Génération de scénarios de tests pour la vérification de systèmes complexes et répartis : application au système européen de signalisation ferroviaire (ERTMS)

Sana Jabri

ECOLE CENTRALE DE LILLE

N° d'ordre : 123

THESE

Présentée en vue d'obtenir le grade de

DOCTEUR

En

AUTOMATIQUE ET INFORMATIQUE INDUSTRIELLE

Par

Sana JABRI

DOCTORAT DELIVRE PAR L'ECOLE CENTRALE DE LILLE

Titre de la thèse :

GENERATION DE SCENARIOS DE TESTS POUR LA VERIFICATION DE SYSTEMES REPARTIS : APPLICATION AU SYSTEME EUROPEEN DE SIGNALISATION FERROVIAIRE (ERTMS)

Soutenue le 22 Juin 2010 devant le jury d'examen :

Président	*M. Armand TOGUYENI, Professeur à l'Ecole Centrale de Lille*
Rapporteur	*M. Abderrafiâa KOUKAM, Professeur à l'Université de Technologie de Belfort-Montbéliard*
Rapporteur	*M. Christos PYRGIDIS, Professeur à Aristotle University of Thessaloniki*
Membre	*Mme. Nathalie DUQUENNE, « Project Officer in European Railway Agency »*
Membre	*M. Alexandre GIRARDI, Chef du département certification à Multitel*
Membre	*M. Thomas BOURDEAUD'HUY, Maître de conférences à l'Ecole Centrale de Lille*
Membre	*M. Etienne LEMAIRE, Ingénieur de recherche à l'INRETS*
Directeur de thèse	*M. El Miloudi EL KOURSI, Directeur d'Unité de recherche ESTAS à l'INRETS*
Co-directeur de thèse	*M. Pascal YIM, Directeur Open Technologies*

Thèse préparée dans les Laboratoires LAGIS de l'Ecole Centrale de Lille et ESTAS de l'INRETS
LAGIS., UMR 8146 – École Centrale de Lille
ESTAS, 20 Rue Elisée Reclus, INRETS
Ecole Doctorale SPI 072

A mon papa adoré

Tu es mon maître et mon idole. J'espère être à la hauteur de tout ce que tu m'apporte dans la vie. Ton soutien sans faille et ta confiance en moi m'ont permis d'arriver là où je suis.

A ma maman chérie

C'est grâce à ton amour, ton affection, ta protection et ton soutien que j'ai pu réussir et surmonter les moments difficiles. Que ce travail soit la reconnaissance de tous tes sacrifices.

A mes sœurs et à mon petit frère

Vous êtes le rayon de soleil de ma vie. Je vous suis reconnaissante pour votre amour, votre soutien, votre joie de vivre et surtout notre complicité.

A mes amies Meriem et Insaf

A mon ami David

Vous comptez énormément pour moi…

A toute ma famille & mes amis.

Ce qui se conçoit bien s'énonce clairement et les
mots pour le dire arrivent aisément

N.Boileau, L'Art poétique

Remerciements

Je tiens d'abord à remercier El Miloudi EL KOURSI de m'avoir accueillie au sein de l'équipe de recherche ESTAS afin d'accomplir mes travaux de thèse, de m'avoir guidée dès les premiers jours et de m'avoir soutenue dans les moments difficiles. Je le remercie vivement pour ses conseils avisés et ses lectures attentives de mon mémoire.

J'adresse aussi un grand merci à Pascal YIM, qui a su m'orienter dans les bonnes directions.

J'exprime ma profonde reconnaissance à Thomas BOURDEAUD'HUY pour avoir dirigé mes travaux de thèse, pour sa passion, ses conseils, pour m'avoir consacré une partie importante de son temps et pour m'avoir aidée à améliorer mon travail. Il a su garder le cap lorsque personnellement je doutais.

Je remercie tout particulièrement Etienne LEMAIRE, qui a dirigé mes travaux de thèse, m'a fait découvrir le monde ferroviaire, m'a guidée depuis mon stage de Master jusqu'à la fin de ma thèse, et m'a aidée à avancer dans chaque étape de mon travail. Je lui suis reconnaissante pour toutes ses explications concernant le système ERTMS.

El Miloudi, Pascal, Thomas et Etienne, vous m'avez invitée à une grande aventure que vous avez conduite d'une main de maître en instaurant un climat de confiance, de patience, d'amitié et de bienveillance. Ce travail est aussi le vôtre.

***

Les travaux développés dans cette thèse ont été financés par une bourse INRETS et Région Nord Pas de Calais. Je les remercie de m'avoir donné la chance de réaliser ma thèse dans de très bonnes conditions. Je remercie également le groupe CISIT (Campus International on Safety and Intermodality in Transportation) qui contribue à la prééminence de la région dans le secteur des transports et dont l'INRETS est membre.

Je remercie sincèrement mes rapporteurs : Messieurs Christos PYRGIDIS et Abderrafiâa KOUKAM qui ont accepté de plancher sur ce travail. Ils ont su tirer sa qualité vers le haut à travers leurs précieuses remarques.

Je remercie également les membres du Jury : M. Armand TOGUYENI, Mme. Nathalie DUQUENNE et M. Alexandre GIRARDI d'avoir bien voulu participer à la soutenance.

***

Je remercie chaleureusement tous les membres de l'INRETS, permanents, doctorants et stagiaires. J'ai toujours adoré notre ambiance « familiale » et les discussions très ouvertes sur divers thèmes. Il m'est particulièrement agréable de remercier : mon amie Meriem pour son soutien, ses encouragements et pour les belles années que nous avons passées ensemble à ESTAS ; mon ami Olivier.L et sa famille (Chris, Orlane et Sélène) pour les bons moments que j'ai partagés avec eux ; ma collègue Nathalie pour son aide et sa compréhension ; mes collègues Philippe, Sonia, Georges, Greg, Isabelle, Olivier.D, Victor, Valérie, Latifa, Bernard, Daniel, pour leur amitié et leur soutien ; enfin mes collègues doctorants Ahmed, François, Nizar, Malik et Sabrine pour les joies et les peines que j'ai partagées avec eux.

***

Une pensée particulière va vers mes parents Rafika et Abdelhamid, ma sœur Asma et mon frère Mohamed dont l'éloignement est très pesant mais tout aussi motivant. Leurs sacrifices m'ont permis d'arriver là où je suis et leur fierté est un carburant infini d'avancement et de dépassement de soi. Je vous remercie de votre affection et de votre soutien inconditionnel dans mes choix de carrières. Vous me faites grandir chaque jour.

Je remercie particulièrement mon père pour ses lectures attentives de mon mémoire et ma mère pour ses bons petits plats qui m'ont encouragée à finir ma rédaction.

Un grand merci à mes deux familles Boukahla et Jabri d'avoir cru en moi. Je remercie en particulier ma tante Bchira pour son amour, ses encouragements et ses bons gâteaux ; ma tante Rawda, mes oncles Samir et Mustapha pour leur soutien et leur confiance en moi ; ma tante Souad et mon oncle Mohamed pour tout leur support pendant mes années d'études primaires et secondaires ; mes cousines et mes grands-mères pour tout leur amour.

***

J'exprime ma profonde gratitude envers ma deuxième famille en France : Ghislaine, Nadine, Delphine et Fabien.L, Léonard ; Delphine et David.N, Mathilde, Maelle ; Martine et Jacques pour leur accueil, leurs encouragements et tous les bons moments que nous avons partagés ensemble lors d'une sortie ou autour d'un bon repas.

***

Je remercie sincèrement mon amie Insaf qui m'a soutenue durant mes années de thèse, m'a toujours encouragée et m'a rendue confiance en moi dans les moments difficiles. Je m'adresse aussi à mes amis de longue date (Mariem.D, Randa.M, Anis.F) pour les remercier de leur soutien. Je remercie enfin mes amis en France qui grâce à eux je me suis toujours sentie en famille (Aymen, Benoit, Mélanie, Maxime, Agnès, Maelle, Yoann, Caroline, Anaïs, Pierres Yves, Meriem.H, Tarik.H, Sanaa.B, Sihem.B, Tarik.C, Hamza.H, Ibrahim.H…)

***

Enfin, je remercie David, qui a tout partagé avec moi ces dernières années, pour son soutien permanent, sa compréhension, sa patience et pour avoir supporté mes humeurs, souvent angoissés ces derniers temps.

Table des matières

Liste des figures

Liste des tableaux

Lexique ferroviaire

A /B	C
ACRUDA : certification des architectures matérielles et mise en place des règles de certification des calculateurs, embarqués ou fixes, développés en Europe. **AEIF** : Association Européenne d'Interopérabilité Ferroviaire **BTM** : Balise Transmission Module	**C/C et Sig** : Contrôle-Commande et Signalisation **CEN** : Comité Européen de Normalisation **CENELEC** : Comité Européen de Normalisation Electrotechnique **CER** : Ensemble des entreprises ferroviaires européennes **CERTIFER** : Organisme de certification de produits et de systèmes pour les transports guidés terrestres

D	E
DD: Dossier de définition **Directive 96/48/CE**: Interopérabilité du système ferroviaire européen à grande vitesse **Directive 2001/16/CE** : Interopérabilité du système ferroviaire européen conventionnel. **Directive 2004/50/CE** : Directive du Parlement Européen et du Conseil du 29 avril 2004 **Directive 2006/679/CE** : STI relatives au sous-système contrôle-commande et signalisation **Directive 91/440/CEE** : Directive du Parlement Européen et du Conseil, du 29 juillet 1991 concernant la séparation des missions des entreprises ferroviaires de celles des gestionnaires d'infrastructures **DPS** : Dossier Préliminaire de Sécurité **DS** : Dossier de Sécurité	**EIM**: Ensemble des gestionnaires d'infrastructures européens **EMF**: Eclipse Modelling Framework **EOQA** : Expert ou Organisme Qualifié Agrée **ERA** : European Railway Agency **ERTMS**: European Railway Traffic Management System **ETCS**: European Train Control System **ETML**: European Trafic Management Layer **ETSI** : Institut européen des normes de télécommunications **EVC**: European Vital Computer

F	G
FDMS : Fiabilité, Disponibilité, Maintenabilité, et Sécurité	**GI** : Gestionnaire d'Infrastructure **GRRT** : Groupement Régional Nord pas

FRS : Functional Requirements specifications
FS : mode Full Supervision

de Calais pour la Recherche dans les Transports
GSM–R: GSM for Railways

H / I / L / M / O / R	S
HEROE : Harmonisation des règles d'exploitation d'ERTMS	**SAMNET**: Safety management and interoperability thematic network for railway systems
IGSF : Ingénierie en Signalisation Ferroviaire	**SB** : mode Stand By
IHM : Interface Homme Machine	**SH** : mode Shunting
LS : mode Limited Supervision	**SNCF** : Société Nationale de Chemins de Fer
MMI : Men Machine Interface	**SN** : mode STM National
MoU: Memorandum of Understanding	**SOM**: Start Of Mission
ON : Organisme Notifié	**SR**: Staff Responsible
OS : mode On Sight	**SRS**: System Requirements specifications
OSTI : Organisme ou Service Technique Indépendant	**STI** : Spécifications Techniques pour l'Interopérabilité
RBC: Radio Block Center	
RFF : Réseau Ferré de France	

T	U
TIU : Train Interface Unit	**UIC** : Union Internationale de Chemins de fer
	UN : mode Unfitted
	UNIFE : Union des industriels fournisseurs ferroviaires
	UNISIG: Union Industry of Signaling

Lexique scientifique

A	C
API : Application Programming Interface	**CIM**: Computation Independent Model
ASP : Abstract Service Primitive	**CM** : Coordination Message
ATL: Atlas Transformation Language	**CP** : Coordination Point

E	G
EMF : Eclipse Modelling Framework	**GSPN**: Generalized Stochastic Petri Net

I / L	M
IDM: Ingénierie Dirigée par les Modèles	**MARTE**: Modelling and Analysis of Real-Time and Embedded Systems
IOLTS: Input Output Labelled Transition System	**MDA**: Model Driver Architecture
LGSPN: Labelled Generalized Stochastic Petri Net	**MOF**: Meta Object Facility
	MTC: Main Test Component

O / P	Q / R / S / T / U / X
OMG: Object Management Group	**QVT** : Query/View/Transformation
OOA: Analyse Orientée Objet	**RdP**: réseaux de Petri
PCOs: Points of Control and Observation	**RdPI**: réseaux de Petri interprétés
PCRT: Protocol Conformance Test Report	**SED** : Système à Evènements Discrets
PDU: Protocol Data Unit	**SPT**: Schedulability, Performance and Time
PICS: Protocol Implementation Conformance Statements	**TGV**: Test Generation with Verification technology
PIM: Plateform Independent Model	**TTCN**: Tree and Tabular Combined Notation
PIXIT: Protocol Implementation eXtra Information for Testing	**UML**: Unified Model Language
PMLP : Protocal Meta Langage Promela	**XML**: eXtensible Markup Language
PNO : Petri Net Object	
PSM: Plateform Specific Model	
PTC: Parallel Test Component	
PVS: Prototype Verification System	

INTRODUCTION GENERALE

Depuis la naissance des Chemins de fer, l'organisation du trafic, les règles et les principes qui les régissent ont toujours été une affaire nationale. En ce qui concerne la signalisation, les systèmes ont été validés au niveau national où chaque pays possède son propre « langage » et ses propres exigences pour gérer les trains sur son réseau. Aujourd'hui, plus de vingt systèmes de signalisation et de contrôle de vitesse existent en Europe. Ces systèmes sont incompatibles entre eux de telle sorte que les locomotives et les rames qui circulent sur plusieurs réseaux doivent transporter à leur bord les équipements requis par chacun de ces réseaux, ce qui augmente les coûts et diminue la performance du transport ferroviaire. En outre, la coexistence de nombreux systèmes de signalisation constitue une barrière au développement du trafic européen et complique la tâche des conducteurs qui doivent jongler entre les interfaces afin de pouvoir « lire » les signaux des différents réseaux en traversant les frontières. Toutes ces contraintes représentent un handicap pour l'intégration du transport ferroviaire à l'échelle européenne, alors que le transport routier profite de l'absence de telles barrières.

Devant cette segmentation du réseau européen et dans le but de promouvoir un marché ferroviaire européen de passagers et de fret, l'Union européenne a décidé de lancer une initiative dédiée à supprimer les entraves techniques créées historiquement par le développement des chemins de fer nationaux. Cette initiative s'est tout d'abord matérialisée par la mise en œuvre de directives européennes d'interopérabilité. Ces directives portent sur l'harmonisation technique et opérationnelle, dite interopérabilité, du réseau à grande vitesse et du réseau conventionnel (un aperçu de ces directives est décrit dans le chapitre suivant). Parallèlement, le système de signalisation européen ERTMS (European Rail Traffic Management System) a fait l'objet d'une fusion de plusieurs projets de recherches lancés par la commission européenne dans le but de créer une nouvelle génération de systèmes de signalisation et de contrôle de vitesse. Le principal objectif de ce projet est de disposer d'un système commun, harmonisé et standardisé pour la gestion du trafic ferroviaire et pour la signalisation en Europe en vue d'assurer l'interopérabilité entre les différents réseaux ferroviaires au sein de l'Europe. Les fonctions d'ERTMS sont implantées pour partie au sol, pour partie à bord des trains, et les moyens de communication entre sol et les trains sont normalisés. L'objectif est de déployer progressivement le système ERTMS afin de remplacer les systèmes nationaux par un système européen unifié. L'harmonisation sera un atout majeur pour améliorer la compétitivité du chemin de fer par rapport à la route.

L'INRETS (Institut national de recherche sur les transports et leur sécurité) est un établissement public de recherche qui a pour missions d'effectuer et d'évaluer toute recherche et tous développements technologiques consacrés à l'amélioration des systèmes et moyens de transports; de mener dans ces domaines tous travaux d'expertise et de conseil; de valoriser les résultats de ses recherches et travaux. Quant à l'unité de recherche ESTAS (Evaluation des Systèmes de Transports Automatisés et de leur Sécurité), sa problématique consiste à développer des outils et des méthodes avancés en matière de développement et d'évaluation des automatismes des systèmes de transport guidés et plus particulièrement ceux liés à la sécurité et ceux liés à l'amélioration de l'exploitation. L'INRETS a participé à plusieurs projets européens et expertises liés à ERTMS ou à l'interopérabilité : HEROE (Harmonisation des règles d'exploitation d'ERTMS), ACRUDA (certification des architectures matérielles et mise en place des règles de certification des calculateurs, embarqués ou fixes, développés en Europe), SAMNET (Safety management and interoperability thematic network for railway systems). Hormis ce cadre européen, l'INRETS contribue depuis sa création au GRRT (pôle d'animation de programmes de recherche scientifique et de développement technologique en transports terrestres, dont ferroviaire, impliquant les acteurs de la région Nord-Pas-de-Calais).

ESTAS est impliquée dans des questions relatives à ERTMS comme l'amélioration du processus de vérification de la conformité, la maîtrise des nouveaux risques liés au déploiement et aux phases transitoires (nouveaux scénarios d'accidents, analyses des bases de données d'incidents). Elle vise à accompagner l'introduction du système ERTMS au niveau de la diminution des coûts de validation et de certification issus de la mise en œuvre de ce nouveau système en Europe. La qualification du produit est en effet indispensable avant sa mise en service. Elle nécessite des campagnes de tests longues et coûteuses. ***Dans le contexte d'innovation des équipements pour le transport ferroviaire, ce projet a pour but de développer des méthodes, modèles et outils pour la génération de scénarios de tests et de valider les constituants***. Ces scénarios représentent des situations différentes d'exploitation et permettent d'étudier à la fois les aspects fonctionnels et dysfonctionnels du système. Ils doivent garantir la couverture complète des exigences fonctionnelles, normatives et sécuritaires. Pour cela, une première étape consiste à examiner l'existant (analyse fonctionnelle et de sécurité) de manière à dégager les scénarios pertinents, avec des critères de couverture, de performance et en particulier de conformité aux normes et réglementations en matière de sécurité ferroviaire. Au plan opérationnel, ces scénarios doivent pouvoir être mis en œuvre, soit « en ligne » par des tests sur site, soit « hors ligne » par des simulations. Pour des raisons évidentes de coûts et de délais, cette dernière approche est privilégiée lorsque cela est possible. Les essais de mise en œuvre sont effectués sur des simulateurs ferroviaires (simulateur ERTMS, simulateur de trafic).

Dans ce contexte régional à rayonnement national et européen, l'INRETS a mis en place une plate-forme de simulation du système ERTMS (cf. Annexe B) afin de développer ses activités de recherche en sécurité et interopérabilité couplées avec des activités d'expertises, d'études et de formations. L'objectif de cette plate-forme de simulation est de contribuer à la mise en œuvre d'une méthodologie d'essais de conformité du système ERTMS. Elle porte sur les points suivants :

- la formalisation de scénarios de tests,
- la définition de nouveaux scénarios de tests,
- la mise au point de procédures de tests et de validation de tests,
- la formalisation et la validation de règles d'exploitation,
- le contrôle de non-régression des performances (disponibilité, sécurité).

La plate-forme de simulation doit permettre d'analyser le fonctionnement d'un dispositif ERTMS dans différents contextes d'exploitation et à différents niveaux de détail.

Cette analyse préliminaire de la problématique industrielle, dans laquelle s'inscrit notre thématique de recherche, permet de dégager un certain nombre d'enjeux scientifiques et techniques :

(1) l'état d'art des méthodes de génération de scénarios de tests dans le contexte de la qualification d'équipements sécuritaires ;
(2) l'analyse des besoins à travers une approche d'ingénierie des exigences fonctionnelles, normatives et sécuritaires ;
(3) la définition d'une méthode de génération de scénarios de tests pour la validation des constituants indépendamment de l'origine de l'équipement ou de la plate-forme de test ;
(4) la conception d'une boîte à outils pour la simulation des scénarios ;
(5) la simulation sur une étude de cas générique d'ERTMS ;
(6) la simulation sur un cas spécifique ;
(7) la mise en œuvre des tests (physiques) sur site nécessaire afin de compléter les validations garantissant la couverture des exigences pour la certification.

Nos travaux de thèse concernent le développement de méthodologies formelles de génération de tests. Il s'agit d'une contribution majeure qui favorise l'innovation en matière d'équipements ferroviaires. Le simulateur ERTMS représente un atout majeur notamment pour valider notre approche.

ERTMS possède une architecture répartie et complexe. ***Notre motivation consiste à vérifier un constituant ERTMS par rapport à sa spécification***. La directive d'interopérabilité ferroviaire sur la grande vitesse [96/48/CE] définit les constituants d'interopérabilité ainsi :

« any elementary component, group of components, subassembly or complete assembly of equipment incorporated or intended to be incorporated into a Subsystem upon which the interoperability of the trans-European high-speed rail system depends either directly or indirectly ». Dans le tableau 1 ci-dessous, nous présentons un exemple de constituant d'interopérabilité : le sous-système ERTMS à bord. Dans la première colonne est mentionné l'identifiant du constituant. Dans la deuxième colonne est fourni le nom du constituant. Dans la troisième colonne sont décrites les interfaces et les fonctions du constituant. Dans la quatrième colonne sont spécifiées les spécifications obligatoires pour l'évaluation de la conformité de chaque fonction ou interface. Ces spécifications sont indiquées dans le chapitre 4 de la directive [96/48/CE]. Dans la cinquième colonne sont mentionnés les modules à appliquer pour l'évaluation de la conformité. Ces modules sont décrits dans le chapitre 6 de la directive.

Table 1: Un exemple d'un constituant d'interopérabilité: ERTMS ETCS [2006/860/EC]

1	2	3	4	5
n	**Interoperability Constituant IC**	**characteristics**	**Specific requirements to be assessed**	**Module**
1	ERTMS ETCS on board	Safety	4.2.1	
		On board ETCS functionnality	4.2.2	H2 Or B with D Or B with F
		Excluding: •Odometry •Data recording for regulatory purposes		
		ETCS and EIRENE air gap	4.2.5	
		Interfaces RBC (level 2 and 3) Radio in-fill unit (optional level 1) Eurobalise airgap Euroloop airgap (optional level 1)		
		Interfaces STM (implementation of interface K optional)	4.2.6.1	
		ERTMS GSM-R on-board	4.2.6.2	
		Odometry	4.2.6.3	
		Key management system	4.2.8	
		ETCS ID Management	4.2.9	
		ETCS Driver Machine Interface	4.2.13	
		Key Management	4.3.1.7	
		Physical environmental conditions	4.3.2.5	
		EMC	4.3.2.6	
		Data interface	4.3.2.8	
		Safety Information recorder	None	

Dans le contexte ERTMS, ***nous ignorons le comportement interne de chaque constituant***. Cependant, la spécification nous informe sur les interactions de chaque constituant avec l'environnement externe. Il s'agit de constituants réactifs. Ce terme a été introduit par Harel et Pnueli [Harel and Pnueli, 1985] pour designer des systèmes qui communiquent de manière permanente avec leur environnement à travers un échange continu d'évènements. Deux propriétés importantes caractérisent ces systèmes réactifs : la synchronisation des évènements. Ces systèmes sont nécessaires, de par leur importance, pour le traitement

d'applications exigeant des réponses immédiates, tant du côté interactions externes que du côté interactions internes du système. Beaucoup de systèmes dans le monde réel peuvent être considérés comme réactifs : les distributeurs automatiques, les installations industrielles, les protocoles de communication, les systèmes avioniques, les réacteurs nucléaires, etc. Ces systèmes sont composés de sous-systèmes qui peuvent être eux-mêmes réactifs. Il est alors essentiel de réaliser l'importance cruciale de ce type de systèmes et qu'une défaillance peut être extrêmement grave. Des techniques de vérification et de validation doivent être effectuées afin de s'assurer du bon fonctionnement de ces constituants.

Vérification & Validation

La vérification et la validation (V&V) forment un processus qui s'étale tout au long du cycle de vie d'un système. Le terme V&V englobe toutes les activités permettant de s'assurer que le logiciel corresponde bien à son cahier des charges [Bouali, 2009]. Concrètement, nous ne cherchons pas à faire la différence entre les activités liées à la vérification et celles liées à la validation étant donné qu'elles ont l'objectif commun d'atteindre un système final qui est fiable. Néanmoins, les distinctions suivantes peuvent être établies :

- ✓ La vérification s'assure que les propriétés du système sont bonnes. Elle répond donc à la question construisons-nous correctement le produit ? [Chapurlat, 2007] Par définition, la vérification est « *la confirmation par examen et apport de preuves tangibles (informations dont la véracité peut être démontrée, fondée sur des faits obtenus par observation, mesures, essais ou autres moyens) que les exigences spécifiées ont été satisfaites* » [ISO 8402].

- ✓ La validation s'assure que le système implémenté correspond aux attentes du futur utilisateur. Elle permet de démontrer donc que les exigences initiales ont bien été prises en compte [Bouali, 2009]. La validation répond donc à la question: construisons-nous le bon produit ? Par définition, la validation est « *la confirmation par examen et apport de preuves tangibles que les exigences particulières pour un usage spécifique prévu sont satisfaites. Plusieurs validations peuvent être effectuées s'il y a différents usages prévus* » [ISO 8402].

Afin de satisfaire les objectifs de la V&V, des techniques d'analyse et de vérification statiques et dynamiques sont utilisées. Concernant les techniques statiques, elles sont consacrées à l'analyse des différentes représentations du système, comme le cahier des charges, les modèles et documents de conception ou les programmes. Ces techniques peuvent être appliquées à tous les niveaux du processus de développement. Quant aux techniques dynamiques, elles exigent l'exécution du système à vérifier afin d'observer son comportement.

L'analyse

L'analyse est l'une des techniques de V&V qui permet de vérifier soit la cohérence des exigences du système soit la satisfaction des exigences par le système:

- ✓ Analyse de la cohérence des exigences : la majorité des travaux sur ce sujet proposent la traduction des exigences dans un langage formel ou de contraintes ayant une sémantique bien définie [Chapurlat, 2007].
- ✓ Analyse de la satisfaction des exigences : elle consiste en un ensemble d'analyses qui peuvent être réalisées sur un modèle afin de vérifier qu'il satisfasse certaines propriétés ou exigences. Ces analyses se basent souvent sur des techniques telles que la preuve formelle et le model checking. Ces analyses nécessitent souvent d'avoir le modèle à vérifier sous un formalisme qui dépend de la technique d'analyse à appliquer (système états transitions, langage formel tel que B ou Z, …). De la même façon les propriétés à vérifier doivent elles aussi être exprimées sous un formalisme adapté au modèle à vérifier [Bouali, 2009].

Le test

Dans le cadre de nos travaux, nous nous intéressons à la validation d'un constituant réactif en utilisant la technique de test de type boîte noire (dans la suite du document, nous utilisons les deux termes vérification et validation pour dire que nous voulons valider un constituant par rapport à sa spécification). A travers la génération automatique de scénarios de test, nous voulons prouver la conformité d'un constituant par rapport à sa spécification. Il ne s'agit pas en effet de vérifier les propriétés des modèles de la spécification mais plutôt de vérifier si le comportement du constituant ERTMS est bien conforme à ce qui a été défini dans la spécification. Une modélisation formelle de cette dernière s'impose afin de pouvoir générer des scénarios de tests de conformité à partir des modèles formels. L'exécution des scénarios de test sur la plate-forme de simulation nous permettra d'avoir les verdicts de test concernant la conformité ou non du constituant par rapport à sa spécification.

Le test et l'analyse

Le test et la vérification sont deux techniques qui permettent de valider les systèmes et d'accroître la confiance dans leur bon fonctionnement conformément à leurs spécifications. La vérification permet de prouver les propriétés d'un système informatique à travers l'application de méthodes formelles sur son modèle mathématique (cf. Fig.I). Elle permet de prouver la satisfaction de ces propriétés dans le modèle du système avec certitude. Quant au test (cf. Fig.1), il permet de tester une implémentation réelle et exécutable d'un système donné avec l'intention de trouver des erreurs. Le test ne peut jamais garantir qu'une

implémentation est sans erreurs. Finalement, le test et la vérification sont deux techniques complémentaires qui interviennent à des moments différents du cycle de vie du système.

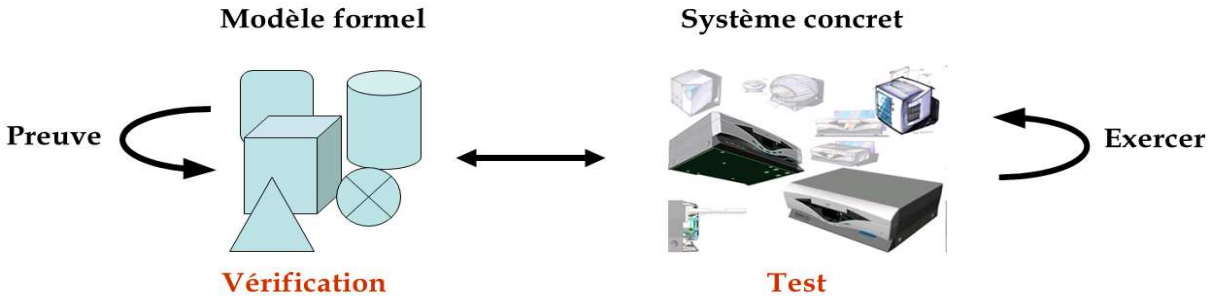


Figure 1: Le test & la vérification

Le test en tant que technique de validation des systèmes ou logiciels est considéré comme l'une des phases les plus importantes du cycle de vie d'un système. Il intervient afin de valider un logiciel ou un système avant sa mise en œuvre. Plusieurs méthodes et « approches » ont été développées pour les différentes techniques de test. Le choix de ces méthodes dépend principalement du type de l'application à tester. Selon [Myers, 1979], le test logiciel est un processus qui consiste à examiner un programme ou un système dans l'intention de trouver des erreurs. L'objectif est de s'assurer que les systèmes fonctionnent correctement au cours de leur utilisation. Ce processus ne peut pas garantir l'exactitude du système car il ne peut pas être complet pour tout système. En effet, contrairement à l'analyse formelle dans laquelle il est possible de générer tous les scénarios du comportement du modèle, le test est une technique dans laquelle on ne peut pas générer tous les scénarios possibles. Il est alors impossible dans le cas du test de couvrir l'ensemble des situations erronées.

E.W. Dijkstra [Dijkstra, 1970] a mentionné que le test ne peut pas être complet:

« Program testing can be used to show the presence of bugs but never to show their absence »

B.W. Boehm [Boehm, 1970] a confirmé que le test demande du temps et des efforts:

« The testing and checkout of production software often consumes upwards of 50 percent of the effort that goes into the development of the system »

Cependant, le test doit être effectué à chaque phase du cycle de développement du logiciel. La principale raison a été développée par B.W. Boehm [Boehm, 1970] :

« The cost of fixing an error increases by an order of magnitude at each stage in development »

De nos jours, les logiciels deviennent de plus en plus complexes, de nombreuses entreprises de développement de logiciels préfèrent consacrer toute une unité pour la génération de

scénarios de tests, plutôt que d'exiger des programmeurs le développement et le test en même temps. De plus, tester une spécification de manière manuelle peut être compliqué, cher et mener à des erreurs quand il s'agit d'une spécification de grande taille. Ce sont les arguments du développement croissant des outils et méthodes de génération automatique de scénarios de test fondés sur des bases théoriques solides. Un aperçu de ces différentes techniques est fourni dans le chapitre 2.

Formalisation de la spécification et Génération de scénarios de test

Un premier verrou scientifique à lever dans le cadre de notre démarche de génération de scénarios de test est la formalisation de la spécification. ERTMS possède en effet une spécification de grande taille et complexe décrivant le comportement dynamique des différents constituants ERTMS. Une première modélisation UML (Unified Modelling Language) orientée objet de cette spécification nous fournit des modèles semi-formels qui restent malheureusement insuffisants pour la phase de génération de test vu le manque d'un support mathématique solide et des outils d'étude correspondants. La formalisation de la spécification constitue ainsi une étape cruciale dans notre démarche. Les modèles semi-formels de la spécification doivent être transformés en modèles formels (cf. section 3.3). Les réseaux de Petri en tant que modèle formel et bien outillé correspondent à nos attentes.

Une fois le problème de la spécification informelle résolu, nous sommes confrontés à un autre verrou scientifique, celui de la vérification d'un constituant réactif. Comme nous l'avons décrit précédemment, l'architecture du système ERTMS est relativement complexe, répartie et composée de plusieurs constituants réactifs dont nous ignorons le comportement interne. Comment peut-on alors vérifier un constituant en tenant compte de son environnement d'interaction ? Est-il possible de tester son bon fonctionnement à travers les interactions qu'il a avec l'environnement extérieur ? Notre démarche (cf. section 4.1) consiste à générer automatiquement des tests de type boîte noire à partir de modèles formels en réseaux de Petri. Les scénarios de tests générés seront ensuite exécutés sur une plateforme de simulation afin de vérifier les constituants par rapport à leur spécification. Cependant, il nous reste à résoudre la problématique de la complétude des scénarios : comment garantir la couverture des scénarios de test et quel critère de couverture faut-il définir ?

Plan de la thèse

Ce manuscrit de thèse est composé de quatre chapitres. Les deux premiers chapitres décrivent le contexte industriel et l'état de l'art scientifique de nos travaux. Les deux derniers forment le noyau de cette thèse et décrivent nos contributions dans le cadre d'une démarche de génération automatique de scénarios de test pour la validation de constituants réactifs. Plus précisément:

- ✓ Dans le **premier chapitre,** nous détaillons le contexte industriel : la genèse du système ERTMS. Nous décrivons les directives d'interopérabilité qui nous ont menés vers ce nouveau système de signalisation et de contrôle commande dont le but est d'assurer l'interopérabilité ferroviaire au niveau européen. Nous présentons les composants de ce système, son architecture et ses spécifications. Ensuite, nous montrons les différentes phases de mise en service et de validation du système ERTMS en expliquant les différentes ambiguïtés quant au développement de ce dernier en Europe. A la fin de ce chapitre, une problématique scientifique est définie à partir de la problématique industrielle mentionnée auparavant.

- ✓ Le **deuxième chapitre** est consacré au positionnement scientifique de notre travail. En effet, nous présentons les différentes techniques de vérification de constituants réactifs en se basant sur les méthodes formelles et le test. Nous détaillons en particulier les principaux concepts utilisés dans les tests de conformité de type boîte noire. Ensuite, les principales approches de génération automatique de scénarios de test pour la vérification de composants sont définies. Suite à cet état d'art, les principaux choix sur lesquels se basent nos contributions sont déterminés. Une partie de ce chapitre est consacrée à l'étude du problème de couverture de scénarios de test.

- ✓ Le **troisième chapitre** propose une approche de transformation de modèles fondée sur l'utilisation combinée du langage UML (Unified Modelling Language) et des Réseaux de Petri interprétés afin de produire des modèles formels de la spécification. Le passage de modèles semi-formels vers des modèles formels se fait à travers une technique de transformation de modèles. Les avantages et les limites du formalisme source de cette transformation sont développés et le choix du formalisme cible est argumenté. Lors de cette démarche de transformation, une sous-classe des réseaux de Petri interprétés a été développée. Les règles de transformation entre les deux formalismes sont ensuite définies et illustrées à travers un exemple complet. Les spécifications présentées ainsi de manière formelle, peuvent servir à la phase de génération de scénarios de tests.

- ✓ Dans le **quatrième chapitre**, nous développons une méthode de génération automatique de scénarios de tests à partir de modèles formels décrits en réseaux de Petri interprétés. Un scénario de test généré à partir d'un réseau de Petri est considéré comme un chemin dans le graphe d'accessibilité correspondant. Ce dernier représente formellement le comportement du réseau. Cependant sa génération est souvent caractérisée par une explosion combinatoire. Les approches

permettant de résoudre cette problématique sont décrites. Ensuite, les différentes étapes de notre méthodologie de génération de séquences de franchissement à partir des réseaux interprétés sont détaillées. Le problème de couverture des scénarios de test est également abordé. A la fin du chapitre, un exemple d'application dans le cadre du système ERTMS/ETCS est présenté afin d'illustrer la méthodologie proposée.

Ce manuscrit s'achève par une partie intitulée « conclusions & perspectives » dans laquelle nous soulignons l'intérêt de combiner des approches de modélisation Objet, de transformation de modèles et de génération automatique de scénarios de test à partir de modèles formels. Nous rappelons les contributions réalisées concernant la formalisation de la spécification et la génération de séquences de franchissement dans le but de vérifier des constituants réactifs. Nous précisons également que le champ d'application de nos contributions, centré initialement sur vérification et la validation du système de contrôle commande et de signalisation ERTMS, peut être élargi au développement sûr de systèmes complexes répartis. Les différentes perspectives de nos travaux sont établies dans cette partie.

1 CONTEXTE ET PROBLEMATIQUE

Résumé:

L'ouverture à la concurrence du réseau ferroviaire européen est une des volontés de la Commission européenne. Celle-ci s'est concrétisée par des réformes tant organisationnelles que techniques du système ferroviaire européen afin de le rendre « interopérable» en permettant la libre circulation des trains sans rupture de charge entre états membres. À ces fins, les Spécifications Techniques pour l'Interopérabilité (STI) ont été produites afin d'harmoniser les différences telles que l'écartement des voies ou l'électrification. Parallèlement, dans les années 90, la commission européenne a sollicité la mise au point d'un système de contrôle commande et de signalisation ferroviaire commun à tous les réseaux des états membres. Ce système a été appelé «European Rail Trafic Management System» (ERTMS). ERTMS est un système réparti complexe caractérisé par des composants, des niveaux de fonctionnement et des spécifications. Le déploiement complet de ce nouveau système est long et coûteux, des évolutions sont nécessaires et soulèvent d'autres défis technologiques. L'objectif global est de diminuer les coûts de validation et de certification issus de la mise en œuvre de ce nouveau système en Europe tout en facilitant l'interopérabilité à travers la reconnaissance mutuelle de composants ERTMS entre états membres. La problématique scientifique réside dans la modélisation formelle de la spécification afin de permettre la génération automatique des scénarios de test. Les verrous scientifiques sont liés d'une part à la transformation de modèle semi-formel en modèle formel en préservant les propriétés structurelles et fonctionnelles du système, et d'autre part à la couverture des tests générés automatiquement.

Sommaire du chapitre 1

1.1 Introduction

Dans le contexte ferroviaire, le franchissement d'une frontière représente un évènement exceptionnel hormis pour quelques locomotives qui doivent être équipées de multiples systèmes. Un des obstacles techniques qui se dressent est celui de l'incompatibilité des systèmes de signalisation. Ces derniers ont été généralement développés au niveau national, par un industriel pour un client spécifique et ils diffèrent donc d'un pays à un autre. Dans les années 90, une mutation technique des systèmes ferroviaires européens a été engagée par la Commission Européenne. Divers projets de recherche ont été lancés dans les états membres afin de concevoir une nouvelle génération de systèmes de signalisation et de contrôle de vitesses plus performants et moins onéreux, tout en profitant des immenses progrès du secteur des télécommunications. Ces projets de recherche ont été ensuite fusionnés pour donner naissance à un grand projet industriel européen : ERTMS, acronyme de European Rail Traffic Management System. Ce système de contrôle commande et de signalisation a ainsi été conçu dans le but d'assurer l'interopérabilité au niveau européen. Il s'agit d'un système normalisé avec un niveau de sécurité supérieur à celui des systèmes existants [Castan, 2004].

ERTMS est décrit tout au long de ce chapitre. Tout d'abord, nous décrivons l'interopérabilité ferroviaire, les directives correspondantes, les spécifications techniques pour l'interopérabilité, la signalisation ferroviaire et l'exploitation. Ensuite, nous présentons les composants du système ERTMS, les trois niveaux de fonctionnement et les spécifications ERTMS. Nous nous intéressons dans la suite au deuxième niveau de fonctionnement dont l'architecture est décrite dans la section 2.4. Dans les sections qui suivent, nous détaillons les étapes de mise en exploitation du système ERTMS, les phases de vérification et de validation et les différentes contraintes quant au déploiement du système ERTMS en Europe. Enfin et avant de conclure, nous développons la problématique scientifique qui nous donnera un aperçu du chapitre suivant.

1.2 Interopérabilité, signalisation et exploitation

1.2.1 Interopérabilité ferroviaire

L'interopérabilité ferroviaire vise à créer un système ferroviaire européen capable de permettre la circulation sûre et sans rupture de charge des trains en accomplissant les performances requises pour les lignes [Jabri et *al*, 2007b]. Cette aptitude repose sur l'ensemble des conditions réglementaires, techniques et opérationnelles qui doivent être

remplies pour satisfaire aux exigences essentielles. Ainsi, le Parlement européen et la Commission européenne ont adopté en 1996 la directive 96/48/CE sur l'interopérabilité du système ferroviaire européen à grande vitesse, suivie en 2001 par la directive 2001/16/CE sur l'interopérabilité du système ferroviaire européen conventionnel. Les deux directives ont été ultérieurement modifiées par la directive 2004/50/CE.

D'un point de vue fonctionnel et technique, l'interopérabilité est caractérisée par quatre points majeurs [Jabri et *al*, 2007b] :

- Le train ne doit pas changer de motrice aux frontières.
- Le train ne doit pas s'arrêter aux frontières.
- Il ne doit pas y avoir de changement d'agent de conduite aux frontières.
- Le conducteur ne doit pas réaliser d'actions de conduite autres que les actions normalisées ERTMS.

La fragmentation technique des réseaux ferroviaires est un handicap majeur entravant le développement de ce mode de transport. Les directives d'interopérabilité mettent en avant des exigences essentielles qui couvrent l'ensemble des conditions à satisfaire pour assurer l'interopérabilité du réseau européen à grande vitesse ou conventionnel. La directive est considérée comme l'élément de base d'une architecture à trois niveaux : la directive proprement dite avec les exigences essentielles que le système doit respecter; les spécifications techniques d'interopérabilité (STI) qui doivent être adoptées dans le cadre établi par la directive; l'ensemble des autres spécifications européennes et notamment les normes européennes des organismes européens de normalisation: CEN (comité européen de normalisation), CENELEC (comité européen de normalisation électrotechnique) et ETSI (institut européen des normes de télécommunications).

Les deux directives d'interopérabilité ont formé le socle législatif européen engendrant la définition des STI. Ces dernières correspondent aux exigences essentielles concernant la sécurité, la fiabilité, la santé des personnes, la protection de l'environnement, la compatibilité technique et l'exploitation. Les États membres sont dans l'obligation de les respecter afin de réaliser les objectifs d'interopérabilité en Europe. Les premières spécifications techniques d'interopérabilité ont été adoptées en 2002 concernant les systèmes à grande vitesse. Ces STI liées aux sous-systèmes infrastructure, énergie, matériel roulant, système de contrôle-commande et de signalisation, maintenance et exploitation sont déjà entrées en vigueur depuis le 1er Décembre 2002. Elles précisent les éléments fondamentaux de chacun de ces sous-systèmes et identifient notamment les constituants qui ont un rôle critique du point de vue de l'interopérabilité.

Les STI concernant le contrôle-commande et la signalisation ont été adoptées le 28 Mars 2006 sous la forme de la directive 2006/679/CE. Depuis sa création en 2004, l'Agence

ferroviaire européenne est chargée de l'élaboration et de la révision des STI sur la base des travaux réalisés ou au moins entamés par l'AEIF (l'Association Européenne d'Interopérabilité Ferroviaire). Le Comité des États membres, établi conformément à l'article 21 des directives 96/48/CE et 2001/16/CE, a fourni à l'Agence ferroviaire européenne le mandat de rédaction du troisième groupe de STI pour les systèmes ferroviaires conventionnels [Lancien, 2004].

Figure 1.2: Directives et STI

Pour résumer, la commission européenne (cf. Fig.1.2) a pour objectif de créer un marché unique assurant la libre circulation des biens et des personnes. Ceci s'est concrétisé par la directive 91/440/CEE qui a séparé les missions des entreprises ferroviaires de celles des gestionnaires d'infrastructure (l'application de cette directive en France a donné naissance à Réseau Ferré de France (RFF), la Société Nationale de Chemins de Fer (SNCF) gardant elle son rôle d'exploitant de services de transports). Ensuite, les conditions d'interopérabilité du système ferroviaire transeuropéen à grande vitesse et conventionnel ont été établies dans les directives 96/48/CE et 2001/16/CE. Le système ERTMS a pour objectif d'apporter au niveau des systèmes de signalisation et de contrôle commande, une solution économique et technique à l'interopérabilité ferroviaire [Castan, 2004]. Il résulte de l'adoption de la directive 96/48/CE qui fait référence aux STI. Il correspond aux deux sous-systèmes suivants : le sous-système *contrôle commande et signalisation* et le sous-système *exploitation*.

Les principes de base de la signalisation ferroviaire et de l'exploitation ferroviaire seront décrits respectivement dans les sections 1.2.2 et 1.2.3.

1.2.2 Les principes de base actuels de la signalisation ferroviaire

L'objectif de la signalisation ferroviaire est d'assurer la sécurité des circulations des trains. En effet pour garantir un fonctionnement cohérent de l'ensemble du système ferroviaire, le dialogue entre le train (conduit par l'homme) et le sol (qui gère l'exploitation) doit être permanent. Ce dialogue est assuré grâce à un langage de communication appelé

signalisation dont l'homme est l'interface [Chapas, 2007]. Cette communication recouvre les informations suivantes : la position du train dans l'espace et le temps, autrement dit ses coordonnées par rapport au programme prévu ; et les ordres susceptibles d'adapter la circulation en fonction des situations dégradées qui peuvent survenir au cours de la marche.

Ceci représente un réel défi technique, car les distances de freinage des trains sont bien plus importantes que celles des voitures [Lacôte & Poré, 2004]. À 100 ou 160 km/h, cette distance est de l'ordre de quelques centaines de mètres. Mais, à très grande vitesse, elle est de quelques kilomètres! Tant sur lignes conventionnelles que sur lignes à grande vitesse, il est dès lors nécessaire que le conducteur du train reçoive longtemps à l'avance les informations nécessaires à la conduite. Jusqu'à 160 km/h, la vitesse n'est pas très élevée, le conducteur peut alors observer la signalisation latérale qui se trouve le long de la voie [DG TREN, 2006]. Au-delà de cette vitesse, le conducteur aura des difficultés à voir et interpréter correctement la signalisation latérale ce qui implique qu'une signalisation en cabine est nécessaire. La figure 1.3 décrit de manière simplifiée le processus mis en œuvre pour éviter qu'un train n'en rattrape un autre.

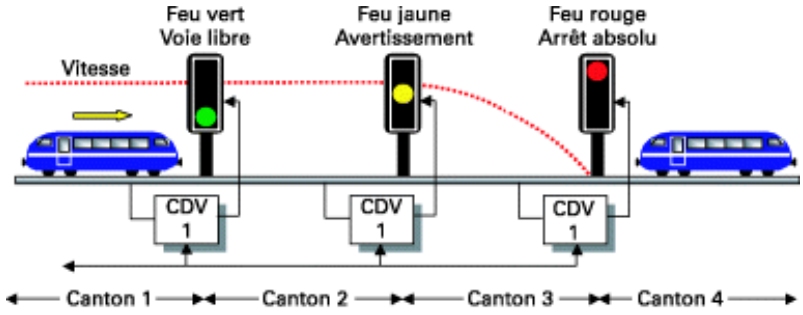


Figure 1.3: Signalisation ferroviaire [Chapas, 2007]

La voie est divisée en sections, appelées «cantons», de longueur variable en fonction des caractéristiques des trains l'empruntant. Tout canton ne peut être occupé que par une seule circulation. Un système de gestion des enclenchements détecte la présence des trains dans chaque canton. Chaque canton est protégé par un signal lumineux, garantissant l'espacement des trains. Si le signal est rouge, il indique au conducteur que le canton suivant est occupé. S'il est jaune, il indique que le prochain signal est rouge. Le conducteur doit donc contrôler sa vitesse afin de pouvoir s'arrêter avant le prochain signal si celui-ci est toujours rouge (cf. Fig.1.3). Autrefois, le canton était à la commande manuelle, grâce au téléphone (c'était le cantonnement téléphonique). Il est maintenant automatique : le système de gestion des enclenchements déclenche la protection des circulations en fonction de l'occupation de la voie via les circuits de voie (qui détectent la présence d'un train sur la voie) et les itinéraires en cours. Le circuit de voie permet la protection des circulations contre les risques de nez à nez, de rattrapage, de présence d'obstacle sur la voie et de rail cassé.

Le principe reste le même sur les lignes à grande vitesse, mais le nombre de cantons entre deux trains augmente en raison des distances de freinage qui deviennent plus longues. De plus, la vitesse importante ne permet pas au conducteur d'appréhender la signalisation latérale. Ainsi, un signal émis par la voie est capté par la locomotive ou la rame pour afficher au conducteur, la vitesse maximale autorisée à cet endroit. C'est ce qu'on appelle la «signalisation en cabine». Le conducteur est alors responsable de la sécurité et doit respecter les indications de la signalisation. De plus, les systèmes de signalisation diffèrent d'un pays à un autre étant donné qu'ils ont été généralement développés au niveau national, par un industriel pour un client spécifique. Ainsi la fréquence d'émission des signaux et la nature des informations transmises diffèrent. En Europe, plus de vingt systèmes de signalisation et de contrôle de vitesses existent (cf. Fig.1.4). Ils sont incompatibles entre eux. Ainsi, un train, qui relie plusieurs pays doit être équipé de systèmes différents ; ce qui signifie, entre autres, des capteurs et des écrans de contrôle spécifiques en cabine. La multiplication des systèmes de contrôle commande en cabine complique la tâche des conducteurs étant donné qu'ils doivent connaître ces systèmes et les procédures associées.

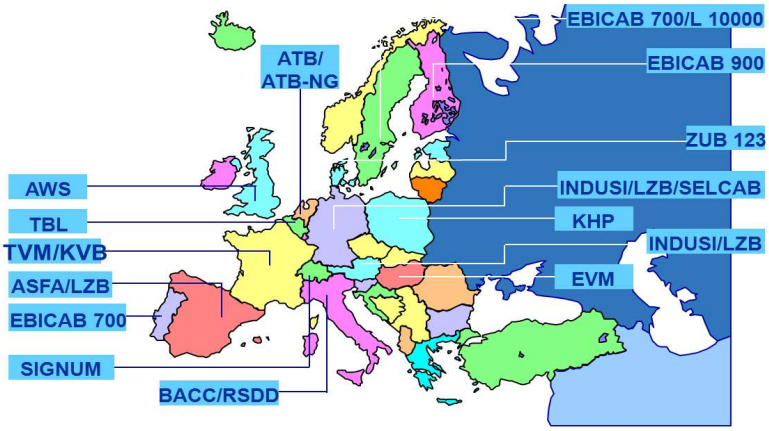


Figure 1.4: Systèmes de signalisation en Europe (copyright UNIFE[1])

Devant le constat d'une telle fragmentation du secteur, la nécessité de grouper les efforts et de travailler au niveau européen s'est progressivement imposé. L'objectif était d'éviter que, dans chaque État membre, des fonds importants ne soient investis dans le développement, les essais et la validation de systèmes incompatibles mais répondant à des besoins similaires.

1.2.3 L'exploitation ferroviaire

La circulation des trains relève de la composante « exploitation » du système ferroviaire. Elle est soumise à des règles et procédures afin de satisfaire le client dans un environnement totalement sécurisé. En effet, dès les origines du chemin de fer, la sécurité de la circulation des trains fut la préoccupation majeure. Cette sécurité s'est renforcée au fur et à mesure de

[1] www.unife.org

l'accroissement des performances en termes de charges remorquées et de vitesses pratiquées. Il est d'ailleurs évident que l'énergie cinétique acquise lors du déplacement d'un lourd convoi met en jeu le paramètre fondamental qui est sa distance d'arrêt [Chapas, 2007]. De plus, dans la plupart des cas, la configuration du réseau est complexe et comporte des itinéraires sécants ou parcourus dans les deux sens de circulation. Tous ces paramètres exigent non seulement une programmation très précise mais une surveillance permanente du trafic, apte à faire face aux aléas et aux situations dégradées pouvant survenir à tout moment.

L'exploitation ferroviaire désigne le gestionnaire d'infrastructure qui se charge de gérer le trafic. Elle a comme objectifs de satisfaire les clients en terme de réponse à la demande de transport, d'assurer la sécurité de la circulation des trains, d'offrir une qualité de prestation au niveau des horaires et finalement d'optimiser le coût contribuant ainsi à la rentabilité du système ferroviaire. Elle dispose de trois moyens : le réseau, le système de sécurité et les entités de régulation.

Le réseau

Le réseau représente l'ensemble des lignes reliant les points de destination à desservir. Au niveau du réseau, une liaison peut être réalisée par un ou plusieurs « itinéraires » (cf. Fig.1.5). L'itinéraire peut être défini comme un chemin parcouru avec un sens de parcours.

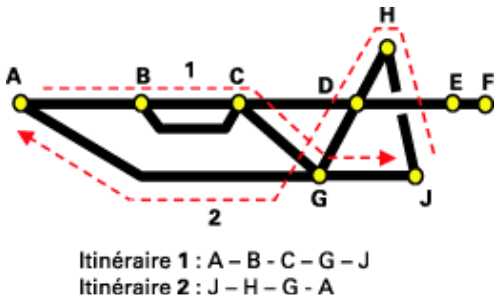


Figure 1.5: Réseau ferroviaire et itinéraires [Chapas, 2007]

L'architecture du réseau ferroviaire se base sur ces différentes configurations :

- La voie unique parcourue dans les deux sens de circulation (cf. Fig.1.6a) ;
- La double voie où chaque voie est dédiée à un seul sens (cf. Fig.1.6b) ;
- La double voie « banalisée » où chaque voie est parcourue dans les deux sens de circulation (cf. fig.1.6c) ;
- La communication qui permet la liaison entre deux voies (cf. Fig.1.6d) ;
- La bifurcation qui permet la création d'un itinéraire (cf. Fig.1.6e) ;
- L'évitement qui consiste en un segment de double voie permettant le croisement de deux circulations (cf. fig.1.6f) ;
- Le tiroir qui ressemble à une voie d'évitement (cf. Fig.1.6g).

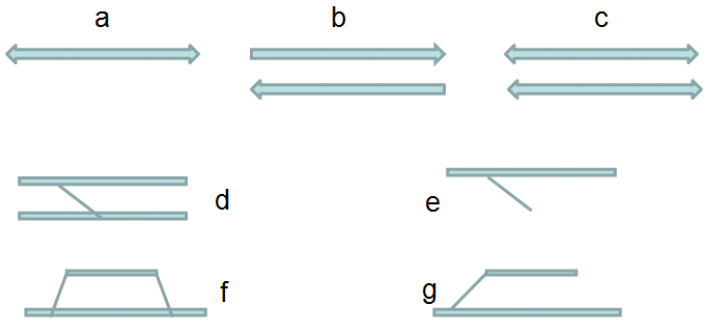


Figure 1.6: Les différentes configurations du réseau

Les systèmes de sécurités associés

Ces systèmes comprennent la signalisation étudiée et la règlementation. Cette dernière explicite les dispositions prises par chacun des acteurs : aiguilleurs, agents de sécurité des terminaux et gares, régulateurs, agents de conduite, procédures d'exploitation, etc.

Les entités de régulation

Afin d'assurer la régulation du trafic, un poste de commandement est mis en œuvre. Il est en liaison avec les établissements gérant le matériel et le personnel. Ce poste coordonne et dispose des aiguilleurs qui sont chargés de l'établissement des itinéraires ; du régulateur « circulations » qui est chargé de leur suivi en temps réel et du régulateur « sous-station » dont la mission est de gérer l'alimentation en énergie électrique d'une ou plusieurs lignes.

Suite à la présentation des notions d'interopérabilité, de signalisation et d'exploitation qui constituent les concepts clés du système ERTMS, nous consacrons la partie suivante à la description des différentes caractéristiques de ce système : les composants, les niveaux de fonctionnement et les spécifications.

1.3 Système ERTMS

Le franchissement d'une frontière européenne présente plusieurs obstacles techniques. Le plus connu est celui de l'écartement des rails. D'autres barrières techniques moins visibles doivent être levées : différents types de courant électrique, hauteur des quais pour les voyageurs,…etc. Concernant la signalisation ferroviaire et le contrôle de la vitesse des trains, les problèmes ont été, en général, résolus pour chaque réseau ferroviaire spécifique par un unique industriel. D'où la fragmentation qui caractérise actuellement le réseau ferroviaire européen. Ce manque de standardisation affecte le trafic international et entraîne des surcoûts importants [DG TREN, 2006]. Le système ERTMS vise à remédier à cette fragmentation en standardisant les multiples systèmes de signalisation qui coexistent en Europe. Ceci devrait procurer plusieurs avantages: accroître la compétitivité, dynamiser le

secteur ferroviaire, favoriser l'intégration des marchés des services ferroviaires de fret et de voyageurs, stimuler le marché européen des équipements ferroviaires, diminuer les coûts et augmenter la qualité du transport ferroviaire [El Koursi et al, 2002]. ERTMS est devenu alors un élément très important qui contribue à l'interopérabilité du système ferroviaire européen, tant à grande vitesse que conventionnel.

Les composants du système ERTMS

Aujourd'hui, ERTMS est constitué essentiellement des deux composants de base et d'un troisième composant en cours d'élaboration [Winter, 2009a] :

- **L'ETCS** (European Train Control System) est le système de contrôle/commande des trains, qui permet non seulement de transmettre au conducteur les consignes de vitesse et les autorisations de mouvement, mais aussi de contrôler en permanence le respect de ces indications. Un ordinateur embarqué compare en effet la vitesse du train avec la vitesse maximale permise et freine automatiquement en cas de dépassement.
- **Le GSM-R** (GSM for Railways) est le système radio utilisé pour échanger des informations entre le sol et le bord. Il est fondé sur le standard GSM de téléphonie mobile mais utilise des fréquences différentes propres au ferroviaire ainsi que certaines fonctions avancées. Il permet au conducteur de dialoguer avec les centres de régulation et peut être utilisé pour transmettre au train la vitesse maximale permise.
- **L'ETML** (European Traffic Management Layer) est le système de gestion de trafic en cours d'élaboration. Il identifie les facilités fonctionnelles de la gestion du trafic ferroviaire européen.

1.3.1 Les niveaux de fonctionnement du système ERTMS

ERTMS est destiné à remplacer les nombreux systèmes de signalisation ferroviaire existants actuellement en Europe. Son déploiement en tant que système de signalisation unique permettra à terme de réduire les coûts d'installation et de maintenance tout en améliorant les performances du système ferroviaire [Jabri et *al*, 2007c]. ERTMS répond à des besoins fonctionnels très variés, ce qui a conduit à imaginer trois niveaux.

1.3.1.1 ERTMS niveau 1

Le niveau 1 est destiné à compléter des systèmes de ***signalisation latérale*** en rajoutant un dispositif unifié de communication ponctuelle sol-train. La signalisation latérale utilisée à ce niveau ne fait l'objet d'aucune spécification ERTMS hormis la détection de présence des trains réalisée au sol par les circuits de voie. L'unification de la transmission ponctuelle

concerne à la fois le support physique de communication et le contenu des messages échangés qui est ainsi compréhensible par tous les équipements de bord ERTMS quelle que soit leur origine. Les industriels européens de la signalisation ferroviaire (UNISIG) ont réalisé le système de transmission **Eurobalise** répondant à ces exigences d'ERTMS [Castan, 2004].

Ce système de transmission Eurobalise (cf. Fig.1.7) est constitué des éléments suivants :

- Un **codeur** en interface entre les installations de signalisation latérale et la balise ;
- Une **balise** en voie qui possède deux interfaces pour les échanges sol-bord et les échanges de la balise vers le codeur ;
- Une **antenne** et un boîtier de réception embarqués, connus sous le nom de **BTM** (Balise Transmission Module).

L'Eurocab (ou EVC, European Computer Vital**)** est le calculateur de bord qui intègre l'antenne et le module de transmission Eurobalise. Il est en interface avec le conducteur à travers l'interface Homme-Machine désignée MMI (Men Machine Interface). Il est également en interface avec le train en cas de freinage d'urgence par l'intermédiaire du TIU (Train Interface Unit). Enfin il utilise les informations fournies par l'odomètre pour contrôler la vitesse.

L'odomètre permet de déterminer la position du train en utilisant les informations suivantes : les informations de recalage obtenues par lecture des Eurobalises rencontrées (références géographiques absolues) et les informations de déplacement fournies par les capteurs de mesure du chemin parcouru. Ces éléments de base qu'on vient de décrire se trouvent à bord et sont identiques pour les trois niveaux d'ERTMS.

Pour conclure, au niveau 1, le sol transmet au train des informations qui lui permettent de calculer en permanence sa vitesse maximale autorisée. Ces informations sont transmises par les Eurobalises placées le long de la voie et connectées à la signalisation latérale [DGTREN, 2006]. Le niveau 1 est particulièrement caractérisé par le cantonnement fixe (voie divisée en cantons fixes) et la détection et vérification de l'intégrité des trains par le sol.

1.3.1.2 ERTMS niveau 2

L'architecture est basée sur un transfert d'informations utilisant un support de communication radio continu : le **GSM-R**. Les informations qui transitaient en niveau1 par les balises sont transmises par le **RBC** (Radio Block Center) via le GSM-R (cf. Fig.1.7). Les Eurobalises sont utilisées principalement pour les besoins de recalage de l'odométrie. Les RBC sont installés en interface avec le système d'enclenchement. Ils regroupent les données de signalisation qu'ils traduisent en autorisations de mouvement et les transmettent au train via GSM-R. Le GSM-R est un dérivé de la norme de téléphonie mobile GSM et utilise les mêmes solutions techniques avec des bandes de fréquences réservées. Afin de satisfaire les

besoins ferroviaires en matière de transmission sol-train, les appels de groupe et diffusion, l'établissement rapide de liaisons, la gestion des priorités et la transmission des données pour la signalisation ferroviaire ont été ajoutés [Castan, 2004].

Le niveau 2 est également basé sur une détection des trains par le sol et cantonnement fixe. Par rapport au niveau 1, les communications sol-train se font par GSM-R, d'où le rajout du matériel GSM-R à l'équipement à bord.

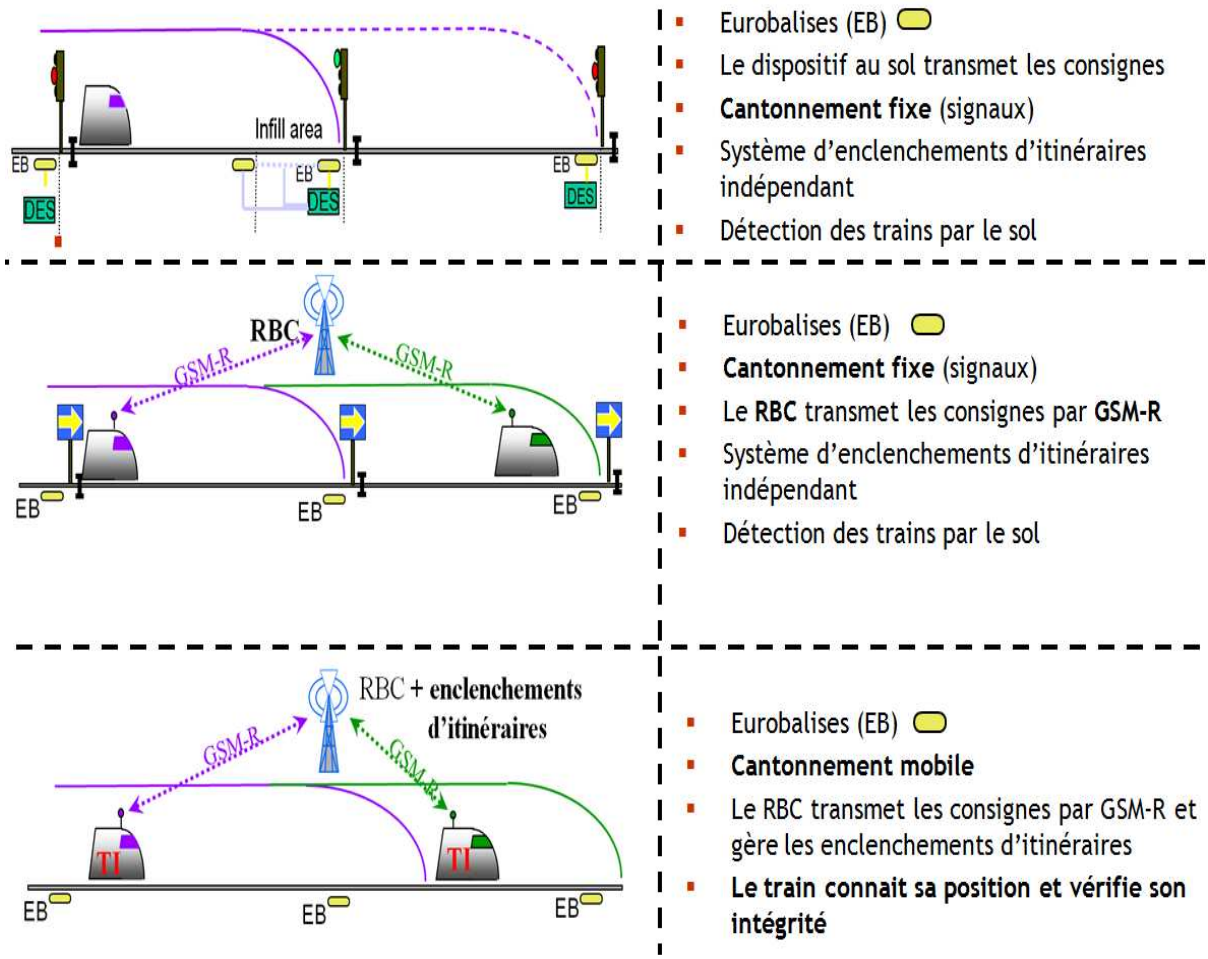


Figure 1.7: Comparaison des niveaux 1, 2 et 3

1.3.1.3 ERTMS niveau 3

Le niveau 3 est destiné à être implanté seul au sol, sans superposition à un autre système de signalisation afin d'obtenir les performances maximales de débit de ligne. Dans une phase transitoire, il peut néanmoins être installé en superposition avec la signalisation existante. Comme pour le niveau 2, les centres de traitement au sol (RBC) utilisent la transmission continue sol train par radio (GSM-R). Mais au niveau 3, les circuits de voie ne sont plus utilisés pour la détection de la présence des trains, et les RBC attribuent des cantons mobiles en utilisant la localisation fournie par les trains eux-mêmes pour connaître l'occupation des voies (cf. Fig.1.7). Ainsi, par rapport au niveau 2, l'équipement de bord est complété par un

dispositif de contrôle d'intégrité du train qui garantit la longueur du train et la stabilité de celle-ci avec le niveau de sécurité requis pour remplacer les circuits de voie [Castan, 2004].Le concept de cantons mobiles considère l'arrière de la circulation précédente comme le point à protéger. L'espacement entre deux trains correspond à tout instant à la distance de freinage à la vitesse considérée, plus une distance de sécurité. Cette zone correspond à un canton qui se déplace avec le train.

Dans le cadre de nos travaux de thèse, ***nous nous intéressons au deuxième niveau de fonctionnement*** dont l'architecture est décrite dans la section suivante.

1.3.1.4 Architecture du système ERTMS au niveau 2

Au niveau 2, l'architecture est basée sur un transfert d'informations utilisant un support de communication radio continu : le GSM-R. ETCS est divisé en deux parties : une sur la voie, et l'autre à bord des trains [Jabri et *al*, 2008b]. Ces deux sous-systèmes sont détaillés dans les paragraphes suivants.

Le sous-système à bord

Afin de pouvoir démarrer, le train doit recevoir une autorisation de mouvement correspondant à la distance de voie réservée à celui-ci. Au fur et à mesure que le train avance, cette autorisation de mouvement est mise à jour [Jabri et *al*, 2008b]. En se basant sur ce concept, l'EVC (European Vital Computer) construit différentes courbes dont la courbe de vitesse maximale autorisée, la courbe d'alerte, la courbe d'intervention du service de freinage et la courbe d'intervention de freinage d'urgence. Ainsi, il peut contrôler la vitesse par rapport à ces courbes en tenant compte de la position du train, sa vitesse et sa capacité de freinage. Dans le cas où le conducteur dépasserait l'une de ces courbes de freinage en approchant une zone à vitesse plus basse, l'EVC interviendra pour alerter le conducteur ou demander un freinage à la motrice. En effet, l'EVC (cf. Fig.1.8) est connecté à plusieurs interfaces, dont les systèmes de la motrice, l'IHM (Interface Homme Machine) qui permet la communication avec le conducteur [Tamarit & Guido, 2004], le BTM (Balise Transmission Module) qui reçoit les informations des balises placées sur la voie et l'antenne du GSM-R qui échange les messages avec le RBC (Radio Block Center) [ERTMS, 2008].

Le sous-système au sol

Au niveau 2, le sous-système au sol est composé du RBC et de balises. L'ensemble des balises constitue les références géographiques. Le RBC [Tamarit & Guido, 2004] est le système qui gère et fournit toutes les autorisations de mouvements des trains en assurant la sécurité des circulations. Il s'interface avec le système d'enclenchement et utilise des données de sécurité provenant de la voie, comme les positions du train pour envoyer les messages à l'équipement

Euroradio. Ce dernier transmet à son tour, par radio, les autorisations de mouvement à tous les trains qui se trouvent sur la voie [Jabri et *al*, 2008b]. Les fonctions du système RBC sont entièrement dupliquées afin de pouvoir déterminer à tout instant, les autorisations de mouvements des trains sous la responsabilité du RBC [ERTMS, 2008].

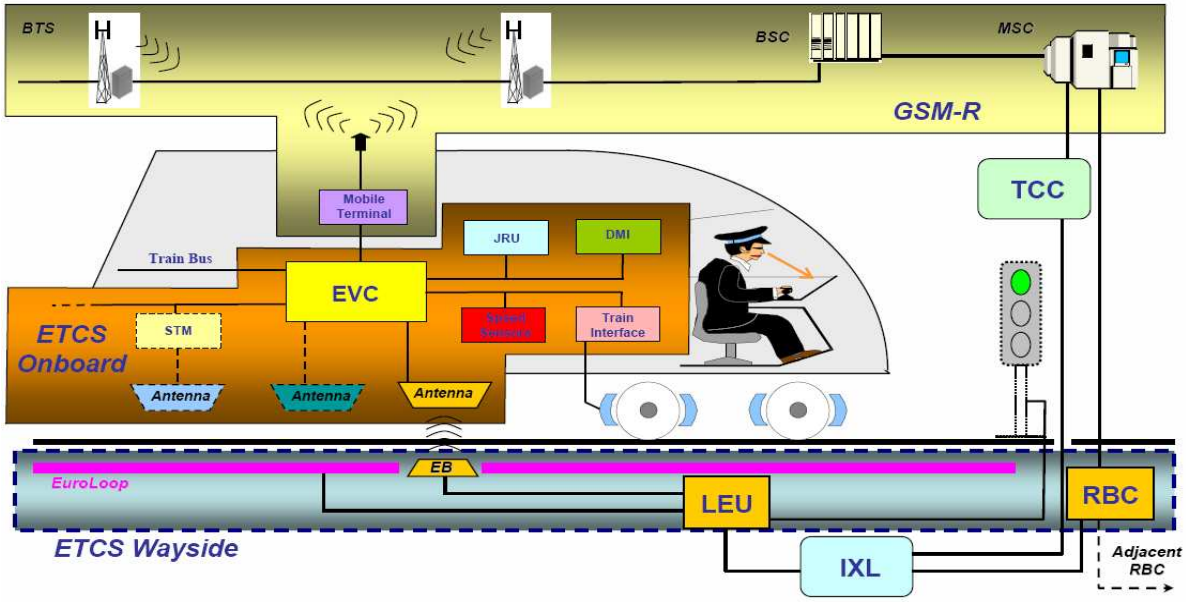


Figure 1.8: Architecture ETCS niveau 2 [Lévêque, 2007]

1.3.2 Les spécifications du système ERTMS

Le « Groupe des Utilisateurs ERTMS » a été constitué afin de définir les règles d'interopérabilité opérationnelle, allant jusqu'à la rédaction d'un règlement d'exploitation, tandis qu'UNISIG[2] (Union Industry of Signaling) avait la charge d'assurer l'interopérabilité sur le plan technique en vue de permettre qu'un équipement bord fourni par un industriel A puisse fonctionner sur une infrastructure dont les équipements sont fournis par un industriel B. Cette division de tâches s'est traduite par différents niveaux de spécifications : la spécification fonctionnelle des besoins (FRS) a été rédigée par l'UIC[3] (Union Internationale de Chemins de fer), la spécification système (SRS) et les autres documents techniques de niveau inférieur étant à la charge d'UNISIG.

UNISIG a été créé en 1999 à la demande de la commission européenne pour la rédaction des spécifications techniques du système ERTMS/ETCS. UNISIG est un membre associé de l'UNIFE[4] (Association Européenne de l'industrie ferroviaire). Aujourd'hui, il a le rôle de développer, maintenir et mettre à jour les SRS en coopération étroite avec l'ERA[5] (European Railway Agency). Les membres de l'UNISIG sont : Alstom, Ansaldo STS, Bombardier, Invensys, Siemens et Thales.

[2] www.ertms.com/2007v2/factcheets/ertmsFactsheet-UNISIG.pdf
[3] www.uic.org
[4] www.unife.org
[5] www.era.europa.eu

Les « Functional Requirements Specifications » (FRS) et les « System Requirements Specifications » (SRS) constituent les spécifications du système ERTMS. Elles sont présentées sous un format textuel [Jabri et *al*, 2008a]. Les FRS[6] [UIC, 2007] identifient les fonctions de haut niveau du système ERTMS telles que les fonctions opérationnelles, les fonctions d'infrastructure, les fonctions de bord, les fonctions de protection, ... Les SRS[6] [UNISIG, 2008] reprennent les fonctions obligatoires des FRS et les décrivent de manière détaillée (cf. Fig.1.9 de l'exemple SRS suivant). En effet, elles définissent les fonctions, leurs comportements et leurs contraintes, ainsi que les interfaces entre les différents sous-systèmes ERTMS.

ERTMS est un système qui permet une exploitation de nouvelles opportunités commerciales, des améliorations opérationnelles et une rationalisation de l'efficacité. Ainsi, la capacité de l'évolution de ses spécifications devrait être intégrée. Il ne doit pas en effet devenir une contrainte ou un obstacle. D'autre part, le développement du trafic international exige un service continu de bout en bout, un objectif que l'interopérabilité ferroviaire permet de remplir. La nécessité d'assurer l'interopérabilité, combinée à des cycles d'évolutions dans la signalisation ferroviaire constituent une forte contrainte dans le contexte du système ERTMS. En effet, l'ERA est chargée d'établir la transparence du processus pour gérer les changements de système et pouvoir assurer la réutilisation des spécifications.

Exemple SRS : Sélection du mode « Shunting ».

Dans la figure 1.9, nous présentons un exemple d'une procédure SRS « Entrée dans le mode de fonctionnement SH (Shunting)». Cette procédure de fonctionnement décrit la sélection du mode SH par le conducteur à partir des modes suivants (cf. annexe A « Modes d'exploitation et transitions ») : FS (Full Supervision), LS (Limited Supervision), OS (On Sight), SR (Staff Responsible), SB (Stand By), PT (Post Trip), SN (STM National) et UN (Unfitted). Cette spécification, modélisée sous la forme d'un graphe d'états dans le chapitre 5 des SRS [UNSIG, 2008], est également décrite dans le tableau 1.2 d'une manière textuelle afin de mieux expliquer le graphe.

[6] www.era.europa.eu/Core-Activities/ERTMS/Pages/ERTMSCurrentBalises.aspx

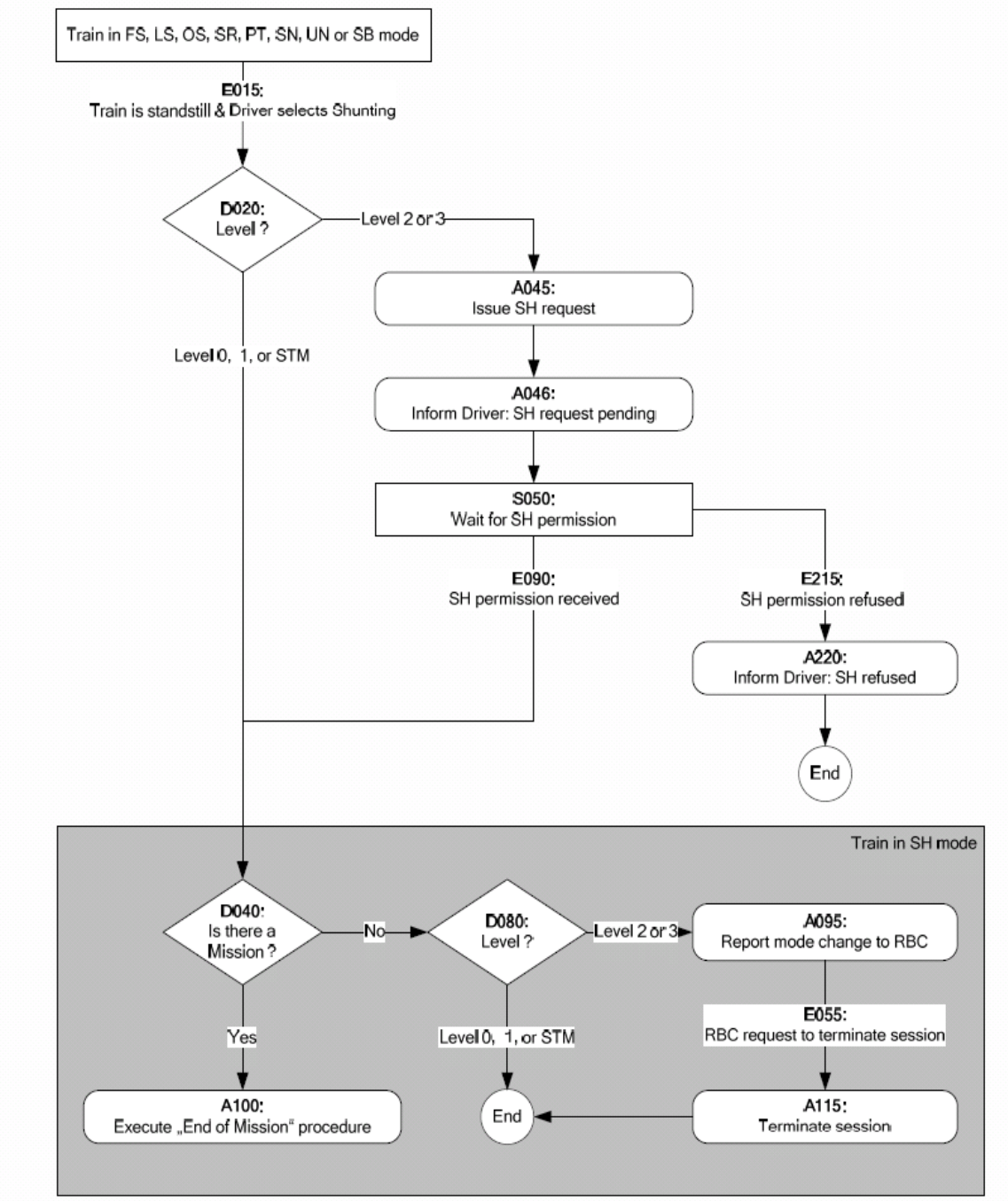


Figure 1.9: Diagramme d'états "Entrée dans le mode SH" (Chapitre 5 des SRS [UNISIG, 2008])

Table 1.2: Description "Entrée dans le mode SH" (Chapitre 5 des SRS, [UNISIG, 2008])

#	Description	Level
E015	The driver selects "Shunting" while the train is at standstill and the ERTMS/ETCS on-board equipment is in one of the following modes: - FS - LS - OS - SR - SN - UN - PT - SB (if necessary preconditions are fulfilled: Driver ID known, Level known, ... see procedure "Start of mission".).	All
D020	What is the current ETCS Level of operation?	All
A045	The ERTMS/ETCS on-board equipment sends the "Request for Shunting" message to the RBC together with a position report (with special value "position unknown" if the position is not known)	2,3
A046	The on-board equipment indicates on the driver MMI, that SH permission request to the RBC in pending	2,3
S050	The ERTMS/ETCS on-board equipment awaits SH permission.	2,3
E090	The ERTMS/ETCS on-board equipment receives the SH permission from the RBC.	2,3
D040	Is there an on-going mission?	All
A100	If there is an on-going mission, the "End of Mission" procedure is executed	All
D080	What is the current ETCS Level of operation?	All
A095	The mode change shall be reported to the RBC.	2,3
E055	An order to terminate the communication session is received from RBC	2,3
A115	On-board equipment terminates the communication session	2,3
E215	The ERTMS/ETCS on-board equipment receives the information from the RBC "SH refused".	2,3
A220	An indication shall be given on the driver MMI, that SH permission was refused by the RBC.	2,3

Le système ERTMS est un système de contrôle commande dont la mise en service nécessite une demande d'autorisation auprès des organismes concernés. Les différentes étapes de mise en service de ce système sont décrites dans la section suivante. Ce système nécessite également des phases de déploiement afin de le faire cohabiter avec les anciens systèmes. Le déploiement se fait à travers des étapes de vérifications que nous décrivons également dans la section suivante.

1.3.3 La mise en œuvre du système ERTMS

1.3.3.1 Les étapes de mise en service du système ERTMS

La mise en service du système ferroviaire nécessite une demande d'autorisation auprès des organismes concernés. En France, cette autorisation d'exploitation exige l'application du processus défini dans le décret 2000-286, à savoir : la rédaction du Dossier de Définition (DD) par le promoteur, l'élaboration d'un Dossier Préliminaire de Sécurité (DPS) par la SNCF qui contient l'analyse préliminaire des risques et enfin l'élaboration du Dossier de Sécurité (DS) par la SNCF [Berieau, 2004].

Le projet ERTMS, étant un projet européen, doit également respecter le décret 2001-129 de transposition de la Directive Européenne : Interopérabilité à grande vitesse- CE 96/48. A partir de cette directive et des STI qui l'accompagnent, il découle que :

- Les constituants d'interopérabilité du sous-système de contrôle commande et de signalisation doivent être accompagnés du certificat de conformité délivré par le constructeur. Ce certificat est établit par l'Organisme Notifié choisi (ON).
- Le sous-système considéré doit avoir une déclaration CE de vérification attribuée également par un organisme notifié choisi.

Selon la loi 2002-3 relative à la «sécurité des infrastructures de transport et aux enquêtes administratives en cas d'accidents », il est indispensable que plus d'un EOQA (Expert ou Organisme Qualifié Agrée) doive intervenir pour donner un avis sur le DPS. En France, CERTIFER (organisme de certification de produits et de systèmes pour les transports guidés terrestres) a été désigné dans le contexte ERTMS afin d'effectuer les missions d'ON et d'OSTI (Organisme ou Service Technique Indépendant) [Ozello, 2004]. L'OSTI est choisi par le promoteur afin d'évaluer la conception et la réalisation d'un nouveau système. Ainsi, pour la mise en service du système ERTMS, tous les éléments : DS, certificats de conformité, déclaration CE de vérification rapport d'EOQA sont nécessaires. Ensuite, l'ensemble sol ERTMS doit être installé sur une infrastructure et l'ensemble bord ERTMS intégré dans un train afin de pouvoir faire fonctionner le sous-système contrôle-commande et signalisation (C/C et Sig).

La première LGV Est européenne sera la première ligne équipée du système ERTMS en France. Afin d'autoriser la mise en service de cette ligne, quatre DS sont au moins nécessaires (cf. Fig.1.10): le DS ensemble sol ; le DS de la ligne sans C/C et Sig qui fait référence au DS ensemble Sol ERTMS ; le DS ensemble bord ERTMS et le DS matériel roulant qui fait référence au DS ensemble Bord ERTMS.

A tous ces dossiers, on rajoute un DS d'intégration de l'ensemble ferroviaire (bord-sol) qui consiste en un plan de démarche de sécurité (cf. Fig.1.10).

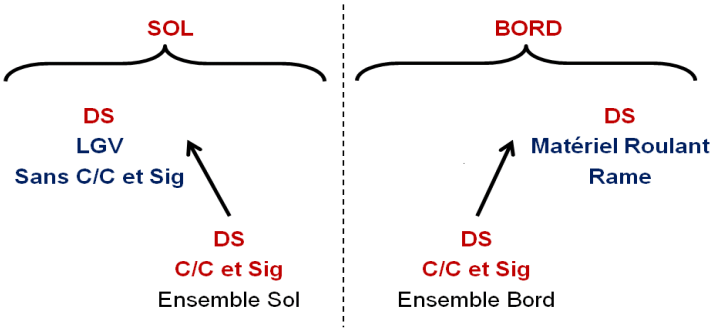


Figure 1.10: Plan de démarche de sécurité [Berieau, 2004].

Tout ce processus compliqué de mise en service du système ERTMS est maîtrisé grâce à l'application de la norme EN 50126 qui a été adoptée par le CENELEC (Comité Européen de Normalisation Electrotechnique) le 01/10/1998. En effet, cette norme explicite la façon d'exprimer les exigences en termes de FDMS (Fiabilité, Disponibilité, Maintenabilité, et Sécurité) ainsi que le détail des phases du cycle FDMS à respecter. Elle définit également ce qu'il faut fournir afin de prouver l'atteinte du niveau de sécurité requis.

EN 50126, section 4.2.2 : « *La FDMS est une caractéristique de l'exploitation d'un système sur une longue durée et elle est assurée par des concepts d'ingénierie, des méthodes, des outils et des techniques reconnus et appliquées tout au long du cycle de vie du système. La FDMS d'un système peut être caractérisée par l'aptitude, évaluée en termes qualitatif et quantitatif, du système, de ses sous-composants ou de ses composants, à fonctionner conformément à des spécifications et à être à la fois disponibles et sûrs* ».

EN 50126, section 4.3.2 : Dans le domaine ferroviaire, « *la sécurité et la disponibilité sont interdépendantes dans le sens où une insuffisance de l'une ou de l'autre ou une mauvaise gestion des conflits entre leurs exigences peut s'opposer à l'obtention d'un système au fonctionnement sûr. Ensuite, les objectifs de sécurité et de disponibilité d'un système en fonctionnement ne peuvent être atteints qu'en satisfaisant aux exigences de fiabilité et de maintenabilité et en contrôlant dans la durée de manière permanente les activités de maintenance et d'exploitation ainsi que l'environnement du système* ». Ce qui explique l'interdépendance des composantes de la FDMS (cf. Fig.1.11).

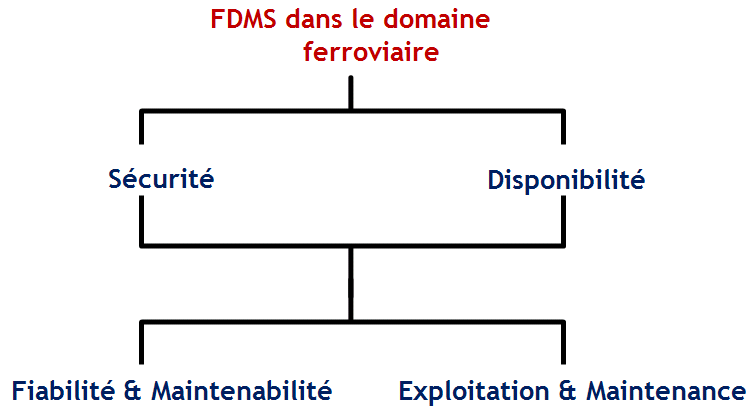


Figure 1.11: Interdépendance des composantes de la FDMS dans le monde ferroviaire [EN 50126]

Dans la figure 1.12 décrire ci-dessous, on présente le cycle FDMS qui respecte d'une part le décret 2000-286, le décret 2001-129 de transposition de la Directive 96/48, d'autre part les normes CENELEC. La partie basse de la figure concerne la réalisation des constituants et se place sous la responsabilité des industriels qui doivent fournir le certificat de conformité pour chaque constituant.

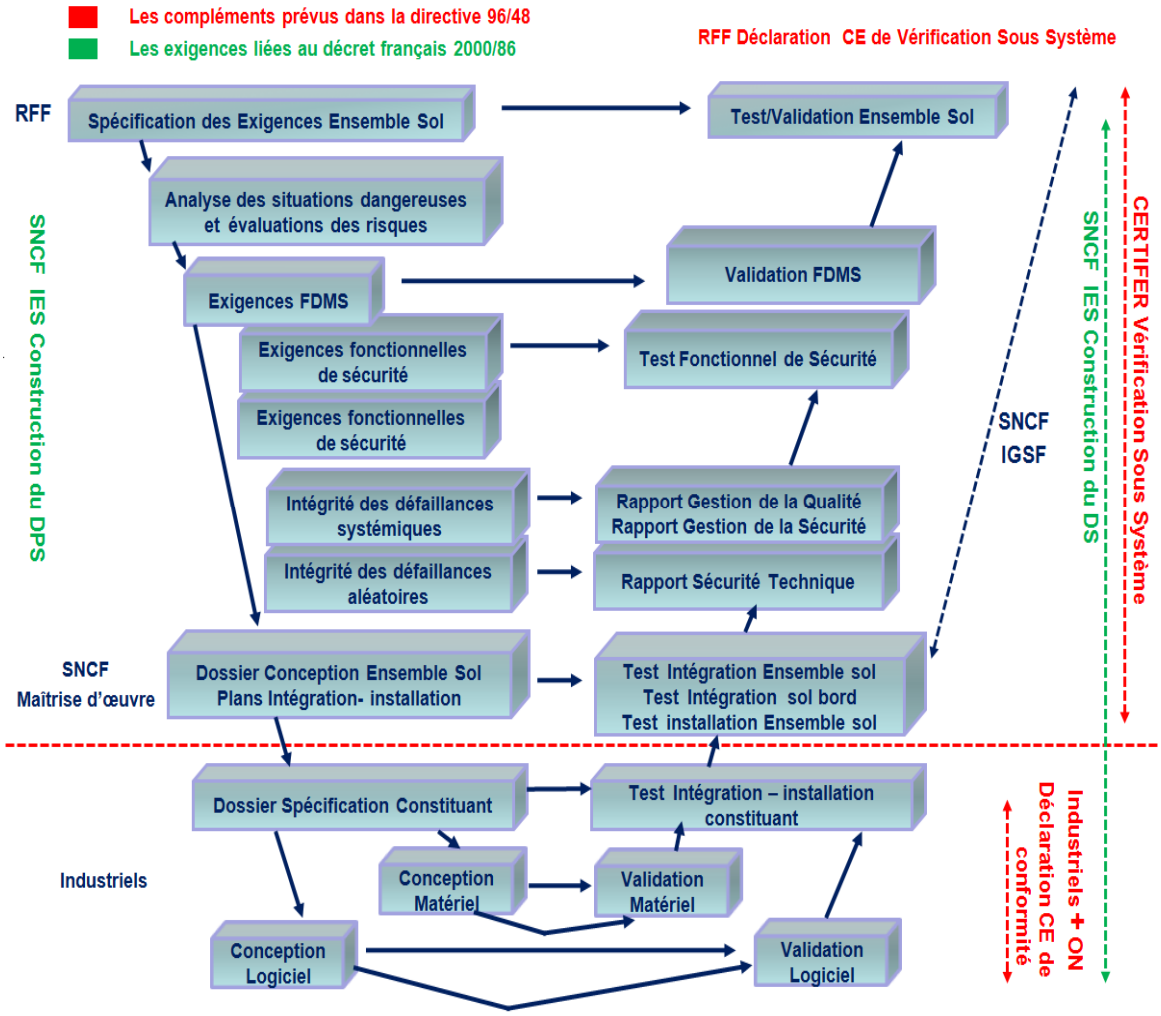


Figure 1.12: Cycle de Vie FDMS conforme à la norme 50126 [Berieau, 2004].

Quant à la partie descendante (de la figure 1.12) qui décrit les objectifs, elle est sous la responsabilité du promoteur. Au niveau de cette partie les dossiers de conception sont produits par la maîtrise d'œuvre SNCF et les DPS par la SNCF-IGSF (Ingénierie en Signalisation Ferroviaire). Enfin, au niveau de la partie remontante sont produites les démonstrations en matière de FDMS sous la responsabilité de l'équipe FDMS SNCF-IGSF indépendamment des constructeurs et de la maîtrise d'œuvre SNCF. Les éléments fournis par SNCF-IGSE sont ensuite rassemblés sous la responsabilité de la SNCF-IES pour produire le DS. Quant aux missions d'ON et d'OSTI, elles sont réalisées par CERTIFER.

1.3.3.2 La vérification et la validation du système ERTMS

Le système ERTMS/ETCS est défini par un ensemble de spécifications référencées dans les spécifications techniques d'interopérabilité pour le sous-système de contrôle-commande et signalisation. Ces spécifications nécessitent une phase de vérification et de validation [Tamarit & Guido, 2004] afin de pouvoir les corriger et les valider.

Actuellement, la mise en œuvre de l'ETCS est en cours, mais la phase de validation ne peut pas être considérée comme entièrement terminée, en raison de:

- La complexité d'ETCS, d'où la nécessité d'obtenir des clarifications et des corrections,
- L'élargissement de l'application d'ETCS au niveau des réseaux ayant des caractéristiques différentes et demandant des fonctionnalités supplémentaires.

Il faut aussi noter que la réalisation de l'interopérabilité nécessite une compréhension commune des spécifications par toutes les parties impliquées. En effet, une spécification qui pourrait être interprétée de différentes manières, est une source de problèmes. Dans ce contexte, l'agence ferroviaire européenne (ERA) a décidé d'étudier la possibilité de développer un système (outil matériel et logiciels) qui prend en charge la gestion transparente des spécifications ETCS, permettant ainsi:

- L'identification et la correction des erreurs et des ambiguïtés,
- La vérification rapide des modifications et la possibilité d'extensions fonctionnelles.

La validation permet ainsi de prouver qu'un système est adapté à l'utilisation et surtout de démontrer que son utilisation est sûre. La validation veut dire la validation des fonctionnalités et des conditions d'applications. Dans le cas d'ERTMS/ETCS, ces conditions d'application sont :

- Les règles opérationnelles,
- Les principes de signalisation,
- L'infrastructure et les conditions environnementales,
- Les performances imposées au train,
- …

Une première étape de la certification du système ERTMS réside dans l'évaluation de la conformité des constituants et sous-système (assemblage) aux STI, ce qu'on appelle l'interopérabilité technique.

Afin d'éviter les risques associés à la mise en œuvre de ERTMS/ETCS, les fabricants ont développé des outils de simulation très complexes et des bancs d'essai. Les premières implémentations (test de voie et application commerciale) ont montré que ces environnements de test peuvent également être utilisés pour valider les produits et systèmes. En outre, il s'est avéré que ces environnements représentent le seul moyen pratique et rentable pour valider les produits et sous-systèmes. En effet, plus de 95% de la validation peut être effectuée dans le laboratoire, chose qui a été prouvée lors de la mise en œuvre de projets réels en Suisse, en Italie, en Espagne, ...

Depuis le début du déploiement du système ERTMS en Europe, beaucoup de travaux ont été effectués: définition des spécifications pour le système ferroviaire à grande vitesse et conventionnel, test de voie et commercialisation de l'équipement à bord et l'équipement au sol. Il faut néanmoins admettre que la migration n'est pas aussi facile et rapide que prévu. Le déploiement est ainsi long et coûteux. Les différentes contraintes et difficultés sont expliquées dans la section suivante.

1.3.3.3 De la théorie à la réalité

Les projets de mise en œuvre d'ETCS diffèrent dans leurs stades d'avancement dans la quasi-totalité des pays européens [Winter, 2009b]. Ainsi, à la fin de 2008, l'exploitation commerciale de l'ETCS existait sur un total d'environ 2.650 km de lignes en Autriche (70 km), Bulgarie (440 km), Allemagne (135 km), Hongrie (210 km), Italie (470 km), Luxembourg (160 km), Pays-Bas (110 km), Espagne (970 km) et la Suisse (80 km), comme indiqué dans le tableau 1.3 ci-dessous.

Table 1.3: Lignes ETCS [Winter, 2009b]

Country	Line	Length (km)	ETCS level	Year	Supplier
Austria	Vienna to Hungarian border section Hegyesschalom	67	1	2006	Siemens / Thales
Bulgaria	Sofia – Plovdiv – Burgas	440	1	2001	Thales
Germany	Berlin – Halle/Leipzig	135	2	2005	Siemens / Thales
Hungary	Hodos – Zalacséb-S.	23	1	2004	Thales
	(Vienna-) Hegyeshalom – Budapest	Thales	1	2007	Thales
Italy	Milano – Bologna	182	2	2008	Ansaldo STS/Alstom
	Torino – Novara	90	2	2006	Ansaldo STS/Alstom
	Roma – Napoli	200	2	2005	Ansaldo STS/Alstom
Luxembourg	60% of the CFL network, including Luxembourg station	162	1	2005 - 2008	Thales
Netherlands	Betuwe Line Rotterdam – Zevenaar	110	2	2007	Alstom
Spain	Lérida – Roda	91	½	2006	Thales
	Córdoba – Malaga	155	½	2006	Invensys
	Roda – Barcelona	99	½	2008	Thales
	Madrid Valladolid	180	½	2007	Thales
	Madrid Lérida	440	½	2006	Ansaldo STS
Switzerland	Mattstetten – Rothrist (Olten – Berne)	45	2	2007	Thales
	Löstchberg base tunnel between Frutigen – Visp (Bern- Brig)	35	2	2007	Thales
Total length of lines with ETCS in commercial operation			2,644 km		

Depuis le début de la mise en œuvre du système ERTMS/ETCS, la migration n'a pas été réalisée comme prévu dans différents pays et ne suit pas la planification initiale [Wendler, 2009]. Les raisons de ces divergences [Winter, 2009c] sont les suivantes:

- L'héritage du passé

Chaque réseau ferroviaire national a ses propres contraintes, il semble alors difficile d'organiser le trafic ferroviaire autour de tous les différents systèmes de signalisation nationaux. En effet, malgré l'énorme similitude, les détails varient considérablement.

- Un problème de calendrier

Le calendrier et les délais de déploiement du système ERTMS le long du corridor est piloté par des considérations d'allocation de budget, du processus d'application de la loi, etc. Au cours de cette période, le déploiement du système ERTMS ne peut pas être interrompu et, plus encore, la définition des normes (STI) est adaptée à son propre rythme d'évolution. Ceci crée une instabilité de l'environnement qui influence le développement, la mise en œuvre, l'évaluation, l'exploitation et la maintenance du système ERTMS.

- Un problème d'organisation

L'introduction du système ERTMS et l'ouverture du marché ferroviaire a introduit de nouvelles parties prenantes, un nouveau partage des responsabilités et de nouveaux contrats. La vitesse de la migration dépend d'une définition claire du rôle de chacun des intervenants, leur champ d'application correspondant et la coordination de leur collaboration.

Ainsi, la Commission européenne exige de prendre en compte le retour de ces expériences en vue d'examiner la stratégie de migration et de l'adapter à la réalité du terrain pour le déploiement des corridors. En effet, le Memorandum of Understanding (MoU), signé en Juillet 2008, entre la Commission européenne et les associations européennes des chemins de fer (CER [ensemble des entreprises ferroviaires européennes], UIC [ensemble des exploitants ferroviaire], UNIFE [union des industriels fournisseurs ferroviaires], EIM [ensemble des gestionnaires d'infrastructures européens], GSM-R Industry Group, ERFA [Association Européenne de Fret Ferroviaire]), concerne le renforcement de la coopération pour accélérer le déploiement du système ERTMS.

Nos travaux de thèse font partie de cette approche européenne. En effet, dans le chapitre 5 de l'accord MoU, les procédures d'essai et de test sont considérées comme un facteur fondamental pour la mise en œuvre du système ERTMS avec succès. De plus, étant donné que le déploiement complet de ce nouveau système est long et coûteux, des évolutions seront nécessaires et soulèveront d'autres défis technologiques. ***Ainsi, l'objectif primordial de cette recherche est de diminuer les coûts de validation et de certification tout en facilitant***

l'interopérabilité par la reconnaissance mutuelle de constituants ERTMS entre états membres. Il s'agit de la problématique industrielle que nous avons déjà mentionnée dans l'introduction générale de cette thèse. ERTMS est considéré comme un système de contrôle commande complexe et réparti composé de constituants réactifs. Notre contribution consiste à générer des scénarios de tests dans le but de vérifier ces composants par rapport à la spécification ERTMS. Nous décrivons dans le paragraphe suivant la problématique scientifique.

1.4 Problématique scientifique

Notre recherche se focalise sur la génération de scénarios de tests pour la vérification des constituants d'un système complexe et réparti dont la spécification est représentée de manière informelle. Dans [Lemoigne, 1990], l'auteur décrit la modélisation des systèmes de complexes. Il définit la notion d'intelligibilité qui n'a pas la même signification pour un système compliqué (explication) que dans un système complexe (compréhension). En effet, au début de son ouvrage, l'auteur distingue ce qu'est la complexité et ce qu'est la complication. Il montre comment face à un problème, il élabore des modèles sur lesquels il est possible de raisonner. Pour l'auteur, « modéliser un système complexe c'est d'abord modéliser un système d'actions. La modélisation de l'action complexe se caractérise par la notion générale de processus qui se définit par son exercice et son résultat ». Quant à Morin [Morin, 1990], il qualifie un système de complexe afin de décrire quelque chose dont le comportement, la structure et les fonctions sont difficiles à comprendre.

Selon [Chapurlat, 2007], un système complexe et réparti est défini sur plusieurs niveaux :

- **La composition** : Un système complexe est composé par des constituants éventuellement hétérogènes et possédant des caractéristiques et des propriétés propres.
- **Les interactions entre les constituants** : Les différents constituants d'un système complexe interagissent et font ainsi apparaître les caractéristiques de chaque constituant dans son environnement global.
- **L'émergence de nouveaux phénomènes**: Les constituants du système complexe sont souvent autonomes et peuvent évoluer indépendamment les uns des autres sous l'influence de l'environnement global et des autres constituants [Morin 1990]. Cette évolution dynamique se concrétise à travers l'évolution des différentes caractéristiques produisant ainsi de nouveaux comportements et de nouvelles interactions.

1.4.1 Gestion de la complexité

Concernant la gestion de la complexité, nous nous intéressons particulièrement à :

- La difficulté d'obtenir une compréhension globale et synthétique du système à vérifier,
- La définition de représentations communes afin de pouvoir dialoguer facilement entre les parties concernées par la conception, le développement et la mise en œuvre du système à tester,
- La difficulté de structuration résultant de la gestion d'un grand nombre d'informations de modélisation.

Une modélisation orientée-objet nous apporte une première réponse à la structuration du système et par conséquent à cette gestion de la complexité. En effet, la technologie orientée-objet nous garantit des facteurs de qualité tels que la modularité, la flexibilité et la réutilisation. [Booch-1994] [Blair-1998]. Ensuite, le choix du formalisme UML (Unified Modelling Language) en tant que langage de modélisation unifié est considéré comme une deuxième réponse apportée à la gestion de la complexité du système et la modélisation de la spécification informelle. Il est en effet utilisé pour spécifier, visualiser, construire et documenter les artefacts d'un système logiciel. Il permet aussi de faciliter la communication entre les différentes équipes intervenant sur le développement du système. Dans le chapitre3, les avantages, mais aussi les limites, de l'utilisation d'UML dans le contexte de génération automatique de scénarios de test pour la vérification d'un système complexe sont détaillés.

1.4.2 Modélisation des aspects dynamiques de la spécification

Lors de la modélisation d'un système complexe et réparti, plusieurs points de vue doivent être pris en compte. Il ne s'agit plus d'un modèle unique pour représenter le système mais plutôt d'un ensemble de modèles complémentaires les uns des autres. Cette approche multi vues permet d'améliorer la compréhension du système :

- **La vue fonction** : décrit ce que le système doit faire dans son environnement. Quels sont : sa finalité, sa mission et ses objectifs ?
- **La vue structure** : décrit de quoi le système est fait. Elle détaille les ressources, les configurations et l'organisation du système qui lui permettent de remplir ses tâches.
- **La vue comportement** : décrit les aspects dynamiques définis dans la spécification du système. Elle définit l'évolution du système ainsi que les scénarios possibles lors de cette évolution. Elle détaille également les conditions et les configurations permettant de passer d'un scénario à un autre. Enfin cette vue permet aussi de décrire comment le système est piloté pour faire face à un changement ou à un incident.

Le langage UML possède la caractéristique d'une modélisation multi vues. Il permet de décrire et de visualiser un système à l'aide de diagrammes. Chaque diagramme constitue une perspective du modèle, possède une structure et véhicule une sémantique précise. Combinés, les différents types de diagrammes UML offrent une vue complète des aspects statiques, fonctionnels et dynamiques d'un système. La génération de scénarios de test dans le but de vérifier un système par rapport à sa spécification repose essentiellement sur la modélisation de la vue comportement. Il est ainsi possible de modéliser les fonctionnalités du système réparti ainsi que les différents flots de communication entre les différents constituants réactifs. En s'appuyant sur la spécification, nous voulons tester ces constituants. Il s'agit du test de conformité. La spécification joue un rôle primordial dans cette approche, une formalisation de cette dernière s'impose, ce qui constitue le premier problème à résoudre dans le cade de nos travaux de thèse.

1.4.3 Vers une approche mixte pour la génération de scénarios de test

La modélisation de la spécification informelle en UML dans l'objectif de vérifier des constituants réactifs en utilisant des tests de type boîte noire nous incite à proposer une approche mêlant l'utilisation du langage UML avec des langages mathématiques formels puissants, tel que les réseaux de Petri. Cette approche mixte convient à nos attentes de génération des scénarios de test à partir de modèles formels. En effet, il nous paraît essentiel de fournir des modèles UML semi-formels mais qui sont facilement utilisables et surtout qui puissent être couplés à des modèles formels. Néanmoins, ce couplage soulève en lui-même un certain nombre de verrous. Par exemple, comment rendre compatibles les modèles formels avec la représentation semi-formelle ? Quelles passerelles, transformations ou relations entre les représentations formelles et semi-formelles peuvent être définies ? Ces différents points sont détaillés dans le chapitre 3.

Nous avons choisi la transformation des modèles UML vers des réseaux de Petri afin de pouvoir générer les scénarios de tests. Ces modèles graphiques et mathématiques constituent des modèles formels décrivant le comportement dynamique et l'échange de messages entre un constituant et son environnement extérieur. Nous parlons dans notre contexte de constituants boite noire, nous ignorons alors le fonctionnement interne de chaque constituant. Notre approche de génération de scénarios de test vise la vérification des différentes communications entre les constituants conformément à la spécification. Le modèle réseau de Petri généré à la suite de la transformation des modèles UML des différents constituants, devra contenir les interactions entre les composants. Un scénario de test généré à partir d'un réseau de Petri représentera une séquence de franchissement partant d'un état initial et arrivant à un état final. Notre méthode de génération de scénarios de tests est détaillée dans le chapitre 4.

1.4.4 Démarche et objectifs

Le contexte industriel de cette thèse impose des exigences de sécurité strictes. La méthodologie proposée consiste à générer des scénarios de test de conformité de type boîte noire à partir de la spécification décrite de manière informelle. Cette dernière n'est pas appropriée pour la phase de génération de scénarios de test, d'où la nécessité de modéliser et formaliser cette spécification ([Jabri et *al*, 2008b], [Jabri et *al*, 2008c], [Jabri et *al*, 2009a]).

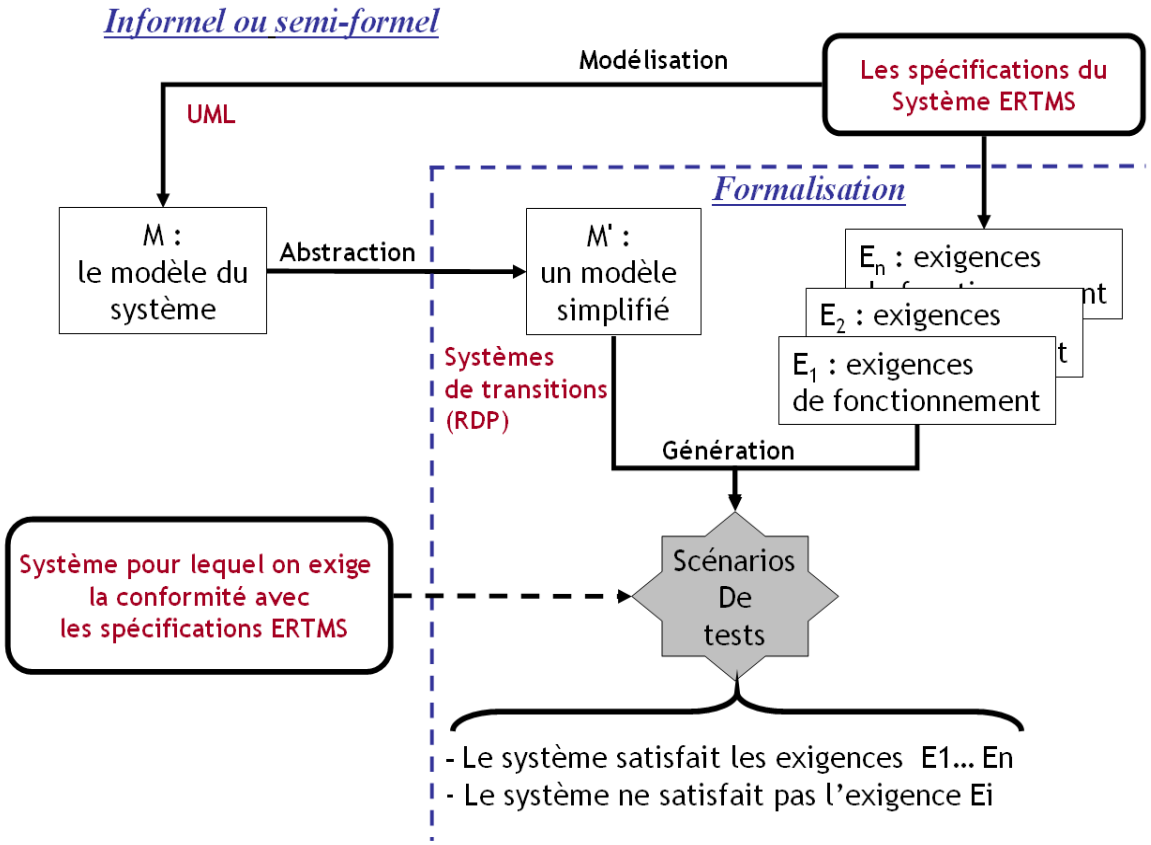


Figure 1.13: Démarche et objectifs

Comme le montre la figure 1.13, le langage UML (Unified Modelling Language), étant un langage de modélisation orienté objet, est utilisé afin de modéliser la spécification produisant ainsi des modèles semi-formels. La première contribution de nos travaux consiste à transformer nos modèles semi-formels en modèles formels en utilisant les réseaux de Petri. Ils permettent de modéliser le comportement dynamique des systèmes réactifs et concurrents. La deuxième contribution consiste à générer automatiquement des scénarios de tests à partir des modèles formels en tenant en compte des exigences définies dans la spécification. Il s'agit du test de conformité permettant de vérifier l'implantation à tester par rapport à la spécification. L'exécution de nos scénarios de test sur une plate-forme de simulation nous permettra de comparer l'implantation à tester par rapport aux attentes de la spécification et de donner les verdicts de test. La couverture des scénarios générés sera également étudiée.

1.5 Conclusion

Nous avons présenté dans ce chapitre le système ERTMS qui représente le contexte industriel de notre recherche. Il s'agit d'un système complexe et critique utilisant un grand nombre d'acteurs et devant se conformer à des contraintes de sécurité très strictes. La mise en service de ce dernier devrait avoir plusieurs avantages : assurer l'interopérabilité ferroviaire européenne, dynamiser le secteur ferroviaire, augmenter la qualité du transport ferroviaire,… Cependant, la mise ouvre de ce système nécessite une demande d'autorisation auprès des organismes concernés et des étapes de vérification et de validation. Actuellement, les acteurs concernés travaillent sur la phase de déploiement de ce nouveau système. Il faut en effet gérer sa cohabitation avec les anciens systèmes. C'est dans cet axe de recherche que s'inscrivent nos travaux. Ainsi, nos principales motivations consistent à vérifier et valider des constituants ERTMS dans leurs environnements d'interactions. Ces interactions sont décrites dans les spécifications fonctionnelles et les spécifications systèmes ERTMS (FRS et SRS). Les constituants ERTMS sont des constituants réactifs. Nous définissons dans le chapitre suivant un état d'art sur les différentes méthodes et outils de vérification de constituants réactifs. Nous nous intéressons en particulier à la technique de test de conformité de type boîte noire comme méthode de vérification. Ensuite, nous présentons les différentes méthodes de génération automatique de scénarios de test de conformité.

2 VERIFICATION DE CONSTITUANTS REACTIFS

« Rien n'est plus pratique qu'une bonne théorie. »
(Hubert Kadima)

Résumé:

Dans ce chapitre nous parlons de la vérification de constituants réactifs. Nous entendons par un constituant réactif un constituant qui interagit continuellement avec son environnement. Dans notre contexte d'étude, le comportement de ces constituants est visible à travers les interactions qu'ils ont avec l'extérieur. L'objectif est de valider ces constituants à partir de la spécification ERTMS. En informatique, deux techniques essentielles de vérification existent : la vérification formelle et le test. Le « theorem proving » et le « model checking » constituent deux méthodes de vérification formelle qui supposent l'existence d'un modèle de l'implantation à tester. Or vu le type boîte noire de nos constituants, nous n'avons pas un modèle précis décrivant l'implantation. La méthode du test et en particulier le test de conformité reste la méthode incontournable pour la validation de systèmes réactifs. Néanmoins, les méthodes formelles sont utilisées dans le test afin de donner plus de confiance dans le processus de génération automatique de scénarios de test. La spécification doit être décrite de manière formelle. Nous distinguons deux grandes familles de génération de scénarios de test de conformité se basant sur la spécification : les méthodes basées sur les automates et les méthodes basées sur les systèmes de transitions à entrées sorties. Des outils académiques et industriels ont été mis en œuvre afin de faciliter la génération automatique de tests. Une fois ces tests générés, des critères permettent de juger de la qualité de ces tests.

Sommaire du chapitre 2

2.1 Introduction

Ce chapitre a pour objectif de préciser le positionnement scientifique de notre travail. Nous avons souligné au chapitre précédent que le déploiement du système de contrôle commande et de signalisation ferroviaire ERTMS nécessite de longues phases de vérification et de test. Il s'agit d'un système réparti et complexe dont la validation nécessite la vérification de ses constituants réactifs. Par réactif, nous entendons un constituant qui réagit aux stimuli de son environnement.

Dans ce chapitre, nous décrivons deux techniques de vérification : la vérification formelle et le test. La vérification formelle permet de s'assurer que les modèles d'un système ou d'un programme sont corrects. Cette technique s'effectue sur un modèle du système et non pas sur le système lui-même dans son environnement d'exécution. Quant à la technique du test, elle reste incontournable dans le contexte de validation de systèmes réactifs et représente la technique la plus utilisée. Nous avons choisi cette technique pour la vérification de nos constituants réactifs et en particulier le test de conformité. Ce type de test appelé test boîte noire convient parfaitement à notre contexte industriel. Les constituants réactifs que nous voulons vérifier sont des constituants dont le code est inconnu et seule l'interface est accessible. Le test de conformité détaillé dans la section 2.3.3 permet de vérifier si le comportement d'une implantation réelle d'un système réactif est correct par rapport à la spécification. Il inclut plusieurs notions : spécification, implantation sous test, génération de cas de test, exécution des cas de test et verdicts de test. Cependant l'écriture manuelle de cas de test est consommatrice en temps et en coût. C'est l'un des arguments qui a encouragé l'automatisation du processus de génération de tests en se basant sur des méthodes formelles. Ces méthodes augmentent la confiance dans les résultats des tests obtenus.

De manière générale, les techniques de test possèdent des caractéristiques communes : fournir des entrées à l'implantation sous test pour la stimuler et observer les sorties, ensuite comparer ces sorties avec les sorties attendues pour conclure enfin sur la correction ou non. Le test ne garantit jamais la correction et ne peut pas être exhaustif. C'est la raison pour laquelle, des tests raisonnables (en termes de coût) mais suffisamment complets doivent être sélectionnés afin d'acquérir une confiance suffisante dans le système quand celui-ci passe correctement les tests. Cette sélection se base sur des critères de couverture (cf. section 2.5) structurels du code dans le cas où ce dernier est connu. Dans le cas de code inconnu (test boîte noire), l'estimation de la qualité des tests peut se faire en faisant des hypothèses (cf. section 2.5.4) sur les implantations à tester [Jéron, 2004].

La première section de ce chapitre introduit les systèmes répartis et les constituants réactifs. Dans la section suivante, nous définissons un état de l'art des techniques de vérification de constituants réactifs. Ensuite, les diverses méthodes et outils de génération automatique de scénarios de test de conformité sont détaillés. A la fin de ce chapitre, nous définissions les critères de couverture de tests.

2.2 Système réparti & constituant réactif

2.2.1 Système réparti

Le système ERTMS constitue le contexte industriel de nos travaux de thèse. Il est en effet un système complexe et réparti, composé par des constituants réactifs.

Définition 2.2.1 / Système réparti. Un système réparti (ou distribué, «distributed system»), est composé de plusieurs systèmes calculatoires autonomes (sinon, c'est un système non réparti) sans mémoire physique commune (sinon c'est un système parallèle) qui communiquent par l'intermédiaire d'un réseau (quelconque) [Tardieu-01].

Les systèmes répartis sont caractérisés par les points suivants [Tardieu-01]:

- ✓ Accès distant: un même service peut être utilisé par plusieurs acteurs, situés à des endroits différents,
- ✓ Redondance: des systèmes redondants permettent de pallier une faute matérielle, ou de choisir le service équivalent avec le temps de réponse le plus court,
- ✓ Performance: la mise en commun de plusieurs unités de calcul permet d'effectuer des calculs parallèles en des temps plus courts,
- ✓ Confidentialité: les données brutes ne sont pas disponibles partout au même moment, seules certaines vues sont exportées.

Le système réparti est alors vu comme une entité composée d'un certain nombre de processus interconnectés par un certain nombre de liens et d'un système de communication. Ce dernier définit une méthode de communication par variables partagées ou par envoi de messages. Comme nous l'avons déjà décrit dans l'introduction générale et dans le chapitre 1, ERTMS est composé de constituants qui interagissent ensemble par l'envoi de messages et l'appel de procédures. Ces constituants sont appelés des constituants réactifs. Comment définit-on un constituant ERTMS réactif ?

2.2.2 Constituant réactif

Dans un contexte ferroviaire, le système ERTMS est composé de constituants. ***Un constituant ERTMS n'est autre qu'un composant logiciel*** car ce dernier représente une boîte qui encapsule du logiciel. Il possède des interfaces au travers desquelles sont fournis des services. Ces interfaces peuvent être fournies par un composant qui joue le rôle de

prestataire du service, ou requises par un composant qui joue alors le rôle de client [Pareaud, 2009]. Cette définition de composant logiciel correspond parfaitement à celle du constituant ERTMS.

L'avantage de baser notre méthodologie sur une approche par composants est le fait que ces derniers permettent la réutilisation. De plus, ils facilitent l'ingénierie des architectures dirigées par les modèles en considérant le logiciel comme un ensemble d'entités qui interagissent et s'échangent des services. Le ***composant est constitué*** par [Pareaud, 2009] :

- ✓ Un ***contenu*** composé par des variables et des instructions, il s'agit d'un algorithme au sens informatique du terme ;
- ✓ Des ***interfaces*** qui permettent d'abstraire le contenu du constituant aux services qu'il rend ou dont il a besoin (appel de procédures, évènements, communication). Les interfaces sont de deux types :
  - Fournies si le composant rend le service ;
  - Requises si le composant nécessite le service pour rendre un service correct;
- ✓ ***D'autres composants***. Dans ce cas, le composant est qualifié de composite.

Suite à cette définition des composants, il est possible de représenter un logiciel complexe comme une composition de boîtes interconnectées à travers un modèle à composants. En effet, le modèle à composants constitue une abstraction du logiciel simplifiant ainsi toute modification de ce dernier pour deux raisons différentes :

- La première concerne l'isolement de l'algorithme et ses données susceptibles d'être modifiées dans un composant.
- La deuxième concerne les interactions entre les composants à travers des interfaces fournissant ou demandant des services.

La description des services est indépendante de leur implémentation, il est alors possible de remplacer l'implémentation d'un service par un autre pour satisfaire à des besoins non-fonctionnels de ce service par exemple, sa consommation de ressources système. Pour intégrer un composant dans son environnement, il suffit de le créer, connecter ses interfaces requises aux interfaces fournies par les composants existants dans le système et puis le démarrer. Ainsi, d'autres composants peuvent être connectés aux interfaces qu'il fournit. Ce même composant peut être supprimé en suivant les étapes inverses : les composants sont déconnectés des interfaces qu'il fournit, le composant est ensuite arrêté, puis ses interfaces requises sont débranchées, et enfin, le composant est détruit.

Le constituant est dit ***réactif*** quand il réagit continûment à son environnement physique à une vitesse déterminée par ce dernier [Lesens, 1997]. Ce concept s'oppose d'une part aux

systèmes transformationnels et d'autre part aux ***systèmes interactifs***. Les systèmes transformationnels sont des systèmes classiques en informatique qui disposent des entrées dès leur initialisation, opèrent sur ces données d'entrée un ensemble de transformations complexes et fournissent les résultats à la fin de leur exécution (cf. Fig.2.14); le compilateur est un exemple de ce type de systèmes.

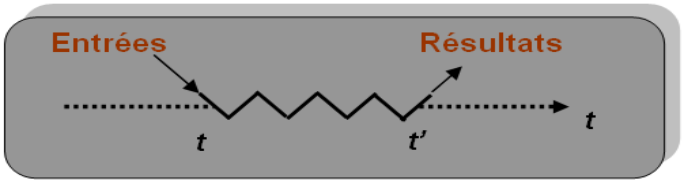


Figure 2.14: Les systèmes transformationnels

Quant aux ***systèmes interactifs***, ils sont semblables aux systèmes réactifs. Ils interagissent en effet à leur environnement mais à une vitesse qui leur est propre (cf. Fig.2.15). Autrement dit, ils peuvent mémoriser certaines entrées avant de les prendre en compte sans que cela ait une influence sur le fonctionnement du système. Les systèmes d'exploitation constituent un exemple de ce type de système [Jourdan, 94].

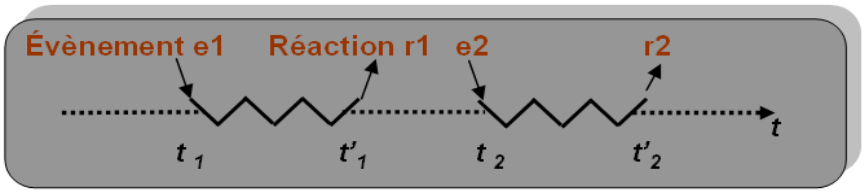


Figure 2.15: Les systèmes interactifs

Nous nous intéressons aux ***systèmes réactifs*** particulièrement utilisés dans le cadre du contrôle des systèmes critiques comme le transport, le domaine nucléaire, la commande de processus industriels, etc. Les systèmes réactifs sont des systèmes critiques et doivent impérativement satisfaire des contraintes strictes de fonctionnement. Une erreur peut entraîner des coûts considérables, aussi bien en terme financier qu'en terme humain. Quand ils interagissent avec leur environnement, les systèmes réactifs ne maitrisent pas le rythme de ces interactions. Ce rythme est en effet imposé par l'environnement, c'est la caractéristique principale des systèmes réactifs qui interdit la perte des entrées. Les sorties produites dépendent alors des valeurs de ces entrées et de leur instant d'occurrence, il s'agit de systèmes déterministes (cf. Fig.2.16).

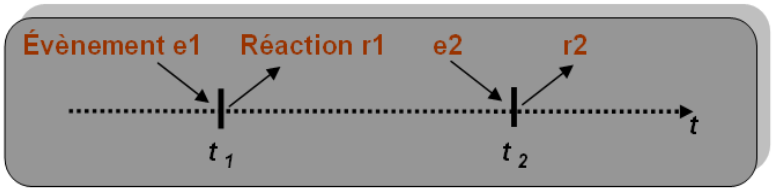


Figure 2.16: Les systèmes réactifs

Ces systèmes sont aussi parallèles car ils contiennent un ensemble de composants fonctionnant en parallèle afin de réaliser le comportement global souhaité. Ce fonctionnement parallèle des différents composants n'exige pas forcément une implantation parallèle du système [Jourdan, 1994].

Dans la section suivante, nous présentons un état de l'art des différentes techniques de vérification de constituants réactifs. Nous définissons en premier lieu les méthodes formelles et en second lieu la méthode du test, en particulier le test de conformité.

2.3 Vérification des constituants réactifs

La complexité des systèmes informatiques ne cesse pas de croître ce qui peut provoquer la présence d'erreurs durant la conception de ces derniers. Malheureusement, dans certains cas de logiciels critiques, un comportement erroné peut conduire à des conséquences dramatiques. D'ailleurs, il existe plusieurs exemples, tristement célèbres, d'erreurs qui ont provoqué des dégâts matériels ou financiers et même parfois des pertes humaines. Un des exemples le plus connu est celui du crash d'Ariane 5, qui aurait pu être évité suite à une vérification plus rigoureuse ([Herbreteau, 2001], [Fleury, 2001]). De plus, dans la plupart des drames qui se produisent, les coûts des dégâts engendrés sont disproportionnés par rapport à la simplicité de l'erreur conduisant au problème.

Dans le cas des transports avioniques et ferroviaires, le niveau de criticité est beaucoup plus élevé car un logiciel erroné peut conduire à la perte de vies humaines. Il est alors indispensable de vérifier le bon fonctionnement de ces systèmes tout au long de leur cycle de développement en V (cf. Fig.2.17). Avant la mise en œuvre d'un produit final, ce cycle contient une phase d'analyse, une phase de conceptualisation, une phase de réalisation et une phase de vérification de la conformité du produit final par rapport aux exigences. La phase d'analyse entraîne la conception des architectures fonctionnelle et opérationnelle à partir de l'analyse des besoins du système. L'architecture opérationnelle est la base de la réalisation physique du système. Les différents constituants du système sont conçus et programmés de manière indépendante puis intégrés pour la conception du système global [Godary, 2004].

Chacune des différentes phases du cycle de développement a sa propre méthodologie de vérification (cf. Fig.2.17) afin de détecter les erreurs à des stades moins avancés. Quant à la vérification du produit final, elle se fait par rapport aux fonctionnalités attendues rédigées dans les spécifications du système. Le cahier des charges permet de décrire ces spécifications de manière informelle. Les méthodes de modélisation formelle permettent de modéliser les spécifications en résolvant les problèmes de redondance, d'ambiguïté et d'erreur liés à la formulation en langage naturel des spécifications décrites dans le cahier des charges. Ces

méthodes formelles produisent des modèles de spécifications qui respectent une sémantique statique et dynamique et offrent des outils permettant de vérifier les fonctionnalités du système concerné. C'est ainsi que plusieurs techniques de vérification peuvent s'appliquer à des niveaux différents du cycle de développement du système selon que l'on souhaite vérifier les modèles formels du système ou bien le produit livré à la fin (cf. Fig.2.17). Dans la section suivante, nous allons détailler ces méthodes formelles de vérification et donner quelques exemples.

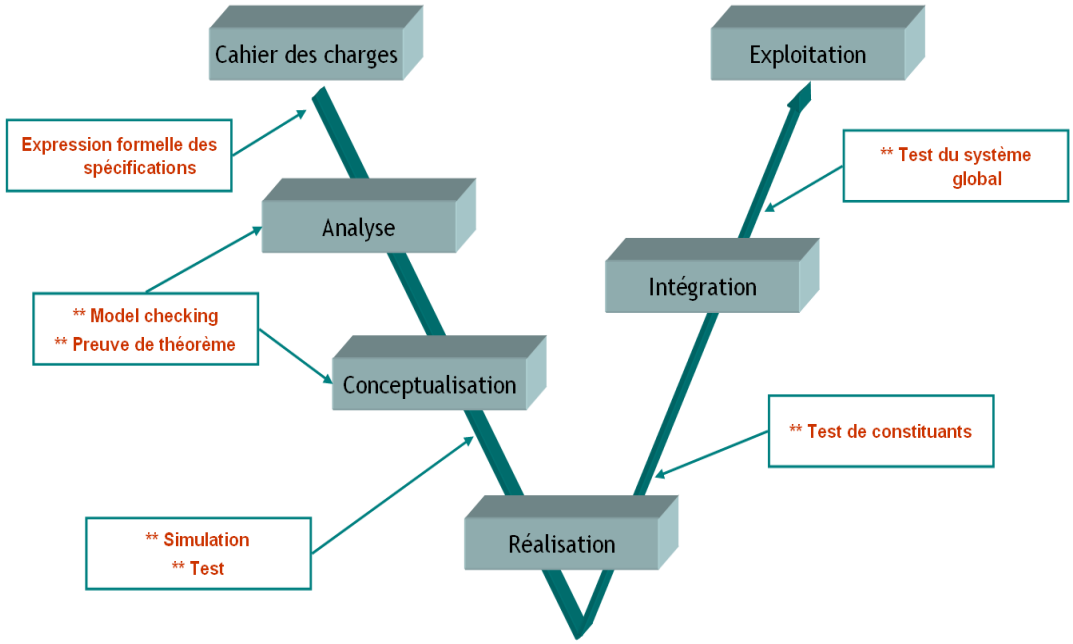


Figure 2.17: Le cycle de développement de logiciel en V

2.3.1 Techniques de vérification formelle

Les techniques de vérification formelle ont été développées dans les années 80 dans le but de vérifier un ensemble de propriétés sur le modèle formel conformément aux besoins informels (exprimés par les utilisateurs). Deux grandes familles de vérification formelle existent [Evrot et al, 2006a] : ***l'évaluation sur modèle*** (« model-checking ») et ***la preuve de théorème*** (« theorem-proving »).

2.3.1.1 La preuve de théorème

Dans la vérification basée sur la démonstration de théorème, nous considérons une spécification formelle et nous essayons de démontrer que les propriétés sont bonnes. Il s'agit d'une approche syntaxique et axiomatique qui permet de déduire des faits d'autres faits en se servant des propositions, des axiomes et des règles de déduction. Les assistants de preuve sont des outils informatiques de démonstration qui permettent de finaliser une preuve suggérée par l'utilisateur grâce aux méthodes de déduction automatique [Chapurlat, 2007]. Parmi les assistants de preuve, nous notons : PVS (Prototype Verification System) [Owre et al,

1996], Coq [Coq Development Team, 2002], HOL [Melham and Gordon, 1993] et Isabelle [Paulson, 1994].

Les techniques de preuve se basent sur des démonstrations mathématiques pour vérifier la compatibilité des modèles. Elles permettent de déduire des conclusions à partir d'une description d'évènements ou d'opérations permettant de faire évoluer le système. L'automatisation complète des techniques de preuve est rarement possible même en utilisant les assistants de preuve. En effet, l'utilisateur est souvent obligé de guider la preuve en interagissant avec l'assistant [Evrot, 2008]. Dans les travaux de [Roussel et Denis, 2002], la technique de preuve a été utilisée dans un contexte de Système à Evènements Discrets (SED) afin de vérifier si les propriétés sont satisfaites par l'ensemble automate et logiciel. Ensuite, dans les travaux de ([Kim et al, 2005], [Son&Seong, 2003]), les outils de vérification de preuve ont été utilisés afin de vérifier les modèles de spécifications du logiciel dans le contexte de systèmes exigeant la sécurité. Cette utilisation a provoqué une contrainte supplémentaire, à savoir la formalisation des spécifications du logiciel dans un langage acceptable par l'assistant de preuve et la traduction des exigences système de sécurité fonctionnelle sous forme de propriétés.

Quant aux méthodes de model-checking, elles se basent généralement sur des modèles de comportement du type états/transitions ; un modèle semblable aux formalismes utilisés pour spécifier les logiciels de contrôle-commande.

2.3.1.2 L'évaluation sur modèle

Les techniques d'évaluation sur modèle permettent à partir d'un modèle à états finis d'un système et d'une propriété énoncée formellement de vérifier si cette propriété est vraie pour ce modèle. Les deux ouvrages de référence qui détaillent ces techniques sont : [Clarke *et al*, 1999] et [Bérard *et al*, 2001].

Le principe du *model-checking* est d'examiner tous les scénarios et les comportements possibles du système à travers son modèle tout en parcourant l'ensemble des états atteignables. Ainsi, cette technique permet de s'assurer qu'aucun des états ne viole les propriétés à vérifier. Ces propriétés comme par exemple la recherche d'une situation de blocage sont à formaliser lors la phase de modélisation. Des méthodes d'analyse automatique du modèle permettent de vérifier ces propriétés et de parcourir les différents états du système. Ces méthodes sont implémentées dans des outils spécifiques : les model-checkers. Ces derniers peuvent générer un contre-exemple, c'est-à-dire une trace du chemin d'exécution ayant mené à un état dans lequel une des propriétés à vérifier n'est pas respectée. L'analyse de cette trace, souvent en utilisant un simulateur, permet de détecter plus facilement l'origine de la défaillance [Rushby, 2002]. Le model-checking permet alors

une vérification formelle exhaustive des propriétés sur le modèle du système [Ortmeier *et al*, 2003]. Les trois model-checker connus et souvent utilisés sont : ***PRISM*** développé à l'Université de Birmingham par l'équipe de Marta Kwiatkowska, ***SMV*** développé par les laboratoires Cadence de Berkeley, et ***UPPAAL*** développé par une collaboration entre le département d'Information Technology de l'université d'Uppsala (Suède) et le département de Computer Science de l'université D'Aalborg au Danemark [Evrot, 2008].

Les techniques de vérification par *model-checking* ont l'avantage d'avoir un processus complètement automatisé, à partir du moment où l'utilisateur fournit les modèles formels des propriétés et du système à vérifier. Néanmoins, plusieurs verrous rendent l'utilisation de ces techniques à une échelle industrielle un peu difficile. Le problème bien connu est celui de l'explosion combinatoire qui restreint l'utilisation du model-checking à des systèmes de petite taille. En effet, la taille des structures de données croît très rapidement si les paramètres du système augmentent. L'algorithme de vérification est alors contraint par la puissance des calculateurs disponibles actuellement. Une des solutions consiste à utiliser un raisonnement compositionnel. Il s'agit en effet de vérifier chaque composant du système de façon indépendante, puis de conclure sur la correction de la composition des composants entre eux. Le système n'est plus vérifié dans son ensemble mais composant par composant. Ceci engendre deux contraintes : ***(1)*** pouvoir définir les propriétés locales qu'un composant doit respecter ; ***(2)*** prouver que la conformité des composants aux propriétés locales implique la conformité de l'ensemble du logiciel aux propriétés globales.

Les techniques de vérification formelles que nous venons de décrire nous permettent de détecter rapidement les erreurs après la phase d'analyse et la phase de conception du cycle de vie du logiciel (cf. Fig.2.17). ***Cependant, ces techniques ne remplacent pas la phase de test qui reste indispensable pour détecter les erreurs liées à la phase de réalisation physique du système***. En effet, ces techniques permettent la vérification formelle d'un modèle du système et non du système lui-même. Quant au test, il s'agit d'une activité consistant à soumettre le produit final à un ensemble d'exécutions simulant son environnement après mise en œuvre. L'analyse et l'observation des réactions du système dans cet environnement d'exécution permettent de détecter les comportements erronés. Dans un contexte de développement industriel, l'activité de test devient très coûteuse, une des raisons pour laquelle ont été développées des méthodes de génération automatique de cas de test à partir d'une spécification formelle.

2.3.2 Vérification par la méthode de test

Comme le montre la figure 2.17, le test fait partie intégrante du cycle de développement du logiciel. Il s'intéresse au produit final et intervient après la phase de réalisation pour tester

des constituants indépendants et puis tester le système global après intégration. C'est une activité qui a pour objectif d'exécuter le système dans l'intention de détecter une erreur [Myers, 1979]. Le principe de cette exécution consiste à appliquer des données d'entrées au système sous test et puis à observer les sorties afin de les comparer aux spécifications du cahier des charges. Le test peut être défini ainsi [Beizer, 1995]:

- ***Pouvoir du test***: le test met seulement en évidence des dysfonctionnements éventuels mais ne permet pas de prouver l'absence d'erreurs.
- ***Coût du test***: le test couvre près de 50 % du coût total du développement d'un logiciel.
- ***Phases du test***: plusieurs niveaux définissent le test tout au long du cycle de développement d'un logiciel : tests unitaires, tests d'intégration, tests système, tests recette et test non-regression (cf. Fig.2.17).

Les différentes phases de test du cycle de développement d'un logiciel sont représentées dans la figure 2.18 et expliquées ci-dessous.

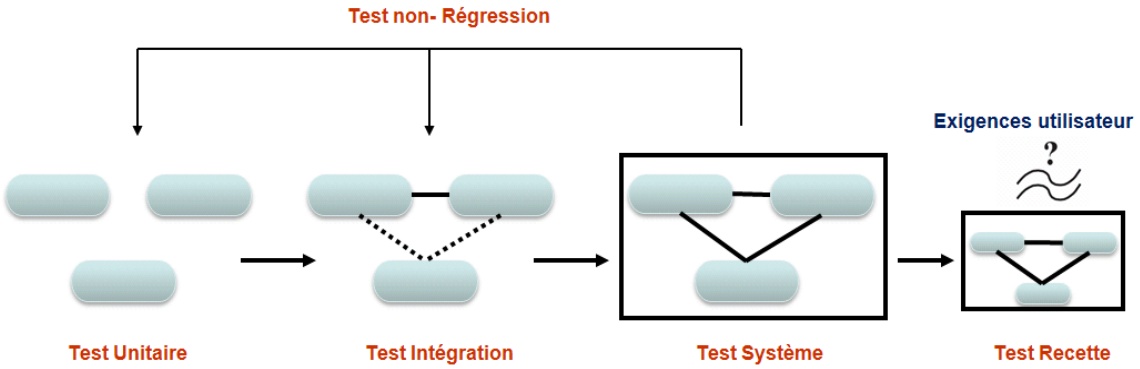


Figure 2.18: Les différentes phases de test

- **Test unitaire** « *Testing of individual software units or groups of related units* » [IEE, 1990]: C'est le processus qui permet de tester les composants d'un petit programme informatique, tels que les sous-programmes. L'objectif de cette phase de test est de vérifier le fonctionnement des différentes parties du programme de manière indépendante.
- **Test intégration** « *Testing in which software components, hardware components, or both are combined and tested to evaluate the interaction between them* » [IEE, 1990]: Il fait suite au test unitaire et détermine si tous les composants du programme fonctionnement correctement quand ils sont liés ensemble. Ce test permet de vérifier les interactions entre les composants et a pour but de trouver des erreurs liées à la mise en interaction de ces derniers.

- **Test système** « *Testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements* » [IEE, 1990]: Ce test examine le programme entier alors que le test d'intégration s'intéressait aux connexions d'interfaces entre composants. Son objectif est de détecter les erreurs qui peuvent apparaître à travers une vision globale du programme.
- **Test recette** « *Test réalisé par l'utilisateur pour vérifier si l'application répond bien à ses besoins* »: Ce test permet au client de s'assurer du bon fonctionnement du logiciel par rapport aux spécifications du cahier des charges. Il conditionne en effet l'acceptation du produit par le client. Le cahier des charges peut contenir des scénarios d'utilisation préétablis qui peuvent servir comme test de recette.
- **Test de non-régression** «*Processus de test d'un programme dont le code a été modifié*» : La modification du code peut se produire lors de l'ajout de nouvelles fonctionnalités par exemple. L'objectif de ce test de non-régression est de s'assurer que le programme modifié satisfait encore les exigences et que ses fonctionnalités précédentes n'ont pas été affectées.

Au niveau du test de logiciel ([Myers, 1979], [Beizer, 1990]), on distingue principalement deux ***types de tests*** : le test structurel et le test fonctionnel (cf. Fig.2.19). Le test structurel se base sur l'analyse de la structure interne d'une implantation tandis que le test fonctionnel consiste à vérifier si une implantation logicielle ou matérielle est conforme à sa spécification. Cette dernière est définie comme une description des comportements désirés qui décrit ce que le système doit faire et non pas comment il est fait.

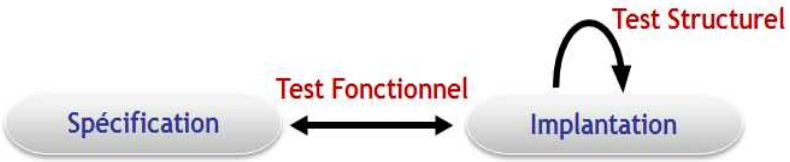


Figure 2.19: Le test fonctionnel et le test structurel

2.3.2.1 Le test structurel

Le test structurel, ou ***le test boite blanche*** [Beizer, 1990], est une méthode qui s'appuie sur la connaissance de la structure interne du programme. L'objectif de cette méthode consiste à tester le code du programme (1) en choisissant des données de test à vérifier par exemple chaque déclaration ou chaque chemin dans le code; ou (2) en vérifiant les conditions de sélection et de répétition des structures de contrôle du programme. Parmi les techniques de test structurel, le Data-Flow Testing [Rapps et Weyuker, 1985] qui permet d'analyser comment les variables d'un programme sont liées aux valeurs et comment ces variables sont utilisées. Plus précisément, l'objectif de cette technique est de construire une suite de tests

qui force l'exécution des différentes interactions entre le moment où les variables sont définies et le moment où elles sont utilisées dans le programme.

2.3.2.2 Le test fonctionnel

Le test fonctionnel, ***connu par le test boîte noire*** [Beizer, 1995], est une méthode qui permet de vérifier que les fonctionnalités d'un programme donné correspondent bien à ses spécifications sans faire aucune référence à sa structure interne. L'objectif de cette méthode consiste à générer des tests à partir d'une spécification, les exécuter sur le programme réel et de s'assurer que ce dernier se comporte correctement en comparant les résultats obtenus par le programme à ceux requis dans la spécification. Ci-dessous, nous présentons des exemples de ***méthodes de test de boîte noire***.

Le test de conformité ([Beizer, 1995], [Tretmans, 1992]) est une technique de test permettant de s'assurer que les fonctionnalités mentionnées dans la spécification d'un programme donné ont été bien implémentées dans ce programme. L'objectif du test de conformité est de générer un ensemble de cas de test qui vérifient que le programme satisfait ses spécifications. La question qui se pose: « Est-ce que le programme fait ce qu'il doit faire ? » Le test de conformité est présenté en détail dans la section suivante.

Le test de performance [IEEE, 1990] est une technique permettant de comparer la conformité d'un programme donné aux critères de performance. L'objectif du test de performance est d'évaluer les ressources (telles que le temps d'exécution, le temps de réponse, l'utilisation de la mémoire, etc.) qui sont nécessaires à la réalisation du programme. La question typique qui se pose est: Quelle est la vitesse du programme pour réaliser sa tâche?

Le test de robustesse est une technique définie comme « the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions » [IEEE, 1990]. L'objectif de cette technique est de vérifier le comportement d'un programme donné dans un environnement erroné. La principale question qui se pose est : Comment le programme va réagir si son environnement ne se comporte pas comme prévu?

Le test de fiabilité [Musa, 1975] est une technique permettant de vérifier le bon fonctionnement d'un programmé donné durant une période de temps spécifiée et dans un environnement spécifié. La question qui se pose : durant combien de temps peut-on compter sur le bon fonctionnement du programme donné ?

Dans la pratique, plusieurs types d'observabilité peuvent être utilisés pour générer les tests. Si le code source du logiciel à tester est disponible, nous avons le choix de générer des tests de type boîte blanche ou boîte noire selon l'observabilité que l'on souhaite. Dans le cas où

nous disposons uniquement des interfaces, le test est uniquement de type boîte noire. Comme nous l'avons déjà expliqué, les méthodes de test de logiciels varient également en fonction de la nature du test. Par exemple, au niveau du test fonctionnel, ces méthodes diffèrent si nous parlons de test de conformité, de test de performance, de test de robustesse ou de test de fiabilité.

Dans la partie qui suit, nous nous focalisons sur ***les tests de conformité*** étant donné que nous voulons tester un système dont on ne connaît pas le code. En effet, ***l'objectif de cette thèse est de tester des constituants réactifs dans leur environnement d'interaction***. On ne connaît pas forcément comment ces constituants fonctionnement mais les spécifications nous informent comment ils doivent interagir ensemble. Dans la figure 2.20, on compare les deux cycles suivants : le premier est celui de la mise en œuvre du système ERTMS conformément à la norme 50126 concernant les FDMS, le deuxième est celui d'un cycle de développement de logiciel classique en V (cf. Fig. 2.20). Nos travaux de thèse correspondent, au niveau du cycle à gauche, aux ***tests fonctionnels de sécurité***, ce qui représente au niveau du deuxième cycle à droite, les ***tests d'intégration des constituants***. Ainsi, le premier axe de recherche que nous avons choisi est celui des tests de conformité vérifiant l'intégration des constituants dans leur environnement. Pour résumer cette section, *au niveau des phases de test*, on se place dans les ***tests d'intégration***, au niveau des *types de tests*, on se place dans les ***tests fonctionnels***, et enfin au niveau des *méthodes*, on se place dans les ***tests de conformité***.

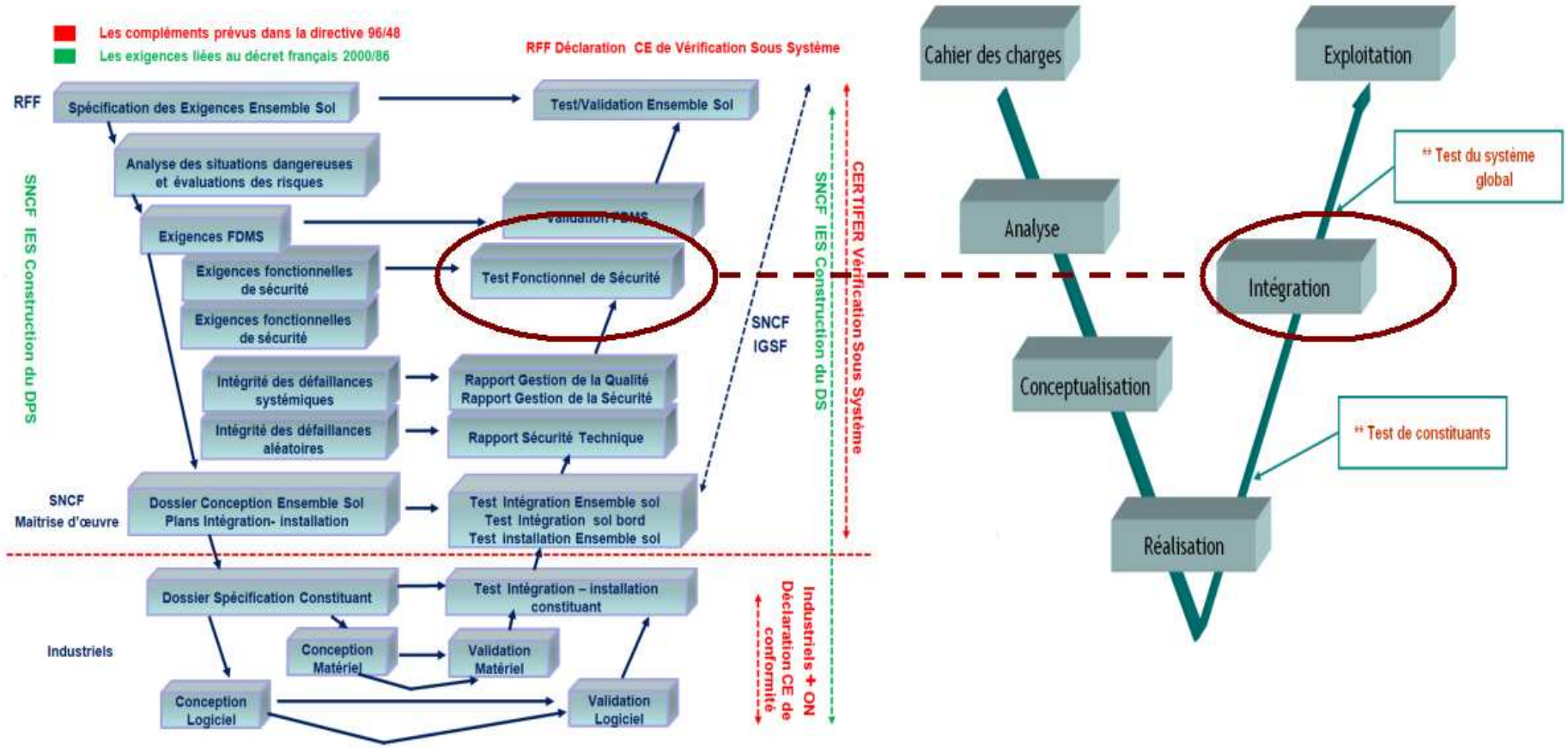


Figure 2.20: Comparaison entre cycle de vie ERTMS et cycle de vie de logiciel classique

2.3.3 Le test de conformité pour la vérification de constituants réactifs

Dans cette partie, nous nous intéressons au test de conformité. Ce type de test a été appliqué dans de nombreux travaux et dans des contextes différents. Il a été formalisé dans des domaines d'application divers et a même été standardisé dans le cadre des protocoles de communication [OSI, 1993]. Comme nous l'avons déjà mentionné, il s'agit d'un test fonctionnel, permettant de vérifier la conformité d'une implantation dont le code est inconnu par rapport à la description des comportements prévue dans la spécification; c'est d'ailleurs le principe des tests de type boîte noire. Ce type de test est défini comme une technique de validation par expérimentation (cf. Fig.2.21) où il est nécessaire de formaliser les différents concepts liés à la conformité : la ***spécification***, les ***interactions*** *entre l'implantation à tester et le testeur*; et *la* ***relation de conformité***. Cette relation de conformité est définie entre le modèle du constituant (implantation) à vérifier et le modèle de la spécification (relation *conforme à ?* sur la figure 2. 21). Les cas de test générés sont issus de la spécification et sont ensuite exécutés sur le constituant à l'aide du testeur afin de produire les verdicts. Les différentes interactions entre le constituant et le testeur dépendent du pouvoir de contrôle du testeur.

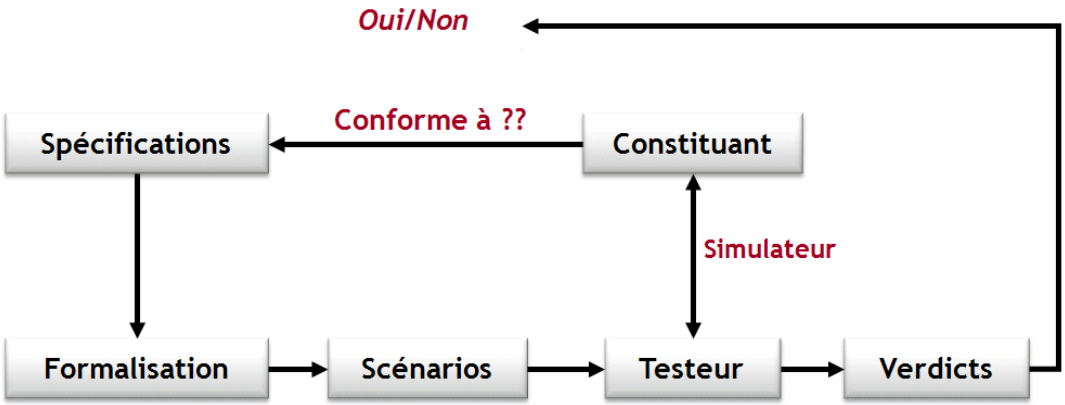


Figure 2.21: Le principe général du test de conformité

2.3.3.1 Spécification, Implantation & relation de conformité

La spécification d'un système réactif est une description formelle ou semi-formelle de tous les comportements qui fixent les propriétés du système. En général, la spécification est développée à partir des exigences des utilisateurs et exprimée en langage naturel ou bien en langages de description spécifique comme LOTOS [Bolognesi & Brinksma, 1988] ou SDL [ITU, 1999]. Elle constitue l'hypothèse de construction pour la génération automatique des cas de test. Sa représentation doit être alors suffisamment exhaustive, précise et non ambigüe. La spécification peut être représentée par des langages de spécification tels que : la logique temporelle [Pnueli, 1986], les machines d'états finies étendues [Petrenko, 2001], les systèmes de transitions [Tretmans, 1992].

L'implantation à tester représente généralement un programme réactif qui interagit continûment avec son environnement. Cette implantation réelle à tester, qu'elle soit logicielle ou matérielle, est sous la forme d'une boîte noire. Ainsi, lors de l'exécution des cas de tests, seul le comportement des interactions est visible. Cependant, afin d'établir la conformité de l'implantation par rapport à la spécification, des modèles formels doivent être utilisés.

Quant à la **relation de conformité** entre l'implantation sous test et la spécification, elle définit exactement les implantations qui sont conformes à la spécification de référence ([Tretmans, 1996], [Phalippou, 1994]). Cette relation est choisie de manière arbitraire suite à un compromis entre l'équipe ayant spécifié le système et celle effectuant les campagnes de test. Dans ses travaux [Phalippou, 1994], Phalippou a défini un ensemble de relations de conformité se différenciant sur la granularité de l'équivalence entre le modèle et l'implantation à tester. Une de ces relations est celle qui confirme que toute sortie de l'implantation en réponse à une entrée donnée a été prévue par la spécification. Dans la section 2.4, nous décrivons les méthodes de génération de tests de conformité et nous détaillons les relations de conformité.

2.3.3.2 Le processus de test de conformité selon la norme ISO9646

La norme ISO9646, OSI Conformance Testing Methodology and framework [ISO, 1994], a été initialement développée pour la validation des protocoles OSI. Elle standardise le test de conformité en définissant un cadre méthodologique composé de trois phases :

(1) la dérivation des suites de test abstraites.

(2) la transformation des suites de test abstraites en suites de test exécutables, il s'agit de l'implémentation,

(3) l'exécution des tests sur l'implantation sous test et l'analyse des résultats obtenus afin d'établir les verdicts concernant la conformité.

Concernant la **première phase**, il s'agit de **dériver les suites de test abstraites** à partir de la spécification. Une suite de test contient un ensemble de cas de test à exécuter sur l'implantation sous test. Tout d'abord, à partir des exigences de conformité définies dans la spécification, les objectifs de test sont définis. Un **objectif de test** décrit des comportements d'un système donné devant être testé et sert à sélectionner une partie de la spécification du système pour laquelle un cas de test sera généré. Un cas de test générique est représenté par une séquence contenant les instructions de haut niveau nécessaires pour vérifier l'objectif de test. Dans la norme ISO9646, il est recommandé de construire un cas de test générique par objectif de test. Chaque cas de test générique permet de produire un cas de test abstrait en tenant compte de la méthode de test et des contraintes propres à l'environnement de test.

La **seconde phase** est celle de **l'implémentation** permettant de transformer les suites de test abstraites en suites de test exécutables pour une implantation spécifique.

- *La sélection des tests* : constitue la première étape de cette deuxième phase. Elle consiste à sélectionner des tests parmi l'ensemble des suites de test abstraites obtenues. Les tests éliminés sont ceux qui correspondent à des comportements non implémentés pour l'implantation sous test. Les tests pertinents sont choisis en fonction du PICS (Protocol Implementation Conformance Statements). Il permet de lister l'ensemble des options d'implémentation d'un protocole spécifique car la plupart des standards de protocoles de communication ne précisent pas les options d'implémentation. Ainsi le testeur d'un protocole particulier saura quelles options doivent être testées. Les restrictions sur la sélection des options est donnée par les exigences de conformité statiques du standard. Elles définissent par exemple les exigences concernant les capacités minimales qu'une implantation doive fournir.
- *L'implémentation des tests* : constitue la deuxième étape de cette phase. Elle permet de construire les suites de test exécutables. Les PICS contiennent des informations dépendantes du protocole. Ceci n'est pas suffisant pour cette deuxième étape, il est en effet nécessaire de tenir compte des particularités de l'implantation sous test et de son environnement. Le PIXIT (Protocol Implementation eXtra Information for Testing) contient ces informations. Ainsi, suite à l'étape de sélection des tests, les informations contenues dans le PIXIT sont rajoutées afin d'obtenir les tests implémentés.

La ***dernière phase*** de la méthodologie de test de conformité est celle ***de l'exécution*** des tests.

- La première étape de cette dernière phase consiste à vérifier la conformité statique. L'observation des PICS permet de déterminer si les tests sont conformes aux exigences de conformité du standard.
- La seconde étape consiste à appliquer les suites de test exécutables sur l'implantation. Suite à cette exécution de tests, chaque séquence fournit un verdict de conformité qui peut être : «***Pass***» dans le cas d'exécution du test avec succès et la vérification de la propriété définie dans l'objectif de test; «***Fail***» dans le cas où le test prouve que l'implantation n'est pas conforme à la spécification; ***«Inconclusive»*** , uniquement utilisé dans le test se basant sur un objectif de test et signifie qu'il n' y a pas d'erreur détectée qui prouve la non-conformité, mais que la propriété définie dans l'objectif de test n'a pas été satisfaite par l'implantation.
- La troisième étape consiste à combiner les résultats de l'exécution de tous les tests statiques ou dynamiques afin de déterminer le verdict final de conformité. Le verdict « Pass » n'est obtenu que dans le cas où aucun résultat « Fail » n'est produit à partir

des différents tests individuels effectués. Le document où sont rapportés les résultats de chaque test individuel ainsi que le verdict final est le PCRT (Protocol Conformance Test Report).

Format des suites de test dans la norme ISO9646. Dans cette norme, il est recommandé d'utiliser le langage TTCN (Tree and Tabular Combined Notation) pour la notation des suites de test. Ce langage est particulièrement utilisé pour spécifier les tests des systèmes de communication. Comme le montre la figure suivante, le format TTCN d'une suite de test comporte quatre parties : une partie de présentation de la séquence de test («suite overview part»), une partie de déclaration («declaration part»), une partie pour les contraintes («constraints part») et une partie dynamique («dynamic part»).

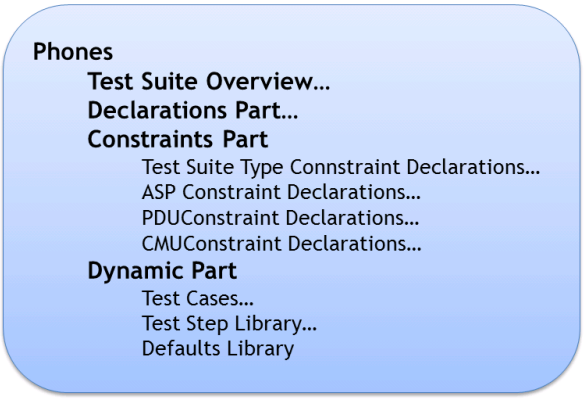


Figure 2.22: Structure TTCN d'une suite de test ©IEC7

Dans la figure 2.22, la partie de **présentation** décrit la séquence de test sous la forme d'un tableau. La partie de **déclaration** est utilisée pour déclarer les types, les variables, les temporisateurs, les points de contrôle et d'observations PCOs (Points of Control and Observation) et les composants de test. Un composant de test peut être un MTC (Main Test Component) ou PTC (Parallel Test Component). La partie de **contraintes** permet de décrire les valeurs envoyées ou bien reçues. Les types de structure ASP (Abstract Service Primitive) et PDU (Protocol Data Unit) (cf. Fig2.22) sont utilisés comme des modèles pour la description des messages envoyés au niveau des PCOs. Un PCO est un concept qui définit un point de connexion avec l'implantation sous test. Le type CM (Coordination Message) décrit un message qui se produit au niveau d'un point de coordination CP (Coordination Point). Le CP est très similaire au PCO, il permet une communication asynchrone entre exactement deux composants de test (par exemple entre deux PTC ou bien entre un PTC et un MTC). Enfin, la partie dynamique (cf. Fig.2.22) permet de décrire les cas de test et les tableaux correspondant aux évènements de test et les verdicts obtenus. Les évènements de test sont représentés par un arbre où chaque branche décrit une séquence d'interaction entre le testeur et l'implantation sous test.

[7] International Engineering Consortium, www.iec.org

Exemple cas de test TTCN. Nous reprenons la figure 2.22 qui décrit la structure TTCN d'une suite de test dans le but de vérifier l'établissement d'une communication téléphonique. Nous considérons l'arbre d'évènement suivant.

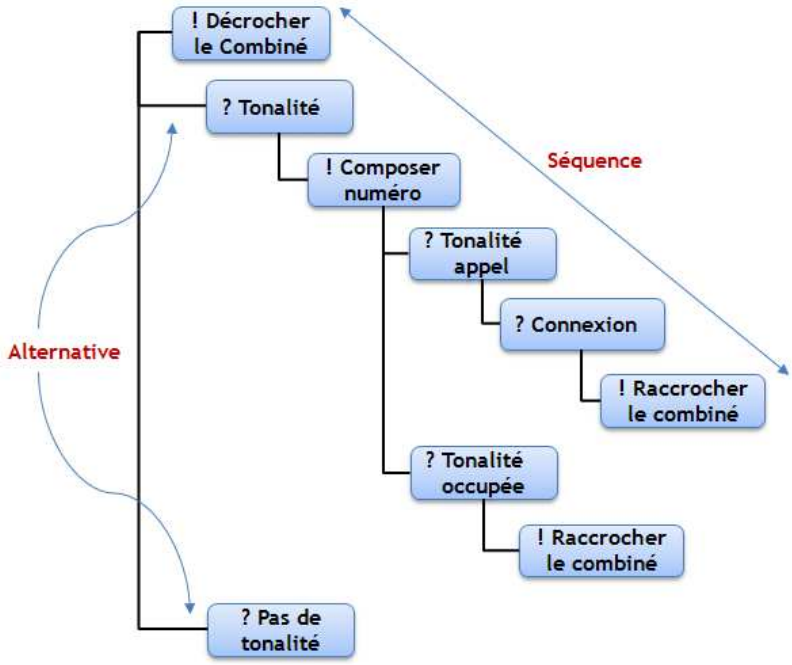


Figure 2.23: Arbre d'évènement d'après IEC8

Dans la figure 2.23, le symbole « ? » définit une entrée au constituant qui est le téléphone et le symbole « ! » définit une sortie du cas de test. Ce dernier (cf. Fig. 2.24) est défini de la manière suivante :

Test Case Dynamic Behavior

Test case name: Connexion basique
Group:
Purpose: Vérifier qu'une communication normale peut être établie
Default:
Comment:

Nr	Label	Behaviour description	Verdict	Comments
1		! Décrocher le Combiné		
2		? Tonalité		
3		! Composer numéro		
4		? Tonalité appel		
5		? Connexion		
6		! Raccrocher le combiné	Pass	Correct Behaviour
7		? Tonalité occupée		
8		! Raccrocher le combiné	Inconclusvie	
9		? Pas de Tonalité	Fail	

Detailed Comments:

Figure 2.24: Le cas de test TTCN correspondant à l'arbre d'évènement de la figure 2.23

[8] International Engineering Consortium, www.iec.org

- ✓ Le cas de test décroche le combiné,
- ✓ Le cas de test vérifie la tonalité,
- ✓ Si la tonalité est entendue (une entrée), le cas de test compose le numéro (sortie),
- ✓ Il vérifie ensuite la tonalité d'appel, si elle est entendue il vérifie la connexion
- ✓ Si la connexion est établie, il raccroche le combiné et attribue le verdict « **Pass** »
- ✓ Sinon dans le cas où une tonalité d'appel occupée est entendue, le cas de test raccroche le combiné et met le verdict « **Inconclusive** ». Ce verdict explique que l'objectif de test (vérification de l'établissement d'une connexion) n'est pas atteint mais qu'une erreur n'a pas été détectée.
- ✓ Enfin, si tout au début, il n' y'avait pas de tonalité, le verdict est mis à « **Fail** » et le cas de test est terminé.

Suite à la présentation du test de conformité selon la norme ISO9646, nous décrivons dans la section suivante les différentes approches et méthodes de génération automatiques de scénarios de test. Nous voulons dire par scénario de test un cas de test. Ces deux termes sont souvent utilisés dans la littérature. Dans la section 2.4.1.2, des définitions formelles des cas de test, de l'objectif de test, de la spécification et de l'implantation sont proposées.

2.4 Les méthodes et les outils de génération de scénarios de test de conformité

Les méthodes manuelles ont longtemps servi à construire des scénarios de test et même à les exécuter. Le principal inconvénient de ces méthodes consiste à commettre des erreurs lors de la construction des tests par les utilisateurs en particulier quand il s'agit d'un grand nombre de tests. Construire des scénarios de tests de manière automatique devient alors un travail indispensable. En effet, plusieurs études ont montré que la génération automatique des scénarios de tests permet de gagner en productivité, en temps, en qualité, et en coût ([Morel, 2000], [Fernandez et *al*, 1996]). Elle présente néanmoins quelques difficultés dont le choix des modèles des entrées et les méthodes de sélections des scénarios de tests pertinents.

Dans le cadre de nos travaux, nous avons uniquement la spécification décrivant les interactions de chaque constituant. C'est à partir de cette spécification que nous voulons générer automatiquement des scénarios de tests pour la validation des constituants ERTMS. Notre hypothèse de départ prend la forme du modèle de spécification des composants. La spécification constitue ainsi notre élément de référence pour la génération de tests. Dans ce

qui suit, nous présentons des méthodes de génération de tests et en particulier nous nous intéressons à la méthode de dérivation de spécification.

2.4.1 Les méthodes de génération de test par dérivation de spécification

Dans le cas de test boîte noire, plusieurs méthodes existent pour la génération de scénarios de test. Les générations symboliques, les générations par contraintes et les **générations par dérivations de spécifications** en sont des exemples. La méthode qui nous intéresse dans le cadre de nos travaux est celle réalisée par dérivation de spécifications. Quant aux deux premières méthodes, nous nous contentons de donner deux exemples de systèmes les utilisant en annexe C (Le système AUTOFOCUS [Pretschner & Lotzbeyer, 2001] et le système Gatel [Marre&Arnould, 2002], [Marre&Arnould, 2004]).

La méthode de génération de tests que nous allons décrire dans la suite est celle par dérivation de spécifications. Deux grandes familles de méthodes existent pour générer ces scenarios de test en fonction des modèles utilisés pour représenter la spécification : la première famille s'appuie sur une modélisation par automates. Quant à la seconde famille, elle est issue des algèbres de processus et produit les scénarios de test à partir d'un système de transitions.

2.4.1.1 Méthodes fondées sur les automates

Cette première famille est la plus ancienne et elle est fondée sur les automates. Elle est issue des méthodes de test de circuits séquentiels et des problèmes de reconnaissance de machines [Moore, 1956]. Elle se base sur une spécification représentée par les machines de Mealy [Gill, 1962].

Nous précisons qu'une machine de Mealy est un automate dans lequel chaque transition est étiquetée par une entrée et une sortie de signal. Pour ce type de modèle, le principe de test consiste à déterminer si les deux machines décrivant respectivement la spécification et l'implantation sont équivalentes. Les différentes techniques de génération de scenarios de test de cette méthode ne sont pas développées dans la suite du document. Dans les travaux de D. Lee et M. Yanakakis [Lee & Yannakakis, 1994] et dans la bibliographie commentée par A. Petrenko [Petrenko, 2001], nous trouvons une synthèse de ces différentes techniques de génération de test. Une variété d'algorithmes de génération de scénarios de test à partir d'automates d'états finis se trouvent également dans les travaux suivants : [Bochmann & Petrenko, 1994], [Naito & Tsunoyama, 1981], et [Chow, 1978].

Définition 2.4.1.1/ **Machine de Mealy** [Prigent, 2003]. Une Machine de Mealy est un t-uplet $(S, I, O, \delta, \omega)$ où :

- S est un ensemble fini d'états,
- I est un alphabet fini d'entrées,
- O est un alphabet fini de sorties,
- $\delta : S \times I \rightarrow S$ est une fonction de transfert,
- $\omega : S \times I \rightarrow O$ est une fonction de sortie.

Une Machine de Mealy est déterministe.

Exemple de Machine de Mealy. Dans la figure 2.25, nous donnons un exemple d'une machine de Mealy. La transition d'un état S_1 vers un état S_2 effectue un entrée i_2 et une sortie o_2. Dans l'état S_2, la machine reste dans le même état après réception de l'entrée i_1 et produit une sortie o_2. Si la machine reçoit l'entrée i_2 dans cet état, alors elle produit la sortie o_1 et retourne dans l'état S_1.

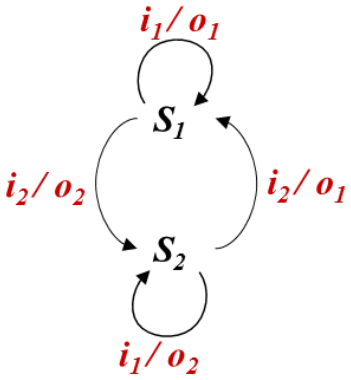


Figure 2.25: Exemple d'une machine de Mealy

Equivalence de deux machines de Mealy [Prigent, 2003]. Deux machines de Mealy M et M' sont équivalentes si pour tout état de M, il existe un état équivalent de M' et inversement. Deux états S et S' sont équivalents si à partir d'un élément i de l'alphabet des entrées, la fonction de sortie de S est égale à la fonction de sortie de S' (ω (S, i) = ω (S', i)).

Test de conformité des machines de Mealy. La conformité consiste à déterminer si la machine de Mealy représentant la spécification (M_S) et la machine de Mealy représentant l'implantation (M_I) sont équivalentes. Les machines M_S et M_I sont équivalentes si aucune des fautes de sortie ou de transfert n'a été détectée. La faute de sortie indique que la sortie produite après une entrée n'est pas celle qui est attendue. La faute de transfert indique que l'état d'arrivée après une transition n'est pas celui qui est attendu. Les tests produits doivent détecter les fautes de sortie et de transfert. L'une des méthodes de génération des séquences de test est la méthode « Tour de transitions » [Naito & Tsunoyama, 1981] qui permet de parcourir toutes les transitions de la machine de Mealy. L'optimisation de cette méthode

consiste à trouver une séquence de test de longueur minimale et qui passe par toutes les transitions de la machine de Mealy.

Avantages et inconvénients [Jéron, 2004]. L'avantage de ce type de méthode est l'exhaustivité. Cependant les algorithmes sont relativement coûteux, ce qui cantonne leur applicabilité à des graphes d'états de taille très modeste (de l'ordre de la centaine d'états et de transitions). L'inconvénient le plus important est celui des hypothèses nécessaires à l'application de ces méthodes portant d'une part sur la spécification mais surtout sur l'implantation. Nous pouvons concevoir que les spécifications et implantations soient déterministes quand elles se rapportent à une entité de protocole. C'est beaucoup moins réaliste si nous considérons un système composé, par exemple dans le cas où la communication est asynchrone entre sous-systèmes. Nous sommes alors confrontés à des situations où plusieurs sorties sont possibles après la même entrée. Un autre inconvénient est dû au modèle des machines de Mealy qui n'est pas le modèle idéal pour représenter la sémantique des langages de spécifications comme SDL [ITU, 1999]. En effet en SDL une transition est souvent constituée d'une entrée suivie de plusieurs sorties ou uniquement de sorties alors que pour les machines de Mealy, une transition comprend une entrée et une sortie. Pour toutes ces raisons, la génération de test basée sur les systèmes de transitions semble la plus appropriée.

2.4.1.2 Méthodes fondées sur les systèmes de transitions

Cette deuxième famille est basée sur les systèmes de transitions à entrées-sorties. Ce type de modèle permet la distinction entre les actions observables (les sorties) et les actions contrôlables (les entrées). Contrairement aux machines de Mealy, chaque transition effectue soit une entrée, soit une sortie. Les modèles restent proches de ceux des machines d'états finis [Petrenko et al, 2003] mais les méthodes de génération sont différentes. Ces modèles sont à la base de l'outil TGV ([Jard & Jeron, 2002], [Jeron & Morel, 1999], [Fernandez et al, 1996]) et de l'outil TorX [Tretmans & Brinksma, 2002] détaillés dans la section suivante (cf. section 2.4.2).

A- les systèmes de transitions étiquetés

Définition 2.4.1.2.a/ **Systèmes de transitions** [Leroux, 2004]. Un système de transitions étiqueté est un quadruplet $(\mathbf{Q}, \mathbf{q_0}, (\mathbf{A} \cup \{ \boldsymbol{\tau} \}), \rightarrow)$ où :

- $\mathbf{Q}$ est l'ensemble des états du système,
- $\mathbf{q_0} \subseteq \mathbf{Q}$ est l'état initial du système,
- $\mathbf{A}$ est un alphabet fini d'actions observables,
- $\boldsymbol{\tau}$ est utilisé pour définir une action interne non observable
- $\rightarrow \subseteq \mathbf{Q} \times (\mathbf{A} \cup \{\boldsymbol{\tau}\}) \times \mathbf{Q}$ est une relation de transitions étiquetées.

Exemple de système de transitions étiqueté. Considérons le système de transitions étiqueté **S** de la figure 2.26. Il est composé par huit états q_0, $q_1 \ldots q_7$, où q_0 est l'état initial et dix transitions. Chaque transition de S est étiquetée soit par une action observable (pièce, lait, chocolat, café, thé), soit par une action interne non observable ($\boldsymbol{\tau}$).

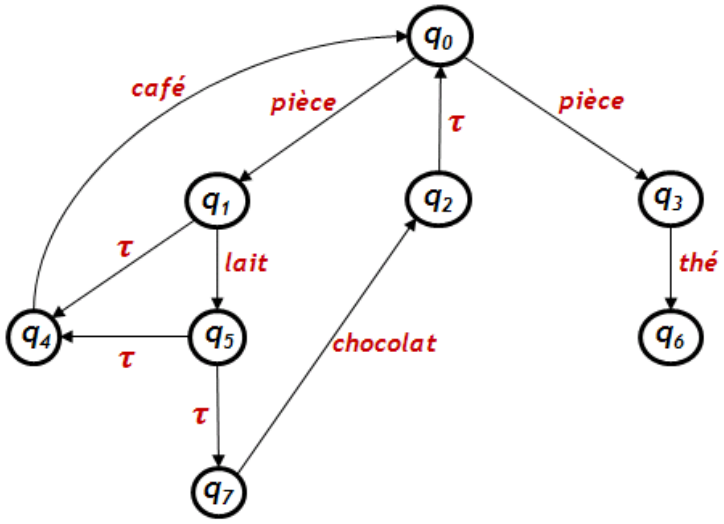


Figure 2.26: Exemple de systèmes de transitions étiqueté, d'après [Leroux, 2004]

Le système de la figure 2.26 décrit une machine à café qui délivre du café, du chocolat ou du thé à l'utilisateur. Le comportement de la machine à café peut être décrit de la manière suivante : suite à la réception d'une pièce, la machine passe à l'état q_1 ou q_3. Si la machine a décidé d'aller à l'état q_3, alors elle délivre du thé à l'utilisateur et son fonctionnement s'arrête à l'état q_6. Si la machine est à l'état q_1 après réception de la pièce, alors il y a deux possibilités de fonctionnement : (1) la machine décide d'aller à l'état q_4 en exécutant une action interne, délivre du café à l'utilisateur et puis revient à l'état initial ; ou (2) l'utilisateur a la possibilité d'ajouter du lait, la machine passe à l'état q_5 pour ensuite exécuter une action interne et délivrer le chocolat à l'utilisateur et finalement revient à l'état initial.

A1- Notations.

Soient **M** un système de transitions sous la forme de $(\mathbf{Q}, \mathbf{q_0}, (\mathbf{A} \cup \{\boldsymbol{\tau}\}), \rightarrow)$ et les données suivantes:

- $\mathbf{Q'} \subseteq \mathbf{Q}$ un sous-ensemble des états de **M** ;
- $\mathbf{A'} \subseteq (\mathbf{A} \cup \{\boldsymbol{\tau}\})$ un sous-ensemble des actions de **M** ;
- **q** et **q'** deux états de **Q** ;
- $\boldsymbol{a}_{(i)} \in \mathbf{A}$ est une action observable de **M** ;
- $\boldsymbol{\mu}_{(i)} \in (\mathbf{A} \cup \{\boldsymbol{\tau}\})$ une action observable ou interne de **M**;
- $\boldsymbol{\sigma} = \boldsymbol{a_1}, \boldsymbol{a_2} \ldots \boldsymbol{a_n} \in (\boldsymbol{A})^*$ est une séquence d'actions observables ;
- $\boldsymbol{\omega} = \boldsymbol{\mu_1}, \boldsymbol{\mu_2} \ldots \boldsymbol{\mu_n} \in (\mathbf{A} \cup \{\boldsymbol{a_r}\})^*$ une séquence d'actions observables et internes ;
- $\boldsymbol{\varepsilon} = \boldsymbol{\tau} \ldots \boldsymbol{\tau} \in (\{\boldsymbol{\tau}\})^*$ est une séquence d'actions internes.

Nous présentons quelques notations correspondant aux systèmes de transitions étiquetées ([Leroux, 2004], [Jéron, 2004]):

Table 2.4: Notations pour les Systèmes de transitions

(1) $q \xrightarrow{\mu} q'$	$\triangleq$	$(q, \mu, q') \in \rightarrow$
1.a) $q \xrightarrow{\mu}$	$\triangleq$	$\exists\, q' \in Q \wedge [q \xrightarrow{\mu} q']$
(2) $q \xrightarrow{\omega} q'$	$\triangleq$	$q \xrightarrow{\mu_1 \dots \mu_k} q'$
		$\exists\, q_0, q_1 \dots q_k \in Q \ \wedge [q = q_0 \xrightarrow{\mu_1} q_1 \xrightarrow{\mu_2} \dots \xrightarrow{\mu_k} q_k = q']$
(2.a) $q \xrightarrow{\omega}$	$\triangleq$	$\exists\, q' \in Q \wedge [q \xrightarrow{\omega} q']$
(3) $q \stackrel{\varepsilon}{\Rightarrow} q'$	$\triangleq$	$[(q = q') \vee (q \xrightarrow{\tau \dots \tau} q')]$
(4) $q \stackrel{a}{\Rightarrow} q'$	$\triangleq$	$\exists\, q_1, q_2 \in Q \ \wedge [q \stackrel{\varepsilon}{\Rightarrow} q_1 \xrightarrow{a} q_2 \stackrel{\varepsilon}{\Rightarrow} q']$
(4.a) $q \not\Rightarrow^{a}$		$\nexists\, q' \in Q \wedge [q \stackrel{a}{\Rightarrow} q']$
(5) $q \stackrel{\sigma}{\Rightarrow} q'$	$\triangleq$	$q \stackrel{a_1 \dots a_k}{\Longrightarrow} q'$
		$\exists\, q_0, q_1 \dots q_k \in Q \ \wedge [q = q_0 \stackrel{a_1}{\Rightarrow} q_1 \stackrel{a_2}{\Rightarrow} \dots \stackrel{a_K}{\Rightarrow} q_k = q']$
(5.a) $q \stackrel{\sigma}{\Rightarrow}$	$\triangleq$	$\exists\, q' \in Q \wedge [q \stackrel{\sigma}{\Rightarrow} q']$
(6) ***Traces*** (M)	$\triangleq$	$\{\boldsymbol{\sigma} \in (\boldsymbol{A})^* \mid q_0 \stackrel{\sigma}{\Rightarrow}\}$
(7) (q ***after*** σ)	$\triangleq$	$\{q' \in Q \mid q \stackrel{\sigma}{\Rightarrow} q'\}$
(7.a) (M ***after*** σ)	$\triangleq$	(q_0 ***after*** σ)
(8) M est déterministe	**Si**	M n'a pas d'actions internes et $\forall\, q \in Q, \sigma \in (A)^*, [\text{card}\,(q$ ***after*** $\sigma) \leq 1]$
(9) ***DTraces*** (M)	$\triangleq$	$\{\boldsymbol{\sigma} \in \boldsymbol{Traces(M)} \mid \exists\, q \in (M$ **after** $\sigma), \forall\, \mathbf{a} \in A$ $[q \not\Rightarrow^{a}]\}$,
(10) Reach (Q')	$\triangleq$	$\{q \in Q \mid \exists \omega \in (A \cup \{\tau\})^*, q' \in Q', \wedge [q \xrightarrow{\omega} q']\}$

Si nous reprenons le système **S** de la machine à café (cf. Fig.2.26), nous pouvons ainsi vérifier que :

Table 2.2: Application des notations du tableau 2.4 sur l'exemple de la figure 2.26

(1) $q_0 \xrightarrow{pièce} q_1$; $q_7 \xrightarrow{chocolat} q_2$; $q_3 \xrightarrow{thé} q_6$; $q_5 \xrightarrow{\tau} q_4$;
(2) $q_1 \xrightarrow{lait.\tau.café} q_0$;
(3) $q_1 \stackrel{\varepsilon}{\Rightarrow} q_4$; $q_2 \stackrel{\varepsilon}{\Rightarrow} q_0$;

(4) $q_5 \xRightarrow{chocolat} q_0$; $q_5 \not\Rightarrow^{thé}$

(5) $q_0 \xRightarrow{pièce.lait.café} q_0$;

(6) S a un nombre infini de traces, par exemple ***Traces*** (M) = { **ε ;** pièce ; pièce.thé ; pièce.café ; pièce.lait.café ; …}

(7) (q_5 **after** ε) = {q_4,q_5,q_7}; (S **after** pièce) = {q_1,q_3,q_4};
(S **after** pièce. chocolat) = ∅;

(8) M est indéterministe étant donné que [card (S **after** pièce)] =3

(9) Les traces suivantes ramènent à des états de blocage:
{pièce.thé ; pièce.café.pièce.thé ; pièce.lait.chocolat.pièce.thé} ;
Toutes ces traces appartiennent à **DTraces**(M).

(10) Les états accessibles à partir de S'= {q_3, q_6} est Reach (S') ;
Reach (S')= {q_3, q_6} ; en effet à partir de l'état q_6, on ne peut atteindre que l'état q_6 à travers une séquence vide d'actions observables. Ensuite à partir de l'état q_3, on peut atteindre l'état q_6 à travers la séquence « thé » et l'état q_3 à travers la séquence vide ;
Si nous considérons S'= {q_5}, Reach (S')= {q_0, q_1, q_2, q_3, q_4, q_5, q_6 q_7}

Remarque. L'exemple que nous venons de présenter peut être considéré comme un système de transitions à entrées-sorties où ***pièce*** et ***lait*** constituent les actions d'entrée ; ***café***, ***chocolat*** et ***thé*** les actions de sortie et τ les actions internes. Les actions d'entrée sont marquées par **« ? »** et les actions de sortie par **« ! »** (cf. paragraphe B).

En se basant sur ces systèmes de transitions étiquetées, la théorie de test s'applique sur une relation de conformité permettant de définir les implantations conformes à la spécification. La relation de conformité entre une implantation et une spécification peut être soit une équivalence soit un pré-ordre [Prigent, 2003]. Si on fait passer exactement les mêmes tests à l'implantation et à la spécification, nous parlons alors d'une relation d'équivalence. Les deux entités sont équivalentes si elles ont passé avec succès les mêmes tests et qu'aucun observateur externe ne peut les distinguer. Par contre, dans le cas où on fait passer à l'implémentation au moins les mêmes tests que la spécification, nous parlons alors d'une relation R de pré-ordre.

Notations.

$\boldsymbol{SPEC}$: Univers de spécifications formelles.

$\boldsymbol{Spec} \in \boldsymbol{SPEC}$: Une spécification qui appartient à l'univers des spécifications.

$\boldsymbol{IMPS}$: Univers des implantations.

$\boldsymbol{Iut} \in \boldsymbol{IMPS}$: Une implantation à tester concrète.

$\boldsymbol{TESTS}$: Univers des cas de tests.

$\boldsymbol{TC} \in \boldsymbol{TESTS}$: Un cas de test.

$\boldsymbol{TS} \subseteq \boldsymbol{TESTS}$: Un ensemble de cas de test appelé suite de test.

$\boldsymbol{ST(A)}$**:** Une classe de systèmes de transitions $\boldsymbol{ST}$ avec un alphabet $\boldsymbol{A}$**.**

$\boldsymbol{ST}$ ne contient pas des séquences infinies d'actions internes**.**

$\boldsymbol{TP}$ **:** Est un objectif de test**.**

A2- Les Pré-ordres de test.

Cette relation a été définie par R. de Nicola and M. Hennessy [de Nicola & Hennessy, 1984]. Une implantation qui peut être modélisée par $\boldsymbol{Iut}$[9] $\in \boldsymbol{ST(A)}$ est en relation de Pré-ordre avec une spécification $\boldsymbol{Spec} \in \boldsymbol{ST(A)}$ si pour un cas de test $\boldsymbol{TC}$[10] $\in \boldsymbol{ST(A)}$, les observations (Traces et blocages) obtenues lors de l'interaction de $\boldsymbol{TC}$ avec l'implantation $\boldsymbol{Iut}$ sont inclues dans les observations obtenues lors de l'interaction du même cas de test $\boldsymbol{TC}$ avec la spécification $\boldsymbol{Spec}$. Une description détaillée des pré-ordres de test se trouve dans [Tretmans, 2002].

A3- Génération de tests de conformité.

Les travaux qui portent sur les relations d'équivalences et de pré-ordre se sont intéressés à l'établissement de critères d'exactitude entre l'implantation à tester modélisée par $\boldsymbol{Iut} \in \boldsymbol{ST(A)}$ et sa spécification formelle $\boldsymbol{Spec} \in \boldsymbol{ST(A)}$ en (1) fournissant un ensemble de cas de test et (2) analysant les exécutions de ces cas de test sur $\boldsymbol{Iut}$ et $\boldsymbol{Spec}$. Cependant le problème de génération de tests diffère de celui de l'établissement des critères d'exactitude. Il peut être formulé de la manière suivante : pour un critère d'exactitude donné et pour une spécification, nous devons générer une suite de tests $\boldsymbol{TS}$[11] $\subseteq \boldsymbol{ST(A)}$ qui sont capables de distinguer une implantation correcte et une implantation incorrecte en se basant sur les observations. Le problème de génération de tests a été initialement étudié par E. Brinksma [Brinksma, 1988] à la fin des années 80. Dans son travail, il présente une méthode qui, à partir d'une spécification donnée, génère un ensemble de cas de test permettant de distinguer entre les implantations correctes et les implantations incorrectes en respectant une relation de conformité $\boldsymbol{Conf}$.

[9] Provient de la terminologie anglaise « Implementation Under Test »
[10] Provient de la terminologie anglaise « Test Case »
[11] Provient de la terminologie anglaise « Test Suite »

La relation ***Conf*** (qui n'est ni une équivalence, ni une relation de pré-ordre) est une modification de la définition précédente des pré-ordres de test dans le sens où : (1) les interactions entre un cas de test et une implantation (ou une spécification) sont modélisées à travers une composition parallèle [Leroux, 2004] et (2) et les traces de la spécification sont uniquement considérées.
La composition parallèle entre deux systèmes de transitions permet de modéliser les interactions entre deux systèmes différents [Leroux, 2004]. Elle est utilisée dans l'exécution du test sur l'implantation (ou la spécification). L'opération de composition parallèle permet à chaque système d'exécuter indépendamment ses actions internes et impose une synchronisation sur les actions d'entrée et de sortie partagées (cette opération est décrite de manière détaillée dans les travaux de [Leroux, 2004] dans le cas de systèmes de transitions étiquetées, de systèmes de transitions étiquetées à entrées sorties et de systèmes de transitions symboliques à entrées sorties).

La relation ***Conf*** est bien adaptée au test étant donné qu'elle limite toutes les observations possibles aux traces d'une spécification. Elle permet de tester uniquement si l'implantation fait ce qui doit être fait. Ceci simplifie la tâche du test car dans ce cas nous ne faisons pas attention aux comportements qui ne sont pas spécifiés.

B- Méthode de génération de test à la volée

Nous présentons dans cette section l'approche de génération de tests à la volée ([Jard & Jéron 2002], [Jeron & Morel, 1999], [Fernandez et al, 1996]) permettant de construire des cas de test à partir de la **spécification** et des **objectifs de test**. L'outil **TGV** (Test Generation with Verification technology) présenté plus tard dans la section 2.4.2.2 se base sur cette méthode.

Définition 2.4.1.2.b/ **Systèmes de transitions à entrées sorties IOLTS**. Un système de transitions étiqueté à entrées sorties est un quadruplet $(Q, q_0, (A \cup \{\tau\}), \rightarrow)$ où :

- Q est l'ensemble des états du système,
- $q_0 \subseteq Q$ est l'état initial du système,
- A est un alphabet fini d'actions partitionné en deux ensembles disjoints :
  - $A = A_E \cup A_S$ où
  - A_E est l'alphabet d'entrées
  - A_S est l'alphabet de sorties
- τ est utilisé pour définir une action interne non observable
- $\rightarrow \subseteq Q \times (A \cup \{\tau\}) \times Q$ est une relation de transition.

Toutes les définitions que nous avons présentées dans la section 2.4.1.2 au niveau du tableau 2.4 sont applicables aux systèmes IOLTS.

B1- Modèle de spécification.

Nous supposerons donc que les comportements de la spécification sont modélisés par un système S de transitions étiquetées à entrées sorties IOLTS (Input Output Labelled Transition System). Nous supposons que S est non-divergent, c'est-à-dire qu'il n'a pas de séquence infinie d'actions internes passant par une infinité d'états distincts.

B2- Modèle d'implantation.

L'implantation (logicielle ou matérielle) à **tester est de type boîte noire**. Elle ne constitue pas un objet formel, mathématique. Cependant, pour pouvoir raisonner formellement sur la conformité d'une implantation vis à vis d'une spécification, nous devons nous baser sur des modèles formels, seule possibilité pour définir une relation mathématique entre implantation et spécification. On prend comme hypothèse que les comportements possibles de l'implantation sont modélisables par un IOLTS.

B3- Comportements observables.

Les tests observent dans la pratique les réactions d'un système à des stimuli du testeur. Il s'agit de comparer les traces d'un système représentant les séquences d'actions observables au comportement attendu, décrits par les traces de la spécification. L'ensemble des traces de la spécification peut être caractérisé par un IOLTS déterministe (cf. définitions du tableau 2.4 pour toutes les notations utilisées).

> **Déterminisation** [Jéron, 2004]. Pour tout IOLTS M**=** (**Q**, $\mathbf{q_0}$, (**A** $\cup\{\boldsymbol{\tau}\}$), $\rightarrow$), nous pouvons définir un IOLTS **det(**M**)** déterministe et de même traces que M : ***Traces*** (M)= ***Traces*** **(det(**M**)).** Nous rappelons que l'ensemble **A** est l'ensemble des actions visibles (observables).
>
> $$\det(M) = (2^Q, q_0 \boldsymbol{after} \varepsilon, A, \rightarrow_{det})$$
> $$\text{Où } P \overset{a}{\rightarrow}_{det} P' \Leftrightarrow P, P' \in 2^Q, a \in A \text{ et } P' = P \boldsymbol{after} a.$$

Les états de **det(**M**)** sont appelés méta-états et représentent des parties de **Q**. L'état initial $q_0 \boldsymbol{after} \varepsilon$ représente l'ensemble des états accessibles depuis q_0 par des actions internes (cf. tableau 2.4). Il suffit alors de considérer les méta-états accessibles depuis $q_0 \boldsymbol{after} \varepsilon$. Le principe de déterminisation est décrit dans l'exemple de la figure 2.27 développé dans les travaux d'HDR de T. Jéron [Jéron, 2004].

B3.1- Blocages. En plus de l'observation des traces d'un système, les tests permettent la détection des blocages (quiescence). Un temporisateur (timer) est utilisé pour attendre une réponse de l'implantation. Dans le cas où la valeur du temporisateur est suffisamment grande, l'expiration du temporisateur est assimilée à un blocage de l'implantation. Pour les systèmes de transitions à entrées sorties IOLTS, trois types de blocages peuvent être considérés :

- ✓ Le blocage de sortie (ou outputlock) : le système est bloqué en attente d'une entrée provenant de l'environnement (états 1 et 2 de l'exemple de la figure 2.27),
- ✓ Le blocage complet (ou deadlock) : le système ne peut plus évoluer (état 4 de l'exemple de la figure)
- ✓ Le blocage vivant (ou livelock) : le système diverge par une suite infinie d'actions internes, (états 3 et 6 de l'exemple de la figure).

Ainsi, pour un IOLTS M, nous considérons $Outputlock(M)$ l'ensemble de ses états d'outputlock, $deallock(M)$ l'ensemble de ses états de deadlock, $livelock(M)$, l'ensemble de ses états de livelock et nous définissons :

$$\boldsymbol{quiescent}\,(\boldsymbol{M}) = \boldsymbol{Outputlock}(\boldsymbol{M}) \cup \boldsymbol{deallock}(\boldsymbol{M}) \cup \boldsymbol{livelock}(\boldsymbol{M}).$$

Le blocage dans la spécification ne représente pas forcément une erreur de celle-ci. Ainsi, le test appliqué à l'implantation doit pouvoir différencier le blocage qui existait dans la spécification (blocage valide de l'implantation) de celui qui n'est pas valide. Lors de la construction des cas de test, il faut alors pouvoir détecter les blocages définies par la spécification. Cependant, dans la figure 2.27 qui illustre le principe de déterminisation, les blocages ne sont pas préservés. En effet, le regroupement de certains états dans des méta-états de **det(**S) permet d'éliminer les blocages. Par exemple le regroupement de l'état 4 dans le méta-état {4,5} fait disparaître le deadlock de l'état 4 étant donné que l'état 5 ne possède pas de deadlock. De même, le regroupement des états 1 et 2 dans le méta-état {0,1,2} fait disparaître les blocages de sortie étant donné que l'état 0 possède une sortie. Quant aux blocages livelock, ils disparaissent toujours lors de la déterminisation car les actions internes sont éliminées.

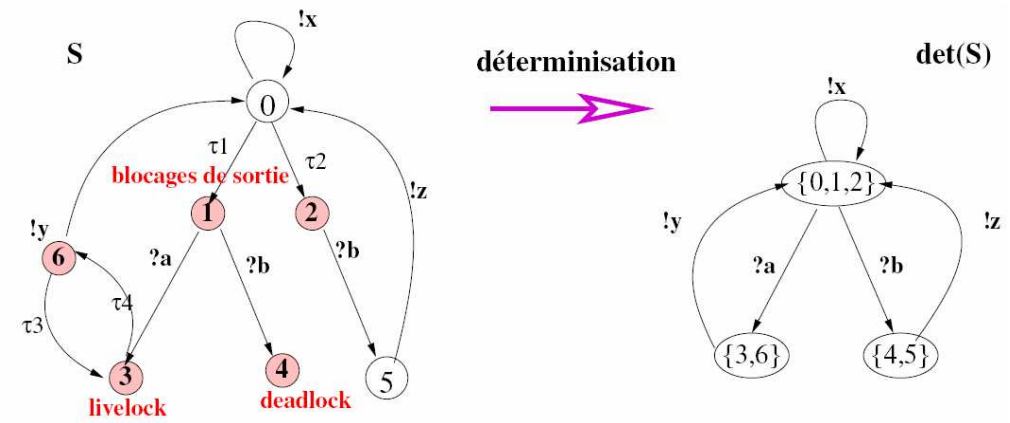


Figure 2.27: Déterminisation et perte des blocages [Jéron, 2004]

La conservation des blocages est importante pour la phase de test. Ainsi, avant de construire **det(**S), il faut modifier l'IOLTS de la spécification. Il faut en effet rajouter à chaque état de blocage de la spécification une boucle étiquetée par une nouvelle action $\boldsymbol{\delta}$. Cette action est classée comme une sortie de la spécification, puisqu'elle est observable mais non contrôlable par l'environnement et elle est notée par $!\boldsymbol{\delta}$ (cf. Fig.2.28). L'IOLTS résultant de cet ajout est

appelé automate de suspension. Pour un cas de test, cette nouvelle action est considérée comme une action d'entrée correspondant à l'expiration d'un temporisateur. Elle est notée ? $\boldsymbol{\delta}$.

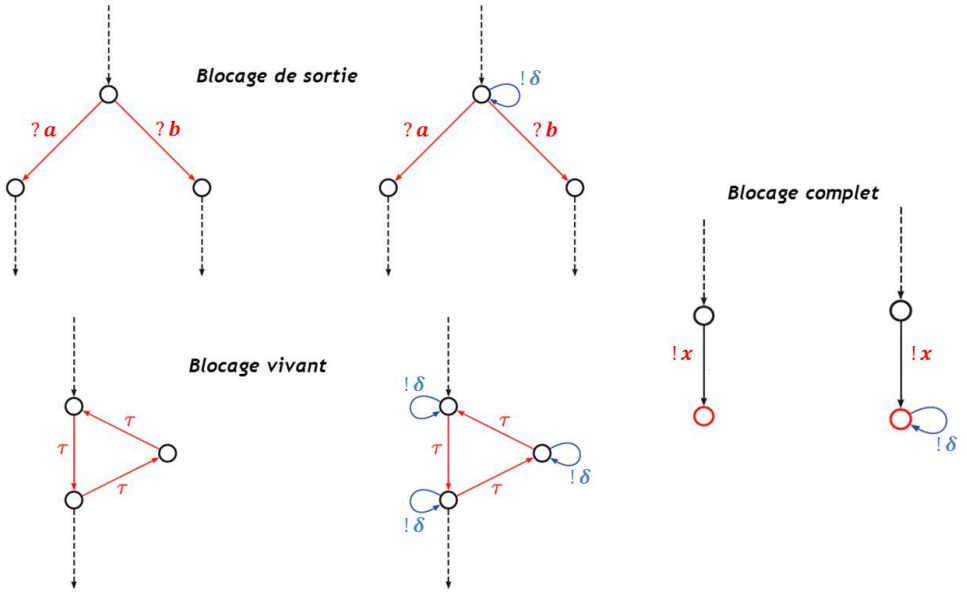


Figure 2.28: Les blocages possibles dans un IOLTS

B3.2- Automate de suspension [Jéron, 2004]. Soit un IOLTS M = $(\mathbf{Q}, \mathbf{q_0}, (\mathbf{A} \cup \{\boldsymbol{\tau}\}), \rightarrow)$. L'automate de suspension de M est l'IOLTS $\Delta(M) = (\mathbf{Q}, \mathbf{q_{0\Delta}}, (\mathbf{A}_\Delta \cup \{\boldsymbol{\tau}\}), \longrightarrow_\Delta)$ tel que $\boldsymbol{A_\Delta} = \boldsymbol{A} \cup \{\boldsymbol{\delta}\}$ avec $\delta \in \mathbf{A_S}$ (δ *est une sortie*) et $\longrightarrow_\Delta$ est définie par:

$$\mathbf{R}_\Delta = \boldsymbol{R} \cup \{q \xrightarrow{\delta} q | q \in \boldsymbol{quiescent}\ (\boldsymbol{M})\}$$ (cf. quiescent (M) plus haut).

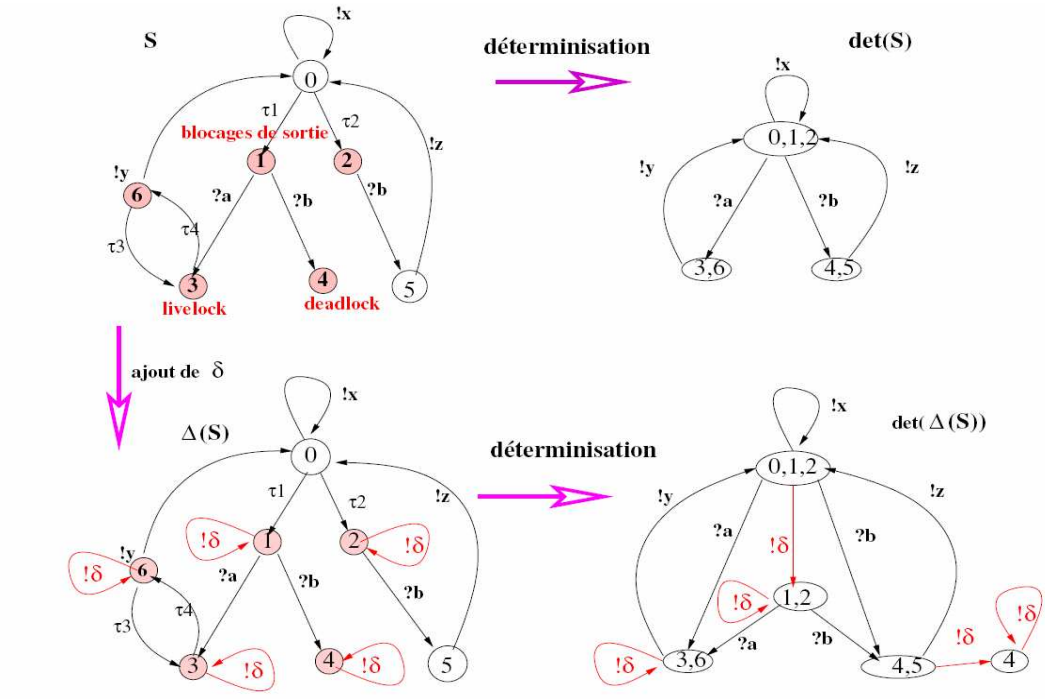


Figure 2.29: Déterminisation après construction de l'automate de suspension [Jéron, 2004]

Dans la figue 2.29, nous montrons la différence entre les deux processus de déterminisation avant et après construction de l'automate de suspension. En effet, la déterminisation de **S** produit **det(S)** mais perd les blocages présents dans l'IOLTS S. Ensuite, la construction de l'automate de suspension $\Delta(S)$ permet de conserver ces blocages. La déterminisation de $\Delta(S)$ produit **det**($\Delta(S)$).

B3.3- Traces suspendues. Les traces suspendues d'un IOLTS M notées ***STraces*(M)** sont les traces (cf. définition dans tableau 2.4) de son automate de suspension $\Delta(M)$ sachant que l'action δ est considérée comme une action observable.

B4- Génération de test à base d'objectifs de test

B4.1- Objectif de test. Comme nous l'avons précédemment défini, un objectif de test ***TP***[12] constitue une description abstraite d'un sous-ensemble de la spécification ***Spec*** permettant de choisir le comportement à tester et de réduire par conséquent l'exploration de la spécification. Un des apports principaux de l'outil TGV a été de formaliser la notion d'objectif de test. Il est modélisé par un automate d'états finis (des IOLTS étendus). Les états finaux de cet automate sont des états d'acceptation ou des états de refus. Les états finaux sont désignés par l'étiquette prédéfinie **« Accept »** ou **« Refuse »**. L'objectif de test doit au moins avoir un état d'acceptation et son utilisation est bénéfique dans le cas d'une spécification de taille importante où il est impossible de tester l'ensemble de l'implantation. Les objectifs de test décrivent les interactions entre le système et son environnement et présentent alors des traces de S ou de $\Delta(S)$. Formellement, ce sont des langages de traces sur les alphabets ($\boldsymbol{A}$) *ou* ($\boldsymbol{A} \cup \{\boldsymbol{\delta}\}$).

Définition formelle [Jéron, 2004]. Un objectif de test est un IOLTS $\boldsymbol{TP} = (\mathbf{Q_{TP}}, \mathbf{q_{0_{TP}}}, \mathbf{A_{TP}}, \rightarrow_{\mathbf{TP}})$ déterministe et complet muni de deux ensembles d'états puits disjoints $\mathrm{Accept_{TP}} \subseteq \mathrm{Q_{TP}}$ et $\mathrm{Refuse_{TP}} \subseteq \mathrm{Q_{TP}}$ et dont l'alphabet consiste en les actions observables et les blocages de la spécification ($\mathbf{A_{TP}} = \mathbf{A} \cup \{\boldsymbol{\delta}\}$). Complet signifie que dans tout état, toute action est tirable ($\forall\, q \in \mathrm{Q_{TP}}, \forall\, a \in A_{TP}, q \xrightarrow{\boldsymbol{a}}_{\boldsymbol{TP}}$). Un état q est un puits s'il boucle pour toute action ($\forall\, a \in A_{TP}, q \xrightarrow{a}_{TP} q$).

Abstraction dans les objectifs de test. Comme nous l'avons déjà mentionné, les objectifs de test constituent une description abstraite des comportements de la spécification. Dans la définition formelle d'un objectif de test, nous l'avons présenté comme un automate complet. Ceci peut être contradictoire avec la notion d'abstraction. Afin de résoudre ce problème et rendre un automate complet, l'outil TGV dispose d'un opérateur de complémentation d'un ensemble d'actions défini par une étiquette spéciale notée "*". Cette étiquette associée à une

[12] Provient de la terminologie anglaise « Test Purpose ».

transition $q \xrightarrow{*} q'$ signifie que pour tout $\boldsymbol{a}$ n'étiquetant pas explicitement la transition d'origine $\boldsymbol{q}$, nous avons $q \xrightarrow{a} q'$. Par défaut, un objectif de test incomplet est complété implicitement par une boucle dans le même état étiquetée par "*".

Exemple. Soit $\mathbf{det}(\Delta(S))$ un IOLTS représentant les comportements d'une spécification (partie gauche de la figure 2.30), nous voulons construire un objectif de test qui permet de tester le système dans les comportements où il exhibe un blocage $!\boldsymbol{\delta}$ et plus tard une émission $!\boldsymbol{y}$ mais sans recevoir $?\boldsymbol{a}$ ni émettre $!\boldsymbol{y}$ avant $!\boldsymbol{\delta}$. L'automate $\boldsymbol{TP}$ représentant l'objectif de test est à droite de la figure 2.30.

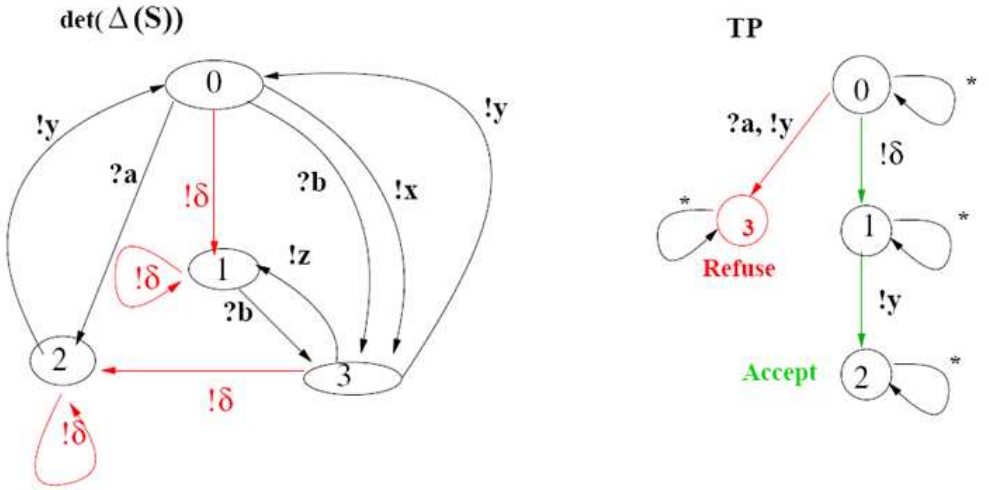


Figure 2.30: Objectif de test [Jéron, 2004]

B4.2- Cas de test [Jéron, 2004]. Un cas de test est un IOLTS $\boldsymbol{TC} = (\mathbf{Q_{TC}}, \mathbf{q_{0_{TC}}}, \mathbf{A_{TC}}, \rightarrow_{\mathbf{TC}})$ muni de trois sous-ensembles distincts d'états sans successeur Pass, Fail et Inconclusive de $\mathbf{Q_{TC}}$. L'alphabet de $\boldsymbol{TC}$ est $\mathbf{A_{TC}} = \mathbf{A_{TC_E}} \cup \mathbf{A_{TC_S}}$ avec $\mathbf{A_{TC_S}} \subseteq \mathbf{A_E}$ ($\mathbf{A_E}$ représente l'alphabet d'entrées de la spécification et $\boldsymbol{TC}$ n'émet que des entrées de la spécification) et $\mathbf{A_{TC_E}} = \mathbf{A_{Iut_S}}$ ($\mathbf{A_{Iut_S}}$ représente l'alphabet de sorties de l'implantation, TC doit prévoir la réception de toute sortie ou blocage de de l'implantation).

Le verdict « Pass » signifie qu'aucune erreur n'a été détectée par le testeur dans l'état atteint par une transition donnée. Quant au verdict « Inconclusive », il se trouve sur une transition qui correspond à une sortie possible de la spécification mais qui mène à un état non satisfaisant pour l'objectif de test. Les transitions « Fail » sont implicites (à partir de chaque état, une action inconnue mène à un état « Fail »).

B4.3- Algorithme de génération de test.

L'algorithme de génération de test est basé sur des objectifs de test externes. Les entrées de cet algorithme que nous appelons $\boldsymbol{Algo\_Gen_{TP}}$ sont la spécification $\boldsymbol{S}$ et un objectif de test $\boldsymbol{TP}$. L'algorithme nous permet d'obtenir en sortie un cas de test $\boldsymbol{TC} = \boldsymbol{Algo\_Gen_{TP}}(\boldsymbol{S}, \boldsymbol{TP})$.

TC focalise le test sur les comportements visibles décrits par l'objectif de test. L'algorithme est composé par un enchainement des opérations suivantes :

- ✓ Calcul de l'automate de suspension $\Delta(S)$ (cf. B3.2)
- ✓ Déterminisation de $\Delta(S)$ afin d'obtenir les comportements visibles ***det*** $(\Delta(S))$ (cf. B3)
- ✓ Extraction des traces acceptées par ***TP*** parmi les traces suspendues à travers la réalisation du produit synchrone entre ***det*** $(\Delta(S))$ et ***TP***.
- ✓ Le résultat du produit synchrone ***PS*** a les mêmes traces que ***det*** $(\Delta(S))$ car ***TP*** est complet mais il peut effectuer un dépliage et étiquette certains états par Accept ou Refuse.
- ✓ Ensuite la phase de sélection permet de construire à partir du produit synchrone un graphe de test complet (**CTG**[13]) qui n'est que le sous-IOLTS contenant les traces acceptées minimales en ne conservant que les états utiles à une trace acceptée. Dans ce graphe, les entrées et les sorties sont inversées par rapport au ***PS*** afin de considérer le point de vue du testeur. Le CTG contient l'ensemble des tests. Ainsi, extraire les cas de test à partir de ce graphe consiste à résoudre les conflits de contrôlabilité. Le CTG possède toutes les propriétés d'un cas de test sauf la contrôlabilité. Nous pouvons en effet à partir d'un état du CTG avoir le choix entre plusieurs sorties ou entre sorties et entrées, ce qui n'est pas autorisé dans un cas de test **TC**.

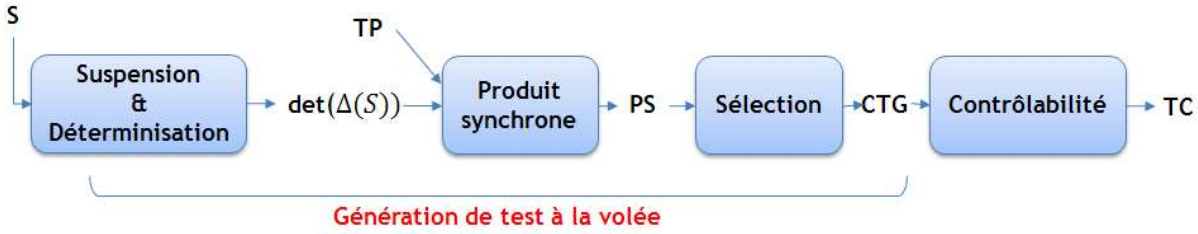


Figure 2.31: Algorithme de génération de tests à la volée

B4.4- Produit synchrone. Soit S un IOLTS et son automate suspendu **det**$(\Delta(S))$= $(Q_{det}, q_{0_{det}}, A \cup \{\delta\}, \rightarrow_{det})$, et TP = $(Q_{TP}, q_{0_{TP}}, A \cup \{\delta\}, \rightarrow_{TP})$ un IOLTS déterministe et complet sur $A \cup \{\delta\}$ et muni des ensembles d'états puits $Acccept_{TP}$ et $Refuse_{TP}$. Le produit synchrone ***det*** $(\Delta(S)) \times TP$ est l'IOTLS PS_{Vis}= $(Q_{Vis}, q_{0_{Vis}}, A_{Vis}, \rightarrow_{Vis})$, muni de deux ensembles d'états disjoints $Accept_{Vis}$ et $Refuse_{Vis}$ et défini comme par :

- ✓ $Son\ ensemble\ d'états\ Q_{Vis} \triangleq Q_{det} \times Q_{TP}$
- ✓ $son\ alphabet\ A_{Vis} \triangleq A_{Vis_E} \cup A_{Vis_S}\ avec\ A_{Vis_E} \triangleq A_E\ et\ A_{Vis_S} \triangleq A_S \cup \{\delta\}$
- ✓ $L'état\ initial\ q_{0_{Vis}} \triangleq (q_{0_{det}}, q_{0_{TP}})$
- ✓ $La\ relation\ de\ transition \rightarrow_{Vis}\ est\ définir\ par$:

[13] Provient de la terminologie anglaise « Complete Test Graph »

$$(q_{det}, q_{TP}) \xrightarrow{a}_{Vis} (q'_{det}, q'_{TP}) \Leftrightarrow q_{det} \xrightarrow{a}_{det} q'_{det} \wedge q_{TP} \xrightarrow{a}_{TP} q'_{TP}$$

✓ $Accept_{Vis} \triangleq Q_{det} \times Accept_{TP}$ *et* $Refuse_{Vis} \triangleq Q_{det} \times Refuse_{TP}$

Les états du produit synchrone sont composés des états de la spécification S et des états de l'objectif de test TP. Une transition est possible dans un état de ce produit synchrone dans deux cas : soit cette transition est possible dans S et dans TP, soit cette transition est uniquement possible dans la spécification S. Durant le parcours du produit synchrone, plusieurs calculs sont effectués. L'algorithme calcule la consistance entre la spécification et l'objectif de test (au cours d'une phase de descente). Un squelette est synthétisé durant la phase de remontée du graphe. Ce graphe contient des séquences du produit synchrone menant à un état d'acceptation de l'automate. Les transitions du squelette générées sont alors décorées avec le verdict.

Exemple. Reprenons l'exemple de la figure 2.30 présentant **det**$(\Delta(S))$ *et* l'objectif de test afin de construire le produit synchrone dans la figure suivante:

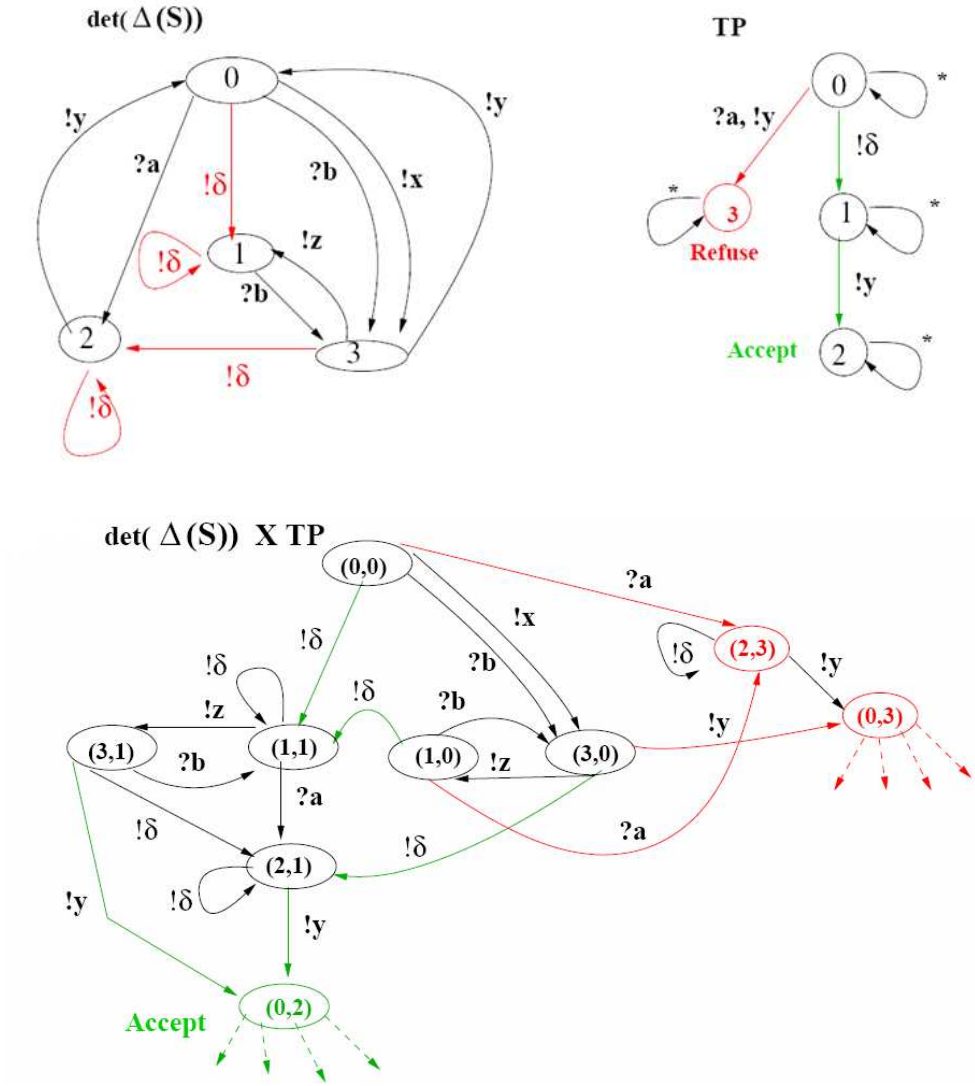


Figure 2.32: Produit synchrone

Remarque. Dans notre approche de génération de test pour la vérification de composants ERTMS, nous nous basons sur la spécification étant donné que nous ignorons le comportement de ces composants. Cette idée explique notre choix du test de conformité de type boîte noire comme méthode de vérification.

Dans la section suivante, nous présentons deux outils de génération automatique de tests de conformité : TorX et TGV.

2.4.2 Les outils utilisés dans la génération de scénarios de test de conformité

2.4.2.1 L'outil TorX

Présentation. TorX [Tretmans & Brinksma, 2002] est un outil résultant d'une collaboration entre l'université de Twente (Tretmans et Brinksma), l'université de Technologie de Eindhoven et les industriels Philips Research Laboratories et KPN Research Groningen dans le cadre du projet "Cote of Resyste" (COnformance TEsting of REactive SYSTEms). L'objectif de ce projet est de définir des méthodologies et des outils vérifiant la conformité des systèmes réactifs par rapport à leurs spécifications formelles. Dans ce contexte, TorX [Tretmans & Belinfante 1999] est l'outil qui permet de générer et exécuter automatiquement des scénarios de test à partir des spécifications définies dans des langages de description formelle comme par exemple: LOTOS [Bolognesi & Brinksma, 1988], PMLP (Protocal Meta Langage Promela) [Basu et *al*, 1998], ou SDL [ITU, 1999]. Ces langages ont une sémantique opérationnelle en termes de systèmes de transitions étiquetées à entrées sorties.

TorX permet principalement de définir une architecture modulaire. Le système à tester est considéré comme un ensemble de composants avec des interfaces bien définies. Cette décomposition en modules indépendants permet de faire évoluer le système à tester en remplaçant un ou plusieurs composants par un composant amélioré ou en ajoutant d'autres relations de communication ou d'exécution de spécifications. TorX dépend des connexions existantes afin de lier les composants entre eux. Il contient plusieurs modules; l'interface utilisée entre ces derniers est assurée par la boîte à outils OPEN/CAESAR offrant ainsi toutes les fonctionnalités requises.

Génération de cas de test. Concernant les cas de test, ils sont construits pendant leur exécution sur l'implantation. Le cas de test $\boldsymbol{TC}$ est généré directement à partir de la spécification $\boldsymbol{Spec}$. Il définit un fonctionnement attendu par l'implantation $\boldsymbol{Iut}$. La génération des cas de test se fait ainsi : lorsqu'un message est émis par l'implantation, le testeur compare l'action envoyée avec les actions possibles de la spécification (cf. Fig.2.33). Si l'action envoyée est correcte, une action à émettre à l'implantation est alors recherchée parmi l'ensemble des actions successeurs de l'action envoyée précédemment. S'il existe

plusieurs possibilités d'actions dans cet ensemble, alors l'outil choisit soit la première action soit une des actions possibles de manière aléatoire [De Vries & Tretmans, 2000]. Concernant la comparaison de l'action envoyée par l'implantation par rapport aux actions possibles de la spécification, deux situations peuvent se présenter. La première est celle de la conformité de l'action envoyée aux attentes de la spécification. Dans ce cas, TorX émet à l'implantation une nouvelle action sélectionnée parmi les successeurs de l'action conforme à la spécification). La deuxième situation est celle de la non-conformité de l'action envoyée aux attentes de la spécification. Dans ce cas, l'exécution des tests est interrompue.

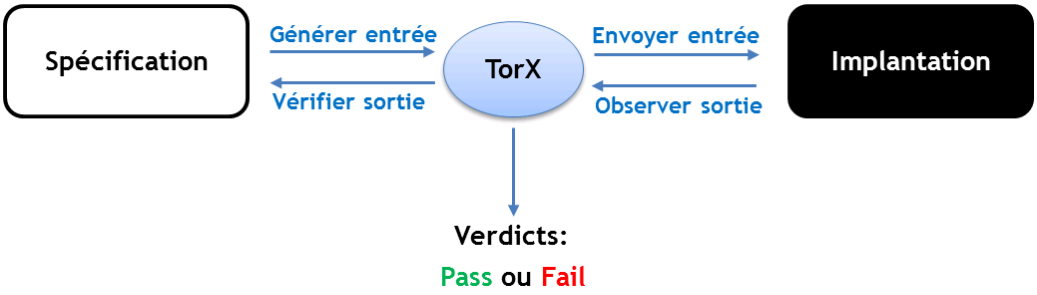


Figure 2.33: Architecture de TorX

Verdicts. Dans cet outil, le concept d'objectif de test est inexistant et deux possibilités de verdicts sont fournies : « **PASS** » et « **FAIL** ». En effet, suite à une interaction conforme et totale vis-à-vis du testeur, un verdict PASS est émis. Quant au verdict FAIL, il est émis suite à une non-conformité de l'action envoyée aux attentes de la spécification. L'algorithme d'exécution du scénario de test ne nécessite pas beaucoup de place mémoire vu que l'état courant et les successeurs sont uniquement stockés de manière temporaire.

2.4.2.2 L'outil TGV

Présentation. TGV (Test Generation with Verification technology) est un outil de génération de tests qui a été initialement produit dans le cadre de l'outil ObjectGéode [Groz et *al*, 1999]. TGV ([Jard & Jéron, 2002], [Jeron & Morel, 1999], [Fernandez et *al*, 1996]) résulte de la contribution commune Vérimag (Grenoble) et l'ancien projet Pampa de l'IRISA (Rennes). Cet outil est devenu une référence dans les communautés industrielles et académiques internationales. Il a été utilisé dans le cadre du projet AGEDIS (cf. Annexe D).

Démarche de génération de test. Tout d'abord, l'outil TGV prend en entrée une spécification formelle (exprimée en LOTOS [Bolognesi & Brinksma, 1988], SDL [ITU, 1999], UML) et un objectif de test utilisé comme un critère de sélection (modélisé par un automate). Ensuite, TGV permet de générer automatiquement des cas de test (cf. Fig.2.34). La principale caractéristique de cet outil est sa méthode de génération à la volée (définie précédemment, cf. section 2.4.1.2). Cette technique permet d'éviter le problème d'explosion combinatoire

[Fernandez et *al*, 1996], au cours de la génération de cas de test pour les systèmes réactif de grande taille. Les cas de test générés par l'outil TGV sont exécutés sur l'implantation afin d'obtenir les verdicts de test. Grâce à une relation de conformité définie dans l'algorithme de TGV, ces cas de test sont valides et ne rejettent pas une implantation conforme à sa spécification. Nous rappelons que dans le cas d'une génération de test basée sur des objectifs de test, trois types de verdicts existent et non pas deux. Le verdict qui se rajoute est « Inconclusive ». Il signifie que, durant l'exécution des tests, l'objectif de test n'a pas été réalisé ; cependant il n'y a pas eu d'erreurs détectées.

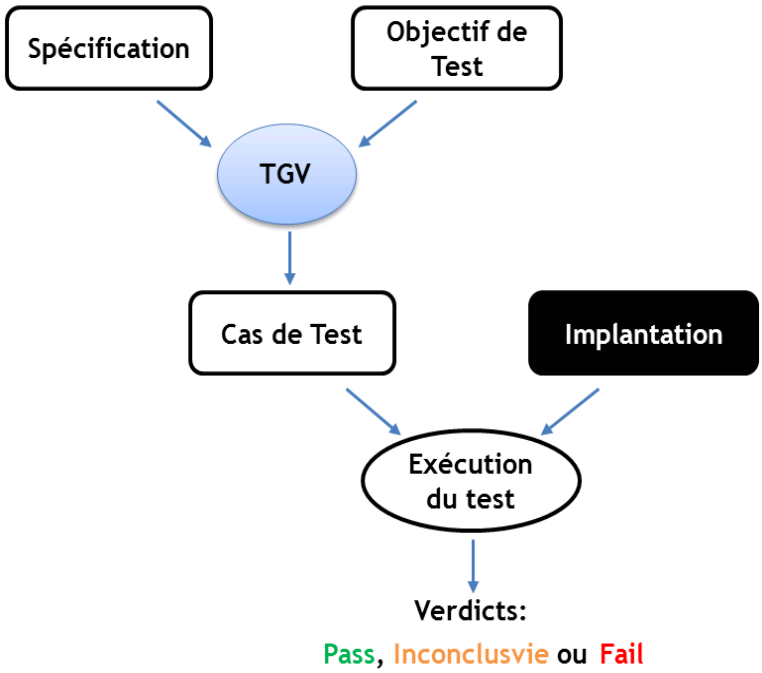


Figure 2.34: Architecture de TGV

Nous savons maintenant que pour générer nos scénarios de test de conformité, nous avons besoin d'un modèle de spécification qui représente les comportements souhaités. Quant aux comportements à tester qui sont issus de la spécification, ils sont aussi représentés par des modèles. Les objectifs de test sont utilisés pour réduire ces modèles à des comportements précis à tester. Comme nous l'avons expliqué auparavant, les objectifs de test sont définis par la norme ISO 9646 [ISO, 1992] pour décrire de manière informelle quel est l'objectif particulier du test à effectuer sur l'implantation. Ils permettent de limiter, au cours de la génération, la production de scénarios de test à certains comportements. Cette phase de génération effectuée entre la spécification et l'objectif de test, se déroule en deux étapes : dans la première étape, est produit un modèle des ensembles des cas de test possibles sous la forme d'un graphe de test, dans la deuxième étape, un cas de test est créé sous la forme d'un sous-graphe issu du graphe précédent. Cette génération de test est ensuite complétée par des verdicts permettant de vérifier, lors de l'exécution d'un cas de test, que l'implantation est

conforme ou non à la spécification. Comme nous l'avons déjà mentionné, la norme ISO9646 propose trois sortes de verdict : Pass (réussi), Fail (échec) et Inconclusive (inconcluant).

Dans le contexte de nos travaux, notre objectif est de générer des cas de test de conformité pour valider un composant. Ces scénarios générés sont supposés alors nous informer sur la conformité de l'implantation à tester vis à vis de la spécification. Dans ce cas il faut qu'un verdict « Fail » implique la non-conformité et que toute non-conformité puisse être détectée par un cas de test. Ceci est défini par le terme de complétude des tests : Une suite de test ST est complète si elle est correcte et exhaustive.

Suite de test correcte. Une suite de test $\boldsymbol{TS} \subseteq \boldsymbol{TESTS}$ est correcte si toutes les implantations $\boldsymbol{Iut} \in \boldsymbol{IMPS}$ qui sont conformes à une spécification $\boldsymbol{Spec} \in \boldsymbol{SPEC}$ passent tous les tests appartenant à cette suite de test :

$$\forall\, \boldsymbol{Iut} \in \boldsymbol{IMPS} \wedge [\,(\,\boldsymbol{Iut}\ conforme\ \text{à}\ \boldsymbol{Spec}) \Rightarrow (\,\boldsymbol{Iut}\ passes\ \boldsymbol{TS})].$$

Cette propriété est réalisable au niveau de la pratique du test mais n'est pas suffisante car les implantations incorrectes et qui ne sont pas conformes à la spécification peuvent passer la suite de test.

Suite de test exhaustive. Une suite de test $\boldsymbol{TS} \subseteq \boldsymbol{TESTS}$ **est** exhaustive si toutes les implantations $\boldsymbol{Iut} \in \boldsymbol{IMPS}$ ayant un verdict Pass lors de l'exécution des tests de $\boldsymbol{TS}$, sont conformes à une spécification $\boldsymbol{Spec} \in \boldsymbol{SPEC}$:

$$\forall\, \boldsymbol{Iut} \in \boldsymbol{IMPS} \wedge [\,(\,\boldsymbol{Iut}\ passes\ \boldsymbol{TS}) \Rightarrow (\,\boldsymbol{Iut}\ conforme\ \text{à}\ \boldsymbol{Spec})].$$

Cette propriété est réalisable au niveau de la théorie comme c'est souvent le cas d'une suite de test infinie, par exemple dans le cas d'une spécification contenant des boucles.

Suite de test complète. Une suite de test $\boldsymbol{TS} \subseteq \boldsymbol{TESTS}$ **est** complète si elle est correcte et exhaustive :

$$\forall\, \boldsymbol{Iut} \in \boldsymbol{IMPS} \wedge [\,(\,\boldsymbol{Iut}\ conforme\ \text{à}\ \boldsymbol{Spec}) \Leftrightarrow (\,\boldsymbol{Iut}\ passes\ \boldsymbol{TS})].$$

Une suite de test complète permet de distinguer exactement les implantations conformes et implantations non conformes.

Les définitions des termes d'exhaustivité, d'exactitude et de complétude diffèrent dans la communauté du test étant donné qu'ils ne sont pas standardisés. Tretmans par exemple, utilise « **sound** » pour définir une suite de test correcte alors que Jéron utilise « **non biais** » pour définir le même concept. Quant à Bernot, Gaudel et Marre [Bernot et al, 1991], ils utilisent « **valide** » pour désigner l'exhaustivité des tests dans la théorie du test algébrique. En test structurel, le terme approprié à l'exactitude et la complétude des tests a un sens

relatif à un critère de couverture. Dans la section suivante, nous définissions la notion de couverture de scénarios de test.

2.5 La couverture des scénarios de test

Dans cette section, notre objectif est de définir la couverture de test et de décrire les différentes formes de couverture qui existent dans la littérature. La couverture correspond à un critère d'arrêt des tests. Ce critère présente une notion importante au cours du processus de développement d'une application et au cours de la phase de validation de cette application. En effet, souvent le nombre de tests générés est infini, d'où la nécessité d'arrêter à un moment donné cette campagne de test. Les types de critères d'arrêt varient selon les priorités. Ils peuvent être de nature financière dans le cas d'épuisement des ressources allouées à la phase de test, ou de nature temporelle dans le cas de fixation de délais précis. Le critère sur lequel nous nous focalisons est la couverture de test. C'est un critère qui repose sur des mesures associées aux tests générés.

Définition 2.5.a/ **Couverture de test** [Bontron, 2005]. La couverture de test d'une suite de test pour une spécification donnée correspond au pourcentage de la spécification exercée par la suite de test suivant un critère donné.

Pour un jeu de test donné, la couverture est le rapport du nombre d'éléments de la spécification (satisfaisant le critère de couverture) qui sont couverts sur le nombre d'éléments qui constituent la spécification et qui satisfont également le critère de couverture. Dans le cas d'une spécification contenant des états non atteignables, obtenir une couverture de 100% est irréalisable. Selon le critère choisi, la couverture d'une même suite de test pour une spécification donnée peut être variable.

Définition 2.5.b/ **Critère de couverture**. Le critère de couverture spécifie les comportements à tester. Dans le cas où il est impossible de générer et exécuter une suite de test couvrant l'ensemble des comportements d'un système, le critère de couverture permet de déterminer l'ensemble des comportements couverts avant la génération des tests.

Dans ce qui suit, nous présentons les différents types de couverture présents dans la littérature. Nous nous intéressons aux couvertures applicables au test fonctionnel étant donné que c'est le type de test que nous pratiquons. Nous considérons également des spécifications formelles sous la forme de systèmes de transitions finis puisque notre objectif est de générer des scénarios de test de conformité à partir d'une spécification formelle. Pour ces systèmes de transitions, plusieurs sortes de critères de couverture [Chow, 1978] existent : la couverture des états, la couverture des branches ou bien la couverture des paires de

branches. Nous présentons brièvement chacun de ces critères ainsi que d'autres critères moins classiques.

2.5.1 La couverture des états

La couverture des états signifie qu'il faut passer par tous les états du système de transitions au moins une fois. Dans la figure 2.35, afin de couvrir tous les états du système, il suffit d'exécuter par exemple la séquence suivante :

- ***a, d, e***

Cette séquence d'appels nous permet de passer par tous les états de ce système et nous assure que les états 1, 2, 3 et 4 sont couverts.

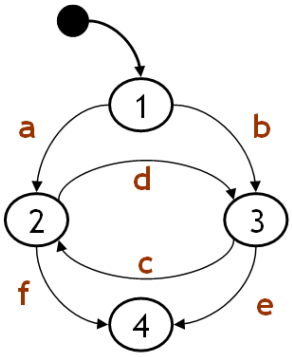


Figure 2.35: Exemple d'un système de transitions fini

2.5.2 La couverture des branches

La couverture des branches signifie qu'il faut parcourir toutes les transitions du système considéré à partir de l'état initial.

Reprenons la figure 2.35, nous considérons les deux séquences de test suivantes:

- ***a, d, e***
- ***b, c, f***

Ces deux séquences permettent de parcourir toutes les transitions du système et assurent ainsi une couverture totale des branches.

2.5.3 La couverture des paires de branches

La couverture des paires de branches est semblable à la couverture des branches sauf que dans le cas de paires de branches, on ne considère pas chaque branche comme une seule entité mais plutôt les branches deux à deux lorsqu'elles sont successives. Dans l'exemple de la figure 2.35, on constate l'existence des paires de branches suivantes : ***ad ; af ; bc ; be ; cd ; cf ; dc ; de***.

Les séquences de tests suivantes nous permettent une couverture totale de ce système au niveau d'une couverture de paires de branches :

- ***a, f***
- ***a, d, c, f***

- ***b, c, d, e***
- ***b, e***

Nous pouvons généraliser la couverture des branches et la couverture des paires de branches en une couverture de n-branches où le critère de couverture consiste à couvrir toutes les séquences possibles de n branches successives. Dans ce cas, la couverture des branches correspond à une couverture 1-branche et la couverture de paires de branches correspond à une couverture 2-branches.

2.5.4 La couverture des chemins

La couverture des chemins consiste à parcourir tous les chemins d'exécution possibles. Un chemin d'exécution est une suite de transitions permettant de partir d'un état initial et arriver à un état final. Cette notion de couverture est plus puissante que toutes celles présentées précédemment. Dans l'exemple de la figure 2.35, nous considérons que l'état 4 est un état final. Dans ce même exemple, existe une boucle entre les états 2 et 3, d'où la nécessité de définir des hypothèses. En effet, sans hypothèse de notre part sur le nombre, jugé suffisant, de parcours de cette boucle, nous aurons un ensemble infini de tests pour couvrir totalement la spécification selon le critère de couverture des chemins. Une autre idée consiste aussi à borner la longueur des chemins acceptables comme séquence de test. Dans le cas où nous considérons qu'un seul parcours de la boucle est suffisant, l'ensemble de tests assurant une couverture totale de la spécification selon le critère de couverture des chemins est le suivant :

- ***b, e***
- ***b, c, f***
- ***b, c, d, e***
- ***a, f***
- ***a, d, e***
- ***a, d, c, f***
- ***b, c, d, c, f***
- ***a, d, c, d, e***

2.5.4 La couverture par hypothèse de test

Dans la littérature existe d'autres formes de couverture de spécifications formelles moins classiques que celles que nous venons de présenter précédemment. Un exemple de ces différentes formes est la couverture par hypothèse de test nommée aussi H-couverture [Bontron, 2005]. Les hypothèses de test constituent des outils de sélection de tests [Bernot et al, 1991]. Elles aident à définir une suite de test en limitant la taille de cette dernière et non pas à la créer depuis rien. Dans le cas de test boîte noire où l'implantation n'est connue que par ses interfaces, la qualité de tests (capacité à détecter les erreurs) est faite à travers des

hypothèses sur l'implantation à tester. Grâce à ces hypothèses, un jeu de test couvrant un ensemble infini de comportements peut être rendu fini. Ainsi, un test exhaustif peut être effectué à partir d'un ensemble de tests fini.

2.6 Conclusion

Nous avons défini dans la première partie de ce chapitre deux méthodes de vérification en informatique : le test et la vérification formelle. Il s'agit de deux méthodes complémentaires garantissant le bon fonctionnement d'un système. Nous avons ensuite développé deux types de test : le test structurel et le test fonctionnel. Nous nous sommes focalisés sur le test fonctionnel et en particulier le test de conformité étant donné notre objectif de génération de scénarios de test pour la validation d'un composant réactif. Selon la norme ISO9646, nous avons identifié les principaux éléments constituant le test de conformité : la spécification, l'implantation à tester et la relation de conformité. Nous avons ensuite présenté un état d'art sur les différentes méthodes et outils de génération de scénarios de test de conformité. Concernant les méthodes, nous nous intéressons à la méthode par dérivation de spécification qui nous permet, à partir d'un modèle de spécification représentant les comportements souhaités, de générer les scénarios de test vérifiant l'implantation à tester. Ces scénarios de test doivent être exhaustifs afin de garantir la couverture de la spécification. Dans la dernière section de ce chapitre, nous avons défini la couverture de test en présentant les différents types de couverture existants dans la littérature.

Nous avons déjà mentionné que l'automatisation de la génération de test est importante dans le sens où elle permet de réduire le coût du test. Cependant l'automatisation nécessite des efforts de formalisation. La spécification constitue une référence dans le cas du test de conformité. C'est à partir de cette spécification que nous déterminons les verdicts. Il est alors nécessaire de la formaliser afin que les comportements qu'elle décrit soient suffisamment précis et non ambigus pour que les verdicts de test aient un sens. Le modèle de la spécification doit posséder un aspect mathématique afin que sa sémantique soit claire et qu'il puisse être analysable. Il est aussi important de formaliser l'interaction entre les cas de test et l'implantation sous test lors de la phase d'exécution des tests.

Dans le chapitre suivant, nous nous intéressons à la formalisation de la spécification. Comme nous l'avons décrit dans le chapitre 1, les spécifications du système ERTMS sont informelles et multiples. D'où la nécessité de regrouper et formaliser ces spécifications afin de produire des modèles formels. L'objectif du chapitre est de décrire notre méthode de formalisation des spécifications à travers une transformation de modèles UML (Unified Modelling Language) vers des modèles de Réseaux de Petri.

3 FORMALISATION DES SPECIFICATIONS

« Rien ne se perd, rien ne se crée, tout se transforme… »
(Lavoisier, chimiste français)

Résumé:

La génération automatique de scénarios de test nécessite une spécification formelle afin d'augmenter la fiabilité des verdicts de tests. L'objectif de ce chapitre est de présenter une méthode de formalisation de spécification en se basant sur des modèles formels. La spécification ERTMS est sous la forme de modèles UML (Unified Modeling Language). Ce langage semi-formel permet de décrire la vue statique ainsi que la vue dynamique d'un système à travers plusieurs types de diagrammes. Il est considéré comme un standard de modélisation objet et possède de nombreux avantages concernant la structuration et l'organisation des données d'un système. Cependant, générer des tests de conformité pour la validation de constituants réactifs nécessite des modèles formels ayant un aspect mathématique et une sémantique claire et non ambigüe.

Nous nous inspirons de l'ingénierie dirigée par les modèles pour la formalisation de notre spécification. Nous développons une approche de modélisation basée sur le couplage de modèles semi-formels et de modèles formels. Ce couplage se fait à travers une technique de transformation de modèles. Le formalisme cible choisi est celui des réseaux de Petri (RdP). Il s'agit d'un langage formel ayant une base théorique et mathématique solide permettant la génération de scénarios de tests. La classe des RdP développée dans ce chapitre est une sous-classe des RdP interprétés. La description de la spécification en UML à travers le diagramme de classes, les diagrammes de machines d'états et les diagrammes de séquences est ensuite transformée en un modèle RdP suivant un ensemble de règles de transformation établies entre le méta-modèle UML et le méta-modèles des réseaux de Petri interprétés.

Sommaire du chapitre 3

3.1 Introduction

Nous avons défini dans le chapitre 2, un état de l'art sur les différentes techniques de vérification de constituants réactifs. Nous nous sommes intéressés en particulier à la méthode de test de conformité. Développé pour la validation des protocoles de communication [Samuel et *al*, 2007], ce type de test correspond aux contraintes de notre contexte industriel. Le système ERTMS est en effet un système complexe réparti intégrant des composants réactifs. Le principal objectif de nos travaux consiste à vérifier ces composants à travers la génération de scénarios de tests à partir des modèles de la spécification ERTMS (cf. Section 2.3.3). La forme de cette spécification ne favorise pas la génération de scénarios de test et requiert une formalisation du cahier des charges. Dans ce chapitre, nous décrivons notre méthodologie de construction de modèles formels à partir de modèles UML (Unified Modeling Language) de la spécification. En tant que standard de l'OMG (Object Management Group), UML [OMG, 2005] est un langage de modélisation objet possédant une popularité sans précédent à la fois dans le monde industriel et académique. Il est constitué d'un ensemble de notation que l'on peut qualifier de semi-formel mais considéré comme indispensable au niveau de la structuration et de la communication. UML possède plusieurs types de diagrammes qui proposent des visions parcellaires de la dynamique des systèmes : soit nous représentons les changements d'états dans un objet par une machine d'états, soit nous représentons un dialogue entre objets lors d'un scénario particulier par un diagramme de séquences… Or le besoin d'une vue d'ensemble de toutes les interactions est indispensable pour l'évaluation des performances des systèmes. De plus, le langage UML n'offre pas de possibilités pour une vérification formelle. Nous avons mentionné dans le chapitre précédent l'importance des méthodes formelles dans la phase du test. Elles augmentent la qualité des tests et la fiabilité des verdicts. C'est la raison pour laquelle nous avons jugé qu'une utilisation conjointe d'une notation semi-formelle et d'un langage formel peut être efficace pour la description de la spécification ERTMS. Les modèles formels de la spécification sont construits à partir des modèles UML à travers une technique de transformation de modèles.

La modélisation formelle permet de rendre clair et explicite un ensemble d'informations largement implicites. Lors de la construction de modèles formels, des ambiguïtés doivent être levées à travers des choix et des prises de décisions [Valette, 1999]. L'objectif de la vérification, souvent associée à une modélisation formelle, est de s'assurer que le modèle correspond bien au système étudié et qu'il est effectivement dépourvu d'incohérences et

d'imprécisions. Cependant, les langages formels se révèlent parfois complexes à maîtriser. Ils obligent généralement l'analyste concepteur à entrer rapidement dans des niveaux de détails qui ne facilitent pas une compréhension globale du système en construction. Ceci constitue une raison de plus pour prouver la complémentarité de la modélisation formelle vis-à-vis de la modélisation semi-formelle. Le but de notre approche est de combiner l'usage de deux types de modélisations pour la formalisation de la spécification et montrer les avantages que l'on peut en tirer.

Le langage formel choisi est le formalisme des réseaux de Petri (RdP) [Murata, 1989], et plus particulièrement, les réseaux de Petri interprétés (cf. Section 3.3.2.3). Ce choix est motivé par un ensemble de facteurs. Tout d'abord les RdP sont des méthodes formelles. De plus, ils renseignent, dans un modèle unique, sur les deux aspects du système représenté: statique (grâce à la structure même du modèle) et dynamique (grâce à l'évolution des jetons dans la structure et éventuellement l'évolution de leurs valeurs). Enfin, en termes de puissance de modélisation, ce formalisme possède également une très grande expressivité (modélisation des communications, des contraintes temporelles et des lois de commande).

Dans la première partie de ce chapitre nous décrivons les approches de modélisation couplant les modèles semi-formels et les modèles formels. Ensuite, nous détaillons notre méthodologie de transformation de modèles UML vers des modèles de réseaux de Petri interprétés. Enfin, nous illustrons notre démarche à travers un exemple complet.

3.2 Approches de modélisation

La modélisation permet d'analyser le comportement d'un système. Il s'agit d'une abstraction de la réalité dans le but de mieux la comprendre [Chapurlat, 2007]. L'abstraction se fait selon plusieurs points de vue (fonctionnel, structurel et dynamique). Chaque point de vue est décrit à l'aide d'un ou de plusieurs modèles hiérarchisés. Chacun d'eux est une instance d'un langage de modélisation [Combemale, 2008]. Le langage de modélisation est une construction linguistique (textuelle et/ou graphique) pouvant exprimer une connaissance sur un système dans une structure définie par un ensemble de règles consistantes. Les règles servent à interpréter le sens de composantes du modèle [Bouali, 2009]. Dans l'ingénierie dirigée par les modèles (IDM), la modélisation occupe une position centrale par rapport à la programmation. L'IDM se base en effet sur le principe du « tout est modèle ». Cette nouvelle approche est considérée en continuité avec la technologie objet (où « tout est objet »). En effet, concevoir des systèmes informatiques sous la forme d'objets communicants entre eux a déclenché le problème de classification de ces objets en fonction de leurs différentes origines (objets métiers, techniques, etc.) [Bézivin, 2005]. Le concept clé de l'IDM est la notion de

modèle que de nombreux travaux ont essayé de définir ([Bézivin & Gerbé *al*, 2001], [Seidewitz, 2003], [Chapurlat, 2007] et [Combemale, 2008]) :

Définition 3.2.a/ **Modèle** [Combemale, 2008]. Un modèle est une abstraction d'un système, modélisé sous la forme d'un ensemble de faits construits dans une intention particulière. Un modèle doit pouvoir être utilisé pour répondre à des questions sur le système modélisé.

Définition 3.2.b/ **Modèle** [Chapurlat, 2007]. Un modèle est une abstraction, un filtre de la réalité qui ne rend compte que d'une partie de la connaissance relative au phénomène étudié et qui masque nombre de détails considérés comme superflus ou sans intérêt lors de la modélisation.

Nous déduisons de ces deux définitions l'importance de la relation qui existe entre le modèle et le système qu'il représente, appelée « ***représentation De*** » [Bézivin, 2005], et nommée μ sur la figure 3.36. Même si cette relation a suscité de nombreuses réflexions, il est toujours difficile de la formaliser et de répondre ainsi à la question « qu'est ce qu'un bon modèle ? ». Néanmoins, le modèle doit être une abstraction pertinente du système modélisé. Une abstraction pertinente signifie un modèle suffisant et nécessaire permettant de répondre à certaines questions exactement de la même manière que le système modélisé aurait répondu lui-même. Il s'agit du principe de substituabilité [Minsky, 1968]. Dans [Valette, 1999], Valette définit un bon modèle comme un modèle que l'on comprend bien et que l'on peut facilement expliquer. Les procédures de vérification doivent être simples et convaincantes. Les exigences, les contraintes et les informations exprimées par le modèle doivent être clairement identifiées. Enfin, Valette considère la modularité comme une caractéristique importante car un bon modèle n'est pas nécessairement minimal.

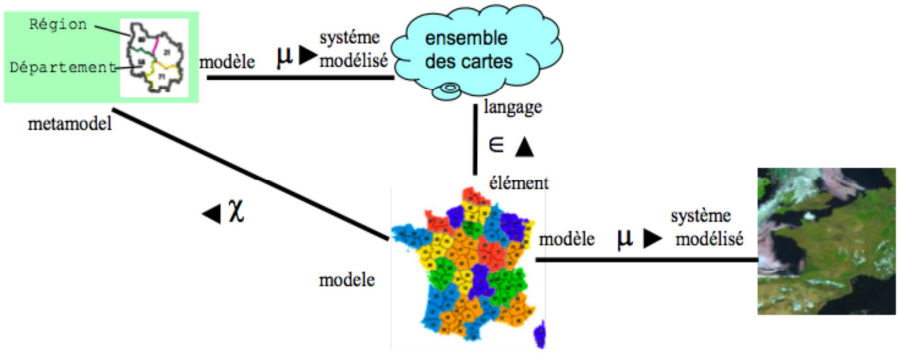


Figure 3.36: Relations entre système, modèle, méta-modèle et langage [Favre et al, 2006]

Sur la figure 3.36, nous reprenons l'exemple utilisé dans ([Favre et *al*, 2006] et [Combemale, 2008]) qui s'appuie sur la cartographie pour décrire les différents concepts de l'IDM. Dans cet exemple, une carte est un modèle (une représentation) de la réalité, avec une intention particulière (carte routière, administrative, des reliefs, etc.). Dans l'IDM, la notion de modèle est explicitement liée à la notion de langage bien défini. En effet, le modèle est exprimé à

travers un langage qui doit être clairement défini. De manière naturelle, la définition d'un langage de modélisation a pris la forme d'un modèle, appelé méta-modèle.

Définition 3.2.c/ **Méta-modèle** [Combemale, 2008]. Un méta-modèle est un modèle qui définit le langage d'expression d'un modèle [OMG, 2006], c'est-à-dire le langage de modélisation.

Cette seconde notion de méta-modèle qui lie le modèle au langage utilisé pour le construire, est appelée « ***conformeA*** » et nommée χ sur la figure 3.36. Dans l'exemple précédent (cf. Fig. 3.36), il est indispensable d'associer à une ou plusieurs cartes la légende utilisée pour la ou les comprendre. Il s'agit de la description du langage utilisé pour réaliser une carte et auquel cette carte doit se conformer. Nous pouvons alors qualifier la légende de méta-modèle. En effet, elle est considérée comme un modèle représentant un ensemble de cartes (μ) et à laquelle chacune de ces cartes doit se conformer (χ). A travers ces deux relations, nous comprenons la différence qui existe entre le langage qui représente le système et le méta-modèle qui représente le modèle de ce langage. C'est sur ce principe de méta-modélisation (cf. définition 3.2.d) que s'appuie l'OMG pour définir l'ensemble de ses standards, en particulier le langage UML [OMG, 2007].

Définition 3.2.d/ **Méta-modélisation** [Combemale, 2008]. La méta-modélisation est une activité qui consiste à définir le méta-modèle d'un langage de modélisation. Elle vise donc à bien modéliser un langage, qui joue alors le rôle de système à modéliser.

Suite à la définition de la méta-modélisation et l'acceptation de la notion de méta-modèle et devant l'inquiétude de voir émerger une grande variété de méta-modèles de façon incompatible, il y a eu un besoin urgent de formaliser la méta-modélisation. La réponse fut la naissance du méta-méta-modèle MOF (Meta-Object Facility) [OMG, 2006]. C'est un langage de définition de méta-modèle qui prend la forme de modèle. Afin de limiter le nombre de niveaux d'abstraction, le méta-méta-modèle MOF possède la capacité de se décrire lui-même ; il s'agit de la propriété de méta-circularité.

Définition 3.2.e/ **Méta-méta-modèle** [Combemale, 2008]. Un méta-méta-modèle est un modèle qui décrit le langage de méta-modélisation, c'est-à-dire les éléments de modélisation nécessaires à la définition des langages de modélisation. Il a de plus la capacité de se décrire lui-même.

Tous les concepts que nous venons de décrire représentent les concepts clés de l'IDM. Dans le contexte de nos travaux, nous utilisons une approche basée sur les modèles afin de formaliser la spécification. Nous avons énoncé dans l'introduction de ce chapitre l'intérêt

d'une utilisation conjointe de modèles formels et de modèles semi-formels. Dans la section suivante, nous définissons les différents principes de couplages de ces deux types de modèles.

3.2.1 Les principes de couplage des modèles semi-formels et formels

L'idée de coupler des modèles semi-formels et formels pour la modélisation de systèmes a permis de résoudre de nombreuses difficultés et de déterminer un certain nombre de pratiques saines.

3.2.1.1 Adjonction ou dérivation de modèles

Pour réaliser une approche couplant des modèles objet et des modèles formels, il suffit d'intégrer ces derniers à la modélisation orientée objet soit par adjonction soit par dérivation. *Dans le cas d'une intégration par adjonction*, la description semi-formelle du modèle Orienté Objet est complétée par une description formelle de certains aspects statiques ou dynamiques. Il n'y a pas de relations explicites entre les modèles formel et semi-formel. Ainsi, l'analyste-concepteur se charge de s'assurer que la modélisation formelle respecte bien la modélisation semi-formelle. Cette approche implique donc naturellement de la rigueur dans le développement, mais la cohérence de la modélisation et le respect des exigences reposent en grande partie sur les compétences de l'analyste-concepteur. Contrairement à l'intégration par adjonction, *l'intégration par dérivation* représente des relations explicites entre les deux modèles. En effet, elle consiste à générer automatiquement ou non, les modèles formels à partir des modèles semi-formels. Des techniques de transformation ou de translation du semi-formel vers le formel sont utilisées. Il s'agit bien de techniques de passage d'un formalisme à un autre et non pas de traduction étant donné que les modèles semi-formels et les modèles formels ne sont pas mathématiquement équivalents. Ceci suppose l'existence de règles de transformation décrivant comment les éléments composant le modèle semi-formel sont translatés vers les éléments composant le modèle formel.

Comme nous l'avons mentionné dans notre problématique (cf. section 1.4.3), nous avons privilégié une approche par dérivation à partir de modèles semi-formels UML. Cette approche correspond à nos attentes et notre souci de génération de scénarios de test à partir de modèles formels.

Le langage UML est un langage de modélisation défini par une architecture de méta-modélisation à quatre niveaux. Le premier niveau M3 (ou le méta-méta-modèle) constitue une spécification du modèle inférieur qui est le M2 (ou le méta-modèle). Ce dernier représente également une spécification du modèle au-dessous qui est le M1 (ou le modèle du système). Enfin, le M1 spécifie les données du système réel qui est le M0 (cf. Fig.3.37).

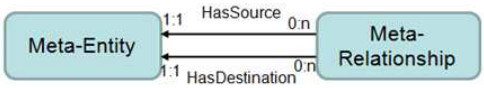
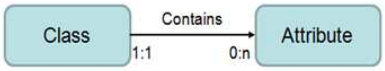
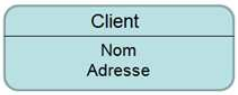
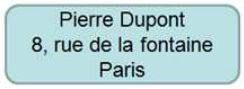
Couche	Description	Exemple
Méta-méta-modèle	Spécification du méta-modèle	
Méta-modèle	Spécification du modèle	
Modèle	Spécification des données réelles	
Système réel	Données	

Figure 3.37: Structuration du langage UML en couches

En raison de cette structuration en couches d'UML, les techniques de dérivation peuvent également être classées sur différents niveaux :

- La dérivation du modèle : Pour effectuer la translation, une correspondance est établie entre chaque élément du modèle UML et les éléments construisant le langage formel [Bruel, 1998].
- La dérivation du méta-modèle : Cette dérivation sert à formaliser le méta-modèle UML à travers un langage formel [Reggio et *al*, 2001].
- La dérivation du méta-méta-modèle : Finalement, cette dérivation permet de spécifier, à la fois, le méta-méta-modèle, mais également et surtout les règles d'instanciation du méta-modèle UML [Fernandez et *al*, 2001].

Ces trois méthodes de dérivation diffèrent dans leurs objectifs. En effet, la troisième technique a comme objectif la formalisation et la détection des erreurs dans les évolutions du langage UML et de ses profils (extensions du langage UML). Quant aux deux premières, elles s'intéressent à une version spécifique d'UML et sont donc plus proches de nos préoccupations. Nous avons opté pour une approche de dérivation de modèle puisque notre objectif est de partir d'un modèle source UML de la spécification pour arriver à un modèle cible formel .Cette approche nous paraît la plus pragmatique et la plus simple à mettre en œuvre. L'objectif principal de nos travaux de thèse consiste à définir des règles de transformation vers un langage formel afin de pouvoir développer une technique de vérification.

Une fois que notre choix de dérivation de modèles UML est spécifié, reste à résoudre la question de l'automatisation ou non du procédé de dérivation.

3.2.1.2 Dérivation automatique ou semi-automatique ?

L'un des sujets relatifs à l'IDM et qui ont fait l'objet de recherches intensives dans la littérature sont les langages de modélisation [Bézivin, 2005], la transformation de modèles [Jouault, 2006], le mapping entre métamodèles [Jouault, 2006], et les méthodologies de conception [Bézivin, 2005]. Parmi ces sujets de recherche, les langages de transformations de modèles jouent un rôle important dans le sens où ils définissent la manière dont un ensemble d'éléments d'un modèle source est transformé en un ensemble d'éléments d'un modèle cible. Toutefois, la réalisation de ces transformations manuelle constitue une source d'erreurs, rendant ainsi la tâche fastidieuse et le processus de transformation coûteux. Ces transformations se basent sur un ensemble de règles faisant intervenir, et en même temps fusionnant, les techniques de mapping et de transformation entre le méta-modèle source et le méta-modèle cible. La semi-automatisation de ce processus de transformation présente un réel challenge dont les avantages sont considérables : la diminution du temps de développement de la transformation et la réduction des éventuelles erreurs pouvant se produire lors d'une spécification manuelle des transformations. Dans nos travaux, nous avons initié une tentative vers la semi-automatisation de notre processus de dérivation des modèles objet UML vers des modèles formels. Nous avons en effet, développé une approche qui définit en premier lieu les règles transformant les éléments constituant le modèle source vers les éléments constituant le modèle cible. En second lieu, nous avons développé un moteur de transformation qui décrit et exécute nos règles de transformation pour produire le modèle formel à partir du modèle UML de manière automatique.

Dans ce qui suit nous présentons le principe de la technique de transformation de modèles.

3.2.2 La technique de transformation de modèles

La transformation de modèles est utilisée par l'IDM afin de rendre les modèles opérationnels : il s'agit du principe de l'approche MDA (Model Driven Architecture). Cette approche a été créé par l'OMG en 2000 afin de promulguer les bonnes pratiques de modélisation et d'exploiter pleinement les avantages de modèles. Elle a été mise en œuvre pour répondre à une problématique d'hétérogénéité des plates-formes de développement et pour pouvoir séparer les besoins techniques des besoins fonctionnels. Elle se base sur le standard de modélisation UML pour décrire des modèles différents le long des étapes de cycle de développement d'un système. De manière plus précise, l'approche MDA [Blanc, 2005] permet de définir des modèles d'exigences indépendants de toute implémentation appelés CIM (Computation Independent Model), des modèles d'analyse et de conception indépendants de la plate-forme PIM (Plateform Independent Model) et des modèles de code spécifiques à la plate-forme PSM (Plateform Specific Model) (cf. Fig.3.38). L'élaboration des modèles indépendants des détails techniques des plate-formes permet de répondre uniquement aux spécifications fonctionnelles. Ensuite, la transformation automatique de ces

modèles en modèles spécifiques des plates-formes permet un gain en productivité.

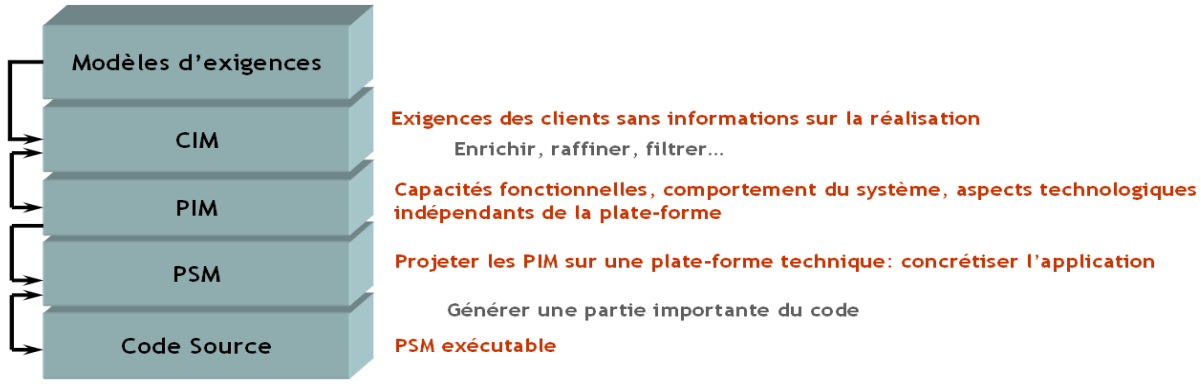


Figure 3.38: Approche MDA

3.2.2.1 Principe

La transformation de modèles permet de transformer un modèle M_a en un modèle M_b conformément à leurs méta-modèles respectifs MM_a et MM_b [Jabri & Lemaire, 2007a]. Ces deux derniers peuvent être identiques dans le cas d'une transformation **endogène** ou bien différents dans le cas d'une transformation **exogène**. D'autre part, la transformation de modèles peut être **verticale** dans le cas d'un changement d'un niveau d'abstraction (passage d'un PIM vers un PSM et inversement) ou bien **horizontale** pour rester au même niveau d'abstraction (d'un PIM à un PIM ou d'un PSM à un PSM) [Gerber et *al*, 2002].

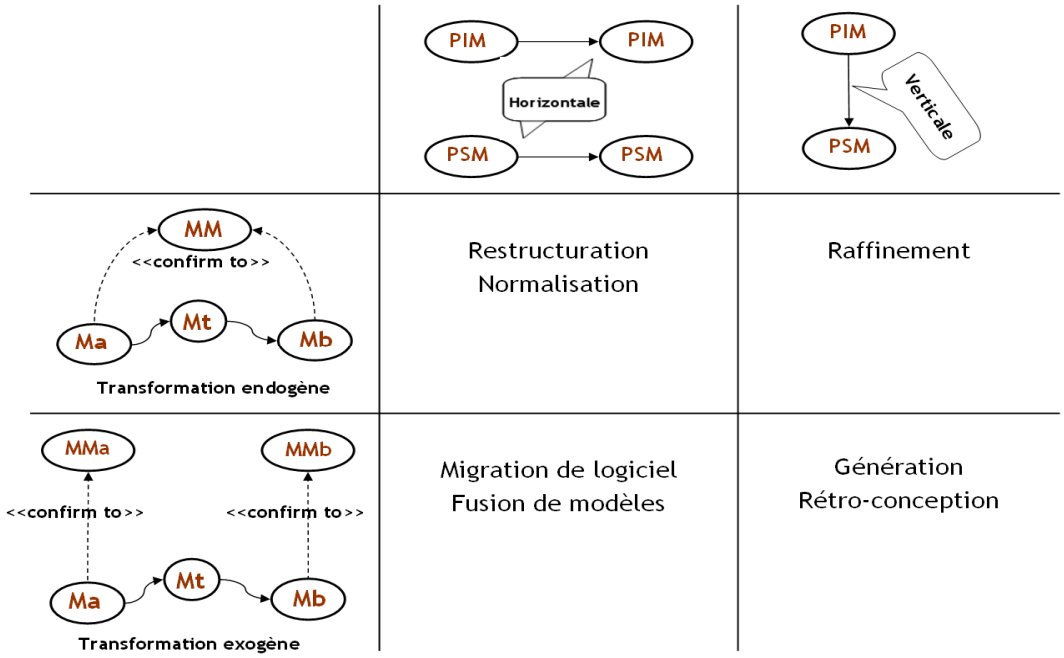


Figure 3.39: Classes de transformation [Combemale, 2008]

La figure 3.39 ci-desssus résume ces différentes classes de transformation en indiquant leurs cas d'utilisation. Le modèle Mt dans la figure désigne le modèle de transformation de M_a vers M_b. Enfin, la transformation de modèle est fréquemment utilisée dans la définition des langages de modélisation dans l'objectif de définir des mappings et des traductions entre différents langages, ce qui constitue notre cas d'utilisation de cette approche. En effet, cette technique nous sert à traduire nos modèles d'un langage de modélisation objet (UML) vers un formalisme de modélisation formel (réseaux de Petri). Le succès de l'IDM réside dans sa capacité de résoudre la problématique de la transformation des modèles. Ce sujet a fait objet de plusieurs travaux académiques et industriels durant ces dernières années.

Dans ce qui suit nous présentons de manière brève un état d'art sur les techniques de transformation de modèles.

3.2.2.2 Historique

La transformation de modèles n'est pas une activité récente. Elle peut être classée en trois générations en fonction de la structure de donnée utilisée pour décrire le modèle [Gerber et *al*, 2002] :

- ✓ (1) La génération de transformation de structures séquentielles qui génère un script décrivant comment un fichier d'entrée est traduit en un fichier de sortie (par exemple les scripts Unix). Ce genre de transformation nécessite une analyse grammaticale du texte d'entrée et une adaptation du texte de sortie.
- ✓ (2) La génération de transformation d'arbres qui permet de parcourir un arbre d'entrée et de générer au cours de ce parcours les fragments de l'arbre de sortie. Généralement, ces méthodes utilisent des documents au format XML (eXtensible Markup Language, langage à balises extensible).
- ✓ (3) La génération de transformation de graphes qui permet de transformer un modèle d'entrée sous le format de graphe orienté étiqueté en un modèle de sortie.

Les méthodes de la dernière génération ont l'objectif de définir la procédure de transformation sous la forme d'un autre modèle M_t (cf. Fig.3.39) respectant son propre méta-modèle. Ce dernier est défini par un langage de méta-modélisation par exemple le MOF (Meta-Object Facility). La relation de transformation d'un modèle M_a conforme à son méta-modèle MM_a en un modèle M_b conforme à son méta-modèle MM_b à travers un modèle de transformation Mt, peut être décrite de la manière suivante :

$$M_b \leftarrow f\,(MM_a, MM_b, M_t, M_a).$$

Cette dernière génération de transformation a fait l'objet de plusieurs études de recherches ([Czarnecki et al, 2003], [Jouault, 2006]). Dans le cadre de nos travaux, nous nous intéressons à ce type de transformation. Tout d'abord, nous transformons des modèles de même niveau d'abstraction, il s'agit donc d'une transformation ***horizontale*** *(cf. Fig.3.39).*

Ensuite, les deux méta-modèles du modèle source et du modèle cibles sont différents étant donné qu'on transforme des modèles de deux langages différents, il s'agit d'une transformation ***exogène*** *(cf. Fig.3.39). Enfin, reste à préciser le langage utilisé pour écrire la transformation. Dans ce qui suit, nous présentons un petit aperçu de ces langages.*

3.2.2.3 Langages de transformation de modèles

Plusieurs langages permettent aujourd'hui d'écrire des transformations de graphes. Nous citons tout d'abord les langages généralistes qui opèrent directement sur la représentation abstraite du modèle. Un exemple de ces langages est l'API (Application Programming Interface, Interface d'application) d'EMF (Eclipse Modelling Framework) [Budinsky et *al*, 2003] qui, couplée au langage Java, permet de manipuler un modèle sous la forme d'un graphe. Dans ces conditions, le programmeur se charge de chercher les informations dans le modèle et de gérer l'application des règles de transformation afin de construire les éléments cibles.

Afin d'éviter ces détails de mise en œuvre, des langages de modélisation dédiés à la transformation de modèle ont été définis. M_t doit être conforme à un méta-modèle et nécessite des outils permettant de l'exécuter. Nous citons l'exemple d'ATL (ATLAS Transformation Language) [Jouault, 2006] qui est un langage hybride (déclaratif et impératif) qui permet de définir une transformation de modèle à modèle (appelée Module) sous la forme d'un ensemble de règles. Ce langage permet également de définir des transformations de type modèle vers texte (appelée Query). Le modèle en entrée est décrit à partir de méta-modèle en Ecore [Jouault, 2006]. L'OMG a défini le standard QVT (Query/View/Transformation) [OMG, 2008] dans le but de donner un cadre normatif pour l'implantation des différents langages dédiés à la transformation de modèle. Le méta-modèle de QVT est conforme à MOF.

Dans le cas de nos travaux, nous avons défini un ensemble de règles de transformation conformément au mapping effectué entre le méta-modèle source MM_a et le méta-modèle cible MM_b. Il s'agit alors d'une transformation horizontale, exogène utilisant des règles de transformation afin de traduire des modèles d'un langage L1 vers des modèles dans un langage L2. L'exécution de ces règles de transformation nous a permis de transformer automatiquement nos modèles sources en modèles cibles. Dans la section suivante, nous détaillons notre méthode de transformation de modèles. Tout d'abord, nous présentons le langage L1 qui décrit nos modèles sources et le langage L2 qui décrit nos modèles cibles. L1 est le langage UML (en particulier la vue dynamique de ce langage) et L2 est le formalisme des réseaux de Petri interprétés. Nous argumentons ensuite les choix définis. Enfin, nous détaillons les règles de transformation développées entre les deux formalismes.

3.3 Transformation des modèles UML en Réseaux de Petri interprétés

Comme nous l'avons déjà mentionné, l'objectif principal de cette recherche est la génération de scénarios de test pour la vérification d'un composant du système de contrôle commande et de signalisation ERTMS par rapport à ses spécifications. L'utilisation de la vue de comportement dynamique du langage UML afin de modéliser les spécifications ERTMS pour un composant donné interagissant avec son environnement externe s'impose alors. En effet, la vue de machines d'états est particulièrement adaptée à ce type de modélisation. Elle décrit en effet le comportement dynamique des objets dans le temps en modélisant les cycles de vie des objets de chaque classe. Chaque objet est traité comme une entité individuelle qui communique avec son environnement en recevant des évènements et en y répondant. Ces évènements représentent les types de changements qu'un objet peut détecter : la réception d'appels ou de signaux explicites entre deux objets, la modification de certaines valeurs ou le passage du temps. Tout ce qui est susceptible d'affecter un objet est défini comme un évènement. Ces évènements de l'environnement réel sont modélisés comme des signaux allant vers le système. Les diagrammes de machines d'états illustrent cette vue de machine d'états. Nous allons les présenter de manière plus détaillée dans la suite. Quant au formalisme cible, nous avons choisi la classe des réseaux de Petri interprétés que nous détaillons plus tard. Nous avons déjà mentionné dans l'introduction de ce chapitre les arguments de notre choix pour le formalisme des réseaux de Petri.

Dans la section 3.3.2, nous détaillons ces arguments et nous décrivons la classe des réseaux de Petri interprétés.

3.3.1 Modèle source : le langage UML

UML est un langage de modélisation visuel à caractère général. Il a été mis en œuvre pour simplifier et consolider la multitude des méthodes orientées objet.

3.3.1.1 Historique du langage UML

L'analyse orientée objet (OOA) utilise les techniques de modélisation objet pour analyser les exigences d'un système. Elle considère le monde comme un ensemble d'objets ayant des structures de données et des comportements. En effet, l'idée qu'un système puisse être considéré comme une population d'objets en interaction, dont chacun représente une entité atomique de données et de fonctionnalités, est le fondement de la technologie objet.

Les méthodes d'analyse objet réalisent une rupture radicale avec les méthodes antérieures de spécification des exigences telle que l'analyse fonctionnelle et l'analyse structurée [Yourdon, 1979]. Contrairement à ces anciennes méthodes d'analyse, l'analyse orientée objet possède les avantages suivants :

(1) maintenabilité grâce au « mapping » simplifié vers le monde réel qui, pour moins d'effort d'analyse, permet moins de complexité dans la conception du système et facilite la vérification pour l'utilisateur;

(2) réutilisabilité des « artefacts » d'analyse économisant ainsi le temps et les coûts ;

(3) gain de productivité à travers le « mapping » direct vers les langages de programmation orientée objet [Baudoin, 1996].

Le monde orienté objet se base sur les concepts suivants: objets, classes, héritage et agrégation. Les objets constituent tout ce qui est physique ou conceptuel existant dans l'univers qui nous entoure, par exemple: le matériel, le logiciel, les documents, les êtres humains, et même les concepts. Un objet possède un état qui n'est qu'un ensemble de circonstances décrivant ce dernier. La classe est une collection d'objets similaires ayant les mêmes attributs et les mêmes méthodes. Pour une classe donnée, les objets crées sont appelés les instances de la classe. Chaque instance aura sa propre identité. Il est possible pour les objets d'être composés d'autres objets, il s'agit d'une relation d'agrégation ou de composition. Lorsque la destruction du composé entraîne la destruction de ses composants, il s'agit d'une forte agrégation, qui est la composition. Toutefois, l'agrégation n'est pas la seule manière dans laquelle deux objets peuvent être liés. Un objet peut être une spécialisation d'un autre objet par héritage. Ce concept peut être défini comme le processus par lequel un objet reçoit les caractéristiques d'un ou plusieurs autres objets [OMG, 2005]. De nombreuses méthodes d'analyse orientée objet ont été décrites depuis 1988 par Shlaer-Mellor [Shlaer et *al*, 1988], Jacobson [Jacobson, 1987], Coad-Yourdon [Coad et *al*, 1991], Booch [Booch, 1991] et Rumbaugh [Rumbaugh et *al*, 1997]. En 1997, Rumbaugh [Rumbaugh et *al*, 1997], Booch [Booch, 1991] et Jacobson [Jacobson, 1987] ont unifié leurs méthodes pour produire le langage de modélisation unifié (UML), qui est devenu le standard d'analyse et de modélisation objet.

3.3.1.2 Les bénéfices du langage UML

Dans le cadre de nos travaux, nous avons choisi de modéliser, dans une première étape, notre spécification informelle (cf. section 1.3.2) en UML [OMG, 2005]. Le principal argument en faveur de ce langage est son statut de standard de modélisation Objet. UML permet de documenter et spécifier graphiquement tous les aspects d'un système à logiciel prépondérant. Il jouit d'une popularité sans précédent à la fois dans le monde industriel et académique [Fontan, 2008]. Il est considéré comme un langage semi-formel et un support de communication performant. Il cadre l'analyse et facilite la compréhension de représentations abstraites complexes [Booch et *al*, 2000]. En effet, UML permet de modéliser et de visualiser un système à l'aide de diagrammes (cf. Fig.3.40). Chaque diagramme constitue une perspective du modèle, possède une structure et véhicule une sémantique

précise [Jabri et *al*, 2008a]. Combinés, les différents types de diagrammes UML offrent une vue complète des aspects statiques, fonctionnels et dynamiques d'un système donné formant ainsi le modèle UML [Audibert, 2009]. A titre d'exemple, le diagramme de classes UML définit la structure statique d'un système en termes de classes et relations entre classes. Le diagramme d'objets quant à lui, est plus concret et souvent utilisé comme instance pour tester le diagramme de classes [OMG, 2005].

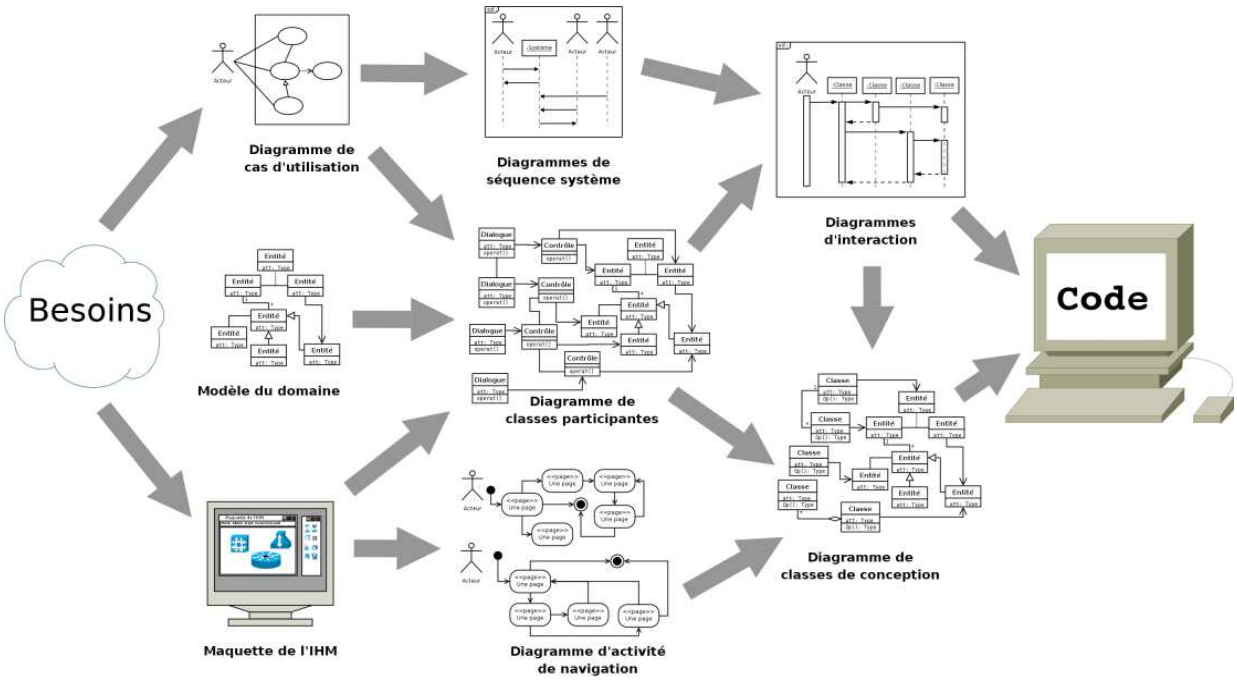


Figure 3.40: Chaîne complète de la démarche de modélisation du besoin jusqu'au code [Audibert,09]

L'évolution du langage UML est due en majeure partie à son succès dans le monde industriel, il fallait ainsi l'adapter aux besoins de différents domaines. Heureusement, UML présente l'avantage de pouvoir se décliner sous forme de « profils » spécialisés en fonction des domaines d'applications considérées. Un profil UML décrit donc une spécialisation du méta-modèle à un domaine particulier. Par exemple, deux profils UML ont été développés par l'OMG afin d'adapter UML au domaine des systèmes temps réel et systèmes embarqués: le profil SPT (Schedulability, Performance, and Time) [OMG, 2005a] et le profil MARTE (Modeling and Analysis of Real-Time and Embedded Systems) [OMG, 2007a]. Le profil SPT a été défini dans le but de fournir des paradigmes adéquats basés sur UML pour traiter des aspects liés à la modélisation du temps, à l'ordonnancement et à la performance dans les Systèmes temps réel. Le profil MARTE a été défini ensuite comme successeur de SPT. Il fournit un support pour les étapes de spécification, conception et vérification de propriétés. Il est structuré autour de deux axes : le premier permet de modéliser les caractéristiques des systèmes temps réel et systèmes embarqués; le second permet d'analyser les propriétés du système. Le concept de profil permet ainsi de résoudre le problème dans le cas des applications bien déterminées et de combler les lacunes d'UML en termes de sémantique.

Dans le cadre de nos travaux, nous n'utilisons pas les profils. Nous ignorons le comportement interne de chaque composant et nous voulons générer des scénarios de test de type boîte noire. La spécification nous informe uniquement sur les communications aux interfaces des composants. Le langage UML est utilisé pour représenter cette spécification et nous informer sur les composants qui interagissent ensemble.

Si UML possède de nombreux avantages, il présente également quelques limites qui contraignent son utilisation dans certaines applications.

3.3.1.3 Les limites du langage UML

Si le méta-modèle UML a une grande richesse d'expressivité du point de vue syntaxique, il ne dispose pas en revanche de sémantique formelle clairement définie. L'autre aspect déficient d'UML est lié à la modélisation de l'aspect dynamique. En effet, malgré le grand nombre de représentations différentes modélisant le comportement dynamique, UML ne possède pas de diagrammes présentant une vue synthétique de la plupart de ces informations. Il est difficile ainsi d'avoir une vue globale d'un objet ou d'une partie du système. Par exemple la communication entre deux objets ne peut être graphiquement modélisée que sur des diagrammes d'interaction (diagramme de séquence et diagramme de collaboration) et non sur des machines d'états. Ainsi, pour rassembler toute l'information concernant l'aspect dynamique, il faut faire appel à plusieurs types de diagrammes. Ceci est gênant quand il s'agit de plusieurs équipes travaillant sur un même projet [Jabri et *al*, 2008c]. Les risques d'incohérences entre les différentes vues risquent également d'être multipliés. Enfin une autre limite du langage UML est la représentation du temps. En effet, il n'est pas possible, sans passer par des profils, de spécifier et de modéliser des systèmes temps réel présentant des contraintes temporelles strictes.

Pour conclure cette partie, notre choix du langage UML repose sur tous les avantages des modèles orientés objet qu'il possède et principalement sa capacité de modéliser une vue opérationnelle d'un système donné de façon standard. Cependant ce langage manque d'une sémantique formelle et ne permet pas la génération directe des scénarios de test. Par contre, UML peut servir de pivot à d'autres outils de validation formelle, d'où l'idée de notre approche mixte de modélisation basée sur la transformation de modèles UML vers des modèles de réseaux de Petri. Dans la partie qui suit, nous détaillons les types de diagrammes UML choisis pour représenter la vue dynamique.

3.3.1.4 Le modèle UML

Tout modèle UML doit d'abord définir les principaux concepts de l'application, leurs propriétés internes et leurs relations mutuelles. Il s'agit de la vue statique. Les concepts de

l'application sont modélisés en classes dont chacune décrit des objets contenant des informations et qui communiquent pour implémenter un comportement. Les informations sont modélisées sous forme d'attributs et le comportement qu'ils induisent l'est sous forme d'opérations [OMG, 2005]. Une relation entre deux classes peut être une simple association ou par exemple une génération dans laquelle une classe enfant hérite des comportements de la classe parent et rajoute ses propres comportements. La vue statique se présente sous forme de diagrammes de classe.

Concernant le comportement dynamique, trois méthodes de modélisation existent. La première méthode modélise les différents états d'un objet pendant qu'il interagit avec l'environnement extérieur. La vue d'un objet isolé est une machine d'états, c'est-à-dire la vue d'un objet au moment où il répond à des évènements en fonction de son état actuel, au moment où il accomplit des actions en guise de réponses, et ses transitions vers un nouvel état. La deuxième méthode modélise l'échange de messages entre différents objets. Les interactions apparaissent dans les diagrammes de séquences et de communications. Un diagramme de séquence montre un ensemble de messages organisés en séquences temporelles. Les objets sont représentés par des lignes de vie verticales. Les messages apparaissent sous forme de flèches entre ces lignes de vie. La troisième méthode modélise le processus d'exécution d'un scénario donné à travers un diagramme d'activités où les activités servent à modéliser à la fois le comportement séquentiel et simultané [OMG, 2005].

Dans nos travaux de thèse, les diagrammes de classes UML donnent un aperçu de l'architecture statique du système ERTMS. Les constituants sont représentés par des classes et les liens entre les constituants par des relations entre ces classes. Quant au comportement dynamique, représenté par la spécification ERTMS, il est modélisé par des machines d'états UML. Les diagrammes de séquences nous permettent de distinguer les objets qui communiquent ensemble et les différents messages échangés. ***Nous avons choisi de transformer les machines d'états UML vers des réseaux de Petri***. Ces diagrammes modélisent chaque objet de manière isolée. Nous nous retrouvons ainsi dans l'obligation d'assembler les machines d'états de chaque objet afin d'avoir une vue globale des communications. Le rôle du digramme de séquences est de fournir les informations permettant d'assembler les diagrammes d'états UML suite à leur transformation en réseaux de Petri. Cette idée sera détaillée dans la section 3.3.3 qui explique notre méthodologie de transformation. Les diagrammes d'états sont des graphiques basés sur des états et des transitions. La machine d'états est habituellement reliée à une classe et elle décrit la réponse d'une instance de la classe aux évènements qu'elle reçoit. Elle peut également être rattachée à des comportements, des cas d'utilisations et à des collaborations.

Sémantique : L'objet à modéliser en machines d'états est isolé afin de pouvoir l'étudier. Nous résumons toute influence en provenance du reste du monde extérieur comme un évènement. Lorsqu'un objet détecte un évènement, il y répond en fonction de son état actuel. Cette réponse peut comprendre l'exécution d'un effet et d'un changement vers un nouvel état. Il est également possible de structurer des machines d'états pour partager des transitions. Elles peuvent également modéliser des concurrences d'accès. Le fait d'isoler un objet du reste du monde pour le modéliser constitue une vue réductrice d'un système. Elle permet de spécifier un comportement avec précision mais pas de comprendre le fonctionnement global d'un système. Dans la figure 3.41, nous présentons un petit exemple d'une machine d'états avec les principaux éléments clés que nous définissons dans la suite.

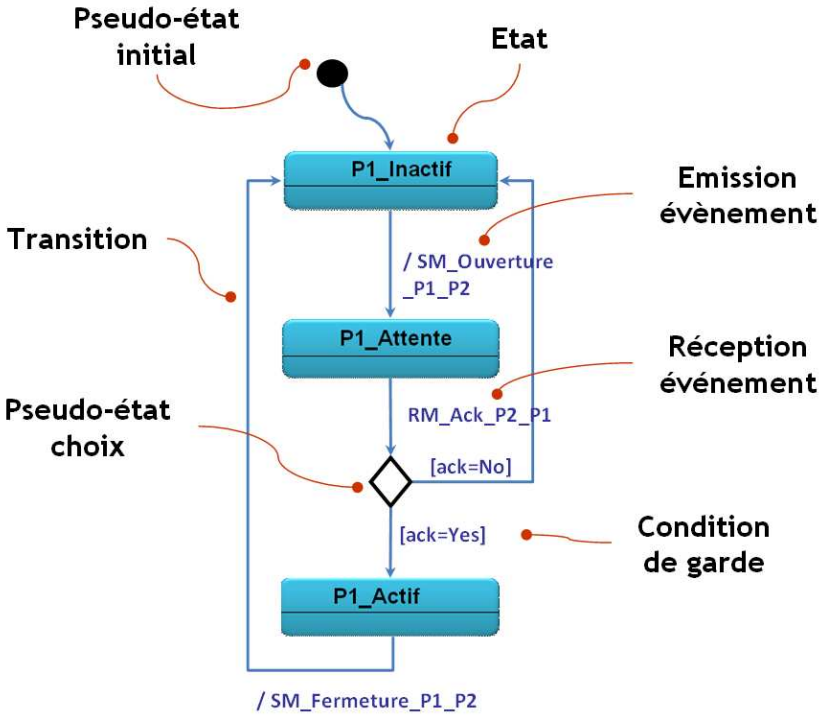


Figure 3.41: Exemple de machine d'états UML

Etat. Un état peut être caractérise de trois manières : comme un jeu de valeurs d'objet qualitativement similaires ; comme un laps de temps pendant lequel un objet attend qu'un ou plusieurs évènements se produisent ; ou comme un laps de temps pendant lequel un objet accomplit une ***activité do*** (faire). La machine d'états contient un jeu d'états connectés par des transitions. Une ***transition*** relie un état source à un état cible. Un état est représenté par un rectangle aux angles arrondis (cf. Fig.3.41). Un seul état est actif à un instant donné par composant. Un état peut être composite et correspond à une situation complexe contenant des sous-états.

Transition. Une transition qui quitte un état source possède, en règle générale, un ***déclencheur d'évènement***, une ***condition de garde***, un ***effet*** et un état cible. Quatre types de

transitions existent : les transitons externes, les transitions internes, les transitions entry et les transitions exit. Le type le plus répandu que nous utilisons dans nos travaux de thèse est la transition externe. Il s'agit d'une réponse à un évènement qui engendre un changement d'état ou une auto-transition, ainsi qu'un effet spécifié. Elle est représentée par une flèche allant de l'état source vers l'état cible, avec d'autres propriétés qui s'affichent sous la forme d'une chaîne textuelle attachée à la flèche (cf. Fig.3.41 & cf. Fig.3.42).

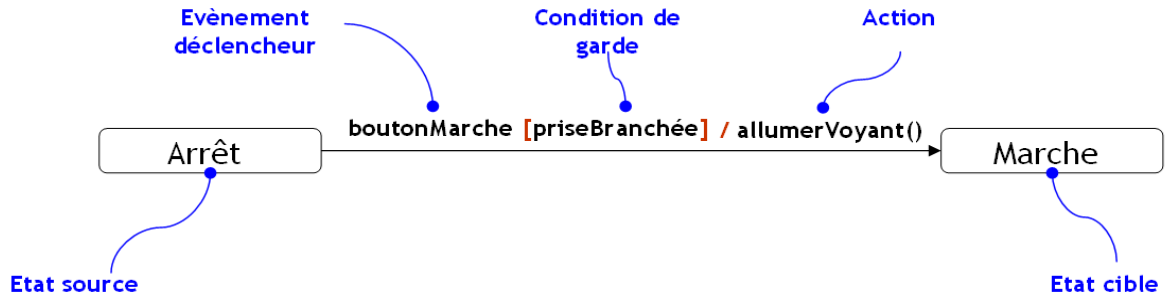


Figure 3.42: Transition

Evènement déclencheur. Le déclencheur spécifie l'évènement qui active une transition. L'évènement peut posséder des paramètres disponibles pour un effet spécifié comme partie de la transition. Par exemple, si le déclencheur d'une transition est un accusé de réception, la réception de cet évènement déclenche la transition (cf. Fig.3.41). Un évènement n'est pas un élément continu ; il se produit à un moment précis. Un objet ne gère qu'une occurrence d'évènements à la fois. Une transition doit se déclencher au moment où l'objet traite l'évènement.

Condition de garde. Une transition peut avoir une condition de garde, qui est une expression booléenne. Elle peut désigner les attributs de l'objet qui détient la machine d'états ainsi que les paramètres de l'évènement déclencheur. La condition de garde est évaluée lorsque l'évènement déclencheur se produit. Si l'expression s'évalue comme étant vraie, la transition se déclenche, c'est-à-dire que ses effets se produisent. Si l'expression s'évalue comme fausse, alors la transition ne se déclenche pas. La condition de garde ne s'évalue qu'une seule fois, au moment où l'évènement déclencheur se produit. Si la condition est fausse et que plus tard, elle devient vraie, il est trop tard pour déclencher la transition. Un même évènement peut être déclencheur de plusieurs transitions mais chacune doit avoir une condition de garde différente. Si l'évènement se produit, une transition déclenchée par l'évènement peut se déclencher si sa condition est vraie.

Effet. Lorsqu'une transition se déclenche, son effet s'exécute. Il s'agit d'une action ou d'une activité (cf. Fig.3.42). Une action est un calcul primitif, comme une instruction d'assignation ou un simple calcul arithmétique. On trouve comme exemples d'autres actions, l'envoi d'un

signal à un autre objet (cf. Fig.3.41), l'appel d'une opération, la création ou la suppression d'un objet et l'obtention et la définition des valeurs d'attributs. Un effet peut aussi être une activité, à savoir une liste d'actions ou d'activités plus simples. Une action ou une activité ne peut pas s'achever ou être affectée par des effets simultanés. Ainsi, il n'est pas possible de traiter un second évènement pendant l'exécution d'un effet. Une activité qui a démarré, doit s'achever et ne pas interagir avec d'autres effets actifs simultanément. Il s'agit de la sémantique dite *run-to-completion*. En revanche, il ne faut pas utiliser les effets comme un long mécanisme de transaction. Leur durée doit être courte par rapport au temps de réponse nécessaire aux évènements externes. Lorsque l'exécution d'un effet est terminée, l'état cible de la transition devient actif (cf. section 3.3.3.3 qui explique la règle de transformation d'une transition UML avec effet en réseau de Petri et qui montre le respect de synchronisation avant de passer à l'état cible).

Pseudo-état initial. Le pseudo-état initial est un état qui définit le point de départ par défaut pour la machine d'états. Il est représenté par un rond noir (cf. Fig.3.41).

Pseudo-état final. Le pseudo-état final est un état qui indique que l'exécution de la machine d'états ou du sous –état est terminée. Il est représenté par un rond noir entouré de cercle.

Pseudo-état choix. Le pseudo-état choix est un nœud de la machine d'états au niveau duquel on réalise une évaluation dynamique des conditions de garde ultérieures (cf. Fig.3.41). Il s'agit d'un point de décision possédant une entrée et au moins deux sorties. Les gardes situées après ce point sont évaluées au moment où il est atteint.

La description des machines d'états que nous venons de présenter nous servira pour définir nos règles de transformation vers les réseaux de Petri interprétés sachant que nous ne transformons ici que les éléments dont nous avons besoin dans le contexte de notre étude et qui correspondent au système ERTMS. La transformation des autres éléments des machines d'états est prévue en perspectives de ces travaux de thèse. Le méta-modèle UML des machines d'états est fourni en annexe E. Quant à notre choix des réseaux de Petri comme formalisme cible et plus particulièrement la classe des RdP interprétés, nous l'argumentons dans la section suivante.

3.3.2 Modèle cible : les réseaux de Petri

Le choix du formalisme des réseaux de Petri comme modèle formel cible s'est imposé grâce à sa base mathématique solide associée à son caractère graphique. Ce formalisme possède également de nombreux outils d'aide, de vérification et de validation, ce qui est en parfaite harmonie avec nos objectifs de recherche [Jabri et *al*, 2009a].

3.3.2.1 Présentation

Les Réseaux de Petri (ou RdP) [Murata, 1989] sont des outils graphiques et mathématiques permettant de modéliser le comportement dynamique des systèmes réactifs et concurrents. Ils constituent un support pour la spécification et la conception de systèmes temps réel automatisés tels que les réseaux de transport. Les réseaux de Petri sont des graphes orientés comprenant deux types de sommets liés entre eux par des arcs orientés : les places et les transitions. Une place correspond à une variable d'état du système qui va être modélisée et une transition à un événement et/ou une action qui va entraîner l'évolution des variables d'état du système [Gamatié, 2004]. A un instant donné, une place contient des marques ou jetons, qui vont évoluer en fonction du temps : un jeton indique la valeur de la variable d'état à cet instant. Quand un arc relie une place à une transition, cela indique que la valeur de la variable d'état associée à la place influence l'occurrence de l'événement associé à la transition. Quand un arc relie une transition à une place, cela veut dire que l'occurrence de l'événement associé à la transition influence la valeur de la variable d'état associée à la place [Murata, 1989].

Définition 3.3.2.1.a/ **Réseau de Petri** [Murata, 1989]. Un réseau de Petri ($\mathfrak{R}$= ($\mathbb{P}$, $\mathbb{T}$, $\mathcal{W}$), m_0) est un graphe biparti pondéré où:

- $\mathbb{P} = \{p_1, p_2 \ldots p_M\}$ un ensemble fini de places avec M= $\|\mathbb{P}\|$. Les places sont représentées par des cercles ;
- $\mathbb{T} = \{t_1, t_2 \ldots t_N\}$ un ensemble fini de transitions avec N= $\|\mathbb{T}\|$. Les transitions sont représentées par des rectangles ;
- $\mathcal{W}$: $(\mathbb{P} \times \mathbb{T}) \cup (\mathbb{T} \times \mathbb{P}) \rightarrow \mathbb{N}$ associe à chaque arc (p, t) (pour (place, transition)) ou (t, p) (pour (transition, place)) un poids $\boldsymbol{\mathcal{W}(p, t)}$ ou $\boldsymbol{\mathcal{W}(t, p)}$. Quand il n'existe aucun arc entre la place p et la transition t, nous avons $\mathcal{W}(p, t) = \mathcal{W}(t, p) = 0$;
- m_0: $\mathbb{P} \rightarrow \mathbb{N}$ associe à chaque place $p \in \mathbb{P}$ un entier $\boldsymbol{m_0(p)}$ appelé marquage de la place p. Les marquages sont appelés des jetons contenus dans les places.
- Pre: M × N est la matrice Precondition définie par :
$$\forall \boldsymbol{p} \in \mathbb{P},\ \forall \boldsymbol{t} \in \mathbb{T}, \boldsymbol{Pre(p, t) = \mathcal{K} \leftrightarrow \mathcal{W}(p, t) = \mathcal{K}}$$
- $Post$: M × N est la matrice Postcondition définie par :
$$\forall \boldsymbol{p} \in \mathbb{P},\ \forall \boldsymbol{t} \in \mathbb{T}, \boldsymbol{Post(p, t) = \mathcal{K} \leftrightarrow \mathcal{W}(t, p) = \mathcal{K}}$$

Exemple (RdP). Un exemple de réseau de Petri est représenté dans la figure 3.43. Nous avons : $\mathbb{P} = \{p_1, p_2, p_3\}, \mathbb{T} = \{t_1, t_2, t_3, t_4\}, \mathcal{W}(p_1, t_2) = 1, \mathcal{W}(p_2, t_1) = 1, \mathcal{W}(p_2, t_3) = 1, \mathcal{W}(p_3, t_4) = 1, \mathcal{W}(t_1, p_1) = 1, \mathcal{W}(t_2, p_2) = 1, \mathcal{W}(t_3, p_3) = 1, \mathcal{W}(t_4, p_2) = 1, m_0(p_1) = 0, m_0(p_1) = 2$ *et* $m_0(p_3) = 0$.

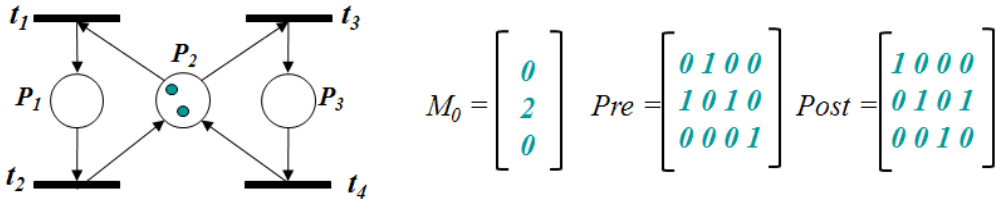


Figure 3.43: Exemple de réseau de Petri

Dans un réseau de Petri, le marquage des places représente l'état du système correspondant à un moment donné. Ce marquage peut être modifié en fonction du franchissement des transitions.

Définition 3.3.2.1.b/ **Franchissement de transition**. Une transition est franchissable pour un marquage m_0 (notée $m_0[t\rangle$), **si** $\forall p \in \mathbb{P}, m_0(p) \geq W(p,t)$. Si cette condition est satisfaite, alors un nouveau marquage m_1 est créé à partir du marquage m_0 (noté $m_0\,[t\rangle m_1$):

$$\forall p \in \mathbb{P}, m_1(p) = m_0(p) - W(p,t) + W(t,p).$$

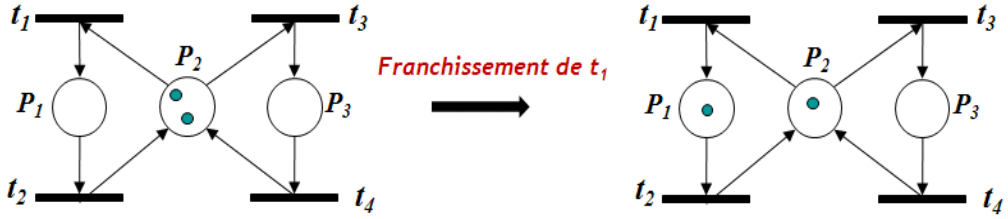


Figure 3.44: Franchissement d'une transition

3.3.2.2 Les apports des réseaux de Petri

Les RdP ont l'avantage de fournir des représentations graphiques qui facilitent la compréhension des spécifications des systèmes. Ces représentations graphiques permettent de visualiser d'une manière naturelle le parallélisme, la synchronisation, le partage des ressources et les choix. Quant au formalisme mathématique, il permet d'établir des équations d'état, à partir desquelles il est possible d'apprécier les propriétés du modèle et de les comparer avec le comportement du système modélisé [Vernadat, 2001]. Les méthodes de recherche de propriétés dans les réseaux de Petri sont basées sur l'élaboration du graphe des marquages accessibles, sur l'algèbre linéaire (calcul des invariants de places et des transitions), la réduction des réseaux ainsi que sur la logique linéaire (caractérisation des relations d'ordre partiel).

Le RdP a l'avantage d'être exécutable. En effet, le marquage permet de représenter l'état d'un système dynamique à événements discrets, où les places sont généralement associées aux différentes variables d'état du système, tandis que les transitions sont associées aux événements susceptibles de modifier cet état. L'évolution du marquage permet de simuler la dynamique du système.

Un autre avantage des RdPs est leur capacité à représenter le temps. En effet, afin de prendre en compte des contraintes temporelles quantitatives, plusieurs extensions temporelles des réseaux de Petri existent. Le temps peut être associé aux transitions, aux places ou aux arcs. Finalement, ce formalisme offre également l'avantage de modélisation par décomposition. Un système complexe dans ce cas-là peut être décomposé en sous-systèmes où chaque sous système est modélisé par un RdP. Le RdP du système complet est alors obtenu en assemblant les RdPs des différents sous-systèmes. Cet assemblage se fait selon deux approches : soit par fusion des places communes aux différents RdPs quand il s'agit de ressources partagées entre les sous-systèmes, soit par fusion des transitions communes aux différents RdPs dans le cas où les sous-systèmes évoluent sous l'occurrence de mêmes évènements.

Dans le contexte de notre étude, nous avons choisi la classe des RdP interprétés pour la représentation de la spécification ERTMS. Cette classe de RdP permet de modéliser les systèmes réactifs. De plus sa sémantique est proche de la sémantique des machines d'états UML. Dans la section suivante, nous détaillons les arguments de notre choix et nous définissons une sous classe de RdP interprétés.

3.3.2.3 Les réseaux de Petri interprétés

La modélisation UML du contexte ERTMS définit la spécification des différents composants réactifs. Dès que l'on spécifie le comportement d'un composant en interaction avec son environnement externe via des entrées (signaux, capteurs…) et des sorties (actionneurs), le modèle est enrichi d'une interprétation traduisant ces interactions. Il s'agit de réseaux de Petri interprétés (ou RdPI) ([Moalla, 1981], [David&Alla, 1992]). L'utilisation de cette classe de réseau de Petri de haut niveau possède deux avantages principaux. Tout d'abord, le fait de disposer d'un langage de description de haut niveau permet d'enrichir la modélisation. Ensuite, toutes les méthodes disponibles pour les formalismes de plus bas niveau peuvent être réutilisées.

Les réseaux de Petri interprétés (cf. Annexe F) sont des réseaux où le franchissement des transitions est conditionné par des événements extérieurs. Ils permettent de décrire des systèmes de contrôle-commande temps réel où la partie commande définit les ordres et la partie opérative réagit sous la forme de comptes rendus. Les ordres correspondent aux actions associées aux différentes places et agissant sur des variables d'état du système. Quant aux comptes rendus, ils sont envoyés sous la forme d'évènements associés aux transitions. Le franchissement de ces transitions est conditionné par des prédicats. Ceci correspond parfaitement à la sémantique des machines d'états UML (cf. Section 3.3.1) qui représentent la modélisation de la spécification ERTMS.

La classe de RdP que nous avons développé lors de cette thèse ***est une sous-classe des RdP interprétés*** [Jabri et *al*, 2010] (cf. Annexe F pour les RdP interprétés). Nous reprenons la définition 3.3.2.1 d'un RdP classique et nous l'étendons de la manière suivante :

Définition 3.3.2.3 / **Sous-classe de RdP interprété**. Notre classe **RdP** consiste en la donnée d'un réseau ordinaire $(\Re = (\mathbb{P}, \mathbb{T}, \mathcal{W}), m_0)$, d'un environnement $(\mathbb{V}, \mathbb{D}_k)$ et de deux fonctions μ et ∝, où :

$\mathbb{P} = \{p_1, p_2 \ldots p_M\}$ est l'ensemble des places avec M = |ℙ| est le cardinal de ℙ ;

$\mathbb{T} = \{t_1, t_2 \ldots t_N\}$ est l'ensemble des transitions avec N = |𝕋| est le cardinal de 𝕋 ;

Pre, Post ∈ $\mathbb{N}^{M \times N}$;

- ***Variables***

Soit $\mathbb{V} = \{v_1, v_2 \ldots v_V\}$ l'ensemble des variables avec V = |𝕍| ;

$\forall k \in [\![1, V]\!]$, on définit un ensemble fini $\mathbb{D}_k$ de valeurs pour chaque variable v_k. $\mathbb{D}_k = \{d_{k1}, d_{k2} \ldots d_{kD_k}\}$ avec $\mathcal{D}_k = |\mathbb{D}_k|$.

- ***Opérations***

Soit $\mathcal{O}_{ki}$ une opération élémentaire ;

$\mathcal{O}_{ki}$ est une expression sous la forme : $v_k \leftarrow d_{ki}$ *avec* $i \in [\![1, \mathcal{D}_k]\!]$;

Soit 𝕆 l'ensemble des opérations $\mathcal{O}_{ki}$, $k \in [\![1, V]\!]$ *et* $i \in [\![1, \mathcal{D}_k]\!]$.

- ***Prédicats***

Un prédicat est une expression construite à partir des règles définies dans la grammaire formelle suivante :

$$< clause > \rightarrow < \mathcal{R}_{ki} > \mid \neg < \mathcal{R}_{ki} >$$

$$< conjonction > \rightarrow < clause > \wedge < prédicat >$$

$$< prédicat > \rightarrow < clause > \wedge < conjonction >$$

$\mathcal{R}_{ki}$ est une expression élémentaire sous la forme :

$$v_k == d_{ki} \text{ avec } i \in [\![1, \mathcal{D}_k]\!] \text{ et } k \in [\![1, V]\!] ;$$

Soit ℝ *l'ensemble des prédicats.*

- ***Fonctions d'étiquetages***

Soit μ : 𝕋 → 𝕆, une fonction d'étiquetage qui associe à chaque transition t une opération $\mathcal{O}_{ki}$. μ (t_i) ∈ 𝕆.

Soit ∝: 𝕋 → ℝ, une fonction d'étiquetage qui associe à chaque transition t un prédicat. α (t_i) ∈ ℝ.

Ces fonctions d'étiquetage graphique permettent d'associer à chaque transition dans le réseau de Petri une opération et/ou un prédicat comme le montre la figure 3.45. La fonction $\propto (t_i)$ se situe juste avant la transition et la fonction $\mu(t_i)$ se situe juste après la transition.

Sémantique. Le marquage du réseau interprété que nous venons de décrire évolue de la même manière qu'un RdP classique. La sémantique de franchissement se base sur les prédicats associés aux transitions et aux opérations d'affectations qui s'exécutent lors du franchissement des transitions. Une transition $t \in \mathbb{T}$ n'est franchissable que lorsque le prédicat associé à cette transition par la fonction $\alpha\ (t)$ est vérifié par l'état actuel du système (prédicat vrai). Ensuite, cet état peut changer à travers l'opération associée à la transition t par la fonction $\mu\ (t)$. L'exécution de cette opération se fait une seule fois lors du franchissement de la transition et produit un nouvel état au réseau.

Exemple. Soit un réseau R constitué par deux places P_1 et P_2 et une transition t_1. L'ensemble de variables est représenté par v_1 et v_2. Soit D_1 le domaine de valeurs de la variable v_1 et D_2 le domaine de valeurs de la variable v_2. La transition t_1 ne peut être franchissable que si l'état actuel du système permet de vérifier le prédicat associé à la transition [(v_1==3) et (v_2==4) sont vrais]. Suite au franchissement de la transition t_1, le jeton de la place P_1 transféré à la place P_2 et une nouvelle valeur est affectée à la variable v_2. Un nouvel état du réseau est ainsi produit.

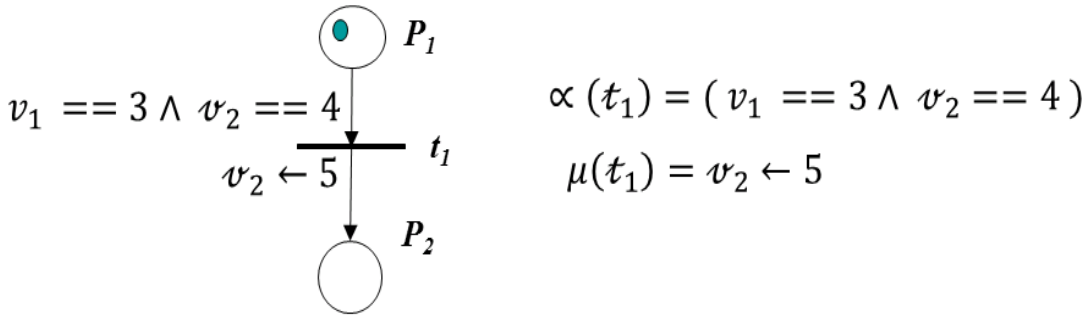


Figure 3.45: Exemple de RdP interprété

Méta-modèle. Suite à la définition de la sémantique de notre classe de RdP, nous avons utilisé les diagrammes de classes UML pour produire un méta-modèle simplifié de cette classe ([Jabri et *al*, 2009b] & [Jabri et *al*, 2010]). La figure 3.46 présente ce méta-modèle sous la forme de classes et de relations entre ces classes. Nous remarquons que la classe Transition est composée par une classe « Predicate » pour définir les prédicats et une classe « Operation » pour définir l'affectation. Ces deux relations de composition dans le diagramme de classe UML correspondent aux deux fonctions d'étiquetage $\propto$ et μ prédéfinies dans la sémantique.

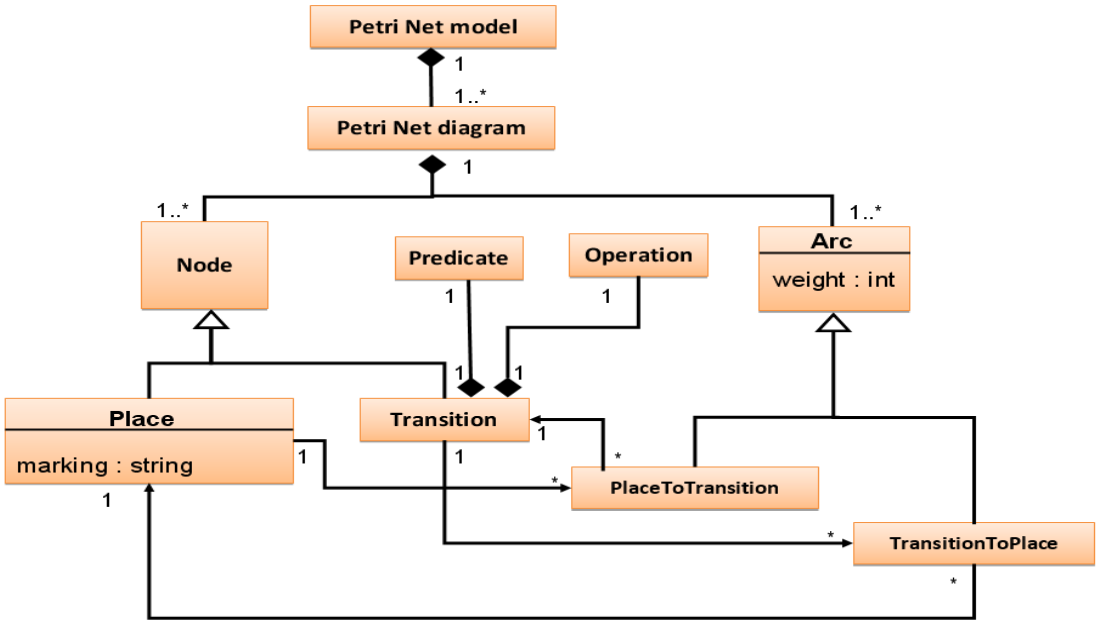


Figure 3.46: Méta-modèle simplifié des réseaux de Petri interprétés

Avant de détailler notre méthode de transformation de diagrammes d'états UML vers les diagrammes RdP, nous présentons brièvement dans ce qui suit, un aperçu des différentes études, dans la littérature, qui ont utilisé les deux formalismes UML et RdP.

3.3.2.4 État d'art sur le couplage UML et réseaux de Petri

Dans leurs travaux [King et Pooley, 1999], King et Pooley ont utilisé les réseaux de Petri stochastiques généralisés (GSPNs, Generalized Stochastic Petri Nets) ([Marsan et *al*, 1995] afin de représenter le comportement des statecharts (digrammes d'état UML). Ainsi, chaque état du statechart devient une place et chaque transition UML devient une transition dans le réseau de Petri correspondant. Dans le cas de plusieurs objets où chaque objet est modélisé par un statechart, les sous-modèles du réseau de Petri obtenus suite à la translation sont ensuite regroupés en utilisant les digrammes de collaboration UML. Dans le cas de nos travaux, le regroupement de nos sous-modèles se fait en utilisant les diagrammes de séquences UML.

Une autre approche de transformation de modèles UML est proposée par Merseguer et Bernardi ([Merseguer, 2004], [Bernardi et *al*, 2002]). Dans leurs travaux, des diagrammes UML étendus sont translatés en des modules GSPN étiquetés (Labelled Generalized Stochastic Petri Nets, LGSPNs), qui sont ensuite fusionnées pour former un modèle de réseau de Petri complet. Quant à l'étude faite dans [Trowitzsch et *al*, 2006], les auteurs ont eu l'idée de modéliser le comportement des systèmes techniques au moyen de statesmachines (machines à états) UML et du profil SPT (UML Profile for Schedulability, Performance, and Time). L'utilisation de ce profil permet de décrire certains aspects quantitatifs, tels que le temps et le

choix probabiliste. La technique de transformation de modèles est utilisée dans ce contexte afin de transformer les diagrammes UML en un modèle de réseau de Petri stochastique. L'objectif de cette transformation est de mesurer la performance par simulation ou par analyse numérique. Le système ERTMS/ETCS (European Railway Trafic Management System/ European Train Control System) sert d'étude de cas pour cette approche. La relation entre la qualité de la communication ETCS et la distance minimale entre les trains est analysée.

Dans les travaux de Lopez-Garo, Merseguer et Campos [Lopez-Garo et *al*, 2004], la vue dynamique du système a été modélisée en utilisant des diagrammes d'activité UML. Ensuite, vu le manque d'une sémantique formelle dans le langage UML, les auteurs ont choisi de traduire chaque diagramme UML en un LGSPN [Marsan et *al*, 1995]. Les LGSPNs sont ensuite rassemblés pour former un modèle analysable d'un système ou d'un scénario particulier. Enfin, les modèles obtenus sont analysés et simulés afin d'obtenir des indices de performance. Le formalisme LGSPN a été choisi en raison du nombre d'outils d'analyse et de simulation disponibles.

Dans [Bernardi et *al*, 2002], Bernardi, Donatelli et Merseguer ont utilisé les diagrammes de séquences et les diagrammes d'états transitions UML (statescharts) dans le but de valider et d'évaluer les performances des systèmes. Les auteurs considèrent que le comportement d'un système est représenté par un ensemble de diagrammes d'états transitions et que les diagrammes de séquence sont utilisés pour modéliser les communications entre les différents sous-systèmes. Ils jugent qu'il est impossible d'appliquer directement des techniques mathématiques sur les diagrammes UML vu leur manque d'une sémantique formelle. Ils proposent ainsi une traduction automatique de diagrammes d'états transitions et des diagrammes de séquence en des GSPNs. Le modèle GSPN obtenu doit permettre l'analyse des performances du système.

Dans [Delatour, 2003], Delatour propose une approche mixte de spécification de systèmes temps réel basée sur la notation UML et le formalisme des RdP. La notation UML est employée comme un langage semi-formel permettant de représenter toute la partie statique du système et d'apporter des aides précieuses à la structuration, deux aspects pour lesquels les RdP sont peu adaptés. En revanche, l'utilisation des RdP, en particulier les RdP objet PNO (Petri Net Object), comble une partie des lacunes d'UML, tant par leur richesse de description des aspects dynamiques (et tout particulièrement des contraintes temporelles) que par la formalisation de la modélisation. L'objectif de ces travaux est la validation de systèmes temps réel dans un cadre de conception orientée objet. Plus spécifiquement il s'agit de vérifier des contraintes temporelles imposées par un objet sur son environnement (par exemple délai de réponse borné). Ces objets sont issus de l'approche de conception temps

réel UML / PNO. L'idée principale consiste à appliquer une technique de réécriture basée sur la Logique Linéaire pour calculer sous forme symbolique les durées de scénario induites par la requête d'un objet.

La première contribution de nos travaux consiste à proposer une traduction des diagrammes d'états transitions (appelés states machines en UML 2.0) en des réseaux de Petri interprétés selon des règles de transformation respectant le méta-modèle UML (cf. Annexe E) et le méta-modèle RdP (cf. Fig.3.46). Les principales caractéristiques de notre approche sont : une traduction automatique des diagrammes UML en utilisant une technique de transformation de modèles et une génération automatique de scénarios de test à partir du modèle réseau de Petri. Dans notre cas, le modèle de réseau de Petri obtenu suite à la transformation n'a pas l'objectif de vérifier si le modèle est bien spécifié mais plutôt si la spécification modélisée est bien respectée.

Dans ce qui suit nous détaillons notre démarche de transformation de modèles UML (diagrammes d'états) vers un modèle réseau de Petri (réseaux de Petri interprétés). Nous commençons tout d'abord par la description du principe de la méthode développée. Ensuite nous définissons les règles de transformations développées.

3.3.3 Méthode de transformation développée

3.3.3.1 Principe

La modélisation des spécifications ERTMS fait intervenir plusieurs acteurs. Ainsi pour modéliser une procédure ou un scénario, il a fallu modéliser le comportement de chaque objet intervenant dans une procédure de fonctionnement donnée. Cette identification des acteurs permet de définir les cas d'utilisations UML sachant qu'un cas d'utilisation détermine une séquence d'actions accomplies par le système et produisant un résultat observable par l'un des acteurs. Une procédure de fonctionnement du système ERTMS correspond ainsi à un cas d'utilisation UML (UML Use case diagram [OMG, 2005]).

Nous avons choisi de modéliser un cas d'utilisation à travers les différentes machines d'états UML (cf. Section 3.3.1) décrivant le comportement dynamique de chaque acteur intervenant dans ce cas d'utilisation. Le regroupement des diagrammes de machines d'états de chaque acteur (objet) permet de fournir toutes les informations nécessaires concernant la procédure de fonctionnement. Dans le cas du système ERTMS, la vérification du composant EVC (European Vital Computer), en tant que composant réactif interagissant avec l'environnement extérieur, exige la modélisation du comportement de chaque objet communicant avec ce dernier. Le comportement dynamique d'un objet O_1 est modélisé à travers d'un diagramme de machines d'états UML. L'objet O_1 peut changer d'un état à un autre par la réception d'un message émis par un autre objet O_2 (évènement déclencheur, cf. Section 3.3.1.4). Le passage d'un état à un autre peut engendrer l'envoi d'un message de la

part de O_1 vers O_2 (effet, cf. Section 3.3.1.4). Le diagramme de séquences UML (cf. Section 3.3.1) permet d'identifier ces différentes interactions entre les objets. Le digramme de classes UML (cf. Section 3.3.1) permet de modéliser la structure statique du système en identifiant les propriétés de chaque objet.

La transformation des diagrammes d'états UML en RdP se base sur la définition de variables et de leurs valeurs. ***En effet, dans le contexte ERTMS, nous ignorons le comportement interne de chaque composant étant donné que la spécification nous informe uniquement sur les interactions de chaque composant avec son environnement. Ces interactions sont considérées comme des transitions aux interfaces entre les RdP de chaque composant*** (cf. Fig.3.47).

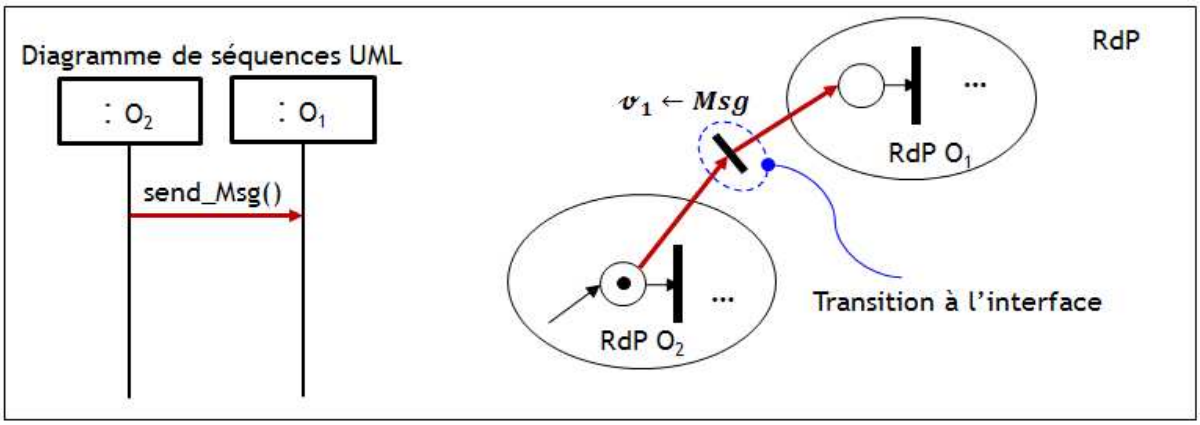


Figure 3.47: Interactions entre objets

La logique des messages échangés entre les objets UML est supportée par les variables étant donné qu'à chaque message UML est associée une variable dans le RdP. La partie dynamique est supportée par les jetons. Dans la figure 3.47, le jeton du RdP de O_2 peut indiquer qu'un message est arrivé à l'objet O_1. En effet, lors du franchissement de la transition à l'interface entre O_1 et O_2, ce jeton est transféré au RdP de O_1 indiquant ainsi qu'un message est arrivé produisant ainsi un évènement dans le RdP de O_1. La valeur du message transmis est affectée à une variable v_1 au niveau de la transition à l'interface (cf. Fig.3.47).

Quant aux différentes propriétés de chaque objet (identifiées dans le diagramme de classes UML), elles sont transformées en variables dans le RdP. Il y a autant de variables dans le RdP qu'il y a de propriétés dans le diagramme de classes UML. Les propriétés peuvent changer de valeur au cours d'une activité interne d'un état dans un diagramme de machines d'états UML. L'affectation d'une nouvelle valeur à une variable donnée est représentée dans le RdP par une opération d'affectation associée à une transition RdP (cf. Définition 3.3.2.3). Quant aux conditions de garde pouvant se trouver sur les transitions d'un diagramme de machines d'états UML, elles sont transformées en prédicats associés aux transitions RdP (cf. Définition 3.3.2.3). La transformation détaillée de ces différents éléments est fournie dans la section suivante.

Exemple. La procédure SOM (Start Of Mission) qui permet le démarrage du train, fait intervenir trois acteurs: EVC (European Vital Computer), conducteur et RBC (Radio Block Center). Ces trois acteurs communiquent entre eux à travers l'échange de messages. La transformation de ces diagrammes UML en RdP permet d'obtenir trois graphes différents (RdP de l'EVC, RdP du conducteur et RdP du RBC). Rassemblés, ces trois graphes forment le modèle RdP global de la procédure (cf. Fig.3.48).

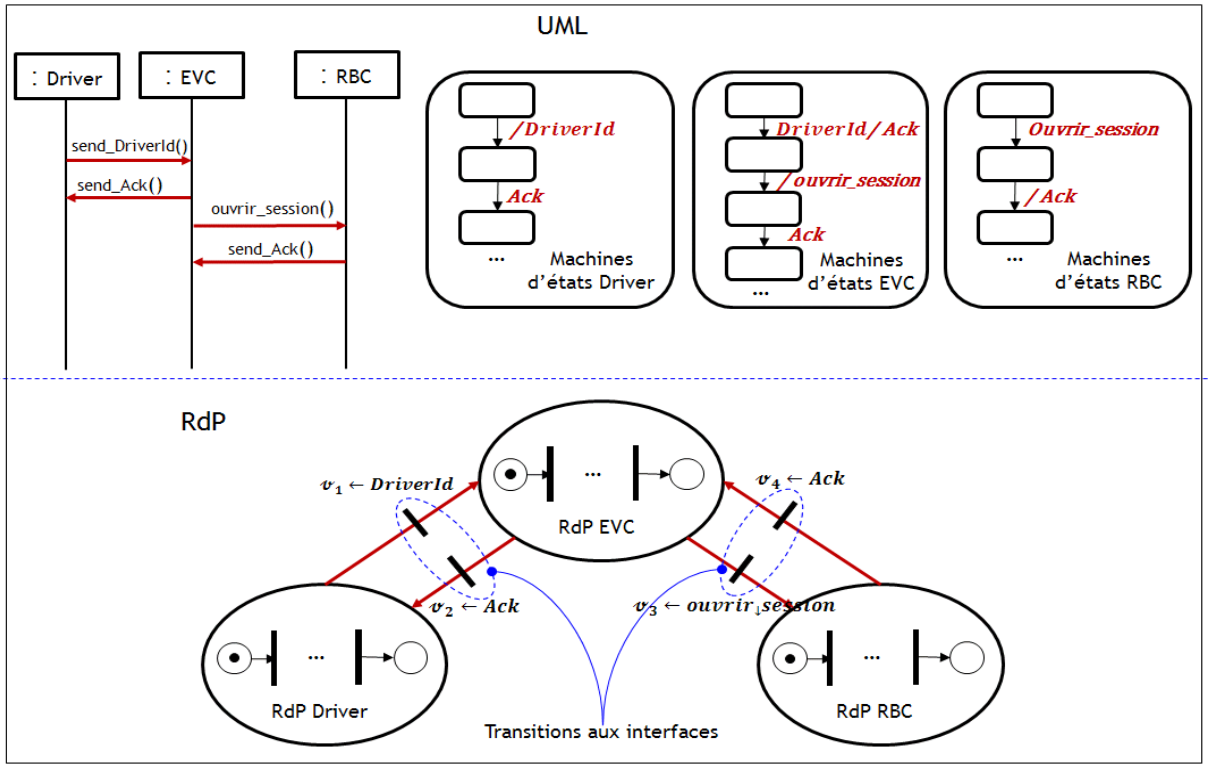


Figure 3.48: Exemple d'interactions entre EVC, Driver et RBC

Le rassemblement des RdP se fait grâce aux places communes entre ces graphes. Les places communes correspondent aux évènements déclencheurs et aux effets qui sont associés aux transitions des machines d'états UML. Dans le cas du système ERTMS, il s'agit plutôt de l'envoi et de la réception de messages. Par exemple dans la figure précédente, un message ***DriverId*** est reçu par le composant « ***EVC*** » (un évènement déclencheur dans le diagramme de machines d'états de l'EVC). Ce même message a été émis par le « ***Driver*** » (un effet ou action dans le diagramme de machines d'états du driver). Nous retrouvons le même principe pour la communication entre « EVC » et « RBC » (cf. Fig.3.48). La transformation des trois diagrammes d'états UML produit trois RdPs pour chaque composant. Le RdP de l'EVC possède une place correspondant au ***message reçu DriverId*** (modélisé sur la transition UML). Le RdP du driver possède aussi une place correspondant au ***message émis DriverId*** (modélisé sur la transition UML). Ces places qui correspondent à la réception et l'envoi d'un ***même message*** permettent finalement de regrouper les RdPs et de créer les transitions aux interfaces entre les RdPs des objets communicants. En se basant sur ce principe, ces RdPs

peuvent être ainsi rassemblés afin de former un modèle global. Dans la section suivante, nous détaillons nos règles de transformations des machines d'états UML vers des RdP interprétés.

Remarque. Nous avons étudié la possibilité d'utiliser les réseaux de Petri colorés [Jensen, 1992] comme formalisme cible de notre méthodologie de transformation. Il s'agit en effet de réseaux de Petri de haut niveau qui apportent une sémantique de différentiation des jetons. A chaque place et à chaque transition est associé un domaine de couleur qui permet, pour le cas des places, de colorer les marques pouvant être contenues dans ces places, et pour le cas des transitions, de choisir une couleur de franchissement. Lors du franchissement des transitions, les jetons, ayant une valeur typée, peuvent changer de valeur grâce aux expressions associées aux arcs. Le franchissement des transitions peut être également conditionné par les valeurs de jetons grâces aux gardes. Ce type de réseau offre une modélisation hiérarchisée et modulaire. Nous avons opté pour les RdP interprétés car leur utilisation dans la formalisation de la spécification ERTMS est plus facile que les réseaux colorés. De plus, les réseaux interprétés sont plus proches des machines d'états UML que les réseaux colorés. Enfin, les réseaux interprétés sont souvent utilisés pour la modélisation des systèmes de contrôle-commande. Ceci est en parfaite harmonie avec notre contexte d'étude.

Lors de la transformation de nos machines d'états UML vers des RdPs interprétés, nous avons choisi les éléments UML que nous avons jugés, actuellement, nécessaires dans le cadre de la modélisation du système ERTMS. La première étape de notre démarche consiste à établir les correspondances possibles (cf. Fig.3.49) entre les éléments du méta-modèle source MM_{UML} (méta-modèle des machines d'états) et le méta-modèle cible MM_{RdP} (méta-modèle des réseaux de Petri interprétés) [Jabri et *al*, 2009b]. Dans la figure 3.49, ces correspondances sont visualisées par des flèches et sont formellement définies par la relation R suivante :

$$R \in MM_{UML} \times MM_{RdP}$$

Par exemple, la correspondance établie de la méta-classe « *PseudoState_Initial* » vers la méta-classe « *Place* » (cf. Fig.3.49) est formalisée par: $(PseudoState\_Initial, Place) \in R$. La règle de transformation qui détaille cette correspondance est fournie dans la section suivante. La figure 3.49 représente les grandes lignes de correspondances entre les éléments du méta-modèle UML et les éléments construisant le méta-modèle RdP. Par exemple, à la méta-classe Transition du méta-modèle UML correspond une sous-partie d'un diagramme RdP que nous avons appelé « PN part_T ». Dans la figure 3.50, cette sous-partie se compose d'une méta-classe « ***Transition To Place*** », d'une Méta-classe « ***Transition*** » et d'une méta-classe « ***Place To Transition*** » [Jabri et *al*, 2009b]. Les méta-classes « Transition To Place » et « Place To Transition » héritent de la méta-classe « Arc» d'après le méta-modèle RdP que nous avons fourni précédemment (cf. Fig.3.46).

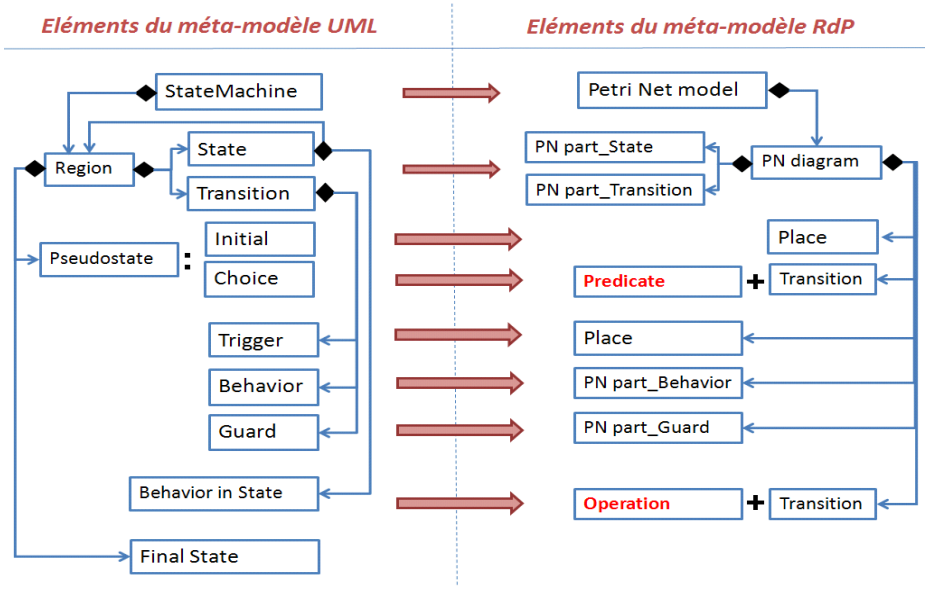


Figure 3.49: Correspondances entre les éléments des méta-modèles simplifiés

PNpart_Transition	:	Place to Transition + Transition + Transition to Place
PNpart_State	:	Place + PNpart_Transition + Place
PNpart_Behavior	:	Transition to Place + Place + PNpart_Transition + Places + PNpart_Transition + Place
PNpart_Guard	:	Transition + Predicate (= guard) + Transition + Predicate (= ! Guard)

Figure 3.50: Description des sous RdP de la figure 3.49

Définition 3.3.3.1.a/ Soient les fonctions d'abstraction $\mathbf{F}_{UML}$ et $\mathbf{F}_{RdP}$ qui associent à chaque élément issu d'une machine d'états UML et à chaque élément d'un diagramme RdP respectivement un méta-élément des méta-modèles respectifs $\mathbf{MM}_{UML}$ et $\mathbf{MM}_{RdP}$.

$$F_{UML} : A \rightarrow MM_{UML}$$
$$F_{RdP} : B \rightarrow MM_{RdP}$$

Où A dénote l'ensemble des éléments issus d'une machine d'états UML, et B l'ensemble des éléments issus d'un diagramme RdP.

Nous nous basons sur les méta-modèles MM_{UML} et MM_{RdP} ainsi que sur la relation de correspondances R en vue d'effectuer la construction de modèle RdP à partir du modèle UML. En effet, la traduction d'un élément d'une machine d'états UML en un élément RdP doit satisfaire la relation R entre les deux méta-classes associées aux deux éléments UML et RdP.

Définition 3.3.3.1.b/ Les correspondances entre les éléments UML et les éléments RdP sont définies par la relation Mapping comme suit :

$$Mapping \in A \times B \text{ tel que}$$
$$\forall (c, d), ((c, d) \in Mapping \Leftrightarrow (F_{UML}(c); F_{RdP}(d)) \in R)$$

Exemple. Nous prenons l'exemple de la figure suivante (cf. Fig.3.51). Dans cette figure, l'état initial E_i de la machine d'états de l'objet 1 est transformé dans le réseau de Petri en une place Obj1_init. D'après les définitions que nous venons de présenter, $E_i \in A$ et Obj1_init $\in B$.

De plus, F_{UML} (E_i)= PseudoState_initial ; F_{RdP} (Obj1_init) = Place. Or, nous avons déjà montré que (PseudoState_initial, Place) $\in R$; (E_i , Obj1_init) $\in$ *Mapping*.

En se basant sur ces différentes définitions, nous présentons dans la section suivante, les différentes règles de transformation afin de générer le modèle cible à partir du modèle UML.

3.3.3.2 Règles de transformation des états (« State »)

Une règle de transformation est définie comme l'unité de structuration dans les langages de transformation de modèles. Nous commençons par la transformation d'un état initial dans une machine d'états UML vers un diagramme RdP.

A- Etat initial

Définition : un état initial est un pseudo-état qui marque le début de la machine d'états (cf. section 3.3.1.4). Dans le diagramme UML, la transition qui quitte l'état initial ne peut porter ni un évènement déclencheur ni une condition de garde. Elle peut porter uniquement un effet.

Règle A : Un état initial est transformé en une place dans le réseau de Petri correspondant. Cette place contient un jeton par défaut. Son nom est composé du nom de l'objet modélisé suivi de « init » comme le montre la figure ci-dessous.

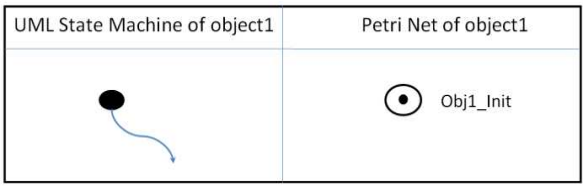


Figure 3.51: Transformation de l'état initial

B- Etat simple

Définition : un état est dit simple quand il ne possède pas de sous-structures mais uniquement un jeu de transitions et éventuellement des activités « entry », « do » et « exit » (cf. section 3.3.1.4).

Règle B1 : Un état simple qui ne comporte aucune activité est transformé en un sous-diagramme RdP composé par ***deux places et une transition*** : une ***place*** qui marque l'entrée dans l'état dont le nom commence par le nom de l'objet modélisé, suivi du nom de l'état, suivi de « ***entry*** »; une autre ***place*** qui marque la sortie de l'état dont le nom commence par le nom de l'objet modélisé, suivi du nom de l'état, suivi de « ***exit*** » et une ***transition*** entre les

deux. Comme le montre la figure 3.52, la transition qui lie l'état initial à l'état suivant dans le diagramme UML est transformée en une transition dans le RdP liant ainsi la place correspondant à l'état initial à la place marquant l'entrée dans l'état suivant.

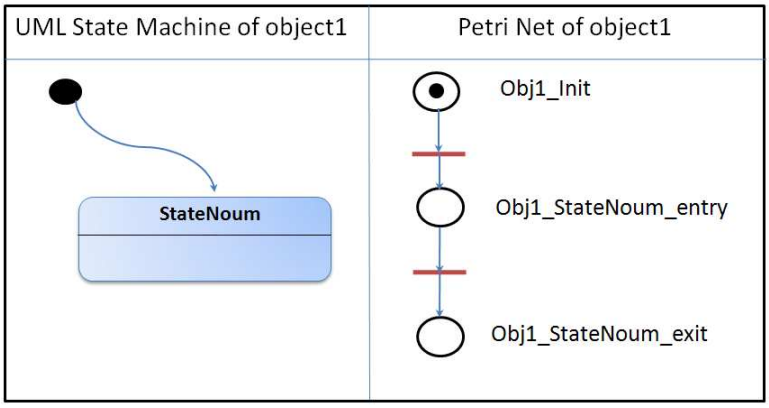


Figure 3.52: Transformation d'un état simple

Règle B2 : Un état simple qui comporte une ***activité interne*** « do activity » est transformé de la même manière que l'état précédent sauf qu'entre la place qui marque l'entrée de l'état et la place qui marque la fin de l'état, nous rajoutons un sous-diagramme RdP de la manière suivante : une ***place*** pour appeler la méthode contenue dans l'état dont le nom porte le nom de la méthode suivi de « ***call*** » ; une ***transition*** à laquelle on associe une ***opération*** permettant d'exécuter ***l'action*** de la méthode ; une ***place*** qui marque la fin de l'exécution de la méthode dont le nom porte le nom de la méthode suivi de « ***return*** » (cf. Fig.3.53).

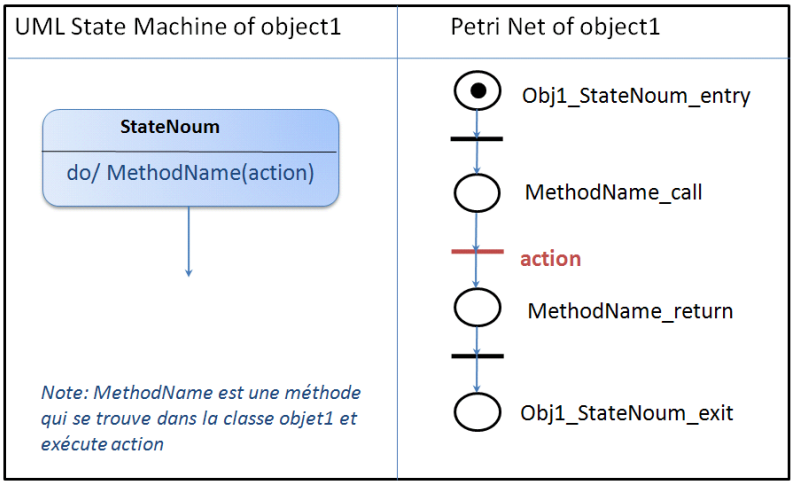


Figure 3.53: Transformation d'un état simple contenant une activité interne

C- Etat final

Règle C : Un état final (cf. section 3.3.1.4) n'a pas de correspondance dans le réseau de Petri. Tout simplement le RdP finit par la place qui marque la fin de l'état (place « obj1_StateNoum_exit » dans la figure 3.54) précédant l'état final dans la machine d'états UML.

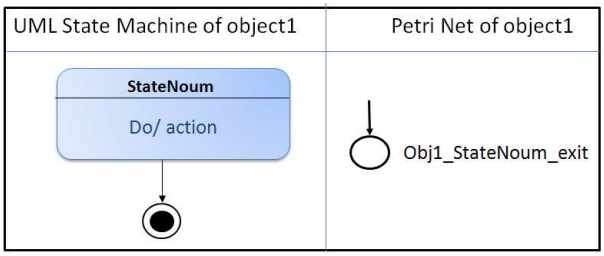


Figure 3.54: Transformation de l'état final

3.3.3.3 Règles de transformation des transitions («Transition »)

D- Simple transition

Définition : Comme nous l'avons définie dans la section 3.3.1.4, la ***transition*** permet de lier un ***état source*** à un ***état cible*** dans les machines d'états UML. Elle peut porter généralement un événement déclencheur, une condition de garde et un effet (d'action). Nous utilisons dans nos travaux des transitions externes (cf. section 3.3.1.4). Une transition est simple si elle ne comporte ni déclencheur ni condition de garde ni effet.

Règle D : Une transition simple T_{UML} liant un ***état source*** E_1 et un ***état cible*** E_2 est transformée en une transition T_{RdP} dans le diagramme RdP avec deux ***arcs***. Le premier arc permet de lier la ***place source*** (marquant la fin de l'état source E1) à la transition T_{RdP}. Le deuxième arc permet de lier la transition T_{RdP} à la ***place cible*** (marquant l'entrée dans l'état cible).

E- Transition contenant un évènement déclencheur (« Trigger »)

Règle E : Tout d'abord, une transition T_{UML} qui contient un évènement déclencheur ED est transformée en une transition T_{RdP} dans le diagramme RdP de la même manière que la règle D. Ensuite, *l'évènement déclencheur ED* qui représente dans le cadre de nos travaux, la réception d'un signal ou d'une communication, est transformé en un sous-diagramme RdP.

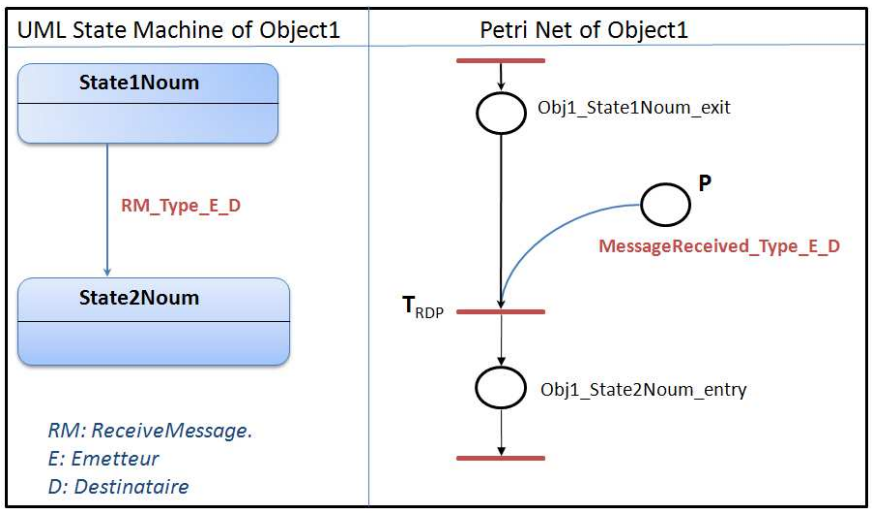


Figure 3.55: Transformation d'une transition comportant un déclencheur

La ***réception de ce signal*** de type « Type» par un émetteur « E » de la part d'un destinataire « D » est transformée dans le diagramme RdP de la manière suivante (cf. Fig.3.55):

- ✓ Une ***place*** P qui indique ***la réception du message***. Le nom de la place comporte : « MessageReceived », suivi du « Type » du message, suivi de l'émetteur « E », suivi du destinataire « D» ;
- ✓ Un ***arc*** liant la place P à la ***transition*** T_{RdP}.

F- Transition contenant un effet (« Behaviour »)

Règle F: Tout d'abord, une transition $\mathbf{T}_{UML}$ qui contient un effet ou une action est transformée en une transition $\mathbf{T}_{RdP1}$ suivie d'une ***place Intermédiaire*** P_I suivie d'une transition $\mathbf{T}_{RdP2}$ dans le diagramme RdP (cf. Fig.3.56). La place P_I (objet1_intermediate dans la figure 3.56) est une place de synchronisation qui marque l'attente de la fin d'exécution de l'action associée à la transition $\mathbf{T}_{UML}$. Ensuite, ***l'action « Ac »*** qui représente dans le contexte de nos travaux, l'envoi d'un signal ou d'une communication, est transformée en un sous-diagramme RdP.

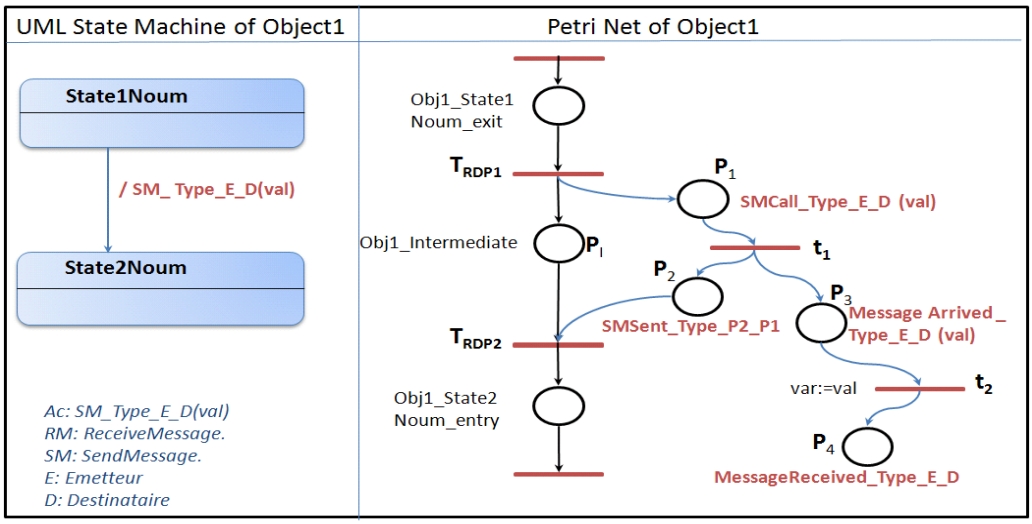


Figure 3.56: Transformation d'une transition contenant un effet

L'envoi d'un signal de type « Type» par un émetteur « E » pour un destinataire « D » est transformé dans le diagramme RdP de la manière suivante (cf. Fig. 3.56):

- ✓ Une ***place*** P_1 qui marque ***l'appel d'envoi du message***. Le nom de la place comporte : « SMCall », suivi du « Type » du message, suivi de l'émetteur « E », suivi du destinataire «D» ;
- ✓ Une ***place témoin*** P_2 qui indique que l'envoi du message a été effectué. Le nom de la place comporte : « SMSent », suivi du « Type » du message, suivi de l'émetteur « E », suivi du destinataire «D» ;
- ✓ Une ***place*** P_3 qui marque ***l'arrivée du message*** chez le destinataire. Le nom de la place comporte : « MessageArrived », suivi du « Type » du message, suivi de l'émetteur « E », suivi du destinataire «D» ;

- ✓ Une ***place*** P_4 qui indique ***la réception du message*** chez le destinataire. Le nom de la place comporte : « MessageReceived », suivi du « Type » du message, suivi de l'émetteur « E », suivi du destinataire « D» ;
- ✓ Un ***arc*** liant la transition T_{RDP1} à la place P_1 ;
- ✓ Un ***arc*** liant la place témoin P_2 la transition T_{RdP2} ;
- ✓ Un arc liant la place P_1 à une ***transition intermédiaire*** t_1 ;
- ✓ De la transition t_1, deux arcs sortent vers les places P_2 et P_3 ;
- ✓ Un arc liant la place P_3 à une deuxième transition intermédiaire t_2 ;
- ✓ Une ***opération*** est associée à la transition t_2 permettant d'affecter la valeur ***val*** envoyée dans le message, à la variable ***var***;
- ✓ Enfin un arc liant t_2 à la place P_4

Nous remarquons que la place dont le nom est composé par « MessageReceived », suivi du « Type » du message, suivi de l'émetteur « E », suivi du destinataire « D» ; est une place ***commune*** entre la règle F (place P_4 dans la figure 3.56) et la règle E (place P dans la figure 3.55) étant donné que l'évènement reçu par un objet est un effet réalisé par un deuxième objet. Ce type de places communes servira à rassembler les diagrammes RdP de deux objets communicants. Dans la figure 3.57, nous donnons l'exemple d'une transition contenant un évènement et un effet afin de montrer le concept de places communes. La place P représente un évènement pour l'objet1 et représente également un effet réalisé par un autre objet qui communique avec objet1. De même la place P4 représente un effet envoyé par objet 1 et un évènement reçu par un autre objet qui communique avec objet1. Il s'agit du principe de places communes.

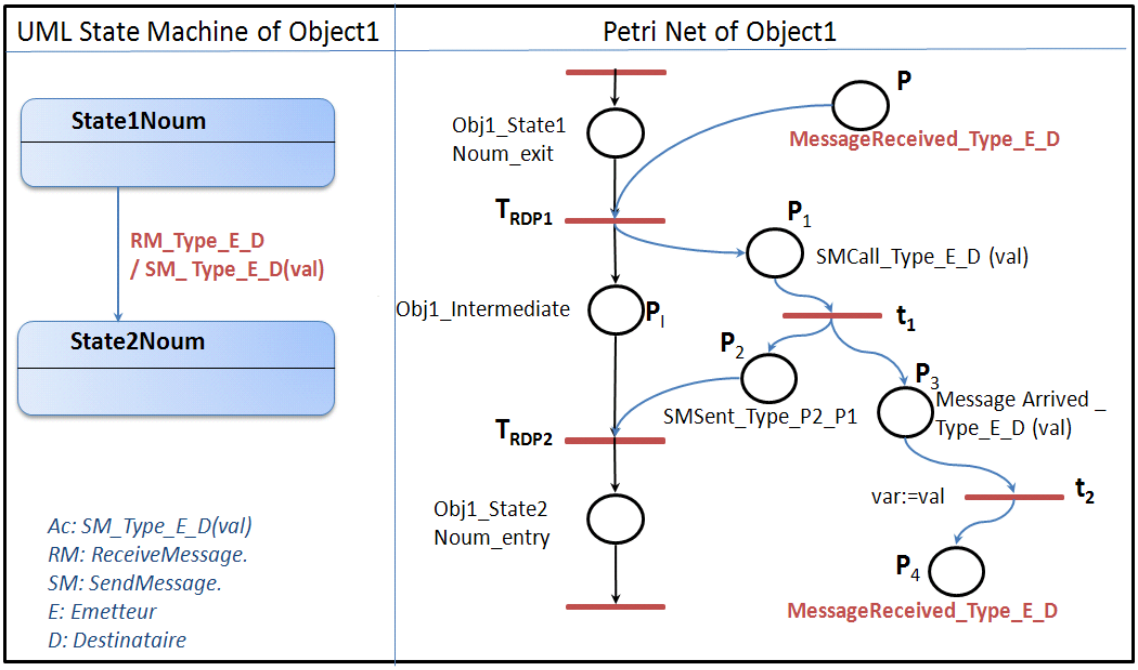


Figure 3.57: Transformation d'une transition contenant un évènement et un effet

G- Transition contenant un déclencheur et une condition de garde (« Guard »)

Règle G : Une transition T_{UML} qui contient un effet et une condition de garde ou une action est transformée en deux transitions T_{RdP1} et T_{RdP2} dans le diagramme RdP (cf. Fig.3.58). Un prédicat P_{r1} est associé à la transition T_{RdP1} pour présenter la condition de garde. Un autre prédicat P_{r2} est associé à la transition T_{RdP2} pour présenter le cas où la condition est fausse. Une condition fausse est représentée par « **!guard** ». La négation d'une condition a été prévue dans la description formelle de notre sous-classe de RdP interprété (cf. section 3.3.2.3). Le déclencheur est lié à la transition T_{RdP1} et également à la transition T_{RdP2}. la liaison avec la transition T_{RdP2} permet de se débarrasser du jeton de la place déclencheur dans le cas où la condition est fausse (cf. Fig.3.58). Ceci correspond parfaitement avec la sémantique de la machine d'états UML étant donné qu'on ne peut pas quitter l'état tant que la condition de garde n'est pas vraie.

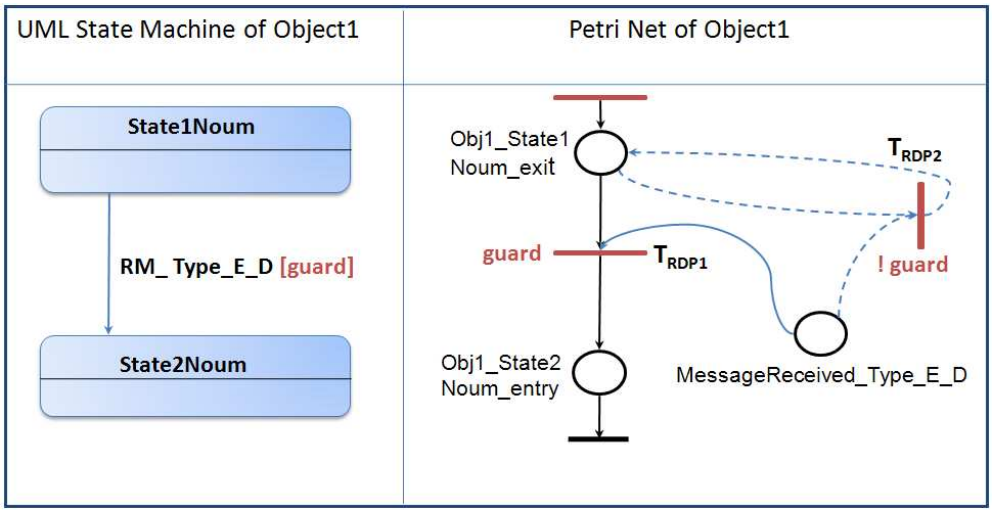


Figure 3.58: Transformation d'une transition contenant une condition de garde

Remarque. Dans le cas où d'un même état, nous avons deux transitions conditionnées par deux gardes, nous nous retrouvons avec un prédicat formé par une conjonction de fausses gardes (cf. transition T_{RDP3} dans la figure 3.59).

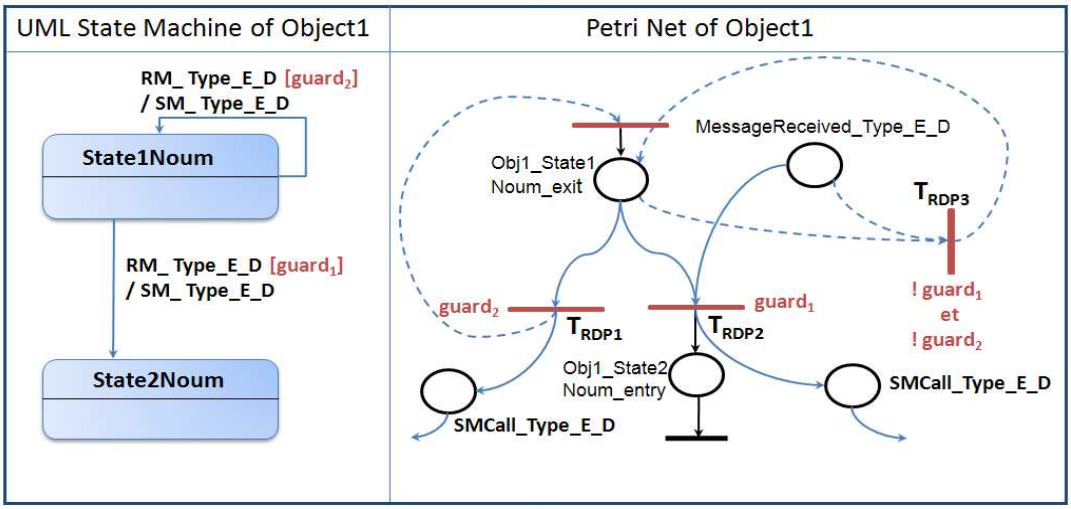


Figure 3.59: Conjonction de conditions de gardes

3.3.3.4 Règle de transformation du choix (« Choice »)

Règle H : Le choix est un pseudo-état permettant de tester des conditions de gardes sur des transitions. Le choix est transformé en deux transitions T_{RdP1} et T_{RdP2} dans le diagramme RdP (cf. Fig.3.60). Chacune des deux transitions possède un prédicat correspondant aux conditions du choix. Le déclencheur précédant le choix dans le diagramme UML est lié aux deux transitions T_{RdP1} et T_{RdP2} dans le diagramme RdP (cf. Fig.3.60).

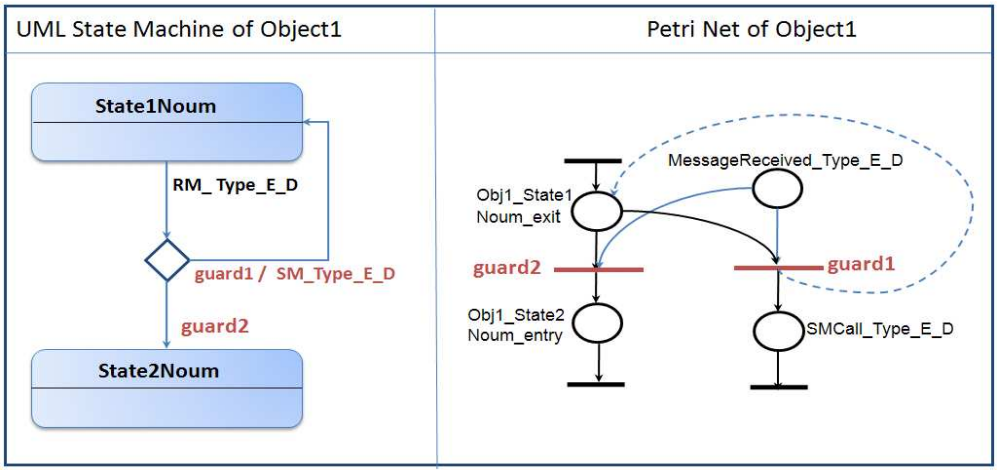


Figure 3.60: Transformation du choix

Les règles que nous venons de décrire nous ont servi à construire notre transformateur en langage **Prolog IV**[14]. Ce dernier est souvent utilisé dans les domaines de l'Intelligence Artificielle et la Programmation Logique avec Contraintes. Il est le premier langage de programmation logique. Ce mode de programmation s'oppose à la programmation impérative et la programmation fonctionnelle qui cherchent à obtenir un résultat grâce à l'application d'une procédure (une suite d'actions ou une composition de fonctions). La programmation logique est dite déclarative étant donné qu'elle ne s'occupe de pas de la

[14] http://prolog.developpez.com/cours/

manière d'obtenir le résultat. Cependant, le programmeur doit décrire le problème à résoudre tout en lisant l'ensemble des objets concernés, les propriétés et les relations qu'ils vérifient. Ensuite le processus de résolution parcourt toutes les possibilités du problème et peut ainsi retourner plusieurs solutions.

Le problème à résoudre est décrit à travers des prédicats. Un prédicat peut être un fait ou une règle. Les faits sont des données élémentaires que nous considérons vraies, ce sont les hypothèses de travail. Les règles sont des relations qui permettent à partir de ces hypothèses de travail de construire d'autres faits par déduction. Le compilateur Prolog utilisé pour exécuter nos règles de transformation est **GNU Prolog**[15] . Le code de notre transformateur UML vers RdP est fourni en Annexe G.

Dans la section suivante, nous présentons un petit exemple complet illustrant notre méthode de transformation.

3.4 Exemple illustratif

Dans cette section, nous présentons un exemple illustratif de la méthode développée pour la formalisation de spécifications ERTMS. Nous considérons deux processus P1 et P2 qui communiquent ensemble. P1 peut ouvrir une session de communication avec P2 et il reste en attente d'un accusé de réception. P2 envoie son accusé de réception. Si l'accusé de réception est favorable alors P1 devient dans un état actif et peut ensuite fermer la session de communication. P2 ne peut pas fermer la session.

3.4.1 Modélisation UML

L'ouverture et la fermeture de la session de communication entre P1 et P2 se fait à travers l'envoi de messages entre les deux objets. Nous considérons un diagramme de classes constitué par la classe « Processus » dont les attributs et les méthodes sont représentés dans la figure 3.61.

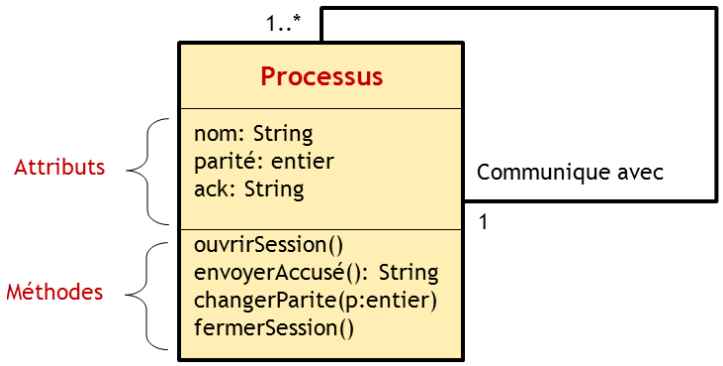


Figure 3.61: Diagramme de classes

[15] http://www.gprolog.org/

Un processus peut communiquer avec 1 ou plusieurs autres processus. Les objets P1 et P2 sont des instances de cette classe (cf. Fig.3.62).

Nous avons construit un diagramme de séquences UML qui permet de tracer les différents messages, envoyés entre les deux objets, dans le temps. Le diagramme de séquences est présenté dans la figure 3.62.

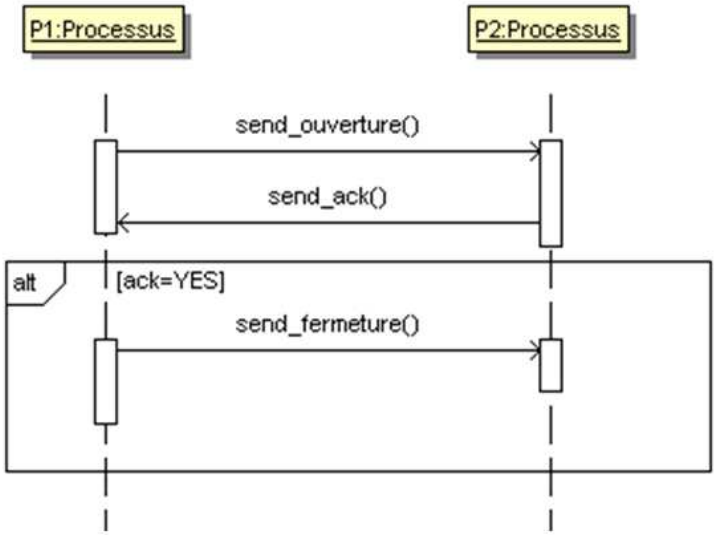


Figure 3.62: Diagramme de séquence UML.

Le comportement de chaque objet est décrit dans un diagramme de machines d'états UML. L'envoi et la réception des messages présentés dans le diagramme de séquence correspondent aux évènements reçus et aux actions effectuées par chaque objet (cf. Fig.3.63). Le scénario présenté dans la figure 3.63 correspond à l'enchainement des étapes suivantes :

- ✓ P1 est dans un état « Inactif », il envoie un message d'ouverture de session à P2 et passe à l'état « Attente ».
- ✓ De son côté P2 est dans un état « Inactif ». Il vérifié la valeur de la parité. Si la parité est égale à 0 il passe à un état « Affectation_A_1 » pour rendre la variable parité égale à 1. Idem dans le cas où la parité est égale à 1, P2 passe à un état « Affectation_A_0 ».
- ✓ Ensuite à la réception du message d'ouverture venant de P1, P2 envoie un accusé de réception. Si la parité est égale à 0, P2 n'ouvre pas la session, envoie un message à P1 contenant la valeur de l'accusé de réception (Ack=No) et revient à l'état inactif. Sinon, dans le cas où la parité vaut 1, P2 ouvre la session, envoie un message à P1 contenant la valeur de l'accusé de réception (Ack=Yes) et passe à l'état « Attente ».
- ✓ A la réception de l'accusé de réception, P1 vérifie la valeur de l'accusé. Si « Ack=No », il revient à l'état « Inactif », sinon il passe à l'état « Actif ».

- ✓ Dans un état « Actif », P1 peut fermer la session en envoyant un message de fermeture à P2 et il revient ensuite à l'état « Inactif ».
- ✓ Dans l'état « Actif » et à la réception d'un message de fermeture, P2 ferme la session et revient à l'état « Inactif ».

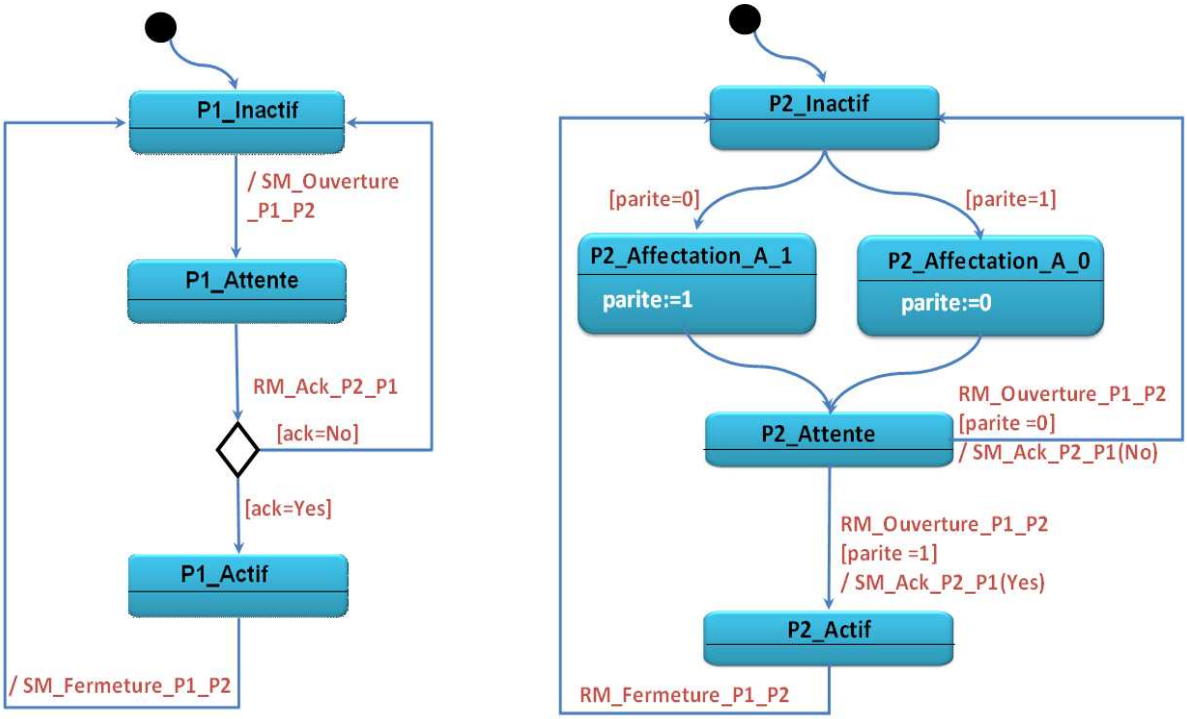


Figure 3.63: Diagrammes de machines d'états de P1 et P2

Chaque diagramme UML est ensuite transformé en un diagramme RdP dans la section suivante.

3.4.2 Transformation des diagrammes d'états UML en Réseaux de Petri

L'application des règles de transformation développées dans la section 3.3.3 fabrique les diagrammes RdP présentés dans les figures 3.64 et 3.65 ci-dessous.

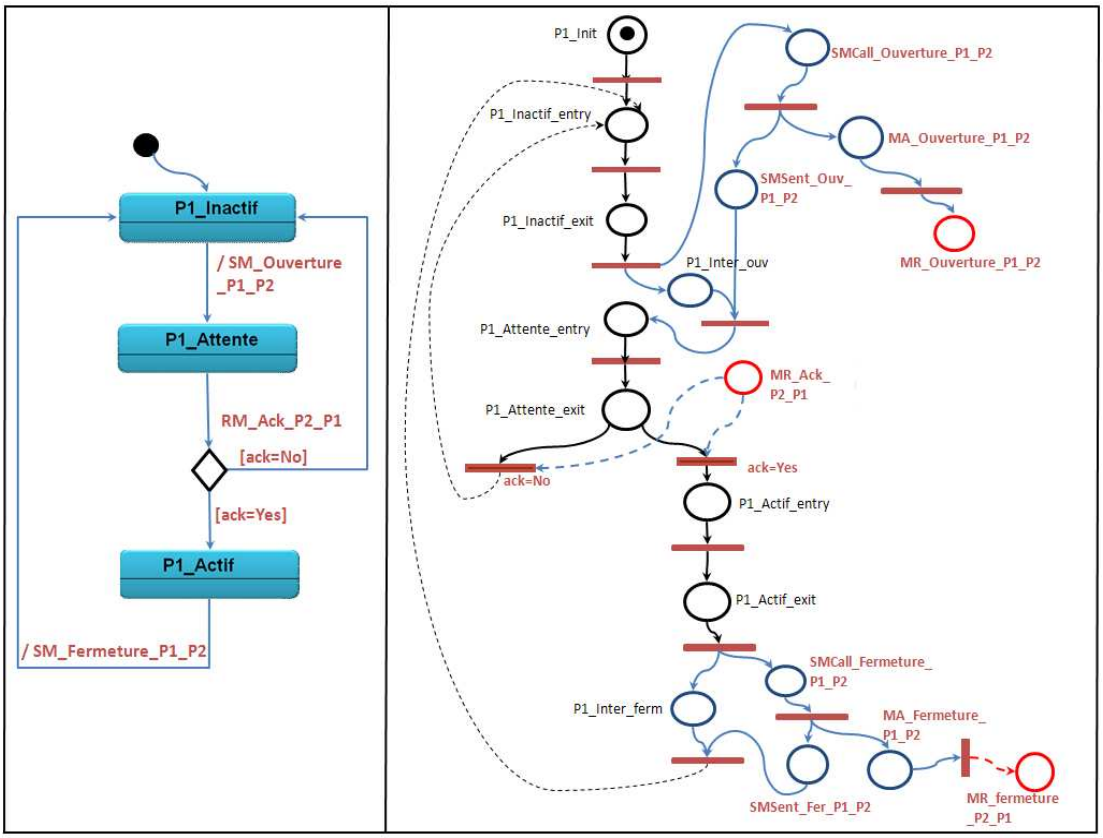


Figure 3.64: Transformation du diagramme de P1

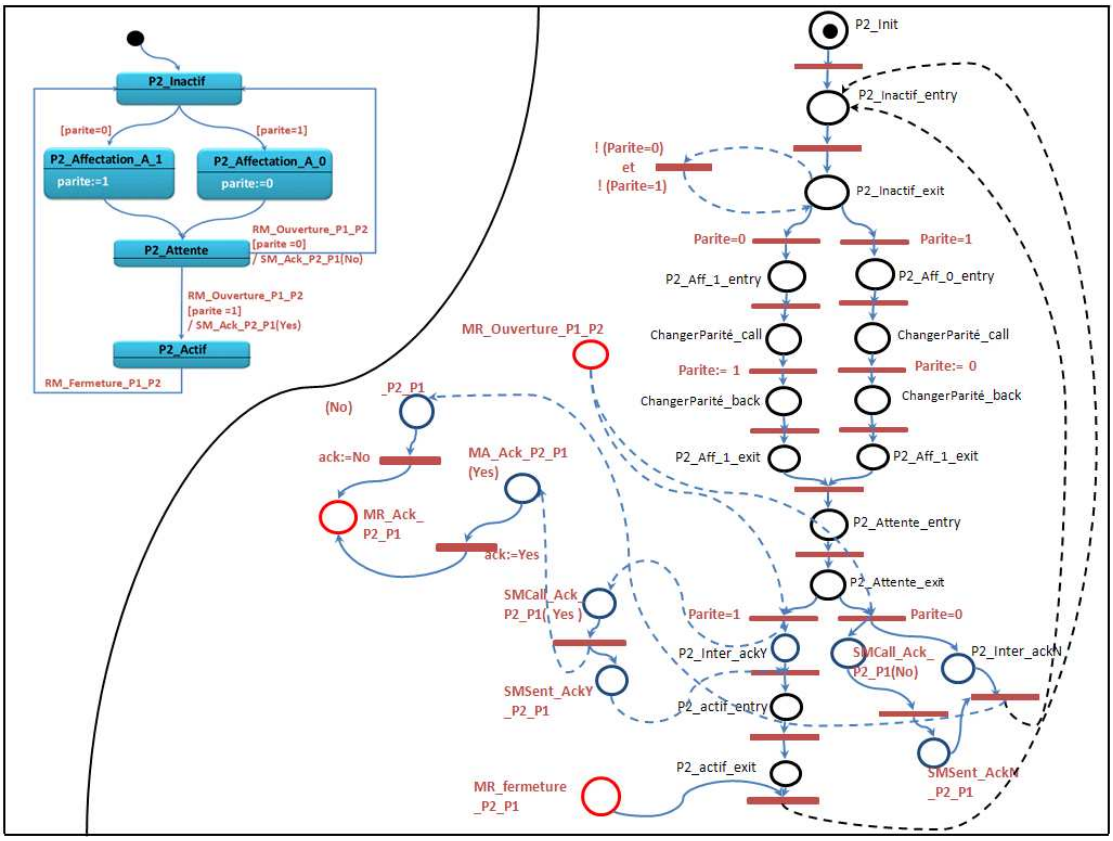


Figure 3.65: Transformation du diagramme de P2

Dans les figures précédentes, nous avons montré la transformation de chaque diagramme d'états UML en un diagramme RdP. Les places de couleur bleu sont les places correspondant aux déclencheurs et actions associés aux transitions. Quant aux places de couleur rouge, elles représentent les places communes entre les deux diagrammes et serviront au rassemblement des deux graphes RdP afin de former le modèle RdP global.

3.4.3 Regroupement des graphes RdP

Le regroupement des deux graphes RdP selon les places communes permet de construire un modèle complet présenté dans la figure 3.66 ci-dessous.

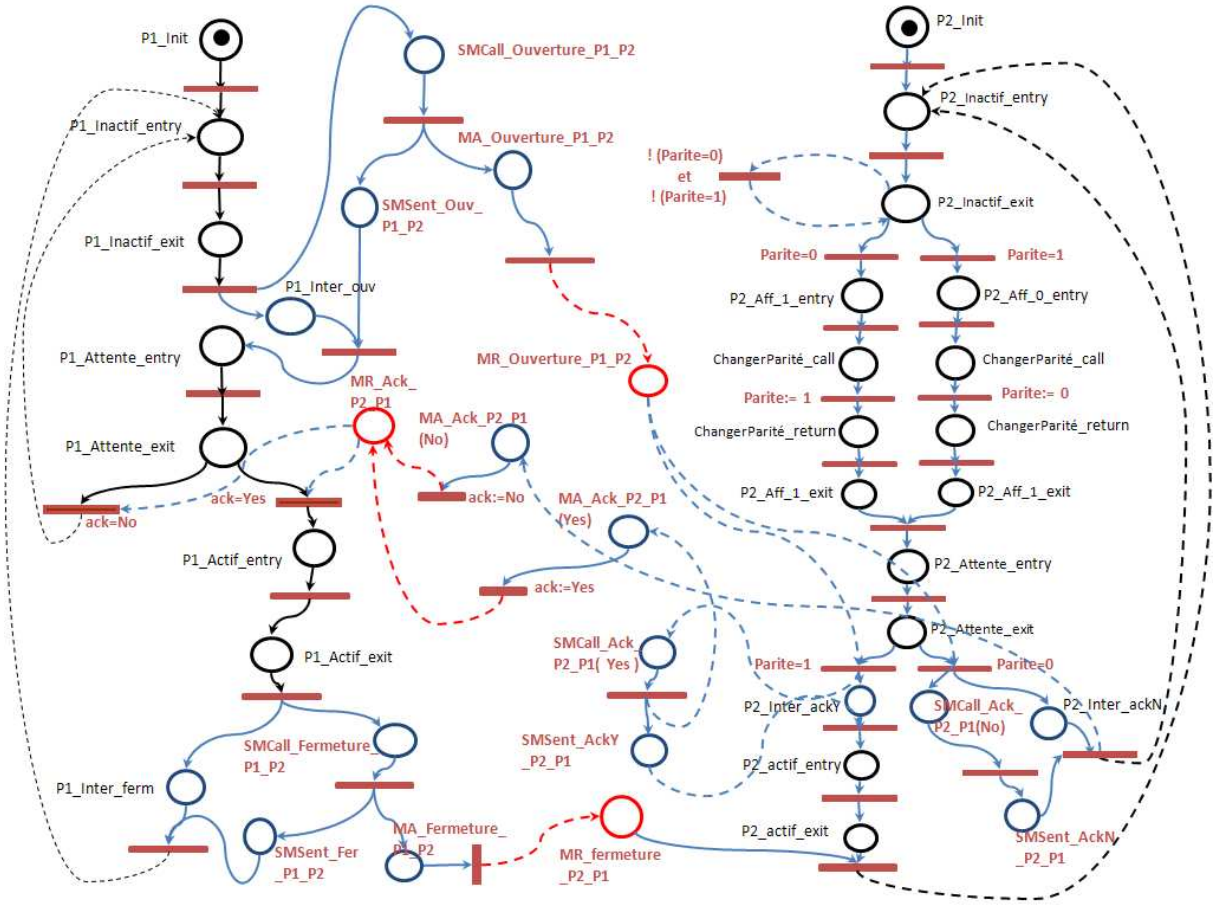


Figure 3.66: Rassemblement des graphes RdP

La transformation automatique du modèle UML en ce modèle RdP est réalisée suite à l'exécution de nos règles de transformation développées avec le langage Prolog. Tout le code de notre transformateur Prolog est fourni en Annexe G. Le RdP de la figure 3.66 est généré sous la forme d'un ensemble de transitions reliant des places en amant et en aval. Par exemple dans le cas où la parité est égale à 1 et que P2 est dans l'état « Inactif », alors comme nous l'avons déjà mentionné dans la description de cet exemple, P2 passe dans l'état « Affectation_A_0 » afin d'attribuer la valeur 0 à la variable « parité ». Nous retrouvons ce scénario dans le fichier des transitions du réseau généré suite à l'exécution de notre

transformateur (cf. Fig.3.67). Dans cette figure, nous donnons juste l'exemple de trois transitions correspondant au changement de parité par P2 dans le cas où la parité vaut 1.

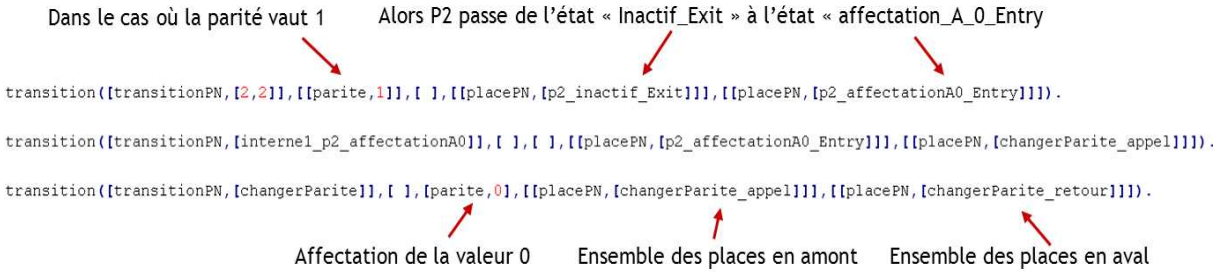


Figure 3.67: Transitions générées suite à la transformation automatique du modèle UML

3.5 Conclusion

Dans ce chapitre, nous avons décrit notre démarche de formalisation des spécifications. Ces dernières représentent le comportement des composants et le fonctionnement attendu. Vérifier ces composants par rapport à leur spécification nécessite une formalisation du cahier des charges. Nous avons opté pour une approche de modélisation de la spécification couplant un formalisme semi-formel objet (UML) et un formalisme formel (RdP). Ce couplage entre le monde objet et le monde formel est dû aux différents apports et avantages de ces derniers dans le contexte de modélisation et de vérification des systèmes complexes réactifs. UML permet de modéliser une vue statique du système à travers un diagramme de classes (représentant les classes et leurs propriétés) et une vue dynamique à travers des diagrammes de séquences (représentant les messages envoyés entre les objets au cours du temps) et des diagrammes de machines d'états (représentant les états d'un objet lors du déroulement d'un scénario donné). Les diagrammes d'états UML (modèle cible) sont ensuite transformés en un modèle RdP possédant des caractéristiques mathématiques et permettant ainsi la génération de scénarios de tests. La technique de transformation utilisée est basée sur l'approche MDA (Model Driver Architecture) et permet de transformer un modèle source (conforme à son méta-modèle) en un modèle cible (conforme à son méta-modèle) suivant des règles de transformation. Nous avons développé des règles transformant les machines d'états UML en une sous-classe des RdP interprétés.

Dans le chapitre suivant, nous allons décrire comment utiliser un modèle RdP afin de produire des scénarios de test permettant de tester un composant par rapport à sa spécification. Nous définissons une méthode de génération de scénarios de test à partir d'un graphe d'accessibilité se basant sur les techniques de programmation par contraintes.

4 GENERATION DE SCENARIOS DE TEST

Résumé:

Dans ce dernier chapitre, nous détaillons une méthode de génération automatique de scénarios de test à partir de modèles en réseaux de Petri interprété. Nous considérons le scénario de test comme une **séquence de franchissement filtrée puis réduite** du réseau de Petri interprété représentant la spécification. Nous développons notre méthode sur trois étapes. Dans la première étape, nous considérons le modèle réseau de Petri interprété de la spécification comme un modèle réseau de Petri simple et nous utilisons les techniques d'abstraction logique et la programmation par contraintes afin de générer l'ensemble des séquences de franchissement. Dans la deuxième étape, nous procédons à une phase de **filtrage** de cet ensemble de séquences en tenant compte des caractéristiques du réseau de Petri interprété (prédicats et opérations associées aux transitions). Enfin, dans la troisième étape de notre méthode, nous procédons à une phase de **réduction** de l'ensemble des séquences générées. Il s'agit en effet de tester des composants réactifs où seuls les messages vers ou depuis ces composants sont visibles. Les séquences de franchissement ne devraient contenir que les **transitions aux interfaces** afin de pouvoir générer l'ensemble des scénarios de test. Ces scénarios de test, exécutés sur une plateforme de simulation, permettent la validation des composants à tester.

Sommaire du chapitre 4

4.1 Introduction

L'objectif de ce chapitre consiste à générer des scénarios de test pour la validation de composants ERTMS par rapport à leur spécification. Dans le chapitre précédent, nous avons détaillé la méthode développée afin de modéliser formellement cette spécification. Le formalisme qui a été choisi est une celui des réseaux de Petri interprétés vu sa capacité à modéliser les systèmes réactifs de manière formelle. Nous avons défini une sous-classe des réseaux de Petri interprétés afin de répondre à notre contexte de travail (cf. section 3.3.2.3). Dans un réseau de Petri interprété, les transitions sont étiquetées par des conditions et des opérations. Des variables sont utilisées afin de représenter les états des composants. Une transition est valide si la condition correspondante est valide (la valeur de la variable en question est égale à la valeur indiquée). Le franchissement de cette transition exécute l'opération correspondante (affectation d'une nouvelle valeur à la variable en question) et permet de changer l'état courant du système. A partir des modèles de la spécification exprimée sous forme de réseaux de Petri interprétés, nous voulons générer des scénarios de test permettant, en les exécutant sur le simulateur ERTMS de vérifier si les réponses de ce dernier sont cohérentes avec la spécification [Jabri et *al*, 2010].

Le modèle RdP définit le comportement global du système distribué. Il s'agit d'interactions entre les divers composants réactifs. D'un point de vue RdP, le scénario de test est considéré comme une **séquence de franchissement filtrée puis réduite** du réseau de Petri interprété représentant la spécification. Cette **séquence de transitions aux interfaces** mène à un marquage final dans le graphe d'accessibilité correspondant au RdP. Le scénario de test est un ensemble d'entrées correspondant à la spécification dont nous voulons récupérer les sorties à comparer avec les sorties du simulateur. L'idée de générer des scénarios de test de type boîte noire pour vérifier des composants ERTMS consiste à déterminer toutes les séquences de franchissement du modèle réseau de Petri modélisant la spécification ERTMS.

Déterminer toutes ces séquences revient à parcourir le graphe d'accessibilité du réseau de Petri de manière exhaustive. C'est précisément dans ce contexte qu'intervient le phénomène de l'explosion combinatoire, puisque la taille de ce graphe peut croître exponentiellement avec la dimension du réseau étudié. Il est ainsi difficile de construire tout le graphe d'accessibilité. De nombreuses méthodes permettant de remédier à ce phénomène sont présentées dans la section suivante de ce chapitre. Nous nous inspirons de la méthode développée par Bennaser [Bennaser, 2000] pour résoudre le problème d'explosion combinatoire. Cette méthode, basée sur la technique d'abstraction logique et la programmation par contraintes, permet de générer à partir d'un réseau de Petri simple

(places–transitions) toutes les séquences de steps de longueur maximale donnée permettant d'atteindre un marquage donné depuis un marquage initial.

Dans la **première étape** de notre méthode de génération de scénarios de tests, nous considérons le modèle réseau de Petri interprété de la spécification comme un modèle réseau de Petri simple. Nous ignorons en effet les conditions (prédicats) et les opérations associées aux transitions du réseau de Petri interprété et nous procédons à la génération de toutes les séquences de transitions (transitions internes et transitions aux interfaces) depuis le marquage initial en se basant sur l'algorithme développé par Bennaser [Bennaser, 2000]. L'ensemble de toutes les séquences de transitions de longueur maximale donnée contient des séquences qui ne sont pas valides pour la spécification étant donné que nous n'avons pas pris en compte les conditions associées aux transitions. *A la fin de cette première phase, l'ensemble des séquences générées est inclus dans le graphe d'accessibilité du* ***réseau de Petri simple*** *comme le montre la figure 4.69.*

Dans la **deuxième étape**, nous procédons à une phase de filtrage de cet ensemble de séquences en tenant compte des prédicats et des opérations associées aux transitions du réseau de Petri interprété. En effet, certaines transitions ne sont pas franchissables si le prédicat correspondant n'est pas valide, et certaines séquences ne sont donc pas également valides. Un état du réseau de Petri est représenté par un couple (marquages, variables). Les valeurs de ces variables caractérisent l'état courant et décident des transitions franchissables. Lors du franchissement d'une transition valide, une opération concernant l'affectation d'une nouvelle valeur à la variable peut changer l'état du système vers un nouvel état. L'algorithme de Bennaser permet de générer des séquences de steps permettant de franchir plusieurs transitions simultanément. Or le franchissement simultané dans notre cas peut créer des conflits entre les opérations associées aux transitions du réseau de Petri. Par exemple, le franchissement simultané de deux transitions auxquelles sont associées deux opérations d'affectation de la même variable produit deux état différents du système d'où la situation de conflit. Nous supposons alors qu'une seule transition peut être franchie à la fois. A la fin de cette phase, l'ensemble des séquences obtenu est un ensemble filtré sur la base de l'ensemble des séquences de la phase précédente. *Cet ensemble filtré est inclus dans le graphe d'accessibilité du* ***réseau de Petri interprété*** *comme le montre la figure 4.70.*

Enfin, dans la **troisième étape** de notre méthode, nous procédons à une phase de réduction de l'ensemble des séquences générées (cf. Fig.4.71). Nous voulons tester en effet des composants réactifs dont nous ignorons le comportement interne (cf. sections 2.3 & 3.3.3.1). Dans les scénarios de test, seuls les messages vers ou depuis ces composants sont visibles. Dans le modèle réseau de Petri de la spécification, nous modélisons les composants ainsi que les communications correspondant à l'envoi et à la réception de messages. Lors du développement de notre méthode de transformation de modèles (cf. section 3.3.3.1), nous

avons représenté les communications entre les composants par des transitions que nous avons appelées des ***transitions aux interfaces. Ce sont alors les séquences de franchissement des transitions aux interfaces qui nous intéressent dans notre démarche de test. L'ensemble des séquences filtré obtenu lors de la deuxième phase est alors réduit*** en un ensemble de séquences ne contenant que les transitions aux interfaces.

Méthodologie

Nous reprenons l'exemple développé à la fin du chapitre 3 et nous présentons ici un scénario nominal simple. Soit deux processus P_1 et P_2 qui communiquent ensemble en envoyant des messages. Le scénario nominal est décrit ainsi :

- le processus P_1 demande à ouvrir une session de communication avec P_2 en envoyant un message d'ouverture à P_2. Il passe ensuite d'un état inactif à un état d'attente.
- A la réception de ce message, P_2 passe dans un état actif et accuse réception. Dans le cas d'un scénario nominal, l'accusé de réception confirme l'ouverture de session de communication.
- A la réception de l'accusé de réception de P_2 (accusé = oui dans le scénario nominal), P_1 passe dans un état actif. Les deux processus sont alors dans un état actif.
- Dans le cas où l'accusé de réception ne confirme pas l'ouverture de session de communication (accusé = non), P_1 revient dans l'état inactif.

Dans la figure 4.68, nous modélisons cet exemple par un réseau de Petri interprété afin de détailler les différentes étapes de notre démarche. Dans la suite, nous considérons uniquement le scénario nominal (accusé = Oui).

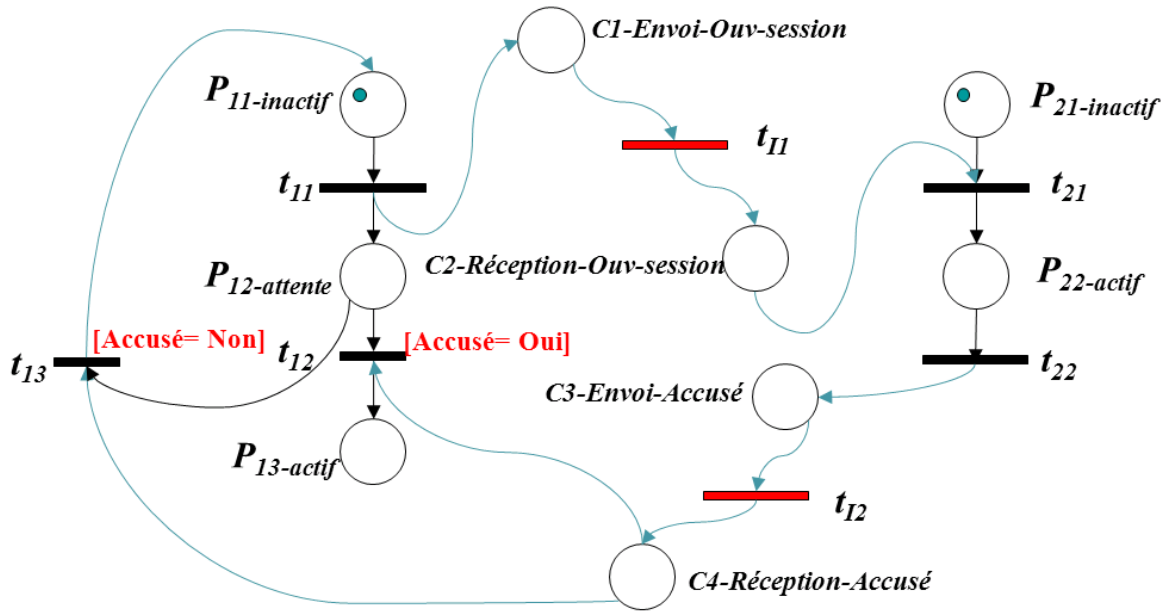


Figure 4.68: Réseau de Petri interprété de l'exemple

Objectif. L'objectif consiste à déterminer les scénarios de test permettant de tester ces deux composants. Nous supposons qu'il s'agit de test de type boîte noire. Nous testons donc uniquement les communications entre ces deux composants. Les communications sont représentées dans le réseau de Petri interprété par des transitions aux interfaces comme le montre la figure 4.68 (t_{I1} et t_{I2} sont deux transitions aux interfaces). Le scénario de test est représenté par une séquence de franchissement (suite de transitions franchissables les unes après les autres). Générer ces séquences consiste alors à parcourir le graphe d'accessibilité du réseau de Petri interprété. Afin d'atteindre cet objectif et d'extraire des séquences de franchissement des transitions aux interfaces dans le graphe d'accessibilité du réseau de Petri interprété, nous développons notre méthode en trois étapes.

A- **Première étape** : Génération des scénarios de test à partir d'un réseau de Petri places-transitions. Dans cette phase, nous ignorons les conditions et les opérations associées aux transitions du réseau de Petri interprété et nous générons l'ensemble des séquences de franchissement à partir d'un réseau de Petri simple (cf. section 4.3.1). Par exemple, dans la figure 4.70 nous pouvons franchir simultanément les transitions t_{12} et t_{13} étant donné que nous ne prenons pas en compte les conditions associées.

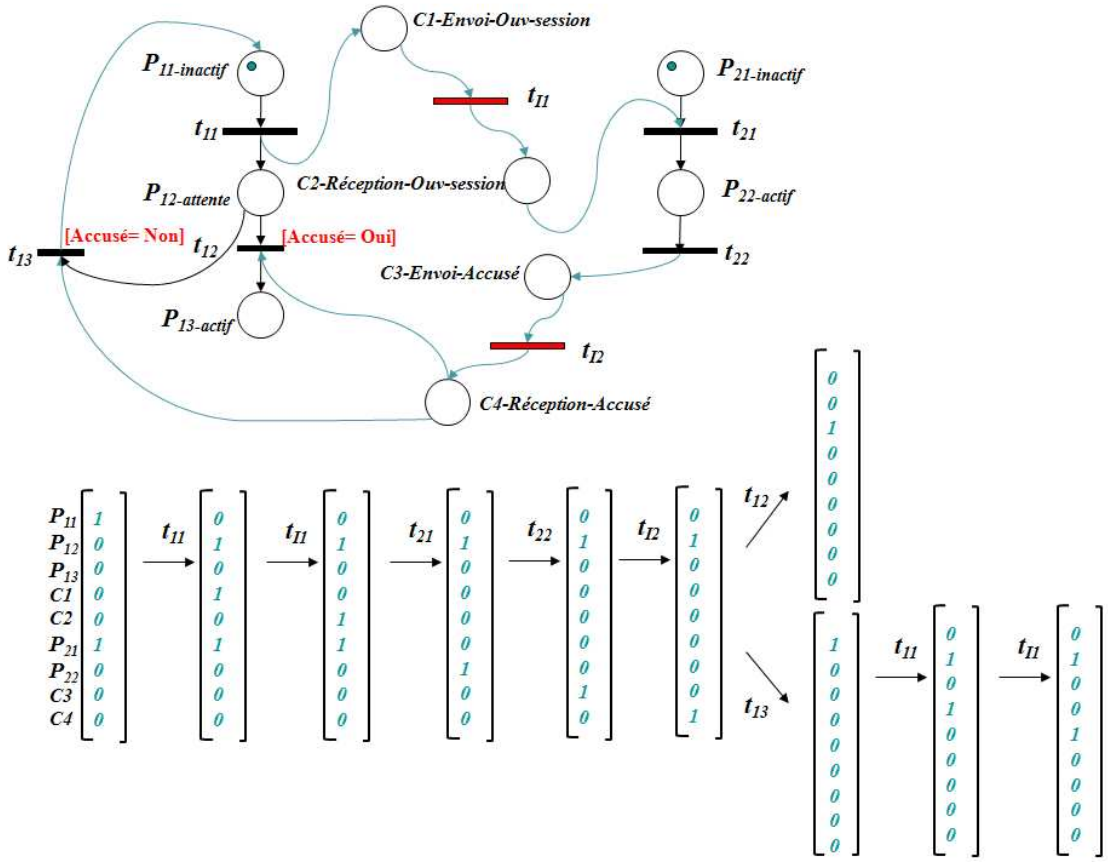


Figure 4.69: Etape1-Graphe d'accessibilité du réseau de Petri simple

B– **Deuxième étape** : Filtrage de l'ensemble des séquences générées (cf. section 4.3.2). Les séquences incluses dans le graphe d'accessibilité du réseau de Petri simple sont filtrées en prenant compte les caractéristiques (prédicats et opérations) du réseau de Petri interprété (cf. Fig.4.70). Dans le cas du scénario nominal de l'exemple énoncé précédemment, l'accusé de réception confirme l'ouverture de la session. Le prédicat [Accusé =Oui] associé à la transition $\mathbf{t_{12}}$ est vrai et la transition $\mathbf{t_{12}}$ est franchissable. Cependant la transition $\mathbf{t_{13}}$ n'est pas franchissable car le prédicat associé n'est pas vrai. Les transitions qui ne sont pas valides éliminent des chemins dans le graphe d'accessibilité. Dans la figure 4.70, le chemin contenant la transition $\mathbf{t_{13}}$ a été supprimé.

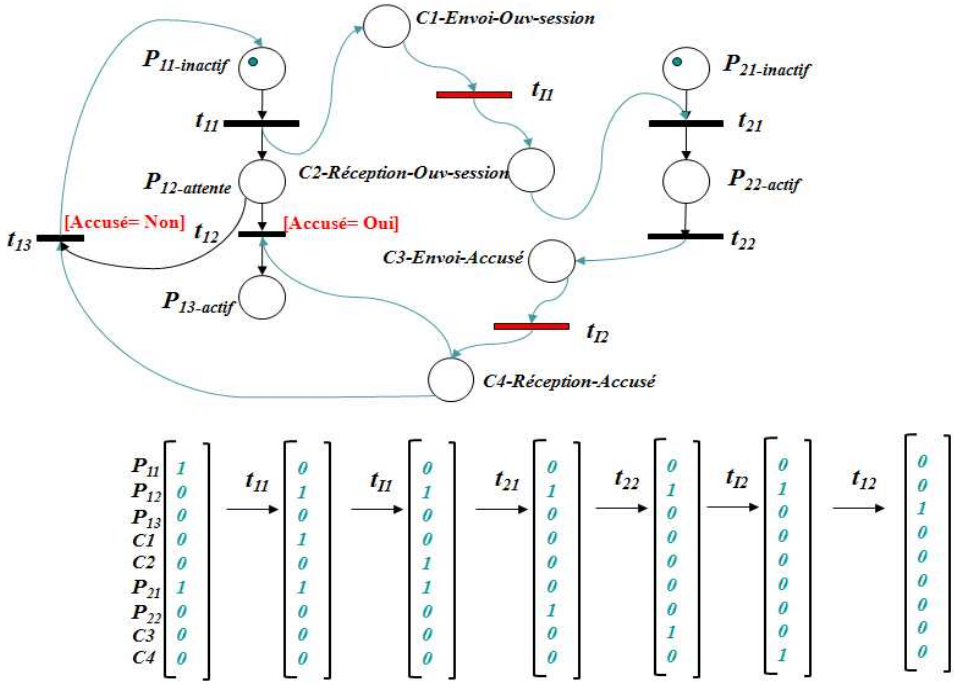


Figure 4.70: Etape 2- Filtrage des séquences de franchissement

C– **Troisième étape** : Réduction de l'ensemble des séquences filtrées (cf. section 4.3.3) en ne considérant que les transitions aux interfaces (cf. Fig.4.71).

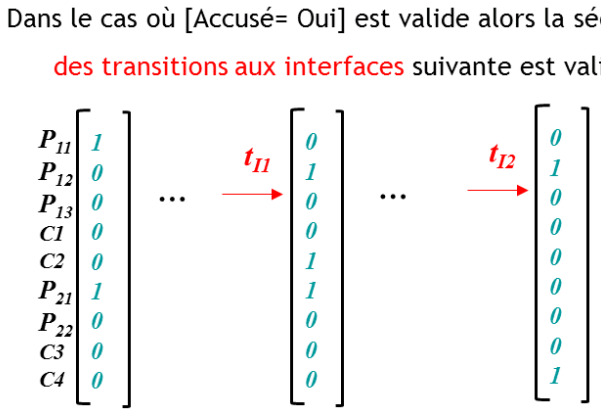


Figure 4.71: Etape3- Réduction de l'ensemble des séquences générées

Pour résumer la démarche, nous présentons les trois étapes dans la figure suivante :

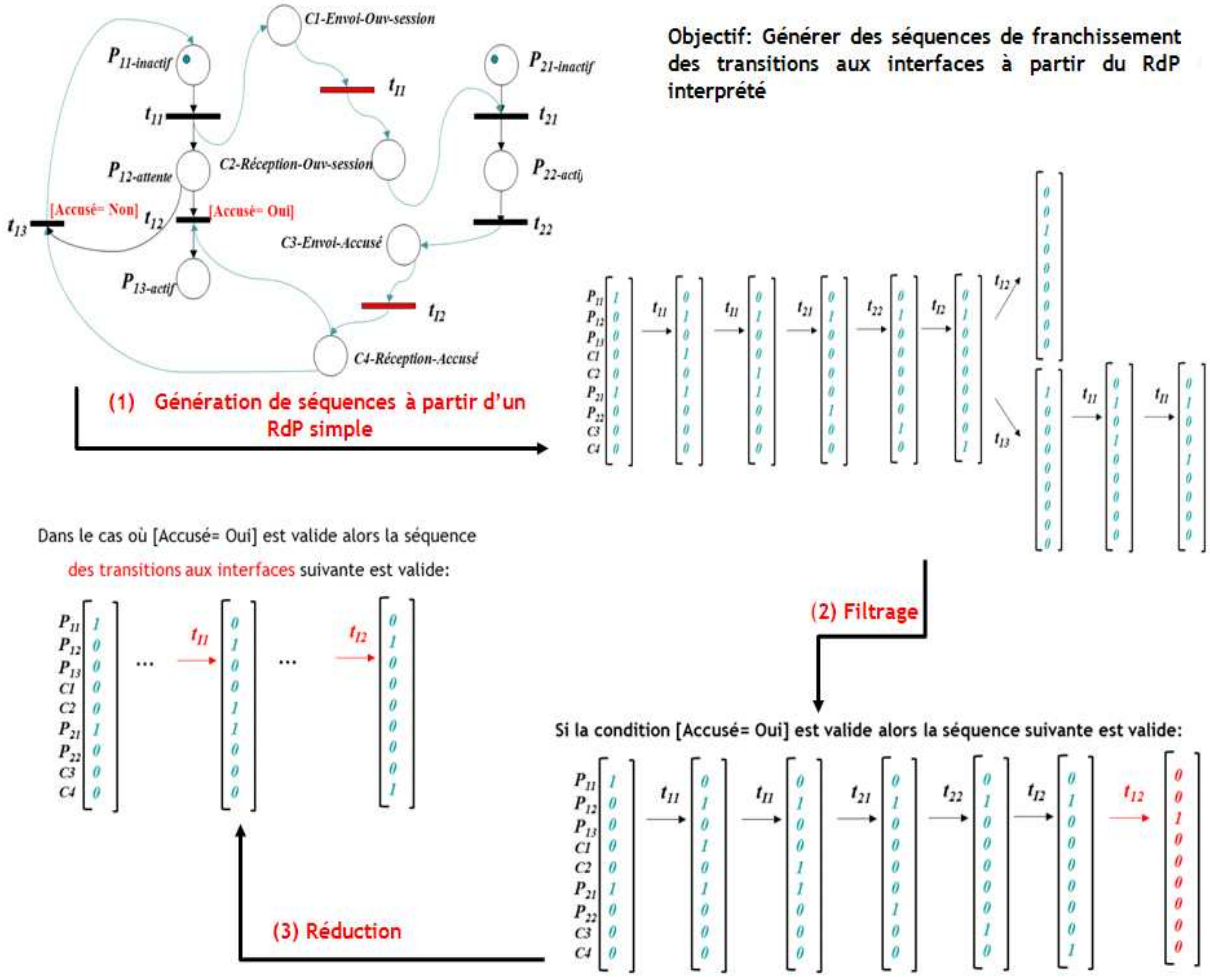


Figure 4.72: Démarche de génération de scénarios de tests

Suite à cette partie introductive de notre démarche de génération de scénarios de test, nous définissons les scénarios de test et le comportement des réseaux de Petri. Ensuite, nous détaillons notre démarche de génération de séquences de franchissement dans le but de produire des scénarios de test. Enfin, nous fournissons un exemple illustrant cette démarche.

4.2 Scénarios de test

Le système à tester constitue une portion de la réalité. La frontière qui définit cette portion est organisée en fonction d'un objectif donné. Dans le cas de système réparti comme le nôtre, les différents éléments le constituant, sont en interaction dynamique [Jabri et *al*, 2010]. Le modèle de ce système consiste à représenter les communications entre les différents constituants. A partir de ce modèle d'interactions, nous voulons générer des scénarios de test pour la vérification de constituants.

4.2.1 Les scénarios de test

Définition 4.2.1 / **Scénario** [Scorletti & Binet, 2006]. Un scénario est défini par un instant initial t_0, un instant final $t_1 > t_0$, la valeur des variables d'état à l'instant initial t_0 et par la valeur des variables d'entrées pour chaque instant appartenant à l'intervalle $[t_0,t_1]$.

4.2.1.1 Approche de test basée sur les scénarios

Au cours des dernières années, nous avons remarqué un certain engouement pour l'utilisation de scénarios dans les méthodes de conception de systèmes [Amyot et *al*, 1997]. L'émergence des cas d'utilisation (use cases) dans le monde orienté objet confirme cette tendance. En effet, de nombreuses méthodologies basées sur les scénarios ont été proposées dans la littérature. Cependant, selon les approches, le terme « scénario » possède des sémantiques bien différentes les unes des autres. On associe parfois les scénarios à des traces (d'événements internes et/ou externes), à des échanges de messages entre composantes, à des séquences d'interactions entre un système et l'utilisateur… Nombreuses sont aussi les notations utilisées, qu'elles soient basées sur une grammaire textuelle plus ou moins formelle, sur des automates, ou sur des diagrammes d'échanges de messages [Amyot et *al*, 1997].

Produire des ***scénarios de test*** consiste à imaginer et créer des cas concrets d'utilisation de l'application à partir d'une description formalisée des spécifications. Une fois que cette dernière est validée, les jeux de tests peuvent être définis afin de prévoir et de formaliser la manière et les critères avec lesquels l'application va être testée. Les scénarios de test, définis dans le cadre des jeux de tests, se basent sur une spécification valide. Un cas d'utilisation de la spécification représente un déroulement normal prévu [Jabri, 2009b]. Il s'agit d'un scénario nominal. Par contre, afin d'assurer un jeu de tests plus complet, il est essentiel de prévoir les cas exceptionnels d'utilisation : ceux qui génèrent des erreurs, les cas de problèmes, les utilisations inhabituelles. Il s'agit d'un scénario non-nominal. Les scénarios de tests permettent de contrôler l'enchaînement des étapes prévues de manière formelle ou implicite dans les spécifications.

Dans le cadre de nos travaux de thèse, nous avons jugé qu'une approche orientée scénarios représente un choix judicieux pour décrire nos composants réactifs et communicants. L'objectif de ce chapitre consiste à définir des scénarios de tests de conformité à partir de la modélisation formelle de la spécification.

4.2.1.2 Scénario de test de conformité

Le scénario de test constitue un cas d'utilisation de la spécification. Or, nous étudions une approche de type « boîte noire » où nous n'avons accès qu'à l'interface offerte par le système. Les scénarios de test de conformité décrivent alors des échanges de

communications entre les composants. Ils sont **générés à partir de la spécification formelle** ERTMS définissant le comportement global des différents composants, puis **exécutés sur le simulateur ERTMS nous permettant ainsi de valider ces composants**.

Le formalisme RdP est utilisé comme un outil de modélisation formelle et un outil d'analyse. Comme nous l'avons mentionné dans l'introduction, le scénario de test généré, exécuté sur le simulateur, nous permet de juger si le composant respecte bien la spécification et produit les sorties attendues. Au niveau des RdP, les jetons qui circulent entre les places lors du franchissement des transitions permettent de modéliser le comportement dynamique d'un système distribué. Ainsi pour déterminer un scénario dans le RdP, il suffit d'établir une séquence de franchissement d'un ensemble de transitions permettant d'atteindre un marquage accessible depuis un marquage initial [Jabri et *al*, 2010]. Ces deux marquages constituent deux états différents du système modélisé. Les marquages accessibles d'un RdP forment le graphe d'accessibilité. Nous nous intéressons ainsi aux séquences de franchissement contenues dans ce graphe. Une séquence de franchissement est définie comme une suite de transitions $t_i t_j \dots t_k$ qui peuvent être franchies successivement depuis un marquage donné. Une seule transition peut être franchie à la fois.

4.2.2 Comportement du réseau de Petri

Le graphe d'accessibilité d'un réseau de Petri est un graphe dont les nœuds et les arcs représentent respectivement les états du système et les actions ou opérations permettant d'évoluer d'un état à un autre. La génération du graphe d'accessibilité consiste à construire tous les marquages accessibles et permet ainsi de résoudre les problèmes d'analyse des systèmes (vivacité, atteignabilité d'un état,…)

4.2.2.1 Définitions

Avant de définir formellement un graphe d'accessibilité, nous décrivons le processus de franchissement d'une séquence de transitions. Nous considérons un RdP $(\Re = (\mathbb{P}, \mathbb{T}, \mathcal{W}), m_0)$ où:

- $\mathbb{P} = \{p_1, p_2 \dots p_M\}$ un ensemble fini de places avec M= $\|\mathbb{P}\|$;
- $\mathbb{T} = \{t_1, t_2 \dots t_N\}$ un ensemble fini de transitions avec N= $\|\mathbb{T}\|$;
- $\mathcal{W}: (\mathbb{P} \times \mathbb{T}) \cup (\mathbb{T} \times \mathbb{P}) \to \mathbb{N}$ associe à chaque arc un poids $\boldsymbol{W}(p, t)$ ou $\boldsymbol{W}(t, p)$;
- $m_0: \mathbb{P} \to \mathbb{N}$ associe à chaque place $p \in \mathbb{P}$ un entier $\boldsymbol{m_0}(p)$ appelé marquage.

Vecteur caractéristique (ou canonique) & vecteur de marquage

(**Vecteur caractéristique**). Le vecteur caractéristique $\overrightarrow{e_{t_j}}$ associé à la transition t_j (resp. $\overrightarrow{e_{p_i}}$ associé à la place p_i) est le vecteur de $\mathbb{N}^N$ (resp. de $\mathbb{N}^M$) dont la j^{ème} composante vaut 1 et les autres sont nulles : $\overrightarrow{e_{t_j}} = (\delta_1^j, \delta_2^j, \dots, \delta_N^j,)^T \in \mathbb{N}^N$, $\overrightarrow{e_{p_i}} = (\delta_1^i, \delta_2^i, \dots, \delta_M^i,)^T \in \mathbb{N}^M$.

(**Vecteur de marquage**). On associe à m son vecteur de marquage $\vec{\mathrm{M}}$ défini par :

$$\vec{\mathrm{M}} = (m(p_1), m(p_2), \dots, m(p_M))^T \in \mathbb{N}^M.$$

Franchissement des transitions

Dans un réseau de Petri, les marquages des places représentent l'état du système correspondant à un instant donné. Ce marquage peut être modifié en fonction du franchissement des transitions. Une transition t est franchissable pour un marquage m_0 (notée $m_0[t\rangle$), si $\forall p \in \mathbb{P}, m_0(p) \geq \mathcal{W}(p,t)$. Si cette condition est satisfaite, alors un nouveau marquage m_1 est créé à partir du marquage m_0 (noté $m_0\,[t\rangle m_1$):

$$\forall p \in \mathbb{P},\ m_1(p) = m_0(p) - \mathcal{W}(p,t)\ + \mathcal{W}(t,p).$$

Les équations précédentes peuvent être généralisées au tir de séquences de transitions franchies successivement. Soit $\sigma = t_{\sigma_1} t_{\sigma_2} \dots t_{\sigma_r}$ une séquence de transitions de (R, m_0), nous définissons le vecteur caractéristique $\vec{\sigma}$ de la séquence σ , où $\vec{\sigma}$ représente le nombre de tirs de la transition t dans la séquence $\sigma, \forall\ t\ \in \mathbb{T}$.
Formellement, $\vec{\sigma} = \sum_{j=1}^{r} \overrightarrow{e_{t_{\sigma_j}}}$ où $\overrightarrow{e_{t_j}}$ est le vecteur caractéristique (ou canonique) de la transition t_j. Nous avons ainsi :

σ est valide à partir d'un marquage $m_0 \overset{def}{\Longleftrightarrow} m_0[\sigma\rangle$ et $Pre.\ \vec{\sigma} \geq \overrightarrow{M_0}$ (1)

$$m_0[\sigma\rangle\, m_1 \ \Rightarrow \overrightarrow{M_1} = \overrightarrow{M_0} + C.\vec{\sigma} \qquad (2)$$

L'équation (2) est appelée équation fondamentale ou équation d'état. La matrice d'incidence $C(p,t) = Post(p,t) - Pre(p,t)$. Le vecteur $\overrightarrow{M_0}$ représente le marquage initial. Nous désignons par T^{∞} l'ensemble des séquences d'éléments de $\mathbb{T}$ et par $\mathcal{T}(R, m_0)$ toutes les séquences de transitions franchissables depuis le marquage m_0.

4.2.2.2 Graphe d'accessibilité

Les règles de franchissement des transitions présentées précédemment peuvent être utilisées pour définir un graphe d'accessibilité associé au réseau de Petri. Le graphe d'accessibilité correspond à la représentation formelle du comportement d'un réseau R. Il est noté par $\mathcal{G}(\mathfrak{R}, m_0)$ et défini par:

- ✓ Un ensemble de nœuds $\boldsymbol{\mathcal{A}(R, m_0)}$ qui représentent tous les marquages accessibles par toute séquence de transitions franchissables (cf. Fig.4.73). Formellement:
  - $\mathcal{A}(R, m_0) = \{\, m_f \,|\, \exists\ \sigma \in \mathcal{T}(R, m_0)$ et $m_0[\sigma\rangle m_f \,\}$;
- ✓ Un ensemble d'arcs où un arc $\boldsymbol{(m_i, m_j)}$ étiqueté t relie les noeuds correponsdants aux marquages $m_i\ et\ m_j$ si $m_i[t\rangle m_j$.

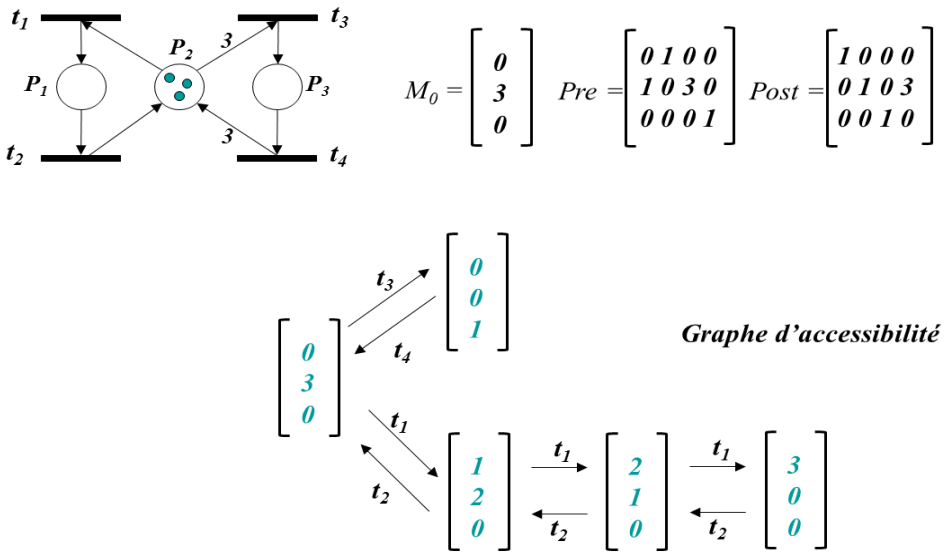


Figure 4.73: Graphe d'accessibilité fini

Pour un marquage initial donné, le graphe d'accessibilité $\mathcal{G}(R, m_0)$ et l'ensemble des nœuds accessibles correspondant $\mathcal{A}(R, m_0)$ ne sont pas nécessairement finis. Par exemple, l'ensemble des marquages accessibles depuis le marquage initial $(2,1,1,0)$ de la figure suivante est infini (cf. Fig.4.74). Afin de le prouver nous considérons n fois de répétitions de la séquence $t_1 t_2 t_3 t_4$ qui permet d'atteindre le marquage$(n+2, 1,1,0)$. Par exemple, si $n = 1$ alors le marquage est égal à $(3,1,1,0)$; par contre si $n = 2$ alors le marquage est égal à $(4,1,1,0)$. Ajouter 2 à n revient au poids de l'arc t_4 $vers$ p_1. En effet, en franchissant la transition t_4 nous rajoutons à la place p_1 3 jetons au lieu d'un seul jeton ce qui fait la différence de deux jetons dans chaque répétition de la séquence $t_1 t_2 t_3 t_4$.

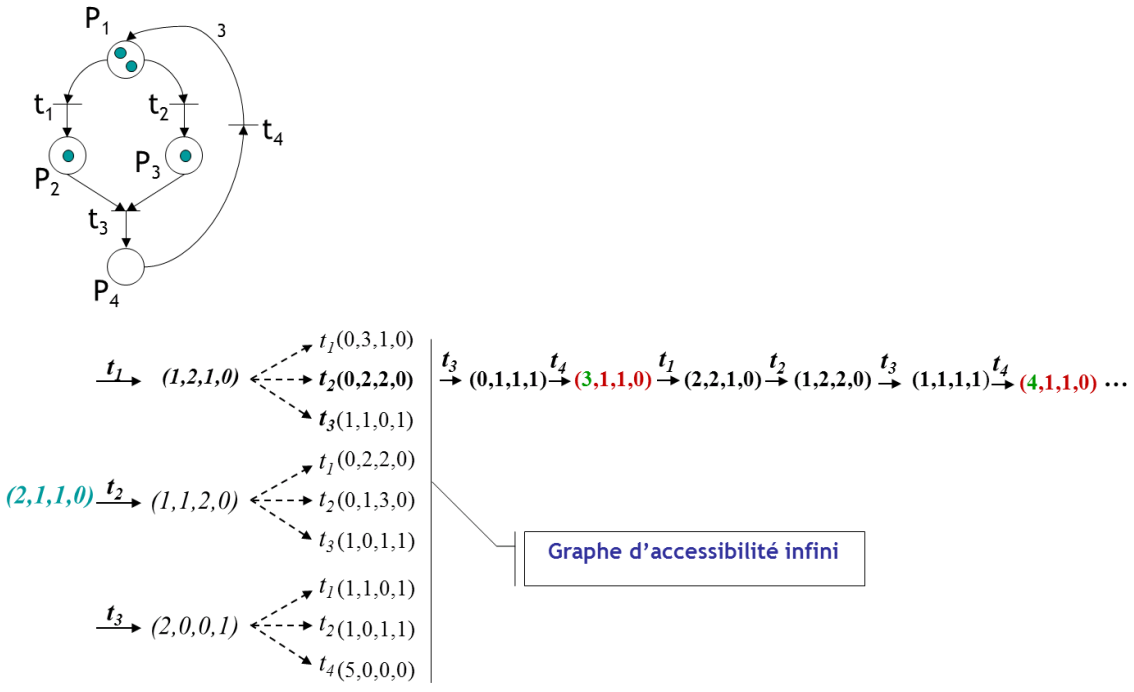


Figure 4.74: Graphe d'accessibilité infini

Dans le cas de réseaux bornés, l'ensemble des marquages accessibles $\mathcal{A}(R, m_0)$ est fini. Un réseau est considéré borné si et seulement si :

$$\exists\, k \in \mathbb{N}, \forall m \in \mathcal{A}(R, m_0), \forall p \in \mathbb{P}\ m(p) \leq k$$

Le cardinal de l'ensemble des marquages accessibles est limité par k^M.

Le problème d'accessibilité est défini ainsi : soit un réseau de Petri R, un marquage initial m_0, un marquage final m_f, comment décider si m_f est accessible depuis m_0 (par exemple si $m_f \in \mathcal{A}(R, m_0)$.) Résoudre ce problème consiste à trouver une séquence de transitions franchissable de l'ensemble $\mathcal{T}(R, m_0)$ tel que $m_0[\sigma\rangle m_f$.

Une des approches consiste à explorer le graphe d'accessibilité d'une manière exhaustive. Cependant en pratique, il est impossible d'explorer exhaustivement le graphe car il est caractérisé par le problème d'explosion combinatoire : la taille de l'espace peut évoluer exponentiellement avec la dimension du réseau étudié. Les méthodes fondées sur la construction et le parcours du graphe d'accessibilité sous-jacent deviennent vite impraticables avec l'augmentation de la taille du réseau. La plupart des problèmes d'analyse de réseaux de Petri (et notamment les problématiques concernant la vivacité ou les blocages, qui présentent un grand intérêt dans la vérification des systèmes) sont réductibles au problème de l'accessibilité, comme l'indiquent Araki et al. [Araki et al, 1977], et sont soumis aux mêmes limitations. De multiples travaux ont essayé de maîtriser cette explosion.

Vernadat propose dans son mémoire d'habilitation à diriger des recherches [Vernadat, 2001] de classer les techniques visant à maîtriser l'explosion combinatoire en deux catégories : **Gérer** (gestion de l'explosion) ou **Combattre** (réduction de l'explosion) :

« La gestion met en œuvre des techniques sophistiquées pour « contourner » la difficulté de traiter des systèmes complexes. Les techniques de réduction tirent parti des régularités, des redondances ou des similitudes existant dans la représentation exhaustive pour réduire la complexité du système. Il faut noter que ces approches sont complémentaires : on peut les appliquer conjointement et en cumuler les bénéfices. »

Nous suivrons cette dialectique dans notre exposé, en nous inspirant largement de la présentation faite par l'auteur dans [Vernadat, 2001].

4.2.3 Gestion de l'explosion combinatoire par vérification à la volée

De manière classique, la vérification de systèmes se déroule en deux temps. On procède d'abord à la construction du graphe du comportement puis à l'exécution d'algorithmes de décision sur ce dernier. Les techniques de vérification à la volée proposent au contraire

d'effectuer ces deux tâches en parallèle. Ainsi, la satisfaction des propriétés est évaluée au cours de l'énumération du modèle de comportement.

Toutes les techniques reposent sur des algorithmes d'exploration du graphe de comportement en profondeur d'abord [Fernandez et *al*, 1996]. Comme seuls les états sur un chemin ont besoin d'être mémorisés (des techniques permettent de détecter les cycles et de ne pas les parcourir), la capacité mémoire nécessaire correspond à la taille du plus long chemin sans cycle. Or, son ordre de grandeur est souvent bien inférieur à celui de la taille totale du graphe. Ainsi, ces algorithmes permettent une analyse moins coûteuse en mémoire que l'analyse en largeur d'abord.

Des techniques de cache d'états élaborées permettent d'éviter de parcourir les mêmes états plusieurs fois. De plus, ces types de parcours sont plus adaptés à la vérification de propriétés plus complexes que l'analyse en largeur d'abord, puisqu'ils permettent de suivre les comportements effectifs du système.

4.2.4 Réduction de l'explosion combinatoire

Les approches présentées dans ce paragraphe opèrent sur des restrictions du modèle de départ, en définissant un modèle de comportement incomplet, plus petit ou moins précis que l'original. Ainsi, les vérifications à mener sur ce modèle peuvent être faisables.

Les mécanismes proposés sont spécifiques à la propriété recherchée, et bien souvent la réduction opérée ne permet de traiter qu'une sous-classe de propriétés dont on sait qu'elles ont été conservées par le mécanisme de restriction utilisé. Ainsi, il pourra être nécessaire de réitérer le processus de réduction, qui est lui-aussi coûteux, pour chacune des classes de propriétés à vérifier.

Les restrictions opérées peuvent être de toutes natures, selon les propriétés à vérifier et les spécificités du système à traiter. Nous distinguons les techniques qui opèrent par des réductions du graphe du comportement, par la transformation du réseau de Petri lui-même ou de manière plus directe encore par l'étude des équations associées au réseau de départ sans modification de sa structure (méthodes algébriques [Lautenbach et *al*, 1987]).

4.2.4.1 Transformation et décompositions de réseau de Petri

Afin de réduire la taille de l'ensemble des marquages accessibles, Berthelot [Berthelot, 1986] a développé des techniques qui permettent de réduire la taille du réseau de Petri tout en préservant, selon les cas, les propriétés concernant le langage, la vivacité (Une condition nécessaire pour qu'un réseau de Petri soit vivant est que chaque transition soit tirée au moins une fois), les blocages. . .

Ces techniques consistent à appliquer des règles de *transformation* ou de *décomposition*. Une *transformation* est une modification d'un réseau de Petri. Cette modification s'applique sur les places ou se fait par fusion de transitions. Une *décomposition* consiste à étudier des sous-réseaux séparément puis d'en déduire le comportement du réseau complet.

La transformation la plus simple consiste à supprimer les places redondantes et les arcs qui lui sont connectés. Il s'agit de places dont la présence ou l'absence n'influe pas sur les comportements possibles du réseau. Il est également possible de procéder à la fusion de places sous certaines conditions. Selon le type de fusion utilisée, il y a conservation du langage du réseau ou des marquages accessibles (la place qui résulte de la fusion contient autant de marques que les deux places fusionnées réunies). Enfin, la fusion de transitions s'applique lorsque toutes les séquences de tirs peuvent être réordonnées de telle sorte que certaines transitions soient toujours appariées. Les paires de transitions ainsi formées sont alors fusionnées (cf. Fig.4.75).

Dans l'exemple de la figure 4.75, la transition t peut toujours être appariée avec soit la transition t_1 soit la transition t_2. Cette transition peut donc être fusionnée avec les transitionst_1 et t_2 pour former les transitions f_1 et f_2. Le tir de f_1 représente la séquence de tir tt_1 ou t_1t.

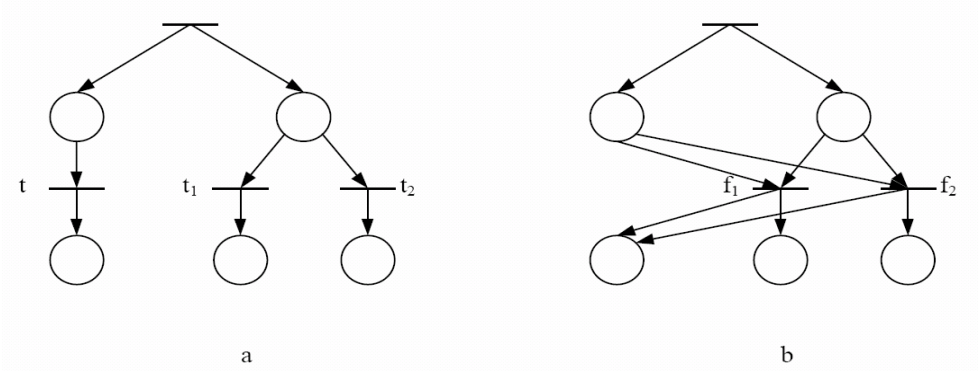


Figure 4.75: Fusion de transitions

Le principe des décompositions consiste à analyser un ensemble de petits réseaux afin d'en extraire des propriétés. L'objectif est d'en déduire les propriétés du réseau global obtenu par composition. La composition se fait en identifiant les places ou les transitions des sous réseaux. Le problème qui se pose est que la propriété n'est pas forcément conservée dans le réseau composé. Selon les propriétés que l'on cherche à vérifier et sous certaines conditions, la composition est possible.

4.2.4.2 Réduction du graphe de comportement

A- Graphe de couverture. Dans le cas des réseaux de Petri non bornés, le graphe de comportement sous-jacent comporte une infinité de marquages. Une première technique de

réduction consiste alors à associer au graphe infini son graphe de couverture, défini par Karp et Miller [Karp et *al*, 1969] et formalisé plus précisément par Finkel [Finkel, 1993]. Ce graphe permet d'avoir une représentation finie partielle de l'ensemble des états accessibles.

Lorsqu'un système possède un ensemble infini d'états, certaines composantes des marquages ne sont pas bornées. On introduit donc la notion de pseudo-marquage qui présente une généralisation du marquage dans le sens où c'est un marquage pouvant contenir un nombre infini de jetons. En fait, un pseudo-marquage représente une classe de marquages dont les places sont susceptibles de contenir un nombre arbitrairement grand de jetons.

Le principe du graphe de couverture est de représenter l'ensemble des marquages accessibles par un ensemble de pseudo-marquages. Cet ensemble couvre tous les marquages accessibles, d'où le terme de graphe de couverture. De plus, tout pseudo-marquage du graphe est la limite d'une suite de marquages accessibles : *le graphe de couverture* ne contient pas de nœuds superflus.

Les principaux résultats quant au graphe de couverture sont les suivants :

- ✓ Le graphe de couverture est fini ;
- ✓ Étant donné un réseau de Petri et un marquage m, on peut décider si l'ensemble d'accessibilité contient un marquage $m_0 > m$.

Le graphe de couverture étant fini, on peut théoriquement le construire puis l'exploiter pour en déduire des propriétés. Cependant, les possibilités d'analyse à l'aide de ce graphe sont limitées. Par exemple, il permet de décider s'il existe un marquage accessible plus grand qu'un autre donné, mais pas si un marquage est accessible.
Nous présentons maintenant des techniques plus « fines » permettant de réduire les graphes de comportement en fonction des propriétés à vérifier ou des particularités du comportement initial.

B-Abstraction de comportement. Certains détails du système étudié peuvent se révéler inutiles au vu des propriétés à vérifier. Les méthodes d'abstraction de comportement définissent un cadre formel permettant de considérer des problèmes simplifiés. Ces techniques utilisent des relations dites d'abstraction entre le graphe initial et le graphe réduit. Ces relations doivent conserver dans une certaine mesure les relations de transition entre nœuds, de façon à ce que les propriétés du graphe réduit construit soient représentatives des propriétés du graphe initial. Le lecteur intéressé pourra se reporter à la thèse de Loiseaux [Loiseaux, 1994] pour une bibliographie complète sur le sujet.

L'abstraction est une des rares méthodes permettant de réduire des graphes de comportement infinis en des graphes finis. De plus, les résultats de préservation concernent

une large classe de propriétés à vérifier. Cependant, la construction du système abstrait freine une plus large diffusion de cette méthode. En effet, celle-ci n'est pas automatique et requiert des démarches spécifiques. Les recherches s'orientent donc actuellement sur l'identification de schémas généraux d'abstraction permettant d'automatiser ce type de démarches.

C- Ordre partiel. Les techniques d'ordre partiel (cf. [Godefroid, 1996] pour un panorama complet) ont été développées pour limiter l'explosion combinatoire en s'attaquant à l'une de ses causes : la représentation du parallélisme par l'entrelacement d'actions. En effet, lorsque deux actions indépendantes mènent au même état global, on reporte l'entrelacement de celles-ci dans le graphe de comportement. Ces deux actions sont indépendantes et mènent donc au même état global.

Formellement, deux actions sont dites indépendantes si et seulement si :

$$\forall p, p_a, p_b \; états: p \xrightarrow{a} p_a \wedge p \xrightarrow{b} p_b \Longrightarrow \exists \, p' état: p_a \xrightarrow{b} p' \wedge p_b \xrightarrow{a} p'$$

Deux actions indépendantes sont dites en conflit. Il est possible d'utiliser cette relation d'indépendance pour définir une relation d'équivalence sur les séquences d'actions du graphe de comportement : deux séquences d'actions sont équivalentes si et seulement si elles peuvent être obtenues l'une à partir de l'autre par permutations d'actions adjacentes et indépendantes.
La terminologie « ordre partiel » vient du fait que la classe d'équivalence correspond à un ordre partiel sur les occurrences d'actions. L'idée est de réduire le graphe de comportement grâce à l'équivalence de trace. Il existe essentiellement deux types d'approches pour ces méthodes donnant des relations de réduction différentes :

- ✓ **Elimination de l'entrelacement**. La première cherche à obtenir un sous-graphe du graphe de comportement comportant le moins de traces équivalentes possibles. C'est par exemple le cas de l'approche de Valmari sur les stubborn sets [Valmari, 1991].
- ✓ **Pas couvrant** La deuxième solution consiste à agglomérer des ensembles d'actions indépendantes. Ces ensembles d'actions forment des pas. Le graphe obtenu est le graphe de pas couvrants, introduit par Vernadat [Vernadat et *al*, 1996].

Les méthodes d'ordre partiel sont très avantageuses sur deux points :

- ✓ les graphes obtenus sont souvent très réduits par rapport aux graphes initiaux ;
- ✓ Le temps nécessaire au calcul des relations d'indépendance est faible par rapport à la complexité du système.

En conséquence, ces techniques permettent de réaliser une économie à la fois sur le temps et l'espace mémoire nécessaire à la génération du graphe de comportement.

D- Symétries. Les systèmes étudiés possèdent souvent des symétries d'architecture. L'élimination de toutes les situations symétriques et donc redondantes dans le comportement du système permettent d'en simplifier l'étude. Huber et al [Huber et *al*, 1985] ont formalisé cette notion de symétrie afin de réduire la taille du graphe d'accessibilité. Ils introduisent donc une relation d'équivalence, appelée symétrie, entre les nœuds du graphe d'une part, et les arcs d'autre part. Cela donne lieu à la construction d'un graphe réduit dont chaque nœud représente une classe d'équivalence des nœuds du graphe d'accessibilité.

Comme précédemment, le passage au graphe quotient permet de préserver certaines propriétés du graphe de départ. Par exemple, les propriétés portant sur la structure du graphe sont généralement des propriétés symétriques. Des analyses peuvent alors être menées à l'aide de ce graphe réduit. Les propriétés que l'on peut vérifier sur le graphe quotient sont les propriétés symétriques : un état satisfait une propriété symétrique si et seulement si tout état symétrique la satisfait aussi.

Puisque le graphe de comportement n'est pas disponible, il faut être capable de déduire des symétries directement de la description dont on dispose. Pour cela, on identifie des symétries de description et on prouve par la sémantique transitionnelle que ces symétries se transforment bien en symétries du graphe de comportement

De manière pratique, les approches se limitent à certaines catégories de symétries pour lesquelles il est plus aisé de déterminer les classes d'équivalence nécessaires au calcul du graphe quotient.

Les méthodes que nous venons de décrire permettent d'analyser un système. Les méthodes de réductions permettent de diminuer la taille du réseau tout en préservant certaines propriétés du réseau : dans la plupart des cas on perd les séquences de tirs. Il en est de même pour les méthodes qui utilisent les symétries du réseau. On peut déterminer si un marquage est accessible, mais on n'obtient pas tout à fait les séquences de tir : plus précisément les arcs du graphe compressé sont des classes d'équivalence des arcs du graphe original. Les techniques d'ordre partiel consistent à n'explorer qu'une partie du graphe d'accessibilité.

Notre problème ne consiste pas seulement à déterminer si un marquage donné est accessible à partir d'un autre. Ce sont les **séquences de tir** qui mènent au marquage désiré qui nous intéressent [Jabri et *al*, 2010]. Ainsi, plutôt que de développer le graphe des marquages possibles, nous utilisons une **approche d'abstraction logique basée sur les contraintes** proposée par Bennaser dans sa thèse [Bennaser, 2000] pour la recherche des séquences de tir. Pour cela nous allons capturer le comportement du réseau par une structure qu'on appelle steps partiels. Nous allons voir que la recherche des séquences de tir du graphe d'accessibilité se ramène alors à une recherche dans une séquence de steps partiels. De plus,

la notion de marquage partiel permet d'obtenir les séquences de tir pour plusieurs marquages initiaux, ou pour un marquage initial qui n'est pas totalement connu.

4.3 Génération de séquences de franchissement

Le problème de l'accessibilité dans les réseaux de Petri nous intéresse dans la mesure où nous cherchons à générer des scénarios de test à travers la recherche de chemins dans le graphe d'accessibilité. Plus exactement, la spécification des composants à tester est modélisée à l'aide des réseaux de Petri. Générer des tests à partir de cette spécification consiste à déterminer des séquences de tirs entre deux marquages. Le scénario de test est ainsi sous la forme d'une séquence de tir et le marquage final atteint détermine les sorties du scénario de test à vérifier. La comparaison de ces sorties avec celles générées par le simulateur permettent de déterminer les verdicts de test.

Dans nos travaux, une simple accessibilité d'un marquage ne nous suffit pas étant donné que nous voulons générer les scénarios de test ; c'est-à-dire les chemins qui mènent au marquage final recherché. Nous avons mentionné que la recherche de ces chemins à travers l'exploration du graphe d'accessibilité mène à une explosion combinatoire. Pour éviter cette explosion, nous nous inspirons de la méthode développée par Bennaser [Bennaser, 2000] pour générer nos scénarios de test. Cette méthode permet de construire une séquence contenant autant d'information que le graphe d'accessibilité. A partir de cette séquence, il est possible de montrer l'accessibilité d'un marquage et de trouver les séquences de tirs qui permettent de l'atteindre.

La formalisation décrite dans ces travaux est fondée sur les notions de **«step»** et de **séquences de «steps»** que nous détaillons par la suite. Informellement, un step représente les tirs simultanés et réentrants d'un ensemble de transitions. Benasser utilise deux structures qui permettent à l'aide de variables et de contraintes, de représenter un ensemble de marquages ou de steps : les **marquages «partiels»** et les **steps «partiels»**. Informellement, il s'agit de considérer les steps et les marquages intermédiaires comme des vecteurs de variables, chacun associé à une formule. Les formules correspondent à des contraintes imposées aux variables de façon à assurer que l'ensemble des instanciations possibles de celles-ci représentent toujours des marquages et des steps concrets valides. Les deux notions de marquage partiel et step partiel permettent de capturer le comportement total du réseau de Petri dans une séquence unique de steps partiels.

Résolution du problème d'accessibilité par l'abstraction logique. Si les steps partiels sont bien choisis, l'indéterminisme, dû au fait que plusieurs steps sont franchissables, sera

capturé par un step partiel. Ainsi, le graphe d'accessibilité peut alors être réduit à une séquence de steps partiels. On peut utiliser cette séquence pour la recherche des séquences de tirs qui, à partir du marquage initial, produisent un marquage donné. Il suffit de rajouter une contrainte sur le dernier marquage partiel obtenu, de façon à ce qu'il s'unifie avec le marquage désiré, et de lancer un mécanisme de résolution de contraintes linéaires. L'exploration du graphe des marquages accessibles est en fait repoussée au niveau de la résolution des contraintes. L'intérêt de cette technique est d'éviter d'explorer les branches du graphe qui ne mènent pas au marquage final désiré.

Dans la section suivante, nous décrivons les différentes étapes de cette technique d'abstraction logique en commençant par la définition des notions de « steps » et de séquences de « steps ».

4.3.1 Technique d'abstraction pour la génération de séquences de tirs

4.3.1.1 Step (étape) et séquences de Step

Nous généralisons la notion de franchissement de transition de la section 4.2.2.1 au franchissement d'un « **step ou étape**». Un « step » dénote un ensemble de transitions simultanément validées pour le marquage. Quel que soit l'ordre du tir de toutes les transitions d'une étape valide, on obtient clairement le même marquage et on peut donc définir le tir d'une étape valide. Autrement dit, le tir d'une étape dénote le même comportement que n'importe quelle séquence de tir constituée d'une permutation des transitions de l'étape. Une même transition peut être franchie plusieurs fois (nous parlons de réentrance). Le tir d'un step peut être interprété comme le tir simultané de toutes les transitions apparaissant dans le step [Yim, 2000].

Plus précisément, un « step » est un multi-ensemble sur l'ensemble des transitions. Rappelons qu'un multi-ensemble sur un ensemble E est une application de l'ensemble E vers l'ensemble des entiers naturels $\mathbb{N}$ et qu'on note habituellement un multi-ensemble sous la forme de somme symbolique. Un « step » peut contenir plusieurs exemplaires d'un même élément, par exemple $\{2t_1, t_2\}$. Nous pouvons le noter tout simplement ainsi : $2t_1 + t_2$.

Un « step » $\varphi = \alpha_1 t_1 + \cdots + \alpha_n t_n$ où $\alpha_1, \ldots, \alpha_n \in \mathbb{N}$ (α_i est le nombre de franchissement de la transition t_i), est franchissable depuis un marquage m si :

$$\forall p \in \mathbb{P}, m(p) \geq \sum_{j=1}^{\mathbb{N}} \alpha_j \mathcal{W}(p, t_j).$$

Le marquage doit contenir suffisamment de jetons pour que chaque transition du « step » puisse consommer ses propres jetons. Nous associons au « step » φ le vecteur caractéristique $\overrightarrow{\varphi}$, comme une combinaison linéaire à coefficients positifs des vecteurs caractéristiques de chaque transition, $\overrightarrow{\varphi} = \sum_{j=1}^{\mathbb{N}} \alpha_j \overrightarrow{e_{t_j}}$.

Les équations (1) et (2) de la section 4.2.2.1 peuvent être généralisées à la notion de « step » et de séquences de step. Dans la suite, nous utiliserons les notations utilisées auparavant : $m_0[\varphi\rangle$, $m_0[\varphi\rangle m_1$, $m_0[\varphi_1\varphi_2 \dots \varphi_k\rangle$, $m_0[\varphi_1\varphi_2 \dots \varphi_k\rangle m_k$ pour indiquer qu'un step ou une séquence de step est franchissable et déterminer le marquage obtenu dans chaque cas. Nous désignons par T^* l'ensemble des steps construits sur T et par $S(P,T,W, m_0)$ l'ensemble des steps franchissables depuis le marquage initial m_0.

Exemple. Dans la figure suivante, nous présentons deux séquences de transitions différentes permettant d'atteindre le marquage (4,1,1,0) depuis le marquage initial (2,1,1,0). La première séquence contient deux fois le step t_1t_2. Cette figure illustre notre objectif de recherche se séquences de transitions entre deux marquages.

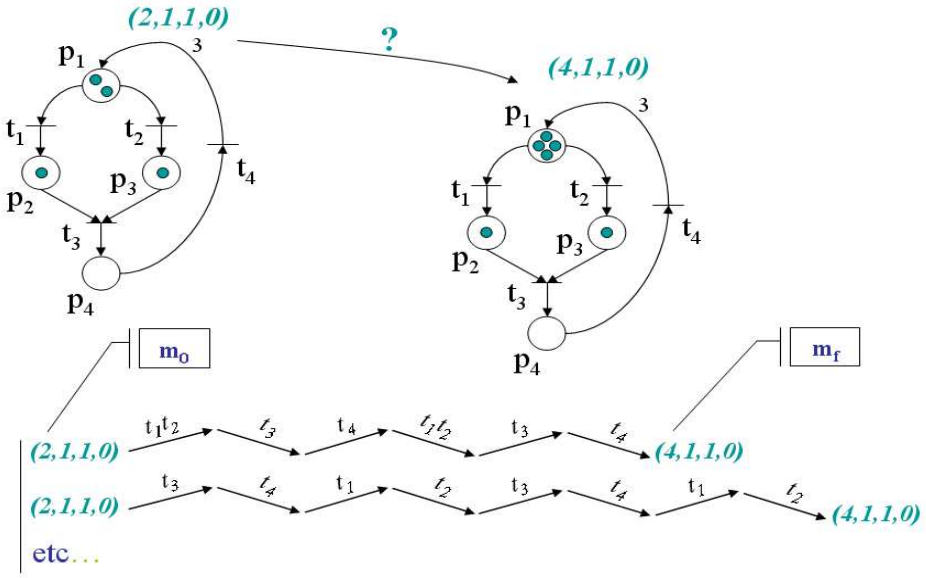


Figure 4.76: Séquences de tir

4.3.1.2 Step partiel et marquage partiel

Le nombre d'états d'un système peut être très important, même pour un système de petite taille. L'énumération complète de l'espace d'états est donc souvent impossible en pratique. Dans l'optique de réduire la taille du graphe d'accessibilité, Bennaser [Bennaser, 2000] a eu l'idée de construire un graphe dont chaque nœud est étiqueté par un **marquage partiel** qui représente un ensemble de marquages. L'objectif est de pouvoir représenter de façon condensée toutes les évolutions possibles du réseau de Petri : les règles de franchissement des transitions et des steps à partir des marquages partiels peuvent être définies. Ces règles de franchissement vérifient certaines propriétés qui garantiront, en particulier, l'obtention d'une représentation complète et condensée des états accessibles ainsi que les chemins qui y mènent.

Les marquages partiels se basent sur des contraintes linéaires. Soit $\mathcal{L}$ un langage du premier ordre avec pour domaine l'ensemble des entiers relatifs $\mathbb{Z}$. Nous notons $\boldsymbol{\varepsilon_{\mathcal{L}}}$ et $\boldsymbol{\mathcal{F}_{\mathcal{L}}}$ respectivement l'ensemble des expressions et des formules de $\mathcal{L}$.

Définition. Un **marquage partiel** est formé d'un vecteur indexé sur l'ensemble des places et d'une contrainte linéaire. Les composantes de ce vecteur sont des expressions contenant des variables qui sont liées par la contrainte. Cette structure permet de représenter des marquages obtenus par une instanciation des variables satisfaisant la contrainte [Bennaser, 2000].

De manière formelle, le marquage partiel est un couple $\boldsymbol{PM} = (\boldsymbol{m}, \boldsymbol{F_m})$ où :

- ✓ $\boldsymbol{m}: \mathbb{P} \mapsto \varepsilon_{\mathcal{L}}$ est une application qui associe à toute place une expression du langage $\mathcal{L}$.
- ✓ $\boldsymbol{F_m}$ est une formule de $\mathcal{L}$.

Quant à la notion de step partiel, elle caractérise un ensemble de steps. En effet, un système qui se trouve dans un état peut évoluer de plusieurs manières différentes. La structure du step partiel [Bennaser, 2000] a été introduite afin d'encapsuler ces différentes évolutions possibles. Il s'agit du même principe que celui du marquage partiel où les composantes contiennent des variables soumises à une contrainte linéaire. Au niveau du step partiel sont introduites d'une part des variables au niveau des composantes des steps, et d'autre part une contrainte linéaire liant ces variables.

Définition. Un **step partiel** est formé d'un vecteur indexé sur l'ensemble des transitions et d'une contrainte linéaire. Les composantes de ce vecteur sont des expressions contenant des variables soumises à la contrainte linéaire. Cette structure permet de représenter un ensemble de steps obtenus par des instanciations des variables respectant la contrainte [Bennaser, 2000].

De manière formelle, le step partiel est un couple $\boldsymbol{PS} = (\boldsymbol{\varphi}, \boldsymbol{F_\varphi})$ où :

- ✓ $\boldsymbol{\varphi}: \mathbb{T} \mapsto \varepsilon_{\mathcal{L}}$ est une application qui associe à toute transition une expression du langage $\mathcal{L}$.
- ✓ $\boldsymbol{F_\varphi}$ est une formule de $\mathcal{L}$.

Dans la suite, nous définissons la règle de tir d'un step partiel à partir d'un marquage partiel qui permet de représenter de façon agrégée un ensemble d'évolutions possibles à partir d'un ensemble d'états. Le graphe d'accessibilité est réduit en une séquence de steps partiels.

Franchissement d'un step partiel à partir d'un marquage partiel.

Nous désignons par $PS(R)$ et $PM(R)$, les ensembles de steps partiels et de marquages partiels. Les propriétés de franchissement peuvent être facilement étendues aux steps et

marquages partiels. Soit $R = (\mathbb{P}, \mathbb{T}, \mathcal{W})$ un réseau de Petri, $PM_0 = (m_0, F_{m_0})$ un marquage partiel de l'ensemble des marquages partiels $PM(R)$, et $PS_1 = (\varphi_1, F_{\varphi_1})$ un step partiel de l'ensemble des steps partiels $PS(R)$.

Comme pour le franchissement d'un step, le franchissement d'un step partiel PS_1 à partir d'un marquage partiel est soumis à la contrainte linéaire suivante :

$$F_m \wedge F_{\varphi_1} \wedge \left(\bigwedge_{p \in \mathbb{P}} \left(m(p) \geq \sum_{t \in \mathbb{T}} \mathcal{W}(p,t)\varphi_1(t) \right) \right)$$

Les formules $F_m et\ F_{\varphi_1}$ permettent d'extraire, du step partiel et du marquage partiel, un step et un marquage. La deuxième partie de la contrainte linéaire garantit que le step est franchissable à partir du marquage.

Le step partiel PS_1 est franchissable à partir du marquage partiel PM_0 (on note $PM_0[PS_1\rangle$) si et seulement si la formule :

$$F_{m_0} \wedge F_{\varphi_1} \wedge \left(\bigwedge_{p \in \mathbb{P}} \left(m_0(p) \geq \sum_{t \in \mathbb{T}} \mathcal{W}(p,t)\varphi_1(t) \right) \right) est\ satisfiable$$

Dans ces conditions, le franchissement du step partiel PS_1 permet de créer le marquage partiel PM_1, (on note $PM_0[PS_1\rangle\, PM_1$) défini par $PM_1 = (m_1, F_{m_1})$ où :

- ✓ $\forall p \in \mathbb{P}, m_1(p) \stackrel{\text{def}}{=} m_0(p) - \sum_{t \in \mathbb{T}} \mathcal{W}(p,t)\varphi_1(t) + \sum_{t \in \mathbb{T}} \mathcal{W}(t,p)\varphi_1(t)$ **(E₁)**
- ✓ $F_{m_1} \stackrel{\text{def}}{=} F_{m_0} \wedge F_{\varphi_1} \wedge \left(\wedge_{p \in \mathbb{P}} (m_0(p) \geq \sum_{t \in \mathbb{T}} \mathcal{W}(p,t)\varphi_1(t)) \right)$ **(E₂)**

Exemple de franchissement d'un step partiel. Soit le réseau de Petri $R = (\mathbb{P}, \mathbb{T}, \mathcal{W})$ de la figure 4.77 avec $\mathbb{P} = (p_1, p_2, p_3, p_4)$ et $\mathbb{T} = (t_1, t_2, t_3, t_4)$.
Nous avons : $m_0(p_1) = 2$; $m_0(p_2) = 1$; $m_0(p_3) = 1$; $m_0(p_4) = 0$ et $F_{m_0} = True$

$$\varphi_1(t_1) = N_1 \ ; \ \varphi_1(t_2) = N_2 \ ; \varphi_1(t_3) = N_3 \ ; \ \varphi_1(t_4) = N_4$$

Le franchissement de $PS_1 = (\varphi_1, F_{\varphi_1})$ à partir de $PM_0 = (m_0, F_{m_0})$ permet de créer le marquage partiel $PM_1 = (m_1, F_{m_1})$ comme le montre la figure 4.77.

L'application de l'équation **(E₁)** permet d'obtenir $\boldsymbol{m_1}$:

- ✓ $m_1(p_1) = 2 - N_1 - N_2 + 3.N_4$;
- ✓ $m_1(p_2) = 1 - N_3 + N_1$;
- ✓ $m_1(p_3) = 1 - N_3 + N_2$;
- ✓ $m_1(p_4) = -N_4 + N_3$;

L'application de l'équation **(E2)** permet d'obtenir F_{m_1}:

$$F_{m_1} = (2 - N_1 - N_2 \geq 0) \wedge (1 - N_3 \geq 0) \wedge (-N_4 \geq 0)$$

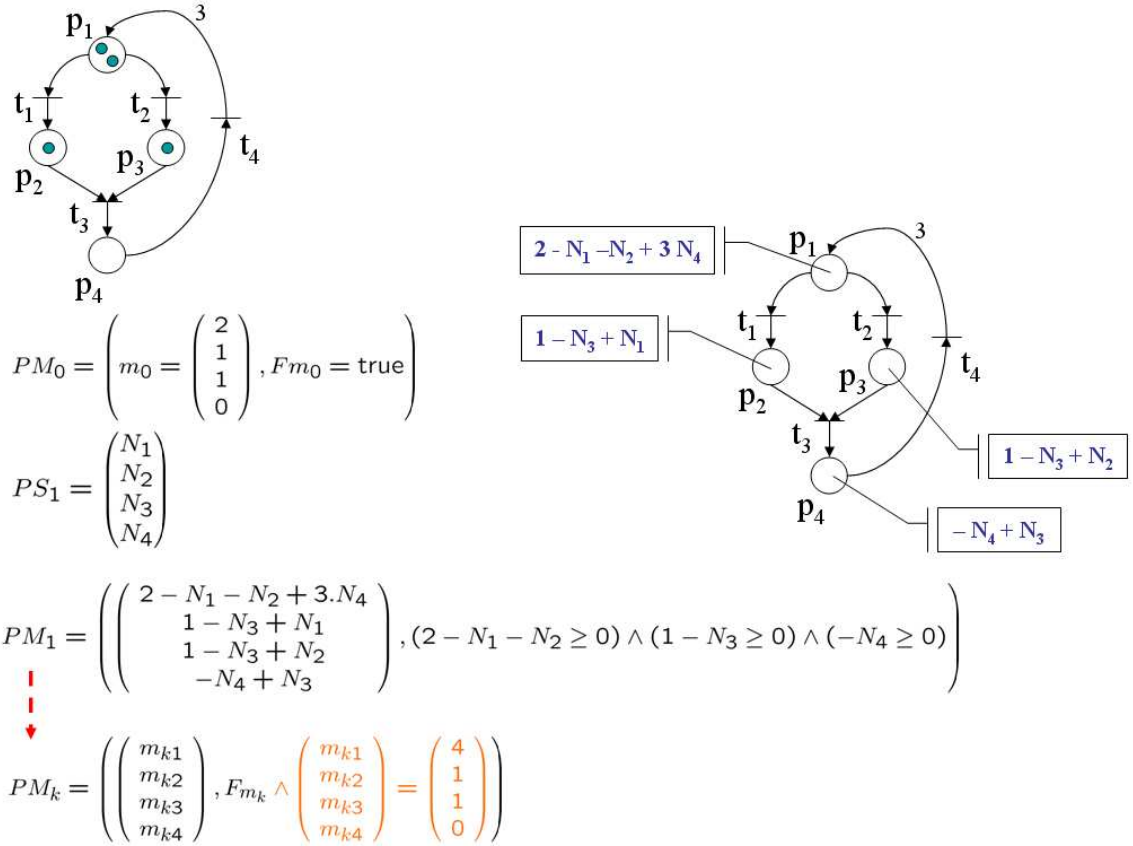


Figure 4.77: Franchissement d'un step partiel à partir d'un marquage partiel

4.3.1.3 Séquences complètes de steps partiels

Pour capturer le comportement complet d'un réseau de Petri, Bennaser [Bennaser, 2000] utilise une approche qui consiste à ne construire qu'une séquence de steps partiels. En effet, si une séquence de steps partiels est quelconque, alors elle ne capture pas forcément tous les marquages accessibles et toutes les séquences franchissables. Dans la suite, nous définissons une séquence de steps partiels et nous déterminons les conditions nécessaires pour qu'elle soit complète à partir du marquage partiel PM_0. Les séquences complètes de steps partiels vont être utilisées pour déterminer toutes les séquences de steps d'une longueur donnée.

Soit un réseau de Petri $R = (\mathbb{P}, \mathbb{T}, \mathcal{W})$. Soit $\boldsymbol{PSS} = PS_1\, PS_2 \ldots PS_k$ une **séquence de steps partiels** de l'ensemble $PS(R)$; $\forall\, i \in [\![1,k]\!]$, $SP_i = (\varphi_i, F_{\varphi_i})$. Soient $PM_1, PM_2, \ldots, PM_k$ des marquages partiels de l'ensemble $PM(R)$,$\forall\, i \in [\![1,k]\!]$, $PM_{i-1}[PS_i\rangle\, PM_i$. Nous désignons par $\varphi_{01}, \varphi_{02}, \ldots \varphi_{0M}$ les variables qui interviennent dans le marquage partiel PM_0 et par $\mathcal{V}_{\varphi i}$ celles qui interviennent dans le step partiel PS_i pour i $\in [\![1,k]\!]$.

Si la séquence de steps partiels PSS satisfait les conditions suivantes, alors elle est considérée complète à partir du marquage partiel PM_0 :

- ✓ $\forall\, i \in [\![1,k]\!], \mathcal{V}_{\varphi i} = \{\varphi_{i1}, \varphi_{i2}, \dots, \varphi_{iN}\}$;
- ✓ $\forall\, i \in [\![1,k]\!], \forall\, j \in [\![1,N]\!], \varphi_i(t_j) = \varphi_{ij}$;
- ✓ Les symboles des variables $\varphi_{ij},\ pour\ i \in [\![0,k]\!]\ et\ j \in [\![1,N]\!]$ sont différents ;
- ✓ $\forall\, i \in [\![1,k]\!], F_{\varphi i} \equiv \bigwedge_{j=1}^{N}(\varphi_{ij} \geq 0)$. **(E3)**

Bennaser [Bennaser, 2000] a prouvé qu'une séquence complète de steps partiels permet de capturer le comportement du réseau de Petri aussi bien du point de vue séquence de steps que du point de vue marquages accessibles. Plus précisément :

- ✓ Toute séquence de steps de longueur k correspond à une instanciation d'une séquence complète de k steps partiels.
- ✓ Chaque énumération d'une séquence complète de steps partiels correspond à une séquence valide de steps.

De plus, d'un point de vue marquages accessibles, les marquages partiels produits par la séquence de steps partiels représentent tous les marquages accessibles dans au plus k steps.

Formulation du problème d'accessibilité. La séquence de steps partiels peut être utilisée pour :

- ✓ la recherche de ***séquences franchissables*** qui permettent de produire un ***marquage final*** m_f depuis un ***marquage initial*** m_0 , ou
- ✓ la recherche de séquences franchissables qui permettent de produire un ***marquage partiel*** m'_f depuis un ***marquage initial*** m_0, ou
- ✓ la recherche de ***toutes les séquences franchissables depuis un marquage initial*** m_0.

De manière formelle, « Soit $(\mathbb{P}, \mathbb{T}, \mathcal{W}, m_0)$ un réseau de Petri, $k \in \mathbb{N}$ et m_f un marquage de $\mathbb{N}^M$. Trouver toutes les séquences de steps permettant d'atteindre le marquage m_f depuis le marquage m_0 dans au plus k steps, ou toutes les séquences de steps permettant d'atteindre le marquage partiel m'_f depuis le marquage m_0 ou bien toutes les séquences de steps franchissables depuis le marquage m_0 ». **(P1)**

En fait, il est suffisant d'utiliser k steps partiels, de remplacer la formule F_{mk} concernant le dernier marquage partiel par (m_k, F_{mk}) par la formule F' définie par :

- $F' \equiv F_{mk} \wedge (m_k = m_f)$, (Dans le cas d'un marquage final connu)
- $F' \equiv F_{mk} \wedge (m_k = m'_f)$, (Dans le cas d'un marquage partiel)
- $F' \equiv F_{mk}$. (Dans le cas de génération de toutes les séquences depuis m_0)

Et de résoudre le système d'équations associé.

Ainsi, l'exploration du graphe d'accessibilité et la résolution du problème d'accessibilité correspondant sont réduites à la résolution du système d'équations. L'intérêt de cette technique est par exemple d'éviter d'explorer les branches du graphe qui ne mènent pas au marquage final désiré. Bennaser [Bennaser, 2000] a proposé un algorithme permettant de résoudre le problème d'accessibilité en utilisant l'abstraction logique et les techniques de programmation par contraintes. Cet algorithme est correct dans la mesure où les séquences fournies sont effectivement des séquences de steps qui produisent le marquage final désiré. Il est également complet dans le sens où il peut énumérer toutes les solutions d'une longueur donnée.

Cette première étape de génération de séquences de franchissement en se basant sur la technique d'abstraction logique nous génère l'ensemble de toutes les séquences de steps de longueur donnée depuis le marquage initial. Comme nous l'avons mentionné plus haut, l'algorithme utilisé nous permet également de générer les séquences qui mènent à un marquage donné connu ou bien à un marquage partiel. En se basant sur ces trois possibilités, nous construisons nos scénarios de test à partir de l'ensemble des séquences générées [Jabri et *al*, 2010].

Cependant, comme nous l'avons précédemment expliqué, cette première étape de génération de séquences se base sur des réseaux de Petri simples. Nous avons en effet ignoré les prédicats (conditions) ainsi que les opérations associées aux transitions de notre modèle réseau de Petri interprété. Ce denier formalisme a été choisi dans le chapitre 3 et cela dans le but de formaliser la spécification ERTMS.

Dans la section suivante, nous présentons la deuxième étape de notre méthodologie concernant le filtrage de l'ensemble des séquences générées lors de la première étape. Cette deuxième étape de filtrage permet d'inclure les caractéristiques des réseaux de Petri interprétés à savoir les prédicats et les opérations.

4.3.2 Filtrage de l'ensemble des séquences de franchissement

Dans un réseau de Petri interprété, les prédicats sont des conditions associées aux transitions. La définition formelle de cette caractéristique a été décrite dans le chapitre 3 en section 3.3.2.3. Un prédicat qui est valide permet le franchissement de la transition qui doit être également valide. Par contre, un prédicat qui n'est pas valide ne permet pas le franchissement de la transition même si cette dernière est franchissable. Les séquences de franchissement générées lors de la première étape de notre démarche peuvent contenir des séquences qui ne sont pas valides étant donné que les prédicats associés à ces transitions n'ont pas été pris en compte.

4.3.2.1 Prédicats associés aux transitions

Nous avons défini un algorithme permettant de valider l'ensemble des séquences générées. Nous considérons une liste de conditions (ou prédicats) représentant une conjonction de conditions. Les expressions possibles dans les conditions sont normalisées. Nous étendons l'état d'un réseau de Petri à un couple (marquage, variables). Les séquences générées peuvent être franchies structurellement mais reste à valider les éventuelles conditions. En effet, les conditions qui ne sont pas validées éliminent les transitions correspondantes.

L'ensemble des séquences générées est composé de plusieurs séquences. Chaque séquence de franchissement est composée par plusieurs steps de franchissement. Chaque step est composé par plusieurs transitions franchies simultanément. Une transition est composée par un prédicat et une opération. Les étapes de validation de l'ensemble des séquences sont décrites dans la figure 4.78. En effet, pour valider un ensemble de séquences de franchissement, il faut parcourir cet ensemble pour valider les séquences une à une. Ensuite, pour valider une séquence, nous devons valider les steps de franchissement composant cette séquence. Enfin, les steps à leur tour sont composés de transitions qu'il faut valider également une à une en vérifiant les conditions associées par rapport à l'état courant du réseau de Petri.

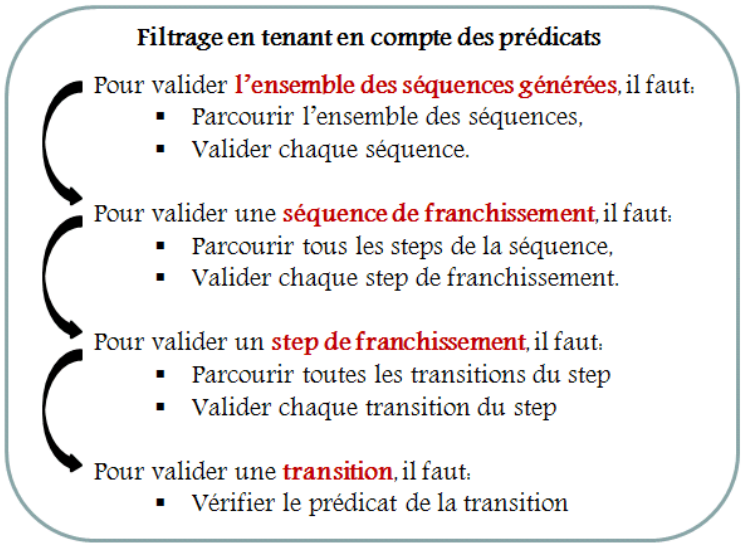


Figure 4.78: Filtrage des séquences de franchissement

4.3.2.2 Opérations associées aux transitions

Dans la section 3.3.2.3 du chapitre 3, nous avons défini des opérations associées aux transitions du réseau de Petri interprété. Une opération s'exécute au franchissement d'une transition et correspond à l'affectation d'une nouvelle valeur à la variable concernée. Comme nous l'avons décrit précédemment, l'état du réseau de Petri est défini par un couple (marquage, variables système). L'exécution d'une opération change la valeur d'une variable donnée. Ainsi l'état courant du réseau change vers un nouvel état.

Les séquences de franchissement filtrées (en ajoutant les conditions des transitions) contiennent des steps de franchissement. Le step correspond à un franchissement simultané de plusieurs transitions. En prenant en compte les opérations associées à ces transitions, des situations de conflit peuvent apparaître. Par exemple, le franchissement parallèle de deux transitions engendre une exécution parallèle des deux opérations d'affectation associées à ces transitions. S'il s'agit de la même variable, nous nous retrouvons alors simultanément dans deux états différents du réseau de Petri.

Dans [David&Alla, 1992], les auteurs ont résolu le problème de conflit d'opérations de cette manière. Ils considèrent l'ensemble des états de l'environnement $D=D_1\times D_2\times\ldots D_k$. C'est-à-dire que l'environnement contient un ensemble de k objets dont les D_i représentent les états (si par exemple les objets sont les variables, alors D_i représente le domaine de définition de la i-ème variable). Aussi, pour chaque opération Op_i est donnée un ensemble d'indices L_i, $L_i \subseteq \{1,2\ldots, k\}$ et tel que $Op_i : D_w \rightarrow X_{w\in Li}$. C'est-à-dire, lors de son activation, l'opération Op_i ne transforme que des objets correspondant aux indices de L_i. Deux opérations Op_i et Op_j sont dits **compatibles** si et seulement si $L_i\cap L_j=\varnothing$. Evidemment, lorsque deux opérations Op_i et Op_j sont compatibles, elles transforment des ensembles d'objets disjoints et on peut les activer en parallèle. En effet, si $d=<d_1, d_2, \ldots, d_k>$ est l'état de l'environnement à un instant donné et deux opérations compatibles Op_i et Op_j sont activées en parallèle à partir de d, alors l'état $d'=<d'_1, d'_2, \ldots, d'_k>$ atteint après leur activation est tel que $d'_s = d_s$ si $s \notin L_i\cup L_j$ et d's égal à la composante de même indice de Op_i ($<d_w>_{w\in Li}$) si $s \in L_i$ ou de Op_j ($<d_w>_{w\in Lj}$) si $s\in L_j$.

Par contre si Op_i et Op_j ne sont pas compatibles, on ne peut pas en principe savoir quel est l'état atteint par l'environnement lorsqu'elles sont appliquées en parallèle. Cette relation de compatibilité entre opérations permet de déterminer des ensembles d'opérations activables en parallèle : ce sont des ensembles d'opérations deux à deux compatibles étant donné que la relation de comptabilité n'est pas transitive (cf. Annexe F).

Dans nos travaux, nous considérons que les opérations opèrent sur le même ensemble de variables. Nous ne construisons pas des ensembles disjoints de variables afin de définir des opérations compatibles entre elles. Nous avons plutôt choisi de développer les séquences de steps de manière à ce qu'une seule transition soit franchie à la fois. C'est-à-dire dans nos séquences de steps filtrées et valides (suite à l'intégration des prédicats des transitions), **nous autorisons le franchissement d'une seule transition à la fois dans un step donné**. Nous évitons ainsi le conflit engendré par exemple par l'exécution de deux opérations à la fois.

Dans cette deuxième étape de notre méthodologie, nous avons introduit les caractéristiques particulières des réseaux de Petri interprétés. Cette étape que nous avons appelé filtrage

nous permet d'obtenir des séquences de franchissement valides. Or nous rappelons que l'objectif primordial de ces travaux de thèse est de tester des composants communiquant entre eux. Les réseaux de Petri les modélisant sont connectés à travers un ensemble de transitions que nous avons appelé ensemble des transitions aux interfaces (cf. section 3.3.3.1).

Tester ces composants revient à générer les séquences de franchissement des transitions aux interfaces, d'où le rôle de la troisième étape de notre méthodologie de génération de scénarios de test et que nous avons appelé : étape de réduction de l'ensemble des séquences de franchissement.

4.3.3 Réduction de l'ensemble des séquences de franchissement

Cette troisième et dernière étape de notre méthodologie concerne l'élimination des transitions internes. L'objectif consiste à réduire les séquences de franchissement en ne gardant que les transitions aux interfaces. De manière plus formelle : considérons la séquence de franchissement $\beta_1 = t_{\beta_{11}} t_{\beta_{12}} t_{\beta_{13}} \dots t_{\beta_{1F}}$ avec $\|\beta_1\| = \{t_{\beta_{11}}, t_{\beta_{12}}, t_{\beta_{13}}, \dots t_{\beta_{1F}}\}$ est l'ensemble des transitions contenues dans la séquence β_1.
Nous désignons par $\mathbb{T}$ l'ensemble de toutes les transitions du réseau et par $\mathbb{TI}$ l'ensemble des transitions aux interfaces. $\mathbb{TI} = \{Transitions\ aux\ interfaces\}\ et\ \mathbb{TI} \subset \mathbb{T}$.

Soit la séquence de franchissement réduite $\beta'_1 = t_{\beta'_{11}} t_{\beta'_{12}} \dots t_{\beta'_{1R}}$.

Les transitions $\{t_{\beta'_{11}}, t_{\beta'_{12}}, \dots t_{\beta'_{1R}}\} \in \|\beta_1\|$.

(Les transitions aux interfaces sont incluses dans la liste des séquences filtrées)

$$\forall\, i \in [\![1, R]\!], \qquad t_{\beta'_{1i}} ;\ t_{\beta'_{1i+1}} \in \|\beta_1\|$$

$$\exists\, (j,k) \in [\![1, F]\!]\ avec\ j < k\ , tel\ que\ t_{\beta'_{1i}} = t_{\beta_{1j}}\ \text{Et}\ t_{\beta'_{1i+1}} = t_{\beta_{1k}}$$

(Une séquence réduite comportant des transitions aux interfaces respecte l'ordre de franchissement des transitions dans la séquence filtrée)

$$\nexists\, l\ avec\ j < l < k\ \text{ Tel que } t_{\beta_{1l}} \in \mathbb{TI}$$

(Une séquence réduite ne comporte que les transitions aux interfaces. Les transitions internes sont éliminées)

A la fin de cette dernière étape, les séquences de franchissement obtenues sont des séquences de transitions aux interfaces incluses dans le graphe d'accessibilité du réseau de Petri interprété. Dans la suite, nous présentons un exemple illustrant notre méthode de génération de séquences de franchissement. Suite à cet exemple, nous expliquons comment les scénarios de test seront produits à partir des séquences générées afin d'être exécutés sur la plate-forme de simulation ERTMS.

4.4 Exemple illustratif

Nous reprenons l'exemple développé dans la section 3.4. Dans cet exemple, nous avons transformé les machines d'états UML de deux processus P_1 et P_2 en un seul modèle en réseau de Petri interprété dans l'objectif de formaliser la spécification. Nous utilisons le modèle réseau de Petri obtenu après la transformation pour générer des séquences de franchissement. Ces séquences constituent les scénarios de test qui valideront le composant à tester. Rappelons le contexte de l'exemple : soient deux processus P_1 et P_2 qui communiquent ensemble. P_1 peut ouvrir une session de communication avec P_2 et il reste en attente d'un accusé de réception. P_2 envoie son accusé de réception. Si l'accusé de réception est favorable alors P_1 devient dans un état actif et peut ensuite fermer la session de communication. P_2 ne peut pas fermer la session.

4.4.1 Génération des séquences de test à partir d'un modèle RdP simple

Le modèle de réseau de Pétri interprété développé dans le chapitre est présenté dans la figure suivante

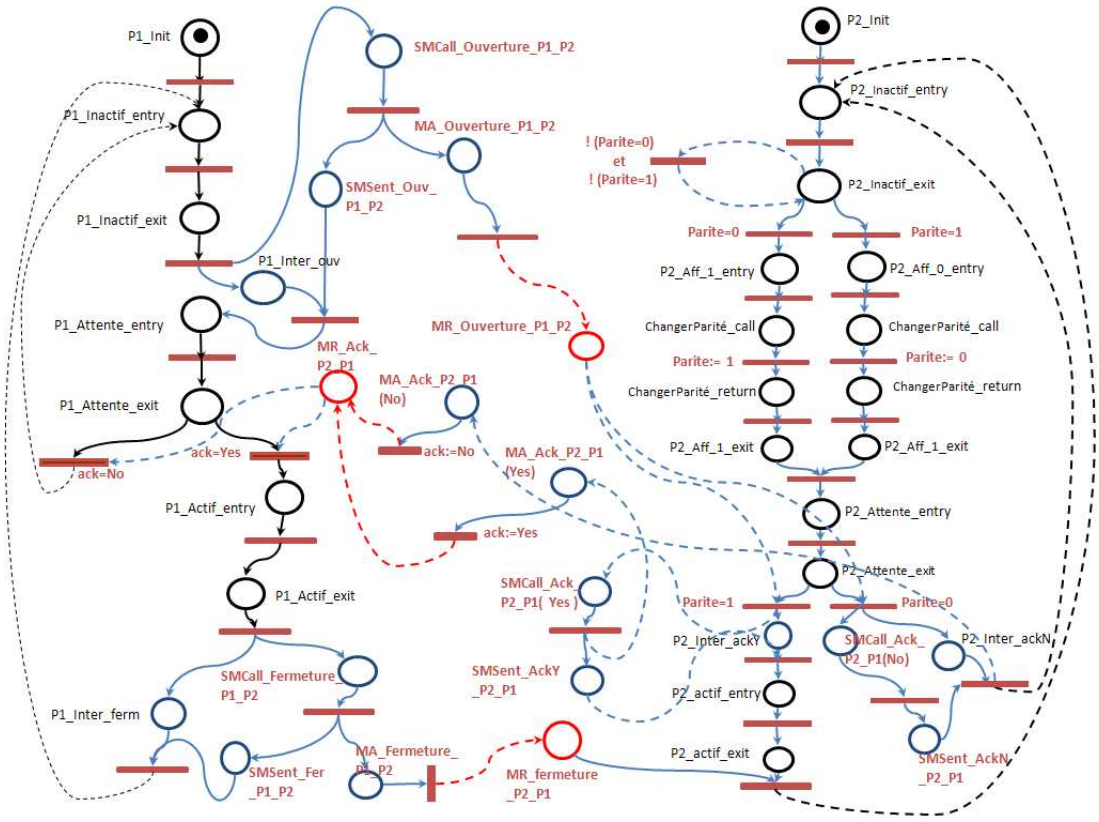


Figure 4.79: Réseau de Pétri interprété de la section 3.4

Afin de pouvoir générer l'ensemble des séquences de franchissement en utilisant la technique d'abstraction logique, nous présentons le modèle avec le langage prolog sous la forme suivante : une liste de transitions reliant des places en amont et des places en aval et un marquage initial (cf. Fig.4.80).

```
transition([transitionPN,[1,0]],[ ],[ ],[[placePN,[p1_init_Exit]]],[[placePN,[p1_inactif_Entry]]]).
transition([transitionPN,[2,0]],[ ],[ ],[[placePN,[p2_init_Exit]]],[[placePN,[p2_inactif_Entry]]]).
transition([transitionPN,[2,1]],[[parite,0]],[ ],[[placePN,[p2_inactif_Exit]]],[[placePN,[p2_affectationA1_Entry]]]).
transition([transitionPN,[2,2]],[[parite,1]],[ ],[[placePN,[p2_inactif_Exit]]],[[placePN,[p2_affectationA0_Entry]]]).
transition([transitionPN,[2,3]],[ ],[ ],[[placePN,[p2_affectationA1_Exit]]],[[placePN,[p2_attente_Entry]]]).
transition([transitionPN,[2,4]],[ ],[ ],[[placePN,[p2_affectationA0_Exit]]],[[placePN,[p2_attente_Entry]]]).
transition(interne_p1_init,[ ],[ ],[[placePN,[p1_init_Entry]]],[[placePN,[p1_init_Exit]]]).
transition(interne_p1_inactif,[ ],[ ],[[placePN,[p1_inactif_Entry]]],[[placePN,[p1_inactif_Exit]]]).
transition(interne_p1_attente_ouv,[ ],[ ],[[placePN,[p1_attente_ouv_Entry]]],[[placePN,[p1_attente_ouv_Exit]]]).
transition(interne_p1_actif,[ ],[ ],[[placePN,[p1_actif_Entry]]],[[placePN,[p1_actif_Exit]]]).
transition(interne_p2_init,[ ],[ ],[[placePN,[p2_init_Entry]]],[[placePN,[p2_init_Exit]]]).
transition(interne_p2_inactif,[ ],[ ],[[placePN,[p2_inactif_Entry]]],[[placePN,[p2_inactif_Exit]]]).
transition(interne_p2_attente,[ ],[ ],[[placePN,[p2_attente_Entry]]],[[placePN,[p2_attente_Exit]]]).
transition(interne_p2_actif,[ ],[ ],[[placePN,[p2_actif_Entry]]],[[placePN,[p2_actif_Exit]]]).
...
transition([transitionPN,[2,7]],[ ],[ ],[[placePN,[p2_actif_Exit]],placePN,[rm_recu_fermeture_p1_p2]]],[[placePN,[p2_inactif_Entry]]]).

transition([transitionPN,[1,1,em_ouverture_p1_p2]],[ ],[ ],[[placePN,[em_appel_ouverture_p1_p2_1_1]]],[[placePN,[em_envoye_ouverture_p1_p2_1_1]],
[placePN,[em_arrive_ouverture_p1_p2_1_1]]]).

transition([transitionPN,[1,1,rm_ouverture_p1_p2]],[ ],[ ],[[placePN,[em_arrive_ouverture_p1_p2_1_1]]],[[placePN,[rm_recu_ouverture_p1_p2]]]).
transition([transitionPN,[1,1,lancement]],[ ],[ ],[[placePN,[p1_inactif_Exit]]],[[placePN,[em_appel_ouverture_p1_p2_1_1]],[placePN,[1,1,placeIntermediaire]]]).
transition([transitionPN,[1,1,synchro]],[ ],[ ],[[placePN,[1,1,placeIntermediaire]],placePN,[em_envoye_ouverture_p1_p2_1_1]]],[[placePN,[p1_attente_ouv_Entry]]]).

transition([transitionPN,[1,3,em_fermeture_p1_p2]],[ ],[ ],[[placePN,[em_appel_fermeture_p1_p2_1_3]]],[[placePN,[em_envoye_fermeture_p1_p2_1_3]],
[placePN,[em_arrive_fermeture_p1_p2_1_3]]]).

transition([transitionPN,[1,3,rm_fermeture_p1_p2]],[ ],[ ],[[placePN,[em_arrive_fermeture_p1_p2_1_3]]],[[placePN,[rm_recu_fermeture_p1_p2]]]).

minitial([[placePN,[p1_init_Entry]],placePN,[p2_init_Entry]]]).
```

Figure 4.80: Modélisation du réseau de Petri interprété sous la forme d'une liste de transitions

Dans cette liste de transitions de la figure 4.80, nous retrouvons des transitions internes et des transitions aux interfaces (colorées en rouge dans la figure 4.79 du réseau de Petri interprété). Notre objectif est de générer des séquences de franchissement à partir du réseau de Petri interprété en ignorant dans une première étape les caractéristiques du réseau de Petri interprété. Pour cette raison, nous avons simplifié la liste précédente de manière à associer à chaque place et à chaque transition des noms plus faciles (cf. Fig.4.81).

```
transition(t1,[ ],[ ],[p19],[p20]).
transition(t10,[ ],[ ],[p10],[p18,p14]).
transition(t14,[[ack,non]],[ ],[p22,p39],[p23]).
transition(t15,[[ack,oui]],[ ],[p22,p39],[p19]).
transition(t16,[ ],[ ],[p9],[p17,p13]).
...
transition(t34,[ ],[parite,1],[p5],[p6]).
transition(t35,[[non,parite,0],[non,parite,1]],[ ],[p34],[p34]).
transition(t6,[ ],[ ],[p33],[p34]).
transition(t7,[ ],[ ],[p35],[p36]).
transition(t8,[ ],[ ],[p37],[p38]).
transition(t9,[ ],[ ],[p26],[p23]).

associationsT([[interne_p1_actif,1],[interne_p1_attente_ouv,2],[interne_p1_inactif,3],[interne_p1_init,4],[interne_p2_actif,5],[interne_p2_attente,6],
[interne_p2_inactif,7],[interne_p2_init,8],[[transitionPN,[1,0]],9],[[transitionPN,[1,1,em_ouverture_p1_p2]],10],[[transitionPN,[1,1,lancement]],11],
[[transitionPN,[1,1,rm_ouverture_p1_p2]],12],[[transitionPN,[1,1,synchro]],13],[[transitionPN,[1,2,choix_1]],14],[[transitionPN,[1,2,choix_2]],15],
[[transitionPN,[1,3,em_fermeture_p1_p2]],16],[[transitionPN,[1,3,lancement]],17],[[transitionPN,[1,3,rm_fermeture_p1_p2]],18],[[transitionPN,[1,3,synchro]],19],
[[transitionPN,[2,0]],20],[[transitionPN,[2,1]],21],[[transitionPN,[2,2]],22],[[transitionPN,[2,3]],23],[[transitionPN,[2,4]],24],
[[transitionPN,[2,5,em_ack_p2_p1]],25],[[transitionPN,[2,5,lancement]],26],[[transitionPN,[2,5,rm_ack_p2_p1]],27],[[transitionPN,[2,5,synchro]],28],
[[transitionPN,[2,6,em_ack_p2_p1]],29],[[transitionPN,[2,6,lancement]],30],[[transitionPN,[2,6,rm_ack_p2_p1]],31],[[transitionPN,[2,6,synchro]],32],
[[transitionPN,[2,7]],33],[[transitionPN,[changerParite]],34],[[transitionPN,[exclusion,p2_attente,[declencheurRecevoirMessage,[ouverture,p1,p2]]]],35],
[[transitionPN,[internel_p2_affectationA0]],36],[[transitionPN,[internel_p2_affectationA1]],37],[[transitionPN,[interne2_p2_affectationA0]],38],
[[transitionPN,[interne2_p2_affectationA1]],39]]).

associationsP([[[placePN,[1,1,placeIntermediaire]],1],[[placePN,[1,3,placeIntermediaire]],2],[[placePN,[2,5,placeIntermediaire]],3],
[[placePN,[2,6,placeIntermediaire]],4],[[placePN,[changerParite_appel]],5],[[placePN,[changerParite_retour]],6],[[placePN,[em_appel_ack_p2_p1_2_5]],7],
[[placePN,[em_appel_ack_p2_p1_2_6]],8],[[placePN,[em_appel_fermeture_p1_p2_1_3]],9],[[placePN,[em_appel_ouverture_p1_p2_1_1]],10],[[placePN,[em_arrive_ack_p2_p1_2_5]],11],
[[placePN,[em_arrive_ack_p2_p1_2_6]],12],[[placePN,[em_arrive_fermeture_p1_p2_1_3]],13],[[placePN,[em_arrive_ouverture_p1_p2_1_1]],14],
[[placePN,[em_envoye_ack_p2_p1_2_5]],15],[[placePN,[em_envoye_ack_p2_p1_2_6]],16],[[placePN,[em_envoye_fermeture_p1_p2_1_3]],17],
[[placePN,[em_envoye_ouverture_p1_p2_1_1]],18],[[placePN,[p1_actif_Entry]],19],[[placePN,[p1_actif_Exit]],20],[[placePN,[p1_attente_ouv_Entry]],21],
[[placePN,[p1_attente_ouv_Exit]],22],[[placePN,[p1_inactif_Entry]],23],[[placePN,[p1_inactif_Exit]],24],[[placePN,[p1_init_Entry]],25],
[[placePN,[p1_init_Exit]],26],[[placePN,[p2_actif_Entry]],27],[[placePN,[p2_actif_Exit]],28],[[placePN,[p2_affectationA0_Entry]],29],
[[placePN,[p2_affectationA0_Exit]],30],[[placePN,[p2_affectationA1_Entry]],31],[[placePN,[p2_affectationA1_Exit]],32],[[placePN,[p2_attente_Entry]],33],
[[placePN,[p2_attente_Exit]],34],[[placePN,[p2_inactif_Entry]],35],[[placePN,[p2_inactif_Exit]],36],[[placePN,[p2_init_Entry]],37],[[placePN,[p2_init_Exit]],38],
[[placePN,[rm_recu_ack_p2_p1]],39],[[placePN,[rm_recu_fermeture_p1_p2]],40],[[placePN,[rm_recu_ouverture_p1_p2]],41]]).

transitions([t1,t2,t3,t4,t5,t6,t7,t8,t9,t10,t11,t12,t13,t14,t15,t16,t17,t18,t19,t20,t21,t22,t23,t24,t25,t26,t27,t28,t29,t30,t31,t32,t33,t34,t35,t36,t37,t38,t39]).
places([p1,p2,p3,p4,p5,p6,p7,p8,p9,p10,p11,p12,p13,p14,p15,p16,p17,p18,p19,p20,p21,p22,p23,p24,p25,p26,p27,p28,p29,p30,p31,p32,p33,p34,p35,p36,p37,p38,p39,p40,p41]).
minitial([0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0]).
```

Figure 4.81: Simplification du réseau de Petri

En appliquant la technique d'abstraction logique, l'ensemble des séquences de steps de longueur maximale 10 généré à partir du réseau de Petri simple est représenté dans la figure suivante.

```
morceau(1,[[[t8]],[[t4]],[[t4,t8]],[[t8],[t20]],[[t8],[t4]],[[t8],[t4,t20]],[[t4],[t9]],[[t4],[t8]],[[t4],[t8,t9]],[[t4,t8],[t20]]]).

morceau(2,[[[t4,t8],[t9]],[[t4,t8],[t9,t20]],[[t8],[t20],[t7]],[[t8],[t20],[t4]],[[t8],[t20],[t4,t7]],[[t8],[t4],[t20]],[[t8],[t4],[t9]],[[t8],[t4],[t9,t20]],
[[t8],[t4,t20],[t9]],[[t8],[t4,t20],[t7]]]).

morceau(3,[[[t8],[t4,t20],[t7,t9]],[[t4],[t9],[t8]],[[t4],[t9],[t3]],[[t4],[t9],[t3,t8]],[[t4],[t8],[t20]],[[t4],[t8],[t9]],[[t4],[t8],[t9,t20]],[[t4],[t8,t9],[t20]],
[[t4],[t8,t9],[t3]],[[t4],[t8,t9],[t3,t20]]]).

morceau(4,[[[t4,t8],[t20],[t9]],[[t4,t8],[t20],[t7]],[[t4,t8],[t20],[t7,t9]],[[t4,t8],[t9],[t20]],[[t4,t8],[t9],[t3]],[[t4,t8],[t9],[t3,t20]],
[[t4,t8],[t9,t20],[t7]],[[t4,t8],[t9,t20],[t3]],[[t4,t8],[t9,t20],[t3,t7]],[[t8],[t20],[t7],[t22]]]).

morceau(5,[[[t8],[t20],[t7],[t21]],[[t8],[t20],[t7],[t4]],[[t8],[t20],[t7],[t4,t22]],[[t8],[t20],[t7],[t4,t21]],[[t8],[t20],[t4],[t9]],[[t8],[t20],[t4],[t7]],
[[t8],[t20],[t4],[t7,t9]],[[t8],[t20],[t4,t7],[t22]],[[t8],[t20],[t4,t7],[t21]],[[t8],[t20],[t4,t7],[t9]]]).
...
```

Figure 4.82: Un sous-ensemble de l'ensemble des séquences de steps

4.4.2 Filtrage et réduction des séquences générées

L'ensemble des séquences de steps est ensuite filtré en considérant les prédicats et les opérations du réseau de Petri interprété (les transitions dont les prédicats ne sont pas vrais sont éliminés et nous autorisons uniquement le franchissement d'une transition à la fois afin d'éviter les situations de conflits d'opérations). Enfin, les transitions internes sont supprimées de l'ensemble filtré des séquences de franchissement et cela afin d'obtenir un ensemble réduit ne contenant que les transitions aux interfaces.

```
valide([[t8]]).
valide([[t4]]).
valide([[t4,t8]]).
valide([[t8],[t20]]).
valide([[t8],[t4]]).
valide([[t8],[t4,t20]]).
valide([[t4],[t9]]).
valide([[t4],[t8]]).
valide([[t4],[t8,t9]]).
valide([[t4,t8],[t20]]).
...
```

Figure 4.83: Filtrage de séquences de franchissement

4.5 Production des scénarios de test

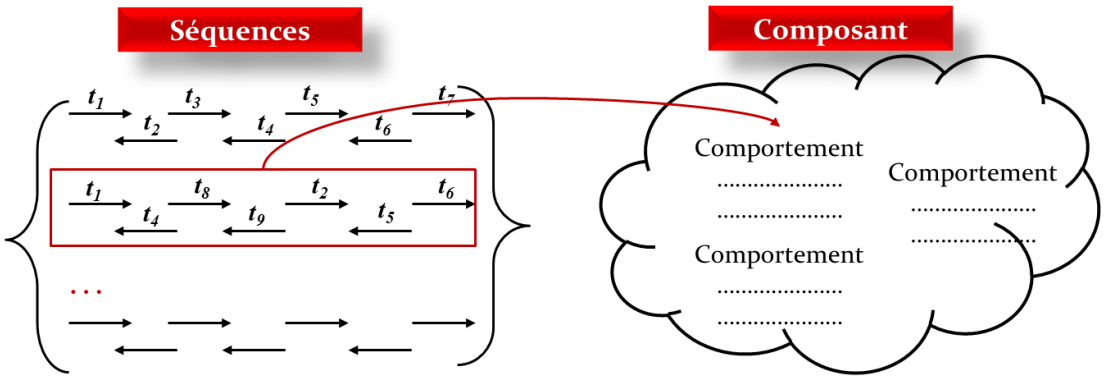
Les séquences de franchissement filtrées puis réduites nous permettent de construire l'ensemble des scénarios de test. En effet, nous pouvons considérer les scénarios comme l'ensemble de toutes les séquences de franchissement valides depuis un marquage initial ou bien comme l'ensemble de toutes les séquences de franchissement valides menant à un marquage final désiré ou bien comme toutes les séquences de franchissement valides menant à un marquage contraint. Dans le contexte ERTMS, les scénarios de test sont

produits à partir des séquences de franchissement et seront ensuite exécutés sur la plateforme de simulation afin de valider les composants ERTMS.

Un composant ERTMS a plusieurs comportements. Un scénario de test correspond à un comportement valide du composant. L'algorithme de Bennaser nous permet de générer l'ensemble de TOUTES les séquences de franchissement de longueur k. Cette méthode est complète et nous garantit la couverture de la spécification ERTMS pour une longueur donnée des séquences générées.

A partir de cet ensemble complet des séquences de franchissement, on produit les scénarios de test de la manière suivante :

- ✓ Soit la première séquence de franchissement, la première transition correspond à une action qu'on envoie au composant afin de voir sa réaction,
- ✓ La sortie du composant est ensuite vérifiée :
  - Si cette sortie correspond à la sortie prévue par la spécification, on continue à exécuter le scénario en envoyant la deuxième action au composant (prévue dans la séquence de franchissement),
  - Par contre si la sortie générée par le composant ne correspond pas à celle qui est prévue par la spécification dans la séquence de franchissement en question, nous cherchons dans le reste des séquences si une des séquences corresponde à cette sortie.
- ✓ Une séquence de franchissement qui correspond à toutes les sorties du composant est un scénario qui valide ce dernier.
- ✓ Dans le cas où une sortie a été produite par le composant et n'a pas été trouvée dans toutes les séquences de franchissement générées, un verdict fail est attribué et le composant n'est pas considéré comme valide.



✓ **Verdict vrai : tous les scénarios exécutés sont prévus par la spécification**

✓ **Verdict faux: un comportement contenant n transitions aux interfaces est considéré faux si nous garantissons la génération de toutes les séquences comportant au moins n transitions aux interfaces**

Figure 4.84 : Production des scénarios de test

Actuellement, nous n'avons pas développé un outil permettant d'interagir directement avec la plate-forme de simulation ERTMS. Cependant, nous pouvons exécuter un cas d'utilisation permettant de vérifier un comportement du composant. Suite à cette exécution, nous pouvons vérifier si toutes les interactions du composant avec l'environnement correspondent à l'une des séquences générées à partir de spécification formelle représentant ce comportement. Des exemples sont donnés en Annexe H afin de décrire quelques scénarios modélisés formellement et exécutés ensuite sur la plate-forme ERTMS.

4.6 Conclusion

Dans ce chapitre, nous avons développé une démarche permettant de générer de scénarios de test à partir de modèles formels en réseaux de Petri interprétés. Le modèle RdP définit le comportement global du système distribué. Il s'agit d'interactions entre les divers composants réactifs. Le scénario de test est un ensemble d'entrées correspondant à la spécification dont nous voulons récupérer les sorties à comparer avec les sorties du simulateur. D'un point de vue RdP, le scénario de test est considéré comme une **séquence de franchissement filtrée puis réduite** du réseau de Petri interprété représentant la spécification. Nous avons utilisé les techniques d'abstraction logique et la programmation par contraintes afin de générer l'ensemble des séquences de franchissement. En effet, nous ne construisons pas l'ensemble du graphe d'accessibilité du réseau de Petri à cause du problème d'explosion combinatoire. Nous recherchons plutôt des chemins dans ce graphe. Finalement, les séquences de franchissement des transitions aux interfaces nous permettent de construire nos scénarios de test afin de valider les composants à tester.

5 CONCLUSION GENERALE

Aujourd'hui, plus de vingt systèmes de signalisation ferroviaire et de contrôle de vitesse existent en Europe. Ces systèmes sont validés au niveau national où chaque pays possède son propre « langage » et ses propres exigences pour faire circuler les trains sur son réseau. Cette segmentation du réseau européen constitue une barrière au développement du trafic européen et complique la tâche des conducteurs au franchissement des frontières. Dans le but de supprimer ces contraintes techniques, la commission européenne a mis en œuvre des directives d'interopérabilité portant sur l'harmonisation du réseau ferroviaire européen et permettant la libre circulation des trains sans rupture de charge entre états membres. Parallèlement, dans les années 90, la commission européenne a sollicité la mise au point du système de signalisation européen ERTMS acronyme de « European Rail Traffic Management System ». Le déploiement du système ERTMS nécessite des campagnes de tests longues et coûteuses. Nos travaux de recherche ont pour objectif de diminuer les coûts de validation et de certification issus de la mise en œuvre de ce nouveau système en Europe tout en facilitant l'interopérabilité à travers la reconnaissance mutuelle de composants ERTMS entre les états membres.

Nos travaux de thèse s'inscrivent dans un processus de vérification de la conformité des constituants ERTMS par rapport à leur spécification. Ces constituants, dits réactifs, interagissent continuellement avec leur environnement. Ils sont sous la forme d'une boîte noire étant donné que leur comportement n'est visible qu'à travers les interactions qu'ils ont avec l'extérieur. Nous avons choisi la méthode de test et en particulier le test de conformité pour valider les constituants ERTMS. Le test de conformité est un test fonctionnel, permettant de vérifier la conformité d'un constituant dont le code est inconnu par rapport à une spécification formelle. En effet, la spécification constitue une référence dans le processus du test de conformité. C'est à partir de cette spécification que nous déterminons les verdicts de test. Il est essentiel qu'elle soit formelle afin que les comportements qu'elle décrit soient suffisamment précis et non ambigus pour que les verdicts de test aient un sens. Or, la spécification ERTMS, complexe et de grande taille, est définie de manière informelle et ne

favorise pas le développement de méthodes et outils de génération automatique de tests. Ainsi, une formalisation de la spécification s'impose.

Afin de résoudre ce premier verrou scientifique, nous avons développé une approche de modélisation combinant des modèles semi-formels et des modèles formels. Le langage UML en tant que langage de modélisation objet nous permet de décrire la spécification selon des points de vue structurel et comportemental. Ce langage est considéré comme un standard de modélisation objet et possède de nombreux avantages concernant la structuration et l'organisation des données d'un système. Cependant, dans un contexte de certification et de validation d'équipements ferroviaires, ce langage semi-formel ne peut pas être utilisé pour générer des scénarios de tests. Nous avons alors choisi de transformer les modèles UML en des modèles formels tout en préservant les propriétés structurelles et fonctionnelles du système à formaliser. Le langage formel choisi est le formalisme des réseaux de Petri (RdP) et plus particulièrement, les réseaux de Petri interprétés. Ce choix est motivé par un ensemble de facteurs. Tout d'abord les RdP sont des méthodes formelles. De plus, ils renseignent, dans un modèle unique, sur les deux aspects du système représenté: structurel (grâce à la structure même du modèle) et comportemental (grâce à l'évolution des jetons dans la structure et éventuellement l'évolution de leurs valeurs). Ensuite, en termes de puissance de modélisation, ce formalisme possède également une très grande expressivité (modélisation des communications, des contraintes temporelles et des lois de commande). Enfin, Ce langage formel possède des outils de vérification et convient parfaitement à notre contexte d'étude.

La technique de transformation que nous avons utilisée est basée sur l'approche MDA (Model Driver Architecture) et nous a permis de transformer un modèle source (conforme à son méta-modèle) en un modèle cible (conforme à son méta-modèle) suivant des règles de transformation. Nous avons développé des règles transformant des machines d'états UML en une sous-classe des RdP interprétés. Nous nous sommes intéressés en particulier à représenter les interactions entre les composants lors de la modélisation. Nous avons en effet distingué les transitions de communication que nous avons appelé transitions aux interfaces des transitions internes de chaque composant. A partir des modèles de la spécification en réseau de Petri interprétés, nous avons généré des scénarios de test dont l'exécution sur le simulateur ERTMS, permettra de vérifier si les réponses de ce dernier sont cohérentes avec la spécification.

Le modèle RdP interprété définit le comportement global du système distribué. Dans ce modèle, nous avons associé aux transitions des conditions et des opérations. Des variables sont utilisées afin de représenter les états des composants. Une transition est valide si la condition correspondante est valide. Le franchissement de cette transition exécute

l'opération correspondante et permet de changer l'état du système. Le scénario de test doit tester les interactions entre les composants. D'un point de vue RdP, nous avons considéré le scénario de test comme une séquence de franchissement filtrée, réduite et composée par les transitions aux interfaces. Générer les scénarios de test consiste alors à générer l'ensemble des séquences de franchissement du réseau de Petri. Nous avons utilisé la technique d'abstraction logique et la programmation par contraintes pour déterminer ces séquences de franchissement depuis un réseau de Petri simple. Ensuite, nous avons développé une méthode de filtrage consistant à inclure les caractéristiques du réseau de Petri interprété (conditions et opérations). Dans cette phase, nous avons éliminé les transitions qui ne sont pas valides en prenant en compte les conditions associées et nous avons obtenu un ensemble de séquences filtrées. Enfin, nous avons réduit l'ensemble filtré des séquences générées en éliminant toutes les transitions internes et ne laissant que les transitions aux interfaces. L'ensemble réduit des séquences générées est utilisé pour construire les scénarios de test. Ces scénarios sont ensuite exécutés en partie sur le simulateur afin de valider les composants ERTMS. En effet, dans le cas où le simulateur génère une réponse qui est incluse dans le scénario donné, alors la réponse est prévue dans la spécification et le verdict de test est « Pass ». Cependant, dans le cas où la réponse générée n'est pas incluse dans le scénario donné mais incluse dans un autre scénario alors le verdict de test est « Inconclusive ». Enfin, dans le cas où la réponse générée n'est prévue dans aucun scénario alors le verdict de test est « Fail ».

Perspectives

L'exposé de ce travail a permis d'entrevoir des voies de recherche encore inexplorées, tant au niveau théorique que pratique. Nous avons choisi de privilégier deux perspectives qui semblent retenir notre attention.

Introduction du temps dans la formalisation de la spécification

Lors de la formalisation de la spécification ERTMS, nous avons choisi de transformer des modèles UML sous la forme de machines d'états en réseau de Petri interprétés. ERTMS est un système de signalisation et de contrôle-commande nécessitant des temps de réponse bien précis lors de l'envoi et de la réception de messages entre composants. En effet, le langage UML possède deux types de données particuliers relatifs au temps : le type « ***Time*** » qui définit une valeur représentant un mouvement absolu ou relatif dans le temps et le type « ***TimeExpression*** » qui permet de spécifier des expressions dont l'évaluation donne une valeur de type Time. Dans le cas de spécifications temporelles modélisées par des machines d'états, UML définit un événement spécifique appelé « ***TimeEvent*** ».

Lors de la transformation de modèles UML vers des réseaux de Petri, la prise en compte du temps devrait être intégrée (réseaux de Petri temporisés).

Amélioration de la méthode de génération de scénarios de test

Nous avons développé une approche de génération de séquences de franchissement sur trois étapes. Il est possible de réduire ces trois étapes pour générer directement des séquences filtrées. Il est envisageable en effet de mettre en œuvre une technique plus directe dans laquelle on retarde l'énumération des séquences de franchissement en rajoutant de nouvelles contraintes dans l'algorithme de génération de séquences de franchissement. Ces contraintes concernent les propriétés spécifiques des réseaux de Petri interprétés. Ainsi, au lieu de générer des séquences de franchissement à partir d'un réseau de Petri simple puis développer une méthode de filtrage des séquences, on peut générer directement des séquences de franchissement filtrées.

Nous avons développé notre méthodologie de génération de scénarios de test en utilisant le langage Prolog. Ce prototypage n'est pas suffisant dans le cadre d'une démarche de certification et de validation de composants ERTMS. La mise en œuvre d'outils plus performants qui permettront d'exécuter directement les scénarios de tests générés sur le simulateur ERTMS devrait être efficace.

Références

AGEDIS Project http://www.agedis.de/

Amyot, D., Logrippo, L. & Buhr, R.J.A. (1997). "Spécification et conception de systèmes communicants : une approche rigoureuse basée sur des scénarios d'usage". Leduc (Ed.), CFIP 97, Ingénierie des protocoles, Liège, Belgique, Septembre 1997. Hermès, 159-174.

Araki, T. & Kasami, T. (1977). "Some decision problems related to the reachability problem for petri nets". *Theoretical Computer Science*, vol. 3, pp. 85–104.

Audibert, L. (2009). UML 2: de l'apprentissage à la pratique. Collection Info+, Editeur Ellipses.

Basu, A., Morrisett, G. and Eicken, T. (1998). Promela++ : A language for Constructing Correct and Efficient Protocols.

Baudoin, C. and Hollowell, G. (1996). Realizing the Object-Oriented Lifecycle. Upper Saddle River, NJ: Prentice Hall.

Beizer, B. (1990). Software Testing Techniques. New York: Van Nostrand Reinhold.

Beizer, B. (1995). Black-Box Testing. John Wiley and Sons.

Bérard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A. and Petrucci, L., Schnoebelen, P. and McKenzie, P. (2001). Systems and software verification. Springer Verlag.

Berieau, M. (2004). Demande d'autorisation de mise en service (ou exploitation) du sous-système ERTMS. *Revue Générale des chemins de fer*, pages 115-118.

Bernardi, S., Donatelli, S. and Merseguer, J. (2002). From UML Sequence Diagrams and Statecharts to analysable Petri Net models. In Proc. of the 3rd Int. Workshop on Software and Performance (WOSP), pages 35–45, Rome, Italy.

Bernot, G., Gaudel, M.-C. and Marre, B. (1991). Software testing based on formal specifications: a theory and a tool. *Softw. Eng.* J., 6(6) :387– 405.

Berthelot, G. (1986) "Transformations and decompositions of nets". Dans: Advances in Petri Nets 1986 Part I, Proceedings of an Advanced Course, éd. par Brauer, W., Reisig, W. et Rozenberg, G. pp. 359–376. – Springer-Verlag. NewsletterInfo : 27.

Bézivin, J. (2005). On the unification power of models. Software and System Modeling (SoSym), 4(2):171–188, 2005.

Bézivin, J. and Gerbé, O. (2001). Towards a Precise Definition of the OMG/MDA Framework. In Proceedings of the 16th IEEE international conference on Automated Software Engineering (ASE), page 273, San Diego, USA.

Blanc, X. (2005). MDA en Action : Ingénierie logicielle guidée par les modèles. Edition Eyrolles. ISBN 2-212-11539-3.

Bochmann, G.V. and Petrenko, A. (1994). Protocol testing : Review of methods and relevance for software testing. *In Thomas Ostrand, editor, Proceedings of the 1994 International Symposium on Software Testing and Analysis (ISSTA)*, pages 109–124, 1994.

Boehm, B.W. (1970). Some Information Processing Implications of Air Force Space Missions: 1970-1980, The Rand Corporation, RM-6213-PR.

Bolognesi, T. and Brinksma, E. (1988). Introduction to the ISO Specification Language LOTOS. *ISDN, 14(1) :25–29*.

Bontron, P., Maury,O., du Bousquet, L., Ledru,Y., Oriat, C., et Potet, M.-L. (2001). TOBIAS : un environnement pour la création d'objectifs de tests à partir de schémas de tests. *In J.C. Rault, editor, In International Conference on Software and Systems Engineering and their Applications (ICSSEA)*, Paris, France

Bontron, P. (2005). Les schémas de test : une abstraction pour la génération de tests de conformité et pour la mesure de la couverture. Thèse de Doctorat de l'Université Joseph Fourier.

Booch, G. (1991). Object Oriented Design with Applications. Redwood City, CA: Benjamen / Cummings

Booch, G., Rumbaugh, J. et Jacobson, I. (2000). Le guide de l'utilisateur UML (version française). Eyrolles.

Bouali, M. (2009). Contributions à l'analyse formelle et au diagnostic à partir de réseaux de Petri colorés avec l'accessibilité arrière. Thèse de Doctorat de l'université de Technologie de Compiègne

Bourdeaud'huy, T., Hanafi, S. and Yim, P. (2004). Efficient Reachability Analysis Of Bounded Petri Nets using Constraint Programming. International Conference on Systems, Man and Cybernetics, pp. 10–13.

Brinksma, E. (1988). A theory for the derivation of tests. *In Aggarwal, S. and Sabnani, K., editors, Protocol Specification, Testing, and Verification VIII, pages 63_74. Elsevier Science Publishers B.V.* North-Holland.

Bruel, J.-M. (1998). Transforming UML Models to Formal Specifications. OOPSLA'98, Workshop on Formalizing UML. Why? How?.

Bryant, R.E. (1986). "Graph based algorithms for boolean function manipulation". *IEEE transactions on computers*, vol. C-35, no 8, août 1986, pp. 677–691.

Budinsky, F., Steinberg, D. et Ellersick, R. (2003). Eclipse Modeling Framework : A Developer's Guide. Addison-Wesley Professional.

Cactus Project. Projet Cactus: http://jakarta.apache.org/cactus/

Carlier, M. (2009). Test automatique de propriétés dans un atelier de développement de logiciels sûrs. Thèse de Doctorat du Conservatoire National des Arts et Métiers. Ecole Doctorale d'Informatique, Télécommunication et Electronique.

Castan, P. (2004). Principes du contrôle commande en mode ERTMS, Aspects techniques, organisationnels et humains. *Revue Générale des chemins de fer*, pages 25-32.

Chapas, P. (2007). Traction ferroviaire, équipements d'exploitation et de sécurité. *Techniques de l'ingénieur*.

Chapurlat, V. (2007). Vérification et validation de modèles de systèmes complexes: application à la Modélisation d'Entreprise. Habilitation à Diriger les Recherches. Université de Montpellier II.

Chow, T.S. (1978). Testing software design modeled by finite-state machines. *IEEE Trans. Software Eng*, 4(3) :178–187.

Clarke, E., Grumberg, O. and Peled, D. (1999). Model Checking. MIT Press.

Clarke, D., Jéron, T., Rusu, V. and Zinovieva, E. (2004). STG : A symbolic Test Generation Tool. *Lecture Notes in Computer Science*, 2280 :470–482.

Coad, P. and Yourdon, E. (1991). Object-Oriented Analysis. Englewood Cliffs, New Jersey: Yourdon Press, Prentice Hall, 2d edition.

Coq Development Team (2002). The Coq Proof Assistant Reference Manual Version 7, INRIA Rocquencourt.

Combemale, B. (2008). Approche de métamodélisation pour la simulation et la vérification de modèle. Thèse de Doctorat en Informatique de l'Université de Toulouse, Institut National polytechnique de Toulouse.

COTE Project. Projet COTE: http ://www.irisa.fr/cote/.

Czarnecki, K. et Helsen, S. (2003). Classification of Model Transformation Approaches. In OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture.

David, R. et Alla H. (1992). Du grafcet aux réseaux de Petri, Paris, Editions Hermès, 1992.

Delatour, J. (2003). Contribution à la spécification des systèmes temps réel : l'approche UML/PNO. Thèse de Doctorat de l'université Paul Sabatier de Toulouse.

DG TREN (2006). ERTMS pour un trafic ferroviaire fluide et sur. DG TREN, Bruxelles, 16papges.

Dijkstra, E.W. (1970). Notes on Structured Programming. T.H.- Report 70-WSK-03.

Directive 96/48/EC. Council Directive 96/48/EC of 23 July 1996 on the interoperability of the trans-European high-speed rail system

de Nicola, R. and Hennessy, M. (1984). Testing equivalences for processes. *Theoretical Computer Science*.

De Vries, R. and Tretmans, J. (2000). On-the-Fly Conformance Testing Using. *International Journal on Software Tools for Technology Transfer (STTT)*, Springer-Verlag Heidelberg, Volume 2(4): pages 382 – 393

El Koursi, E.M. and Kampmann, B. (2002). Qualitative and quantitative safety assessment of ERTMS Operating rules. *Comprail*, pages 671-680.

ERTMS website, www.ertms.com, date de consultation: Janvier 2010

Evrot, D., Lamy, P. et Petin, J-F. (2006). Un comparatif d'outils formels. *Actes du congrès Lambda Mu* 15, Lille.

Evrot, D. (2008). Contribution à la vérification d'exigences de sécurité: application au domaine de la machine industrielle. Thèse de doctorat, Université Henri Poincaré, NancyI.

Favre, J.-M., Estublier, J. et Blay, M. (2006). L'Ingénierie Dirigée par les Modèles : au-delà du MDA. Informatique et Systèmes d'Information. Hermes Science, lavoisier édition, février 2006.

Fernandez J.L et AmbrosioToval, J. (2001). Seamless formalizing the UML semantics through metamodels. in Unified Modeling Language: System Analysis, Desing and Development Issues, U. o. N.-L. Dr. Keng Siau, Dr Terry Halpin,, Ed.: Microsoft Corporation, Idea Group,

Fernandez, J-C., Jard, C., Jeron, T. and Cesar Viho. (1996). Using On-The-Fly Verification Techniques for the Generation of Test Suites. *In Computer Aided Verification*, p 348–359.

Finkel, A. (1993). "The minimal coverability graph for petri nets". Advances in Petri nets 1993, lecture notes in computer science, vol. 674, pp. 210–243.

Fleury, E. (2001). Automates temporisés avec mises à jour. Thèse de doctorat, Ecole Normale supérieure de Cachan.

Fontan, B. (2008). Méthodologie de conception de systèmes temps réel et distribués en contexte UML/SysML. Thèse de l'Université de Toulouse III - Paul Sabatier, France.

Gamatié, A.E.H. (2004). Modélisation polychrone et évaluation de systèmes temps réel. Thèse de l'Université de Rennes 1, France.

Gerber, A., Lawley, M. Raymond, K. Steel, J. and Wood, A. (2002). Transformation : The Missing Link of MDA. In A. CORRADINI, H. EHRIG, H. KREOWSKI et G. ROZENBERG, éditeurs : Proceedings of the First International Conference on Graph Transformation (ICGT), volume 2505 de Lecture Notes in Computer Science, pages 90–105, Barcelona, Spain.

Gill ,A. (1962). Introduction to the Theory of Finite State Machine. New-York McGraw-Hill.

Godary, K. (2004). Validation temporelle de réseaux embarqués critiques et fiables pour l'automobile. Thèse de doctorat, Institut National des Sciences Appliquées de Lyon.

Godefroid, P. (1996). "Partial-order methods for the verification of concurrent systems: an approach to the state-explosion problem". – New York, NY, USA, Springer-Verlag Inc., volume 1032, 142p.

Grabowski, J. (1994). Test Case Generation and Test Case Specification with Message Sequence Charts. PhD thesis, Université of Berne, Institue for Informatics and Applies Mathematics.

Grabowski, J., Hogrefe, D., Toggweiler, D. (1995). Partial Order Simulation of SDL Specifications. *In O. Braek and A. Sarma (eds), SDL'95 with MSC in CASE*, Noth-Holland.

Grabowski, J., Hogrefe, D., Toggweiler, D. and Scheurer, R. (1996). Dealing with the Complexity of State Space Exploration Algorithms. *In Proceedings of the 6th. GI/ITG technical meeting on 'Formal Description Techniques for Distributed Systems' University of Erlangen*.

Groz, R., Jeron, T. and Kerbrat, A. (1999). Automated Test Generation from SDL specifications. *In R. Dssouli, G. von Bochmann, and Y. Lahav, editors, SDL'99 The Next Millenium, 9th SDL Forum, Montréal, Québec*, pages 135–152, Elsevier.

Halbwachs, N. (1998). Synchronous programming of reactive systems, a tutorial and commented bibliography. *In Tenth International Conference on Computer-Aided Verification, CAV'98, Vancouver (B.C.)*, LNCS 1427, Springer Verlag.

Harel, D. and Pnueli, A. (1985). On the development of reactive systems. *In Apt, K., editor, Logics and Models of Concurrent Systems, volume F-13 of NATO ASI Series*, pages 477-498, New York. Springer-Verlag.

Herbreteau, F. (2001). Automates à file réactifs embarqués, Application à la vérification de systèmes temps-réel. Thèse de doctorat, Ecole Centrale de Nantes.

Huber, P., Jensen, A. M., Jepsen, L. O. & Jensen, K. (1985). "Towards reachability trees for high-level petri nets". *Lecture Notes in Computer Science : Advances in Petri Nets 1984*, vol. 188, 1985, pp. 215–233. – NewsletterInfo : 27.

Hutzler, G. (2000) Du Jardin des Hasards aux jardins de Données : une approche artistique et multi-agent des interfaces homme / systèmes complexes, Thèse de doctorat, Université Paris 6, janvier 2000

IEEE (1990). IEEE Standard Glossary of Software Engineering Terminology. Technical Report IEEE Std 610.12-1990, Institute of Electrical and Electronic Engineers, New York.

ISO/IEC. (1994) - Information Technology - Open Systems Interconnection – Conformance Testing Methodology and Framework. International Standard ISO/IEC 9646.

ISO 8402 (1994). Quality management and quality assurance: Vocabulary, Second edition 1994-04-01, International Standard Organization.

ITU (1992). Message Sequence Chart (MSC), ITU-T Recommendation Z.120, International Telecommunication Standards Sector SG 10, Geneva.

ITU (1999). Recommendation Z.100. Specification and Description Language (SDL). *Technical Report Z-100, International Telecommunication Union – Standardization Sector*, Genève.

Jabri, S. and Lemaire, E. (2007a). Modeling of the European railway system for automatic chekings. *International Symposium EURNEX - ZEL 2007*, Université de Zelina, Zelina, Slovaquie.

Jabri, S., El Koursi, E.-M., Lemaire, E., Yim, P. & Bourdeaud'huy, T. (2007b). Contribution à l'étude de la couverture des scénarios de tests de systèmes critiques. *Workshop international "Logistique & Transport" LT' 2007*, a technically IEEE/SMC co-sponsored workshop, 18-20 novembre 2007, Sousse, Tunisie, IEEE

Jabri, S., El Koursi, E.-M., Lemaire, E., Yim, P. & Bourdeaud'huy, T. (2007c). Contribution à l'étude de la couverture des scénarios de tests de systèmes critiques – Application au système de signalisation ferroviaire ERTMS/ETCS. *Journée des doctorants 2007,* INRETS.

Jabri, S., El Koursi, E.-M., Lemaire, E., Yim, P. & Bourdeaud'huy, T. (2008a). Contribution à l'étude de la couverture des scénarios de tests de systèmes critiques – Application au système de signalisation ferroviaire ERTMS/ETCS. *Journée des doctorants 2008*, INRETS.

Jabri, S. El Koursi, E.-M. & Lemaire, E. (2008b). Deployment method of tests scenarios for critical systems- Application to the ERTMS/ETCS system. *8th World Congress on Railway Research*, Seoul, Corée du Sud.

Jabri,S. El Koursi, E.-M. Lemaire, E. & Bourdeaud'huy, T. (2008c). Modelling of the ERTMS specifications for checking and tests generation objectives". Symposium on Formal Methods for Automation and Safety in Railway and Automotive Systems (FORMS/FORMAT 2008), Budapest.

Jabri,S. El Koursi, E.-M. Lemaire, E. & Bourdeaud'huy, T. (2009a). A generation method of test scenarios based on models- Application to the ERTMS/ETCS system. *Third International Conference on Safety and Security Engineering (SAFE 2009)*, Rome.

Jabri,S. El Koursi, E.-M. Lemaire, E. & Bourdeaud'huy, T (2009b). Modelling of the ERTMS specifications for checking objectives. *12th IFAC Symposium on Transportation Systems (CTS'09)*, Californie, USA, Septembre 2009.

Jabri, S., El Koursi, Bourdeaud'huy, T. & E.-M. Lemaire, E. (2010). European Railway Traffic Management System validation using UML/Petri nets modelling strategy. The European Transport Research Review (ETRR). Volume 2, Numéro 2, pp 113-128.

Jacobson, J. (1987). Object Oriented development in an industrial environment. Proceedings on Object-Oriented programming systems, languages and applications, pp 183-191.

Jard, C. and Jéron, T. (2002). TGV : theory, principles and algorithms. *In The Sixth World Conference on Integrated Design & Process Technology (IDPT'02)*, Pasadena, California, USA.

Jensen, K. (1992). Coloured Petri Nets. Basic concepts, analysis method and practical use, vol. 1, EATC monographs on Theoretical Computer Science, Springer Verlag.

Jéron, T. (2004). Contribution à la génération automatique de tests pour les systèmes réactifs. Habilitation à Diriger des Recherches, Université de Rennes I, France.

Jéron, T. and Morel, P. (1999). Test Generation Derived from Model-Checking. *In Computer Aided Verification'99, Trento, Italy, N. Halbwachs, D. Peled (eds.)*, volume 1633 of Springer-Verlag, LNCS, pages 108–122.

Jézéquel, J-M., Ho, W.-M., Le Guennec, A. and Pennaneac'h, F. (1999). UMLAUT: an extendible UML transformation framework. *In Robert J. Hall and Ernst Tyugu, editors, Proc. of the 14th IEEE International Conference on Automated Software Engineering, ASE'99*, Florida.

Jouault, F. (2006). Contribution à l'étude des langages de transformation de modèles. Thèse de doctorat, Université de Nantes.

Jourdan, M. (1994). Etude d'un environnement de programmation et de vérification de systèmes réactifs, mutli langages et multi outils. Thèse de doctorat, Université Joseph Fourier-Grenoble I.

Kansomkeat, S., Offutt, J., Abdurazik, A. and Baldini, A. (2008). A Comparative Evaluation of Tests Generated from Different UML Diagrams. *In Ninth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD 2008)*, pages 867–872, Phuket Thailand.

Karp, R.M. et Miller, R.E. (1969). "Parallel program schemata". Journal of Computer and Systems Sciences, vol. 3, no 2, pp. 147–195.

Kim, T., Stringer-Calvert, D. And Cha, S. (2005). Formal verification of functional properties of a SCR-style software requirements specification using PVS. *Reliability Engineering and System Safety, vol 87, p351-363*.

King, P. and Pooley, R. (1999). Using UML to derive stochastic Petri net models", In Proceedings of the 15th UK Performance Engineering Workshop, pages 45–56, Bristol, UK, July.

Lacôte, F. and Poré, J. (2004). La signalisation ferroviaire européenne ERTMS/ETCS devient une réalité. *Revue Générale des chemins de fer*, pages 39-47.

Lancien, D. (2004). La genèse du projet ERTMS. *Revue Générale des chemins de fer*, pages 15-23.

Lautenbach, K. (1987). "Linear algebraic techniques for place/transition nets". Dans: Advances in Petri Nets 1986, Part I, Proceedings of an Advanced Course, éd. par Brauer, W., Reisig, W. et Rozenberg, G. pp. 142–167. – Springer-Verlag. NewsletterInfo : 27.

Le Guennec, A. (2001). Génie Logiciel et Méthodes Formelles avec UML Spécification, Validation et Génération de tests. Thèse de doctorat, Université de Rennes.

Lee, D. and Yannakakis, M. (1994). Testing Finite State Machines : State Identification and Verification. *IEEE Trans Computer*, 43(3) :306–320, 1994.

Le Moigne, J-L. (1990). La modélisation des systèmes complexes, Paris, Bordas, Dunot, 1990

Leroux, E.Z. (2004). Méthodes symboliques pour la génération de tests de systèmes réactifs comportant des données. Thèse de Doctorat de l'Université de Rennes 1.

Lévêque, O. (2007). European Railway Traffic Management System. 2nd International Seminar on Safety and Security of Railway Operations, Rabat.

Lesens, D. (1997). Vérification et synthèse de systèmes réactifs. Thèse de doctorat, Insitut National Polytechnique de Grenoble.

Loiseaux, C. (1994). Vérification symbolique de programmes réactifs à l'aide d'abstractions. Grenoble, Thèse de PhD, Université Joseph Fourier.

Lopez-Garo, J.P., Merseguer, J. and Campos, J. (2004). From UML activity diagrams to stochastic Petri Net models: Application to Software Performance Engineering. Proceedings of the 3rd Int. workshop on Software and performance, pp. 25-36.

Marre, B. and Arnould, A. (2000). Test Sequences Generation From Lustre Descriptions: GATeL. *ASE'00, Fifteen IEEE Int. Conf. on Automated Software Engineering, pages 229–237, IEEE Computer Society Press*.

Marre, B. and Arnould, A. (2004). Test Selection Strategies for Lustre Descriptions in GATeL. MBT Workshop, ETAPS'04 Satellite Event, 2004.

Marsan, M.A., Balbo, G., Conte, G., Donatelli, S. and Franceschinis, G. (1995). Modelling with Generalized Stochastic Petri Nets. Series in parallel computing. John Wiley and Sons.

Melham, T.F. and Gordon, M.J.C. (1993). Introduction to HOL: A theorem proving Environment for higher-order logic. *Cambridge University Press*.

Merseguer, J. (2004). On the use of UML State Machines for Software Performance Evaluation. In Proc. of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS).

Minsky, M. (1968). Matter, mind, and models. Semantic Information Processing, pages 425–432.

Moalla, M(1981). Spécification et conception sure d'automatismes discrets complexes, basés sur l'utilisation du grafcet et des réseau de Petri. Thèse de Doctorat de l'université Scientifique et Médicale de Grenoble et l'Institut National de Polytechniques de Grenoble.

Moore, E.F. (1956). Gedanken-experiments on sequential machines. *Automata Studies*, pages 129–153.

Morel, P. (2000). Une algorithmique efficace pour la génération automatique de tests de conformité. PhD thesis, UFR IFSIC/ laboratoire IRISA.

Morin, E. (1990). La Méthode (tome3) : La Connaissance de la Connaissance, Le Seuil. Edition de poche, collection "Points", 1990

Murata, T. (1989). Petri nets: Properties, analysis and applications. Proceedings of the IEEE, Vol. 77, N°4, pp. 541-574.

Musa, J. (1975). A theory of software reliability and its application. *IEEE Transactions on Software Engineering*, SE-1(3):312_327.

Myers, G. (1979). The Art of Software Testing. John Wiley & Sons.

Nahm, R. (1994). Confornace Testing Based on Formal Description Techniques and Message Sequence Charts. PhD thesis, Université of Berne, Institue for Informatics and Applies Mathematics.

Naito, S. and Tsunoyama, M. (1981). Fault Detection for Sequential Machines by Transition Tours. *In Proceedings of the 11th IEEE Fault Tolerant Computing symposium*, pages 238–243.

Offutt, J. and Abdurazik, A. (1999). Generating tests from UML specifications. *In R. France and B. Rumpe, editors, UML'99 - The Unified Modeling Language. Beyond the Standard. Second International Conference*, volume 1723, pages 416–429, Fort Collins, CO, USA.

OMEGA Project http://www.cs.umd.edu/projects/omega/

OMG (Object Management Group). (2002). Xml metadata interchange (XMI) specification. Technical report.

OMG (Object Management Group). (2005). Unified Modelling Language: superstructure. Version 2.0, (2005), http://www.omg.org.

OMG (Object Management Group). (2005a). UML Profile for Schedulability, Performance, and Time (SPT) v1.1. www.omg.org/docs/formal/05-01-02.pdf.

OMG (Object Management Group). (2006). Meta Object Facility (MOF) 2.0 Core Specification.

OMG (Object Management Group). (2007). Unified Modeling Language: 2.1.2 Infrastructure.

OMG, (Object Management Group). (2007a). UML Profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE). www.omg.org/docs/ptc/07-08-04.pdf.

OMG (Object Management Group). (2008). Meta Object Facility (MOF) 2.0 Query/ View/Transformation (QVT) Specification, version 1.0, avril 2008.

Open Systems Interconnection, OSI (1993). OSI Conformance Testing Methodology and Framework, International Standards Organization. Open Systems Interconnection - OSI Conformance Testing Methodology and Framework, part 3 : TTCN Extensions,ISO/IEC JTC 1 DAM-1.

Ortmeier, F., Schellhorn, G., Thums, A., Reif, W., Hering, B. and Trppschuh, H. (2003). Safety analysis of the height control system for the Elbtunnel. *Reliability Engineering and System Safety, N°81, p259-268.*

Owre, S., Rajan, S., Rushby, J., Shankar, N. and Srivas, M.K. (1996). PVS: combining specification, proof checking and model checking. *Computer Aided Verification 1996, LNCS 1102, pp 411-414. Springer-Verlag.*

Ozello, P. (2004). Certification du système ERTMS. *Revue Générale des chemins de fer*, pages 109-114.

Paulson, L. C. (1994). Isabelle: A Generic Theorem Prover. *Lecture Notes in Computer Science, vol. 828. Springer-Verlag, New York, NY.*

Pareaud, T. (2009). Adaptation en ligne de mécanismes de tolérance aux fautes par une approche à composants ouverts. Thèse de doctorat, Université de Toulouse.

ParTeG (Partition Test Generator), Weiβleder, S. http://parteg.sourceforge.net.

Petrenko, A. (2001). Fault Model-Driven Test Derivation from Finite State Models: Annotated Bibliography. *In F Cassez, C Jard, B Rozoy, M D Ryan (Eds) Modeling and Verification of Parallel Processes, volume 2067 of LNCS*, pages 196–205.

Petrenko, A., Yevtushenko, N. and Huo, J.L. (2003). Testing Transition Systems with Input and Output Testers. *In Proceedings of the IFIP TC6/WG6.1 XV International Conference on Testing of Communicating Systems (TestCom 2003)*, Sophia Antipolis, France

Phalippou, M. (1994). Test Sequence Generation using Estelle or SDL Structure Information. *In : Proceedings of FORTE/PSTV (Berne ,Switzerland), éd. par et S. Leue (D. H.),* pp. 405–420.

Pickin, S., Jard, C., Heuillard, T., Jézéquel, J-M. and Desfray, P. (2001). A UML-integrated test description language for component testing. *In Andy Evans, Robert France, Ana Moreira, and Bernhard Rumpe, editors, Practical UML-Based Rigorous Development Methods-Countering or Integrating the eXtremists. Workshop of the pUML-Group held together with the UML 2001*, volume P-7 of LNI, pages 208–223, Toronto, Canada.

Pickin, S., Jard, C., Jéron, T., Jézéquel, J.M. and Traon, Y.L. (2007). Test synthesis from UML models of distributed software. *In IEEE Transactions on Software Engineering, volume 33, pages 252–268.*

Pnueli, A. (1986). Applications of temporal logic to the specification and verification of reactive systems: a survey of current trends. *In Current Trends in Concurrency. Overviews and Tutorials, pages 510_584, New York. Springer-Verlag.*

Pretschner, A. and Lotzbeyer, H. (2001). Model based testing with constraint logic programming : First results and challenges. *In Proc. 2nd ICSE Intl. Workshop on Automated Program Analysis, Testing and Verification (WAPATV'01)*, Toronto.

Prigent, A. (2003). Le test des systèmes temps-réel paramétrés : application à la conception d'architectures avioniques. Thèse de Doctorat, Ecole Centrale de Nantes et Université de Nantes.

Rapps, S. and Weyuker, E. J. (1985). Selecting software test data using data flow information. *IEEE Transactions on Software Engineering*, SE-11(4):pp. 367_375.

Reggio, G., Cerioli, M. and Astesiano, E. (2001). Towards a Rigorous Semantics of UML Supporting its Multiview Approach. In Proc. FASE 2001. Lecture Notes in Computer Science, vol. 2029, Berlin, Springer Verlag.

Roussel, J.M. and Denis, D. (2002). Safety properties verification of ladder diagram programs. Européen des Systèmes Automatisés, 36(7), pp. 905-917.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. (1997). OMT, Tome1 : Modélisation et conception orientées Objet", Dunod.

Rushby, J. (2002). Using model checking to help discover mode confusions and other automation surprises. Reliability Engineering and System Safety, N°75, p167-177.

Samuel, P., Mall, R. and Kanth, P. (2007). Automatic test case generation from UML communication diagrams. Information and Software Technology, Vol. 49, pp. 158-171.

Seidewitz, Ed. (2003). What models mean. IEEE Software, 20(5):26–32.

Scorletti, G. & Binet, G. (2006) "Réseaux de Petri", Document de cours à l'Université de Caen.

Shlaer, S. and Mellor, S. (1988). Object-Oriented Systems Analysis. Yourdon Press Computing Series.

Son, S. H. and Seong, P. H. (2003). Development of a safety critical software requirements verification method with combined CPN and PVS - *A nuclear power plant protection system application. Reliability Engineering and System Safety*, N°80, p19-32.

STI 2006/860/EC (2006). EN Commission Decision of 7 November 2006 concerning a technical specification for interoperability relating to the control-command and signaling subsystem of the trans-European high speed rail system.

Tamarit, J. and Guido, P. (2004). ERTMS : concrétisation, harmonisation de la certification et déploiement en Europe. *Revue Générale des chemins de fer*; pages 91-107.

Tardieu, S. (2001). Cours de systèmes répartis. http://www.rfc1149.net/documents/.

Tretmans, J. (1992). A Formal Approach to Conformance Testing. PhD thesis, University of Twente, the Netherlands.

Tretmans, J. (2002). Testing techniques. Lecture Notes.

Tretmans, J. and Belinfante, A. (1999). Automatic testing with formal methods. *In Proceedings of the 7th European International Conference on Software Testing Analysis and Review EuroSTAR'99*, Barcelona, Spain.

Tretmans, J. and E. Brinksma, E. (2002). Côte de Resyste Automated Model Based.

Trowitzsch, J. and Zimmermann, A. (2006). Using UML state machines and Petri nets for the quantitative investigation of ETCS. Proceedings of the 1st Int. conference on Performance evaluation methodologies and tools, Italy.

UIC (2007). Functional System Requirements Specification (FRS), version 5.

UNISIG, ERTMS Users Group (2008). Subset026, System Requirements Specification (SRS), version 3.0.0.

Valette, R. (1999). Qu'est-ce qu'un bon modèle ?, Notes de cours, LAAS, Toulouse, http://www.laas.fr/~robert.

Valmari, Antti. (1991). "Stubborn sets for reduced state space generation". Lecture Notes in Computer Science; Advances in Petri Nets, vol. 483, pp. 491–515. – NewsletterInfo: 33, 39.

Vernadat, F. (2001). Contribution à la modélisation et à la vérification des systèmes communicants. Habilitation à Diriger les Recherches. Université Pau Sabatier, Toulouse.

Vernadat, F., Azéma, P. & Michel P. (1996). "Covering steps graphs. Dans : *17 th Int. Conf on Application and Theory of Petri Nets 96,* éd. par Springer-Verlag. – Osaka - Japan.

Weiβleder,S. and Sokenou, D. (2008). Automatic Test Case Generation from UML Models and OCL Expressions. *In Testmethoden f ̈ur Software - Von der Forschung in die Praxis (TESO'08), pages 423–426*, Munchen.

Wendler, E. (2009). Influence of ETCS on the capacity of lines. *In Compendium on ERTMS, edited by UIC*.

Winter, P. (2009a). Train control-command: the ETCS developments. *In Compendium on ERTMS, edited by UIC*.

Winter, P. (2009b). European ERTMS applications in commercial operation. *In Compendium on ERTMS, edited by UIC*.

Winter, P. (2009c). Conclusions and outlook. *In Compendium on ERTMS, edited by UIC*.

Yim, P. (2000). "Réseaux de Petri, logique et théorie des ensembles: contributions à l'étude des systèmes dynamiques discrets". Habilitation à diriger les recherches, Université de Lille I.

Yourdon, E. and Constantine, L. (1979). Structured Design", Englewood Cliffs, NJ: Prentice Hall.

Annexe - A : Modes d'exploitation ERTMS

Les modes d'exploitation d'ERTMS sont divers et constituent l'ensemble des modes opératoires et des procédures nécessaires pour assurer, en sécurité, les échanges d'informations entre le conducteur et le sous-système "embarqué". Chaque mode est associé à une configuration particulière (train, voie et conditions) dans laquelle le train pourrait se trouver. La liste suivante énumère tous les modes possibles et fournit une définition simple de chaque mode [UNISIG, 2008] :

Full Supervision **(FS)**: est le mode nominal qui permet la circulation normale des trains à la vitesse maximale de la ligne. Toutes les données relatives au train et à la voie sont disponibles à bord. Elles permettent à l'équipement de bord de calculer et d'afficher la vitesse maximale autorisée, la vitesse et la distance à respecter et naturellement la vitesse de circulation. Ces données permettent également de superviser les déplacements du train c'est-à-dire d'élaborer les différentes courbes d'avertissement, d'alerte et de contrôle conduisant à la prise en charge du train par le système si un risque de non-respect du point but visé est détecté (freinage d'urgence).

On Sight **(OS)**: Dans ce mode, si toutes les informations relatives à l'itinéraire sont bien connues du système, celui-ci ne peut obtenir l'assurance que le canton aval est libre (soit parce qu'il est occupé, soit parce qu'un dérangement de circuit de voie est en cours, soit parce qu'aucune information fiable ne lui est parvenue). Le système ne peut pas alors délivrer une autorisation de mouvement en mode FS. Cette incertitude sur l'occupation de voie en aval se traduit par un passage en mode OS. Ce mode impose au conducteur, en plus des autres contraintes, une vitesse maximale paramétrable nationalement.

Staff Responsible **(SR)**: C'est un mode qui correspond à l'état le plus dégradé du système. Celui-ci ne possède aucune information ni sur l'occupation de la voie en aval, ni sur les itinéraires. Ainsi, aucune autorisation de mouvement ne peut être délivrée. Dans ce cas, le conducteur pourra déplacer le train dans une zone équipée ERTMS/ETCS, sous sa propre responsabilité. Comme pour le mode OS, une vitesse maximale paramétrable nationalement, devra être respectée par le conducteur.

Shunting **(SH)**: C'est le mode manœuvre qui permet au train de se déplacer sans que les données du train (telles que la position, la vitesse limite…) ne soient parvenues au train.

No Power **(NP)** : Ce mode intervient lorsque l'équipement embarqué ERTMS/ETCS n'est pas actionné. Le freinage d'urgence est donc commandé.

Stand-By **(SB)** : Constitue un mode par défaut qui intervient lors de la mise en route du système ERTMS/ETCS ou lorsque la cabine de signalisation n'est pas en service. Il correspond à un mode d'attente.

Sleeping **(SL)**: le train est contrôlé à distance par la locomotrice «maitresse».

Unfitted **(UN):** Permet à un train équipé du système ERTMS de traverser une zone non-équipée ERTMS.

Non leading **(NL):** Le train est couplé à un autre train (pas d'ATP). Le conducteur est responsable de la conduite. Intervient lorsque le matériel embarqué et le conducteur ne sont pas dans la cabine principale.

Isolation **(IS)** : le conducteur a isolé le système embarqué.

Trip **(TR)** : Prévoit un déclenchement du freinage d'urgence. Il intervient, par exemple, lorsque le train a dépassé la distance qu'il avait à parcourir ou lorsqu'il a reçu l'information "DANGER".

Post Trip **(PT)** : Intervient après un arrêt d'urgence du train en mode Trip et dès que le conducteur a accusé réception du mode Trip.

System Failure **(SF)** : Intervient lorsqu'une panne détectée pourrait influer sur la sécurité. Le freinage d'urgence est commandé.

STM European **(SE)** : Permet l'utilisation d'un système national de signalisation tout en appliquant les fonctions d'ERTMS/ETCS. Il est utilisé uniquement au niveau STM.

STM National **(SN)** : Permet l'utilisation d'un système national et applique les règles nationales.

Reversing **(RV)** : Autorise le train à rouler en marche arrière sur une distance limitée, tout en restant dans la même cabine. Ce mode permet au train d'échapper à des situations dangereuses.

Limited Supervision **(LS)** : Le mode Limited Supervision permet au train de fonctionner dans des zones où les informations au sol peuvent être fournies afin de réaliser la supervision du train. Le conducteur ne peut pas sélectionner ce mode mais doit absolument entrer dans ce mode s'il reçoit les commandes et que toutes les conditions nécessaires sont accomplies.

- **Les transitions entre modes ERTMS**

Le passage d'un mode à l'autre s'appelle une transition, mais il n'existe pas forcément de transitions entre tous les modes, c'est-à-dire que certains modes ne peuvent être consécutifs à d'autres. Les transitions entre les modes nécessitent l'établissement de différentes conditions obligatoires à respecter pour que la transition se fasse correctement [UNISIG, 2008]. Ces conditions sont présentées sous forme de liste.

Le tableau ci-dessous représente les différentes transitions. La lecture de ce tableau se fait très simplement. Différents symboles apparaissent dans le tableau :

- Le symbole "4>" signifie que la condition n°4 doit être remplie pour déclencher la transition.

- Le sens du symbole >, à savoir > ou <, est primordiale c'est-à-dire qu'il faut respecter le sens de la flèche dans la lecture du tableau. Par exemple, 1ère ligne/2ème colonne : la transition du mode SB au mode NP se fera si la condition n°29 est remplie.
- Chaque transition reçoit un ordre de priorité, p… pour éviter un conflit entre différentes transitions qui auraient lieu en même temps. Certaines transitions ont reçu le même ordre de priorité car il est évident qu'elles ne peuvent avoir lieu en même temps.

Table: Les transitions entre les modes ERTMS

NP	<29 -p2-	<29 -p2-	<29 -p2-	<29 -p2-	<29 -p2-	<29 -p2-	<29 -p2-	<29 -p2-	<29 -p2-	<29 -p2-	<29 -p2-	<29 -p2-	<29 -p2-		<29 -p2-	<29 -p2-
4> -p2-	SB		<19, 27 -p5-	<28 -p5-	<28 -p5-	<28, -p5-	<28, -p5-	<2, 3 -p4-	<28, 47 -p3-	<28, -p6-		<28, -p4			<28 -p6-	<28 -p4-
		PS	<26 -p5-													
	5, 6, 50> -p7-	2> -p4	SH	<5,6, 50,51 -p6-	<5,6, 50,51 -p6-	<5,6, 51 -p6-	<5,6 50,51 -p6-			<5,61 -p7-	<68 -p4	<5,6, 50 -p5-			<5,61 -p7	
	10> -p7-			FS	<31,32 -p6-	<31,32 -p6-	<31,32 -p6-			<25 -p7-		<31 -p5-			<25 -p7-	
	70> -p7-			70,72> -p6-	LS	<72 -p6-	<70,72 -p6-			<71 -p7-		<70 -p5-			<71 -p7-	
	8,37> -p7-			37> -p6-	37> -p6-	SR	<37 -p6-			<44,45 -p4-		<8,37 -p5-			<44,45 -p4-	
	15> -p7-			15,40> -p6-	15,40> -p6-	40> -p6-	OS			<34 -p7-		<15 -p5-			<34 -p7-	
	14> -p5-	14> -p4						SL								
	46> -p6-		46> -p5-	46> -p6-	46> -p6-	46> -p6-	46> -p6-		NL							
	60> -p7-			21> -p6-	21> -p6-	21> -p6-	21> -p6-			UN	<62 -p4-				<21 -p7-	
	20> -p4-		49,52, 65> -p4-	12,16, 17,18, 20,41, 65,66, 69> -p4-	12,16, 17,18, 20,41, 65,66, 69> -p4-	18,20, 42, 43, 36, 54,65> -p4-	12,16, 17,18, 20,41, 65,66, 69> -p4-			67,39, 20> -p5-	TR				<67, 39,38, 20 -p5-	
											7> -p4-	PT				
	13> -p3-	13> -p3-	13> -p3-	13> -p3-	13> -p3-	13> -p3-	13> -p3-			13> -p3-	13> -p3-	13> -p3-	SF		<13 -p3-	<13 -p3-
1> -p1-	1> -p1-	1> -p1-	1> -p1-	1> -p1-	1> -p1-	1> -p1-	1> -p1-	1> -p1-	1> -p1-	1> -p1-	1> -p1-	1> -p1-	1> -p1-	IS	<1 -p1-	<1 -p1-
	58> -p7-			56> -p6-	56> -p6-	56> -p6-	56> -p6-			56> -p7-	63> -p4-				SN	
				59> -p6-	59> -p6-		59> -p6-									RV

Pour les conditions à respecter, lorsqu'il est indiqué par exemple"<5,6,50,51", cela signifie en fait "<5 ou 6 ou 50 ou 51" c'est-à-dire que pour déclencher la transition, c'est la condition 5 ou 6 ou 50 ou 51 qui devra être remplie.

Annexe - B : Plateforme de simulation ERTMS INRETS

L'INRETS, avec le soutien de la région Nord-Pas de Calais et du FEDER, a acquis une plateforme de simulation du système ERTMS. Cette plateforme, conforme aux spécifications ERTMS, sert de support aux thématiques de l'équipe ESTAS.

La plate-forme est constituée de 3 systèmes indépendants :

- **Un simulateur de conduite** qui comporte un pupitre conforme aux spécifications CENELEC de l'interface de la cabine du conducteur. Il permet d'exécuter un scénario à un seul train sur une infrastructure donnée. Toutes les données du scénario sont enregistrées afin de permettre une analyse a posteriori.

- **Une restitution 3D** de l'environnement de conduite qui plonge le conducteur du simulateur de conduite dans un environnement simulé lui permettant d'appréhender tous les événements issus de l'infrastructure se présentant à lui.

- **Un simulateur de trafic** qui est constitué de plusieurs modules dont un contrôleur d'itinéraires, un système de gestion des enclenchements, jusqu'à 2 Radio Block Center, Jusqu'à 11 trains, dont le simulateur de conduite, gérés simultanément. Ce simulateur de trafic sert à la fois de poste central de commande du trafic ferroviaire et de gestionnaire des trains en manuel ou en automatique. Il permet en outre de plonger le conducteur du simulateur de conduite dans un trafic à plusieurs trains.

Figure: Simulateur de conduite et restitution 3D

- **Des outils hors ligne** sont nécessaires à la préparation des scénarios et à leur analyse.

- Un **éditeur de voie** permet de construire une infrastructure et de poser les différents éléments constitutifs du plan de voie : voie, aiguillages, balises, panneaux, signaux, profils…

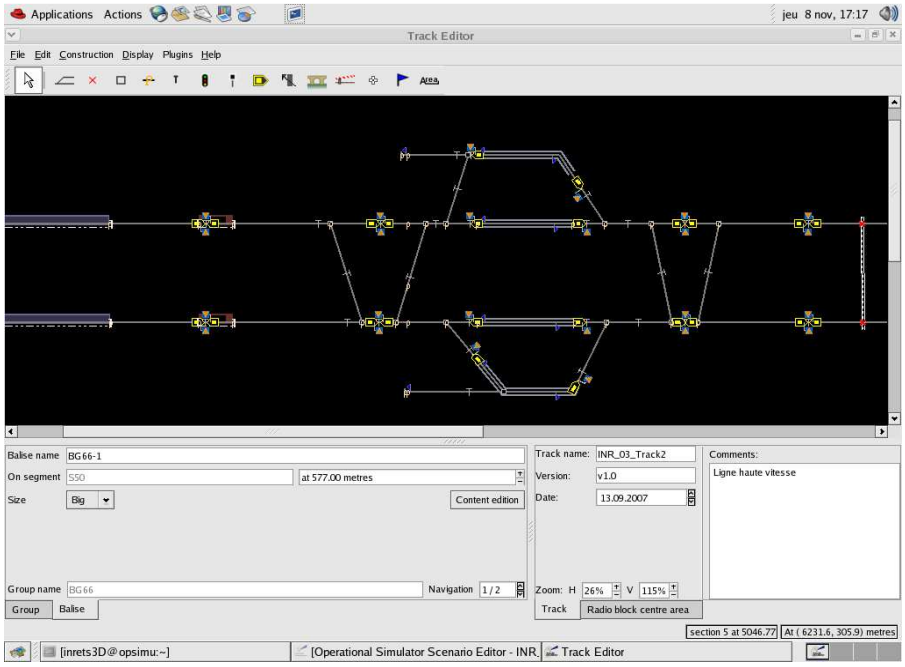


Figure: Editeur de voie

- Un **éditeur de scénario** pour paramétrer les aspects dynamique de la simulation

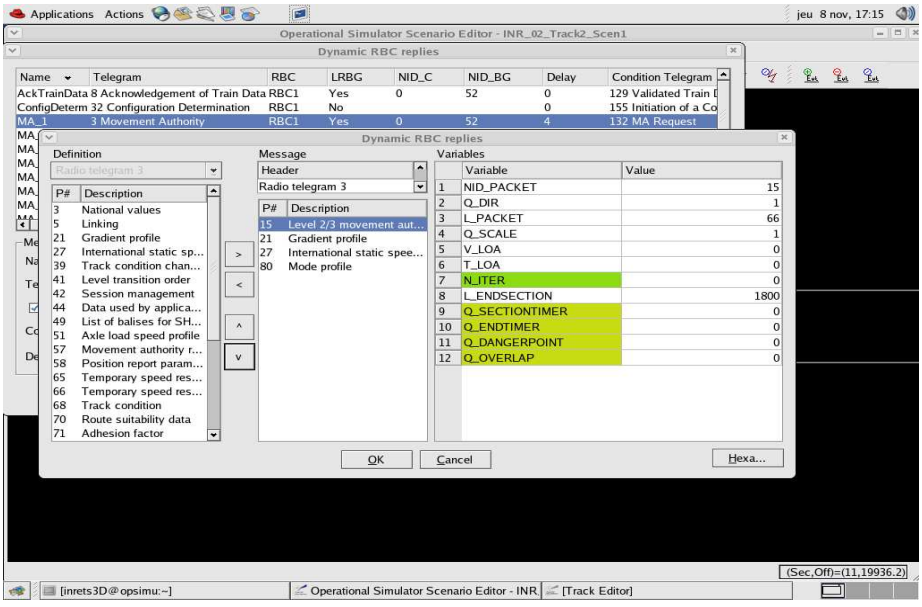


Figure: Editeur de scénario

Un analyseur de scénarios pour décrypter les résultats des simulations a posteriori.

Annexe - C : Systèmes AUTOFOCUS et GATEL

Les méthodes de générations symboliques et les méthodes de générations par contraintes sont deux méthodes de génération de scénarios de test de type boîte noire. Nous décrivons dans la suite deux exemples illustrant ces deux méthodes de test : Le système AUTOFOCUS et le système Gatel.

- ***Le système AUTOFOCUS*** [Pretschner & Lotzbeyer, 2001] : se basant sur l'exécution symbolique et la programmation par contraintes, ce système nécessite un modèle abstrait du programme à tester. Ce dernier est composé par des unités indépendantes communiquant avec un environnement extérieur à travers des ports de communications. Ces unités constituent les composants du programme. Leurs comportements sont décrits par une variante des machines de Mealy. L'ensemble des ports de communication, eux-mêmes des composants, constitue l'interface du programme. Des digrammes de transitions d'états sont ensuite utilisés pour décrire le programme. Ils contiennent des variables locales, des transitions, et des états de contrôle. Enfin, les contraintes à vérifier représentent un scénario de test dont l'exécution permet de tester ces contraintes sur le programme même.

- ***Le système GATEL*** ([Marre&Arnould, 2000], [Marre&Arnould, 2004]) : ce système représente une méthode et un outil d'assistance à la génération automatique de scénarios de tests à partir de descriptions LUSTRE (Langage formel pour les applications à flots de données synchrones) [Halbwachs, 1998]. Il se base sur la programmation logique par contraintes. La description du programme sous test en LUSTRE est fournie par l'utilisateur à l'outil. Elle contient un ou plusieurs nœuds, dont l'interface du plus englobant détermine le nombre et le type des entrées et sorties de test. Cette description est alors utilisée par GATEL pour produire les contraintes liant les sorties aux entrées. Pour chaque variable de sortie est construit un ensemble de scénarios de test en prenant en compte les différentes manières d'obtenir les valeurs possibles des variables. Finalement, générer les scénarios de test revient à résoudre les contraintes en utilisant les techniques de dépliage et de programmation logique par contraintes (symbolique).

Annexe - D : Outils de génération de tests de conformité

Nous décrivons dans cette annexe quelques outils de génération de tests de conformité.

1- Le projet AGEDIS

AGEDIS (Automated Generation and Execution of Test Suites for Distributed Component-based Software) [AGEDIS] est un projet européen qui a été réalisé entre 2000 et 2003. L'objectif de ce projet était de développer des méthodes et des outils afin d'automatiser les tests de logiciels répartis, en particulier dans le domaine des télécommunications.

Le projet AGEDIS est issu d'une collaboration entre IBM Haifa (Israël) et IBM Hursley (G.-B.), France Télécom R D (Lannion), IntraSoft (Grèce), Imbus (Allemagne), l'universitè d'Oxford, l'IRISA (Rennes) et Vérimag (Grenoble). Un profil UML [OMG, 2005] (Unified Modelling Language) est utilisé pour modéliser les systèmes à travers des diagrammes de classes, des diagrammes d'objets et des diagrammes d'´etats transitions. Dans le langage IF de Vérimag est compilée une description UML avec ses critères de sélection. Le langage UML est compilé en une API (*Application Programmable Interface*) de simulation lui offrant des fonctionnalités pour parcourir et observer un produit. Le profil UML permet aussi de décrire les critères de sélection des scénarios de test à travers des digrammes d'états transitions UML, regroupant ainsi les objectifs de test à la TGV et les critères de couverture d'expressions, en utilisant les diagrammes d'états UML.

Pour générer les scénarios de test, AGEDIS utilise un outil qui n'est qu'une extension de l'outil TGV. Cet outil de génération utilise l'API du simulateur IF et intègre des fonctionnalités de l'outil d'IBM Gotcha, en particulier les critères de couverture. Suite à leur génération, les scénarios de test sont ensuite traduits dans un format XML pour être exécutés en utilisant l'outil d'exécution de test Spider [AGEDIS].

2- L'outil STG

STG (Symbolic Test Generation) [Clarke et al, 2002] est un outil de génération de scénarios de test symboliques pour les systèmes réactifs. Une spécification et un objectif de test symboliques constituent les entrées de cet outil. Ces dernières sont modélisées par des IOSTS (In/Output Symbolic Transition System, extension du langage IF).

Dans la spécification sont décrits les comportements attendus. Quant à l'objectif de test, il permet de sélectionner un sous-graphe intéressant parmi les comportements possibles décrits dans la spécification. Le produit entre l'objectif de test et la spécification est ensuite calculé, analysé et simplifié. Un scénario de test est produit suite à cette analyse. Il est permet d'émettre un verdict au plus tôt et d'orienter l'exécution en fonction du comportement de

l'implantation, afin d'´éviter au maximum les verdicts inconclusifs. A l'exécution, les paramètres de l'implantation et du scénario de test doivent être fixés afin de les rendre exécutables. La résolution de contraintes lors de l'exécution dépend des choix des émissions du testeur définies en arithmétique de Presburger par le solveur du projet Omega [OMEGA].

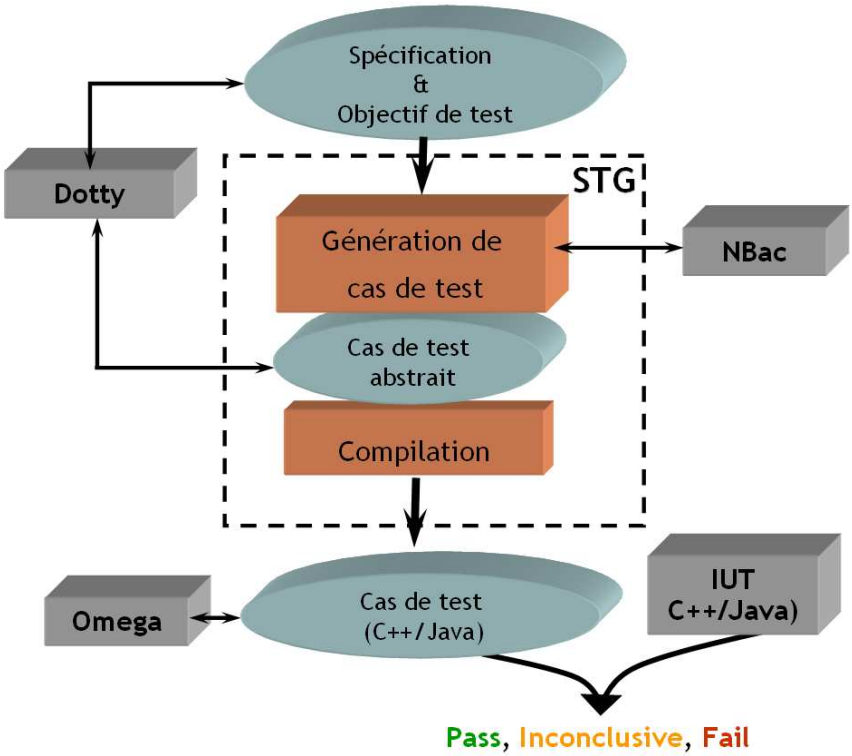


Figure: Outil STG, d'après [Leroux, 2004]

3- L'outil SAMSTAG

SAMSTAG {Sdl And Msc baSed Test cAse Generation} ([Nahm, 1994], [Grabowski, 1994]) est un outil issu d'un projet développé par l'Université de Berne entre 1991 et 1993. Ce dernier avait pour objectif le développement d'une méthodologie pour générer automatiquement des scénarios de test abstrait sous le format de TTCN (Tree and Tabular Combined Notation). La méthode se base sur des spécifications SDL [ITU, 1999] et des objectifs de test sous le format de diagramme MSC (Message Sequence Chart) [ITU, 1992]. Ces objectifs de test sont nécessaires pour générer les scénarios de test et analyser les résultats obtenus. La méthode SAMSTAG est conforme à la génération de séquence de test (ensemble de scénarios de test) de conformité donnée par la norme ISO 9646 [ISO, 1994]. De 1993 à 1995, cette méthode a été améliorée afin de tenir en compte les problèmes de l'explosion du graphe d'états [Grabowski et *al*, 1996]. Ainsi, une méthode de simulation d'ordre partiel pour les spécifications SDL a été développée et implémentée [Grabowski et *al*, 1995].

A partir d'une spécification SDL et d'un objectif de test sous la forme d'un SMC, la méthode de génération de test de l'outil SAMSTAG consiste à produire des scénarios de test sous le format de TTCN. La spécification SDL décrit tout le système à tester (SUT: System Under Test). Ce système contient le protocole à tester (IUT: Implementation Under Test) et tous les autres protocoles et paramètres dépendant de lui. La première étape de la génération des scénarios de test dans l'outil SAMSTAG consiste à simuler toute la spécification afin d'y chercher, à partir de l'état initial, les traces avec celles de l'objectif de test et qui s'exécutent avec un début et une fin. Suite à cette recherche de traces finies de l'objectif de test, une sélection des traces pertinentes est réalisée. Celles qui conduisent à un état final « Pass » sont considérées comme des traces candidates (PPO : Possible Pass Observable). Cette trace candidate ne contient que des actions observables qui sont définies en fonction du SUT. Suite à une sélection d'une trace pertinente PPO, une vérification est nécessaire afin de s'assurer que cette dernière produise bien un verdict « Pass ». La trace vérifiée est alors appelée UPO (Unique Pass Observable). Parfois, les systèmes à vérifier possèdent des comportements non déterministes, d'où la possibilité d'avoir un verdict « Inconclusif » ; ceci est en accord avec la norme ISO 9646. Afin de prendre en compte cette éventualité, SAMSTAG permet de générer des verdicts inconclusifs pour les UPO.

A la fin, le scénario de test est généré sous format TTCN à partir de l'UPO. Il décrit les comportements dynamiques attendus en combinaison avec toutes les actions observables. Ensuite, à partir du TTCN, les définitions des types et les déclarations des contraintes pour tous les messages envoyés et reçus par le SUT, sont générées. Les types sont définis en se basant sur les signaux de la spécification SDL et les valeurs des contraintes sont attribuées lors de l'exécution de l'UPO et des observations inconclusives de l'UPO. Le verdict « Fail » est produit dans tous les autres cas que ceux déterminés par le TTCN.

4- Le projet Cote

Le projet COTE (COmponent TEsting) [COTE] qui entre dans le cadre du test fonctionnel de conformité est issu d'une collaboration entre les entreprises France Télécom Recherche et Développement, Gemplus, Softeam et les laboratoires IRISA et LSR-IMAG. Ce projet a duré deux ans (octobre 2000 à octobre 2002) et avait pour objectif la création d'une plate-forme de synthèse de test en se basant sur la notation UML (Unified Modelling Language). Il s'intéresse aux aspects comportementaux des spécifications dans le but d'automatiser le processus de test.

Dans ce projet, une approche MDA (Model Driven Architecture) [Blanc-05] a été adoptée pour générer les scénarios de tests. Différents niveaux de granularité sont proposés pour décrire les tests, et ceci d'une façon indépendante d'une plate-forme cible [Bontron, 2005]. La plate-forme de synthèse de test proposée correspond à une chaîne de traitement des tests à partir d'une spécification décrite en UML. Dans la figure suivante, est présentée une

architecture de l'environnement COTE où l'outil Objecteering de Softeam constitue l'outil central. Ce dernier communique avec les autres outils en utilisant le format XMI [OMG, 2002] qui fournit une description XML d'un modèle UML.

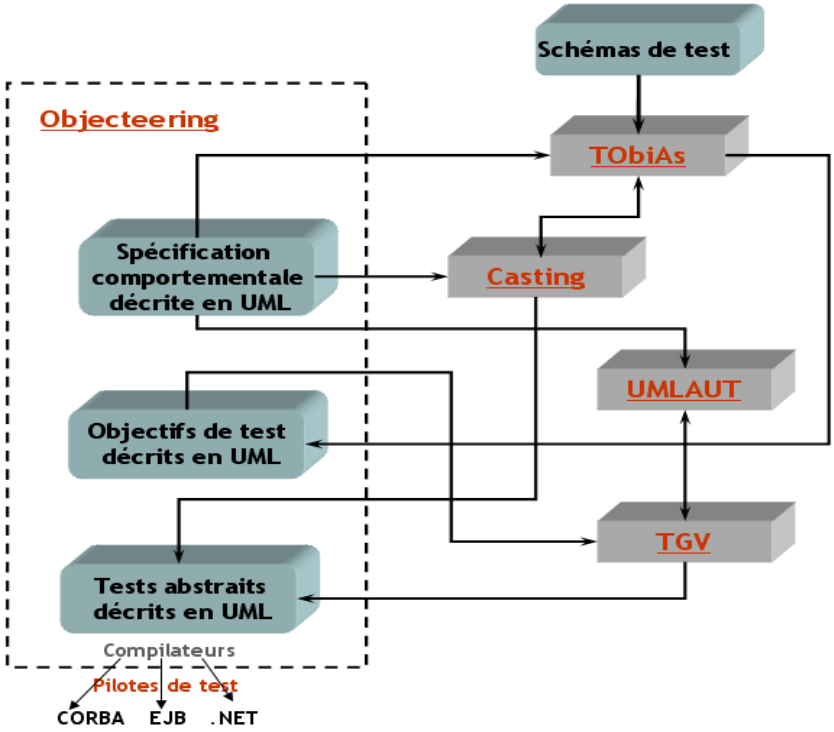


Figure: Architecture du projet COTE, d'après [Bontron, 2005]

Dans la suite, nous présentons en détail les différents outils constituant l'architecture du projet Cote.

Objecteering

Cet outil constitue l'environnement de modélisation UML. Il est utilisé dans le cadre de projet pour décrire la spécification en utilisant trois types de diagrammes UML : les digrammes de classes, les diagrammes d'états transitions et les diagrammes de séquence. Les diagrammes de classe et d'états transitions permettent de décrire la spécification. Une partie du langage UML a été adaptée à la description de tests sous la forme d'un profil UML. Ceci a permis la description des suites de tests avec des tests abstraits et des objectifs de test sous la forme de diagrammes de séquences [Pickin, 2001]. Cet outil de modélisation permet également de générer des tests exécutables sur différentes plates-formes.

Les pilotes de test

Les pilotes de test, tel que Cactus [Cactus] pour le test d'EJB (Entreprise Java Bean), sont utilisés pour l'exécution d'une séquence de tests concrets sur l'implémentation considérée. Ils permettent aussi de vérifier si les tests se déroulent sans erreur. Dans le cas où une erreur se produise, le pilote de test permet de déterminer l'origine de l'erreur.

Les compilateurs

Ce sont des composants intégrés dans l'outil de modélisation Objecteering permettant la transformation des tests abstraits en tests exécutables pour une cible donnée. Ces compilateurs ont été développés pour les langages Java et Corba, entre autres.

TGV

L'outil TGV (cf. section 2.4.2.2) est utilisé dans ce projet pour générer les tests abstraits à partir de la spécification et des objectifs de test. Ces derniers sont fournis soit manuellement soit via une interface d'Objecteering.

UMLAUT

Cet outil a été développé par l'IRISA dans l'objectif de pouvoir simuler une spécification UML [Jézéquel, 1999] à travers le calcul du graphe d'accessibilité de la spécification. Cette dernière doit être sous la forme d'un diagramme d'états-transitions étendu par du code Eiffel permettant ainsi l'animation de la spécification. Dans le cadre du projet COTE, l'outil UMLAUT est utilisé pour construire le produit de l'ensemble des diagrammes d'´etats-transitions afin de calculer la spécification comportementale du système [Le Guennec, 2001]. Il est couplé à l'outil TGV pour animer la spécification comportementale et fournir à cet outil les transitions possibles. Ceci permet à l'outil TGV de travailler sur la spécification comportementale à la volée.

TObiAs

L'outil TObiAs (Test OBjective desIgn ASsitant) ([Bontron, 2005], [Bontron, 2001]) est utilisé dans le cadre du projet COTE pour la construction des objectifs de test correspondants à des schémas de test. Il prend en entrée le diagramme de classes décrivant le système à tester. Ce diagramme est fourni sous la forme d'un fichier XMI généré par l'outil Objecteering. TObiAs communique avec l'outil Casting dans le but d'obtenir des jeux de valeurs pour les paramètres.

Casting

L'outil Casting [Bontron, 2005] est utilisé dans le cadre du projet COTE de deux manières :

- en le couplant à TObiAs pour qu'il lui fournisse des valeurs de tests ou valide des suites de test ;
- en synthétisant des tests abstraits à partir de la spécification UML, et en offrant une mesure de couverture de la spécification en termes de couverture des branches du diagramme d'états-transitions.

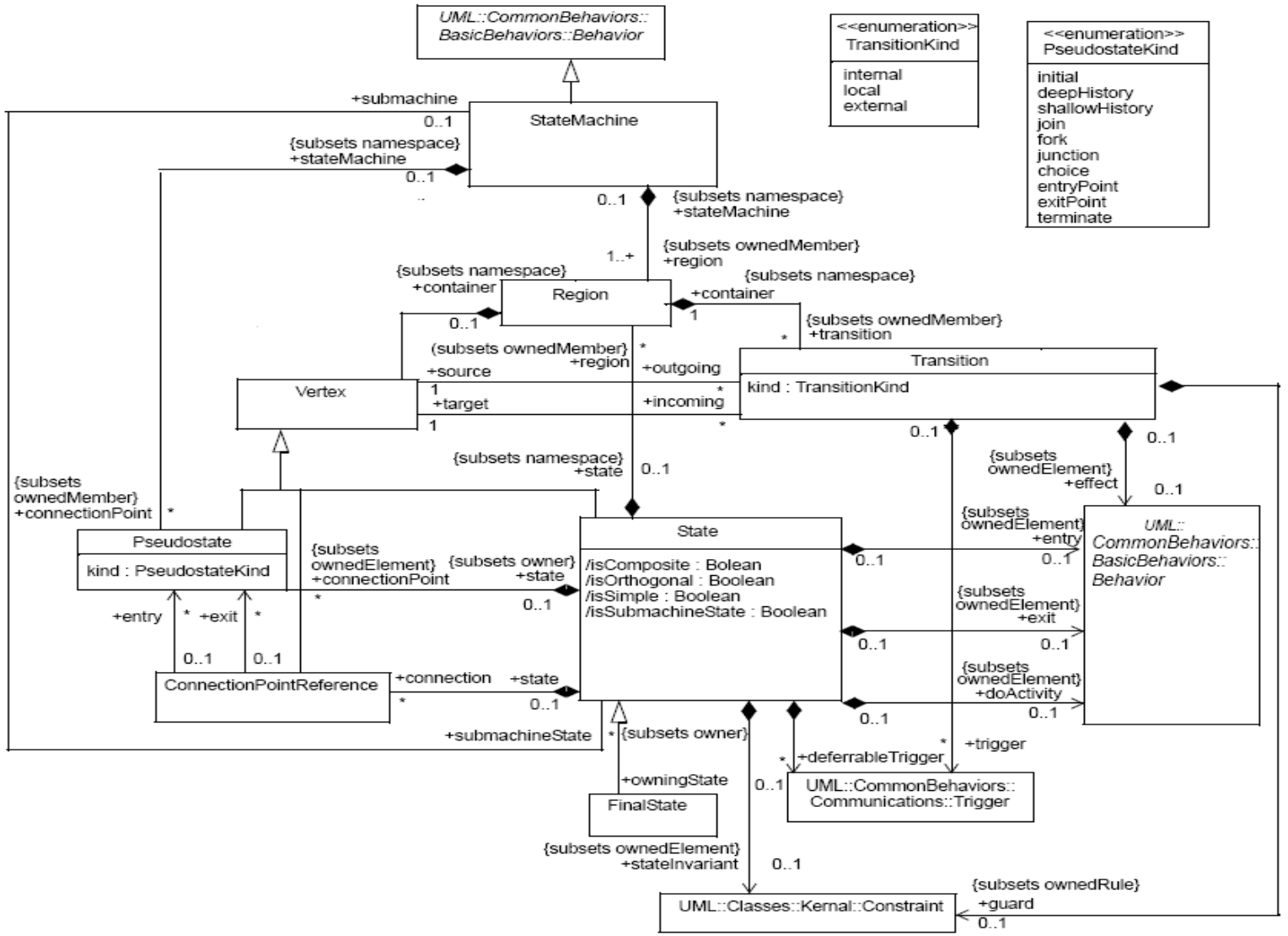
5- <u>L'outil ParTeG</u>

L'outil ParTeG (Partition Test Generator) [ParTeG] est utilisé pour tester un programme à partir d'un modèle UML. En effet, à partir d'une machine à états UML, d'un diagramme de classes UML et des spécifications OCL (Object Constraint Language), il est capable de

générer des jeux de test aux limites [Carlier, 2009]. La détermination de ces jeux de tests se fait par une analyse des chemins de la machine à états UML. Dans une première étape, les paramètres des appels de méthodes rencontrées ne sont pas déterminés et les valeurs restent abstraites. Ensuite, lorsqu'une garde est rencontrée, ParTeG analyse un à un les nœuds de la machine à états jusqu' à rencontrer une méthode dont la post-condition influe sur la garde [Weiβleder & Sokenou, 2008]. Cela permet de définir une corrélation entre les gardes et les post-conditions sous la forme de contraintes. L'attribution de valeurs aux paramètres permet de résoudre ces contraintes et d'obtenir ainsi des jeux de test concrets. Différents travaux ont porté sur l'utilisation de modèles UML pour générer des jeux de test ([Pickin et *al*, 2007], [Offutt & Abdurazik, 1999], [Bouquet et *al*, 2007]). Dans les travaux de [Kansomkeat et *al*, 2008] existe une étude comparative permettant de constater que les jeux de test obtenus à partir de diagrammes d'activité UML et de statecharts UML couvrent des classes d'erreurs différentes.

Annexe – E : Méta-modèle des machines d'états UML

Dans cette annexe, nous représentons le méta-modèle correspondant aux machines d'états UML 2.0 [OMG, 2007].



Annexe – F : Réseaux de Petri interprétés

Dans cette annexe, nous représentons les réseaux de Petri interprétés à travers une définition formelle, une description du fonctionnement et de la sémantique.

Définition Anx.B/ **RdP interprété** [David&Alla, 1992]. Une RdP interprété consiste en la donnée d'un réseau **R** < ***P, T, A, M*** >, d'un environnement < ***D, OP, PR*** > et de deux fonctions **σ** et **ψ** servant à établir un lien entre la partie commande et l'environnement, où :

- ✓ ***D*** est l'ensemble des états de l'environnement,
- ✓ ***OP*** = $\{op_1, op_2, \ldots, op_s\}$ est un ensemble d'opérateurs ; $op_i : D \rightarrow D$.
- ✓ ***PR*** = $\{pr_1, pr_2, \ldots, pr_s\}$ est un ensemble de prédicats sur D; $pr_i : D \rightarrow \{vrai, faux\}$.
- ✓ **σ** : P→OP associe à chaque place du réseau un opérateur.
- ✓ **ψ** : T→PR×OP associe à chaque transition du réseau un doublet prédicat-opérateur

Fonctionnement [David&Alla, 1992]. Un réseau interprété représente un système dont l'état est un doublet <**d,M**> où d ∈ D et M le marquage du réseau associé. L'état d de l'environnement peut changer par l'exécution d'une opération $\mathbf{op_i}$; l'application de op_i à d donne un état $\mathbf{d' = op_i\ (d)}$. Les conditions de franchissement des transitions sont modifiées pour pouvoir tenir compte de l'état de l'environnement. Comme pour les réseaux temporisés, les marques du réseau associé à un réseau interprété ont deux états possibles : disponible et indisponible. Le passage de l'état disponible à l'état indisponible s'effectue au moment du franchissement des transitions. Dès qu'un marquage M est atteint, toutes les opérations associées par σ à des places qui viennent de recevoir une marque (indisponible) sont activées. Les marques passent de l'état indisponible à l'état disponible dès que l'opération commandée est terminée. Une transition **t** est franchissable si ses places d'entrée contiennent un nombre suffisant de marques disponibles et le prédicat du doublet **ψ(t)** est vérifié par l'état présent de l'environnement.

Le franchissement d'une transition franchissable **t** consiste à appliquer la règle usuelle des réseaux d'une part, et d'autre part à transformer l'état de l'environnement par l'opérateur du doublet **ψ(t)**, c'est-à-dire si **<d, M>** est l'état d'un réseau interprété pour lequel **t** est franchissable alors après le franchissement de **t**, son état est **<d', M'>** avec **M** [t⟩**M'** et $\mathbf{d' = op_j\ (d)}$ où $\mathbf{\psi(t) = < pr_i , op_j >}$.

Pour simplifier la définition précédente, aucune hypothèse n'a été faite concernant la représentation de l'ensemble D et la fonctionnalité des opérateurs. Ainsi, il n'est pas possible d'exprimer, dans le modèle présenté, le résultat de l'action simultanée de deux opérateurs op_i et op_j sur l'état de l'environnement d.

Généralement on complète le modèle proposé en supposant que $D=D_1\times D_2\times\ldots D_k$. C'est-à-dire que l'environnement contient un ensemble de k objets dont les D_i représentent les états (si par exemple les objets sont les variables, alors D_i représente le domaine de définition de la i-ème variable). Aussi, avec chaque opérateur op_i est donné un ensemble d'indices L_i, $Li \subseteq \{1,2\ldots, k\}$ et tel que $op_i : X_{w\in Li}\ D_w \rightarrow X_{w\in Li}\ D_w$. C'est-à-dire, lors de son activation, l'opérateur op_i ne transforme que des objets correspondant aux indices de L_i. Deux opérateurs op_i et op_i sont dits compatibles si et seulement si $L_i\cap L_j=\varnothing$. Evidemment, lorsque deux opérateurs op_i et op_j sont compatibles, ils transforment des ensembles d'objets disjoints et on peut les activer en parallèle. En effet, si $d=<d_1, d_2, \ldots, d_k>$ est l'état de l'environnement à un instant donné et deux opérateurs compatibles op_i et op_j sont activés en parallèle à partir de d, alors l'état $d'=<d'_1, d'_2, \ldots, d'_k>$ atteint après leur activation est tel que $d'_s = d_s$ si $s \notin L_i\cup L_j$ et d's égal à la composante de même indice de op_i ($<d_w>_{w\in Li}$) si $s \in L_i$ ou de opj ($<d_w>_{w\in Lj}$) si $s\in L_j$.

Par contre si opi et opj ne sont pas compatibles, on ne peut pas en principe savoir quel est l'état atteint par l'environnement lorsqu'ils sont appliqués en parallèle. Cette relation de compatibilité entre opérateurs permet de déterminer des ensembles d'opérateurs activables en parallèle : ce sont des ensembles d'opérateurs deux à deux compatibles étant donné que la relation de comptabilité n'est pas transitive.

Annexe – G : Code de transformation du modèle UML vers le modèle RdP interprétés

Le modèle UML sous Prolog :

```
/* DIAGRAMMES UML ***************************************************************************************************************************************/

% Diag de sequence
messages([ouverture,ack,fermeture]).
ds(1, 1, p1, p2, ouverture).
ds(1, 2, p2, p1, ack).
ds(1, 3, p1, p2, fermeture).
descriptionEtat(p1_init,[]).
descriptionEtat(p1_inactif,[]).
descriptionEtat(p1_attente_ouv,[]).
descriptionEtat(p1_actif,[]).
descriptionEtat(p2_init,[]).
descriptionEtat(p2_inactif,[]).
descriptionEtat(p2_affectationA1,[changerParite,[parite,1]]).
descriptionEtat(p2_affectationA0,[changerParite,[parite,0]]).
descriptionEtat(p2_attente,[]).
descriptionEtat(p2_actif,[]).

% etats associes aux objets
etats(p1,[p1_init,p1_inactif, p1_attente_ouv, p1_actif]).
etats(p2,[p2_init, p2_inactif, p2_affectationA1, p2_affectationA0, p2_attente, p2_actif]).
% etats initiaux pour chaque objet
etat_initial(p1,p1_init).
etat_initial(p2,p2_init).
% transitions : NET,NAC, Etat_initial, Etat_Final,[PredicatDeclencheur, Arguments], [ListeConditions], [PredicatAction, Arguments]
% Arguments est une liste de valeurs
% ListeConditions est une liste de couples [nomVar,Valeur]
transitionDET(1,0,p1_init,p1_inactif,[],[],[]).
transitionDET(1,1,p1_inactif,p1_attente_ouv,[],[],[actionEnvoyerMessage,[ouverture,p1,p2,[]]]).

% choix : NET,NAC, Etta_Initial,[PredicatDeclencheur,Arguments], [Etat_Final,[ListeConditions]]*
transitionDET(1,3,p1_actif,p1_inactif,[],[],[actionEnvoyerMessage,[fermeture,p1,p2,[]]]).
transitionDET(2,0,p2_init,p2_inactif,[],[],[]).
transitionDET(2,1,p2_inactif,p2_affectationA1,[],[[parite,0]],[]).
transitionDET(2,2,p2_inactif,p2_affectationA0,[],[[parite,1]],[]).
transitionDET(2,3,p2_affectationA1, p2_attente,[],[],[]).
transitionDET(2,4,p2_affectationA0, p2_attente,[],[],[]).
transitionDET(2,5,p2_attente,p2_actif,[declencheurRecevoirMessage,[ouverture,p1,p2]],[[parite,1]],[actionEnvoyerMessage,[ack,p2,p1,[[ack,oui]]]]).
transitionDET(2,6,p2_attente,p2_inactif,[declencheurRecevoirMessage,[ouverture,p1,p2]],[[parite,0]],[actionEnvoyerMessage,[ack,p2,p1,[[ack,non]]]]).
transitionDET(2,7,p2_actif,p2_inactif,[declencheurRecevoirMessage,[fermeture,p1,p2]],[],[]).
% chaque etat destination est conditionné par une liste de couples [[nomVar,Valeur]]
choix(1,2,p1_attente_ouv,[declencheurRecevoirMessage,[ack,p2,p1]],
    [
    [p1_inactif,[[ack,non]]],   %NET,NAC,MESSAGE,ExpressionCondition
    [p1_actif,[[ack,oui]]]
    ]
```

Le code du transformateur Prolog :

```
/*
Notes sur le développement du transformateur
La requete générique de création de sous-réseau s'appelle 'mkMethode'
Elle permet de traiter :
* les actions dans l'état (traitementInterne)
* les actions dans les transitions, eg l'action actionEnvoyerMessage
* les déclencheurs , eg declencheurRecevoirMessage


*/


/*************************************************************************************************************************
    mkMethode(traitementInterne, [NomMethode,Affectation])

    NB : on avait prévu de pouvoir spécifier une liste d'actions

    Création d'une transition qui porte le nom de la méthode
    place en amont : nomMethode_appel
    place en aval : nomMethode_retour
    affectation portée par la transition == LActions

    test du predicat :

    mkMethode(traiterAction,[changerParite,[parite,1]]).

    Ecriture de :
    transition([transitionPN,[changerParite]],[ ],[parite,1],[[placePN,[changerParite_appel]]],[[placePN,[changerParite_retour]]]).

*************************************************************************************************************************/

mkMethode(traitementInterne, [NomMethode,Affectation]):-
    mkConcat([NomMethode,'_appel'],StringNom1),
    mkNomPlacePN([StringNom1],PlacePN1),
    mkConcat([NomMethode,'_retour'],StringNom2),
    mkNomPlacePN([StringNom2],PlacePN2),
    mkNomTransitionPNSimple([NomMethode],NomTransition),
    mkFormatTransition(NomTransition,[PlacePN1],[PlacePN2],[],Affectation, TransitionPN),
    /*pas d'effet de bord ici : on relie directement la transition representant la methode à la place 'retour'*/
    mywrite(TransitionPN),mynl.
```

Ici construction du sous-RdP correspondant à l'effet d'une transition UML

```
/*************************************************************************************************************
    mkMethode(actionEnvoyerMessage, [Type,Emetteur,Destinataire,LValeurs],PlaceDebut,PlaceTemoin):-

    test du predicat :

    M = [actionEnvoyerMessage,[ack,p2,p1,[[ack,oui]]]], M = [_,M1], mkMethode(actionEnvoyerMessage,1,2,M1,PDeb,PTem).

    Production de :
    PDeb = '[placePN,[EMappel_ack_p2_p1_1_2]]'
    PTem = '[placePN,[EMenvoye_ack_p2_p1_1_2]]'

    Ecriture de deux transitions :

    transition([transitionPN,[1,2,EM_ack_p2_p1]],[ ],[ ],
    [[placePN,[EMappel_ack_p2_p1_1_2]]],
    [[placePN,[EMenvoye_ack_p2_p1_1_2]],[placePN,[EMarrive_ack_p2_p1_1_2]]]).

    transition([transitionPN,[1,2,RM_ack_p2_p1]],[ ],
    [[ack,oui]],
    [[placePN,[EMarrive_ack_p2_p1_1_2]]],[[placePN,[RMrecu_ack_p2_p1]]]).

    Création d'une transition sans condition et sans affectation
    une place en amont : [placePN,[EMappel_ack_p2_p1,[[ack,oui]]]]
    deux places en aval : l'une sera relié au destinataire du message, l'autre pourra servir pour se synchroniser sur la réception
    C'est nécessaire si il faut attendre que l'action soit terminée pour passer dans l'état UML suivant, en fonction de la sémantique UML utilisée
    (est-ce qu'on arrive dans l'état suivant après la fin de l'action ou pas ?)

    Pour resoudre le probleme des messages generiques qui portent des valeurs,
    on propose que le réseau créé dépende de NET et NAC
    et crée deux transitions :

    EM_appel_TypeMessage_Src_Dest_Net_Nac -> t1  -> EM_arrive_(même nom) -> t2 (Gard: affectation de la valeur portée) -> RM_recu_(nom réduit)
                                                 -> Em_envoye (témoin)

*************************************************************************************************************/
```

```
mkMethode(actionEnvoyerMessage, NET, NAC, [Type,Emetteur,Destinataire,LValeurs],PlaceDebut,PlaceTemoin):-
    mkConcat(['em_appel_',Type,'_',Emetteur,'_',Destinataire,'_',NET,'_',NAC],StringNom1),
    mkNomPlacePN([StringNom1],PlaceDebut),

    %place temoin
    mkConcat(['em_envoye_',Type,'_',Emetteur,'_',Destinataire,'_',NET,'_',NAC],StringNom2),
    mkNomPlacePN([StringNom2],PlaceTemoin),

    %place de transfert
    mkConcat(['em_arrive_',Type,'_',Emetteur,'_',Destinataire,'_',NET,'_',NAC],StringNom3),
    mkNomPlacePN([StringNom3],PlaceMessage),

    %nom transition1
    mkConcat(['em_',Type,'_',Emetteur,'_',Destinataire],StringNom4),
    mkNomTransitionPNSimple([NET,NAC,StringNom4],NomTransition1),
    mkFormatTransition(NomTransition1,[PlaceDebut],[PlaceTemoin,PlaceMessage],[],[], TransitionPN1),

    %ecriture t1
    mywrite(TransitionPN1),mynl,

    %place pour reception
    mkConcat(['rm_recu_',Type,'_',Emetteur,'_',Destinataire],StringNom5),
    mkNomPlacePN([StringNom5],PlaceReception),

    %Les affectations necessaires éventuellement se trouvent deja au bon format dans LValeurs

    %nom transition2
    mkConcat(['rm_',Type,'_',Emetteur,'_',Destinataire],StringNom6),
    mkNomTransitionPNSimple([NET,NAC,StringNom6],NomTransition2),
    mkFormatTransition(NomTransition2,[PlaceMessage],[PlaceReception],[],LValeurs, TransitionPN2),

    %ecriture t2
    mywrite(TransitionPN2),mynl.
```

Ici construction du sous-RdP correspondant à un déclencheur d'une transition UML

```
/*************************************************************************************************************
    mkMethode(declencheurRecevoirMessage,[Type,Emetteur,Destinataire,LValeurs],PlacePN2)

    test du predicat :
    M = [declencheurRecevoirMessage,[ouverture,p1,p2,[]]], M = [_,M1],mkMethode(declencheurRecevoirMessage,M1,PlacePN2).

    Production de :
    PlacePN2 = '[placePN,[RMrecu_ouverture_p1_p2]]'

    Un declencheur produit une place de synchonisation:logique!

*************************************************************************************************************/


%recevoir un message est un méthode dont l'appel est declenche par l'arrivee d'un message !
mkMethode(declencheurRecevoirMessage,[Type,Emetteur,Destinataire],PlacePN2):-
    mkConcat(['rm_recu_',Type,'_',Emetteur,'_',Destinataire],StringNom1),
    mkNomPlacePN([StringNom1],PlacePN2).
```

Ici traitement du choix

```
/*************************************************************************************************************
    traiterChoix(NET,NAC,Einitial,[NomDeclencheur,Args], [[Efinal,Condition]|R])

    test du predicat :

    Production de :

    transition([transitionPN,[1,2,choix_1]],[[ack,non]],[ ],
    [[placePN,[p1_attente_ouv_Exit]],[placePN,[RMrecu_ack_p2_p1]]],
    [[placePN,[p1_inactif_Entry]]]).

    transition([transitionPN,[1,2,choix_2]],[[ack,oui]],[ ],
    [[placePN,[p1_attente_ouv_Exit]],[placePN,[RMrecu_ack_p2_p1]]],
    [[placePN,[p1_actif_Entry]]]).

    choix(1,2,p1_attente_ouv,[declencheurRecevoirMessage,[ack,p2,p1]],
    [
    [p1_inactif,[[ack,non]]],   %NET,NAC,MESSAGE,ExpressionCondition
    [p1_actif,[[ack,oui]]]
    ]
    ).

*************************************************************************************************************/
    % à appeler avec NUmChoix =1
traiterChoix(_,_,_,_, [],_NumChoix):-!.
traiterChoix(NET,NAC,Einitial,[NomDeclencheur,Args], [[Efinal,Condition]|R],NumChoix):-
    mkPlaceExit(Einitial,StringPlaceExitAmont),
    mkMethode(NomDeclencheur,Args,StringPlaceDeclencheur),
    LPAmont = [StringPlaceExitAmont,StringPlaceDeclencheur],

    mkPlaceEntry(Efinal,StringPlaceEntryAval),
    LPAval = [StringPlaceEntryAval],
    mkConcat(['choix_',NumChoix],StringNomT),
    mkNomTransitionPNSimple([NET,NAC,StringNomT],NomTransition),
    mkFormatTransition(NomTransition,LPAmont,LPAval,Condition,[], RdP),
    mywrite(RdP),mynl,
    NextNumChoix is NumChoix +1,
    traiterChoix(NET,NAC,Einitial,[NomDeclencheur,Args],R,NextNumChoix).
```

Ici construction des transitions du RdP

```
/*****************************************
Les transitions correspondant aux états
*****************************************/
transitionEtat(NomEtat):-
    descriptionEtat(NomEtat,[NomMethode,Affectation]),
    mkPlaceEntry(NomEtat,PlacePN1),
    mkPlaceExit(NomEtat,PlacePN4),
    mkMethode(traitementInterne, [NomMethode,Affectation]),
    mkConcat([NomMethode,'_appel'],StringNom1),
    mkNomPlacePN([StringNom1],PlacePN2),
    mkConcat([NomMethode,'_retour'],StringNom2),
    mkNomPlacePN([StringNom2],PlacePN3),
    mkConcat(['interne1_',NomEtat],ST1),
    mkConcat(['interne2_',NomEtat],ST2),
    mkNomTransitionPNSimple([ST1],NomTransition1),
    mkNomTransitionPNSimple([ST2],NomTransition2),
    mkFormatTransition(NomTransition1,[PlacePN1],[PlacePN2],[],[], TransitionPN1),
    mywrite(TransitionPN1),mynl,
    mkFormatTransition(NomTransition2,[PlacePN3],[PlacePN4],[],[], TransitionPN2),
    mywrite(TransitionPN2),mynl.
transitionEtat(NomEtat):-
    descriptionEtat(NomEtat,[]),
    mkConcat(['interne_',NomEtat],NomTransition),
    mkPlaceEntry(NomEtat,PlacePN1),
    mkPlaceExit(NomEtat,PlacePN2),
    mkFormatTransition(NomTransition,[PlacePN1],[PlacePN2],[],[], TransitionPN),
    mywrite(TransitionPN),mynl.
transitionsPourEtats(LT):-
    setof(T,transitionEtat(T),LT).
/*****************************************
Les transitions correspondant aux choix
*****************************************/
transitionChoix([NET,NAC]):-
    choix(NET,NAC,Einitial,[NomDeclencheur,Args],ListeChoix),
    traiterChoix(NET,NAC,Einitial,[NomDeclencheur,Args],ListeChoix,1).
% Fabrication de toutes les transitions
transitionsPourChoix(LT):-
    setof(T,transitionChoix(T),LT).
```

```
    /*****************************************
Les transitions correspondant aux DET
*****************************************/


mkPlacesAmont(Einitial,[],PlacesAmont):-!,
    mkPlaceExit(Einitial,P1),
    PlacesAmont = [P1].

mkPlacesAmont(Einitial,[NomDeclencheur,Args],PlacesAmont):-
    mkPlaceExit(Einitial,P1),
    mkMethode(NomDeclencheur, Args,P2),
        PlacesAmont = [P1,P2].

mkPlacesAval(Efinal,PlacesAval):-
    mkPlaceEntry(Efinal,P1),
    PlacesAval=[P1].


/* transition DET sans action*/

transition([NET,NAC]):-
    transitionDET(NET,NAC,Einitial,Efinal,Declencheur,LConditions, []),
    mkPlacesAmont(Einitial,Declencheur,LPAmont),
    mkPlacesAval(Efinal,LPAval),
    mkNomTransitionPN(NET,NAC,NomTransition),
    mkFormatTransition(NomTransition,LPAmont,LPAval,LConditions,[], TransitionPN),
    mywrite(TransitionPN),mynl.
```

```
/*
Quand il y a des actions
ex : [actionEnvoyerMessage,[ack,p2,p1,[[ack,oui]]]]
4 cas comme d'hab
*/

transition([NET,NAC]):-
    transitionDET(NET,NAC,Einitial,Efinal,Declencheur,LConditions,[NomAction,Args]),
    mkMethode(NomAction, NET, NAC,Args,PlaceDebut,PlaceTemoin),

    mkPlacesAmont(Einitial,Declencheur,LPAmont),
    mkPlacesAval(Efinal,LPAval),

    mkNomPlacePN([NET,NAC,placeIntermediaire],PlaceIntermediaire),
    LPAvalIntermediaire = [PlaceDebut,PlaceIntermediaire],
    LPAmontIntermediaire = [PlaceIntermediaire,PlaceTemoin],

    mkNomTransitionPNSimple([NET,NAC,lancement],NomTransition1),
    mkFormatTransition(NomTransition1,LPAmont,LPAvalIntermediaire,LConditions,[], TransitionPN1),
    mywrite(TransitionPN1),mynl,

    mkNomTransitionPNSimple([NET,NAC,synchro],NomTransition2),
    mkFormatTransition(NomTransition2,LPAmontIntermediaire,LPAval,[],[], TransitionPN2),
    mywrite(TransitionPN2),mynl.

transitionsPourDET(LT):-
    setof(T,transition(T),LT).
```

```
/* une transition qui a un declencheur et une condition*/
trouverCondition(Einitial,Declencheur,LC):-
    transitionDET(_,_,Einitial,_,Declencheur,LC,_),
    Declencheur \= [],
    LC \= [].

/* les transitions qui ont des declencheurs et des conditions */
listerConditions(Einitial,Declencheur,ListCond):-
    setof(LC,trouverCondition(Einitial,Declencheur,LC),ListCond).


negationListeConditions([],[]):-!.

%en toute rigueur, il ne faudrait pas encadrer le Cond par des crochets !
% on le fait car on se sait traiter que ce cas
negationListeConditions([[Cond]|RCond],[[non|Cond]|RNon]):-
    negationListeConditions(RCond,RNon).


traiterCas(Cle):-
    listerConditions(Einitial,Declencheur,ListCond),
    mkListeConcat([Einitial,Declencheur],Cle),
    negationListeConditions(ListCond,Neg),
    mkPlaceExit(Einitial,P1),
    LPAmont = [P1],
    LPAval = [P1],
    mkNomTransitionPNSimple([exclusion,Einitial,Declencheur],NomTransition),
    mkFormatTransition(NomTransition,LPAmont,LPAval,Neg,[], TransitionPN),
    mywrite(TransitionPN),mynl.

traiterTousCas(L):-
    findall(Cle,traiterCas(Cle),L).
```

```
mkGeneral([L1,L2,L3,L4]):-
    consult('chaines.pl'),
    consult('formatages.pl'),
    consult('io.pl'),
    consult('uml.pl'),
    vider,
    traiterTousCas(L1),
    transitionsPourDET(L2),
    transitionsPourChoix(L3),
    transitionsPourEtats(L4),
    marquage_initial(MI),
    mkListeConcat(MI,StringMi),
    mkConcat(['minitial(',StringMi,').'],PredicatMinitial),
    mywrite(PredicatMinitial).


    %il MANQUE LE MARQUAGE INITIAL

    etat_initial_anonyme(E):-
        etat_initial(_,E).

    marquage_initial(MI):-
        setof(E,etat_initial_anonyme(E),LM),
        traiterMI(LM,MI).

    traiterMI([],[]):-!.
    traiterMI([X|R],[Y|R2]):-
        mkPlaceEntry(X,Y),
        traiterMI(R,R2).
```

Annexe – H : Exemples de simulation ERTMS

Soit le diagramme de classes en UML représentant les deux composants : EVC et RBC

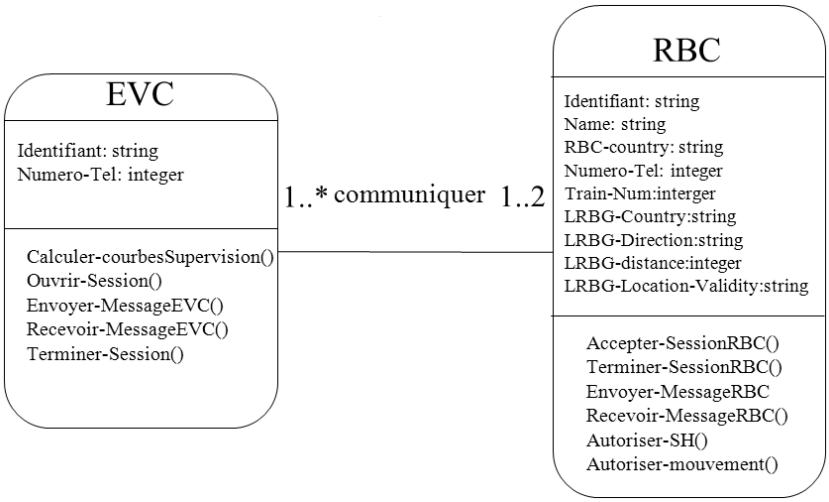


Figure a– un diagramme de classes simplifié

Premier scénario : établissement d'une session de communication avec le RBC et train rejeté par le RBC.

Le scénario est décrit de la manière suivante :

- EVC envoie au RBC une demande d'ouverture de session (message 155) ;
- RBC accepte d'ouvrir la session et envoie à l'EVC le message 32 « configuration determination »
- EVC vérifie la position du train enregistrée
  - Si la position est valide alors il envoie le message 157 au RBC avec un Q-status= 01 et ensuite il reste en attente de sélection du conducteur.
  - Si la position est invalide ou inconnue alors il envoie le message 157 avec un Q-status= 00 ou 10
    - ✓ RBC rejette le train et répond à l'EVC en envoyant le message 40
    - ✓ EVC supprime la position enregistrée du train et la met à inconnue
    - ✓ EVC envoie le message 156 au RBC pour terminer la session
    - ✓ RBC répond par le message 39 pour accuser réception
    - ✓ RVC affiche au DMI « train rejeté ».

Ce scénario est détaillé dans le diagramme de séquences UML et les diagrammes d'états suivants :

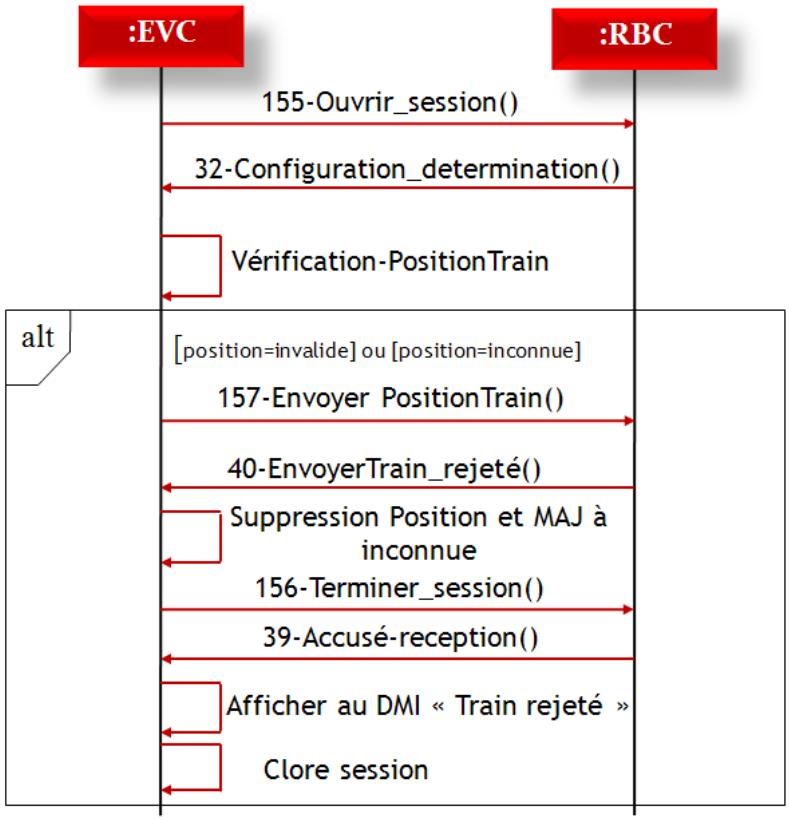


Figure b– diagramme de séquences du scénario

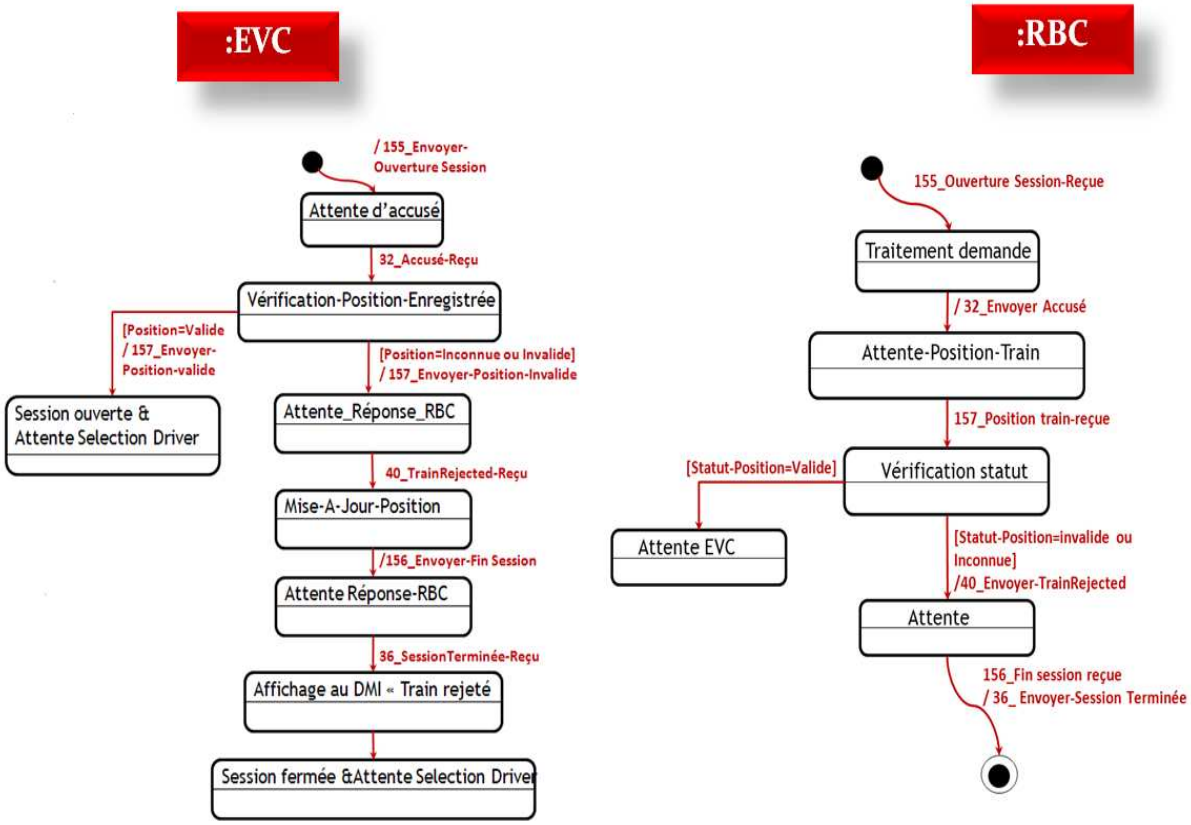


Figure c– diagrammes d'états du RBC et du EVC

L'exécution de ce scénario de test sur la plate-forme de simulation ERTMS.

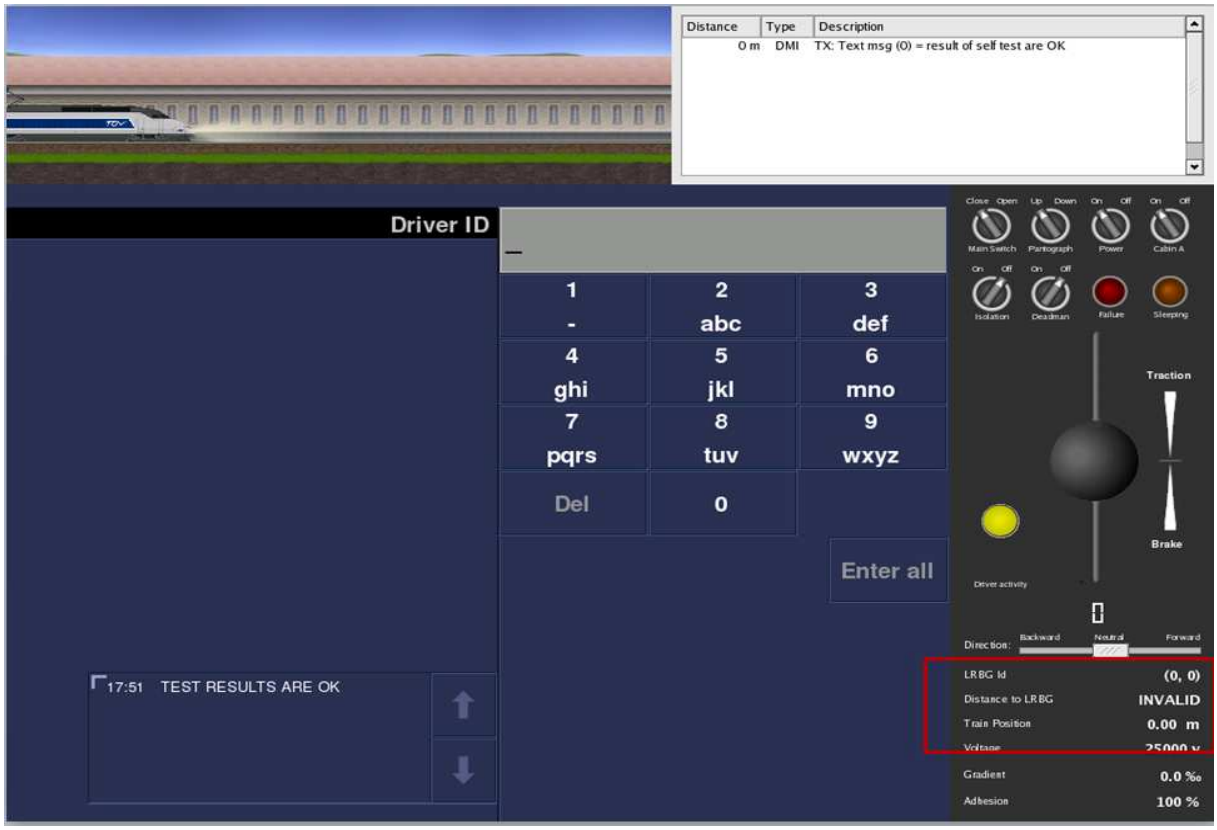


Figure d- Position enregistrée du train invalide

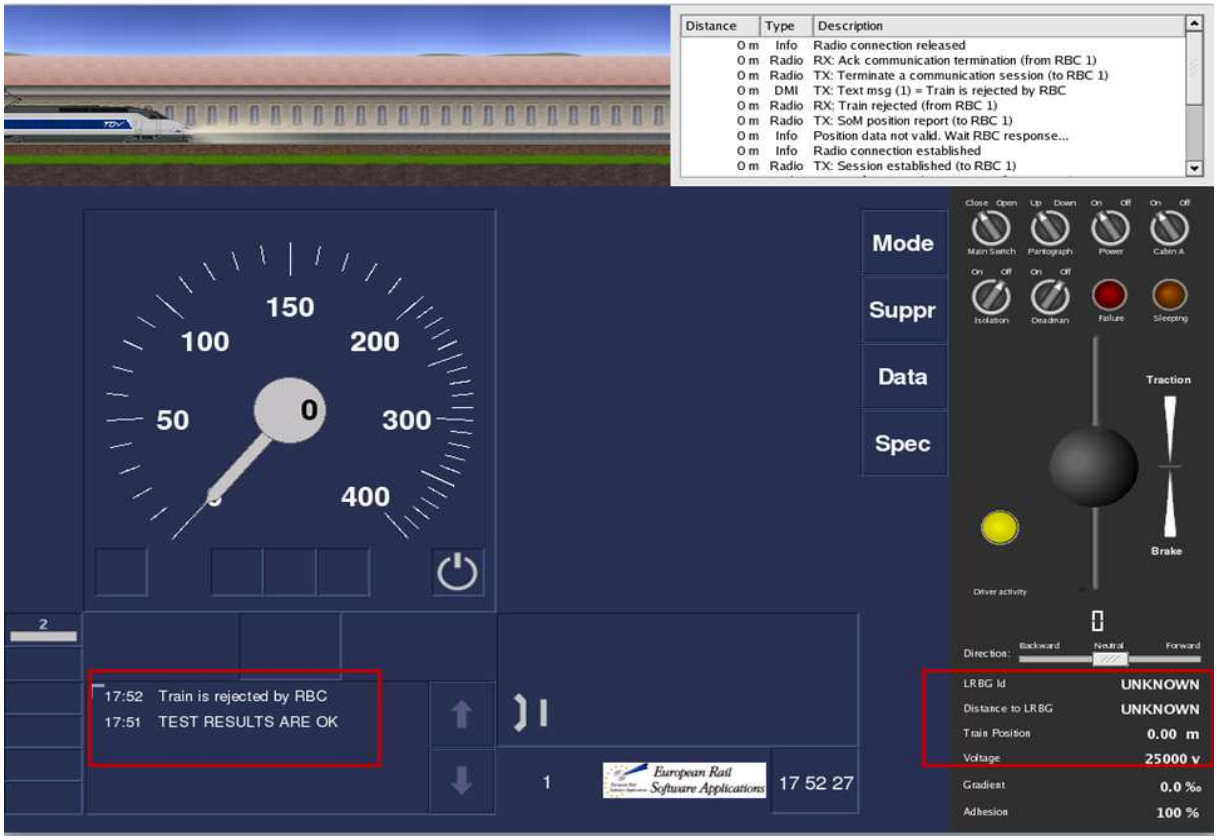


Figure e- Position mise à jour et train rejeté par RBC

```
(15.44 s,  0.00 m) - TIU - RX: Power = ON

(15.44 s,  0.00 m) - TIU - TX: DeadManIsolation = ON

(15.54 s,  0.00 m) - BRAKE - EB: ON --> OFF

(15.54 s,  0.00 m) - SPEED - front end location = 0 m, train speed = 0 km/h (P = 0 km/h, W = 0 km/h, SBI = 0 km/h, EBI = 0 km/h)

(16.25 s,  0.00 m) - TIU - RX: Cabin = OPEN

(16.33 s,  0.00 m) - MMI - TX: Text msg (0) = result of self test are OK

(25.16 s,  0.00 m) - MMI - RX: Entry of driver id

(25.27 s,  0.00 m) - RADIO - TX, RBC = 1, PhoneNb = 1802030001FFFFFF CONNECTION

(25.72 s,  0.00 m) - RADIO - RX, RBC = 1, PhoneNb = 1802030001FFFFFF CONNECTION

(25.70 s,  0.00 m) - RADIO - TX, RBC = 1, PhoneNb = 1802030001FFFFFF DATA NID_MESSAGE = 155, content = 9D 02 00 00 02 03 00 00 00 40

(26.13 s,  0.00 m) - RADIO - RX, RBC = 1, PhoneNb = 1802030001FFFFFF DATA NID_MESSAGE = 32, content = 20 02 C0 00 02 83 5F FF FF E4 3F

(26.19 s,  0.00 m) - RADIO - TX, RBC = 1, PhoneNb = 1802030001FFFFFF DATA NID_MESSAGE = 159, content = 9F 05 40 00 02 8D 00 00 00 40 C0 B4 10 52 20 30 00 1F FF FF F0

(26.29 s,  0.00 m) - RADIO - TX, RBC = 1, PhoneNb = 1802030001FFFFFF DATA NID_MESSAGE = 157, content = 9D 06 80 00 02 8F 80 00 00 40 00 40 BF FF FF FF FF E8 00 28 00

(28.86 s,  0.00 m) - RADIO - RX, RBC = 1, PhoneNb = 1802030001FFFFFF DATA NID_MESSAGE = 40, content = 28 02 80 00 02 C3 DF FF FF FF

(29.00 s,  0.00 m) - MMI - TX: Text msg (1) = Train is rejected by RBC

(29.04 s,  0.00 m) - RADIO - TX, RBC = 1, PhoneNb = 1802030001FFFFFF DATA NID_MESSAGE = 156, content = 9C 02 80 00 02 D4 00 00 00 40

(29.37 s,  0.00 m) - RADIO - RX, RBC = 1, PhoneNb = 1802030001FFFFFF DATA NID_MESSAGE = 39, content = 27 02 80 00 02 D4 9F FF FF FF

(29.45 s,  0.00 m) - RADIO - TX, RBC = 1, PhoneNb = 1802030001FFFFFF DISCONNECTION

(29.47 s,  0.00 m) - RADIO - RX, RBC = 1, PhoneNb = 1802030001FFFFFF DISCONNECTION
```

Figure f– fichier des interactions contenant que les transitions aux interfaces

Deuxième scénario : Mode « shunting initié par le conducteur.

Hypothèse : train en arrêt en mode SB et session a été ouverte par scénario précédent

Le scénario est décrit de la manière suivante :

- Sélection du passage en mode « shunting » par le conducteur ;
- EVC envoie au RBC le message 130 « Request for shunting » pour demander une autorisation de passage en mode « shunting » ;
- EVC affiche le message « SH request pending » et reste en attente de la réponse du RBC
- RBC vérifie l'existence d'une zone de « shunting » et vérifie la position du train qui est reconfirmée à travers le message 130 envoyé par EVC ;
    - Si le train est sur une zone de « shunting » alors RBC autorise le « shunting » et envoie le message 28 « SH authorised » à l'EVC et reste en attente de la fin de mission initiée par le conducteur.
    - Sinon RBC refuse le passage en mode « shunting » et envoie le message 27 « SH refused » à l'EVC.

Si nous prenons le cas où le train n'est pas sur une zone de « « shunting » alors le diagramme de séquences et les diagrammes d'états UML correspondants à ce scénario sont décrits dans ce qui suit :

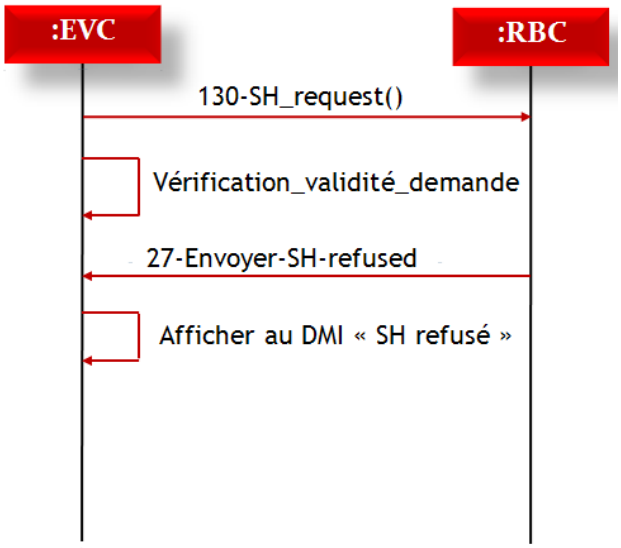


Figure g- diagramme de séquences du scénario

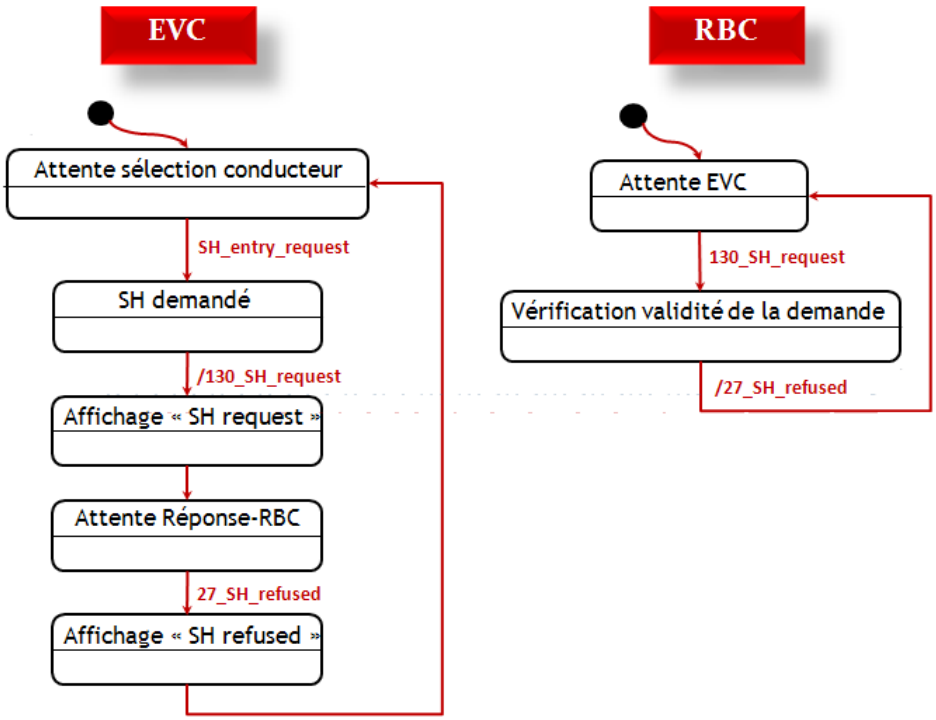


Figure h- diagramme d'états du RBC et de l'EVC

L'exécution de ce scénario de test sur la plate-forme de simulation ERTMS.

Figure i– Début du scénario

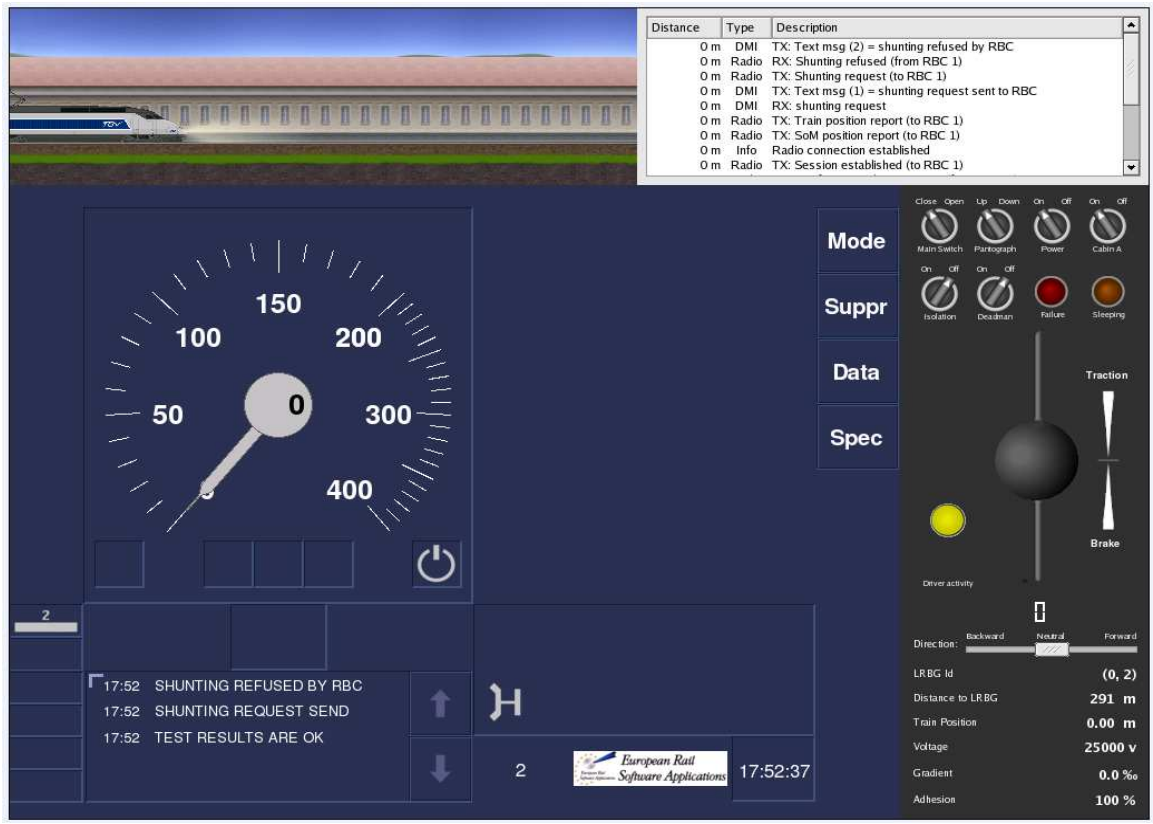


Figure j– Fin du scénario

Titre : Génération de scénarios de tests pour la vérification de systèmes répartis : application au système européen de signalisation ferroviaire (ERTMS).

Dans les années 90, la commission européenne a sollicité la mise au point d'un système de contrôle commande et de signalisation ferroviaire commun à tous les réseaux des états membres : le système ERTMS « European Railway Traffic Management System ». Il s'agit d'un système réparti complexe dont le déploiement complet est long et coûteux. L'objectif global consiste à diminuer les coûts de validation et de certification liés à la mise en œuvre de ce nouveau système en Europe. La problématique scientifique réside dans la modélisation formelle de la spécification afin de permettre la génération automatique des scénarios de test. Les verrous scientifiques, traités dans cette thèse, sont liés d'une part à la transformation de modèle semi-formel en modèle formel en préservant les propriétés structurelles et fonctionnelles des constituants réactifs du système réparti, et d'autre part à la couverture des tests générés automatiquement. Les constituants sont sous la forme d'une boîte noire. L'objectif consiste à tester ces derniers à travers la spécification ERTMS. Nous avons développé une approche de modélisation basée sur le couplage de modèles semi-formels (UML) et de modèles formels (Réseaux de Petri). Ce couplage se fait à travers une technique de transformation de modèles. Nous avons développé ensuite une méthode de génération automatique de scénarios de test de conformité à partir des modèles en réseaux de Petri. Les scénarios de test ont été considérés comme une séquence de franchissement filtrée puis réduite du réseau de Petri interprété représentant la spécification. Ces scénarios ont été exécutés sur notre plateforme de simulation ERTMS.

Mots clés : Génération de scénarios de test, modélisation objet et modélisation formelle, transformation de modèles, interopérabilité ferroviaire, système ERTMS.

Title: Generation of test scenarios for distributed system checking: application to the European Railway Traffic Management System (ERTMS).

European Union set up a European rail traffic management system "ERTMS" to ensure, with high level of safety, train operation on different European networks. As the full deployment of this system is long and expensive, evolutions are necessary. The goal is to determine how to use ERTMS specifications to produce test scenarios. This work presents methods, models and tools dedicated to the generation of test scenarios for the validation of ERTMS components based on functional requirements. Evaluation and certification of the ERTMS system can be done by generating test scenarios applying formal methods. The Unified Modelling Language (UML) is a widely accepted Modelling standard in industry. However, it is a semi-formal language and it does not allow verification of system behavior. In this case, formal models like Petri Net can be used. These methods are used in order to formalize ERTMS specification. Tests scenarios are generated on the basis of Petri net models. One scenario is considered like a firing sequence in the reachability graph of the Petri net. Then, test scenarios are applied on ERTMS platform simulator in order to check the components and to give test verdicts. Finally, the approach, developed in this document, has been applied to ERTMS components in order to demonstrate the validation and certification costs reduction and also to minimize the upgrade and retrofit constraints and validation cost.

Keywords: Test scenarios generation, Object and formal modeling, model transformation, railway interoperability, ERTMS system.