBM Rational Rose, Entreprise Architect, Power AMC, etc.

- **Compositionalité**

UML dans sa version 2, propose le diagramme de structure composite qui permet de représenter un système par ses composants, et de donner la structure interne de chacun des composants. Ce modèle à composants permet d'obtenir de proche en proche l'architecture hiérarchique d'un système.

- **Aspect temps-réel**

UML à travers certains de ses diagrammes tels que le diagramme de séquences, d'activités, d'états transitions (évènements temporels) et de structures composites, permet tant bien que mal, de gérer les aspects relatifs au temps-réel. On peut faire des spécifications temporelles sur un diagramme de séquence, en précisant par exemple le temps écoulé entre deux messages envoyés. Les auteurs de [AdeSK05] proposent par exemple la synthèse d'une conception UML temps-réel à partir de diagrammes de séquences. La dernière version d'UML propose le diagramme de timing pour la gestion des aspects temporels d'un système.

- **Sûreté de fonctionnement**

Les notations offertes par UML ne sont pas formelles. Mais avec le langage OCL, il est possible de définir sur des diagrammes UML, des contraintes qui seront formellement vérifiées.

- **Forces et faiblesses**

UML est facile à prendre en main, et offre un large panel de diagrammes permettant de modéliser quasiment tous les aspects et vues d'un système. La disponibilité d'un grand nombre d'outils libres rend le langage accessible à tous les utilisateurs. Son succès a été fulgurant, et il est maintenant utilisé dans la grande majorité des entreprises développant des logiciels à des fins non seulement de documentation et d'analyse, mais de plus en plus à des fins productives de génération automatique ou semi-automatique de code, ou de tests. De plus, UML tient compte de nouvelles notations (graphiques) de stéréotypes étendus exige moins de compétence dans la conception des langages.

UML présente quelques limites : i) les études montrent que maintenir des diagrammes UML peut devenir un processus complexe ; ii) Bien que le langage OCL permette de définir des

contraintes sur des diagrammes UML, il y manque une définition plus formelle de la notion de temps et le degré de vérification formelle de ses diagrammes reste bas ; iii) les mêmes choses peuvent être modélisées sous plusieurs angles différents, et tous pourraient être corrects, ce qui signifie qu'il y manque un peu de consistance ; iv) Dans UML on se focalise plus sur la modélisation du système plutôt que sur le processus de développement du logiciel. Cela implique qu'UML devrait être utilisé dans une structure comme USDP (Unified Software Development Process) créé par l'OMG ou COMET (Concurrent Object Modeling archirtectural design method) [Gom06] ou tout autre processus de développement de logiciel. De plus, il y a un manque de standardisation entre les outils supportant UML et ses versions créant ainsi des confusions lors de la modélisation sur les notations à utiliser selon l'outil, car les constructeurs d'outils ont créé leurs propres notations qui diffèrent parfois de celles d'UML.

2.5.2 UML-RT

- **Utilisabilité**

L'UML-RT offre des notations graphiques et définit de nouveaux concepts par rapport à UML : capsule, port et protocole. Il existe quelques outils supportant UML-RT, comme par exemple RoseRT (RoseRT CASEtool). Il existe aussi pour UML-RT, un Framework d'exécution appelé TargetRTS (Système d'exécution cible) qui sert de machine virtuelle supportant le modèle d'exécution défini par le langage de modélisation UML-RT. Il tourne au-dessus d'un système d'exploitation temps-réel pour masquer les détails spécifiques au fournisseur de la plate-forme d'exécution, et présenter un ensemble uniforme d'API pour l'ingénieur concevant le système.

- **Compositionalité**

Les capsules peuvent être structurées de façon hiérarchique, en y joignant un certain nombre de *sous capsules* (subcapsules) qui peuvent elles-mêmes contenir des *sous capsules*.

- **Aspect temps-réel**

UML-RT enrichit le diagramme de séquence d'UML en y ajoutant des exigences temporelles (le temps d'exécution d'une séquence, le délai d'exécution, etc.), par exemple la vitesse à laquelle le régulateur de vitesse doit effectuer sa boucle de commande [FOW01]. Des machines à états simultanées sont modélisées sous forme de capsules distinctes, qui communiquent par passage de messages en mémoire tampon asynchrone.

- **Sûreté de fonctionnement**

UML-RT étant une extension d'UML, il supporte naturellement le langage OCL, qui permet de définir des contraintes vérifiables formellement sur un modèle.

- **Forces et faiblesses**

En plus des atouts d'UML, UML-RT apporte des éléments pour la modélisation des aspects temps-réel. Il ajoute notamment le concept de capsule qui est un composant pouvant interagir avec son environnement. UML-RT a néanmoins quelques limites, parmi lesquelles, il n'est pas largement utilisé et donc très peu supporté par des outils, Il manque une sémantique précise.

Un autre inconvénient est qu'il se restreint à des liens de communications binaires entre les composants.

2.5.3 MARTE

- **Utilisabilité**

Nous avons vu à la section précédente que MARTE offre des représentations graphiques et des notations textuelles. Ce langage semi-formel se veut suffisamment précis, pour pouvoir être exploité automatiquement par logiciel à des fins d'analyse, de génération de code, de test ou encore de documentation. MARTE est un standard assez jeune, et par conséquent, le nombre d'outils le supportant reste faible, mais est destiné à augmenter rapidement dans les années à venir. Il existe des modeleurs UML supportant actuellement le profil MARTE. Nous pouvons citer entre autres :

— *Papyrus* : il est libre et fait partie du projet Eclipse. Il fournit l'implémentation de référence de MARTE ;

— *RSA* (Rational Software Architect) : il est commercialisé par IBM. Il supporte MARTE par le biais d'un profil fourni sous licence libre Eclipse par Thales Researchand Technology ;

— *MagicDraw* commercialisé par NoMagic;

— *OpenEmbeDD*: c'est une plate-forme générique, reposant sur les technologies d'ingénierie dirigée par les modèles, et intégrant des outils d'aide à la conception d'applications temps-réel embarquées. Les technologies d'IDM utilisées dans OpenEmbeDD sont basées sur l'environnement Eclipse.

- **Compositionalité**

La conception de l'ensemble du système est modulaire et en fonction de ses besoins, l'utilisateur peut choisir tel ou tel sous-ensemble des 14 sous profils de MARTE.MARTE a un modèle à composant qui permet de définir les différents éléments qui constituent un système, leurs interactions (connexion entre composants) et de spécifier les types et les éléments qu'ils échangent. Ceci permet de définir une architecture (hiérarchique) du système.

- **Aspect temps-réel**

Le sous profil NFP et sa spécialisation Time sont une des contributions majeures de MARTE. Ils permettent la description des propriétés non fonctionnelles (qualitatives ou quantitatives) des systèmes, avec en particulier un raffinement pour la notation des propriétés temporelles des systèmes temps-réel. Le sous profil HLAM fournit des concepts de modélisation de haut niveau pour faire face aux modélisations des fonctionnalités des systèmes temps réel et embarqués [And07]. Il propose des concepts permettant la modélisation des caractéristiques aussi bien quantitatives (période, délais) que qualitatives (comportement, communication, concurrence) des systèmes temps-réel.

Le concept central proposé ici est celui d'unité temps-réel (RtUnit) qui encapsule une ressource d'exécution autonome, et est le concept central de la gestion de la concurrence. Il est crucial lors

du développement de systèmes temps-réel de pouvoir analyser les possibilités d'ordonnancement temps- réel de ces systèmes. Ainsi, le paquetage SAM fournit des annotations dédiées à cette analyse d'ordonnançabilité. Le paquetage PAM s'intéresse plus particulièrement à l'analyse des performances temporelles des systèmes temps-réel mous.

- **Sûreté de fonctionnement**

Les modèles produits avec MARTE sont en général annotés. En effet, MARTE permet de définir des propriétés non fonctionnelles (NFP en anglais Non Functional Properties) d'un système, rendant ainsi les modèles un peu plus formels. Ces NFP fournissent aux modèles les informations nécessaires pour réaliser des analyses spécifiques. Ainsi un modèle MARTE pourra supporter des analyses de performance et d'ordonnancement. Mais, MARTE définit aussi un Framework général pour l'analyse quantitative qui vise à raffiner ou spécialiser tout autre genre d'analyse. D'ailleurs MARTE offre des paquetages pour l'annotation (SAM) et l'analyse (PAM) des modèles.

- **Forces et faiblesses**

L'accroissement de la complexité des systèmes temps-réel et embarqués, a nécessité une montée dans le niveau d'abstraction des langages destinés à leur modélisation. MARTE est le premier exemple de langages permettant la modélisation de la plupart des aspects de ces systèmes avec une grande cohérence. MARTE est un profil UML, et donc en tant que tel bénéficie des avantages qu'offrent les profils UML. En plus des avantages communs aux profils UML, MARTE présente des avantages spécifiques. Entre autres :

— Il fournit une façon commune de modéliser les aspects matériels et logiciels d'un STRE, pour améliorer la communication entre développeurs ;

— Il permet une interopérabilité entre les outils de développement utilisés pour la spécification, la conception, la vérification et la génération du code, etc.

MARTE présente cependant quelques limites, non seulement il est contraint par le méta modèle UML, mais la notion de temps est reste ambigüe [Bou11] et chaque domaine peut avoir sa propre modélisation et interprétation du temps. En effet, on distingue principalement le temps physique qui correspond au temps utilisé dans les lois de la physique et de la mécanique, et le temps logique qui renvoie à une relation d'ordre sur les occurrences d'événements dans le système.

2.5.4 SysML

- **Utilisabilité**

Comme mentionné à la section précédente, SysML est basé sur UML, et réutilise plusieurs de ses diagrammes. Cela signifie que SysML offre une notation graphique. SysML supporte l'échange de données et de modèles via XMI (XML Meta data Interchange). Il a été intégré par de nombreux éditeurs d'outils commerciaux comme Sparx Systems Enterprise Architect (plugin SysML), IBM Rational Software Modeler,MagicDraw (plugin SysML requis), et des open source comme TopCased (environnement Eclipse), Papyrus, OpenEmbeDD et TTools

(outil d'analyse de modèles SysML qui allie simulation et vérification formelle, et apporte des réponses au besoin de détecter des erreurs de conception au plus tôt dans le cycle de vie du système).

- **Compositionalité**

SysML offre des diagrammes permettant de modéliser les différents composants d'un système par une approche de décomposition hiérarchique. Ainsi, le diagramme de bloc permet de représenter les systèmes en différents blocs qui le composent. La structure interne de chaque bloc peut ensuite être détaillée, de façon à présenter les sous composants qui entrent en jeux dans le fonctionnement du bloc. On obtient l'architecture hiérarchique du système.

- **Aspect temps-réel**

SysML étant une extension de UML, il conserve comme nous l'avons mentionné plus haut, certains éléments de UML, parmi lesquels les diagrammes de séquence, d'état-transition. Ces diagrammes permettent de définir des aspects temporels. A cela nous pouvons ajouter le diagramme paramétrique propre à SysML, et qui permet de définir le modèle paramétrique d'un système (différents paramètres du système, avec les règles et équations qui permettent de les avoir), ce modèle pouvant bien contenir des équations temporelles du système.

- **Sûreté de fonctionnement**

Le langage AVATAR [deSA11] peut être utilisé pour enrichir les diagrammes de conception SysML, notamment pour les doter d'une sémantique formelle. Ce langage est supporté par l'outil TTool, qui est interfacé avec l'outil UPPAAL [LP97], en vue de vérifier la logiqueet la temporalité de conceptions modélisées en SysML. Ainsi, un modèle SysML peut être utilisé pour des simulations et l'analyse des exigences.

- **Forces et faiblesses**

SysML offre l'avantage d'être accessible aux développeurs logiciels qui vont retrouver de nombreuses similitudes avec UML2. Il permet de produire des spécifications dans un langage unique pour des équipes hétérogènes, responsables de la réalisation des blocs matériels et logiciels. Les connaissances sont ainsi modélisées dans un référentiel unique qui améliore la communication entre les différentes équipes participantes, qui est accessible à tous, et permet la réutilisation des blocs réalisés. Une particularité de SysML est de regrouper les trois points de vue d'un système (structurel, fonctionnel et comportemental) au sein d'un unique modèle "multi-points de vue". L'utilisation d'un modèle unique présente deux avantages importants :

— Assurer la cohérence des données, car les règles de SysML donnent à chaque élément du modèle une définition unique, construite en rassemblant les informations issues de ses différentes représentations, et interdisent à celles-ci de se contre dire. Cela permet de réduire à la fois le risque d'erreurs ;

— Faciliter l'utilisation de la simulation, car un modèle SysML peut regrouper toutes les informations permettant de modéliser le système, de simuler son comportement, et de comparer les résultats aux exigences afin de valider (ou non) des solutions.

SysML présente aussi des faiblesses : La généricité de certains concepts, comme celui de bloc est source d'incompréhensions, ce concept nécessite donc d'être rapidement précisé lors de la modélisation. SysML peut être vu comme un langage de haut niveau permettant l'analyse et la conception de systèmes complexes jusqu'à une certaine granularité, mais qui n'est pas suffisant pour le développement complet d'un système. En effet, à un certain stade de la modélisation, on se trouve confronté à des diagrammes sur lesquels on a identifié les éléments matériels et logiciels, et qu'il faut spécifier complètement, ce que SysML ne permet pas de faire.

2.5.5 AADL

- **Utilisabilité**

AADL a une notation graphique et textuelle. Des outils permettent d'obtenir à partir de ces notations, une notation XML d'un modèle, mais aussi de passer d'un type de notation à un autre (du graphique au texte). Il existe de nombreux outils [Gee15, SEI] supportant le standard AADL à différents niveaux (modélisation, analyse, ordonnancement).Nous pouvons citer des outils libres comme : Osate, fournit le support AADL pour Eclipse. Osate1 ne supporte que la version1 de AADL et depuis août2011 Osate2 est disponible et supporte AADL version2. *Cheddar*, outil d'analyse d'ordonnancement temps-réel qui fonctionne typiquement avec les outils Stood et AADL Inspector. Ocarina, outil capable de charger un projet AADL version 1 ou version 2, d'exécuter des analyses et de générer du code à partir du modèle. Topcased, environnement pour Eclipse comprenant un générateur d'éditeur graphique. Aussi, il y a des outils commercialisés comme : Stood, outil de modélisation et de conception de logiciels embarqués, et Taste ensemble d'outils dédié au développement de systèmes temps-réel embarqués développé par l'Agence spatiale européenne et des partenaires de l'industrie spatiale.

- **Compositionalité**

Le langage AADL permet la spécification de sous composants contenus dans un composant. Ainsi, il est possible de décrire des compositions hiérarchiques de composants ou encore de modéliser un système hiérarchique complet, c'est-à-dire obtenir une vue complète (du point de vue structurel) de toutes les instances qui composent un système. Pour ce faire, AADL définit des composants et des interactions entre eux. Dans AADL, un composant est constitué de deux parties : une interface (component type) qui définit les services fournis et requis avec lesquels les autres composants peuvent interagir, et une (ou des) implémentation(s) (component implémentation) qui spécifie la structure interne du composant. Les composants AADL sont regroupés en trois (03) grandes familles :

— Les composants logiciels (software component) : Data (structures de données), Subprogram et Subprogram group (code de sous-programme et bibliothèque), Thread et Thread Group (unité d'exécution ordonnançable), Process (espace d'adressage virtuel) ;

— Les composants de plate-forme d'exécution (hardware component) : Processor (ordonnancer et exécuter un thread), Virtual Processor (machine virtuelle par exemple), Memory (modélise des composants de stockage), Bus et Virtual Bus (respectivement canaux de communication et protocoles de communication), Device (entité externe au système).

— Les composants systèmes : System, qui modélise un assemblage hiérarchique de composants logiciels et matériels. Les interactions entre composants sont réalisées par des *features*, ce sont des points de communication spécifiés dans leurs interfaces et par des *connexions*, spécifiées dans leurs implémentations. D'autres concepts sont disponibles dans [Las08] et [Ver06].

- **Aspect temps-réel**

AADL dans sa définition vise à développer l'architecture logicielle et matérielle de systèmes temps-réel, critiques ou répartis. Il permet donc la spécification de propriétés pour chaque type de composant. Le standard fournit un ensemble de propriétés (Compute_Execution_Time, Deadline, etc.) prédéfinies traitant des aspects temps-réel, de la concurrence, de la répartition, de l'intégrité, de la sécurité et de la performance. Chaque concepteur peut également définir ses propres propriétés. De plus, contrairement à la version 1 du langage où les sémantiques relatives au temps pour l'exécution de threads, la communication ou encore les changements de modes sont définis de manière globalement synchrone, donc exprimées avec un seul temps de référence (e.g : une seule horloge globale), AADL version 2 permet maintenant de définir différentes références temporelles (plusieurs horloges) dans le cas d'un système globalement asynchrone. La propriété Reference_Time permet de spécifier les différentes horloges pour les processeurs, périphériques, bus et mémoires.

- **Sûreté de fonctionnement**

AADL permet de définir de façon formelle des propriétés non fonctionnelles sur des modèles élaborés, ce qui permet à des outils d'analyse d'effectuer des analyses d'ordonnancement temps-réel, de performance, etc. Des outils existent pour la simulation des modèles AADL, pour une vérification formelle des propriétés du système. AADL assure la communication entre les composants en donnant la possibilité de définir le délai et d'analyser le taux d'envoi et de réception de messages dans un modèle.

- **Forces et faiblesses**

AADL est un langage de modélisation architecturale. La force du standard AADL est qu'il permet de couvrir à la fois la plupart des différentes aspects du système (matériels, réseaux, systèmes, intergiciels et applicatifs), et la plupart des étapes du cycle de développement (spécification, analyse, intégration). Historiquement issu du monde de l'avionique, il propose aujourd'hui une sémantique forte particulièrement adaptée à la description de systèmes temps-réel embarqués critiques. L'approche MDA et la sémantique forte facilitent également le développement d'outils d'analyse et de transformation du modèle représentant le système, ouvrant ainsi la voie vers une meilleure intégration des techniques de vérification formelle. Une caractéristique de la puissance de AADL est sa capacité à modéliser les composants matériels du système cible. Lier des composants logiciels à des composants matériels associés, permet au concepteur de préciser et d'évaluer les effets des interactions du système complet. L'une des principales faiblesses d'AADL est qu'il n'offre pas des éléments permettant de modéliser les horloges lors de la conception d'un système.

2.6 Choix d'un langage de modélisation

En conclusion de cette étude de l'existant, nous pouvons affirmer que le langage de modélisation choisi est le formalisme central autour duquel s'articulent les différentes activités du processus de conception. En effet, le choix judicieux du langage de modélisation adopté pour la spécification d'un système est d'une importance primordiale, car il peut avoir une influence directe sur les autres étapes de conception. Ce choix est souvent guidé par l'adéquation des propriétés et caractéristiques du langage avec les spécificités du système à concevoir. Dès lors, l'évaluation des différents langages doit donc être établie en fonction des aspects à modéliser en termes de sûreté de fonctionnement, de temps réel, ainsi que de distribution et de parallélisme. Ces notions, qui constituent quelques-unes des préoccupations parmi les plus importantes des concepteurs d'applications temps-réel ou embarqués, sont précisément des concepts fondamentaux qui forment le socle sémantique des langages de spécification.

Suivant notre contexte de travail, le critère temps-réel prends importance majore dans le choix du langage de modélisation à utiliser. Afin de savoir quelle est le langage ou le profil qui convient aux objectifs de cette thèse, le tableau 2.1 suivant résume les différentes propriétés prises en compte dans l'évaluation des aspects temps-réel gérés par les langages UML, UML-RT, MARTE, SysML et AADL, et donne à la dernière colonne le degré de couverture de ce critère pour chaque profil ou modèle.

		UML	**UML-RT**	**MARTE**	**SysML**	**AADL**
Propriétés	**PM**	0	0	0	0	1
	CM	1	1	1	1	1
	ED	0	1	1	0	1
	GE	1	1	1	1	1
	GS	1	1	1	0	1
	EM/S	0	0	1	0	1
	M	0	0	1	0	0
	R	0	0	1	0	1
	QT	1	1	1	1	1
	GH	0	0	1	0	0
	GHM	0	0	1	0	0
Notes temps réel		**4**	**5**	**10**	**3**	**8**

TABLEAU 2.1– Evaluation de la prise en compte des aspects temps-réel

PM : Priorité sur les messages,

CM : Communication Par Message,

ED : Echange de Données,

EM/S : Exclusion Mutuelle/sémaphore,

M : Moniteur,

RV : Rendez-vous,

GE : Gestion des Evénement, **QT** : Quantification du Temps,

GS : Gestion des Signaux, **GH** : Gestion d'Horloge,

GHM : Gestion d'Horloge Multiple.

La figure 2.11 présente le graphique construit à partir du tableau évaluant le critère des aspects temps-réel. D'après ce graphe on constate que MARTE a le plus grand degré de couverture de ce critère. Ainsi, MARTE est le meilleur profile à choisir selon ce critère.

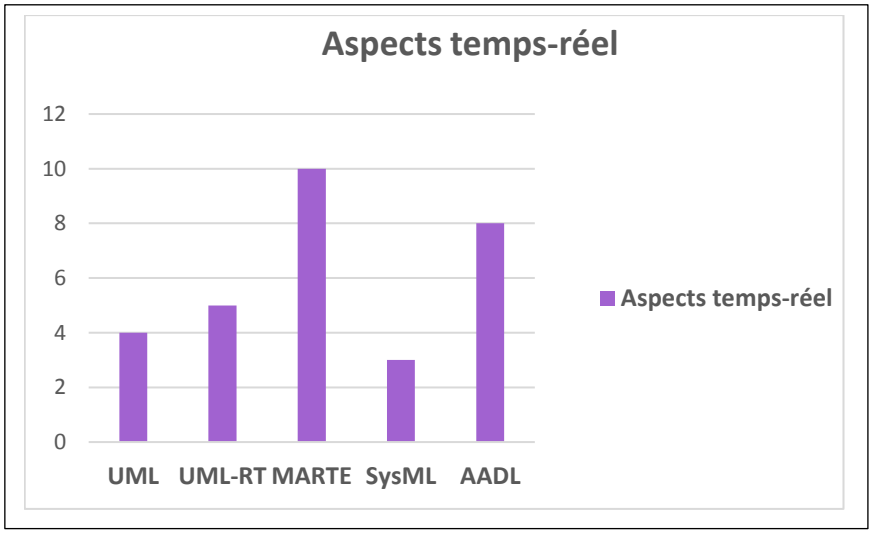


FIGURE 2.11– Graphique : Aspects temps-réel

2.7 Conclusion

Ce chapitre nous a permis d'établir un état de l'art sur les différentes techniques offertes par l'ingénierie dirigée par les modèles et qui peuvent être utilisées pour apporter des améliorations significatives dans le développement de systèmes temps-réel embarqués, en permettant de se concentrer sur l'élaboration de modèles plus abstraits que l'algorithmique ou la programmation. Il s'agit d'une forme d'ingénierie dans laquelle tout ou partie d'une application est générée à partir de modèles. Un système peut ainsi être décrit par différents modèles liés les uns aux autres.

Dans la deuxième partie, nous avons ainsi passé en revue les langages de modélisation dédiés au domaine des applications temps-réel et embarqués. Ceci nous a permis de comprendre pour chacun des langages ou profils, les principaux concepts de base offerts, les outils (modèles, diagrammes, etc.) et les notations graphiques et/ou textuelles utilisées pour gérer au mieux le développement des systèmes. Ces profils peuvent cependant présenter certaines limitations. D'après l'analyse que nous avons faite, nous pouvons constater que la plupart des profils manquent de support formel pour l'analyse, le raffinement et la validation. En effet, l'utilisation d'un profil pour un développement reste difficile sans au préalable définir des critères que l'utilisateur privilégie pour son choix. Ces derniers permettent de guider les concepteurs de ces systèmes, pour le choix de profils à utiliser, en fonction de leurs besoins et des aspects à modéliser.

3. Modèles formels de temps

Sommaire

Chapitre 3

Modèles formels de temps

3.1 Introduction

Les systèmes temps-réel et embarqués sont considérés souvent comme étant critiques. Ils doivent répondre à des contraintes temporelles strictes, en plus de la satisfaction de leurs besoins fonctionnels. Parmi ces systèmes, on trouve le système de freinage ABS embarqué dans une voiture, les moniteurs vitaux dans un hôpital, le système de pilotage automatique dans un avion, ou encore les systèmes de supervision dans une centrale nucléaire. Donc, une modélisation inappropriée peut causer des aux conséquences dangereuses sur les vies humaine, les matériels ou sur les financiers [Fle02 dysfonctionnements]. Les modèles devront, de ce fait, être vérifiés ensuite validés de sorte que les défauts de conception peuvent être détectés avant même qu'un prototype soit construit. C'est ici que les méthodes formelles trouvent tout leur intérêt et sont de plus en plus utilisées pour répondre aux exigences auxquelles sont soumis ce type de systèmes. Ces méthodes reposent sur l'utilisation de formalismes de spécification dotés des sémantiques rigoureuses et de techniques de vérification formelle.

Afin de modéliser les systèmes temps-réel embarqués, plusieurs modèles temporisés ont été proposés dans la littérature. On trouve les modèles de bas niveau, comme les modèles états-transitions, dotés d'un mécanisme décrivant les contraintes temporelles, et les modèles de haut niveau permettant de modéliser de manière concise les systèmes ayant des comportements assez complexes. Les langages formels sont maintenant largement utilisés pour la description des systèmes embarqués critiques. Les facteurs essentiels ayant motivé leur utilisation sont : (1) la description rigoureuse des systèmes à modéliser ; (2) la possibilité d'analyser, vérifier et valider ces systèmes, et éventuellement ; (3) la capacité de génération automatique de code, à partir de ces langages. Un autre facteur est l'existence des langages formels adaptés aux domaines d'application considérés

Ce chapitre introduit les modèles formels temporisés et/ou temporels les plus répondus parmi les plus connus, en particulier ceux considérés dans cette thèse. De nombreuses extensions à partir des modèles formels atemporels de base ont été proposées pour prendre en compte les caractéristiques exigées par les applications ou les systèmes temps réel ou embarqués. Nous décrivons deux catégories de formalismes une catégorie de haut niveau et une deuxième de bas niveau. Dans la première catégorie, nous présentons les extensions temporisés des algèbres des processus et des réseaux de Petri. Dans la deuxième catégorie, nous introduisons les systèmes de transitions temporisés étiquetées et les automates temporisés

3.2 Extensions temporisées des algèbres de processus

Les algèbres de processus sont un formalisme pour la modélisation, la conception et l'analyse de systèmes concurrents ou distribués. De manière générale, une algèbre de processus est dotée d'une syntaxe et d'une sémantique. La syntaxe comporte un ensemble restreint d'opérateurs (composition parallèle ou séquentielle, choix non déterministe, etc.) qui permettent la description de comportements complexes. La sémantique quant à elle est définie formellement par le biais d'une axiomatique ou d'une sémantique opérationnelle. Plusieurs algèbres de processus ont été proposés. Les plus connues sont CSP (Communicating Sequential Processes) qui a été présenté par Hoare dans [Hoa78, Hoa85], ACP (Algebra of Communicating Processes) [BK85] et CCS (Calculus of Communicating Systems) qui a été proposé par Milner [Mil82, Mil89]. D'autres algèbres ont vu les jours plus récemment tels que LOTOS [BB89] et π-calcul [MPW92]. LOTOS (Language Of Temporal Ordering Specifications) est un standard ISO utilisé pour la spécification formelle des protocoles dans les standards OSI [BB87, ISO88]. π-calcul est une continuation de CCS par Milner ayant pour objectif de pouvoir exprimer la mobilité des processus.

Toutes ces algèbres sont atemporelles, du fait qu'elles ne permettent que de raisonner sur l'ordre d'exécution des évènements et des étapes d'exécution. Dans le but de spécifier et vérifier les systèmes temps-réel, les algèbres de processus non temporisées ont été étendues par la notion de temps à travers l'ajout d'opérateurs temporels à l'ensemble d'opérateurs conventionnels. Par conséquent, plusieurs algèbres de processus temporisées ont vu le jour comme Timed CSP dans [RR88], Real time CCS [MT90], Real time ACP [BB91], Time LOTOS [LL98] et RT LOTOS (Real-Time LOTOS) [CDS93].

3.3 Extensions temporisées des Réseaux de Petri

L'idée fondamentale des réseaux de Petri (PNs) a été introduite par Carl Adam Petri en 1962 [Car62]. Ils représentent un outil mathématique puissant pour la modélisation, l'analyser et la vérification des systèmes. Le formalisme des réseaux de Petri, adapté à la prise en compte des problèmes de concurrence, de synchronisme et de parallélisme, constitue un excellent outil de spécification fonctionnelle d'un problème et de mise en évidence des contraintes. Ils sont utilisés dans de nombreux domaines, y compris dans les milieux industriels. Les applications sont diverses et couvrent les domaines des systèmes de production, de transports et de communication. Les réseaux de Petri, du fait de leur sémantique du parallélisme et de leur représentation graphique, apparaissent comme formalismes attractifs pour modéliser les comportements des systèmes, en particulièrement les systèmes temps-réel embarqués interagissant avec leur environnement. Dans cette section, nous présenterons dans un premier temps en section 3.3.1 le modèle des réseaux de Petri dans lequel le temps n'est pas encore représenté. Nous introduirons, ensuite deux extensions temporisées de ce modèle, les réseaux de Petri temporels et l'extension à durées d'actions (DTPNs), qui est utilisé au travail de cette thèse (en section 3.3.2), et les réseaux de Petri temporisés (en section 3.3.3).

Dans son modèle graphique, un réseau de Petri est constitué de places, de transitions et d'arcs orientés. Les arcs relient une transition à une place, ou inversement une place à une transition. Chaque place contient un nombre positif ou nul de jetons. La disposition des jetons dans les différentes places définit le marquage du système autrement dit l'état courant du système. Une transition est définie par un passage de jetons d'une place à une autre le long des arcs. Si chaque place d'entrée des arcs entrants dans une transition possède suffisamment de jetons, cette transition peut être tirée. Alors toutes les places reliées par arc sortant de cette transition reçoivent des jetons supplémentaires.

Formalisons maintenant la notion de réseau de Petri :

Définition 1. *Petri Net(PN). Un réseau de Petri est défini par un quintuplet (S, T, F, W, M_0) où :*

– S : l'ensemble des places ;

– T : l'ensemble des transitions ;

– $F \subset (S \times T) \cup (T \times S)$: l'ensemble des arcs ;

– $W : F \rightarrow N$: la valuation de chaque arc. Cette fonction associe à chaque arc de F un entier positif. Cette valeur indique combien de jetons seront consommés pour activer la transition en sortie de l'arc correspondant ou combien de jetons seront produits si la transition en entrée est tirée.

– $M_0 : S \rightarrow N$ est le marquage initial c'est-à-dire le nombre de jetons dans chaque place à l'état initial

Le modèle initial des réseaux de Petri est atemporel. Il se contente d'effectuer une modélisation et une vérification du comportement logique d'un système hors son contexte temporel. Afin de palier à cela, et pour pouvoir profiter de la puissance des réseaux de Petri, des extensions temporisées ont été proposées. Le but principal de telles extensions est de pouvoir effectuer une modélisation et une vérification du système sur le plan logique du séquencement des évènements, et surtout sur le plan temporel primordial dans le cadre de la représentation des systèmes temps-réel. De nombreux extensions de modèle ont été proposées, notamment avec les notions de couleur, temps ou encore hiérarchie [VdA92, Zub91, Jen13]. Les extensions qui permettent de modéliser le temps et des propriétés temps réel sont regroupées en deux familles : les réseaux de Petri temporels introduits par Merlin [Mer74], qui associent une durée aux transitions ou aux places, et les réseaux de Petri temporisés introduits par Ramchandani [Ram74], qui préfèrent la notion de délai (entre événements) à celle de durée (d'un état ou d'une action).

3.3.1 Réseaux de Petri temporels

La première extension temporisée de réseau de Petri est appelée réseaux de Petri temporels (TPN) et introduite dans [Mer74], consiste à ajouter une notion quantitative de temps dans le modèle par le biais des périodes d'activation des transitions.

Les réseaux de Petri temporel sont une extension des réseaux de Petri. Ils sont utilisés pour modéliser les systèmes où le temps joue un rôle critique. L'idée principale de ce type de réseaux est d'associer un intervalle temporel aux transitions (T-TPNs), aux places (P-TPNs), ou à des arcs (A-TPNs), [Mer74, Kha97, Wal83]. Concernant l'expressivité de T-TPN et de P-TPN, Kha et autres ont prouvé que ces deux modèles sont incomparables [KDC96]. A-TPNs et P-TPNs sont similaires, cependant, la seule différence concerne la sémantique forte de P-TPNs et la sémantique faible de A-TPNs [Boy01].

Un réseau de Petri temporel est défini comme étant un réseau de Petri où on associe à chaque transition deux dates : a et b ($0 \leq a \leq b$, b est éventuellement infini) [Ber01]. L'intervalle ainsi obtenu, $[a, b]$ appelé intervalle statique, est relatif à la durée où la transition t a été sensibilisée. Si t reste sensibilisée, sans interruption, après qu'elle ait été sensibilisée alors :

— a ($0 \leq a$) est le temps minimum qui doit s'écouler, à partir de l'instant où t a été sensibilisée, avant que t ne puisse être tirée.

— b ($a \leq b$) est la durée maximale pendant laquelle t peut rester sensibilisée sans être tirée.

Les dates a et b, pour une transition t sont associées à la date où t a été sensibilisée. Soit τ cette date. Alors t ne peut être tirée avant $\tau + a$ et doit être tirée avant $\tau + b$, à moins qu'elle ne soit désensibilisée avant avec une autre transition. La durée de tir d'une transition est supposée nulle [BBD09].

La figure 3.1 montre un exemple d'un T-TPN. Un intervalle de temps $[a, b]$ est associé à chaque transition t, dans lequel la transition t peut être sensibilisée. Ainsi, $a(t)= \{1\}$ et $b(t)= \{2\}$ pour $t \in \{t_1, t_3, t_5, t_6\}$, $a(t_2)= \{0\}$, $b(t_2)= \{3\}$ et $a(t_4)= b(t_4)= \{1\}$. Une transition t_i de T-TPN ne peut être tirée que si elle est sensibilisée de façon continue pendant a unités de temps. Si une transition est sensibilisée continuellement au moins b unités de temps, elle doit être tirée immédiatement car elle a atteint son délai maximum de sensibilisation, sauf si elle est désactivée par le franchissement d'une autre transition.

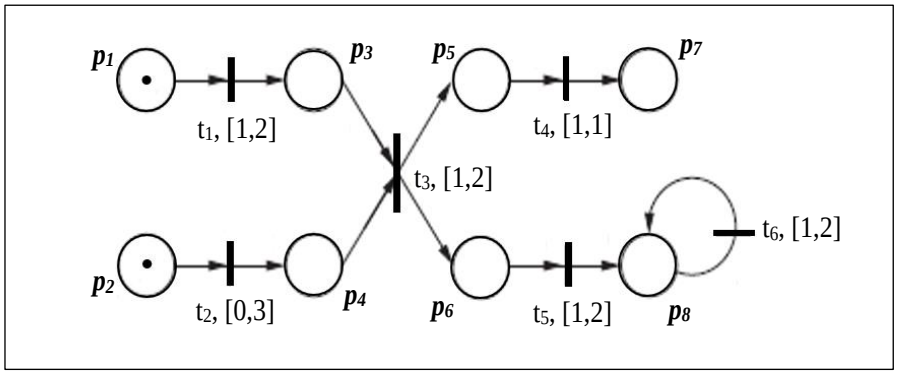


FIGURE 3.1–Un réseau de Petri T-temporel

Maintenant nous donnons une définition des TPNs. Nous noterons $\mathbb{I} \subseteq R_{\geq}$ l'ensemble des intervalles réels non vides avec des bornes rationnelles non négatives.

Définition 2. *Time Petri Nets(TPN). Un réseau de Petri temporel A est un sextuplet : A= (P, T, F, W, M_0, I_S)* où :

(P, T, F, W, M_0): est un réseau de Petri ;

I_S : T $\rightarrow \mathbb{I}$: est une fonction appelée intervalle statique.

Dans les TPNs, le franchissement des transitions est de durée nulle [Boy01]. Ils expriment nativement les spécifications dans le temps. En expliquant les débuts et les fins des actions avec la spécification de la progression dans le temps, ils peuvent également exprimer des spécifications en termes de durée. Cependant, cette manière de modéliser la durée d'action présente de nombreux inconvénients. Tout d'abord, la taille de la structure sémantique associée est augmentée. Ce phénomène est connu sous le nom de problème d'explosion combinatoire dans l'espace d'états. Deuxièmement, la spécification obtenue empêche structurellement l'énoncé du système à spécifier. Troisièmement, la sémantique sous-jacente, généralement la sémantique d'entrelacement, suppose une atomicité structurelle et temporelle des actions, c'est-à-dire que les actions sont indivisibles et ont une durée nulle. Ainsi, cette sémantique donne des résumés de l'exécution parallèle d'actions. Pour remédier à l'atomicité d'actions dans un contexte temps-réel et au problème d'expression naturelle du parallélisme d'actions, une sous-classe des TPN appelée réseaux de Petri temporels avec durées d'actions (DTPNs) a été proposé dans [BSB+13].

Dans les DTPNs, la sensibilisation de transitions est liée à un intervalle de temps et les transitions représentent des actions ayant une durée explicite. Le DTPN a pris en considération à la fois les contraintes temporelles et les durées sous une sémantique de vrai-parallélisme, automates à action temporelle, dans le but de mieux exprimer les comportements concurrents et parallèles des systèmes temps réel. Dans ce qui suit, nous présentons en détaille les TPNs avec durées d'actions.

Réseaux de Petri temporels avec durées d'actions

Le réseau de Petri temporel avec durée d'action est une extension de réseau de Petri temporel. Il vise à exprimer la vraie concurrence de manière naturelle sans diviser les actions dans leurs évènements de début et de fin. Dans ce modèle deux annotations sont associées à chaque transition, à savoir sa contrainte de temps qui restreint la date à laquelle elle peut être sensibilisée et la durée de son action correspondante. Par conséquence, Les réseaux de Petri temporels avec durés d'actions peuvent être considérés comme une généralisation de Merlin TPNs [Mer74], T-TdPN [Ram74] et P-TdPN [Sif77]. L'idée de base de DTPN est d'associer deux dates *min* et *max* à chaque transition qui définissent son intervalle de tir (intervalle temporel). Bien que le franchissement d'une transition soit instantané, la durée d'exécution de l'action associée à cette transition peut avoir une durée non nulle. Par exemple, soit *t* une transition associée à l'action qui a une durée *d*. Si θ est la date d'activation de la transition alors le tir de *t* sera dans l'intervalle de temps $[\theta + min, \theta + max]$. Le tir de *t* marque le début de l'exécution de l'action associée.

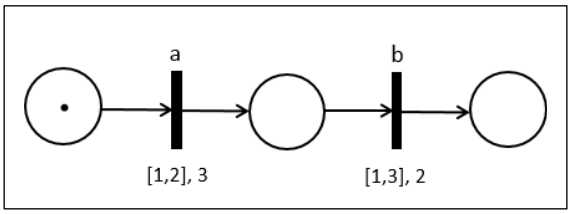


FIGURE 3.2– Un DTPN

La figure 3.2 montre un exemple de réseau de Petri temporel avec durées d'actions. Une place d'un DTPN correspond à deux ensembles : un ensemble de *jetons disponibles* ou de *jetons libres* et un ensemble de *jetons indisponibles* ou de jetons *liés*. Les jetons indisponibles, placés sur le côté droit d'une place, sont liés au franchissement des transitions associées aux actions en cours d'exécution. Dans un DTPN, un jeton indisponible devient disponible si la fin de l'exécution de l'action associée à la transition qui a produit ce jeton est atteinte. Un jeton dans la place *p* au moment ϑ devient *disponible* (sur le côté gauche de *p*) au temps $\vartheta + d$. Ainsi, le jeton est lié au franchissement de la transition pendant l'intervalle de temps [ϑ, ϑ +d [et il devient libre au temps $\vartheta + d$.

Dans la figure 3.3. (a), le jeton dans la place p_1 n'est pas *lié* à aucune transition. Ce jeton est appelé libre. Dans le cas où la transition est franchie, nous pouvons déduire que l'action associée au franchissement de t a commencé son exécution. Ceci est marqué par la présence du jeton dans la place $p2$ (Figure 3.3. (b)). Ainsi, le jeton dans place $p2$ est lié au tir de t, mais après la fin de l'action, c'est-à-dire après 3 unités de temps, ce jeton deviendra libre (Figure 3.3. (c)). Dans une place, l'ensemble de jetons libres sera dénoté par *FT*, tandis que l'ensemble de jetons liés sera dénoté par *BT*.

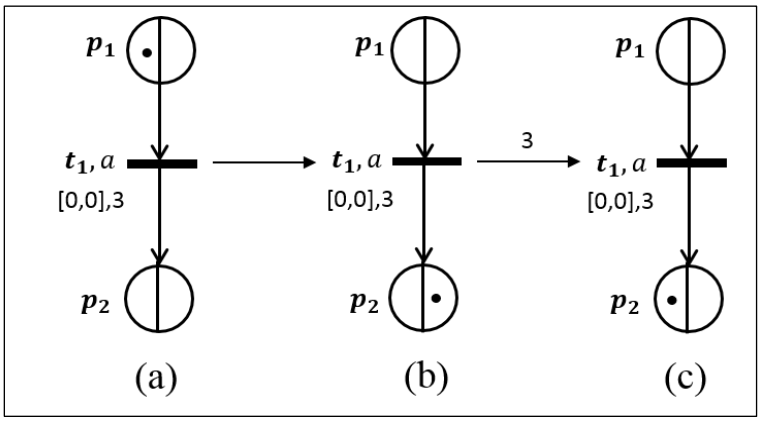


FIGURE 3.3–DTPN marqué

Définition 3. *Time Petri Nets with Action Duration (DTPN). Soit $\mathbb{T}$ un domaine temporel non négatif (comme $\mathbb{Q}+$ ou $\mathbb{R}+$) et Act un ensemble fini d'actions, c'est-à-dire un alphabet. Un réseau de Petri temporel avec durée d'action (DTPN) sur $\mathbb{T}$ et de support Act est un tuple $N = (P, T, B, F, \lambda, SI, \Gamma)$ tel que :*

- *$Q = (P, T, F, B)$ est réseau de Petri où : P est ensemble de places, T est ensemble de transitions tel que $P \cap T = \emptyset$;*
- *$B : P \rightarrow T$ est une fonction d'incidence en arrière telle que $B (p_i ; tj)$ représente un arc de p_i à tj ;*
- *$F : P \rightarrow T$ est une fonction d'incidence en avant telle que $F (p_i ; tj)$ représente un arc de tj à p_i ;*
- *$\lambda: T \rightarrow Act \cup \{\infty\}$ est une fonction d'étiquetage d'un DTPN. Si $\lambda (t) \in Act$ alors t est appelée observable ou externe ;*
- *$SI: T \rightarrow \mathbb{T} \times \mathbb{T} \cup \{\infty\}$ est une fonction qui associée à chaque transition un intervalle statique de franchissement ;*
- *$\Gamma: Act \rightarrow D$ est une fonction qui associée à chaque action sa durée statique.*

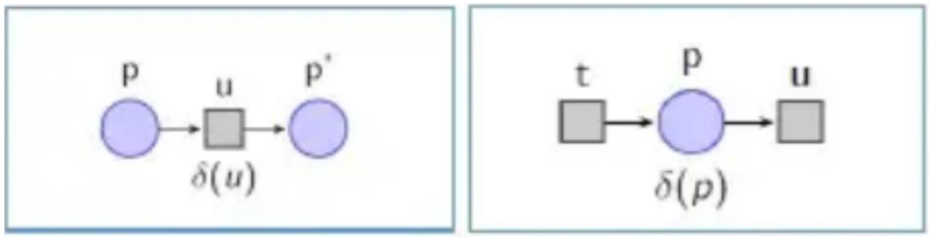
I est l'ensemble de tous les intervalles d'un DTPN tels que I (t)= [min, max] est l'intervalle associée à la transition t. On note $\downarrow I(t) = min$ et $\uparrow I(t) = max$ deux fonctions qui donnent respectivement les bornes inférieure et supérieure d'un intervalle.

Comme couramment utilisé dans la littérature, nous écrivons °t (resp t°) pour désigner l'ensemble des placets tels que $°t = \{p \in P/ B (p, t) > 0\}$ (resp. $t° = \{p \in P/ F (p, t) > 0\}$), °p et (resp. p°) pour représenter l'ensemble des transitions telles que $°p = \{t \in T/ B (p, t) > 0\}$ (resp. $p° = \{t \in T/ F (p, t) > 0\}$).

3.3.2 Réseaux de Petri temporisés

Le réseau de Petri temporisé (Timed Petri Nets, TdPN) fut la première extension temporelle de réseaux de Petri. Cet outil permet de représenter les mécanismes temporels associés à un processus avec une logique de description des temps nécessaires aux opérations. Un réseau de Petri temporisé est performant pour la spécification et la validation des systèmes comprenant des temps minimum. Par exemple, le modèle T-temporisé est surtout utilisé dans le cadre de l'évaluation de performances temporelles d'un système, il s'agit d'évaluer le temps d'exécution d'une série d'opérations [Ram74, Sif77].

TdPNs sont une sous-classe des réseaux de Petri temporels. Il y a deux (02) modèles de réseau de Petri temporisé : les réseaux de Petri T- temporisé (T-TdPN) et les réseaux de Petri P-temporisé (P-TdPN). Dans les T-TdPN, un paramètre temporel est associé à chaque transition. La sémantique de ce paramètre correspond à la durée de tir de cette transition (FIGURE 3.4.(a). Dans les P-TdPN, un paramètre temporel est associé chaque place dont la sémantique correspond au temps de séjour minimal d'un jeton dans une place (temps d'indisponibilité), FIGURE 3.4.(b). Les deux sous-classes correspondantes à savoir, T-TdPN et P-TdPN sont expressivement équivalentes [Sif79, Sif80, Her89] et sont une sous-classe de TPN. Il est possible de transformer un réseau de Petri P- temporisé en réseau de Petri T- temporisé et inversement.



(a) temps associé à une transition (b) temps associé à une place

FIGURE 3.4– Exemple de modèle T-temporisé et de modèle P-temporisé

Dans un réseau de Petri temporisé, le temps représentant une durée opératoire est associé à une place ou à une transition. L'obligation de franchissement n'existant pas dans les règles d'évolution, seul un temps minimum de séjour d'un jeton dans une place est spécifié (durée minimale d'une opération). Dans certains cas il faut ajouter des interprétations complémentaires pour obtenir une modélisation correcte avec les réseaux de Petri temporisés. Ils sont pas capables de modéliser les systèmes pour lesquels les opérations de traitement dont la durée doit être comprise entre un minimum et un maximum. C'est le cas du traitement de surface par exemple. Il est évident que le non-respect de ces contraintes entraîne une altération de la qualité du produit et son rejet au niveau du contrôle.

3.4 Systèmes de transitions étiquetées temporisés

Le comportement d'un système temporisé peut être représenté par un système de transitions étiquetées (STE). Ce dernier représente l'ensemble des états possibles du système ainsi que les transitions entre ces mêmes états. Chaque transition est étiquetée par un label, noté ℓ. Cette dernière représente une occurrence d'action ou une progression du temps.

3.4.1 Notions préliminaires

Dans cette première section, nous introduisons quelques notions fondamentales afin de définir formellement les automates temporisés.

Soit $\mathbb{T}$ un domaine de temps tel que $\mathbb{Q}+$ ou $\mathbb{R}+$, l'ensemble des nombres rationnels positifs ou des nombres réels positifs, selon le fait que le temps soit dense ou discret. Dans [NSY93] une proposition d'un modèle unifié du domaine du temps sous la forme d'un monoïde commutatif ($\mathbb{T}$, +) où l'élément neutre est noté (0) vérifiant les propriétés suivantes :

- $\forall t, t' \in \mathbb{T}$ et $t + t' = t \Leftrightarrow t' = 0$,
- La relation $\leq$ définie par $\{t \leq t'$ si $\exists\ t'' \in \mathbb{T}$ tel que $t = t' + t''\}$ est un ordre total sur $\mathbb{T}$,
- Les éléments de $\mathbb{T}$ représentent des dates et la différence entre deux dates corresponde à une

durée également dans $\mathbb{T}$,

- Une séquence temporelle sur $\mathbb{T}$ est une séquence croissante $(t_i)_{i \geq 1} \in \mathbb{T}^{\infty}$.

Nous considérons $\mathcal{H}=\{x_1, ..., x_n\}$ un ensemble de n horloges de valeurs non négatives dans un domaine temporel $\mathbb{T}$. Une valuation ou interprétation v sur $\mathcal{H}$ est une fonction qui associe à toute horloge $x \in \mathcal{H}$ une valeur dans $\mathbb{T}$, l'ensemble de toutes les valuations de $\mathcal{H}$ est noté $\Xi(\mathcal{H})$. Soit H l'ensemble de contraintes temporelles sur $\mathcal{H}$. Cet ensemble est défini par la syntaxe $\gamma ::= \{x \sim t,\ x - y \sim\ t,\ \gamma \wedge\}$ où x, y sont des horloges de $\mathcal{H}$, $\sim \in \{=, <, >, \leq, \geq\}$ et $t \in \mathbb{T}$. Nous considérons aussi un sous-ensemble propre de ces contraintes d'horloges, cet ensemble conserve les contraintes qui comparent une horloge et une constante, mais n'autorise plus de comparaisons entre les horloges. Ces dernières contraintes sont appelées diagonales. L'ensemble des contraintes d'horloges non diagonales peut être décrit par la grammaire suivante : $\gamma ::= \{x \sim t, \gamma \wedge\}$.

La relation de satisfaction " $\vDash$ " sur les contraintes temporelles est définie sur l'ensemble des valuations de $\mathcal{H}$ par : $v \vDash x \sim t \Leftrightarrow v(x) \sim t$ tel que $v \in \Xi(\mathcal{H})$. Une valuation v sur $\mathcal{H}$ satisfait une contrainte temporelle γ sur $\mathcal{H}$ ssi est vérifiée lors de la valuation des horloges par v. Pour $\mathcal{T} \subseteq \mathcal{H}$, $v\,[\mathcal{T} \mapsto 0]$ v désigne la valuation sur $\mathcal{H}$ qui affecte la valeur 0 à toute horloge $x \in T$.

Chaque transition t est associée une contrainte temporelle (ou *garde*) sur la valeur des horloges décrivant quand la transition t peut être exécutée, et un ensemble d'horloges qui doivent être remises à zéro lors du franchissement de la transition t. Un état peut contenir un *invariant* qui peut *restreindre* le temps d'attente dans l'état. Un invariant est donné sous la forme d'une contrainte d'horloges et devrait être satisfait pour que le système puisse demeurer dans l'état correspondant.

3.4.2 Systèmes de transitions étiquetées

Avant de présenter les systèmes de transitions temporisés, le modèle le plus général pour décrire les systèmes temps-réel, nous présentons d'abord dans cette section, les systèmes de transitions étiquetés, dans lesquels le temps n'est pas encore représenté.

Définition 4. *Labelled Transition System (LTS). Un système de transitions étiquetées* [Arn92, Arn94, BT00] *est un quadruplet $(Q, q_0, \Sigma, \rightarrow)$ tel que :*

- *Q est un ensemble dénombrable d'états ;*
- *$q_0 \rightarrow Q$ est l'état initial ;*
- *Σ est un alphabet dénombrable d'actions et*
- *la relation de transition $\rightarrow$ est un sous-ensemble de $Q \times \Sigma \times Q$. Un élément de $\rightarrow$ est appelé transition. Chaque transition est étiquetée par $a \in \Sigma$ et représente une occurrence de cette action. Une action d'un LTS peut prendre plusieurs formes : une entrée, une sortie, un appel de méthode, etc. Une transition $t= (q,a, q')$ sera représentée simplement par $q \xrightarrow{a} q'$. Une exécution d'un STE S est une séquence (finie) de transitions telle que : pour tout indice $i \geq 1$, il existe une transition $t= (q_i, a_i, q_{i+1})$ dans S.*

3.4.3 Systèmes de transitions temporisés

Le besoin de spécifier des systèmes temps-réel a fait naître l'idée d'incorporer le temps dans les systèmes de transitions étiquetés, d'où le nom des systèmes de transitions temporisés. Les transitions dénotent cette fois-ci soit une exécution d'une action discrète soit un écoulement du temps. Nous définissons les systèmes de transitions temporisés comme une sous-classe des systèmes de transitions étiquetées dans laquelle certaines transitions représentent l'écoulement quantitatif de temps.

Définition 5. *Timed Transition System (TTS). Un système de transitions temporisés est un système de transitions* $\mathcal{T}=(E, e_0, \rightarrow)$ *où :*

- *E est l'ensemble des états ;*
- $e_0 \in E$ *est l'état initial, et*
- *la relation de transition* $\rightarrow$ *est composée de transitions de délai* $e \xrightarrow{d} e'$ *avec* $d \in \mathbb{T}$*, et de transitions discrètes* $e \xrightarrow{a} e'$ *avec* $a \in \sum$*. De plus, la relation de transition* $\rightarrow$ *doit vérifier les quatre propriétés suivantes :*
- *Déterminisme temporel : pour tous les états* e, e', e'' *de E et pour tout* $d \in \mathbb{T}$*, si* $e \xrightarrow{d} e'$ *et* $e \xrightarrow{d} e''$ *alors* $e' = e''$ *;*
- *Zéro-délai : pour tous les états e,* e' *de E,* $e \xrightarrow{0} e'$ *si et seulement si* $e = e'$ *;*
- *Additivité : pour tous les états* e, e', e'' *de E et pour tous d,* $d' \in \mathbb{T}$*, si* $e \xrightarrow{d} e'$ *et* $e' \xrightarrow{d'} e''$ *alors* $e \xrightarrow{d+d'} e''$ *;*
- *Continuité : pour tous les états e,* e' *de E et pour tout* $d \in \mathbb{T}$*, si* $e \xrightarrow{d} e'$ *alors pour tous* $d', d'' \in \mathbb{T}$ *tels que* $d = d' + d''$ *il existe* e'' *tels que* $e \xrightarrow{d'} e' \xrightarrow{d''} e''$*.*

3.5 Automates temporisés

Les automates sont des formalismes très connus dans le domaine de la spécification formelle de systèmes. Un automate se compose d'un ensemble d'états ou nœuds, un ensemble d'actions et un ensemble de transitions étiquetées entre les états. Les actions sont modélisées par des étiquettes et une transition étiquetée modélise ainsi l'exécution de l'action lors du franchissement de celle-ci. Il existe plusieurs types d'automates comme les automates de Büchi [Ric62] les automates temporisés sont le type d'automate qui nous intéresse le plus.

Les automates temporisés sont introduits pour la première fois par Alur et Dill en 1994 [AD90, AD94] afin de spécifier et vérifier le comportement des systèmes contraints par le temps. Il s'agit d'automates classiques munis d'horloges qui évoluent au cours du temps, toutes à la même vitesse. Des conditions sur la valeur de ces horloges permettent d'intégrer des contraintes temporelles dans les états et les transitions de ces modèles. Il existe deux types d'évolutions possibles pour un tel formalisme : Une évolution du temps, exprimée par la progression des valeurs des horloges au sein d'un état ou localité de l'automate et une évolution entre les états de l'automate, exprimée par des transitions atomiques, permettant de capturer l'exécution des actions.

Une telle transition est conditionnée par la satisfaction d'une contrainte (garde), en fonction des valeurs actuelles des horloges.

Dans ce qui suit, nous commençons d'abord par une définition de l'automate temporisé. Ensuit nous introduisons les formalismes qui nous seront utiles et nécessaires lors de l'utilisation des automates temporisés. Nous présentons premièrement le modèle des automates temporisés classique avec quelques notions qui lui sont dévolues. Deuxièmes, nous détaillons les automates de régions et les automates temporisés avec durées d'actions qui sont deux extensions intéressantes des automates temporisés.

Définition 6. *Timed Automata (TA). Un automate temporisé est défini par A=(S, S_0,X, Σ, T), tels que :*

- *S : est un ensemble fini d'états ;*
- *$S_0 \subseteq S$: est l'ensemble d'états initiaux ;*
- *X : est un ensemble fini d'horloges (à valeurs réelles positives) ;*
- *$\sum$: est un ensemble fini d'actions ;*
- *$T \subseteq S \times C(X) \times \sum \times 2^{X \times S}$ est un ensemble fini de transitions tel que C(X) est l'ensemble des contraintes d'horloges.*

$t=\langle q, g, a, r, q' \rangle \in T$ représente une transition de q vers q', g est la garde associée à t, a $\in \sum$ est l'étiquette de la transition t, et λ est l'ensemble d'horloges réinitialisées à zéro pour l'action *a. On note $q \xrightarrow{g,a,r} q'$ une telle transition ;*

Une exécution d'un automate temporisé *A* est un chemin fini (ou infini) de transitions consécutives. Une exécution finie ou infinie est uniformément noté par $\rho = (q_i)$, $i \in I$.

— Une exécution finie d'un TA est un chemin fini $\rho = (q_i)$, $i \in \{0,..., m\}$ de TA dénoté par $\rho = q_0 \rightarrow q_1 \rightarrow \cdots \rightarrow q_m$. Nous pouvons écrire aussi $\rho = q_0 \rightsquigarrow q_m$.

— Une exécution infinie de *A* est un chemin infini $\rho = (q_k)$, $k \in \mathbb{N}$ de *A* dénoté par $\rho = q0 \rightarrow q1 \rightarrow \cdots \rightarrow qi \ldots$

Remarque

Quelques définitions ajoutent un ensemble d'états finaux et un ensemble d'états répétés comme dans le cas des automates temporisés de BüCHI (TBA's) [Ric62].

Un exemple d'automate temporisé est donné dans la figure 3.4. Cet automate décrit une minuterie : on part de l'état initiale OFF. A tout moment, on peut appuyer sur le bouton (action b), et p.asser à l'état ON avec l'horloge x remise à zéro. Le temps peut s'écouler tant que la valeur de l'horloge est inférieure à 10. A tout moment, une pression sur le bouton permet de remettre l'horloge x à zéro. Lorsque x = 10, la transition i retourne l'automate à l'état OFF.

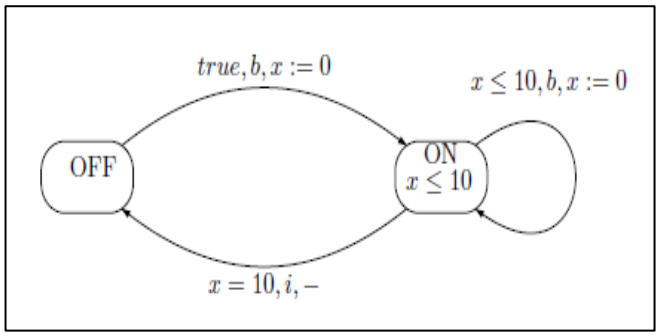


FIGURE 3.5– Un automate temporisé (TA)

3.5.1 Automates de régions

Les automates des régions sont des automates finis, particuliers qui imitent le fonctionnement infini d'un automate temporisé. Aussi, l'automate de régions est très utilisé car il permet de vérifier toutes les séquences possibles d'un automate temporisé. Une séquence a un nombre infini d'états du fait que le temps est considéré comme dense. L'automate des régions sert à discrétiser ce temps en introduisant la notion de région temporelle. Dans un premier temps nous définissons cette notion puis nous donnons la construction de cet automate. La définition classique et la formalisation des automates de régions de cette section sont tirées de [AD94, Bou02].

Région d'horloges

Une région est un ensemble de valuations d'un ensemble fini d'horloges, tel que deux valuations d'une même région, permettent (exactement) les mêmes transitions d'un TA d'être franchissables. C'est pour cela que l'ensemble des régions standards d'un TA a été défini. Pour $t \in \mathbb{R}^+$, on note $\lfloor t \rfloor$ sa partie entière et *frac(t)* sa partie fractionnaire. Pour $x \in X$, soit C_x la plus grande constante comparée avec x dans l'ensemble des gardes de T.

Définition 7. *Regions. Soit A=(S, S_0, X, Σ, T) un automate temporisé, l'ensemble des régions standards de A est l'ensemble des classes de la relation d'équivalence* "≡", *définie sur les valuations d'horloges comme suit :* $v \equiv v'$ *si* $\forall (x, y) \in X$:

- $v(x) > C_x \Leftrightarrow v'(x) > C_x$;
- $v(x) \leq Cx) \Rightarrow ((\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor)$ *et (frac (v(x)) = frac (v'(x)))) ;*
- $(v(x) \leq Cx$ *et* $v(y) \leq Cy) \Rightarrow ((frac\ (v'(x)) \leq frac\ (v(y))) \Leftrightarrow (frac\ (v'(x)) \leq frac\ (v(y))))$.

Étant donné r un ensemble de valuations, on note $r[X' \leftarrow 0] \subseteq \mathbb{R}_+^X$ l'ensemble des valuations v $[X'_0 \leftarrow 0]$ *tel que* $v \in r$.

Un ensemble de régions standards R dépend des constantes maximales C_x ainsi que du nombre d'horloges. L'utilisation de la constante maximale permet d'avoir un ensemble fini de régions standards. Ces derniers vérifient les trois propriétés suivantes :

- Toutes les valuations d'une région vérifient exactement les mêmes gardes,

- Toutes les valuations d'une région atteignent la même région en laissant s'écouler le temps,

- Toutes les valuations d'une région se projettent sur la même région (par toutes les projections [X′0]).

Définition 8. *Successeur d'une région. Soient deux régions r et r′, r′ est un successeur temporel de r si seulement si* : $\forall v \in r, \exists t \in \mathbb{R}^+$ *tel que* $v+t \in r'$

La relation d'équivalence "≡" est étendue aux configurations de TA comme suit :

q = (*s*, *v*) ≡ q′ = (s′, *v*′) si et seulement si, s = s′ et $v \equiv v'$.

Automates de régions

Rappelons que l'exécution d'un automate temporisé est infinie, l'idée des graphes des régions consiste en le partitionnement de cet espace d'états en régions finis. Ensuite, la construction effective du graphe qui imite le fonctionnement de l'automate initial est entreprise.

A partir de la notion de région, on peut construire une abstraction finie d'un automate temporisé, tout en préservant des propriétés particulières, ce qui permettra de se décider sur l'accessibilité des états dans un TA. Formellement, un automate des régions est défini comme suit :

Définition 9. *Region automata*. *Un automate des régions R(A) de l'automate temporisé* $A = (S, S_0, X, \Sigma, T)$ *est un automate fini défini par :*

- *L'ensemble des états est* $S \times \mathbb{R}_+^X$ *, où l'état initial est* (s_0, v_0) *;*
- *Les transitions sont définies par :* $(s, r) \xrightarrow{a} (s', r')$ *si* $\exists (s, G, a, \gamma, s')$ *dans A et r′′est le successeur de r, avec* $r'' \subseteq [\![G]\!]$ *et* $r' = r\,[\gamma \leftarrow 0]$.

Il faut noter que le nombre d'états d'un automate de régions est exponentiel en fonction du nombre d'horloges du TA. Selon les études et les preuves présentées dans [AD94], nous mentionnons quelques limites des automates de régions correspondants à des automates temporisés. L'accessibilité d'un état est PSPACE-complet, l'universalité est indécidable, l'inclusion de traces est indécidable et les TA ne sont pas clos par passage au complémentaire.

Depuis leur introduction, de nombreuses variantes d'automates temporisés ont été proposées. Les daTA's sont structurellement une sous-classe des automates temporisés [AD90, AD94, Alu99], néanmoins, la différence à souligner est celle qui concerne la sémantique associée au modèle. Dans la section suivante nous étudions ce type de modèle d'automate temporisé qui est le modèle choisi dans ce travail.

3.5.2 Automates temporisés avec durées d'actions

Le modèle des automates temporisés est construit en se conformant à l'hypothèse d'atomicité structurelle et temporelle des actions (actions indivisibles et de durées nulles). La sémantique sous-jacente est celle de l'entrelacement. Intuitivement elle suppose l'atomicité

des actions et réduit l'exécution des actions parallèles en leurs exécutions entrelacées dans le temps. Cette hypothèse peut rendre invalides certaines spécifications de système à modéliser. Les modèles temporisés traitent trois types d'actions : les actions observables, les actions internes, et le passage de temps matérialisé par une durée [Mok06, BB05, BPDG98]. La durée des actions n'a jamais été prise en compte explicitement. Dans ce contexte le modèle des automates temporisés avec durées d'actions (daTAs) a été introduit, particulièrement pour son utilisation dans la validation des systèmes.

Le daTA est un modèle temporisé qui observe dans sa sémantique propre les durées des actions en plus des notions importantes pour la spécification des systèmes temps réel telles que l'urgence et la latence [BST97]. Ce modèle [Bel10, SB06] issu des travaux qui reposent sur une sémantique dite de maximalité [Dev92, CS95, Saï96], et qui prône le vrai parallélisme, de ce point de vue, se prête bien à la modélisation des systèmes temps réel, concurrents et distribués [SB05, SBB08]. Ainsi, il nous paraît naturel d'introduire en premier lieu cette sémantique, puis nous décrivons plus en détail les différents concepts du modèle des automates temporisés avec durées d'action non nulles et sa définition formelle.

3.5.2.1 Sémantique de maximalité

Les modèles à vérifier sur lesquels la vérification est appliquée sont souvent générés à partir de spécifications en considérant la sémantique d'entrelacement. Pourtant, cette dernière n'exprime pas naturellement le vrai parallélisme, en plus elle ne permet pas de représenter correctement le comportement des systèmes concurrents dès que les actions ne sont plus atomiques (indivisibles et/ou instantanées). Pour remédier cette problématique, on peut utiliser les modèles du vrai parallélisme qui permettent d'éviter ces difficultés et de conserver la nature parallèle des programmes concurrents. En profitant de cet avantage, une sémantique permettant d'exprimer le vrai parallélisme, la sémantique de maximalité, a été définie [CS95, Sai96] et utilisée dans la littérature [SB03, SB05, SB04, SDAB04].

La sémantique de maximalité permet de préserver la causalité entre les occurrences des actions dans une expression comportementale, et aussi de capturer le vrai parallélisme entre celles-ci. Le principe de cette sémantique consiste à utiliser les relations de causalités pour associer, à chaque état de l'expression, les actions qui sont potentiellement en cours d'exécution. Dans ce sens, les transitions sont des événements qui ne représentent que le début d'exécution des actions. En conséquence, l'exécution concurrente de plusieurs actions devient possible, c'est-à-dire que l'on peut distinguer exécutions séquentielles et exécutions parallèles d'actions. Une présentation détaillée de la sémantique de maximalité peut être trouvée dans [CS95, Saï96].

3.5.2.2 Spécification des actions avec durées explicites

Le modèle des daTA's est défini afin de prendre en compte la non atomicité structurelle et temporelle des actions. L'idée est basée sur le principe de la sémantique de maximalité dans laquelle seuls les débuts des actions sont modélisés (les fins d'exécution des actions étant capturés par les durées correspondantes). Dans l'état résultant on dit que l'action est éventuellement en cours d'exécution, aucune conclusion ne peut être tirée concernant la fin de son exécution. Cependant, cette information peut être déduite dans un état ultérieur dans lequel une action qui lui est

causalement dépendante est exécutée. L'association de durées explicites aux actions va nous permettre de matérialiser le début et la fin d'exécution des actions.

Pour clarifier cette idée, considérons l'exemple d'un système *S* composé de deux processus P_1 et P_2 s'exécutant en parallèle et se synchronisant sur une action *d*. Le sous-système P_1 exécute l'action *a* suivie de *d*, tandis que P_2 exécute *b* puis *d*. Si on suppose que toutes les actions ont une durée nulle, *S* aura le comportement représenté par la figure 3.5.(a). A partir de l'état initial s_0, les deux actions *a* et *b* peuvent commencer leur exécution indépendamment l'une de l'autre. Puisque nous pouvons avoir le cas où ces deux actions s'exécutent en parallèle, nous allons attribuer à chacune d'elles une horloge *x* et *y* respectivement, pour distinguer leurs occurrences. Donc, à partir de l'état s_0, les deux transitions suivantes sont possibles : $s_0 \xrightarrow{a,x:=0} s_1$ et $s_0 \xrightarrow{b,y:=0} s_2$. Une transition étiquetée par *a* indique le début d'exécution de l'action *a*, l'horloge qui lui est associée comptabilise l'évolution dans le temps de cette action. En suivant le même raisonnement, les deux transitions suivantes sont possibles : $s_1 \xrightarrow{b,y:=0} s_3$ et $s_2 \xrightarrow{a,x:=0} s_3$.

Supposons maintenant que les actions *a*, *b* et d ont les durées respectives de 10, 12 et 4. Le comportement du système *S* est illustré par la figure 3.5. (b). A partir de l'état s_3, l'action *d* ne peut évidemment commencer son exécution que si les deux actions *a* et *b* ont terminé leur exécution. Donc, la transition *d* ne peut être tirée que si une condition portant sur les exécutions de *a* et de *b* est satisfaite. Cette condition, appelée condition sur les durées (DC en anglais Duration Condition), est construite en fonction des durées de *a* et de *b*. D'abord, nous montrons la construction des conditions sur les durées pour s_0, s_1 et s_2. Après le tirage de la transition $s_0 \xrightarrow{a,x:=0} s_1$, nous avons besoin d'une information sur l'exécution éventuelle de l'action *a* dans l'état s_1. On est sûr que l'action *a* termine son exécution lorsque l'horloge correspondante *x* atteint la valeur 10, donc, on ajoute à l'état s_1 la condition sur la durée de *a*, $\{x \geq 10\}$, qui veut dire que si la valeur de *x* est supérieure ou égale à 10 alors on est sûr que l'action *a* a fini de s'exécuter. La même chose pour l'état s_2 qui sera étiqueté par $\{y \geq 12\}$. A l'´état s_0, aucune action n'est en exécution, ce qui implique que l'ensemble des conditions sur les durées soit vide. A l'état s_3, les actions *a* et *b* peuvent éventuellement s'exécuter en parallèle, et chacune d'elles ne peut terminer que si son horloge atteint une valeur égale à sa durée. D'où l'ensemble des conditions sur les durées $\{x \geq 10, y \geq 12\}$. La condition d'exécution de l'action *d* devient alors $x \geq 10 \wedge y \geq 12$. A l'état s_3 la condition sur les durées des actions *a* et *b* implique la possibilité de leurs évolutions parallèles sur les durées $\{x \geq 10, y \geq 12\}$. La condition d'exécution de l'action *d* devient alors $x \geq 10 \wedge y \geq 12$. A l'état s_3 la condition sur les durées des actions *a* et *b* implique la possibilité de leurs évolutions parallèles.

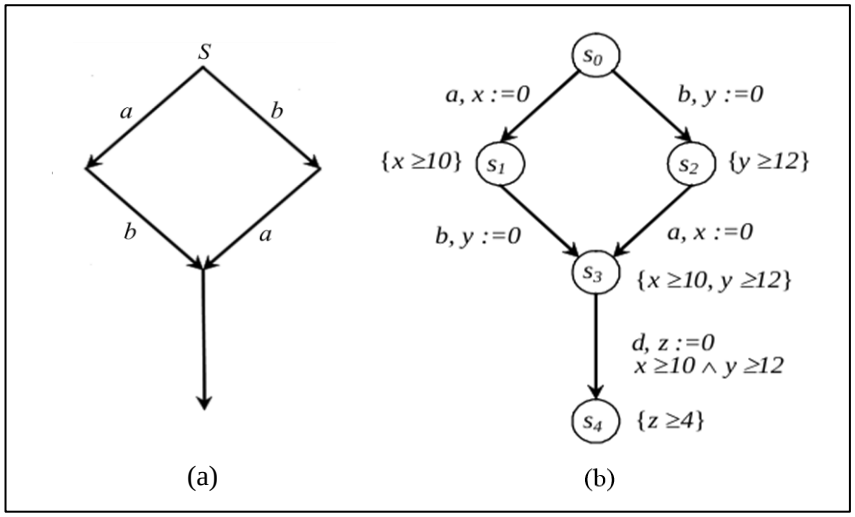


FIGURE 3.6– Représentation des durées explicites d'actions

3.5.2.3 Automate temporisé avec durées d'actions

Les daTA's sont structurellement une sous-classe des automates temporisés [AD90, AD94, Alu99], néanmoins, la différence à souligner est celle qui concerne la sémantique associée au modèle. Le modèle des daTAs permettant la prise en compte de la non atomicité temporelle et structurelle des actions dans les automates temporisés sans passer par l'éclatement des actions. Il considère que chaque action dure dans le temps, cette quantité est capturée par la condition de durée sur l'état destination d'une transition étiquetée par cette action, toutes les transitions offertes sont possibles alors que toutes celles qui dépendent de la terminaison de l'action en cours sont inhibées jusqu'à ce que la condition d'exécution attribuée à ces transitions soit vérifiée. Les actions qui doivent s'exécuter en parallèle sont au contraire lancer sans être concernée par les conditions de durées. Dans ce cas la condition de durée offre explicitement l'information sur l'évolution du système. La condition de durée d'une action *a* décore chaque état atteint par son exécution. Alors que toute transition est gardée par une condition d'exécution, c'est une conjonction sur des conditions de durée, elles concernent une transition dont le passage est conditionné par la terminaison d'un ensemble d'actions (dépendance causale dans le sens d'événements maximaux).

Un daTA est défini comme un automate temporisé sur un alphabet qui représente les actions à exécuter. Chaque horloge dans ce modèle est une variable réelle qui enregistre la durée de l'action, dans ce sens il existe une association entre les actions et les horloges. Les durées associées aux actions sont représentées par des contraintes sur les transitions et dans les états cibles de chacune d'elles. En ce sens, toute transition sensibilisée représente le début de l'exécution de l'action. Sur l'état cible de la transition une formule temporisée exprime que l'action est éventuellement en cours d'exécution.

Du point de vue opérationnelle, chaque action est associée à une horloge qui est remise à zéro au début de son exécution. Cette horloge est utilisée dans la construction des contraintes d'horloges comme des gardes des transitions. Pour illustrer la structure des daTAs, nous considérons l'exemple de la figure 3.7. Ce daTA est composé de trois états et deux actions consécutives *a* et *b*, *a* de durée

2 unités de temps et *b* de 3 unités de temps. A partir de l'état initial s_0, l'exécution de l'action entraîne une initialisation de l'horloge *x* qui lui est associée. $x \geq 2$ est une condition de durée sur l'action *a*, elle est enregistrée dans l'état s_1 et qui signifie que *a* est en cours d'exécution, de même pour la condition $y \geq 3$ associée à l'action *b*. La transition de s_1 à s_2 est franchissable si et salement si la condition sur la transition est vérifiée, celle-ci est dite condition d'exécution et elle signifie que l'action *b* ne peut être exécuté que si l'action *a* est achevée. Les formules temporelles $x \geq 2$ et $y \geq 3$ sur les états, représentent des informations sur les durées d'exécution des actions a et b.

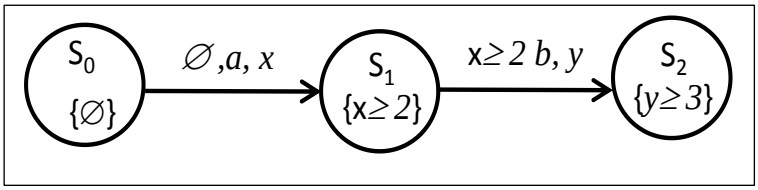


FIGURE 3.7– Automate temporisé avec durées d'actions

Définition 10. *Durational Action Timed Automaton (DATA). Un daTA A est un quintuplet* $(S, L_S, S_0, \mathcal{H}, T)$ *tel que :*

- *S est un ensemble fini d'états ;*
- $L_S : S \rightarrow 2^{\Phi,(\mathcal{H})}_{f_n}$ *est une fonction qui fait correspondre à chaque état s l'ensemble F des conditions de terminaison des actions potentiellement en exécution dans s ;*
- $s_0 \in S$ *est d'état initial ;*
- *H est un ensemble fini d'horloges ;*
- $T \subseteq S \times 2^{\Phi,(\mathcal{H})}_{f_n} \times Act \times \mathcal{H} \times S$ *est l'ensemble des transitions.*

Une transition (s,γ, a, x, s') représente le passage de l''etat s à l'état s', en lançant l'exécution de l'action a et en réinitialisant l'horloge x. est la contrainte correspondant qui doit être satisfaite pour tirer cette transition. (s,γ, a, x, s') peut être écrit $s \xrightarrow{\gamma,a,x} s'$.

Définition 11. *(Sémantique d'un daTA.). La sémantique d'un daTA,* $A = (S, L_S, S_0, \mathcal{H}, T)$ *est définie par un système de transitions infini* $\sum_A$ *sur l'alphabet* $(Act \cup \mathbb{T})$*.Une configuration de* $\sum_A$ *est un couple* $\langle s, v \rangle$ *tel que s est un état de A et v est une valuation sur* $\mathcal{H}$*.Une configuration est initiale* $\langle s_0, v_0 \rangle$ *si* s_0 *est l'état initial de A et* $\forall x \in \mathcal{H}$ *, v(x) = 0.*

Deux types de transitions entre les configurations de $\sum_A$ *sont possibles, correspondant au tirage d'une transition de A et au passage de temps :*

1- transition discrète : $\langle s, v \rangle \xrightarrow{a} \langle s', [\{x\} \mapsto 0]\, v \rangle$ *tel que* $\exists\ (s, G, D, a, x, s') \in T$ *et* $v \vDash G$ *;*

2- transition temporelle : $\langle s, v \rangle \xrightarrow{d} \langle s, v + d \rangle$ *tel que* $d \in \mathbb{T}, \forall\ d' \leq d, v + d' \nvDash D$*, ou* $v + d \vDash D$*,* $d \leq \eta$.

Où η est la plus petite quantité réelle de temps dans laquelle aucune action ne se produit.

3.5.3 Autres sous-classes des automates temporisés

Le modèle original d'Alur et Dill [AD90, AD94] a été vivement utilisé dans la spécification, la vérification et le test des systèmes temps-réel. A cet effet, plusieurs travaux ont été élaborés, nous pouvons citer à titre d'exemple : Timed Safety Automata [HNSY94], Event-Clock Automata englobe les Event-Recording Automata et les Event Predicting Automata [AFH94], Updatable Timed Automata [BDFP00, BDFP04, Bou02], Dynamic Timed Automata (DTA's) [CdO95b, Loh02], Timed Automata with non Instantaneous Actions [BFT01], Timed Automata with Deadlines (TADs) [BS98, BST97].

3.6 Conclusion

Au cours de ce chapitre, nous avons présenté avec plus ou moins de détails, certains formalismes avec l'aspect temps, et en particulier ceux considérés dans ce travail, à savoir les algèbres de processus, les réseaux de Petri, les systèmes de transitions temporisés étiquetées et les automates temporisés. Nous avons mis l'accent sur les différentes extensions temporelles de chaque modèle utilisées pour la modélisation de comportement dynamique des systèmes à temps contraint. D'après cette étude, on peut conclure que le choix d'un tel modèle dépend de type du système à modéliser et des conditions de son évolution, ainsi il doit être ouvert pour accepter de nouvelles contraintes de modélisation. Les deux modèles formelles, réseaux de Petri temporels avec durées d'actions et automates temporisés avec durée d'action seront utilisés et ont été utiles dans la partie suivante des contributions.

4. Vérification formelle

Sommaire

Chapitre 4

Vérification formelle

4.1 Introduction

Comme nous l'avons vu dans le chapitre 3 il existe plusieurs modèles temporels et/ou temporisés pour la spécification des systèmes temps réels et/ou embarqués, à savoir les algèbres de processus, les réseaux de Petri ou les réseaux d'automates temporisés. Une fois spécifiés dans un formalisme donné, les modèles de systèmes doivent être vérifiés. Il faut donc démontrer que les contraintes non fonctionnelles sont satisfaites.

Par vérification formelle des modèles, nous entendons toute technique permettant de confronter un modèle (sa description opérationnelle) à ses spécifications (aux propriétés que l'on attend de lui). Ce type d'approches nécessite trois (03) éléments :

1. Une description opérationnelle du système (son graphe de comportement), générée à partir d'un modèle de spécification,
2. Un langage de spécification permettant d'exprimer les propriétés du système que l'on souhaite vérifier,
3. Une procédure de décision qui permet de contrôler la conformité entre la description opérationnelle du système et sa spécification, c'est-à-dire une procédure qui permet de vérifier que le système satisfait effectivement les propriétés que l'on attend de lui.

La vérification peut être réalisée à l'aide de plusieurs techniques telles que preuve de théorèmes, model-checking et le test d'équivalence. Le model-checking est une technique de vérification automatique des propriétés d'exactitude dans un espace d'états fini. Les propriétés sont exprimées dans une logique (e.g temporelle), le modèle est exprimé dans un langage formel, et un algorithme vérifie la satisfiabilité d'une propriété sur le modèle du système réel [CGP99, CGKP18]. Dans la preuve de théorèmes (démonstration automatique), à partir d'un système et d'une propriété exprimée dans un même langage de spécification, à bâtir un raisonnement logique pour démontrer la correction ou non du système vis-à-vis de cette propriété [Duf91]. Cette correction est exprimée comme un théorème mathématique en utilisant des règles de déduction. Des assistants automatiques à la preuve fondés sur des règles de déduction sont utilisés. Un utilisateur doit donner des indications pour guider ces assistants [HKP02, Rus01]. L'avantage de cette technique par rapport au model-checking réside dans sa prise en compte des données complexes, ce qui lui permet d'être appliquée sur des espaces d'états non finis. L'inconvénient majeur est qu'un opérateur humain doit guider les assistants lorsqu'ils ne parviennent pas à démontrer certains lemmes. Ainsi, le processus global n'est pas totalement

automatisé et reste une technique plus ou moins lente. Comme pour model-checking, la technique de la vérification d'équivalence nécessite que les systèmes soient finis même si nous pouvons trouver des techniques où la vérification se fait en parallèle à la conception du système. Pour vérifier que deux modèles partagent une propriété donnée, les tests d'équivalence utilisent des relations d'équivalence particulières. Il s'agit dans la plupart des cas de prouver une propriété relativement abstraite sans avoir assez de données.

Dans ce chapitre, nous présentons comment les méthodes de vérification formelles peuvent être utilisées pour la vérification des systèmes à temps contraint. Notre attention se porte plus particulièrement sur les techniques de model-checking, qui sont la base de tous les outils de modélisation des réseaux de Petri. Ils interviennent plus tôt dans les cycles d'ingénierie et permettent d'éviter les reprises tardives qui correspondent à la majeure partie des coûts de développement des systèmes embarqués.

4.2 Vérification formelle

Lors de l'élaboration de la spécification d'un système, le développeur crée en ensemble de blocs réalisant les comportements demandés avant de les interconnecter afin de modéliser le fonctionnement global. Cependant, il existe des subtilités lors de la spécification de systèmes complexes qui font que le système n'aura pas forcément le comportement attendu une fois implémenté. Le développeur doit donc être en mesure de s'assurer que la spécification est correcte avant même de procéder à l'implémentation du système. Les outils de vérification formelle permettent de répondre à ce besoin.

La vérification formelle est le processus systématique de vérification, à travers des techniques algorithmiques exhaustives, qu'une implémentation est conforme à sa spécification [PF05]. En utilisant la vérification formelle, tous les chemins d'exécution possibles sont analysés mathématiquement et ainsi prouver de manière exhaustive que son comportement est correct.

Il existe plusieurs classes de méthodes de vérification formelle divisées selon le moyen utilisé pour exprimer les propriétés attendues du système. On distingue deux grandes classes : celle basée sur l'approche comportementale et celle basée sur l'approche logique.

— **L'approche comportementale :** Dans cette approche, la description opérationnelle du système (son graphe de comportement) ainsi que sa spécification, sont tous les deux exprimés par des comportements. La procédure de décision revient alors à décider de l'équivalence entre deux graphes. De nombreuses relations d'équivalence ont été proposées pour la comparaison et l'analyse des systèmes concurrents, allant de l'équivalence langage à l'équivalence observationnelle, en passant par les modèles de refus et les équivalences de test. Cette diversité s'explique d'une part, par la variété des propriétés spécifiques aux systèmes étudiés, et d'autre part, de la difficulté de définir formellement une sémantique des systèmes de processus.

— **L'approche logique :** Dans cette approche, les propriétés attendues du système sont exprimées par des assertions dans une logique, par exemple la logique temporelle. Nous allons voir dans les sections suivantes comment il est possible d'exprimer les propriétés requises de manière précise à l'aide de la logique mathématique. Dans ce contexte on distingue deux approches : une basée sur la preuve de théorèmes (proof checking) et une autre basée sur le contrôle de modèle

(model-checking). La première approche consiste à modéliser le programme dans un système formel, puis prouver la correction du programme avec des raisonnements syntaxiques. L'avantage de ces méthodes est qu'elles permettent de traiter des systèmes ayants un nombre infini d'états. En plus, l'intuition humaine peut guider le processus de vérification. En revanche, elles ne peuvent pas être complétement automatisées et nécessitent beaucoup d'effort et d'ingéniosité humaine. La deuxième approche restreinte à des systèmes ayant un nombre fini d'états, permet une vérification (relativement) simple et efficace, qui s'avère particulièrement utile dans les premières phases du processus de conception, quand les erreurs sont susceptibles d'être plus fréquents. Ces méthodes offrent l'avantage d'être complétement automatisables, mais souffrent du problème de l'explosion combinatoire d'espace d'états des comportements possibles du système.

4.3 Vérification formelle par model-checking

Le model-checking (*vérification de modèle*) est une méthode formelle qui a été créée par Clarke et Emerson [CE81], ainsi que par Queille et Sifakis [QS82] afin de permettre une vérification automatique et exhaustive des systèmes. Cette méthode a été s'appliqué à une large classe de systèmes : systèmes embarqués, protocoles de communication, industrie manufacturière, etc. Elle comporte trois étapes : la modélisation formelle du système étudié (un automate fini ou une variante de cette représentation), la spécification formelle de la propriété souhaitée par une formule logique (en logique temporelle) [Del99] et la vérification de la correction du modèle à l'aide d'un model-chacker, applique à ces données un algorithme qui vérifie si le modèle du système satisfait ou non le modèle de sa spécification. Cet algorithme dépend de la nature des modèles choisis pour le système et la propriété. La plupart des model-checker sont capables de générer une réponse positive si la propriété est garantie pour tous les comportements du modèle et une réponse négative complétée par un *contre-exemple* illustratif en cas de violation de la propriété.

Le gros *avantage* de model-checking est qu'il est (idéalement) complètement automatique, et qu'habituellement un contre-exemple est retourné quand la propriété n'est pas vérifiée. Ce dernier point a été déterminant pour une adoption industrielle. Cependant cette technique présente quelques limites réside principalement dans le parcours de l'espace des comportements possibles est très coûteux en ressource de calcul, et il est aussi impossible à réaliser sur certains types de modèles.

4.3.1 Principe

La démarche de model-checking consiste à vérifier de façon exhaustive des propriétés imposées sur un modèle du système étudie. Le model-checking permet de répondre à la question : "est-ce qu'un système satisfait une certaine propriété ?". La réponse à cette question se fait en trois (03) phases, et qui est schématisée par la figure 4.1. Nous détaillons maintenant de façon plus approfondie le principe général du model-checking [Odd03].

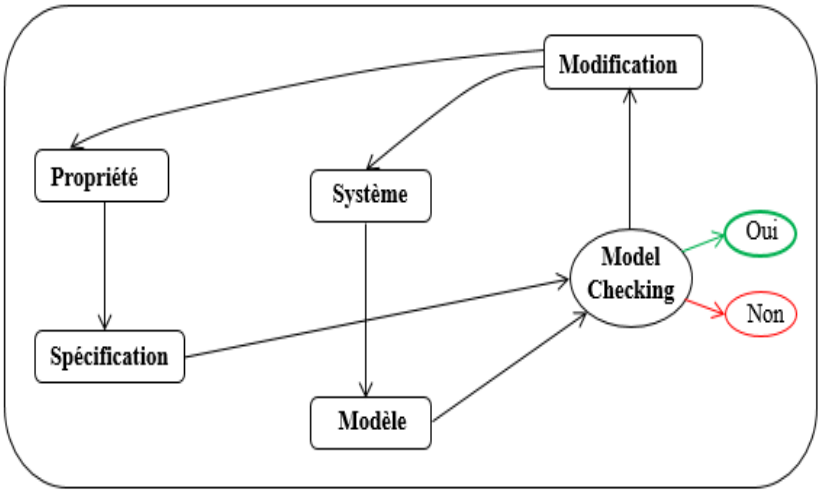


FIGURE 4.1– Principe du Model-checking

i. Modélisation

Un outil de model-checking ne travaille pas sur un programme, mais sur une modélisation de celui-ci. La modélisation d'un programme consiste à construire, dans un cadre formel précis et imposé par le model-checker, un objet symbolique dont le comportement est aussi proche que possible de celui du programme, et qui est particulièrement fidèle dès que des données ayant un lien avec la spécification sont en cause. Certaines applications permettent de générer automatiquement un modèle à partir du code source de certains programmes. Il est cependant en général utile d'effectuer une simplification du modèle obtenu, en ôtant tout ce qui n'a pas de rapport avec la spécification, afin de réduire le coût ultérieur du model-checking.

Un modèle peut aussi être construit à partir d'un simple cahier des charges qui décrit de façon informelle un algorithme. La vérification de ce modèle permettra ainsi de vérifier la validité des contraintes du cahier des charges avant de commencer l'implémentation du programme lui-même. Suivant la nature du programme et le model-checker utilisé, le modèle pourra être un simple système de transitions, ou aura une forme plus évoluée, permettant de modéliser la communication entre processus par exemple. Un modèle peut éventuellement être infini si son nombre de configurations n'est pas borné.

ii. Spécification

La spécification exprime, dans un contexte formel imposé par le model- checker, une ou plusieurs propriétés que le modèle doit vérifier. Le langage d'expression d'une spécification se doit en général d'inclure les prédicats de la logique utilisée. Cependant il est courant de devoir exprimer des propriétés incluant des contraintes temporelles. Inclure des contraintes en temps réel complique fortement le model-checking. C'est pourquoi nous nous intéressons le plus souvent à l'ordre dans lequel les événements s'enchainent, on parle alors de la logique temporelle [Pnu77, Eme90]. Celle-ci permet d'exprimer des contraintes naturelles comme "*jamais il n'y aura une erreur*" ou encore "*toute demande sera suivie d'une réponse*". Les premiers travaux sur le model-Checking de formules de logique temporelle ont été dirigés par Edmund M. Clarke et E. Allen Emerson en 1981 [EC82], et Jean-Pierre Queille et Joseph Sifakis en 1982 [QS82a]. Lorsqu'on souhaite vérifier les propriétés d'un système, il est nécessaire de pouvoir les exprimer dans un langage adéquat.

La logique temporelle répond à ce besoin. Des formalismes simples de la logique temporelle permettent d'exprimer les propriétés usuelles des programmes parallèles.

Logique temporelle

La logique temporelle est un outil formel qui procure une syntaxe sûre, précise et sans ambiguïté des propriétés qualitatives et quantitatives. Contrairement à la logique classique, où il existe une fonction d'interprétation unique à partir de laquelle il est possible de déduire la valeur de vérité de chaque formule, dans la logique temporelle la fonction d'interprétation est d´définie sur un graphe. Dans ce graphe (modèle), chaque nœud correspond à une fonction classique. A cause de l'introduction d'opérateurs temporels (**A** : quel que soit le chemin, **E** : pour un chemin donné, **G** : toujours, **F** : parfois, **X** : prochain et **U** : jusqu'à), l'interprétation d'une formule dans un nœud ne dépend pas seulement des connaissances de ce nœud, mais peut dépendre des connaissances des autres nœuds du graphe d'interprétation. Les opérateurs précédents sont ceux de la logique temporelle arborescente CTL [CES86, Eme90]. En prenant comme graphe d'interprétation le système d'états-transitions associé à un programme, il est possible d'exprimer des propriétés sur les valeurs des formules logiques du programme à chaque état.

Plusieurs logiques temporelles existent, avec des pouvoirs d'expression différents. Deux (02) familles des logiques temporelles sont couramment utilisées : les premières sont des logiques temporelles linéaires, et les secondes sont des logiques temporelles arborescentes.

Logique LTL

Dans le domaine de la spécification et de la vérification des systèmes, la logique temporelle linéaire LTL (Linear Temporal Logic) constitue la logique temporelle de base [Pnu77]. Elle a émergé comme sémantique standard pour l'expression des propriétés de fonctionnement. La LTL présente l'avantage de pouvoir raisonner le comportement attendu au cours d'une séquence linéaire d'états. Le futur est alors vu comme une séquence d'états ou plus généralement un chemin tel que dans la figure 4.2 où une séquence d'actions qui se suivent :

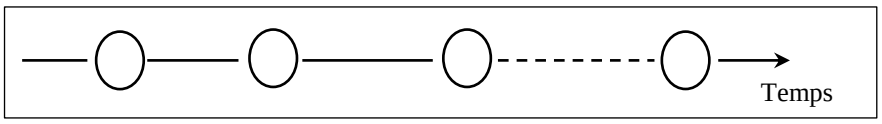


FIGURE 4.2– Evolution temporel d'un système dans une logique linéaire

Cette manière de spécifier est intuitive pour le développeur puisque la progression dans le temps peut être vue comme une trace d'exécution.

Logique CTL

La CTL (Computational Tree Logic), introduite au début des années 80 par Clarke et Emerson, est une logique temporelle propositionnelle de branchement utilisée fréquemment dans les techniques de model-checking [CE81, BAMP83, CES86, BBL+99]. Elle permet de considérer des modèles de systèmes dans lesquels il existe plusieurs séquences d'exécutions finies à partir d'un état donné du système, d'où la notion d'arbre. Cette logique s'oppose à la logique

"Propositional Linear Temporal Logic" (PLTL) où l'on ne considère à chaque fois qu'une seule séquence finie d'état du système vérifiant une propriété donnée.

Logique TCTL

Pour pouvoir exprimer des propriétés spécifiques des systèmes temps-réel c-à-d des propriétés quantitatives, des extensions temporelles des différentes logiques temporelles ont été proposées, telle que la logique TCTL (Timed CTL). La "Timed Computation Tree Logic" (TCTL) étend la logique CTL [ACD93]. Ainsi des formules peuvent par exemple être vérifiées dans un état pendant un certain temps *t* borné avant qu'elles ne soient invalidées. La logique TCTL permet, grâce à la notion de temps borné, de vérifier des propriétés temporisées des systèmes à temps réel où les séquences d'exécutions ne sont pas finies. Un exemple d'une propriété temporisée : la barrière s'ouvre en moins de 5 secondes à l'approche d'une voiture. Cette logique est très souvent associée aux automates temporisés.

Formellement on définit la logique TCTL de la manière suivante :

- **Syntaxe de TCTL**

Soit AP un ensemble de propositions atomiques, les formules de TCTL sont définies par les règles suivantes :

- S1 : chaque proposition atomique P est une formule de TCTL ;
- S2 : si f et g sont des formules de TCTL alors (f∨g) et (¬f) sont des formules de TCTL ;
- S3 : si f est une formule de chemin alors E f U∼c g et A f U∼c g sont des formules de TCTL, avec ∼∈ {<, <=, =,>,>=} et c∈ N. "c" peut être un intervalle de temps.

- **Sémantique de TCTL**

Pour une séquence d'exécution (chemin) π= (s0, s1,...) du système, on note π(t) l'état du système à l'instant t sur cette séquence π et suc(s) l'ensemble des chemins issus de s. La relation de satisfaction des formules de TCTL est définie par :

s|= p ⇔ p∈L(s)

s|= f∨g ⇔s|= f ou s|= g

s|= ¬ f⇔¬ s|= f

s|= **E** f **U**∼c g ⇔∃ π ∈ suc(s) ; ∃ t∼c tel que π(t) |= g et ∀ t'(0≤ t'<t), π (t')|= f

s|= **A** f **U**∼c g⇔ ∀ π ∈suc(s) ; ∃ t∼c tel que π (t) |= g et ∀ t'(0<= t'<t), π (t') |= f

On définit également quelques abréviations classiques :

- **EF**∼c f pour **E** (true **U**∼c f), signifie qu'il existe une exécution pour laquelle f devient vraie avant l'instant c (potentialité bornée) ;

- **AF**~c f pour **A** (true **U**~c f) ;

- **EG**~c f pour ¬**AF**~c ¬f, signifie qu'il existe une exécution pour laquelle f reste vraie jusqu'à l'instant c (quasi_invariance borné) ;

- **AG**~c f pour ¬**EF**~c¬f.

On peut également utiliser des intervalles de temps pour la spécification des propriétés.

iii. Vérification

C'est la dernière étape du processus de model-checking. Elle consiste à concevoir des algorithmes de vérification de propriétés sur les modèles. On utilise pour cela un algorithme qui prend en entrée un modèle exprimé dans un formalisme imposé, et une spécification qui exprime une propriété qui doit vérifier certaines données du modèle. Il effectue ensuite un calcul à partir de ces données, et peut produire deux résultats différents : soit *toute*s les exécutions du modèle satisfont la spécification, et le résultat est positif, soit *au moins* une exécution du modèle ne satisfait pas la spécification, et dans ce cas le résultat est négatif et le model-checker donne cette exécution ou une simplification de celle-ci comme un contre-exemple d'exécution non satisfaisante. A partir de ce contre-exemple généré, l'utilisateur peut essayer de corriger la source du problème puis effectuer une nouvelle vérification du modèle. La source du problème peut être soit une erreur de l'application ou du cahier des charges, soit une erreur de modélisation, soit enfin une erreur de spécification. Le contre-exemple permet de retrouver ces trois types d'erreurs.

Le calcul, effectué par l'implémentation d'un algorithme de vérification, sera plus ou moins complexe suivant le format accepté pour le modèle et la spécification. La complexité se traduit par un coût en temps de calcul d'une part, et en espace mémoire utilisé d'autre part. En règle générale, plus le formalisme permet d'exprimer des propriétés riches, plus le calcul sera coûteux. Même sur des modèles finis, la complexité des systèmes étant de plus en plus grande, la taille des modèles a fortement augmenté : c'est ce que l'on appelle *l'explosion combinatoire.* Cette augmentation exponentielle de la difficulté ne peut pas être compensée par l'augmentation de la puissance des machines, et de nombreux travaux ont donc été effectués pour trouver des solutions : ne représenter qu'une partie de modèle en mémoire, réduire le nombre de chemins à explorer, décomposer le système à vérifier, utiliser des techniques d'abstraction, ou utiliser une représentation symbolique plus compacte. Dans certains cas, en particulier quand le modèle est infini, le problème *indécidable*. Il existe des model-checker qui essaient de résoudre ces problèmes, mais sur certaines données ils sont incapables de retourner un résultat. Ils utilisent des méthodes comme l'interprétation abstraite ou l'accélération. Dans le cas des modèles finis, les logiques temporelles couramment utilisées sont en général décidables.

Type de propriétés à vérifier

Les propriétés qu'on voulait vérifier sont généralement divisées en deux grandes classes : classe des propriétés de sûreté (*safety*) et celle des propriétés de vivacité (*liveness*) [Lam83]. Ce sont des propriétés classiques permettent d'assurer le bon fonctionnent des systèmes temps-réel qui sont concurrents et critique ainsi, plusieurs model-checker ont été développés pour les vérifier. Les propriétés de première classe assurent que quelque chose de bon se produira immanquablement

durant l'exécution du système. Les propriétés de deuxième classe expriment le fait que quelque chose de mauvais ne se produira jamais durant l'exécution du système.

En utilisant les logiques temporelles détaillées dans la section précédente, on pouvait exprimer et ensuit vérifier des propriétés relatives à la sûreté et la vivacité :

- Une propriété de sûreté est *l'exclusion mutuelle*. Soit deux processus, p_1 et p_2 utilisant un objet commun, appelé *ressource critique*. On dit que ces deux processus sont en exclusion mutuelle lorsque l'un des deux a une section critique dans son programme. A un instant donné, un et un seul processus exécute sa section critique. C'est-`a-dire que durant toute l'exécution du système les deux processus p_1 et p_2 ne peuvent jamais être en même temps en section critique. Une autre propriété de sûreté est *l'absence de blocage* dans un ensemble de processus. Cette propriété exprime le fait qu'à un moment donné, au moins un processus n'est pas en attente.

- *La terminaison* est un exemple d'une propriété de vivacité. Elle exprime que toute opération qui commence son exécution doit se terminer à un moment donné.

- Une autre classe de propriétés de vivacité est la classe des *propriétés d'équité*. Quand un processus demande l'entrée dans sa section critique, son désir sera satisfait éventuellement à un moment donné.

4.3.2 Méthodes

A. Les méthodes explicites

Une méthode explicite consiste à représenter le système sous la forme d'un automate. La première étape du model-checking consiste à exprimer le modèle considéré au moyen d'un graphe orienté, formé de nœuds et de transitions. Chaque nœud représente un état du système, chaque transition représente une évolution possible du système d'un état donné vers un autre état. Parallèlement, le système est décrit par un ensemble de propositions logiques atomiques (exemple, i=2, le processeur 3 est en attente). Chaque état du graphe orienté est étiqueté par l'ensemble des propositions atomiques vraies à ce point d'exécution. Un tel graphe est appelé structure de Kripke [Sau63].

La deuxième étape du model-checking consiste à exprimer la négation de la formule de logique temporelle que nous souhaitons tester. La négation de cette formule est donc elle-même transcrite sous forme d'une structure de Kripke, capable de reconnaître exactement l'ensemble des exécutions satisfaisant la négation de la formule donnée. Par exemple, on pourra transcrire une formule LTL en un automate de Büchi non-déterministe (ou parfois en un automate de Rabin).

La troisième et dernière étape consiste à réaliser le produit cartésien synchrone des deux structures de Kripke obtenues précédemment. Si le langage reconnu par le produit est vide (pour une propriété d'accessibilité, mais plus généralement pour une propriété d'acceptation mettant en jeu la vivacité et l'équité), alors le système satisfait la formule de logique. Sinon, toute séquence appartenant au langage du produit constitue un contre-exemple à la spécification.

Énumérer explicitement tous les états de l'automate peut être coûteux, c'est pourquoi on procède généralement par des méthodes symboliques, introduites par Ken.McMillan et Ed.Clarke. La figure 4.3 illustre un exemple d'automate associé au code suivant :

```
Random r = new Random (2) ; \ \
int a = 3 ; \ \
a = r.nextInt (0) ; \ \
if (a) \{ \ \
\hspace*{7mm} a++; \ \
\}else \{\ \
\hspace*{7mm} a--; \ \
\}\ \
a = a*r.nextInt (1); \ \
```

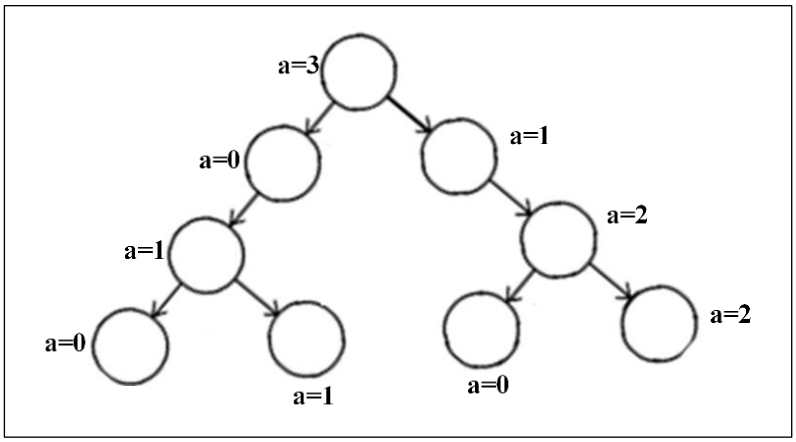


FIGURE 4.3 – Exemple d'un automate

Le programme va parcourir exhaustivement les états atteignables par lui-même et confirmer que les propriétés sont vérifiées sur chacun d'eux.

B. Les méthodes symboliques

Les méthodes symboliques fonctionnent d'une autre manière et utilisent généralement la structure des BDD **(**Binary Decision Diagram) [Sal09]**.** Un BDD est un diagramme binaire utilisé pour représenter une fonction booléenne. Chaque nœud de décision du BDD est étiqueté par une variable booléenne et possède deux nœuds fils :

- Fils bas : représente l'affectation de la variable à 0.
- Fils haut : représente l'affectation de la variable à 1.

Les nœuds terminaux sont *1* et *0* qui signifient que l'état correspondant appartient ou n'appartient pas à l'ensemble. La figure 4.5. (a) représente le BDD de l'ensemble explicité à sa gauche.

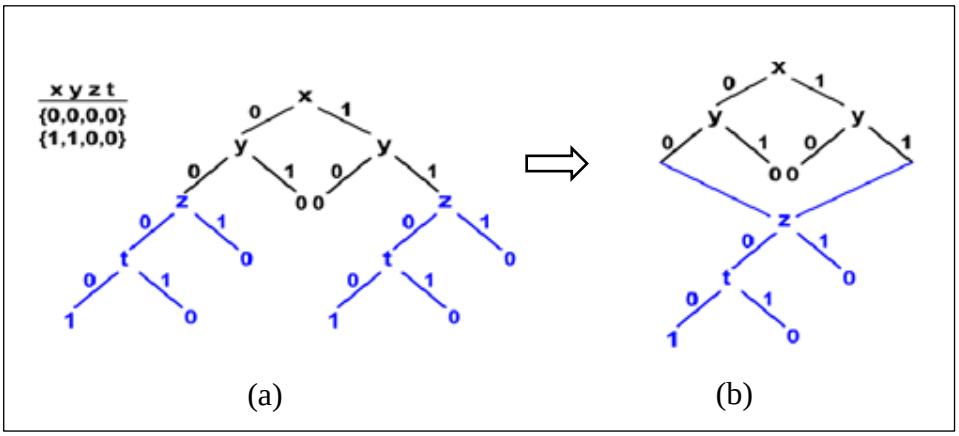


FIGURE 4.4 – (a) : le BDD, (b) : le ROBDD correspondant

Cependant, les model-checker utilisant ce type de structure peuvent optimiser ces BDDs. Le graphe dans la figure 4.5.(b) peut par exemple être optimisé en regroupant les deux zones bleues qui sont identiques. Ainsi, un BDD peut être réduit. En ajoutant un ordre sur les variables booléennes (dans la figure 2.5.(b), *x*> *y*> *z*>*t*), on obtient un ROBDD. L'avantage d'un ROBDD est qu'il est *unique* : il est donc aisé de déterminer l'équivalence Diagramme de Décision Binaire Ordonné Réduit de deux (02) fonctions. Ainsi, une formule sera toujours vraie si son ROBDD est réduit au nœud racine 1.

Les BDDs sont utilisés par le model-checker car ils permettent de stocker les données sur les automates. En effet, un automate est un ensemble de fonctions booléennes (qui symbolisent les états et les fonctions de transition), il peut donc être représenté par un BDD.

C. Résolution SAT

Il existe une autre méthode, au lieu de la considérer l'ensemble des traces d'exécution du système, on peut se limiter à des traces finies de longueurs bornées. L'existence d'une trace vérifiant une certaine propriété est équivalente à satisfiabilité d'une certaine formule logique. Par exemple, si *I* identifie les états initiaux du système, *F* les états dont on veut tester l'accessibilité, et *T* est une relation de transition, alors on considère la formule suivante :

$$\exists x_1 \dots, x_n \; I(x_1) \wedge T(x_1, x_2) \wedge \dots \wedge T(x_{n-1}, x_n) \wedge F(x_n)$$

Si les états du système sont donnés par des *n-uplets* de booléens, alors on se ramène au problème de la satisfiabilité d'une formule propositionnelle (problème SAT). Il existe divers logiciels, nommés *SAT solvers*, qui peuvent décider, efficacement en pratique, le problème SAT. De plus, ces logiciels fournissent habituellement un exemple de valuation satisfaisant la formule en cas de succès. Certains peuvent produire des éléments d'une preuve de non satisfiabilité en cas d'échec.

Une évolution récente est l'ajout de variables entières ou réelles. Les formules atomiques ne sont alors plus seulement les variables booléennes, mais des prédicats atomiques sur ces variables entières ou réelles (e.g. des prédicats pris dans une théorie). On parle alors de satisfiabilité modulo une théorie. Par exemple, on considère comme prédicats atomiques les égalités et inégalités

linéaires. Une approche consiste à remplacer les prédicats atomiques par des variables booléennes supplémentaires, et à résoudre le système via SAT. S'il n'y a pas de valuation vérifiant la formule booléenne, la formule originale n'était pas non plus satisfiable. S'il existe une valuation, il faut vérifier que celle-ci soit cohérente par rapport à la théorie. Par exemple, si on a remplacé $x < 5$ par un booléen b1 et $x > 6$ par un booléen b2, la valuation b1 = b2 = vrai est incohérente par rapport à la théorie des inégalités linéaires. En pratique, il faudra donc savoir décider effectivement la satisfiabilité d'une conjonction de prédicats atomiques.

4.3.3 Outils de vérification

Il existe nombreux outils model-checker avec pour la plupart un langage d'entrée dédié permettant de modéliser un certain type de système. Le choix d'un model-checker dépend donc de deux paramètres qui sont : le type de système que l'on souhaite modéliser et le type d'exigence que l'on souhaite vérifier. Une démarche d'utilisation du model-checking est la suivante :

1. Quel type de propriété veut-on vérifier sur le système ? ;
2. Quel model-checker peut traiter cette propriété ? ;
3. Exprimer le système et la propriété dans les formalismes de l'outil.

Les outils model-checker peuvent être répertoriés en deux classes : mode-checker qualitatifs, se basent sur une modélisation qualitative, et model-checker quantitatifs à temps continu, se basent sur une modélisation quantitative.

Les model-checker qualitatifs sont puissants et permettent de vérifier des propriétés telles que l'ordre d'événement et la fin de tache. Ils ne permettent pas de vérifier des propriétés faisant référence au temps quantitativement tel qu'une propriété de vivacité bornée « *toute requête finira par être satisfaite en moins de 5 minutes* ». Ceci est dû au fait que ces outils se basent sur des automates non temporisés. Pour traiter ce genre de propriétés il faut avoir recours à des model-checker quantitatifs, qui se basant sur des automates temporisés ou hybrides rajoutant des structures permettant de calculer le temps.

- **Model-checker qualitatif**

Les model-checker qualitatifs se basent sur des logiques qualitatives pour la description du comportement et des propriétés à vérifier d'un modèle. Nous citons :

- SMV [McM93] : est un model-checker symbolique comporte un langage de description d'automates dont le nombre d'états est fini. Il permet de démontrer des propriétés écrites sous la forme de formules logiques LTL. L'utilisation des structures BDDs dans cet outil a permis de vérifier des propriétés sur des modèles de taille importante : [CGL94] présente une étude de cas d'un circuit logique dont l'automate possède 10^{1300} états.

- *NuSMV* [CGR99] : est une extension de SMV. Il permet la représentation d'automates finis synchrones et asynchrones. Les propriétés peuvent être exprimées par de formule LTL ou CTL.

- *SPIN* [Hol97, RH04] : est un model-checker développé par Bell Labs. Ce model-checker est spécialisé dans la vérification de systèmes concurrents ou asynchrones. Le langage du modèle

est *Promela*. Les propriétés que l'on peut vérifier sont non fonctionnelles, absence d'état bloquant, ou fonctionnel : propriété de sûreté ou de vivacité.

- **Model-checker quantitatif à temps continu**

Les model-checker à temps continu ont été introduits pour la vérification des systèmes temps réel temporisés ou hybrides. Ils utilisent des variables d'horloges valuées réellement. Nous citons les model-checker les plus répandus :

- *KRONOS* [DTY95] : est développé par le laboratoire Verimag. Ce model-checker symbolique est dédié à la vérification de systèmes temps réel modélisés sous la forme d'automates temporisé. Il peut vérifier des *propriétés fonctionnelles de vivacité* ou des propriétés non fonctionnelles telles que l'absence de *zeno* ou les réponses à requêtes en temps fini. KRONOS a été étendu et combiné avec d'autres outils de vérification dans l'environnement TAXYS [BCP+01].

- *UPPAAL* [LP97] : est un outil de simulation et de vérification d'automates temporisés développé par Uppsala et Alborg. Il est capable de vérifier des propriétés de sûreté et d'atteignabilité.

- *Hytech* [AHH96, HHW97] : est un outil de vérification dédié à la modélisation et vérification des systèmes embarqués. Il est développé par l'université de Berkeley. Cet outil possède un model-checker symbolique capable de calculer les paramètres pour lesquels le modèle satisfait des propriétés temporelles. Les modèles sont des automates hybrides (un système est dit hybride s'il contient des composants discrets et continus).

- *CMC* [LL98] (Compositional Model-checking) : est un outil dédié aux automates temporisés. Il propose une approche de vérification de formules par composition. CMC a pour but d'utiliser la structuration des spécifications en un réseau d'automates temporisés communiquant pour éviter l'explosion combinatoire de l'espace de recherche.

Autres outils

- *SAL* [DOR+04] : est un environnement de spécification et d'analyse de systèmes concurrents spécifiés sous forme d'automates. Son langage est une extension de celui proposé dans PVS [DR97]. L'environnement SAL propose une série de techniques (abstraction, de génération d'invariants, de slicing) et d'outils (démonstrateurs de théorèmes et des model-checker) permettant d'analyser les automates. Quatre (04) model-checker sont disponible : un model-checker symbolique (SMC), un model-checker borné (BMC), un Witness Model-Checker (WMC), et un infini borné model-checker (infBMC).

SMC et WMC représentent l'automate de manière symbolique en utilisant les BDDs. BMC utilise un solver SAT et permet d'analyser des systèmes définis avec des types finis. infBMC utilise le solveur SMT Yices [DD06] qui permet aussi bien d'analyser des systèmes définis avec des types finis qu'avec des types infinis. BMC et infBMC peuvent chercher des contre-exemples de longueur finie *k.* de plus, ils peuvent démontrer des propriétés par *k*-induction.

- *Prover Plug-In* : est commercialisé par Prover Technology. Ce modèle est intégré aux environnements de travail de SCADE Suite et de SIMULINK. La transformation des modèles (SCADE ou SMILINK) vers le modèle d'entrée du model-checker est automatique.

Prover Plug-In propose deux algorithmes, *Bounded model-checking* [BCC03] qui est adapté à la recherche d'erreurs, et *Induction over time* [SSS00] qui est adapté à la démonstration de propriété.

4.3.4 Problème de l'explosion combinatoire

La principale limite à l'utilisation du model-checking est liée au problème de l'explosion combinatoire, le nombre d'états augmente de façon exponentielle en fonction de la complexité du système. La taille de l'espace d'états peut alors excéder la quantité de mémoire disponible, rendant impossible la vérification de propriétés sur l'espace d'états. Par exemple, considérons un système composé d'un ensemble de *n* processus à deux états. L'état global du système peut être représenté par un vecteur de *n* bits et le nombre maximum d'états dans le système est 2^n. Pour réduire l'impact de ce problème, plusieurs solutions ont été proposées. Nous présentons dans cette section quelques techniques de réductions parmi les plus connues.

Il est important de noter que la modélisation est une étape privilégiée pour la prévention de l'explosion combinatoire : Le choix des abstractions et des simplifications influe énormément sur la taille de l'espace d'états. Une bonne abstraction est une abstraction qui ne conserve que le minimum de comportements nécessaires pour la vérification des propriétés. Il est alors important de ne vérifier qu'un nombre limité de propriétés. Les simplifications doivent n'éliminer, dans la mesure du possible, que les comportements inutiles à la vérification des propriétés. Finalement, il n'est pas possible de définir des abstractions ou simplification sans connaitre précisément la sémantique du langage de modélisation ni les comportements, souvent subtils, du système à vérifier.

i. Model-checking symbolique par les BDD

Le model-checking symbolique [BCM+92] est une technique où les états sont représentés symboliquement par des diagrammes de décision. Cette représentation symbolique permet une représentation plus compacte de l'espace d'états.

Les valeurs de vérité d'une proposition booléenne peuvent être représentées par un arbre de décision binaire. Dans cet arbre, une branche représente une assignation aux variables de la formule et la feuille de la branche porte la valeur de vérité de la formule étant donnée cette assignation. Cette représentation n'est pas optimale : l'arbre peut contenir des sous arbres isomorphes et par définition il n'y a que deux valeurs de vérité possibles. La solution à ce problème est un graphe acyclique, appelé diagramme de décision binaire (BDD), qui identifie de manière unique les sous arbres isomorphes et les feuilles identiques (Section 4.3.2). Il a été démontré [Bry92] qu'il existe un diagramme unique pour chaque ensemble d'états, représentés par des tuples booléens sur un ensemble de propositions atomiques ordonnées.

Des structures analogues aux BDD, appelées diagrammes de différences d'horloges (CDD), ont été proposées pour la vérification des automates temporisés [BLP+99, Wan02]. Un CDD est un graphe orienté acyclique qui capture une union de zones. Un nœud terminal représente une valeur booléenne. Un nœud non terminal est associé à une paire d'horloges. Chaque arc sortant est associé à un intervalle bornant la différence entre les horloges du nœud. Un chemin dans le CDD représente une zone définie par la conjonction des contraintes associées au chemin. Étant donnée une zone, elle appartient à l'union s'il existe un chemin dans le graphe qui se termine par une constante vraie.

Comme pour les BDD, la taille d'un CDD est fonction de l'ordre (total) choisi pour les nœuds mais contrairement aux BDD il n'existe pas de CDD unique pour un ensemble d'horloges. L'efficacité de cette approche dépend fondamentalement de l'ordre choisi pour ordonner les paires d'horloges. En fonction du modèle à vérifier, il n'est pas toujours possible de trouver un ordre pour lequel la méthode est efficace.

ii. Ordres partiels

Une des causes de l'explosion combinatoire réside dans la représentation du parallélisme par l'entrelacement d'actions. L'idée des ordres partiels est d'éliminer autant que se peut les entrelacements inutiles lors de la construction de l'espace d'états. La méthode est basée sur une analyse structurelle préalable des relations de dépendances entre les différentes transitions. La réduction exploite la commutativité des transitions concurrentes indépendantes, pour lesquelles l'état obtenu est le même quel que soit l'ordre d'exécution. Suivant le cas, seul un sous-ensemble de transitions sera localement exploré *stubborn sets* [Val90], les *persistent sets* [GLH+96] et les *ample sets* [Pel94], soit, sous certaines conditions, on franchira en un seul pas atomique un ensemble de transitions indépendantes pas couvrants [VAM96].

Il est possible de combiner la technique des *stubborn sets* et des pas couvrants, puis de construire le graphe de pas persistants [RB02]. Les techniques de réduction ordres partiels préservent l'absence de blocage, et sous certaines conditions, la structure linéaire ou arborescente de l'espace d'états.

iii. Exploitation des symétries

La notion de symétrie d'un objet peut se définir comme la stabilité de cet objet vis-à-vis de certaines transformations. On dit que l'objet est symétrique. En géométrie par exemple, un polygone régulier est invariant par rotation et réflexion. L'intérêt pratique des symétries est qu'elles simplifient l'étude des objets symétriques : puisque l'objet est stable vis à vis de certaines transformations, certaines de ses propriétés le sont aussi. On peut réduire l'étude des propriétés aux parties de l'objet qui ne sont pas stables par ces transformations.

Intuitivement, un système est structurellement symétrique lorsque ses composants et leurs relations sont identiques modulo un identifiant. Nous pouvons utiliser ces symétries pour réduire l'espace d'états grâce au *principe de symétrie* [Ros98] Ce principe énonce que : *Il y a au moins autant de symétries dans les effets que dans les causes*. Par analogie, on identifie les *causes* avec la description du modèle du système et les *effets* avec les comportements du système, capturés dans l'espace d'états. Les symétries structurelles du système induisent des symétries sur le graphe de l'espace d'états. Les symétries du graphe, qui sont des automorphismes, induisent une relation d'équivalence sur ses sommets.

Les méthodes de réduction par symétries construisent un quotient de l'espace d'états par cette relation d'équivalence, réduisant sa taille par un facteur proportionnel à la quantité de symétries dans le système. Les méthodes de réduction par symétries préservent l'accessibilité, modulo symétrie, mais peuvent aussi préserver les formules de logique en temps linéaire [HI98].

4.4 Vérification formelle basée sur la sémantique de maximalité

Nous avons vu dans le chapitre 3 qu'une sémantique de maximalité permet de distinguer entre exécutions séquentielles et exécutions parallèles d'actions. Ainsi, l'utilisation d'une approche de vérification formelle basée sur une sémantique de vrai parallélisme nous donne la possibilité de vérifier d'autres propriétés ne pouvant pas être vérifiées dans une approche de vérification basée sur l'entrelacement des actions concurrentes. Un travail présenté dans [Bel10] a montré comment adapter un algorithme de model-checking pour pouvoir vérifier d'autres propriétés telles que l'exclusion mutuelle ainsi que de nouvelles propriétés qui portent sur les actions et leur exécution parallèle. L'expression de ces propriétés ne nécessite pas l'utilisation d'une nouvelle logique ou l'introduction de nouveaux opérateurs, car on peut utiliser les logiques temporelles arborescentes CTL ou TCTL et considérer les actions dans les états comme étant des formules atomiques. Cependant, ce qui change c'est l'intuition derrière les formules. A titre d'exemple, la formule **EF** ($a \wedge b$) où a et b sont des noms d'actions, signifie qu'il existe au moins un chemin dans lequel l'exécution en parallèle de a et b peut avoir lieu, de manière similaire on peut expliquer intuitivement toutes les formules de la logique CTL dont le modèle sur lequel elles vont être vérifiées est un modèle sémantique, système de transitions étiquetées maximales (*STEMs*) [Bel05, Bel10].

- a ∧ b dans un état *s* signifie que a et b peuvent être exécutées en parallèle dans l'état *s*.
- ¬ a dans un état *s* signifie que l'exécution de a dans l'état *s* ne peut pas avoir lieu.
- **EX**a dans un état s_0 signifie qu'il existe au moins un chemin (s_0, s_1, ...) où a pourra s'exécuter dans l'état s_1.
- **AX**a dans un état s_0 signifie que quel que soit le chemin (s_0, s_1, ...) issu de l'état s_0, a pourra s'exécuter à l'état s_1.

En outre, dans la structure des modèles STEMs, chaque action est associé un nom d'événement qui permet de distinguer entre plusieurs actions de même nom qui sont en exécution parallèle. Par conséquent, on peut avoir plusieurs actions du même nom dans un même état mais les noms des événements associés seront différents. En partant de ce point, on peut envisager des formules qui nous permettront de raisonner sur le nombre d'occurrences d'une action qu'on peut avoir en parallèle, c'est-à-dire vérifié le degré d'autoconcurrence d'une action.

4.5 Conclusion

La vérification formelle par model-checking apporte une approche exhaustive qui a l'avantage d'être totalement automatisable. C'est ce qui fait sa force mais aussi sa limitation.

En théorique, la limitation principale des techniques standards de model-checking est que le modèle sémantique du système à vérifier doit être fini : grossièrement, le programme ne doit manipuler que des variables à domaine fini. C'est souvent le cas en pratique, mais pas toujours. Soit que le système est infini par nature (ex : systèmes dépendants du temps et donc variables dans R), ou que les bornes sont impossibles à estimer (ex : bornes des canaux de communication de l'internet), ou bien encore que le système dépend de paramètres (mémoire disponible, nombre de clients) et qu'il doit fonctionner pour n'importe qu'elles valeurs de ces paramètres.

En pratique, les algorithmes de model-checking sont basés sur un parcours de l'espace d'état. Pour des systèmes industriels cet espace est beaucoup trop grand pour pouvoir faire un parcours exhaustif. Par exemple, le système de contrôle des panneaux solaires d'un véhicule spatial comme l'ATV [CDD+12] est composé d'une vingtaine de blocs modèle SysML et donne lieu à un espace d'états trop grand pour être vérifié avec les techniques actuelles. Ce phénomène d'explosion de l'espace d'état a deux sources distinctes : la taille du système de transitions augmente exponentiellement d'une part avec le nombre de variables (états et horloges), d'autre part avec le nombre de composants des systèmes dans le cas où le système est concurrent (très courant). Le problème d'explosion combinatoire est une limitation qui empêche l'application des méthodes de model-checking sur des modèles de taille industrielle. Plusieurs techniques cherchent à contourner cette limitation. Nous avons cité les techniques d'abstraction [DGG93], les techniques de model-checking symboliques [Mcm93] (e.g. à travers l'utilisation de *diagrammes de décision binaires* [Bry86]) et la réduction d'ordre partiel [God90].

Il existe aussi d'autres pistes qui peuvent être explorées pour arriver à résoudre le problème de l'explosion de l'espace d'états [CES09]. Par exemple combiner les techniques de model-checking avec d'autres techniques de vérification comme l'analyse statique ou la preuve, trouver des algorithmes efficaces ou encore utiliser la symétrie pour réduire l'espace d'états.

Une autre limitation est celle de l'interprétation des contre-exemples générés par le model-checker. Ces contre-exemples détaillent le scénario qui mène à un état du système où la propriété spécifiée n'est pas satisfaite, des travaux de Clarke et autres. [CGJV03] traite la question du contre-exemple, mais ici les auteurs s'intéressent à l'extraction des abstractions à partir d'un contre-exemple dans le cadre du model-checking symbolique. Les travaux de Groce et autres. [GCKS06] s'intéressent à l'extraction à partir d'un contre-exemple de traces correctes qui permettraient de comprendre l'erreur.

Deuxième partie

Contributions

" Les erreurs ...

Pour un architecte, c'est dramatique, car il devra vivre avec toute sa vie et ses œuvres resteront le témoignage de son incompétence même après sa mort.

Pour un médecin, c'est angoissant, car une erreur médicale peut briser une carrière à moins qu'il n'enterre lui-même son ancien patient.

Pour un informaticien, ce n'est pas grave, car personne ne s'en rendra compte "

— Eric B. Motta Jr.

5. Translation des diagrammes de séquence avec des annotations MARTE en modèles DTPNs

Sommaire

Chapitre 5

Translation des diagrammes de séquence avec des annotations MARTE en modèles DTPNs

5.1 Introduction

Nous présentons, dans ce chapitre une approche de vérification formelle des spécifications en profile UML MARTE basée sur une sémantique de vrai parallélisme pour exprimer les comportements concurrents et parallèles des systèmes temps-réel embarqués. Notre objectif principal est de surmonter le problème de l'explosion combinatoire d'espaces d'états et nombre d'horloges des comportements possibles des systèmes critiques afin de rendre leur vérification formelle par model-checking est possible. Ainsi, nous proposons en premier une définition formelle aux diagrammes de séquence annotés par des contraintes temporelles en utilisant le profile UML MARTE. Ensuite, les règles graphique et formelle de translation des diagrammes de séquence UML2, les constructions de base et les fragments combinés, avec des annotations UML MARTE en modèles de réseaux de Petri temporels avec durées d'actions équivalents sont détaillées.

5.2 Approche pour la vérification formelle des spécifications UML MARTE

Dans ce travail, nous proposons une approche pour translater une spécification semi-formelle écrit dans un modèle UML MARTE en modèle temporel formel spécifique, qui est le réseau de Petri temporel avec durée d'actions afin de supporter la vérification formelle. La figure 1.5 donne un aperçu général de l'architecture de l'approche proposée. Tout d'abord, nous supposons que le comportement du système est décrit par le diagramme de séquence UML2. Ensuite, nous utilisons les annotations de profil MARTE pour exprimer le temps d'exécution, les contraintes de temps comme délais, latence et durées d'actions. Après ça, le diagramme de séquence avec des annotations MARTE est traduit opérationnellement en modèle DTPN afin de vérifier les propriétés spécifiques du système comme l'accessibilité sur le graphe de marquage associé. Ce graphe de marquage est vu comme un système de transitions étiquetées maximales (STEM). Puis, le modèle DTPN validé est automatiquement traduit en structure daTA, qui se base sur une sémantique opérationnelle de vraie concurrence. Après, l'automate daTA généré est soumis à une étape de vérification par model-checking, qui est basée sur la génération de l'espace d'états et la vérification des propriétés de sûreté et de vivacité du système sur cet espace. Les propriétés peuvent être génériques, relatives à la bonne construction du modèle ou bien spécifiques, écrites par le modélisateur en logiques temporelles pour contrôler la cohérence de la modélisation.

Pour chacune de ces deux approches, après la vérification d'une propriété donnée, une réponse positive ou négative est rendue. Dans le cas où la propriété n'est pas satisfaite, un contre-exemple est généré.

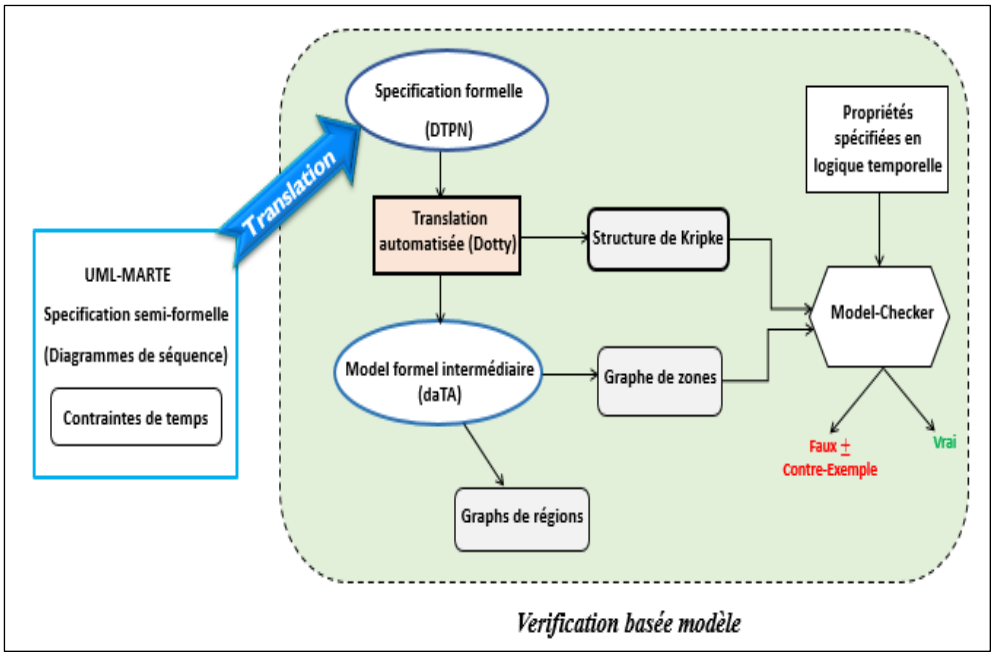


FIGURE 5.1– Processus de vérification

5.3 Formalisation des diagrammes de séquence annotés par le profil MARTE

Les notations basées sur les diagrammes, telles que UML sont classées comme semi-formelles ou en d'autres termes, des notations à syntaxe formelle et à sémantique ouverte à différentes interprétations. Dans la littérature, il existe plusieurs travaux de recherche qui ont porté sur la formalisation formelle des modèles UML MARTE, en particulier les diagrammes de séquence [StoS04, CK04, EFS+05, HHR+05, Kus06, Ham06, FTJ+07, HM08, BM10, CG19]. Parmi ces travaux, nous trouvons l'approche présentée dans [BM10] est intéressante en tant que traduction formelle. Dans ce travail, les auteurs ont proposé une définition formelle au digramme de séquence UML2 ,et des règles de transformation formelles pour obtenir un modèle de réseau de Petri temporel coloré avec arcs inhibiteurs (TCPNIA) équivalent au diagramme de séquence UML2 donné.

Néanmoins, dans le diagramme de séquence le traitement de transmission entre deux (02) objets communicants est spécifié comme une transition unique avec une abstraction des différentes primitives de communication, c'est-à-dire processus d'envoi, de transmission et de réception. De plus, cette approche modélise la transmission sans autre considération des détails relatifs à son exécution comme le temps d'exécution et la latence.

Pour surmonter ces limitations et afin de prendre en compte les contraintes de temps et les durées d'exécution d'actions, nous proposons une translation formelle des diagrammes de séquence UML2 avec des notations MARTE en modèle de réseaux de Petri temporels avec durées d'actions. Outre la translation formelle et la considération de contraintes de temps et les durées explicites des

actions, nous étudions la spécification de l'exécution des comportements parallèles. À cette fin, le modèle DTPN basé sur une sémantique de vraie concurrence [BSI18] est utilisé pour obtenir des automates temporels avec durée d'action (daTAs) (représentation sémantique). Dans daTA, on avait la possibilité d'exprimer les comportements parallèles et de supporter à la fois les contraintes temporelles, l'exécution potentielle des actions qui durent dans le temps, la non-atomicité structurelle et temporelle des actions et la notion d'urgence. Cette structure permet la vérification des propriétés liées à l'évolution parallèle des actions sous des contraintes de temps. Nous notons que certaines propriétés liées à l'accessibilité peuvent être vérifiées à l'aide des outils model-checker par exemple, KRONOS et UPPAAL [DTY95, LP97]. Cependant, pour les propriétés traitant les comportements de vrai parallélisme le model-checker FOCOVE peut être utilisé [SBB08].

Définitions formelles

En des termes plus formels, nous définissons un diagramme de séquence UML2 annoté par le profil MARTE comme suit :

Définition 1 : *MARTE Sequence Diagram* (MARTESD). Un diagramme de séquence MARTE est défini par le n-uplet *MARTESD = < N, O, E, <$_g$, Msg, Act, D, Tc, λ, S, T, Pre, Post, CF)* où :

- *N* est un ensemble des noms de diagramme ;
- *O* est un ensemble fini des objets ;
- $E = \bigcup_{oi \in O} E_i$ est un ensemble des événements tel que $\forall O_i, O_j \in O$ et $O_i \neq O_j, E_i \cap E_j = \varnothing$;
- $<_g = < \cup \{ \bigcup_{i,j \in O, i \neq j} <_{i,j} \}$ est un ordre globale, où :
  - $<= \bigcup_{i \in O}$ est un ensemble d'ordres partiels sur les événements de E_i tel que : $\forall i \in O$, $<_i \subseteq E_i \times E_i$;
  - $<_{i,j}$ est défini un ordre (partiel ou totale) entre les deux événements e, e' tel que : $\forall (e, e') \in <_{i,j}, ((e \in O_i \wedge e' \in O_j) \vee (e \in O_j \wedge e' \in O_i)) \wedge (e$ précède $e')$;
- *Msg* est un ensemble fini des étiquettes de messages ;
- *Act* est un ensemble des actions où chaque action $a \in Act$ peut être déclenchée par un événement. Selon le contexte de spécifications, nous pouvons distinguer différents types d'actions, par exemple: sending a synchronous message (Ssyn), sending an asyn-chronous message (Sasyn), receiving a synchronous message (Rsyn), receiving an asynchronous message (Rasyn), sending a reply message (Sreply), receiving a reply message (Rreply), Transmitting a message (Trans), performing an activity (Activity), ou tout comportement à exécuter dans le système.
  Act= {Ssyn, Rsyn, Sasyn, Rasyn,...,Trans$_1$,... ,Trans$_n$, Activity$_1$,... , Activity$_n$,...} ;
- *D : Act* $\rightarrow \mathbb{T}$ est une fonction qui associe pour chaque action une durée qui peut être nulle ;

- *Tc : E →* $\mathbb{T} \times \mathbb{T}$ est une fonction qui associe un domaine de temps à chaque événement tel que : $\forall e \in E$, *Tc (e) = [d, d + t]* signifie que l'événement *e* doit être exécuté dans l'intervalle de temps *[d, d + t]* ;
- λ*: E →Act* une fonction d'étiquetage qui associe un nom d'action à chaque événement ;
  Pour chaque événement $e \in E$, *e* peut être défini par le 3-uple $\langle e, D\ (\lambda(e)), Tc\ (e)\rangle$.
  Par exemple, soit *e* un événement associe à l'action $a \in Act$ avec une durée égale 5 unités de temps et exécuté dans l'intervalle de temps [1, 3]. L'occurrence d'un tel événement est définie par $\langle e, 5, [1, 3]\rangle$. Une durée non nulle peut être associée à l'événement *e*. L'ensemble de tous 3-uple $\{\langle e, D\ (\lambda(e)), Tc\ (e)\rangle\}_{e \in E}$ est noté E^t. -
- *S* est l'ensemble de tous les états possibles avec $S = \bigcup_{o_i \in o} S_i$, où S_i est l'ensemble d'états d'un objet o_i. $S_i = \bigcup_{e \in E_i} S_e$ tel que :
  – $\forall\ e \in E, S_e = \{ s_e^0, s_e^1 \}$;
  – $\forall \langle e, d, tc \rangle \in E^t$;
  – Si $e \in E_i$ est associé à l'action d'envoi alors $S_e = (s_e^0, s_e^1, s_e^{out})$;
  – Si $e \in E_i$ *est* associé à l'action de réception alors $S_e = (s_e^0, s_e^1, s_e^{in})$.

  De même aux événements, les différents objets ne peuvent pas partager les états, $S_i \cap S_j = \varnothing$ *pour* $o_i \neq o_j \in O$;

- *T* est un ensemble des transitions, tel que pour chaque transition $t \in T$ est associée à un événement $e \in E$, qui a la forme$\langle e, D\ (\lambda(e)), Tc\ (e)\rangle$. Les arcs reliant les états aux transitions sont étiquetés par la fonction *Pre*, tandis que les arcs reliant les transitions aux états sont étiquetés par la fonction *Post* ;
- $Pre: N \rightarrow 2^{S \times T}$ est une fonction d'entée qui associée pour chaque diagramme de séquence *sd*, un ensemble d'arcs, où $(s_i, t_j) \in Pre(sd)$ définit l'arc de s_i à t_j.
- $Post: N \rightarrow 2^{S \times T}$ est une fonction de sortie qui associée pour chaque diagramme de séquence *sd*, un ensemble d'arcs, où $(s_i, t_j) \in Post(sd)$ définit l'arc de t_j à s_i.

  Pour un diagramme de séquence *sd* donné :
  – L'ensemble d'arcs d'entrée de transition $t \in T$ est dénoté : $^\circ t = \{(s,t) \in Pre(sd)/s \in S\}$;
  – L'ensemble d'arcs de sortie de transition $t \in T$ est dénoté : $t^\circ = \{(s,t) \in Post(sd)/s \in S\}$;
  – L'ensemble d'arcs d'entrée de state $s \in S$ est dénoté $^\circ s = \{(s,t) \in Post(sd)/t \in T\}$;
  – L'ensemble d'arcs de sortie de state $s \in S$ est dénoté $s^\circ = \{(s,t) \in Pre(sd)\ /\ t \in T\ \}$.
- *CF* est un fragment combiné défini par un type d'opérateur et d'un ou plusieurs opérandes.

  – *Type= {seq, alt, par, loop, strict, opt, break, ref, critical, neg, assert, ignore, consider};*
  – *Op* est un ensemble fini d'opérandes ;

- *Garde* est une expression booléenne ou temporel associée à un opérande ou à un fragment d'interaction ;
- *Frag* est un ensemble de sous fragments combinés à l'intérieur de n^{ième} opérande de k^{ième} fragment d'interaction.

Exemple illustratif

La figure 5.2 décrit une représentation graphique d'une transition temporelle *t* dans un diagramme de séquence, une telle transition est représentée comme suit :
t= $\{\langle e, 0, [1,3]\rangle, \langle e'', 4, [0,2]\rangle, \langle e', 0, [1,5]\rangle\}$ cette notation est motivée par :
L'action d'envoi définie par l'événement *e* de l'objet O_1 a une durée nulle (*d1= 0*) et une contrainte de temps [1, 3]. L'action de réception définie par l'événement e' de l'objet O_2 a une durée nulle (*d2=0*) et contrainte de temps [1, 5]. Pour modéliser la transmission de message entre les deux objets O_1 et O_2 aux événements *e* et e', un événement e'' qui correspond à l'action de transmission est créé avec une durée égale 4 unités de temps et qui peut être retardée 2 unités de temps. Puisque nous considérons l'intervalle de contrainte [0, 2]. Dans ce cas, dès que le message est envoyé, sa durée de transmission sera entre 4 et 6 unités de temps.

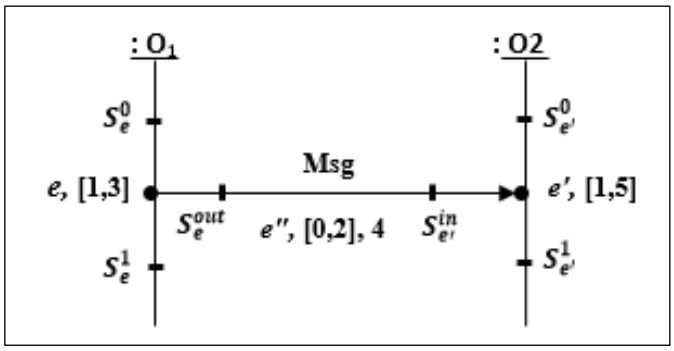


FIGURE 5.2– Représentation d'une transmission asynchrone

5.4 Translation des diagrammes de séquence UML MARTE en modèles DTPNs

Dans cette section, nous présentons les règles de translation des DSMARTE en modèles DTPNs, et nous expliquons à travers des exemples la traduction entre les éléments d'entrée d'un diagramme de séquence annotés par le profil MARTE et les éléments de sortie de modèle DTPN. Nous considérons les différents types de transmission entre les objets communicants, les durées des activités souhaitées à exécuter et les contraintes de temps imposées sur l'envoi et la réception des messages échangés. En utilisant le modèle DTPN, les durées des actions sont fixées une fois pour toutes au moment de la validation du système. Le cas où les actions ont des durées choisies parmi un intervalle *[m, M]* n'est pas considéré. Ainsi, nous notons que le stéréotype *TimeConstraint* présenté dans le chapitre2 (section 2.4) est exprimé par l'intervalle de temps *[min, max]*.

5.4.1 Constructions de base

Dans une interaction, selon le positionnement des évènements sur la ligne de vie associée à chaque objet et le type des messages transmis entre les objets participants dans l'interaction, on distingue deux (02) types de transmission entre les objets communicants :

— **Transmission inter-objets** : Les deux événements de message échangé appartiennent à deux objets différents.

— **Transmission intra-objets** : Les deux événements de message échangé appartiennent au même objet.

De fonçon formelle, on peut définir ces deux types de transmission comme suit :

Définition 2 :

Pour un diagramme de séquence *sd* donné, soit $\langle e_1, d_1, ct_1\rangle$, $\langle e_2, d_2, ct_2\rangle$ deux transition différentes associées à une transmission donnée telle que :

1- Si $O_i, O_j \in O$ et $i \neq j$ tel que $e_1 \in E_i$ et $e_2 \in E_j$, alors les transitions représentent une évolution externe entre les deux objets O_i et O_j. Elle est appelée une transmission *inter-objets*

2- Si $\exists\ O_i \in O$ tel que $e_1, e_2 \in E_i$ et $e_1 < e_2$ et $\nexists\ e \in E_i$ tel que $e_1 < e < e_2$, alors les transitions représentent une évolution locale dans l'objet O_i . Elle est appelée une transmission *intra-objets*. Dans ce cas $d_1 = d_2 = 0$ *et* $ct_1 = ct_2$ *= [0, 0]*.

5.4.1.1 Transmission Inter-objets

— **Transmission synchrone**

Les transmissions synchrones sont utilisées lorsque l'émetteur attend une réponse de récepteur pour continuer son exécution. Une transmission synchrone bloque la progression des opérations de son émetteur jusqu'à ce que le message échangé soit reçu (*synchronisation faible*) ou jusqu'à réception de la réponse par l'émetteur (*synchronisation forte*).

Nous considérons l'exemple de la figure 5.3, qui représente une transmission synchrone entre les deux objets O_1 et O_2. Dans cette section, nous détaillons comment traduire les différents éléments de ce diagramme de séquence (modèle source) en éléments de modèle DTPN (modèle cible), qui doit spécifier le même comportement du système.

D'après la figure 5.3, la transmission synchrone représente une interaction entre deux objets O_1 et O_2 associés à deux événements e_1 et e_2 nécessairement différents. La traduction de cette transmission en modèle DTPN équivalent est basée sur l'interprétation de chaque événement et son action associée dans le diagramme avec la prise en compte des deux états (avant et après) de chaque événement, ainsi que les contraintes temporelles imposés sur chaque élément. Pour voir comment construire les DTPNs équivalents, nous avons divisé la transmission synchrone en deux (02) étapes principales : transmission synchrone d'envoi et transmission synchrone de réception de réponse.

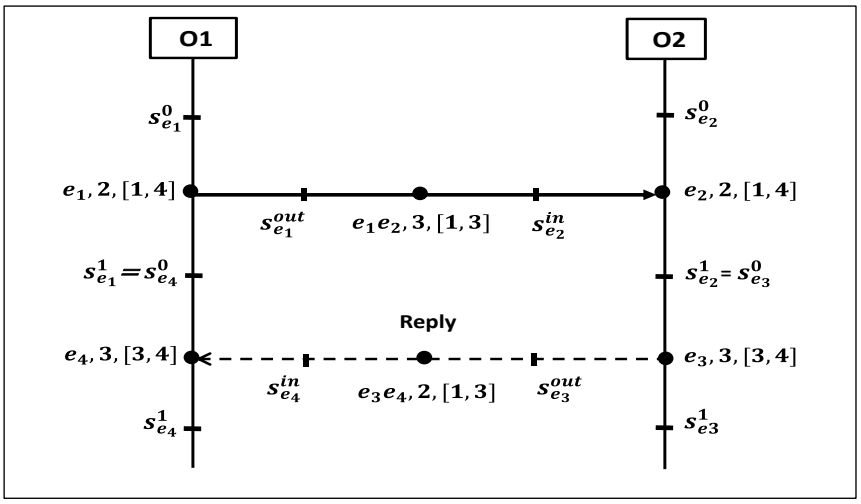


FIGURE 5.3– Transmission synchrone

— **Étape 1**

L'objet O_1 lance une opération d'envoi de transmission à l'événement e_1 vers l'objet O_2, qui est contrainte par l'intervalle de temps [1, 4] et la durée d'action *a* égale à 2 unités de temps. L'évènement e_1 est traduit à une transition de réseau de Petri t_{e1} caractérisée par un intervalle de temps [1, 4] et une action de durée d'exécution *a=2*. Les deux états avant et après l'événement e_1 sont traduits en deux places $s^0_{e_1}$, $s^1_{e_1}$de réseau DTPN. Ensuite, nous ajoutons les arcs qui relient les places aux transitions ou les transitions aux places. Cette traduction est illustrée par la figure 5.4.

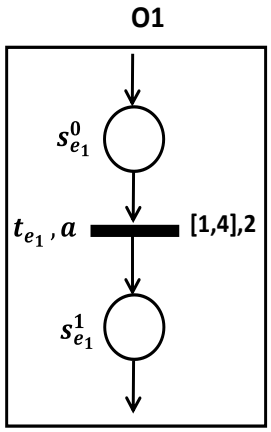


FIGURE 5.4– DTPN correspondant à l'action d'envoi

L'objet O_2 reçoit l'action d'envoi de transmission à l'événement e_2, avec une durée égale à 2 unités de temps, qui peut être retardée de 4 unités de temps. Puisque, nous considérons l'intervalle de contrainte [1, 4]. Le DTPN correspondant est représenté par la figure suivante.

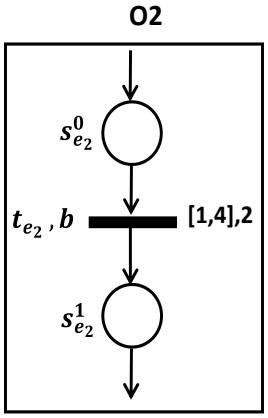


FIGURE 5.5– DTPN correspondant à l'action de réception

Pour représenter l'action de transmission intermédiaire entre l'objet O_1 et objet O_2, nous créons un événement nommé e_1e_2 et deux états $s^{out}_{e_1}$, $s^{in}_{e_2}$. L'événement e_1e_2 modélise l'action qui prend dans notre cas une durée comprise entre 4 et 6 unités de temps. Pour considérer cette caractéristique, on suppose l'intervalle de contrainte [1, 3] et une durée d'action de transmission égale à 3 unités de temps. La figure 5.6 montre la construction de réseau DTPN correspondant.

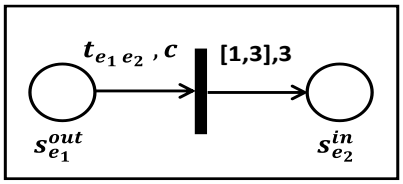


FIGURE 5.6– DTPN correspondant à l'action de transmission

Après avoir ajouté les arcs reliant les trois sous-réseaux DTPNs, deux sous-réseaux sont liés à deux sous-réseaux, la construction résultante complète de cette étape est donnée par la figure 5.7.

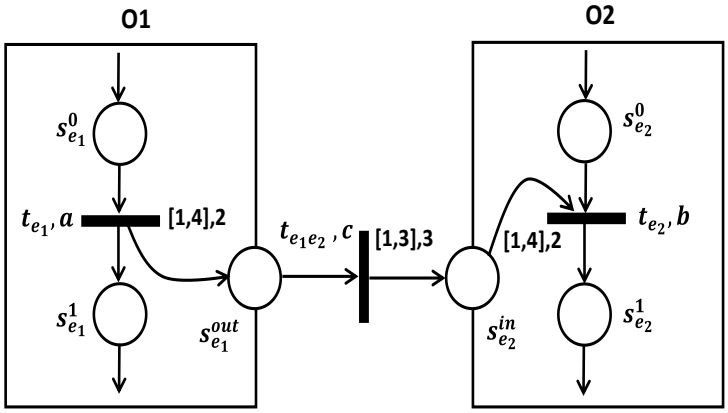


FIGURE 5.7– DTPN correspondant à la transmission d'envoi

Formellement, considérons *S et T* sont les plus petits ensembles vérifiant les conditions de la construction suivante :

Soit $\{t_{e_1}, t_{e_1e_2}, t_{e_2}\} = \{\langle e_1, 2, [1,4]\rangle, \langle e_1e_2, 3, [1,3]\rangle, \langle e_2, 2, [1,4]\rangle\}$ un ensemble des transitions modélise la transmission d'envoi entre les deux objets O_1 et O_2 .

Soit $e_1 \in E_1$ *et* $e_2 \in E_2$ deux événements, tel que $\lambda(e_1)$ =*Ssyn,* $\lambda\ (e_2)$ = *Rsyn,* avec $(e_1, e_2) \in <_{i,j}$, $\{s^0_{e_1}, s^1_{e_1}, s^{out}_{e_1}, s^0_{e_2}, s^1_{e_2}, s^{in}_{e_2}\} \subseteq S$ *et* $\{t_{e_1}, t_{e_2}, t_{e_1e_2}\} \subseteq T$ alors :

$^\circ t_{e_1} = \{s^0_{e_1}\}$; $\quad t_{e_1}{}^\circ = \{s^1_{e_1}, s^{out}_{e_1}\}$;

$^\circ t_{e_2} = \{s^{in}_{e_2}, s^0_{e_2}\}$; $\quad t_{e_2}{}^\circ = \{s^1_{e_2}\}$;

$^\circ t_{e_1e_2} = \{s^{out}_{e_1}\}$; $\quad t_{e_1e_2}{}^\circ = \{s^{in}_{e_2}\}$.

— **Étape 2**

En raison de sa similitude avec la transmission d'envoi dans l'étape 1, il est possible d'appliquer le même schéma de traduction dans cette étape, comme conséquent de la réception de transmission par l'objet O_2. L'objet O_2 , tant que le récepteur répond en exécutant le comportement correspondant à cette action de transmission.

Après le traitement de l'opération demandée, l'objet O_2 renvoie la réponse de transmission à l'événement e_3 vers l'objet O_1. L'événement e_3 est caractérisé par une durée égale à 3 unités de temps et un intervalle de temps [3, 4]. L'interprétation de ce ci est représenté par le réseau DTPN de la figure 5.8.

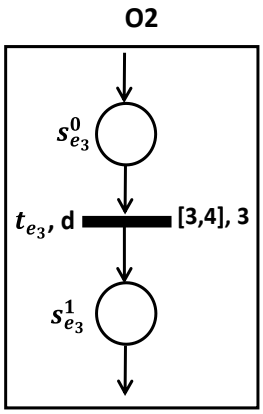


FIGURE 5.8– DTPN correspondant à l'action d'envoi de réponse

Les objets répondent aux messages qui sont produits par des objets exécutant des actions de communication. Ainsi, l'objet O_1 reçoit la réponse de transmission à l'événement e_4, qui a 3 unités de temps comme durée, et [3, 4] comme contrainte de temps. Le réseau DTPN correspondant est illustré par la figure 5.9.

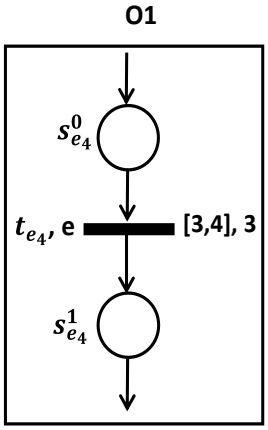


FIGURE 5.9– DTPN correspondant à l'action de réception de réponse

Pour modéliser l'action de transmission de réponse, entre l'actions d'envoi à l'événement e_3 et l'action de réception à l'événement e_4, nous créons un événement nommé e_3e_4 et deux états $s^{out}_{e_3}$, $s^{in}_{e_4}$. Cet événement prend dans ce cas une durée comprise entre 3 et 5 unités de temps. Puisque nous considérons l'intervalle de contrainte [1,3] et une durée d'action égale à 2 unités de temps. Le modèle DTPN correspondant est illustré dans la figure 5.10.

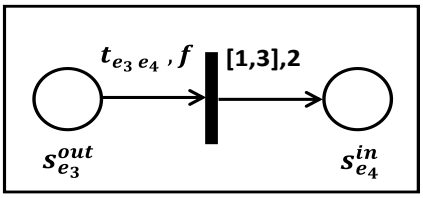


FIGURE 5.10– DTPN correspondant à l'action de transmission de réponse

À présent, les arcs pour composer les trois sous-réseaux de DTPNs sont ajoutés. Le réseau DTPN résultant est décrit par figure 5.11.

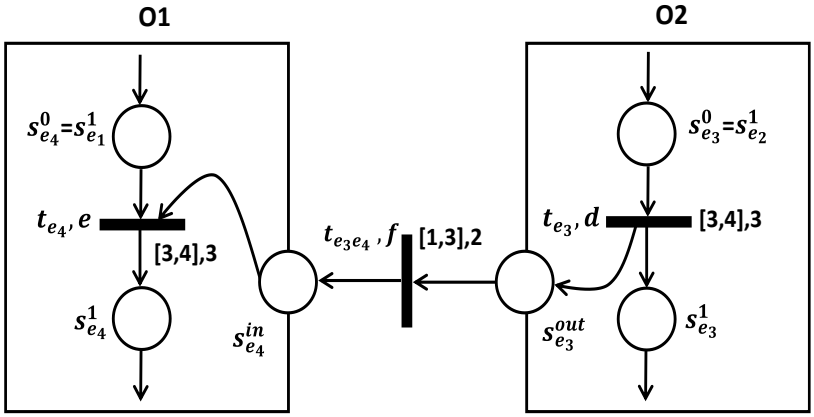


FIGURE 5.11– DTPN correspondant à la transmission de réponse

Formellement, considérons *S et T* sont les plus petits ensembles vérifiant les conditions de la construction suivante :

Soit $\{t_{e_3},\ t_{e_3e_4}, t_{e_4}\} = \{\langle e_3, 3, [3,4]\rangle, \langle e_3e_4, 2, [1,3]\rangle, \langle e_4, 3, [3,4]\rangle\}$ un ensemble des transitions modélise la transmission de réponse entre les deux objets $O_1\ et\ O_2$.

Soit $e_3 \in E_2$ *et* $e_4 \in E_1$ deux événements, tel que $\lambda(e_3)$= *Sreply*, $\lambda(e_4)$= *Rreply* avec $(e_3, e_4) \in <_{i,j}$, $\{s^0_{e_3}, s^1_{e_3}, s^{out}_{e_3}, s^0_{e_4}, s^1_{e_4}, s^{in}_{e_4}\} \subseteq S$ *et* $\{t_{e_3}, t_{e_4}, t_{e_3e_4}\} \subseteq T$ alors :

$^\circ t_{e_3} = \{s^1_{e_2}\}$; $\quad t_{e_3}{}^\circ = \{s^1_{e_3}, s^{out}_{e_3}\}$;

$^\circ t_{e_4} = \{s^{in}_{e_4}, s^1_{e_1}\}$; $\quad t_{e_4}{}^\circ = \{s^1_{e_4}\}$;

$^\circ t_{e_3e_4} = \{s^{out}_{e_3}\}$; $\quad t_{e_3e_4}{}^\circ = \{s^{in}_{e_4}\}$.

Enfin, la traduction complète de l'exemple de la figure 5.3 est représentée par figure 5.12.

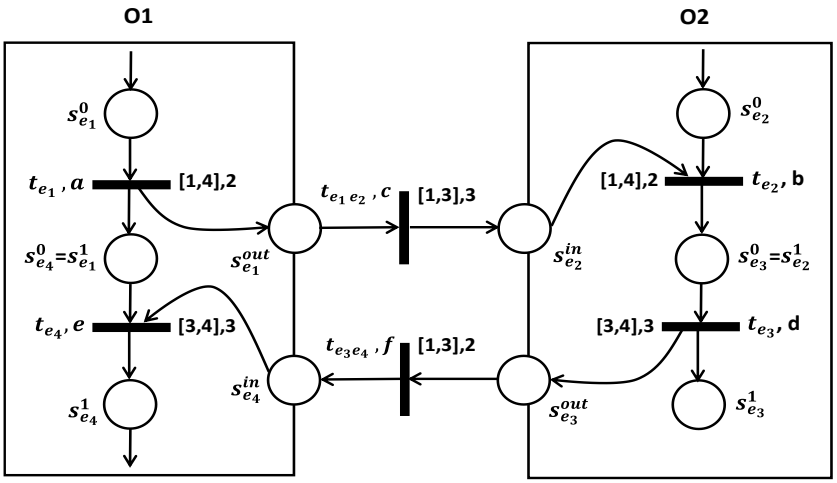


FIGURE 5.12– DTPN correspondant à la transmission synchrone

— **Transmission Asynchrone**

La transmission asynchrone est utilisée lorsque l'émetteur n'a pas besoin d'attendre la réponse du récepteur, il continue son exécution après l'envoi de message. Donc, La fonctionnalité de base est la même que la transmission synchrone faible. Par exemple, nous considérons le diagramme de séquence de la figure 5.13, qui représente la transmission d'un message asynchrone entre les deux objets O_1 et O_2 aux événements e_1 et e_2.

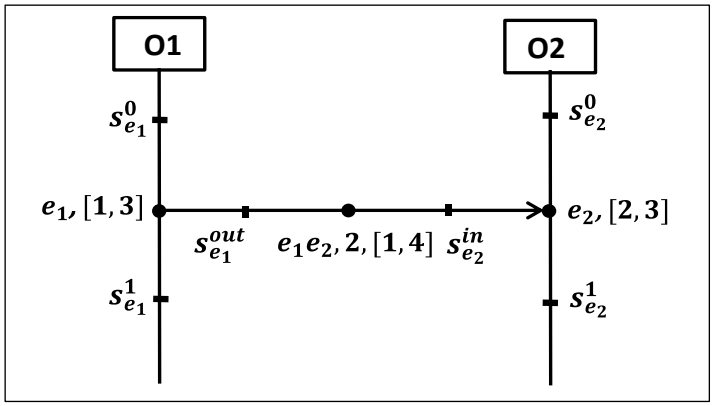


FIGURE 5.13– Transmission asynchrone

L'événement e_1 sur la ligne de vie associé à l'objet O_1, représente l'action d'envoi qui est instantanée (d=0) et avec contrainte de temps [1,3]. L'événement e_1 est interprété comme une transition t_{e_1} de réseau DTPN, qui contraint par l'intervalle de temps [1,3], ses deux états s_e^0, s_e^1 sont traduits en deux places de réseau DTPN avec les même noms, contrainte de temps et durée d'exécution. De plus, nous ajoutons les arcs reliant les places aux transitions ou les transitions aux places. Cette translation est représentée par la figure 5.14.

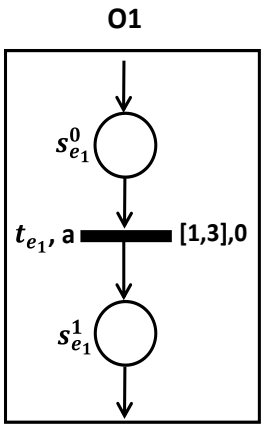


FIGURE 5.14– DTPN correspondant à l'action d'envoi

L'action de réception dans l'objet O_2 est représentée par l'événement e_2 et le message de réception doit être dans un intervalle de temps compris entre 2 et 3 unités de temps. L'interprétation de cette partie est illustrée par le DTPN donné par la figure 5.15.

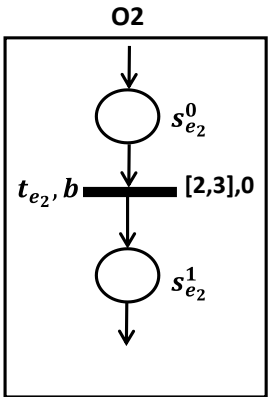


FIGURE 5.15– DTPN correspondant à l'action de réception

Pour représenter l'opération de transmission asynchrone entre l'action d'envoi à l'évènement e_1 et l'action de réception à l'évènement e_2 nous ajoutons un événement nommé e_1e_2 entre les deux événements e_1 et e_2. Cet événement prend dans notre cas une durée comprise entre 3 et 6 unités de temps. Puisque nous considérons l'intervalle de contrainte [1,4] et une durée de l'action de transition $t_{e_1e_2}$ égale à 2 unités de temps. La figure 5.16 illustre la structure de DTPN résultant.

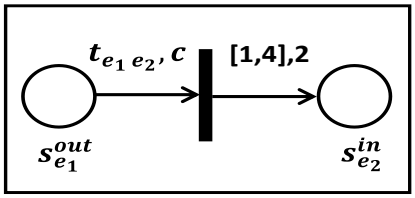


FIGURE 5.16– DTPN correspondant à l'action de transmission

Pour assurer une transmission asynchrone entre les deux objets O_1 et O_2 , nous ajoutons deux arcs, le premier arc de transition t_{e_1} à l'état d'entrée de transition $t_{e_1e_2}$, et le deuxième arc relie l'état de sortie de cette dernière transition à transition t_{e_2}. Comme le montre la figure 5.17, nous obtenons la construction complète de DTPN associée au diagramme de séquence de la figure 5.13.

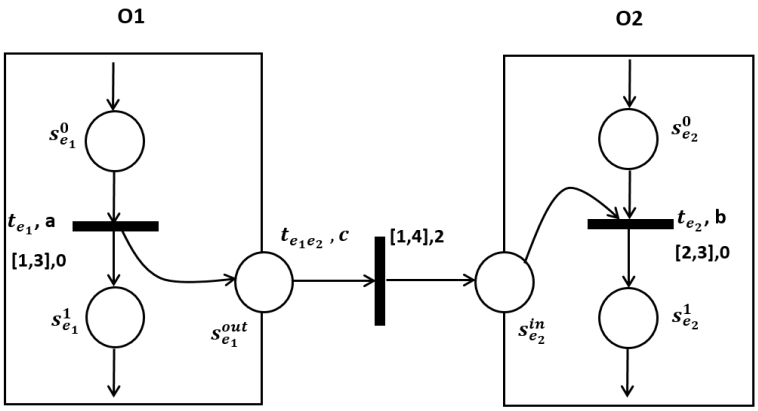


FIGURE 5.17– DTPN correspondant à la transmission asynchrone

Formellement, considérons *S* et *T* sont les plus petits ensembles vérifiant les conditions de la construction suivante :

Soit $\{t_{e_1}, t_{e_1e_2}, t_{e_2}\} = \{\langle e_1, 0, [1,3]\rangle, \langle e_1e_2, 2, [1,4]\rangle, \langle e_2, 0, [2,3]\rangle\}$ un ensemble des transitions modélise la transmission asynchrone entre les deux objets O_1 *et* O_2 .

Soit $e_1 \in E_1$ *et* $e_2 \in E_2$ deux événements, tel que $\lambda(e_1)$=*Sasyn*, $\lambda(e_2)$= *Rasyn avec* $(e_1, e_2) \in \prec_{i,j}$, $\{s_{e_1}^0, s_{e_1}^1, s_{e_1}^{out}, s_{e_2}^0, s_{e_2}^1, s_{e_2}^{in}\} \subseteq S$ *et* $\{t_{e_1}, t_{e_2}, t_{e_1e_2}\} \subseteq T$ alors :

$^\circ t_{e_1} = \{s_{e_1}^0\}$; $t_{e_1}{}^\circ = \{s_{e_1}^1, s_{e_1}^{out}\}$;

$^\circ t_{e_2} = \{s_{e_2}^{in}, s_{e_2}^0\}$; $t_{e_2}{}^\circ = \{s_{e_2}^1\}$;

$^\circ t_{e_1e_2} = \{s_{e_1}^{out}\}$; $t_{e_1e_2}{}^\circ = \{s_{e_2}^{in}\}$.

5.4.1.2 Transmission intra-objets

Une transmission intra-objets est une auto-transmission dans un diagramme de séquence dans laquelle les objets émetteur et récepteur d'un message sont les mêmes. Dans ce type de transmission, si les deux événements e et e' appartiennent au même objet O_i alors l'événement e' suit immédiatement l'événement e sans autre événement intermédiaire, la figure 5.18 montre un tel exemple.

Dans le diagramme de séquence, les actions d'envoi et de réception aux événements e et e' respectivement $\langle e, d_e, tc_e\rangle$ et $\langle e', d_{e'}, tc_{e'}\rangle$ sont instantanés et sans contraintes de temps ($d_e = d_{e'} = 0$ *et* $tc_e = tc_{e'} = [0,0]$). L'action de transmission à l'événement ee' a 3 unités de temps pendant l'intervalle [0,2]. Dans ce qui suit, nous détaillons comment ce transmission est translatée en un réseau DTPN équivalent.

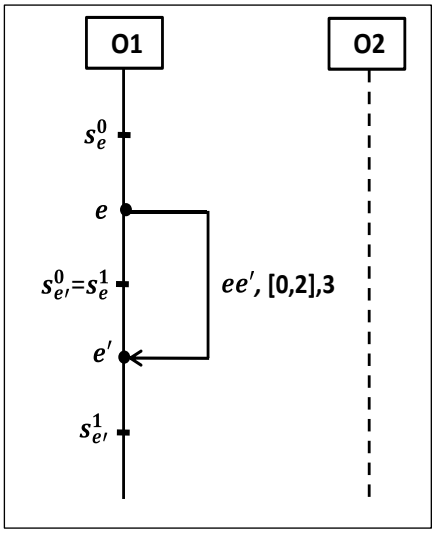


FIGURE 5.18– Auto-transmission

Sur la ligne de vie de l'objet O_1, l'action d'envoi à l'événement e est traduite en transition t_e de DTPN sans durée et contrainte de temps. Les deux états s_e^0, s_e^1 de cet événement e sont traduits en deux places s_e^0 et s_e^1 avec deux arcs, chacun reliant une place à la transition t_e. La Figure 5.19 représente cette construction.

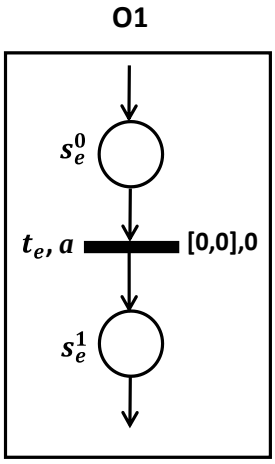


FIGURE 5.19– DTPN correspondant à l'action d'envoi

L'action de réception à l'événement e' a les mêmes caractéristiques de l''action d'envoi à l'événement e . Une construction similaire peut être générée en utilisant une transition $t_{e'}$ et deux places $s_{e'}^0$ et $s_{e'}^1$ avec deux arcs, le premier relie $s_{e'}^0$ à $t_{e'}$, et le deuxième relie $t_{e'}$ à $s_{e'}^1$. Le sous réseau DTPN correspondant est décrit par la figure 5.20.

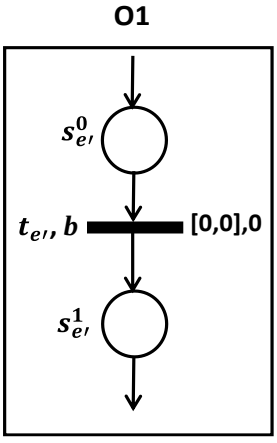


FIGURE 5.20– DTPN correspondant à l'action de réception

Pour modéliser l'opération d'action de transmission entre l'action d'envoi et l'action de réception dans le même objet O_1, nous ajoutons un événement nommé ee' entre les deux événements e et e', et qui prend dans notre cas une durée comprise entre 3 et 5 unités de temps. Pour prendre en compte cette caractéristique on considère l'intervalle [0,2] et une durée d'action associée à la transition $t_{ee'}$ égale à 3.

Le DTPN correspondant est donné par la figure 5.21. L'action de transmission à l'événement ee' se traduit à une transition $t_{ee\prime}$ avec une durée d=3 et une contrainte de temps [0,2]. Les deux places de cette transition sont la place de sortie s_e^1 de transition t_e et la place d'entrée $s_{e\prime}^0$ de transition $t_{e\prime}$. De plus, deux arcs sont créés, chaque arc relié l'une de ces places à la transition $t_{ee\prime}$.

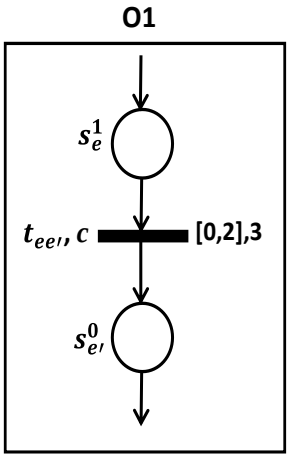


FIGURE 5.21– DTPN correspondant à l'action de transmission

Pour relier l'opération d'action de transmission aux deux parties précédentes de la transmission globale, nous ajoutons deux arcs, le premier arc de la place de sortie s_e^1 de la transition t_e vers la transition $t_{ee\prime}$ et le deuxième arc de la transition $t_{ee\prime}$ vers la place d'entrée $s_{e\prime}^0$ de la transition $t_{e\prime}$. La construction finale de réseau DTPN correspondant à l'exemple de ce type de transmission (figure 5.18) est décrite par la figure 5.22.

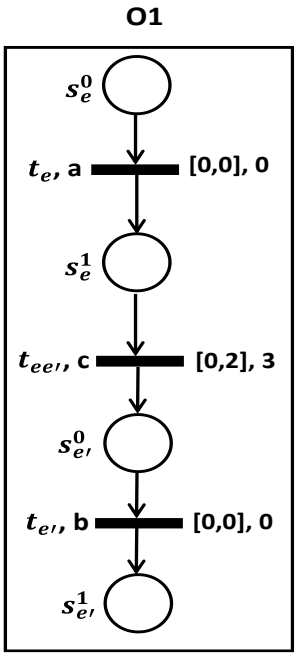


FIGURE 5.22– DTPN correspondant à auto-transmission

Formellement, considérons *S* et *T* sont les plus petits ensembles vérifiant les conditions de la construction suivante :

soit $T= \{t_e, t_{ee'}, t_{e'}\} = \{\langle e, 0, [0,0]\rangle, \langle ee', 3, [0,2]\rangle, \langle e', 0, [0,0]\rangle\}$ un ensemble des transitions modélise l'auto transmission entre les deux évènement e et e' dans l'objets O_1 .

Soit $e, e' \in E_1$, deux événements, tel que $\lambda(e)$=*Activity₁*, $\lambda(e')$ = *Activity₂* avec $(e, e') \in <_i$, $\{s_e^0, s_e^1, s_{e'}^0, s_{e'}^1\} \subseteq S$ *et* $\{t_e, t_{e'}, t_{ee'}\} \subseteq T$ alors :

$^\circ t_e = \{s_e^0\}$; $t_e^\circ = \{s_e^1\}$;

$^\circ t_{e'} = \{s_{e'}^0\}$; $t_{e'}^\circ = \{s_{e'}^1\}$;

$^\circ t_{ee'} = \{s_e^1\}$; $t_{ee'}^\circ = \{s_{e'}^0\}$.

Dans les différents cas de transmissions (inter-objets et intra-objets), chaque objet dans le diagramme de séquence a un seul état initial et a un seul état final sont définis comme suit :

- Un état *s* est dit initial si seulement si $^\circ s = \emptyset$.
- Un état *s* est dit final si seulement si $s^\circ = \emptyset$.

5.4.2 Fragments combinés

Les fragments combinés sont un type important d'interactions, qui sont utilisés pour créer des interactions plus complexes. Les fragments combinés fournis par UML sont choisis dans ce travail pour l'analyse formelle parce qu'ils permettent de grouper des scénarios multiples et complexes dans un seul diagramme de séquence. Ainsi, ils permettent à plusieurs scénarios d'augmenter la puissance expressive de diagrammes de séquence par divers flux de contrôle tels que l'entrelacement et le branchement pour décrire le comportement complexe et concurrent des systèmes. Un fragment combiné est défini par un opérateur d'interaction et un nombre des opérandes correspondants d'interaction (un ou plusieurs opérandes). L'opérateur de l'interaction est le plus important pour définir la sémantique de diagramme de séquence. Il détermine le type d'instructions conditionnelles qui décrit le comportement de fragment combiné. Les opérandes d'un fragment combiné peuvent avoir des gardes (expression booléenne ou conditionnelle) sur elles comme dans le comportement alternatif (Alt) et le comportement itératif (Loop).

La méthode de traduction proposée est appliquée sur un diagramme de séquence UML2 avec des annotations de profil MARTE pour spécifier les comportements temporelles de système. En utilisant le profil MARTE, on peut spécifier les contraintes de temps sur un opérande ou un fragment combiné à l'aide de la balise "execTime" dans le stéréotype «ResourceUsage». Selon le contexte de la spécification de diagramme de séquence annoté par le profil MARTE, l'intervalle de contrainte de temps [a, b] associé à ce stéréotype peut prendre les valeurs suivantes :

$\forall\, a, b \in \mathcal{R}^*$:

1- Si l'exécution d'un opérande ou d'un fragment combiné est urgente alors l'intervalle de temps [a,b]= [0,0] ou [a,b]=[t, t'] avec t=t' *;*
2- Si l'exécution d'un opérande ou d'un fragment combiné n'est pas urgente alors l'intervalle de temps [a,b]= [0,∞] *;*
3- Si l'exécution d'un opérande ou d'un fragment combiné est spécifiée avec latence alors l'intervalle de temps [a,b]=[v, v+t] avec a≥0 et a ≤b.

Dans cette sous-section, nous donnons les règles de traduction des fragments combinés les plus utilisés tels que le comportement séquentiel faible (Seq), le comportement séquentiel stricte (Strict), le comportement parallèle (Par), le comportement alternatif (Alt), le comportement optionnel (Opt) et le comportement itératif (Loop). Les opérandes de fragments combinés sont traduits en sous-réseaux DTPNs en utilisant les règles de traduction des constructions de base précédentes. Puis ils sont intégrés dans le réseau DTPN résultant. Comme nous le verrons plus tard, après l'application des règles de traduction le modèle DTPN résultant doit modéliser le même comportement que le modèle source, diagramme de séquence mais dans une notation différente.

5.4.2.1 Fragment Seq

Le fragment combiné de séquence faible, défini par l'opérateur "Seq" est un fragment d'interaction représente un séquencement faible entre les comportements des opérandes de l'interaction. Le fragment contient deux opérandes ou plusieurs. Si aucun autre opérateur n'est présent dans le diagramme de séquence, un séquençage faible doit être appliqué aux fragments d'interaction. Le séquençage faible se réduit à un fragment parallèle lorsque ses opérandes se trouvent sur différents ensembles d'objets. Un exemple illustratif est représenté par la figure 5.23.

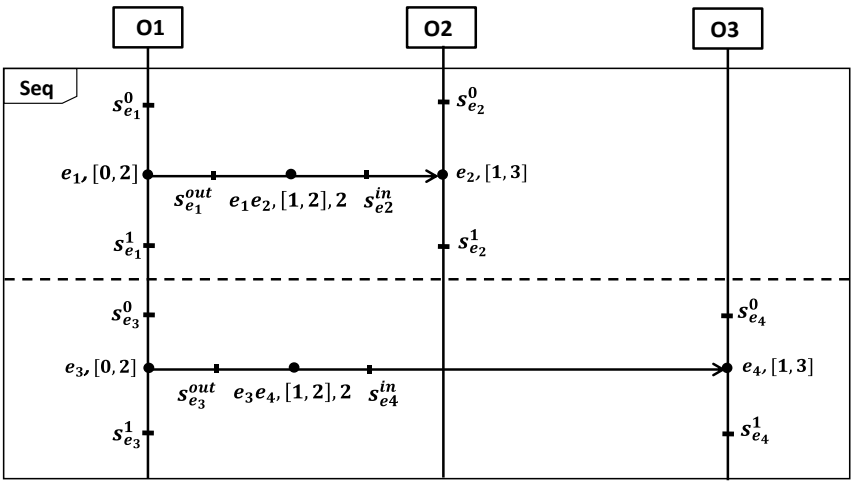


FIGURE 5.23– Fragments combinés type Seq

1. Comme première étape dans le processus de translation est d'initialiser le réseau DTPN en créant la première place $s^{0}_{f_i}$, qui lance l'exécution de DTPN. Cette place a le marquage initial comme un seul jeton, $M(s^{0}_{f_i}) = 1$;
2. Pour synchroniser tous les opérandes qui seront impliqués dans l'exécution du fragment d'interaction Seq, la transition $t^{0}_{f_i}$ est créée. Un arc reliant la place initiale $s^{0}_{f_i}$ à la transition $t^{0}_{f_i}$ est ajouté ;
3. En utilisant les règles de traduction détaillées dans la section précédente, les deux sous-réseaux DTPNs correspondants aux deux opérandes de fragment combiné Seq sont générés ;
4. Puisque, le but de cet opérateur est de permettre l'exécution d'un seul opérande seulement, autrement dit, les opérandes sont exécutés en exclusion mutuelle. En termes de réseau de Petri, les transitions liées aux opérandes doivent être en conflit. Ainsi, une place nommée *LP* (*Lunch Place*) est créée et reliée aux deux transitions initiales, t_{e_1} et t_{e_3} des deux sous-réseaux DTPNs précédents (respectivement) ;
5. La transition finale $t^{1}_{f_i}$ de réseau DTPN est créée et connectée par un arc à la dernière place de chaque sous-réseau DTPN ($s^{1}_{e_2}$ et $s^{1}_{e_4}$ respectivement) comme transition de sortie ;
6. La place $s^{1}_{f_i}$, qui représente la place finale de réseau DTPN est créé et connectée, en tant que place de sortie à la transition finale $t^{1}_{f_i}$.

La structure de réseau DTPN résultant de la traduction de fragment combiné Seq est donnée par la figure 5.24.

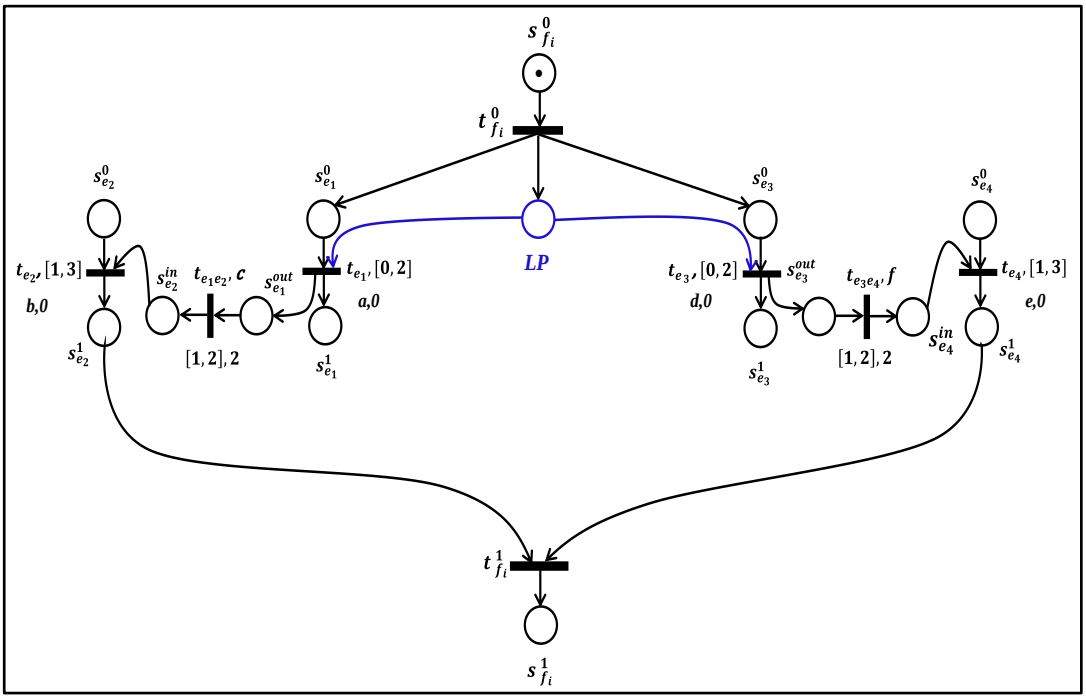


FIGURE 5.24– DTPN correspondant au fragment Seq

5.4.2.2 Fragment Strict

Le fragment combiné de séquence strict, noté par l'opérateur "Strict" est un fragment d'interaction inclut deux ou plusieurs opérandes. Les comportements des opérandes doivent s'exécuter dans un ordre donné. L'ordre dans chaque opérande est préservé. Un exemple de ce type de fragment combiné est représenté par la figure 5.25.

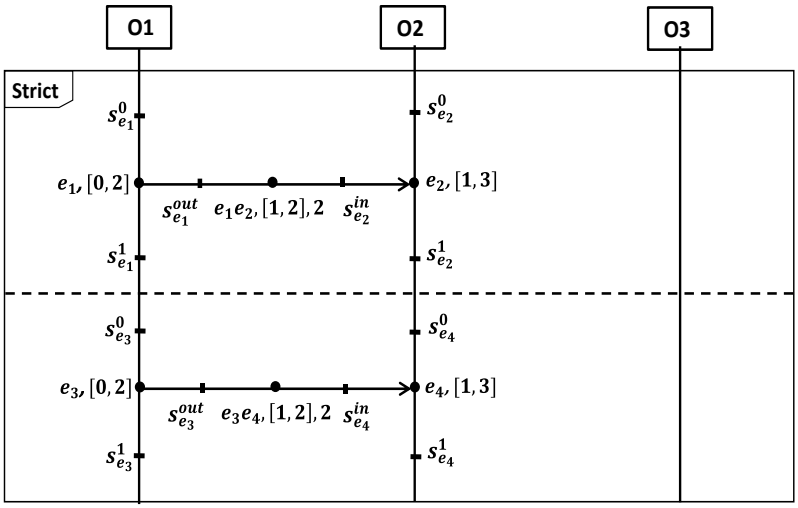


FIGURE 5.25– Fragments combinés type Strict

1- La place initiale $s^{\,0}_{f_i}$ est créée avec le marquage initial $M(s^{\,0}_{f_i}) = 1$;
2- La transition initiale $t^{\,0}_{f_i}$ de réseau DTPN globale est créée et connectée, comme transition de sortie à la place initiale $s^{\,0}_{f_i}$;
3- En utilisant les règles de traduction précédentes, les deux opérandes de fragment d'interaction Strict sont traduits en deux sous-réseaux DTPNs ;
4- Deux arcs sont générés, le premier arc connecté la transition initiale $t^{\,0}_{f_i}$ à la place initiale $s^{\,0}_{1}$ de sous-réseau DTPN à droite, et l'autre arc connecté la transition $t^{\,0}_{f_i}$ à la place initiale $s^{\,0}_{e_3}$ de sous-réseau DTPN à gauche ;
5- Pour respecter l'ordre strict imposé sur les opérandes de fragment combiné, un arc reliant la dernière transition t_{e_2} de premier sous-réseau DTPN à la place initiale $s^{\,0}_{e_3}$ de deuxième sous-réseau DTPN est créé ;
6- La transition finale $t^{\,1}_{f_i}$ de DTPN est générée et un arc relie la place finale $s^{\,1}_{e_4}$ de deuxième sous-réseau DTPN à la transition finale $t^{\,1}_{f_i}$ est ajouté ;
7- La place finale $s^{\,1}_{f_i}$ de DTPN est créée et un arc relie la transition $t^{\,1}_{f_i}$, en tant que transition d'entrée à la place $s^{\,1}_{f_i}$ est ajouté.

La figure 5.26 illustre la construction complète du DTPN correspondant au fragment d'interaction Strict.

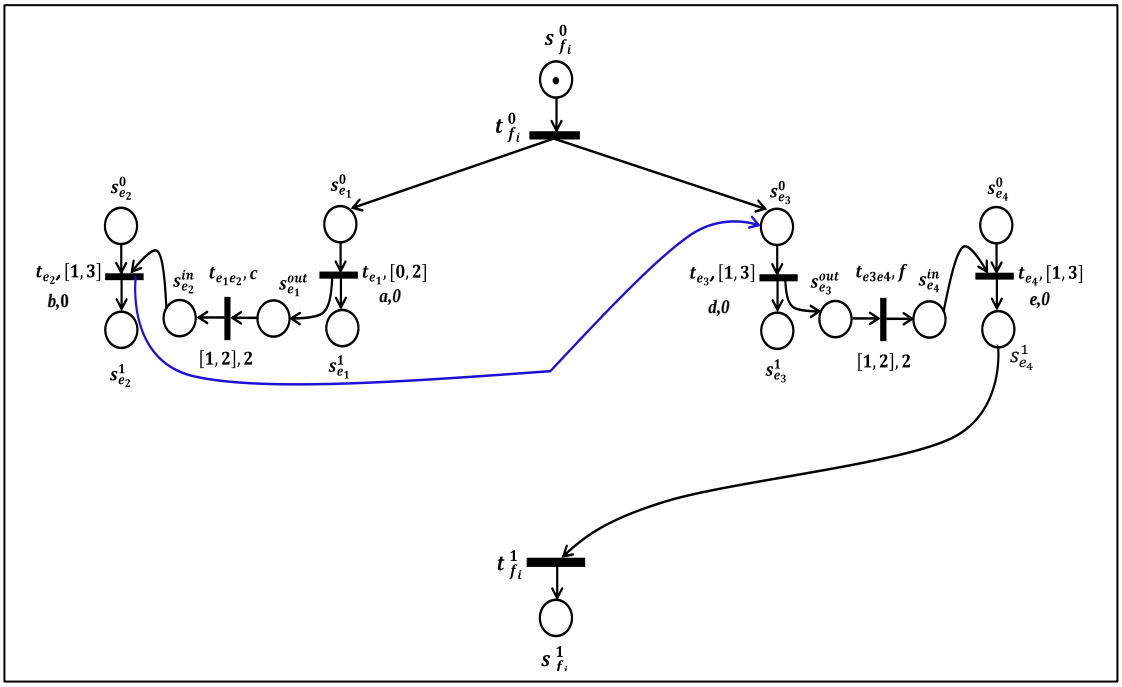


FIGURE 5.26– DTPN correspondant au fragment Strict

5.4.2.3 Fragment Par

Un fragment d'interaction de type "Par" est utilisé pour spécifier le comportement concurrent des systèmes temps réel. Il décrit une exécution parallèle des comportements liés aux différents opérandes de fragment d'interaction. Les comportements de ces opérandes peuvent être entrelacés de n'importe quelle manière. Cependant, chaque comportement d'opérande dans le fragment préserve son ordre prédéfini.

Pour détailler les règles de traduction de ce type de fragment, considérons l'exemple de la figure 5.27. D'après l'exemple, le fragment d'interaction est une exécution parallèle de deux opérandes *Opérande-1* et *Opérande-2* avec une contrainte de temps globale, *execTime= [(2, ms, min), (5, ms, max)]* imposée sur les fragments combinés. L'objet O_1 lance deux opérations concurrentes de type transmission asynchrone. En conséquent, les opérations de réception correspondantes dans l'objet O_2 peuvent également se produire simultanément. Les deux objets O_1 et O_2 peuvent continuer leur exécution selon la contrainte temporelle du fragment d'interaction parallèle. Ce dernier est mappé en modèle de réseaux DTPNs comme suit :

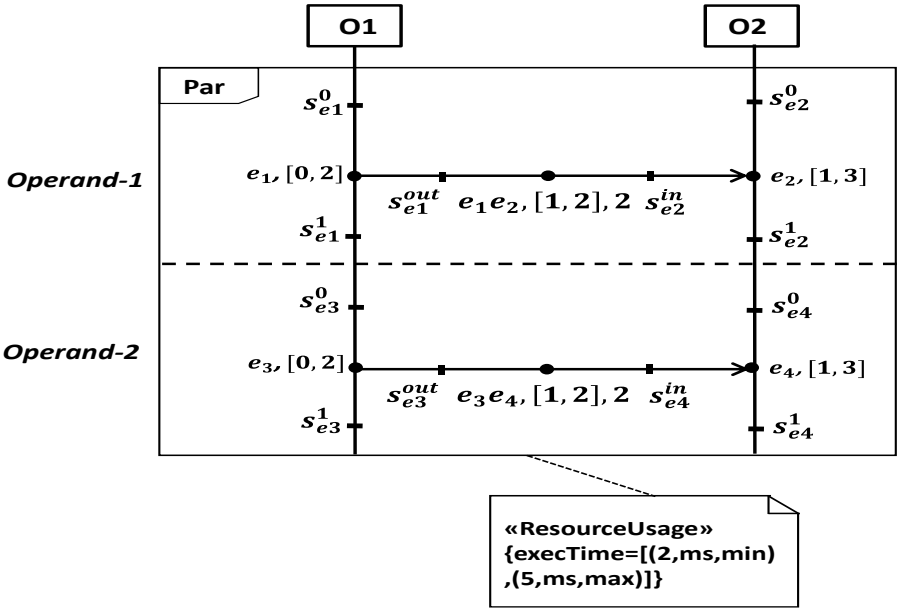


FIGURE 5.27– Fragments combinés type Par

1- Comme première étape, une place initiale $s^{0}_{f_i}$ est créée avec le marquage initial $M(s^{0}_{f_i}) = 1$. Cette place de marquage est utilisée dans l'évaluation de la condition de franchir des transitions connectées ;
2- Pour synchroniser tous les opérandes impliqués dans l'exécution parallèle des opérandes de fragment combiné, la transition $t^{0}_{f_i}$ est créée. Elle est instantanée avec une contrainte de temps remise à zéro (*d =0, t =0*). Un arc reliant la place initiale $s^{0}_{f_i}$ à la transition initial $t^{0}_{f_i}$ est donc ajouté ;
3- En utilisant les règles de traduction, deux sous-réseaux DTPNs, *DTPN-1* et *DTPN-2*, qui correspondent aux deux opérandes du fragment Par, *Opérande-1* et *Opérande-2*, respectivement sont générés. Nous notons que les places $s^{0}_{e_1}$ et $s^{0}_{e_3}$ des sous-réseaux *DTPN-1* et *DTPN-2* n'ont pas de marquage initial puisqu'elles deviennent des places non initiales, ainsi $M(s^{0}_{e_1}) = 0$ et $M(s^{0}_{e_3}) = 0$. La transition initiale $t^{0}_{f_i}$ est ensuit reliée par deux arcs à ces deux places ;
4- Pour synchroniser à la sortie tous les opérandes impliqués dans le fragment parallèle, la transition finale $t^{1}_{f_i}$ est créée. Elle caractérise par un intervalle de temps [2,5] comme contrainte de synchronisation. Cette transition est par la suite connectée par deux arcs, comme transition de sortie à la place finale $s^{1}_{e_2}$ de sous-réseau *DTPN-1* et à la place finale $s^{1}_{e_4}$ de sous-réseau *DTPN-2*. Ainsi, la transition finale $t^{1}_{f_i}$ ne sera franchie que lorsque chaque opérande de fragment combiné parallèle atteint son état final (un jeton à sa place finale) ;
5- La place $s^{1}_{f_i}$ correspondante à l'état final de fragment combiné est créée. La transition finale $t^{1}_{f_i}$ est ensuite reliée, en tant que transitions de sortie à cette place.

La structure complète de réseau DTPN obtenu est donnée par la figure 5.28.

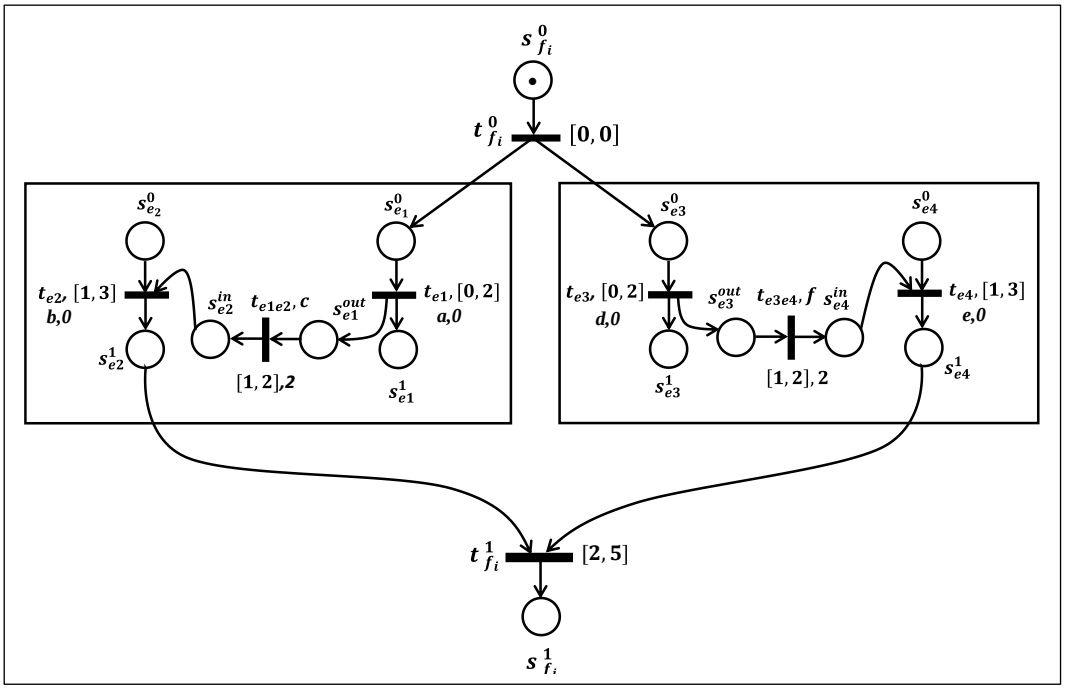


FIGURE 5.28– DTPN correspondant au fragment Par

5.4.2.4 Fragment Alt

Le fragment combiné alternatif dénoté par l'opérateur "Alt", représente un choix ou des alternatives de comportement. Cet opérateur implémente un choix non déterministe entre des opérandes. Il contient deux opérandes ou plusieurs, et il sera exécuté un seul opérande seulement. L'opérande choisi pour être exécuté doit avoir une expression de garde qui prend la valeur "vrai".

La figure 5.29 représente un exemple d'opérateur Alt. Nous remarquons qu'il y a exactement deux opérandes dans le fragment combiné. Le premier opérande sera exécuté si la condition de la garde est évaluée à vrai et le deuxième opérande ne sera pas exécuté. Le deuxième opérande dans le fragment sera exécuté si la condition de la garde est évaluée à vrai et le premier opérande ne sera pas exécuté dans ce cas la condition de la garde est fausse. Si les deux opérandes du fragment combiné évaluent les conditions de la garde comme fausses, aucun des opérandes de fragment combiné n'est exécuté, ainsi que fin de comportement et sorte de fragment.

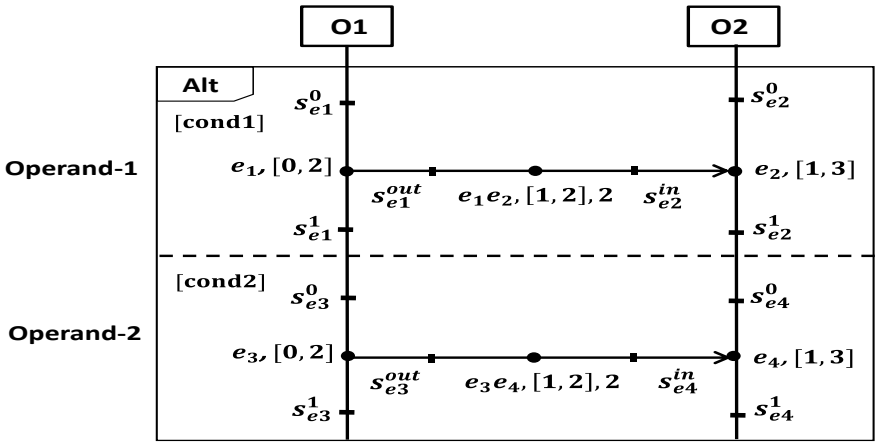


FIGURE 5.29–Fragments combinés type Alt

Les règles de traduction du fragment combiné alternative en réseau DTPN équivalent sont les suivantes.

1- La place initiale $s\,^{0}_{f_i}$ avec le marquage initial *M*, $M\,(s\,^{0}_{f_i}) = 1$ est créée avec ;
2- Deux transitions conflictuelles $t\,^{1}_{f_i}$ et $t\,^{2}_{f_i}$ sont générées. De cette manière, qu'un seul de deux opérandes, *Opérande-1* et *Opérande-2* est sélectionné, même si les gardes (cond1 et cond2) des transitions sont toutes les deux évaluées comme vraies. Deux arcs reliant les deux transitions $t\,^{1}_{f_i}$ et $t\,^{2}_{f_i}$, en tant que transitions de sortie à la place initiale $s\,^{0}_{f_i}$ sont ajoutés ;
3- En appliquant les règles de traduction dans la section précédente, deux sous-réseaux DTPNs, *DTPN-1* et *DTPN-2* sont générés, qui correspondent à *Opérande-1* et Opérande-*2*, respectivement dans le fragment combiné Alt. Les transitions précédentes $t\,^{1}_{f_i}$ et $t\,^{2}_{f_i}$ sont reliées par deux arcs aux places initiales $s\,^{0}_{e_1}$et $s\,^{0}_{e_3}$, respectivement qui lancent les exécutions des deux opérandes internes de fragment (*Opérande-1* et *Opérande-2*) ;
4- Pour représenter la sortie de chaque sous-réseau DTPN, deux transitions finales $t\,^{3}_{f_i}$ et $t\,^{4}_{f_i}$ sont générées. Ces transitions sont alors liées, en tant que transitions de sortie aux deux places finales $s\,^{1}_{e_1}$ et $s\,^{1}_{e_4}$, qui sont associées aux sous-réseaux *DTPN-1 et DTPN-2*, respectivement ;
5- La place finale $s\,^{1}_{f_i}$ de DTPN est créé et connecté, comme place de sortie aux deux transitions finales $t\,^{3}_{f_i}$ et $t\,^{4}_{f_i}$.

La figure 5.30 montre la construction complète de réseau DTPN équivalent au fragment combiné de type Alt.

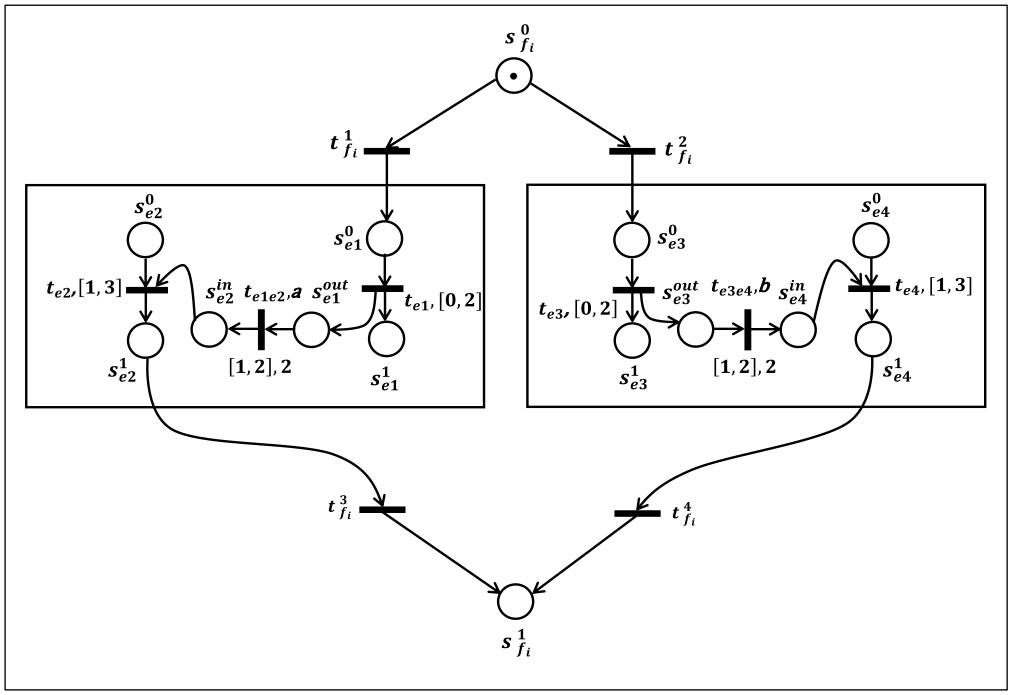


FIGURE 5.30– DTPN correspondant au fragment Alt

5.4.2.5 Fragment Opt

Un fragment combiné optionnel, définis par l'opérateur "Opt", ne contient qu'un seul opérande qui est exécuté selon une condition de garde. L'opérateur Opt est implémenté comme une instruction conditionnelle *if…then… endif*. La figure 5.31 donne un exemple illustrant cet opérateur.

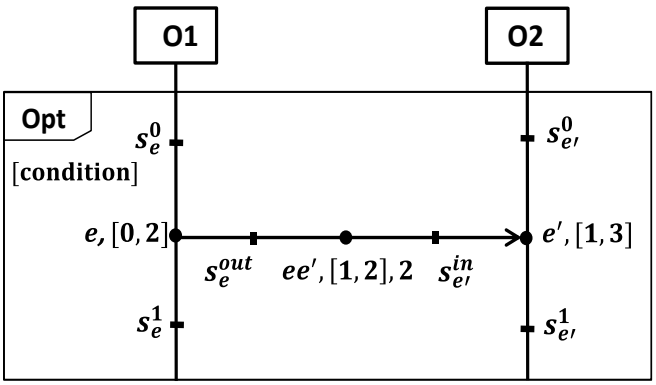


FIGURE 5.31–Fragment combiné type Opt

Le fragment optionnel est sémantiquement équivalent au fragment alternatif. Donc, ce fragment se traduit par une simplification de fragment combiné Alt avec un seul opérande, pris en considération lorsque la condition de la garde est évaluée à vraie.

La figure 5.32 montre le modèle DTPN correspondant. On peut voir que dans le DTPN obtenu, la transition $t_{f_i}^{0}$ n'est pas sensibilisée que si l'expression de la garde (*[condition])* est vraie. Sinon, la transition $t_{f_i}^{1}$ met fin au fragment combiné Opt.

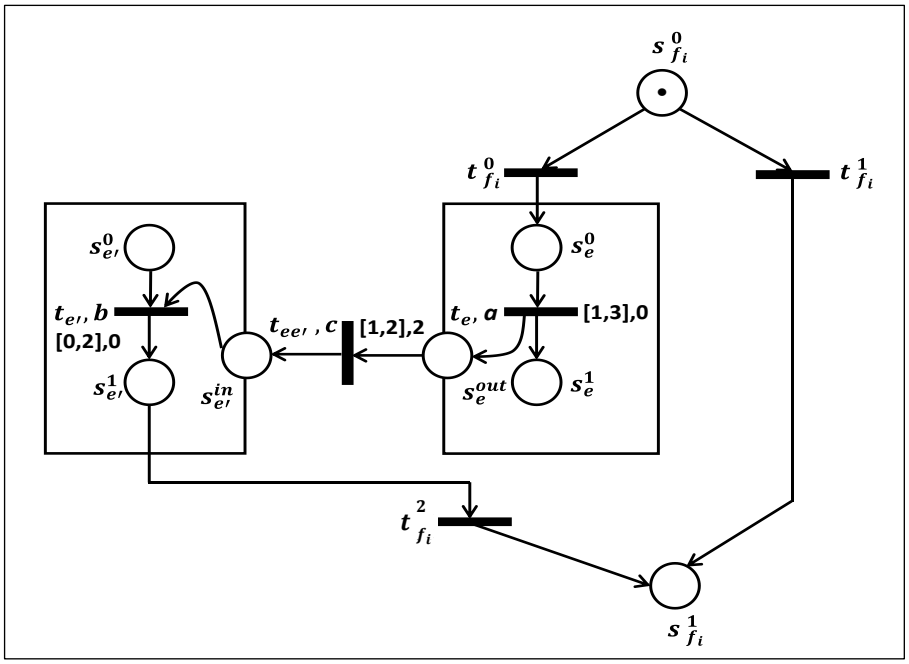


FIGURE 5.32– DTPN correspondant au fragment Opt

5.4.2.6 Fragment Loop

Le fragment combiné itératif dénoté par l'opérateur "Loop", n'a qu'un seul opérande définit un comportement récursif avec une condition de garde. La condition de la garde permet d'indiquer le nombre de répétitions ([minimum, maximum]) à exécuter ou une expression booléenne. S'il n'y a pas de borne spécifiée à la boucle, alors le zéro et l'infini sont les bornes inférieure et supérieure, respectivement. Si la borne inférieure est donnée, la borne supérieure est égale à la borne inférieure. La boucle sera exécutée exactement par le nombre spécifié des fois. Si seule la borne supérieure est donnée, la borne inférieure est inférieure ou égal à la borne supérieure. La figure 5.33 représente un exemple représentant des fragments combinés de type Loop.

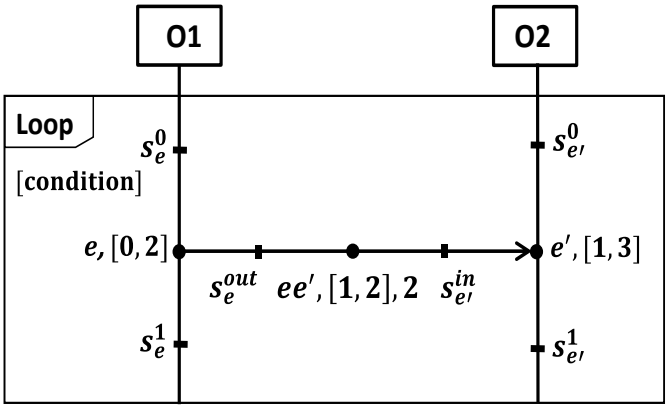


FIGURE 5.33– Fragment combiné type Loop

A présent, nous détaillons les règles de traduction de l'opérateur Loop en réseau DTPN équivalent.

1- Comme première étape dans la processus de translation, La place initiale $s_{f_i}^{0}$ avec le marquage initial *M* est créée, $M(s_{f_i}^{0}) = 1$;
2- Deux transitions initiales $t_{f_i}^{0}$ et $t_{f_i}^{1}$ sont générées. Ces transitions permettent d'évaluer l'expression de la garde qui sert comme étant une contrainte sur le fragment Loop ;
3- Deux arcs reliant la place initial $s_{f_i}^{0}$ aux deux transitions $t_{f_i}^{0}$ et $t_{f_i}^{1}$ sont ajoutés ;
4- En utilisant les règles de traduction détaillées précédemment, le sous-réseau DTPN correspondant à l'opérande du fragment d'interaction de type Loop est généré ;
5- Puisque, le but de cet opérateur est de répéter l'exécution de l'opérande du fragment d'interaction jusqu'à ce que l'expression de la garde soit évaluée à faux, la transition $t_{f_i}^{2}$ est créé et connecté à la place initial $s_{f_i}^{0}$ du fragment. La place finale $s_{e'}^{1}$ de sous-réseau DTPN est ensuit connecté à la transition $t_{f_i}^{2}$;
6- Pour considérer le cas où l'expression booléenne associée à la transition $t_{f_i}^{1}$ est évaluée à faux, la place finale $s_{f_i}^{1}$ est créée et connecté à la transition $t_{f_i}^{1}$ en tant que place de sortie.

La figure 5.34 illustre la structure finale de réseau DTPN résultante de fragment combiné type Loop dans la figure 33.

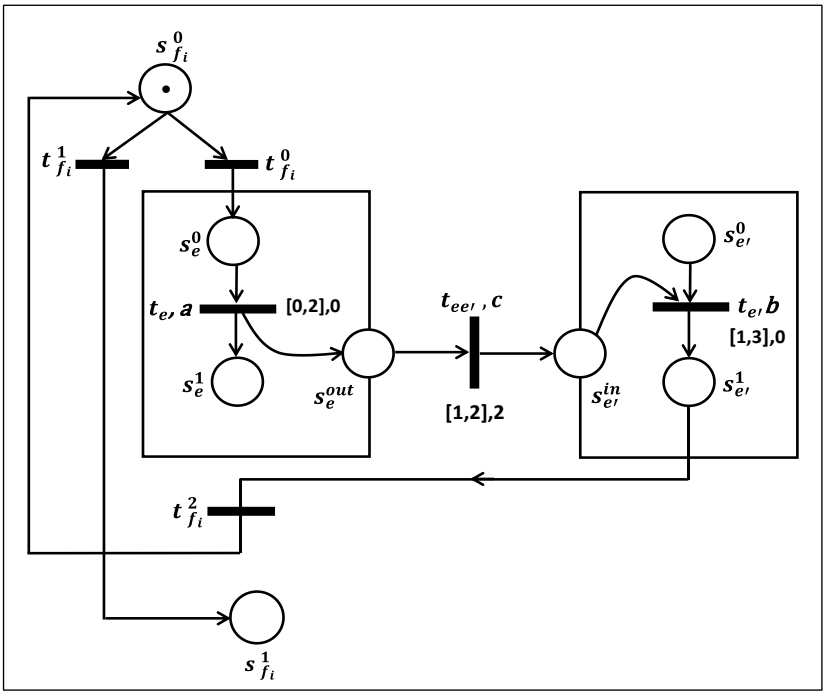


FIGURE 5.34– DTPN correspondant au fragment Loop

5.5 Conclusion

Dans ce chapitre, après avoir évoqué succinctement le principe de l'approche proposée pour la vérification formelle des spécifications UML MARTE, à savoir la vérification par model-cheking, nous avons proposé une sémantique formelle aux diagrammes de séquence UML2 avec des annotations UML MARTE pour indiquer les contraintes temporelles de système étudié, à savoir les contraintes de temps et les durées d'exécution d'actions. Ensuite, nous avons présenté en détail à travers des exemples les règles formelles et graphiques de translation d'un diagramme de séquence UML2 annoté par le profil MARTE en modèle de réseaux de Petri temporellement temporisés (DTPN). La méthode de traduction a été définie inductivement à partir des constructions de base.

Afin de montrer notre approche, nous allons l'appliquée sur un fragment de diagramme de séquence UML2, avec des annotations temporelles, qui représente un système temps-réel embarqué en l'occurrence le système de contrôle d'ascenseur. C'est l'objectif de chapitre suivant.

6. *Étude de cas*

Sommaire

Chapitre 6

Étude de cas

6.1 Introduction

L'objectif de ce chapitre est de présenter la spécification de comportement d'un système embarqué à temps contraint en utilisant les DTPNs, réseaux de Petri temporels avec durées d'actions comme modèle de spécification d'haut niveau, et les daTAs, automates temporisés avec durées d'actions comme modèle sémantique de bas niveau afin d'appliquer la vérification formelle des différentes propriétés de système et même de valider son bon fonctionnement. Nous considérons comme exemple celui de système de contrôleur d'ascenseur, qui est un exemple classique largement employé comme étude de cas pour la spécification des systèmes temps-réel et/ou embarqués. Nous proposons d'utiliser un fragment d'interaction dans un diagramme de séquence représentant le cas d'utilisation demande d'ascenseur (*Elevator request*) dans un immeuble couvrant plusieurs étages.

6.2 Système de contrôle d'ascenseur

6.2.1 Architecture globale

Le système de contrôle d'ascenseur (SCA) est un système embarqué temps-réel qui a pour but de permettre aux objets usuels de réagir avec l'environnement. Ces derniers peuvent aussi apporter une interface avec l'utilisateur. La figure 6.1 donne un aperçu des interactions qui existent entre les entités physiques constituant l'ascenseur et le contrôleur (application temps réel) qui est le cerveau du système : l'environnement est mesuré par divers capteurs. L'information issue des capteurs est échantillonnée pour être traitée par le cœur du système embarqué (le contrôleur). Puis le résultat du traitement est converti en signaux analogiques qui génèrent les actions sur l'environnement et les différents objets (cabine, portes, moteur, écrans d'information pour l'utilisateur,.. etc).

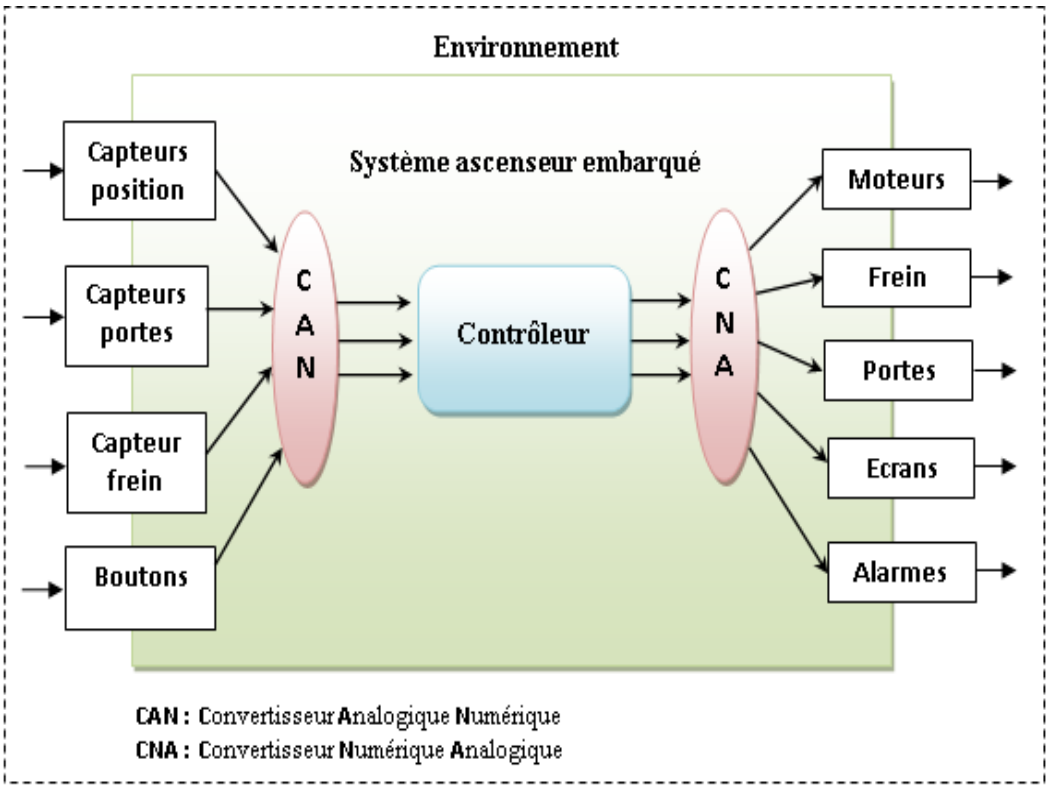


FIGURE 6.1– Architecture de système d'ascenseur

6.2.2 Description

Nous considérons, un seul système de contrôle de deux ascenseurs devront servir dans un immeuble comprend *N* étages. Ce système de contrôle est un système distribué où le comportement est réparti dans différents processus concurrents. Le problème qui nous intéresse ici est la logique requise pour spécifier le mouvement de deux ascenseurs en parallèle entre les différents étages afin d'avoir deux activités, qui durent dans le temps et s'exécutent en parallèle. Par exemple, nous supposons *User1* et *User2* sont deux utilisateurs à deux étages différents. À un instant donné, chacun demande un ascenseur. L'utilisateur *User1* est au premier étage tandis que l'utilisateur *User2* est dans le septième étage. On suppose encore que l'utilisateur *User1* doit se rendre au septième étage, et *User2* au premier étage. Puisque notre objectif est consacré à la modélisation des activités parallèles, nous supposons que les deux ascenseurs sont stationnés aux premier et septième étages respectivement dont leurs portes sont ouvertes et les utilisateurs sont à l'intérieur. Les deux utilisateurs ont simultanément appuyé sur le bouton spécifiant l'étage à visiter dans le panneau de boutons dans chaque ascenseur. Le contrôleur d'ascenseur doit déplacer en parallèle les deux ascenseurs vers les étages demandés.

Pour simplifier l'étude, nous avons fait une abstraction aux certains composants du système d'ascenseur participant dans l'interaction et qui n'affectent pas l'objectif de l'étude par exemple, la cabine, le capteur d'étage et la porte. Une fois que le contrôleur d'ascenseur a reçu les deux demandes

et en même temps, il crée et active deux processus en parallèle *Elevator1* et *Elevator2* (deux instances). *Elevator1* et *Elevator2* sont respectivement responsables de déplacer la cabine du premier ascenseur et la cabine du deuxième ascenseur aux étages demandés.

Concernant le contexte temporel, deux aspects de temps sont pris en compte, à savoir la durée d'exécution d'une action donnée et les contraintes temporelles relatives à la restriction temporelle (un délai). Dans l'exemple, l'action de mouvement de la cabine d'ascenseur est très limitée par le temps, elle peut prendre 3 unités de temps (*d=3000ms*). Un autre aspect à considérer est la possibilité d'avoir un retard d'exécution d'une action par une certaine quantité de temps comme par exemple l'action de mouvement de la cabine peut prendre de temps pour démarrer, donc elle peut être en retard un certain temps qui peut être représenté par l'intervalle de temps *[0,1]*.

6.3 Modélisation

La figure 6.2 illustre le fragment d'interaction parallèle dans un diagramme de séquence modélisant les aspects temporels et les interactions entre les objets impliqués dans la fonctionnalité de cas d'utilisation *Elevator request*, avec une contrainte de temps globale (*tc = [10, 30]*) associée au fragment d'interaction Parallèle.

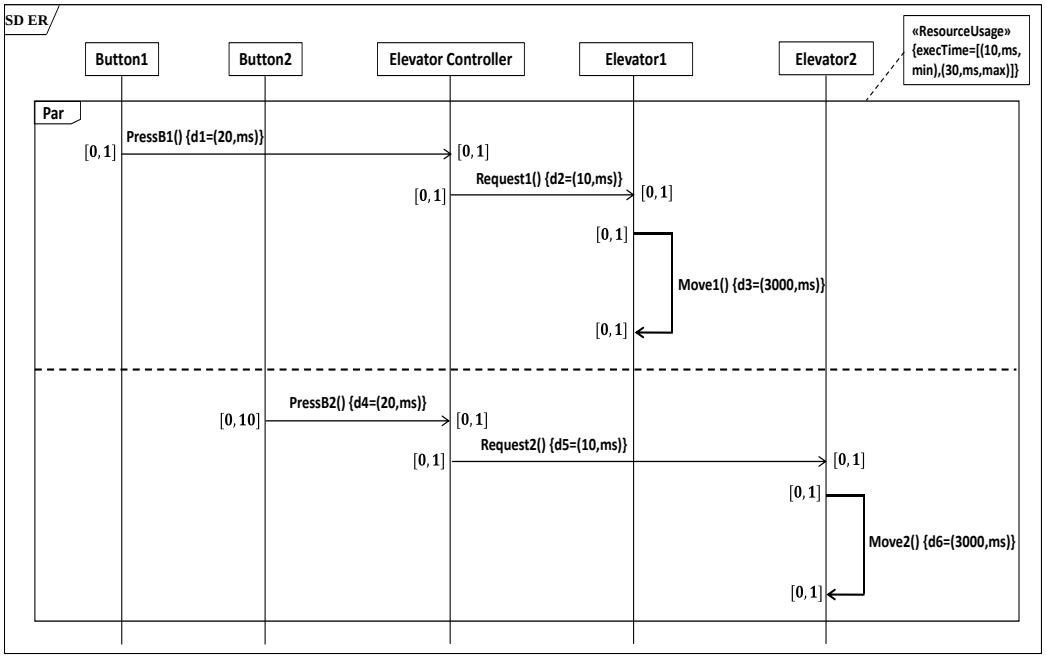


FIGURE 6.2– Les interactions entre les objets participant dans le fragment *Elevator request*

Pour appliquer la méthode de translation, MARTESDtoDTPN sur le fragment d'interaction *Elevator request*, nous annotons d'abord les événements temporisés d'envoi et de réception des messages transmis entre les différents objets impliqués dans l'interaction. D'après la figure 6.3, l'ensemble des événements temporisés est $E^t = \{e_1, e_2, e_3, .., e_{12}\}$.

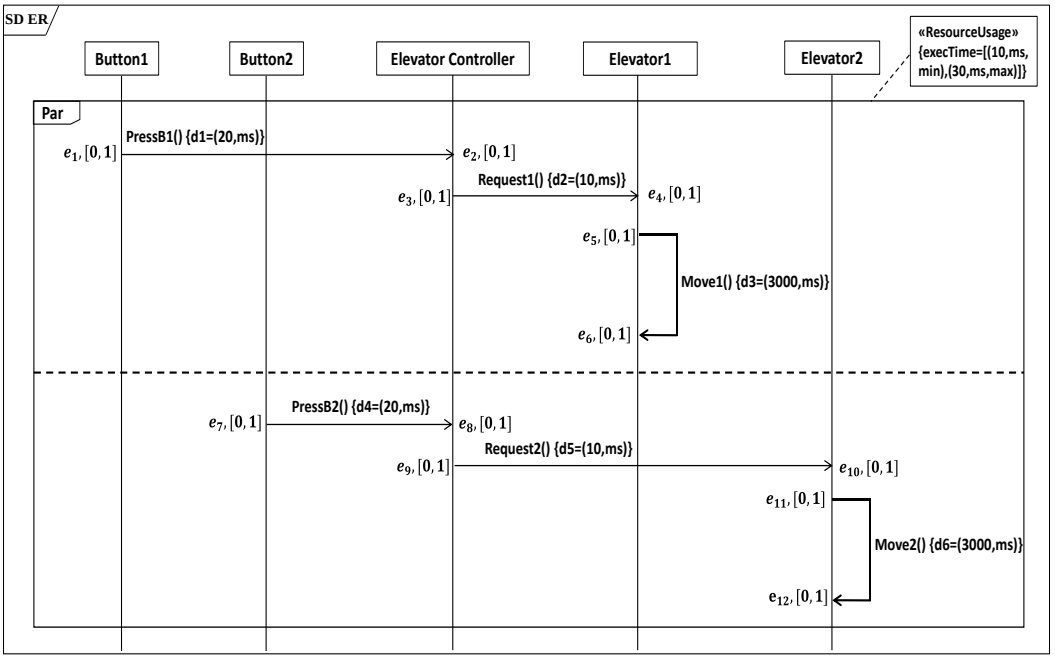


FIGURE 6.3– Fragment *Elevator request* annoté par les événements d'envoi et de réception des messages

Ensuite, nous ajoutons les événements représentant les transmissions intermédiaires entre les objets avec leurs contraintes temporelles associées. D'après la figure 6.4, l'ensemble des événement temporisés devient E^t={e_1, e_2, e_3,…, e_{12}}∪{e_1e_2, e_3e_4, e_5e_6,…,$e_{11}e_{12}$}.

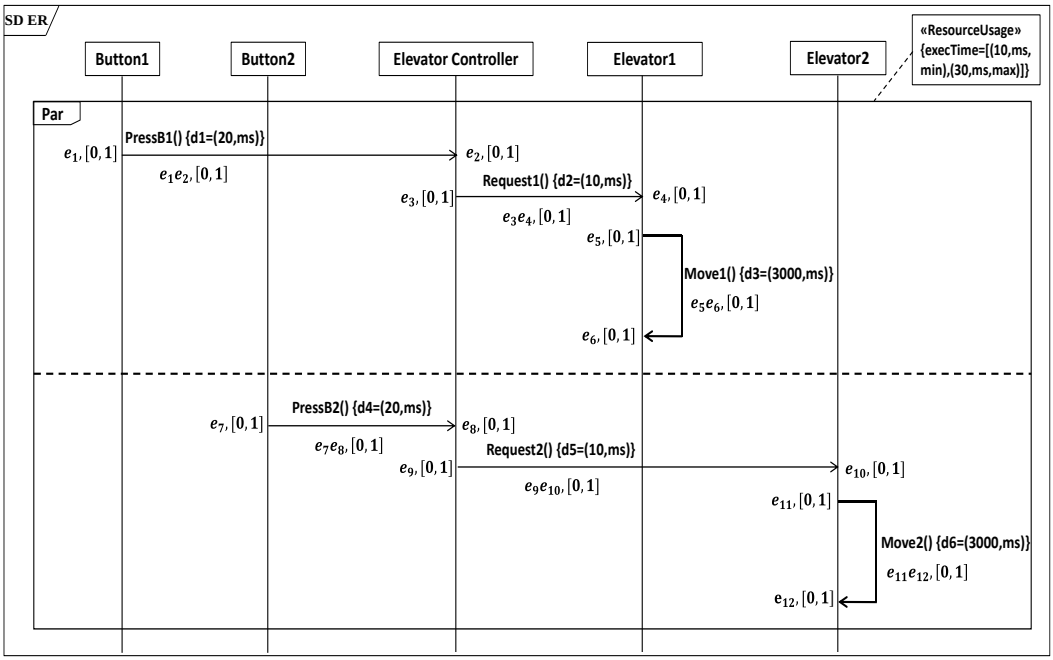


FIGURE 6.4– Fragment *Elevator request* annoté par les événements des transmissions intermédiaires

Puis, pour chaque objet impliqué dans l'interaction, nous associons deux locations d'état (*state locations*), avant et après, à chaque événement sur sa ligne de vie, ainsi $S= \{s^0_{e_1}, s^1_{e_1}, \dots, s^0_{e_{12}}, s^1_{e_{12}}\}$. De façon similaire, nous associons deux locations d'état (avant et après) aux événements temporisés associés aux transmissions intermédiaires dans le fragment d'interaction parallèle, $S= \{s^0_{e_1}, s^1_{e_1}, \dots, s^0_{e_{12}}, s^1_{e_{12}}\} \cup \{s^{out}_{e_1}, s^{in}_{e_2}, \dots, s^{out}_{e_9}, s^{in}_{e_{10}}\}$ (voir figure 6.5). Nous notons que les locations d'état associées aux événements temporisés qui correspondent aux transmissions intermédiaires ne sont pas représentées sur le fragment d'interaction Par dans le but d'éviter le chargement de ce dernier et la lisibilité graphique, ainsi que les événements temporisés et les locations d'états sont colorés pour une visualisation facile.

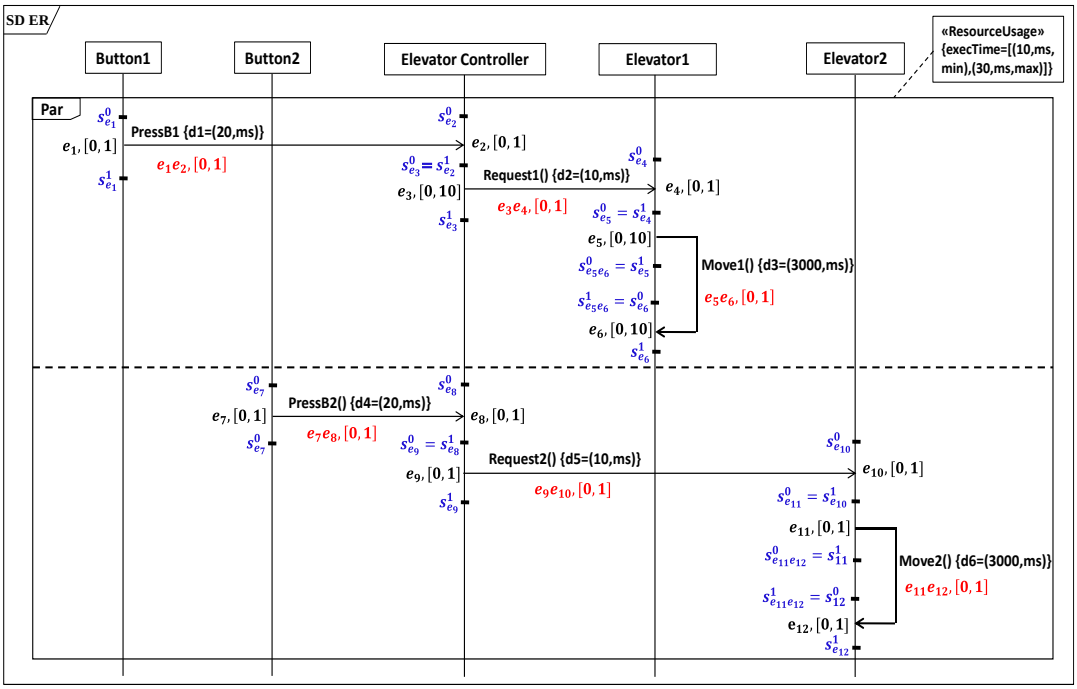


FIGURE 6.5– Fragment *Elevator request* annoté par state locations

6.3.1 Transformation des modèles

Maintenant, la méthode de translation du diagramme de séquence avec des annotations UML MARTE en modèle de DTPN détaillée dans le chapitre 5 (section 5.4) peut être appliquée. La figure 6.6 représente le réseau DTPN sous l'éditeur Dotty [Dot11]. Puisque le réseau DTPN obtenu a 30 places et 20 transitions la figure 6.7 montre un fragment du DTPN généré sous format Dotty. Pour obtenir la construction complète du réseau DTPN, nous avons utilisé l'outil Draw.io [App], qui est un moyen efficace et riche en fonctionnalités pour créer et visualiser des diagrammes avec d'excellentes performances (voir figure 6.8).

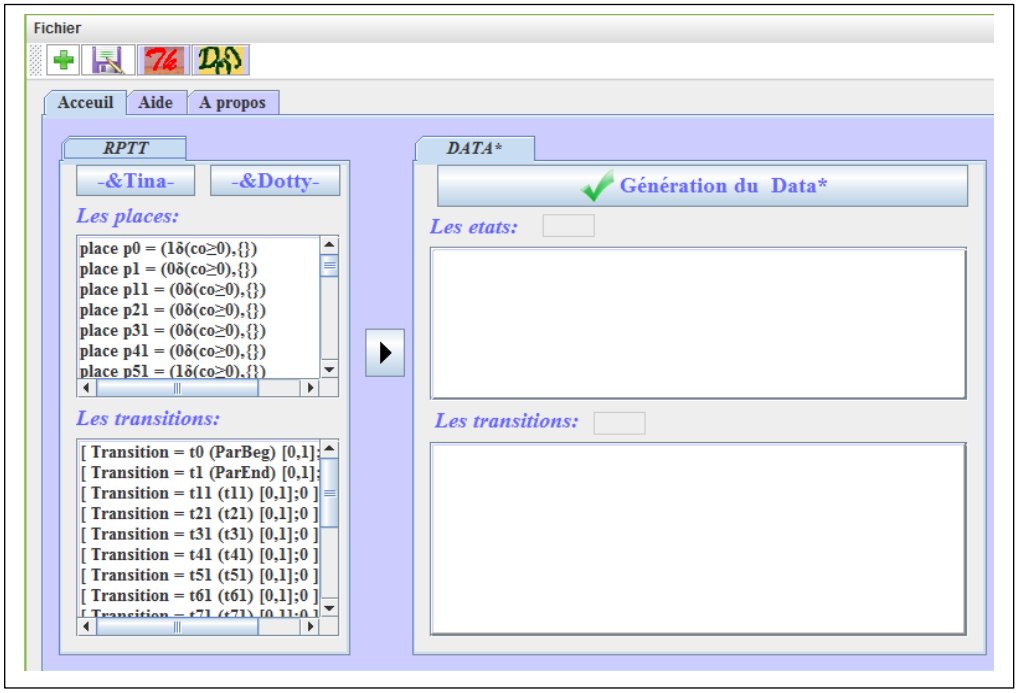


FIGURE 6.6– Représentation textuelle du DTPN avec l'éditeur Dotty

La figure 6.8 illustre la construction complète du réseau DTPN présenté par l'outil Draw.io.

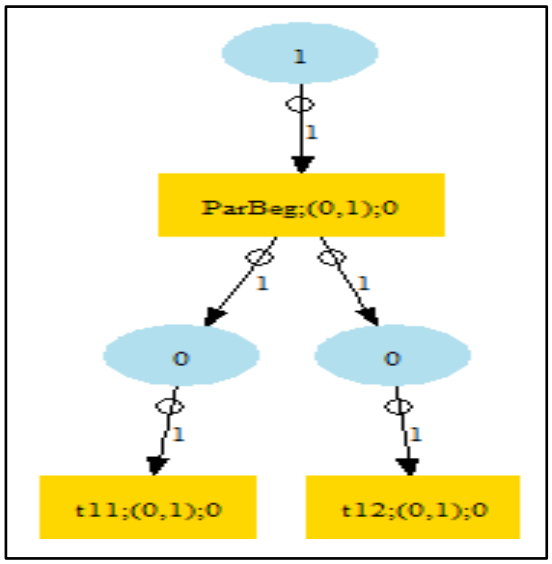


FIGURE 6.7– Fragment du DTPN présenté en Dotty

Selon la figure 6.8, la place initiale du DTPN est $s\,{}^{0}_{f_i}$, avec le marquage initial un seul jeton $M\,(s\,{}^{0}_{f_i}) = 1$, et la place finale est$s\,{}^{1}_{f_i}$. La transition initiale est $t\,{}^{0}_{f_i}$, avec la contrainte de temps [0, 0]. La transition finale est $t\,{}^{1}_{f_i}$, avec la contrainte de temps [2, 5]. Les places correspondantes aux *state locations* sont $P = \{s^0_{e_1}, s^1_{e_1}, \dots, s^0_{e_{12}}, s^1_{e_{12}}\}$. Les transitions du réseau DTPN correspondantes

aux événements sont *T*= $\{t_{e_1}, t_{e_2},\ldots,t_{e_{11}e_{12}}, t_{e_{12}}\}$. Dans le DTPN résultant, le franchissement des transitions est lié à l'intervalle de temps [10,30] imposé sur le fragment d'interaction Par. Le franchissement d'une transition représente le lancement d'une action, qui a une durée explicite.

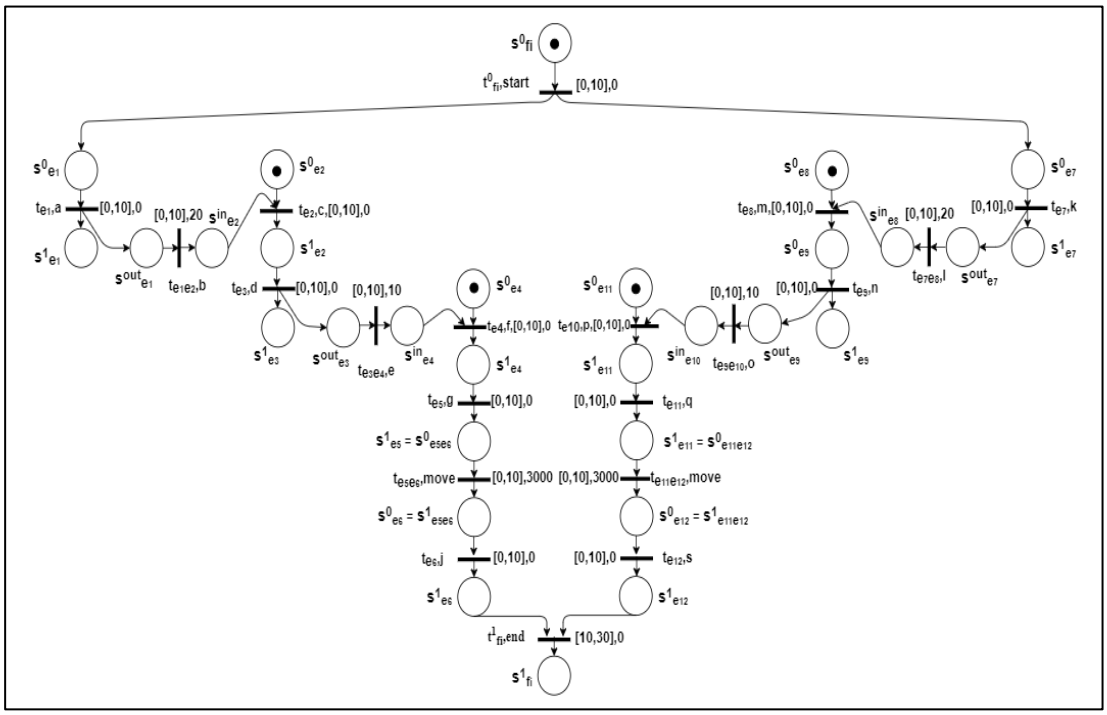


FIGURE 6.8– DTPN correspondant au fragment *Elevator request* avec l'éditeur graphique Draw.io

Avant de passer à l'étape de la vérification formelle des propriétés de bon fonctionnement et afin d'assurer que le modèle DTPN est structurellement correcte, on peut l'analyser (ex, reachability analysis, structural analysis) et le simuler (stepper simulator) par l'outil TINA.

6.4 Vérification par model-checking

Pour étudier la vérification et la validation de système de contrôle d'ascenseur, nous optons pour la sémantique opérationnelle développée dans [BSB+13] afin de générer le modèle sémantique daTA aux spécifications DTPNs. En se basant sur la sémantique de maximalité qui est une sémantique de vrai parallélisme, le daTA obtenu est représenté en Dotty dans la figure 6.9. La présentation graphique Dotty d'un fragment de daTA est donnée par la figure 6.10. Le modèle daTA résultant a 42 états et 68 transitions, il est généré automatiquement à l'aide d'un outil intégré à l'environnement FOCOVE. Pour cette raison, nous ne représentons qu'un fragment de graphe (voir figure 6.11). Cette structure permet la vérification des propriétés liées à l'évolution parallèle et concurrente des actions comme montré dans les différents états de daTA résultant. A titre d'exemple, l'état s36 est étiqueté par les conditions de durée $\{x \geq 3000ms, y \geq 3000ms\}$ tels que, l'horloge *x* est relative à l'exécution de l'action *Move1* et l'horloge *y* représente l'exécution de *Move2*. Dans cet état,

les deux actions *Move1et Move2* sont potentiellement en exécution parallèle, et chacune ne se terminera que si son horloge attendra une valeur égale à sa durée.

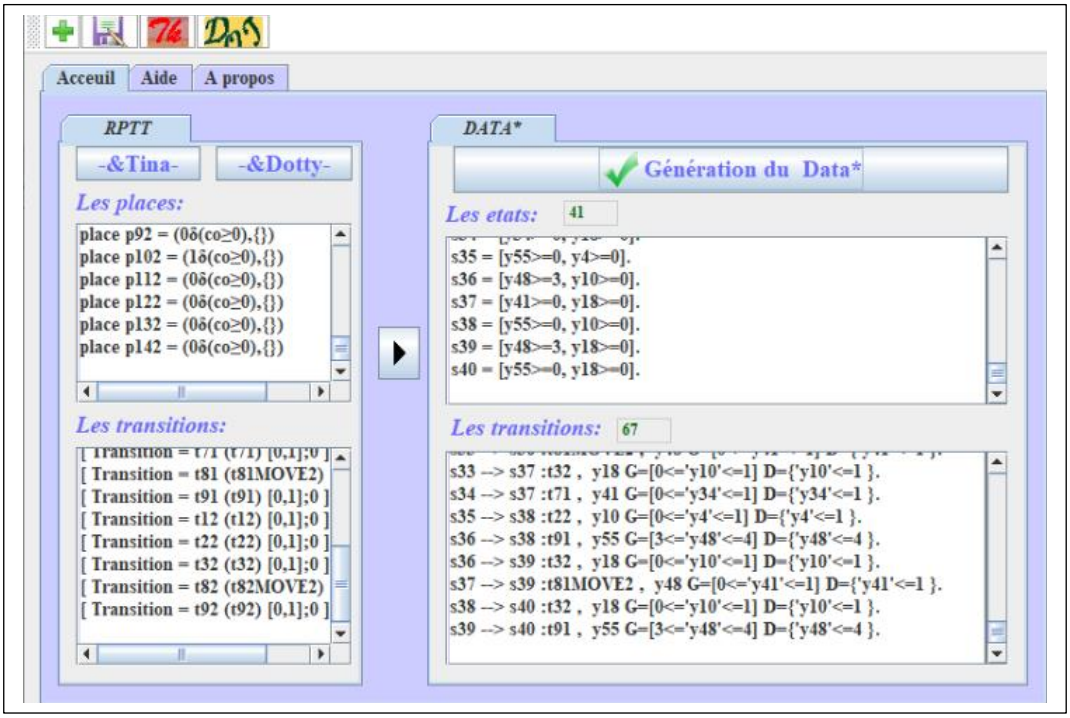


FIGURE 6.9– Représentation textuelle du daTA avec l'éditeur Dotty

D'après le modèle daTA obtenu, remarquons que nous n'avons utilisé que deux horloges (*x et y*) pour spécifier toutes les actions de cet exemple, car nous avons aux plus deux actions en cours d'exécution à un moment donné. Ceci est possible grâce à la création dynamique des horloges avec la réutilisation des horloges libres, ce qui permet à remédier au problème d'explosion combinatoire d'espace d'états des systèmes modélisés. La vérification par model-checking est principalement basée sur les graphes de régions et les graphes de zones à partir du daTA. Notons que la complexité de model-checking sur TA, comme daTA, est exponentielle relativement au nombre d'horloges utilisées. Nous pouvons observer qu'en utilisant le DTPN basé sur la sémantique de vrai parallélisme, le daTA généré a un nombre d'horloges considérablement faible pour deux raisons. Le premier point est la création dynamique et au fur à mesure des horloges durant le processus de génération et pas avant. Le deuxième avantage est la réutilisation des horloges déjà libérées après l'achèvement des actions.

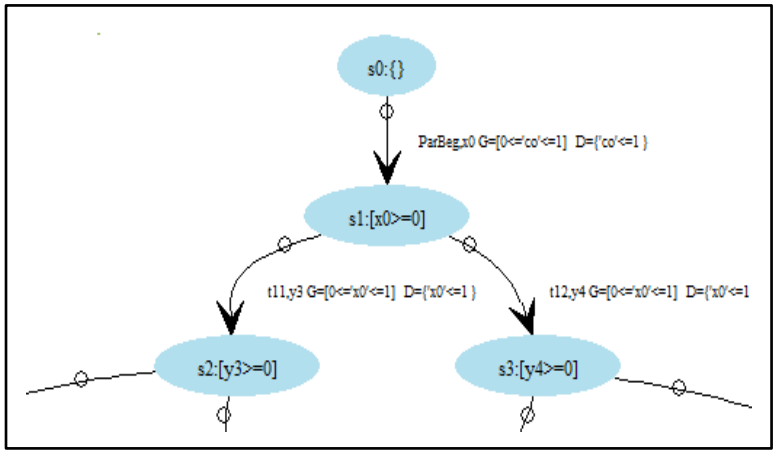


FIGURE 6.10– Fragment du DTPN présenté en Dotty

Nous pouvons ensuite appliquer le model-checking pour vérifier certaines propriétés de bon fonctionnement du système, à savoir les propriétés de sûreté et les propriétés de vivacité. Plusieurs caractéristiques et bonnes propriétés du modèle des daTAs sont établies dans [KHBS12] tels que la déterminisation et l'expressivité.

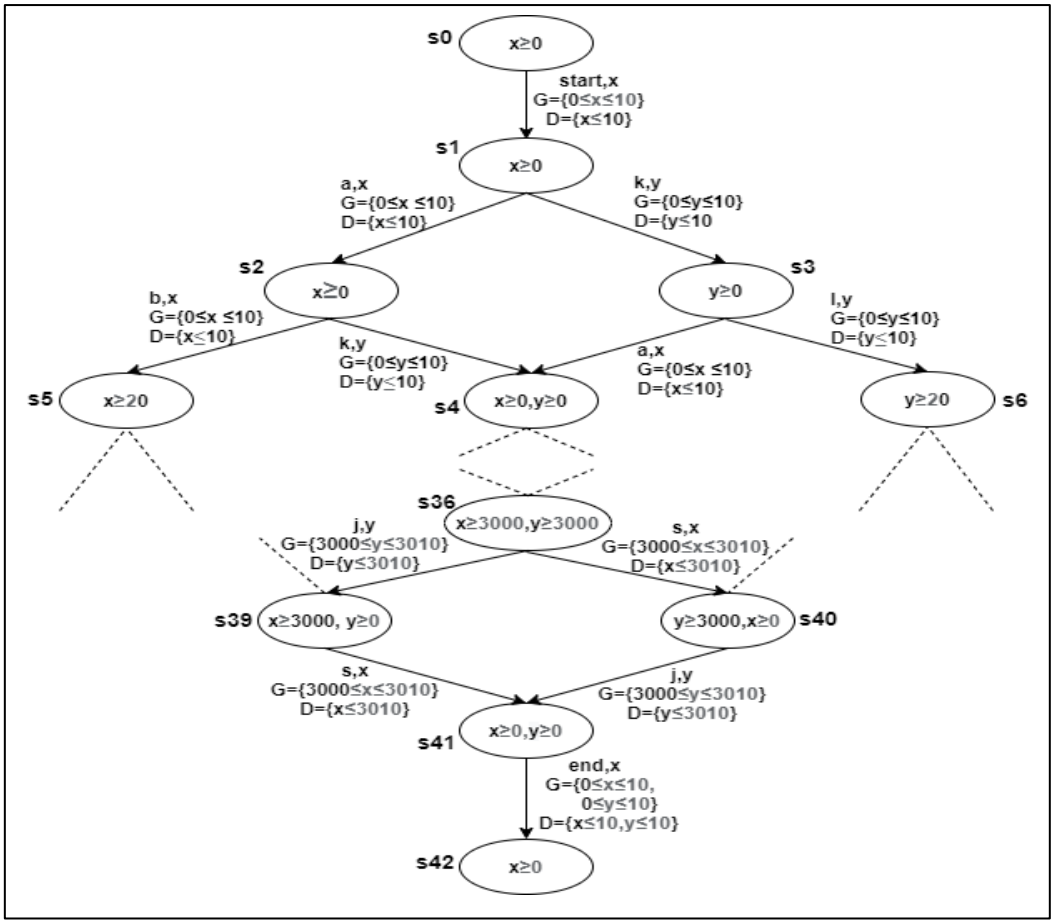


FIGURE 6.11– Fragment du daTA correspondant à DTPN

Vérification

La vérification des propriétés avec l'outil UPPAAL qui implante un model-checker dont les propriétés à vérifier sont exprimées en CTL et TCTL (logique temporelle du temps arborescent). Dans ce qui suit, nous allons présenter quelques propriétés (qualitatives et quantitatives) typiques du système de contrôle d'ascenseur, tout en donnant leurs spécifications dans les logiques CTL et TCTL.

La figure 6.12 présente le modèle daTA sous l'éditeur graphique de l'outil UPPAAL. Après l'étape d'analyse et de validation de la syntaxe du modèle daTA avec succès, nous pouvons vérifier les propriétés suivantes :

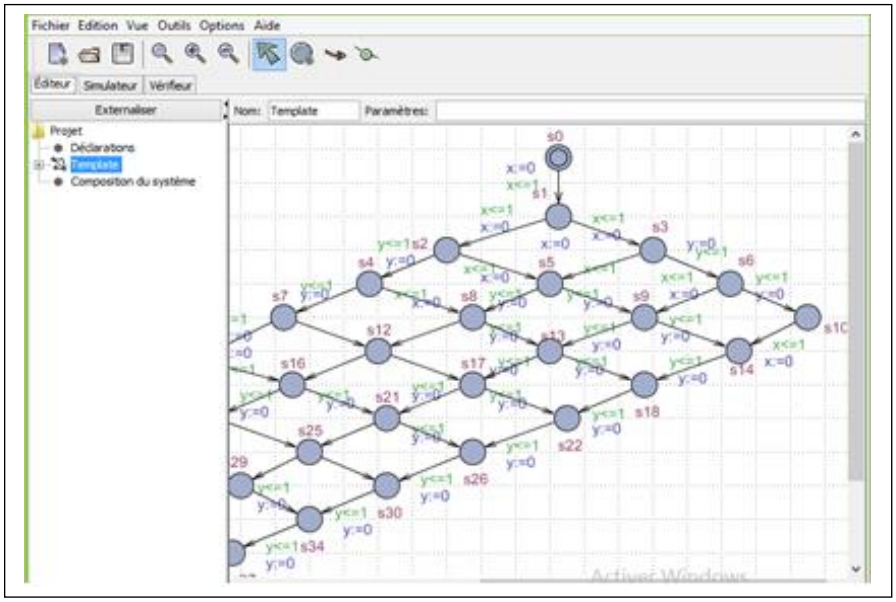


FIGURE 6.12– Représentation du daTA avec l'éditeur graphique de l'outil UPPAAL

1. L'accessibilité (reachability) : indique qu'un état du système peut être atteint :

   – **EF** s25

   – **EF** s42

La figure 6.13 illustre les résultats de la vérification de cette propriété pour les deux (02) états s25 et l'état final s42 du daTA.

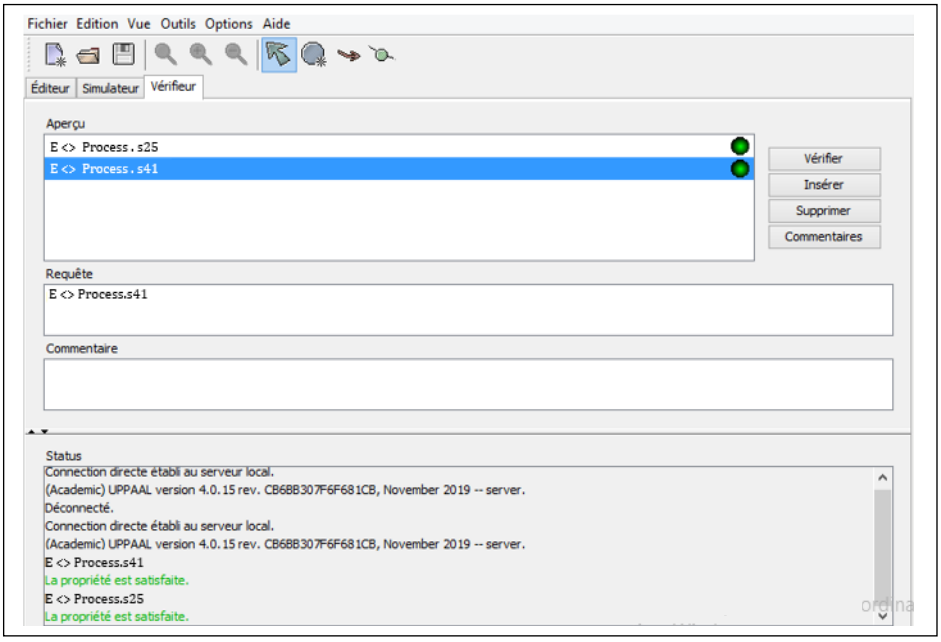


FIGURE 6.13– Résultats de la vérification de la propriété d'accessibilité

2. Le système de contrôle d'ascenseur doit prendre en considération les demandes de deux utilisateurs *User1 et User2* simultanément juste après les frappes de deux boutons indiquant étaages à visiter :

$$\mathbf{EF}\ (a \wedge k)$$

D'après le DTPN présenté dans la figure 6.8, *a* est l'action de recevoir de la demande de *User1 et k* est l'action de recevoir de la demande de *User2*.

3. Le système de contrôle d'ascenseur doit déplacer l'ascenseur (*Elevator1* ou *Elevator2*) de l'étage courant à l'étage demandé dans 3 secondes :

$$\mathbf{A}\ [\ ]\ (b \text{ imply } time - x \geq 3000)$$

4. Le système de contrôle d'ascenseur a toujours la possibilité de satisfaire la demande de l'utilisateur *User1* ou *User2* dans un intervalle de 0 seconde à 1 seconde :

$$\mathbf{E}\ [\ ]\ (c \text{ imply } 0 \leq time - x \text{ and } time - x \leq 1000)$$

6.5 Conclusion

Dans ce chapitre, nous avons présenté une étude d'un exemple de système embarqué temps- réel, Dans ce système, nous considérons les contraintes temporelles et les durées explicites des actions. L'étude concerne principalement un système de contrôle d'ascenseur en le spécifiant ses comportements, fonctionnelle et temporelle à l'aide du diagramme de séquence avec des annotations de profil MARTE puis sa translation vers le modèle de réseaux DTPNs. Par la suite, on procède à la génération des structures daTAs à partir de la spécification DTPN obtenue en utilisant l'outil FOCOVE. A cette fin, ces modèles de vrai parallélisme nous permettent la vérification formelle des propriétés, en particulier celles liées à l'évolution parallèle des actions qui ont une durée non nulle et sous des contraintes de temps. Cette étude nous semble insuffisante quant à la variété d'exemples étudiés dans la littérature. Donc, nous avons l'intention de valider l'approche proposée sur plus d'études de cas de systèmes temps-réel réalistes afin de mieux bénéficier et exploiter les avantages de cette approche.

7. Travaux connexes et discussion

Sommaire

Chapitre 7

Travaux connexes et discussion

Dans ce chapitre, nous présentons quelques travaux de recherche qui ont traité la spécification et la vérification formelle par model-cheking des systèmes temps-réel embarqués en utilisant le profil UML MART, particulièrement les diagrammes de séquence UML2. Ensuite, nous discutons les avantages de notre approche qui ont été mises en évidence par rapport à ces travaux.

7.1 Travaux connexes

Plusieurs travaux de recherche, réalisés au cours des années antérieures, ont porté sur l'intégration des techniques de vérification formelle dans le processus de développement des systèmes temps réel décrits avec des langages de modélisation semi-formels comme UML et UML MARTE, en spécifiant les exigences de comportement et de temps qui doivent être garanties par ces systèmes. Yin et al. ont proposé dans [YM11] une transformation correcte des spécifications UML MARTE en langage Promela pour vérifier par la suite son exactitude par le model-checker SPIN, par exemple la vérification des propriétés des inter-blocages, des assertions et des propriétés attendues. Toutefois, l'inconvénient de cette transformation est l'implémentation des horloges dans la spécification temporelles par des variables globales, ce qui influe négativement sur le temps de vérification, et ce par conséquent, n'évite pas le problème de l'explosion combinatoire pendant l'exploration des modèles. Menad et al. ont proposé une approche de vérification plus générale dans [MDDM16], qui permet une transformation automatique des modèles UML MARTE en modèle FIACRE [Pro]. Cette approche vérifie non seulement les implémentations des contraintes temporelles, mais aussi les propriétés du modèle y compris toutes les composantes fonctionnelles. Les propriétés sont exprimées en automates d'observateurs dans le langage CDL (Context Description Language), et sont vérifiées par l'outil de model-checking OBP (Observer Based Prover). Dans cette technique la séparation des propriétés fonctionnelles de l'application à travers le langage CDL facilite la séparation des préoccupations. Nous notons que les deux travaux [GPC12, PGC12] utilisent l'outil de model-checking TINA (**Ti**me Petri **N**et **A**nalyzer) pour la vérification formelle des modèles comportementaux UML MARTE et des propriétés temporelles. Les auteurs dans [GPC12] proposent une approche de vérification permettant d'assurer l'exactitude des propriétés temporelles par la transformation des diagrammes d'activité et d'états-transitions dans un système de transition temporel (TTS en anglais Time Transition System). Cette approche ne prend pas en considération l'aspect du temps du système, et elle est limitée par la vérification de deux types de propriétés, la synchronisation et l'ordonnancement. En revanche, le travail dans [PGC12] présente un Framework destiné à la transformation des modèles

structurels et comportementaux UML vers des modèles Time Petri Nets (TPN) afin de vérifier les propriétés temporelles par model-checker TINA.

D'autres approches qui se basent sur l'extension d'un langage formel par le temps pour la vérification formelle ont été proposées dans [BD98a, BD98b]. Bosnacki et al. ont proposé une extension de temps discret de langage *Promela* pour les systèmes concurrents dans [BD98a]. Une variable nommée *timer* est définie et correspond à un temporisateur de temps discret. Cependant, l'extension proposée est difficile à utiliser pour exprimer les ticks d'horloge de coïncidence. Bosnacki et al. ont également proposé un autre travail, voir [BD98b], qui modélise les spécifications de temps de l'Algèbre des Processus Communicants, (ACP) par macro-définitions au niveau de *Promela*. Ce travail présente une extension de *Promela* et du model-checker SPIN avec un temps discret qui permet de modéliser le bon fonctionnement des systèmes dont dépendent essentiellement des paramètres de synchronisation.

Les travaux présentés ci-dessus se basent sur la translation d'un modèle de haut niveau écrit en langage UML MARTE vers un modèle formel spécifique. Ce dernier est utilisé pour la vérification soit en utilisant des outils model-checker dédiés tels que UPPAAL, CRONOSE, Romeo et SPIN ou à travers des langages formels basés sur des sémantiques formelles fortes tels que Promela, CSP, LOTOS ou FIACRE. Cependant, peu de travaux ont été menés sur la vérification formelle des diagrammes de séquence UML2 avec des annotations MARTE en spécifiant les aspects de temps, de manière quantitative (durée, échéance, délai,..etc) du système dans le processus de translation [YYSQ12, MDDM16, APD18]. Néanmoins, les travaux précédents ne traitent que le suivi d'ordre des événements sur les lignes de vie dans les diagrammes de séquence avec une expression implicite du temps dans le processus de transformation, et ne considèrent que la sémantique d'entrelacement.

A l'opposé, dans ce travail, nous avons défini une approche de translation pour générer des modèles DTPNs à partir des diagrammes de séquence UML2 enrichis par des contraintes temporelles en UML MARTE afin d'appliquer la vérification formelle par model-checking en prenant en compte explicitement les contraintes de temps. Notre travail vise principalement la spécification et l'expression de manière intuitive des comportements des STRE, à savoir l'état d'exécution des actions, les contraintes temporelles et les contraintes d'urgence. En d'autres termes, les modèles de spécification formelle temps-réel utilisés dans notre approche (DTPN et daTA) se basent sur une sémantique de vraie concurrence. L'utilisation des structures de daTAs comme sémantique du modèles des réseaux de Petri temporellement temporisé (DTPNs) nous permet dans un premier temps d'exprimer les comportements concurrents et parallèles de manière naturelle, c'est-à-dire que ce modèle fait la distinction entre les exécutions séquentielles et parallèles des actions avec des durées d'exécution non-nulles. Ce n'est pas le cas de la sémantique d'entrelacement où les actions sont considérées atomiques et leur exécution parallèle se traduit par leur exécution entrelacée dans le temps. Deuxièmement, la réinitialisation des horloges d'états, qui sont utilisées pour spécifier le temps et les contraintes de temps conduit d'une part, à réduire le temps de la vérification formelle qui est souvent exponentiel pour les STRE complexes, et d'autre part

réduire le nombre d'états et d'horloges dans le graphe sans perte d'information, puis échapper à l'explosion combinatoire du graphe sous-jacent

7.2 Discussion

Dans notre méthode de traduction des diagrammes de séquence UML2 annotés par des contraintes temporelles UML MARTE en modèles DTPNs, nous nous sommes inspirés de la méthode du principe de passage présenté dans [BM10]. L'idée est de spécifier les évènements du diagramme de séquences par des états intermédiaires, c'est à dire d'associer des d'états (state locations) aux événements placés sur les lignes de vie correspondantes aux objets participants dans l'interaction du diagramme de séquence. Cependant, la méthode de traduction proposée dans [BM10] considère le diagramme de séquence comme une entité entière, en comparaison avec notre méthode de traduction, elle possède un ensemble de règles définies pour rendre la méthode de traduction inductive et compositionnelle. Les règles de traduction commencent par considérer les composants élémentaires, qui sont les événements du diagramme de séquence. Par conséquent, pour chaque élément composé, une règle de traduction est définie de manière inductive en utilisant le résultat de traduction de ses sous-composants. De cette manière, une construction d'un outil mettant en œuvre le procédé peut être facilement réalisé. A titre d'exemple, la figure 7.1.(a) montre les événements d'un message asynchrone qui représentent les moments dans lesquels les actions sont envoyées, transmises ou reçues. En termes de réseau de Petri chaque événement du message *Msg* (e, e' *et* e'') est interprété par une transition, chacune possède une place d'entrée et une place de sortie comme indiqué sur la figure 7.1.(b).

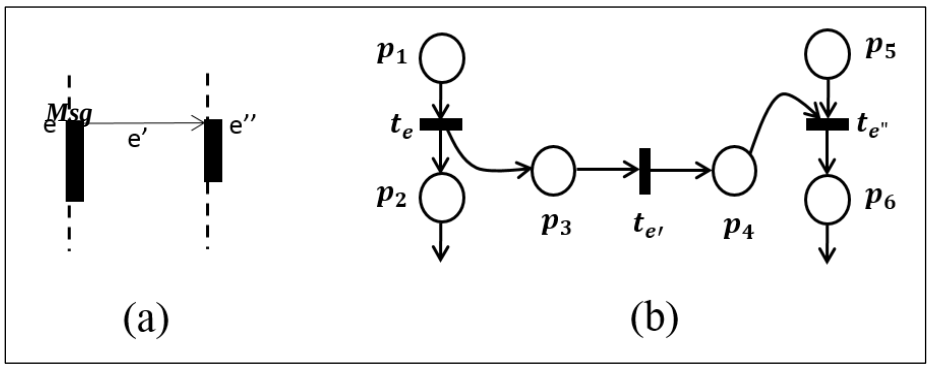


FIGURE 7.1– Translation d'un message asynchrone

Les modèles de spécifications formelles existant reposent sur une hypothèse primordiale qui suppose l'atomicité structurelle et temporelle des actions. L'interopération entrelacée de la concurrence est justifiée par le fait que les actions sont instantanées. Mais une conséquence directe est que deux actions ne peuvent s'exécuter en vrai parallélisme. Dans le travail [YYSQ12], les auteurs ont interprété les activités parallèles, modélisées dans le diagramme de séquence UML2 avec des annotations MARTE par des transitions parallèles dans la spécification TCPNIA sous une sémantique d'entrelacement. Dans cette approche, seule la durée de l'occurrence d'exécution

est modélisée. Cette dernière est spécifiée par un intervalle de temps associé à une transition du réseau TCPNIA. Dans notre méthode de traduction proposée, le début d'occurrence, la fin d'occurrence et l'occurrence de message (ou spécification d'occurrence de message), qui représentent l'envoi et la réception d'événement, l'appel ou la réception d'opération, sont considérés.

D'autre part, les travaux proposés ont supposé l'hypothèse d'atomicité de l'action imposée par la sémantique d'entrelacement qui traite les comportements parallèles comme leur évolution séquentielle combinée. Pour plus de précisions, considérons l'exemple de la figure 7.2.(a). En utilisant la méthode de traduction proposée dans [YYSQ12], ces activités parallèles sont traduites à la spécification TCPNIA de la figure 7.2.(b).

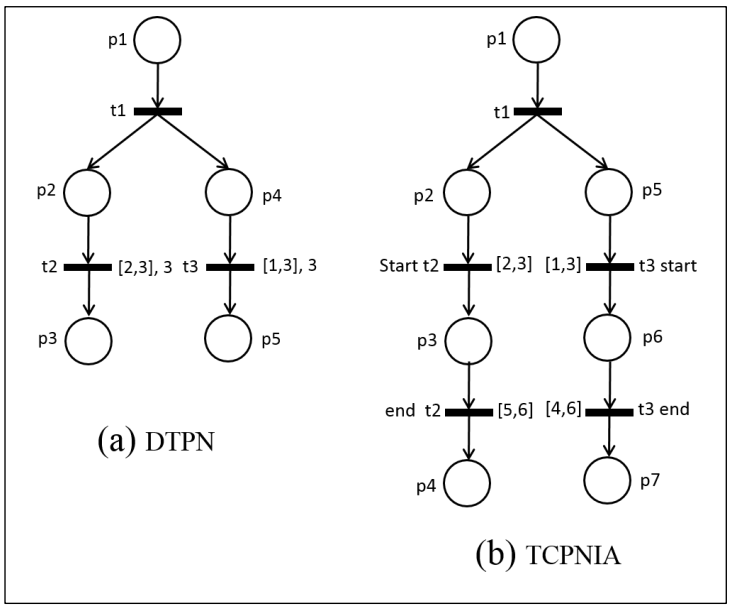


FIGURE 7.2– Méthode d'éclatement d'actions

Pour la vérification formelle des propriétés requises, la spécification de la figure 7.2.(b) est interprétée en automates temporisés (TA). Puisque l'exécution de la transition est instantanée, il n'y a aucun moyen d'observer l'exécution parallèle des deux activités. Comme solution pour une telle situation, il est possible d'interpréter chaque activité ayant une durée non nulle par deux transitions séquentielles modélisant le début et la fin de l'activité. D'après la figure7.2.(b), la durée d'activité est capturée dans l'état intermédiaire conditionnant l'exécution de la transition de fin par le temps écoulé. Nous notons que le système composé peut être dans l'état où les deux transitions de début sont exécutées avant d'exécuter les transitions de fin. Un tel état capture le parallélisme entre les deux activités. Cependant, ces méthodes augmentent le nombre d'états, de transitions et d'horloges ce qui contribue au problème d'explosion de l'espace d'états du graphe de zone correspondant à la spécification d'automate temporisé (TA). En alternative, l'utilisation de deux modèles DTPN et daTA est une solution intéressante. La figure 7.3 montre la structure du modèle sémantique daTA correspondante au modèle DTPN dans la figure 7.2.(a).

En conséquence, nous remarquons que le nombre de transitions de chaque structure est comparable respectivement à ceux de la figure 7.2.(a). Un autre avantage concerne la construction du jeu d'horloges. Dans notre contexte, une horloge est créée dynamiquement lors de la génération des modèles sémantiques avec la réutilisation d'horloges libres. Au contraire, d'autres modèles comme *Timed Automata*, *Petri Nets with Deadlines* et *Time Petri Nets manage* [AD94, LMS04, CR06, DDSS07], au début de modélisation, un nombre fini et constant d'horloges enregistrant le nombre d'actions à exécuter.

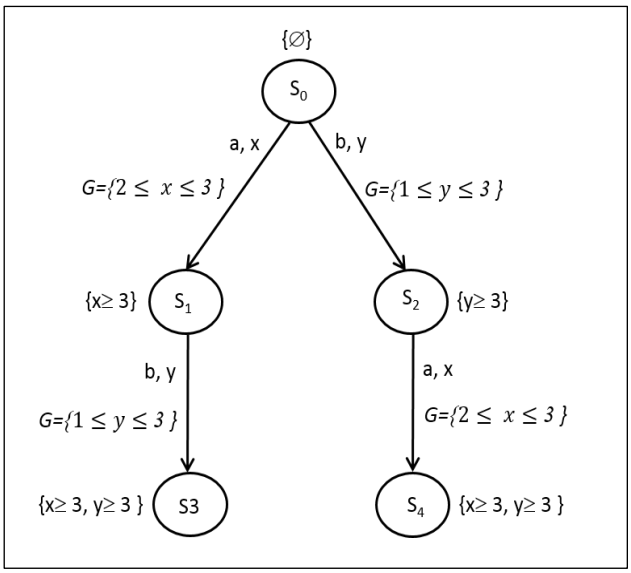


FIGURE 7.3– daTA correspondant à DTPN

7.3 Conclusion

Dans le cadre des travaux de cette thèse, nous nous intéressons à une approche de vérification formelle basée sur un modèle formel d'une sémantique de vrai parallélisme pour exprimer, de façon naturelle, les comportements dynamiques concurrence ou parallèles des STRE. Elle nous permet premièrement, de distinguer entre exécutions séquentielles et exécutions parallèles d'actions. Deuxièmes, de considérer à la fois contrainte temporelles, durées explicites des actions, non atomicité structurelle et temporelle des actions et notion d'urgence. Troisièmes, d'exprimer les durées des actions de façon naturelle sans éclatement des actions. Quatrième, de supporter la création dynamique d'horloges avec la réutilisation d'horloges libres, qui sont utilisées pour spécifier le temps et les contraintes temporelles. Par conséquent, l'approche proposée est plus naturelle, soit pour modéliser les comportements dynamiques et temporels des systèmes temps-réel embarqués, en particulier concurrents et distribués, soit pour valider leurs propriétés.

Conclusion générale et perspectives

L'accroissement de la complexité des systèmes temps-réel et embarqués a nécessité l'augmentation du niveau d'abstraction des langages destinés à leur modélisation. Le profil UML MARTE est le premier exemple des langages semi-formels UML permettant la modélisation cohérente de la plupart des fonctionnalités des systèmes temps réel embarqués. Ce profil fournit à la fois les concepts aussi bien de base qu'avancés inhérents à la modélisation du temps, tels que les stéréotypes permettant la prise en compte des comportements temporels des STRE. Bien qu'étant le profil standard pour modéliser et analyser les STRE, MARTE manque de précision dans sa sémantique, ce qui rend ses spécifications difficilement analysables par une machine. En conséquent, l'analyse des STRE spécifiés par les modèles UML MARTE nécessite leur transformation vers des langages et modèles formels supportant les méthodes formelles et les techniques d'analyse et de vérification formelle. D'autre part, ce travail a abordé un problème crucial de l'utilisation des techniques de vérification par model-checking. Bien qu'il existe un nombre important d'approches et outils de vérification à base de model-checking, leur utilisation reste très limitée. Parmi les raisons qui expliquent ce décalage est que les modèles issus des énoncés formels sont souvent compliqués et parfois même inexploitables si le nombre d'horloges mises en œuvre du système à vérifier est relativement important. En effet, l'espace d'état est exponentiel par rapport au nombre d'horloges.

De ce fait, l'objectif principal de ce travail est de prouver les aptitudes de deux modèles DTPN et daTA à spécifier, de manière intuitive, les comportements des STRE concurrents et parallèles, et particulièrement leurs aptitudes à exprimer la non-atomicité temporelle et structurelle des actions, les durées explicites d'actions, les contraintes temporelles ainsi que les contraintes d'urgence. Ainsi, nous avons proposé une méthode opérationnelle pour translater les spécifications d'un STRE décrites en diagrammes de séquence UML MARTE en modèles DTPNs à des fins de la vérification formelle des propriétés attendues, en particulier celles liées à l'évolution parallèle des actions, qui ont une durée non nulle, sous des contraintes de temps. Les propriétés liées à l'accessibilité peuvent être vérifiées au moyen des outils KRONOS et UPPAAL, et les propriétés spécifiées les comportements de vraie concurrence peuvent être vérifiées à l'aide du model-checker FOCOVE. L'approche proposée nous a permis d'une part, obtenir une spécification DTPN réduite, et d'autre part réduire, dans certains cas éviter le problème de l'explosion combinatoire du nombre d'états de graphe, qui est l'une des principales limites de l'application des model-checking sur des modèles de taille importante, ainsi que réduire le temps de vérification, qui est souvent exponentielle pour les systèmes complexes.

Pour conclure, l'analyse et la vérification formelle des modèles UML MARTE, particulièrement les diagrammes de séquence UML2 annotés par le profil MARTE reste un défi en raison de

la sémantique vague de ces diagrammes. Les travaux sont en cours pour étendre les règles et les procédures de transformation des diagrammes de séquence UML2 avec des annotations MARTE aux modèles formels, ainsi que les concepts avancés de diagrammes séquence UML2.

Perspectives

Comme perspectives dérivées de ce travail de recherche, nous envisageons le développement des points suivants :

— L'étude d'autres exemples de systèmes temps-réel embarqués, tels que les protocoles de communication comme [DSB04], qui est un protocole de multicast fiable dont la conception formelle peut être trouvée dans [SDAB04], afin de mieux tirer profit de l'approche proposée.

— L'approche proposée peut être étendue pour vérifier des autres diagrammes UML2 MARTE similaires comme par exemple, les diagrammes de temps (TD en anglais Timing Diagrams) et les digrammes globaux d'interaction (IOD en anglais Interaction Overview Diagrams) combinant les diagrammes de séquences et les diagrammes d'activités.

— Étant donné qu'une telle représentation formelle n'est pas directement utilisable par les développeurs. Cette approche peut être, soit directement intégrée à l'environnement de développement incorporé et/ou aux outils de génie logiciel assistés par un ordinateur auxquels les développeurs sont habitués. Soit, développer un outil prototype utilisable pour transformer, vérifier et valider un modèle DSMARTE via son modèle DTPN correspondant.

— Développer un model-checker TCTL complet pour la vérification formelle des modèles DTPNs liés aux spécifications des STRE en utilisant le profil MARTE sans passer par les structures de type automate temporisé.

— L'intérêt de l'approche proposée pour la vérification formelle par model-checking des STRE spécifiés par le profil UML MARTE peut aussi être s'exploitée à travers sa combinaison avec d'autres techniques de validation, tel que les tests basés sur les modèles ou les tests logiciel.

Liste des publications

— Chabbat N. & Ghanemi S. *(2019).* A process-oriented verification and validation for real time embedded systems. Synthèse: Revue des Sciences et de la Technologie, *25(1), 72-81*

— Chabbat N., Saidouni DE., Boukharou R., & Ghanemi S. : Using time Petri nets with action duration as a semantics of UML MARTE sequence diagrams. Computing and Informatics, Vol. 39, N: 5, pp. 1-37 (2020).

[

Liste des Acronymes

AADL	Architecture Analysis and Design Language
AGG	Attributed Graph Grammar
API	Application Programming Interface
BDD	Binary Decision Diagram
BMC	Bounded Model-Checker
BOTL	Bidirectional Object-oriented Transformation Language
CMC	Compositional Model-checking
CTL	Computational Tree Logic
daTA	durational action Timed Automata
DTPN	Time Petri Net with Duration action
EMF	Eclipse Modeling Framework
FOCOVE	FOrmal COncurrency Verification Environment
GCM	Generic Component Model
GL	Génie logiciel
GRM	Generic Resource Modeling
HLAM	High Level Application Modeling
HOOD	Hierarchical Object Oriented Design
IDM	Ingénierie Dirigée par les Modèles
KMMM	Kernel Meta Meta Model
KMTF	Kent Model Transformation Language
LTL	Linear Temporal Logic
MARTE	Modeling Analyse Real Time Embedded system
MASCOT	Modular Approach to Software Construction, Operation and Test

MDA	Model Driven Architecture
MIC	Model Integrated Computing
MOF	Meta Object Facility
MTL	Model Transformation Language
NFP	Non-Functional Properties
OCL	Object Constraint Language
OMG	Object Management Group
OMT	Object Modeling Technique
OOSE	Object-Oriented Software Engineering
PAM	Performance Analysis Model
QVT	Query View Transformation
ROBDD	Reduced Ordered Binary Decision Diagram
ROOM	Real time Object Oriented Modeling
SD	Sequence Diagram
SMC	Symbolic Model-checker
SMV	Symbolic Model Verifier
SPT	Schedulability Performance and Time
SysML	Systems Modeling Language
TA	Timed Automata
TCPNIA	Timed Colored Petri Nets with Inhibitor Arcs
TCTL	Timed Computation Tree Logic
UML	Unified Modeling Language
UML-RT	Unified Modeling Language – Real Time
VMTS	Visual Modeling and Transformation System
WMC	Witness Model-Checker
XMF	eXtensible Modeling Framework
XMI	XML Metadata Interchange

YSM	Design Approach for Real Time

Bibliographie

[ACD93] Alur, R., Courcoubetis, C., & Dill, D. (1993). Model-checking in dense real-time. Information and computation, 104(1), 2-34.

[AD90] Alur, R., & Dill, D. (1990, July). Automata for modeling real-time systems. In International Colloquium on Automata, Languages, and Programming (pp. 322-335). Springer, Berlin, Heidelberg.

[AD94] Alur, R., & Dill, D. L. (1994). A theory of timed automata. Theoretical computer science, 126(2), 183-235.

[AdeSK05] Apvrille, L., de Saqui-Sannes, P., & Khendek, F. (2005, March). Synthèse d'une conception UML temps-réel à partir de diagrammes de séquences. In Colloque Francophone sur l'ingénierie des protocols (CFIP'05), Bordeaux, France (Vol. 112).

[AFH94] Alur, R., Fix, L., & Henzinger, T. A. (1994, June). A determinizable class of timed automata. In International Conference on Computer Aided Verification (pp. 1-13). Springer, Berlin, Heidelberg.

[AHH96] Alur, R., Henzinger, T. A., & Ho, P. H. (1996). Automatic symbolic verification of embedded systems. IEEE Transactions on Software Engineering, 22(3), 181-201.

[Alu99] Alur, R. (1999, July). Timed automata. In International Conference on Computer Aided Verification (pp. 8-22). Springer, Berlin, Heidelberg.

[And07] André, C. (2007). Le temps dans le profil UML MARTE. Université Nice Sophia Antipolis.

[And11] André, C. (2011). Modèles de temps et de contraintes temporelles de MARTE et leurs applications.

[Ant01] Antonsson, M. (2001). Modeling of real-time systems in UML with Rational Rose and Rose Real-Time based on RUP.

[APD18] Andrade, V. C., Peres, L. M., & Del Fabro, M. D. (2018, March). Handling Global and Local Time and Energy Constraints of Sequence Diagrams. In 2018 UKSim-AMSS 20th International Conference on Computer Modelling and Simulation (UKSim) (pp. 73-78). IEEE.

[App] https://app.diagrams.net.

[Arn92] Arnold, A., & Arnold, A. (1992). Systèmes de transitions finis et sémantique des processus communicants. Masson.

[Arn94] Arnold, A. (1994). Hypertransition systems. STACS'94, LNCS, Springer-Verlag (pp. 775:327-338.).

[BAPM83] Ben-Ari, M., Pnueli, A., & Manna, Z. (1983). The temporal logic of branching time. Acta informatica, 20(3), 207-226.

[BB04] Bézivin, J., & Briot, J. P. (2004). Sur les principes de base de l'ingénierie des modèles. Obj. Logiciel Base données Réseaux, 10(4), 145-157.

[BB05] Briones, L. B., & Brinksma, E. (2005, September). A test generation framework for quiescent real-time systems. In International Workshop on Formal Approaches to Software Testing (pp. 64-78). Springer, Berlin, Heidelberg.

[BB87] Bolognesi, T., & Brinksma, E. (1987). Introduction to the ISO specification language LOTOS. Computer Networks and ISDN systems, 14(1), 25-59.

[BB91] Baeten, J. C., & Bergstra, J. A. (1991). Real time process algebra. Formal Aspects of Computing, 3(2), 142-188.

[BBD09] Berthomieu, B., Boyer, M., & Diaz, M. (2009, January). Time petri nets. In Petri Nets (pp. 123-161). Wiley-Blackwell.

[BBL+99] Berard, B., Bidoit, M., Laroussinie, F., Cécé, G., Dufourd, C., & Schnoebelen, P. (1999). Vérification de logiciels.

[BCC03] Biere, A., Cimatti, A., Clarke, E. M., Strichman, O., & Zhu, Y. (2003). Bounded model checking.

[BCM+92] Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. L., & Hwang, L. J. (1992). Symbolic model checking: 1020 states and beyond. Information and computation, 98(2), 142-170.

[BCP+01] Bertin, V., Closse, E., Poize, M., Pulou, J., Sifakis, J., Venier, P., & Yovine, S. (2001, December). Taxys= Esterel+ Kronos. A tool for verifying real-time properties of embedded systems. In Proceedings of the 40th IEEE Conference on Decision and Control (Cat. No. 01CH37228) (Vol. 3, pp. 2875-2880). IEEE.

[BCW17] Brambilla, M., Cabot, J., & Wimmer, M. (2017). Model-driven software engineering in practice. Synthesis lectures on software engineering, 3(1), 1-207.

[BD98a] Bosnacki, D., & Dams, D. (1998). Integrating real time into Spin: A prototype implementation. In Formal Description Techniques and Protocol Specification, Testing and Verification (pp. 423-438). Springer, Boston, MA.

[BD98b] Bosnacki, D., & Dams, D. (1998, September). Discrete-time promela and spin. In International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (pp. 307-310). Springer, Berlin, Heidelberg.

[BDFP00] Bouyer, P., Dufourd, C., Fleury, E., & Petit, A. (2000, July). Are timed automata updatable?. In International Conference on Computer Aided Verification (pp. 464-479). Springer, Berlin, Heidelberg.

[BDFP04] Bouyer, P., Dufourd, C., Fleury, E., & Petit, A. (2004). Updatable timed automata. Theoretical Computer Science, 321(2-3), 291-345.

[Bel05] Belala, N. (2005). Formalisation des systèmes temps-réel avec durées d'actions.

[Bel10] Belala, N. (2010). Modèles de temps et leur intérêt à la vérification formelle des systèmes temps-réel (Doctoral dissertation).

[Ber01] Berthomieu, B. (2001, October). La méthode des classes d'états pour l'analyse des réseaux temporels. In 3me congrès Modélisation des Systèmes Réactifs (MSR'2001) (pp. 275-290).

[Béz05a] Bézivin, J. (2005, October). Nets: A first generation model-engineering platform. In International Conference on Model Driven Engineering Languages and Systems (pp. 169-181). Springer, Berlin, Heidelberg.

[Béz05b] Bézivin, J. (2005). Model driven engineering: Principles, scope, deployment and applicability. In Summer School on Generative and Transformational Techniques in Software Engineering (pp. 1-33).

[BFLM18] Bouyer, P., Fahrenberg, U., Larsen, K. G., Markey, N., Ouaknine, J., & Worrell, J. (2018). Model checking real-time systems. In Handbook of Model Checking (pp. 1001-1046). Springer, Cham.

[BFT01] Barbuti, R., De Francesco, N., & Tesei, L. (2001). Timed automata with non-instantaneous actions. Fundamenta Informaticae, 47(3-4), 189-200.

[BJGB10] Bendraou, R., Jezequel, J. M., Gervais, M. P., & Blanc, X. (2010). A comparison of six uml-based languages for software process modeling. IEEE Transactions on Software Engineering, 36(5), 662-675.

[BK85] Bergstra, J. A., & Klop, J. W. (1985). Algebra of communicating processes with abstraction. Theoretical computer science, 37, 77-121.

[BLP+99] Behrmann, G., Larsen, K. G., Pearson, J., Weise, C., & Yi, W. (1999, July). Efficient timed reachability analysis using clock difference diagrams. In International Conference on Computer Aided Verification (pp. 341-353). Springer, Berlin, Heidelberg.

[BM10] Bowles, J., & Meedeniya, D. (2010, November). Formal transformation from sequence diagrams to coloured petri nets. In 2010 Asia Pacic Software Engineering Conference (pp. 216-225). IEEE.

[Bou02] Bouyer, P. (2002). Modèles et algorithmes pour la vérification des systèmes temporisés. These de doctorat, Laboratoire Spécification et Vérification, ENS Cachan, France.

[Bou11] Boulet, P. (2011). Modélisation et analyse de systèmes embarqués ou temps-réel avec le profil UML MARTE.

[Boy01] Boyer, M. (2001). Contribution à la modélisation des systèmes à temps contraint et application au multimédia (Doctoral dissertation, Toulouse 3).

[BPDG98] Bérard, B., Petit, A., Diekert, V., & Gastin, P. (1998). Characterization of the expressive power of silent transitions in timed automata. Fundamenta Informaticae, 36(2, 3), 145-182.

[Bry86] Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. Computers, IEEE Transactions on, 100(8), 677-691.

[Bry92] Bryant, R. E. (1992). Symbolic boolean manipulation with ordered binary-decision diagrams. ACM Computing Surveys (CSUR), 24(3), 293-318.

[BS98] Bornot, S., & Sifakis, J. (1998, April). On the composition of hybrid systems. In International Workshop on Hybrid Systems: Computation and Control (pp. 49-63). Springer, Berlin, Heidelberg.

[BSB13] Belala, N., Saidouni, D. E., Boukharrou, R., Chaouche, A. C., Seraoui, A., & Chachoua, A. (2013). Time petri nets with action duration: a true concurrency real-time model. International Journal of Embedded and Real-Time Communication Systems (IJERTCS), 4(2), 62-83.

[BSI18] Bouneb, M., Saidouni, D. E., & Ilie, J. M. (2018). Hierarchical System Design Using Renable Recursive.

[BST97] Bornot, S., Sifakis, J., & Tripakis, S. (1997, September). Modeling urgency in timed systems. In International Symposium on Compositionality (pp. 103-129). Springer, Berlin, Heidelberg.

[BT00] Brinksma, E., & Tretmans, J. (2000, June). Testing transition systems: An annotated bibliography. In Summer School on Modeling and Verification of Parallel Processes (pp. 187-195). Springer, Berlin, Heidelberg.

[CDD+12] Conquet, E., Dormoy, F. X., Dragomir, I., Graf, S., Lesens, D., Nienaltowski, P., & Ober, I. (2012, February). Formal model driven engineering for space onboard software.

[CDO95] Courtiat, J. P., & De Oliveira, R. C. (1995, October). A reachability analysis of RT-LOTOS specifications. In International Conference on Formal Techniques for Distributed Objects, Components, and Systems (pp. 117-124). Springer, Boston, MA.

[CDS93] Courtiat, J. P., De Camargo, M. S., & Saidouni, D. E. (1993). RT-LOTOS : LOTOS temporisé pour la spécification de systèmes temps réel. Ingénierie des Protocoles (CFIP'93), 427-441.

[CE81] Clarke, E. M., & Emerson, E. A. (1981, May). Design and synthesis of synchronization skeletons using branching time temporal logic. In Workshop on Logic of Programs (pp. 52-71). Springer, Berlin, Heidelberg.

[CES09] Clarke, E. M., Emerson, E. A., & Sifakis, J. (2009). Model checking: algorithmic verification and debugging. Communications of the ACM, 52(11), 74-84.

[CES86] Clarke, E. M., Emerson, E. A., & Sistla, A. P. (1986). Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Transactions on Programming Languages and Systems (TOPLAS), 8(2), 244-263.

[CG19] Chabbat, N., & Ghanemi, S. (2019). A process-oriented verification and validation for real time embedded systems. Synthèse : Revue des Sciences et de la Technologie, 25(1), 72-81.

[CGJV03] Clarke, E., Grumberg, O., Jha, S., Lu, Y., & Veith, H. (2003, July). Counterexample-guided abstraction refinement. In International Conference on Computer Aided Verification (pp. 154-169). Springer, Berlin, Heidelberg.

[CGKP18] Clarke Jr, E. M., Grumberg, O., Kroening, D., Peled, D., & Veith, H. (2018). Model checking. MIT press.

[CGL94] Clarke, E. M., Grumberg, O., & Long, D. E. (1994). Model checking and abstraction. ACM transactions on Programming Languages and Systems (TOPLAS), 16(5), 1512-1542.

[CGP99] Clarke, E. M., & Grumberg, O. (1999). & D. A. Peled. Model Checking. MIT Press, 1, 999.

[CGR99] Cimatti, A., Clarke, E., Giunchiglia, F., & Roveri, M. (1999, July). NuSMV: A new symbolic model verifier. In International conference on computer aided verification (pp. 495-499). Springer, Berlin, Heidelberg.

[CH03] Czarnecki, K., & Helsen, S. (2003). Classification of Model Transformation Approaches, in online proceedings of the 2nd OOPSLA'03 Workshop on Generative Techniques in the Context of MDA. Anaheim, October.

[CK04] Cavarra, A., & Kuster-Filipe, J. (2004, May). Formalizing liveness-enriched sequence diagrams using ASMs. In International Workshop on Abstract State Machines (pp. 62-77). Springer, Berlin, Heidelberg.

[Cle96] Clements, P. C. (1996). A Survey of Architecture Description Languages. IWSSD'96: Proceedings of the 8th International Workshop on Software Specification and Design, Washington, DC, USA. IEEE Computer Society, 16.

[Com08] Combemale, B. (2008). Ingénierie Dirigée par les Modèles (IDM) -État de l'art.

[CR06] Cassez, F., & Roux, O. H. (2006). Structural translation from time Petri nets to timed automata. Journal of Systems and Software, 79(10), 1456-1468.

[CS95] Courtiat, J. P., & Saïdouni, D. E. (1995). Relating maximality-based semantics to action refinement in process algebras. In Formal Description Techniques VII (pp. 293-308). Springer, Boston, MA.

[CY92] Courcoubetis, C., & Yannakakis, M. (1992). Minimum and maximum delay problems in real-time systems. Formal Methods in System Design, 1(4), 385-415.

[DD06] Dutertre, B., & De Moura, L. (2006). The yices smt solver. Tool paper at http://yices. csl. sri. Com/tool-paper. pdf, 2(2), 1-2.

[DDSS07] D'Aprile, D., Donatelli, S., Sangnier, A., & Sproston, J. (2007, March). From time Petri nets to timed automata: An untimed approach. In International Conference on Tools and Algorithms for the Construction and Analysis of Systems (pp. 216-230). Springer, Berlin, Heidelberg.

[Del99] De logiciels, Vérification. (1999). Techniques et outils du model-checking. Ouvrage collectiv–coordination Philippe Schnoebelen. Vuibert Informatique. Paris.

[deSA11] de Saqui-Sannes, P., & Apvrille, L. (2011). AVATAR/TTool: un environnement en mode libre pour SysML temps réel.

[Dev92] Devillers, R. (1992). Maximality preservation and the ST-idea for action refinements. In Advances in Petri Nets 1992 (pp. 108-151). Springer, Berlin, Heidelberg.

[DGG93] Dams, D., Grumberg, O., & Gerth, R. (1993, June). Generation of reduced models for checking fragments of CTL. In International Conference on Computer Aided Verification (pp. 479-490). Springer, Berlin, Heidelberg.

[DLC10] Diaw, S., Lbath, R., & Coulette, B. (2010). État de l'art sur le développement logiciel basé sur les transformations de modèles. Technique et Science Informatiques, 29(4-5), 505-536.

[DOR+04] De Moura, L., Owre, S., Rueß, H., Rushby, J., Shankar, N., Sorea, M., & Tiwari, A. (2004, July). SAL 2. In International Conference on Computer Aided Verification (pp. 496-500). Springer, Berlin, Heidelberg.

[Dot11] Dotty (2011). Graphviz home page. http ://www.graphviz.org/.

[DR97] Deo, M. C., & Rao, T. V. M (1997). Publications and Outreach Activities.

[DTY95] Daws, C., Olivero, A., Tripakis, S., & Yovine, S. (1995, October). The tool KRONOS. In International Hybrid Systems Workshop (pp. 208-219). Springer, Berlin, Heidelberg.

[Duf91] Duffy, D. A. (1991). Principles of automated theorem proving. John Wiley & Sons, Inc.

[EC82] Emerson, E. A., & Clarke, E. M. (1982). Using branching time temporal logic to synthesize synchronization skeletons. Science of Computer programming, 2(3), 241-266.

[EFS+05] Eichner, C., Fleischhack, H., Meyer, R., Schrimpf, U., & Stehno, C. (2005, June). Compositional semantics for UML 2.0 sequence diagrams using Petri Nets. In International SDL Forum (pp. 133-148). Springer, Berlin, Heidelberg.

[EL03] Edwards, S. A., & Lee, E. A. (2003). The semantics and execution of a synchronous block-diagram language. Science of Computer Programming, 48(1), 21-42.

[Eme90] Emerson, E. A. (1990). Temporal and modal logic. In Formal Models and Semantics (pp. 995-1072). Elsevier.

[EMP14] E. M. Projet. (2014, February). Eclipse modeling framework (emf), https : //www.eclipse.org/modeling/emf/.

[Fle02] Fleury, E. (2002). Les automates temporisés avec mises à jour (Doctoral dissertation).

[FLV06] Feiler, P. H., Lewis, B. A., & Vestal, S. (2006, October). The SAE Architecture Analysis & Design Language (AADL) a standard for engineering performance critical systems. In 2006, IEEE Conference on Computer Aided Control System Design.

[FMS06] Friedenthal, S., Moore, A., & Steiner, R. (2006, July). OMG systems modeling language (OMG SysML) tutorial. In INCOSE Intl. Symp (Vol. 9, pp. 65-67).

[FOW01] Fischer, C., Olderog, E. R., & Wehrheim, H. (2001, April). A CSP view on UML-RT structure diagrams. In International Conference on Fundamental Approaches to Software Engineering (pp. 91-108). Springer, Berlin, Heidelberg.

[FSW12] Feiler, P. H., Seibel, J. R., & Wrage, L. (2012). What's New in V2 of the Architecture Analysis & Design Language Standard?. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST.

[FTJ+07] Fernandes, J. M., Tjell, S., Jorgensen, J. B., & Ribeiro, O. (2007, May). Designing tool support for translating use cases and uml 2.0 sequence diagrams into a coloured petri net. In Sixth International Workshop on Scenarios and State Machines (SCESM'07: ICSE Workshops 2007) (pp. 2-2). IEEE.

[Gar13] Garredu, S. (2013). Approche de méta-modélisation et transformations de modèles dans le contexte de la modélisation et simulation à évènements discrets : application au formalisme DEVS (Doctoral dissertation).

[GCKS06] Groce, A., Chaki, S., Kroening, D., & Strichman, O. (2006). Error explanation with distance metrics. International Journal on Software Tools for Technology Transfer, 8(3), 229-247.

[Gee15] Geensyde. Outils aadl. http ://www.geensyde.fr/Outils-AADL.html (visité en Juin 2015).

[Gér07] Gérard, S. (2007). MARTE: A new standard for modeling and analysis of real-time and embedded systems. In Proc. of Euromicro Conf. on Real-Time Systems (ECRTS 07), Pisa, Italy, Jul.

[GLH+96] Godefroid, P., van Leeuwen, J., Hartmanis, J., Goos, G., & Wolper, P. (1996). Partial-order methods for the verification of concurrent systems: an approach to the state-explosion problem (Vol. 1032). Heidelberg : Springer.

[Gli09] Glitia, C. (2009). Optimisation des applications de traitement systématique intensives sur systems-on-chip (Doctoral dissertation, Lille 1).

[God90] Godefroid, P. (1990, June). Using partial orders to improve automatic verification methods. In International Conference on Computer Aided Verification (pp. 176-185). Springer, Berlin, Heidelberg.

[Gom06] Gomaa, H. (2006, May). Designing concurrent, distributed, and real-time applications with UML. In Proceedings of the 28th international conference on Software engineering (pp. 1059-1060).

[Gom84] Gomaa, H. (1984). A software design method for real-time systems. Communications of the ACM, 27(9), 938-949.

[GPC12] Ge, N., Pantel, M., & Crégut, X. (2012). Time properties dedicated transformation from UML-MARTE activity to time transition system. *ACM SIGSOFT Software Engineering Notes*, *37*(4), 1-8.

[GRR06] Gardey, G., Roux, O. H., & Roux, O. F. (2006). State space computation and analysis of time Petri nets. Theory and Practice of Logic Programming, 6(3), 301-320.

[GS04] Greenfield, J., & Short, K. (2004). Software factories: Assembling applications with patterns, frameworks, models and tools.

[Ham06] Hammal, Y. (2006, September). Branching time semantics for UML 2.0 sequence diagrams. In International Conference on Formal Techniques for Networked and Distributed Systems (pp. 259-274). Springer, Berlin, Heidelberg.

[Her01] Herbreteau, F. (2001). Automates à file réactifs embarqués : application à la vérification de systèmes temps-réel (Doctoral dissertation, Nantes).

[HHR+05] Haugen, ∅., Husa, K. E., Runde, R. K., & St∅len, K. (2005). STAIRS towards formal design with sequence diagrams. Software & Systems Modeling, 4(4), 355.

[HHW97] Henzinger, T. A., Ho, P. H., & Wong-Toi, H. (1997). HyTech: A model checker for hybrid systems. International Journal on Software Tools for Technology Transfer, 1(1-2), 110-122.

[HI98] Haddad, S., & Ilié, J. M. (1998, March). Exploiting symmetry in linear time temporal logic model checking: One-step beyond. In International Conference on Tools and Algorithms for the Construction and Analysis of Systems (pp. 52-67). Springer, Berlin, Heidelberg.

[Her89] Hervé Hillion, M. (1989). "Modélisation et analyse des systèmes de production discrets par les réseaux de Petri.». Thèse de Doctorat, L'université Pierre et Marie Curie Paris VI, 23 Janvier.

[HKP02] Huet, G., Kahn, G., & Paulin-Mohring, C. (2002). The Coq proof assistant: a tutorial: version 7.2.

[HM08] Harel, D., & Maoz, S. (2008). Assert and negate revisited: Modal semantics for UML sequence diagrams. Software & Systems Modeling, 7(2), 237-252.

[HNSY94] Henzinger, T. A., Nicollin, X., Sifakis, J., & Yovine, S. (1994). Symbolic model checking for real-time systems. Cornell University.

[Hoa78] Hoare, C. A. R. (1978). Communicating sequential processes. Communications of the ACM, 21(8), 666-677.

[Hoa85] Hoare, C. A. R. (1985). Communicating Sequential Processes. Prentice-Hall. Englewood Cliffs, NJ, USA.

[Hol97] Holzmann, G. J. (1997). The model checker SPIN. IEEE Transactions on software engineering, 23(5), 279-295.

[ISO88] ISO/IEC. (1988, September). LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour, International Standard 8807. International

Organisation of Standardization- Information Processing Systems-Open Systems Interconnection, Genève.

[Jen13] Jensen, K. (2013). Coloured Petri nets: basic concepts, analysis methods and practical use (Vol. 1). Springer Science & Business Media.

[JGB06] Jézéquel, J.-M., Gérard, S., and Baudry, B. (2006). Le génie logiciel et l'idm : une approche unificatrice par les modèles. L'ingénierie dirigée par les modèles.

[JS05] Jantsch, A., & Sander, I. (2005). Models of computation and languages for embedded system design. *IEE Proceedings-Computers and Digital Techniques*, *152*(2), 114-129.

[KDC96] Khansa, W., Denat, J. P., & Collart-Dutilleul, S. (1996, August). P-Time Petri Nets for manufacturing systems. In International Workshop on Discrete Event Systems, WODES (Vol. 96, pp. 94-102).

[Kha97] Khansa, W. (1997). Réseaux de Petri P-temporels : Contribution à l'étude des systèmes à événements discrets (Doctoral dissertation, Université Savoie Mont Blanc).

[KHBS] Kitouni, I., Hachichi, H., Bouaroudj, K., & Saidouni, D. E. (2012). Durational actions timed automata: determinization and expressiveness. International Journal of Applied Information Systems (IJAIS), 4(2), 1-11.

[KSdeV00] Kocik, R., Sorel, Y., & de Voluceau BP105, I. R. D. (2000, March). De la modélisation à la réalisation : réduction du cycle de développement des applications temps réel distribuées. In RTS2000 8th conference on Real-time and embedded systems (Vol. 29).

[Kus06] Kuster-Filipe, J. (2006). Modelling concurrent interactions. Theoretical Computer Science, 351(2), 203-220.

[KWWB03] Kleppe, A. G., Warmer, J., Warmer, J. B., & Bast, W. (2003). MDA explained: the model driven architecture: practice and promise. Addison-Wesley Professional.

[Lab10] Labrie, M. (2010). Usine logicielle de composants de simulation de procédés CAPE-OPEN.

[Lam83] Lamport, L. (1983, September). What good is temporal logic?. In IFIP congress (Vol. 83, pp. 657-668).

[Las08] Lasnier, G. (2008). Étude et Support du Standard AADLv2 dans Ocarina (Doctoral dissertation, Master's thesis, Université Pierre & Marie Curie, Paris VI).

[LL98] Laroussinie, F., & Larsen, K. G. (1998). CMC: A tool for compositional model-checking of real-time systems. In Formal Description Techniques and Protocol Specification, Testing and Verification (pp. 439-456). Springer, Boston, MA.

[LL98] Léonard, L., & Leduc, G. (1998). A formal definition of time in LOTOS. Formal Aspects of Computing, 10(3), 248-266.

[LMS04] Laroussinie, F., Markey, N., & Schnoebelen, P. (2004, August): Model checking timed automata with one or two clocks. In International Conference on Concurrency Theory (pp. 387-401). Springer, Berlin, Heidelberg.

[Loh02] Lohr, C. (2002). Contribution à la conception de systèmes temps-réel s'appuyant sur la technique de description formelle RT-LOTOS (Doctoral dissertation).

[LP97] Larsen, K. G., Pettersson, P., & Yi, W. (1997). UPPAAL in a nutshell. International journal on software tools for technology transfer, 1(1-2), 134-152.

[McM93] McMillan, K. L. (1993). Symbolic model checking. In Symbolic Model Checking (pp. 25-60). Springer, Boston, MA.

[MDDM16] Menad, N., Dhaussy, P., Drey, Z., & Mekki, R. (2016). Towards a transformation approach of timed uml marte specifications for observer-based formal verification. Computing and Informatics, 35(2), 338-368.

[Mer74] Merlin, P. M. (1974). A study of the recoverability of computing systems PhD thesis. Department of Information and Computer Science, University of California, Irvine, CA.

[Mil82] Milner, R. (1982). A calculus of communicating systems. (pp 30-94). Springer-Verlag, New York, USA.

[Mil89] Milner, R. (1989). Communication and concurrency (Vol. 84). Englewood Cliffs: Prentice hall.

[MNKT13] Monthé, V., Nana, L., Kouamou, G., & Tangha, C. (2013, August). A comparison approach for the choise of languages and methods for real-time and embedded systems design: application to robotic systems. In Proceedings of EWiLi'13, the 3rd embedded operating workshop (Vol. 94).

[Mok06] Mokadem, H. B. (2006). Vérification des propriétés temporisées des automates programmables industriels (Doctoral dissertation).

[MPW92] Milner, R., Parrow, J., & Walker, D. (1992). A calculus of mobile processes. Information and computation, 100(1), 1-40.

[MT00] Medvidovic, N., & Taylor, R. N. (2000). A classification and comparison framework for software architecture description languages. IEEE Transactions on software engineering, 26(1), 70-93.

[MT90] Moller, F., & Tofts, C. (1990, August). A temporal calculus of communicating systems. In International Conference on Concurrency Theory (pp. 401-415). Springer, Berlin, Heidelberg.

[MVG06] Mens, T., & Van Gorp, P. (2006). A taxonomy of model transformation. Electronic notes in theoretical computer science, 152, 125-142.

[MW08] Micskei, Z., & Waeselynck, H. (2008). UML 2.0 sequence diagrams semantics.

[NH10] Nian-hua, Y., & Hui-qun, Y. (2010). Modeling and verification of embedded systems using timed colored Petri net with inhibitor arcs. Journal of East China University of Science and Technology, 36(3), 411-417.

[NSY93] Nicollin, X., Sifakis, J., & Yovine, S. (1993). From ATP to timed graphs and hybrid systems. Acta informatica, 30(2), 181-202.

[Odd03] Oddoux, D. (2003). Utilisation des automates alternants pour un model-checking efficace des logiques temporelles linéaires (Doctoral dissertation, Paris 7).

[OMG03] O. M. Group. Uml 2 infrastructure (final adopted specification). (septembre 2003). http://www.omg.org/cgi-bin/doc?ptc/03-09-15.pdf.

[OMG05] OMG, U. (2005). Profile for Schedulability. Performance, and Time Specification, In Version 1.1.

[OMG08] OMG, A. (2008). Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems, OMG document number.

[OMG11] OMG, Q. (2011). Meta Object Facility (MOF) 2.0 Query/View/Transformation, v1. 1.

[OMG15] O. M. Group. Unified modeling language (uml) v2.50. (June 2015). http://www.omg.org/spec/UML/2.5/.

[Pel94] Peled, D. (1994, June). Combining partial order reductions with on-the-fly model-checking. In International Conference on Computer Aided Verification (pp. 377-390). Springer, Berlin, Heidelberg.

[Car62] Carl, A. (1962). Petri. kommunikation mit automaten. PhD, University of Bonn, West Germany.

[PF05] Perry, D., & Foster, H. (2005). Applied formal verification. McGraw-Hill, Inc.

[PGC12] Panel, M. Ge, N., & Crégut, X. (2012). A framework dedicated to time properties verification for UML-MARTE specifications. Conférence en IngénieriE du Logiciel (CIEL).

[Pnu77] Pnueli, A. (1977, September). The temporal logic of programs. In 18th Annual Symposium on Foundations of Computer Science (sfcs 1977) (pp. 46-57). IEEE.

[Poe06] Poernomo, I. (2006, April). The meta-object facility typed. In Proceedings of the 2006 ACM symposium on Applied computing (pp. 1845-1849).

[Pri03] Prigent, A. (2003). Le test des systèmes temps-réel paramétrés : application à la conception d'architectures avioniques (Doctoral dissertation, Nantes).

[Pro] http://projects.laas.fr/fiacre/papers.php.

[QS82] Queille, J. P., & Sifakis, J. (1982, April). Specification and verification of concurrent systems in CESAR. In International Symposium on programming (pp. 337-351). Springer, Berlin, Heidelberg.

[QS82a] Queille, J. P., & Sifakis, J. (1982, November). A temporal logic to deal with fairness in transition systems. In 23rd Annual Symposium on Foundations of Computer Science (sfcs 1982) (pp. 217-225). IEEE.

[Ram74] Ramchandani, C. (1974). Analysis of asynchronous concurrent systems by Petri nets (No. MAC-TR-120). MASSACHUSETTS INST OF TECH CAMBRIDGE PROJECT MAC.

[RB02] Ribet, P. O., & Berthomieu, B. (2002, November). On combining the persistent sets method with the covering steps graph method. In International Conference on Formal Techniques for Networked and Distributed Systems (pp. 344-359). Springer, Berlin, Heidelberg.

[RH04] Ruys, T. C., & Holzmann, G. J. (2004, April). Advanced spin tutorial. In International SPIN Workshop on Model Checking of Software (pp. 304-305). Springer, Berlin, Heidelberg.

[RHCF05] Rayner, M., Hockey, B. A., Chatzichrisafis, N., & Farrell, K. (2005). OMG unified modeling language specification. In Version 1.3, c 1999 Object Management Group, Inc.

[Ric62] Richard, J. (1962). Büchi. On a decision method in restricted second-order arithmetic. In Proceedings of the International Congress on Logic, Math, and Philosophy of Science. Standford University Press.

[Ros98] Rosen, J. (1998). Symmetry discovered Concepts and applications in nature and science. Courier Corporation.

[RR88] Roscoe, A. W., & Reed, G. M. (1988). A timed model for communicating sequential processes. Theoretical Computer Science, 58.

[Rus01] Rushby, J.M. (2001). Theorem proving for verification. Proceedings of Modelling and verification of parallel process (MOVEP2k). Lecture notes in Computer Science (Vol. 2067, pp. 39-57). Springer, Verlag.

[Sai96] Saidouni, D. (1996). Sémantique de maximalité : application au raffinement d'actions dans LOTOS (Doctoral dissertation).

[Sal09] Salles, R. (2009). Model-checking de programmes Java Rapport de TER.

[Sau63] Saul, A. (1963). Kripke. Semantical analysis of modal logic I: Normal modal propositional calculi. Zeitschrift für mathematische Logik und Grundlagen der Mathematik, 9(5-6), 67-96.

[SB03] Saïdouni, D. E., & Belala, N. (2003, October). Vérification de propriétés exprimées en ctl sur le modèle des systèmes de transitions étiquetées maximales. In CIP'2003, Conférence Internationale de Productique.

[SB04] Saïdouni, D. E., & Belala, N. (2004). Straightforward adaptation of interleaving based solutions for true concurrency-based logic verification approaches. In Proceedings of International Conference on Complex Systems (CISC'2004).

[SB05] Saïdouni, D. E., & Belala, N. (2005). Using Maximality-Based Labelled Transition System Model for Concurrency Logic Verification. Int. Arab J. Inf. Technol., 2(3), 199-205.

[SB06] Saïdouni, D. E., & Belala, N. (2006, December). Actions duration in timed models. In The International Arab Conference on Information Technology (ACIT).

[SBB08] Saïdouni, D. E., Benamira, A., Belala, N., & Arfi, F. (2008, June). FOCOVE: Formal concurrency verification environment for complex systems. In AIP Conference Proceedings (Vol. 1019, No. 1, pp. 375-380). American Institute of Physics.

[SDAB04] Saıdouni, D. E., Derdouri, L., Alidra, A., & Belala, N. (2004). Conception formelle du protocole AMRHy. Technical report, Département d'Informatique, Université Mentouri, 25000 Constantine, Algérie.

[SEI] SEI. Aadl tools. https://wiki.sei.cmu.edu/aadl/index.php/AADL_tools#AADL Tools_Summary_Paper.

[Sel03] Selic, B. (2003). The pragmatics of model-driven development. IEEE software, 20(5), 19-25.

[Sif77] Sifakis, J. (1977). Use of Petri nets for performance evaluation in measuring modelling and evaluating computer systems.

[Sif79] Sifakis, J. (1979). Le contrôle des systèmes asynchrones : concepts, propriétés, analyse statique (Doctoral dissertation).

[Sif80] Sifakis, J. (1980). Performance evaluation of systems using nets. In Net Theory and Applications (pp. 307-319). Springer, Berlin, Heidelberg.

[Shi13] Shiraishi, S. I. (2013). Qualitative comparison of adl-based approaches to real-world automotive system development. Information and Media Technologies, 8(1), 196-207.

[SK97] Sztipanovits, J., & Karsai, G. (1997). Model-integrated computing. Computer, 30(4), 110-111.

[Sol00] Soley, R. (2000). Model driven architecture. OMG white paper, 308(308), 5. http ://www.omg.org/mda/.

[Spe07] Specification, O. A. (2007). Omg unified modeling language (omg uml), superstructure, v2. 1.2. Object Management Group, 70.

[SSS00] Sheeran, M., Singh, S., & Stålmarck, G. (2000, November). Checking safety properties using induction and a SAT-solver. In International conference on formal methods in computer-aided design (pp. 127-144). Springer, Berlin, Heidelberg.

[StoS04] Storrle, H. (2004). Trace semantics of interactions in UML 2.0. J. Visual Languages and Computing.

[Tea06] Team, S. (2006). Systems modeling language (SysML) specification. OMG document: ad.

[Val90] Valmari, A. (1990, June). Stubborn sets for reduced state space generation. In International Conference on Application and Theory of Petri Nets (pp. 491-515). Springer, Berlin, Heidelberg.

[VAM96] Vernadat, F., Azéma, P., & Michel, F. (1996, June). Covering step graph. In International Conference on Application and Theory of Petri Nets (pp. 516-535). Springer, Berlin, Heidelberg.

[VdA92] Van der Aalst, W. M. P. (1992). Timed coloured Petri nets and their application to logistics.

[Ver06] Vergnaud, T. (2006). Modélisation des systèmes temps-réel répartis embarqués pour la génération automatique d'applications formellement vérifiées (Doctoral dissertation).

[VJ04] Vojtisek, D., & Jézéquel, J. M. (2004). MTL and Umlaut NG-Engine and framework for model transformation.

[VSB+13] Völter, M., Stahl, T., Bettin, J., Haase, A., & Helsen, S. (2013). Model-driven software development: technology, engineering, management. John Wiley & Sons.

[Wal] Walter, B. (1983, May). Timed Petri-Nets for Modelling and Analyzing Protocols with Real-Time Characteristics. In Protocol Specification, Testing, and Verification (pp. 149-159).

[Wan02] Wang, F. (2002, August). Symbolic verification of complex real-time systems with clock-restriction diagram. In International Conference on Formal Techniques for Networked and Distributed Systems (pp. 235-250). Springer, Boston, MA.

[YM11] Yin, L., & Mallet, F. (2011). Correct transformation from ccsl to promela for verification. Technical Report 7491, INRIA.

[YYSQ12] Yang, N., Yu, H., Sun, H., & Qian, Z. (2012). Modeling UML sequence diagrams using extended Petri nets. Telecommunication Systems, 51(2-3), 147-158.

[Zeb91] Zuberek, W. M. (1991). Timed Petri nets definitions, properties, and applications. Microelectronics Reliability, 31(4), 627-644. LAAS technical report no. 08389, 37.