Spécification et Vérification des Propriétés Temporelles Qualitatives et Quantitatives des Systèmes Temps Réel Complexes

Abdia Hamdani

N° d'ordre :10/2023.D/IN

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITÉ DES SCIENCES ET DE LA TECHNOLOGIE HOUARI BOUMÉDIÈNE

FACULTÉ D'INFORMATIQUE

THÈSE DE DOCTORAT EN SCIENCES

Présentée pour l'obtention du grade de DOCTEUR

En: Informatique

Spécialité: Informatique

Par:

HAMDANI ABDIA

THÈME :

Spécification et Vérification des Propriétés Temporelles Qualitatives et Quantitatives des Systèmes Temps Réel Complexes

---

Soutenue Publiquement le 16/05/2023, Devant le jury composé de:

M. Y.HAMMAL	Professeur à L'USTHB	Président
M. A .ABDELLI	Professeur à L'USTHB	Directeur de Thèse
M. M.C BOUKALA	Maître de Conférences à L'USTHB	Examinateur
Mme. F .BELALA	Professeur à L'Université de Constantine	Examinatrice
Mme. K.BENATCHBA	Professeur à L'ESI	Examinatrice
M. L. SAKHRI	Professeur à L'Université Oran1	Examinateur

A mes chers parents...
...et Mari...

Résumé

L'analyse et la vérification des systèmes temps réels complexes permettent de répondre aux questions liées, essentiellement, à la fiabilité et à la sûreté de fonctionnement de ces derniers. En effet, le *Model-Cheking* : vérification du/sur le modèle, est une méthode très connue permettant de vérifier les propriétés qualitatives (e.g : accessibilité d'un état, vivacité, bloquage, etc...) et/ou quantitatives (e.g : la réponse du temps borné) sur le modèle. Le formalisme des réseaux de Petri et ses extensions sont largement utilisés dans la littérature, offrant un nombre considérables d'avantages citons : (sa représentation graphique et intuitive, ses techniques abondantes d'analyses...etc.). Nous nous intéressons aux réseaux de Petri comme outil de modélisation, étendu au temps et aux mécanismes de synchronisation. En conséquence, une nouvelle extension du modèle de base est proposée notée $RTPN$ pour : (Time Petri Nets with Rendezvous). Ensuite, nous proposons une sous-classe du modèle proposé, noté $RTWFN$ pour : (Time Workflow Nets with Rendezvous) dédiée aux réseaux de Workflows. Une méthode d'abstraction de l'espace d'états (généralement infini) du modèle est proposée par la suite. La méthode consiste à regrouper un ensemble d'états partageant les même propriétés et basé sur le formalisme de *classe d'état*. Enfin, l'analyse et la vérification des propriétés *qualitatives* et *quantitatives* sur le modèle ,et bien évidement, sur le système en question devient possibles.

Mots Clé : Analyse et vérification, model-cheking, propriétés qualitatives, propriétés quantitatives, réseaux de Petri Temporels (RdpT), Rendezvous, Workflow, abstraction de l'espace d'états.

Abstract

Analyzing and verifying a complex real time systems permits to give answers about their correctness, reliability and safety. Indeed, *Model-Cheking* : check on the model, is a very well known method to verify the qualitative (e.g. : stat accessibility, liveness, blocking, etc...) and quantitative properties (e.g. : time response) on the model. The Petri net formalism and its extensions are widely used in the literature, with an important number of advantages as : (theirs graphical and intuitive representations, an abondants analysis techniques...etc.). We are interested on the Petri net formalism as a modelling tool, extended to time and synchronization mechanisms. Thus, a new extension of the basic model is defined noted $RTPN$ for : (Time Petri Nets with Rendezvous). Then, a sub-class of the proposed model is defined noted $RTWFN$ for : (Time Workflow Nets with Rendezvous) dedicated to the workflow nets. A method for the state space (generally infinite) abstraction of the proposed model is proposed. The method consists on gethering a set of states sharing some properties and based on the *State Class* formalism. Finally, analysing and verifying a qualitative and quantitative properties on the model as well as on the system itself became possible.

Keywords : Analysis and verification, model-cheking, qualitatives properties, quantitatives proprerties, Time Petri Nets (TPN), Rendezvous, Workflow , state space abstraction.

Table des matières

1 Introduction Générale **1**
- 1.1 Contexte général . . . 1
- 1.2 Problématique et motivation . . . 1
- 1.3 Objectif . . . 3
- 1.4 Contribution . . . 3
- 1.5 Organisation du document . . . 4

2 Modélisation, spécification et vérification des systèmes temps réel **5**
- 2.1 Introduction . . . 5
- 2.2 La modélisation des systèmes temps réel . . . 5
  - 2.2.1 Les modèles temporels de synchronisation . . . 6
  - 2.2.2 Les réseaux de Petri et le temps... . . . 6
  - 2.2.3 Les réseaux de Petri temporels : Modularité et synchronisation... . . . 8
  - 2.2.4 Réseaux de Petri et Workflow... . . . 11
  - 2.2.5 Modèles Non basés sur les réseaux de Petri . . . 11
- 2.3 Analyse et vérification des systèmes temps réel . . . 11
  - 2.3.1 Propriétés qualitatives et quantitatives . . . 12
  - 2.3.2 Et la logique dans tous ça ? . . . 12
  - 2.3.3 Espace d'états des réseaux de Petri temporels . . . 12
- 2.4 Conclusion . . . 13

3 Les réseaux de Petri : état de l'art et extensions **14**
- 3.1 Introduction . . . 14
- 3.2 Les réseaux de Petri . . . 14
  - 3.2.1 Présentation informelle . . . 14
  - 3.2.2 Présentation formelle . . . 15
  - 3.2.3 Marquage d'un RdP . . . 15
  - 3.2.4 Règles de fonctionnement . . . 16
  - 3.2.5 Transitions nouvellement sensibilisées . . . 16
  - 3.2.6 Sémantique du parallélisme . . . 16
  - 3.2.7 Conflit et parallélisme . . . 17
  - 3.2.8 Co-begin/Co-end et parallélisme . . . 18
  - 3.2.9 Propriétés qualitatives des *RdP* . . . 18
  - 3.2.10 Graphe de couverture (de marquage) . . . 19
- 3.3 Extensions temporisées des RdP . . . 20
  - 3.3.1 Les réseaux de Petri t-Temporels : *RdPT* . . . 21
- 3.4 Autres extensions... . . . 24
  - 3.4.1 Réseau de Petri à Flux ($TSPN$) . . . 24
  - 3.4.2 Réseaux de Petri modulaires . . . 26
- 3.5 Conclusion . . . 28

4 Espace d'états des RdPT **29**
- 4.1 Introduction . . . 29
- 4.2 La méthode du graphe des classes d'états de *Berthomieu et al* . . . 29
  - 4.2.1 Le graphe des classes d'états . . . 29
  - 4.2.2 Graphe des classes d'état fortes . . . 34
  - 4.2.3 Graphes de classes d'états atomiques . . . 37

4.3 La méthode des matrices de distances de *Boucheneb* pour le calcul du graphe des classes . 38
4.4 Discussion et Comparaison . . . 40
4.5 Conclusion . . . 41

5 Synchronisation et contraintes temporelles : Notion de *Rendez-vous* 42
5.1 Introduction . . . 42
5.2 Présentation du concept *Rendez-vous* . . . 42
5.3 Les stratégies d'un *Rendez-vous* . . . 44
5.3.1 Les patterns *One-Begin* . . . 45
5.3.2 Les patterns *One-End* . . . 45
5.3.3 Les patterns *Co-Begin* . . . 47
5.3.4 Les patterns *CoEnd* . . . 48
5.4 Les Règles génériques d'un *Rendez-vous* . . . 50
5.5 Discussion et Comparaison . . . 50
5.6 Conclusion . . . 51

6 Les réseaux de Petri Temporels avec Rendez-vous : $RTPN$ 52
6.1 Introduction . . . 52
6.2 Syntaxe formelle d'un $RTPN$ . . . 52
6.3 Sémantique formelle d'un $RTPN$ . . . 53
6.4 Application : les $RTPN$ et les systèmes Workflow . . . 56
6.5 Exemple : Le Workflow d'un patient . . . 56
6.6 Comparaison . . . 58
6.7 Conclusion . . . 60

7 Construction du graphe des classes d'un $RTPN$ 61
7.1 Introduction . . . 61
7.2 Graphe des classes d'un $RTPN$ . . . 61
7.2.1 Complexité . . . 64
7.2.2 Équivalence entre classes d'état . . . 65
7.2.3 Finitude du graphe des classes d'un $RTPN$ . . . 65
7.3 Algorithme d'énumération du graphe des classes d'un $RTPN$ . . . 65
7.4 Exemple illustratif . . . 66
7.5 Conclusion . . . 73

8 Analyse de l'espace d'état d'un $RTPN$ 75
8.1 Introduction . . . 75
8.2 Les propriétés qualitatives et quantitatives du graphe des classes d'un $RTPN$ . . . 75
8.3 Temps minimal et maximal d'une séquence de franchissement d'un $RTPN$ . . . 76
8.4 Conclusion . . . 79

9 Conclusion Générale 80
9.1 Bilan . . . 80
9.2 Perspectives . . . 81

10 Annexe A : Preuves 88
10.1 Preuve de la Proposition 7.2.1 . . . 88
10.2 Preuve de la Proposition 7.2.2 . . . 89
10.3 Preuve de la Proposition 8.3.1 . . . 93

Table des figures

2.1 Relations temporelles basiques d'Allen. 6
2.2 Relations temporelles avec délais de Wahl 7
2.3 Modèle de Kermane 7

3.1 Exemple d'un RdP 16
3.2 Transitions sensibilisées : a) conflit, b) parallélisme 17
3.3 Co-begin / Co-end 18
3.4 Co-begin / Co-end et le parallélisme 18
3.5 Graphe de marquage : a) Conflit, b) Parallélisme 20
3.6 Evolution d'un Rdp [1] 20
3.7 Graphe de couverture 21
3.8 Un réseau de Petri T-temporel 21
3.9 Evolution d'un *RdpT* 22
3.10 Synchronisations dans un Réseau de Petri à Flux 25
3.11 Exemple d'un $TSPN$ 25
3.12 Exemple de réseau de Petri modulaire 26
3.13 Exemple de deux composants (a) : Composant C_1, (b) : Composant C_2 27
3.14 Composants NON-Composables $C_{1x2} = C_1 \parallel C_2$ 27
3.15 Exemple d'un $STPTPN$ [1] 28

4.1 Etat et classe d'états [2] 30
4.2 Exemple illustratif 32
4.3 Graphe de classes d'états du *RdPT* Fig.4.2 33
4.4 Les graphes de classes du $TSPN$ Fig. 3.11 [3] et $STPTPN$ Fig.3.15 [1] 34
4.5 Exemple illustratif [4] 36
4.6 a) RdPT b) graphe des classes LTL c) graphe des classes CTL 37
4.7 Valuations d'horloges 38
4.8 a) Exemple d'un *RdpT* et b) son graphe de classes. 40

5.1 Schéma de synchronisation : retard / avance 43
5.2 Les patterns : One-Begin et One-End 46
5.3 Les patterns : a)-CoBegin et b)- CoEnd. 48

6.1 Exemple d'un RTPN 53
6.2 Evolution d'un $RTPN$ 53
6.3 L'approche de modélisation pour le $RTWFN$ 56
6.4 Le $RTWFN$ représentant : Patient workflow 59
6.5 $RTPN$ vs *RdPT* et $TSPN$ 60

7.1 Exemple d'un $RTPN$ 68
7.2 Le graphe de classes correspondant au $RTPN$ 8.3 74

Liste des tableaux

2.1 Modèle de Weiss 6
2.2 Comparaison des modèles basées *Rdp* les plus proches 10

3.1 Interprétation des Places /Transitions dans un *RdP* [5] 15

4.1 Tableau comparatif 41

5.1 Règles de synchronisation 50
5.2 Comparaison 51

7.1 La matrice représentant le système D_0 66
7.2 Les matrices représentant le système β_0 68
7.3 $\lambda_0[t]$ et $DR_0[R]$ 68
7.4 Les matrices représentant les système D_1 et β_1 71
7.5 $\lambda_1[t]$ et $DR_1[R]$ 71
7.6 Les matrices représentant les système $D_2, \beta_2, \lambda_2[t]$ et $DR_2[R]$ 71

8.1 Distance de temps au point (0) 78
8.2 Distance de temps au point (1) 78
8.3 Distance de temps au point (2) 78

Chapitre 1

Introduction Générale

1.1 Contexte général

De nos jours, avoir un système informatique *sûr* (c'est-à-dire sans aucune erreur ou risque), et *fiable* (c'est-à-dire répondant à toute exigence et attente) est un vrai challenge. En effet, le monde évolue à une vitesse considérable et les exigences aussi, et tous les domaines sont concernés : (par exemple santé, économie, transport, finance, ..., etc).

Un système temps réel est un système (application ou ensemble d'applications) informatique dont le fonctionnement est assujetti à l'évolution dynamique d'un procédé extérieur qui lui est connecté et dont il doit contrôler le comportement. La correction d'un système temps réel dépend non seulement de la *justesse* des calculs, mais aussi du *temps* auquel les résultats sont produits (contraintes temporelles)[6]. Le mauvais fonctionnement de tels systèmes peut mettre en péril de nombreuses vies humaines ou des investissements financiers colossaux. D'ailleurs, des exemples de dysfonctionnements informatiques ne manquent pas dans la littérature, ces derniers auraient pu être évités si des procédures strictes de vérification avaient été mises en place (e.g.la catastrophe nucléaire de Tchernobyl le 26 avril 1986, URSS).

Les systèmes temps réels complexes sont des systèmes étendus avec de nouvelles sémantiques (e.g : modularité, préemption, synchronisation, etc.). Dans de tels systèmes, la *communication* et la *synchronisation* sont deux mécanismes importants, différents, mais très liés. En effet, si la communication exprime le fait d'échanger, de partager et de collecter des informations entre applications ou processus, la synchronisation, quand t- à elle, permet que cet échange se passe au mieux et en de bonnes conditions. Les bonnes conditions expriment aussi le fait que les contraintes temporelles exigées ont été belles et bien respectées, plus particulièrement, celles liées au délai (imposées ou permis) entre évènements [7].

Le rôle majeur que joue la synchronisation dans les systèmes temps réels complexes (e.g, composition de services Web, systèmes multimédia, les réseaux IoT avec leurs contraintes, protocoles de communication, ..., etc), nous pousse à proposer de nouveaux modèles plus expressifs pour capturer ces nouvelles sémantiques et couvrir un large panel de possibilités de synchronisation (en plus de celles déjà définies dans la littérature). En effet, nous avons la modélisation des processus concurrents (en parallèle) d'une part, et d'autre part, la vérification de leurs propriétés les plus intéressantes (e.g : sûreté, fiabilité, temps de réponse, etc.).

C'est dans ce contexte général que se situe cette thèse. Nous y développons une nouvelle technique pour une vérification plus efficace et un nouveau modèle permettant de prendre en compte l'évolution de la complexité des systèmes temps réel.

1.2 Problématique et motivation

Le besoin de définir des modèles formelles et des méthodologies d'analyse et de vérification de la correction des systèmes informatiques devient un vrai challenge, notamment avec l'avènement des nouvelles technologies réseaux apportant leurs lots de contraintes de synchronisations et de temps. Concrètement, pour éviter des dysfonctionnements et toute mauvaise surprise, les concepteurs des systèmes ont besoin d'anticiper toute erreur et ceci grâce particulièrement à la *simulation* et au *Model-checking* dans le but de détecter les propriétés de ces systèmes avant de pouvoir les implémenter.

La communication, la *synchronisation* et les *délais* sont des aspects très importants à prendre en considération lorsque plusieurs occurrences de logiciels et de processus sont planifiés pour être exécutées en parallèle. Plus particulièrement, lorsque des objets et des ressources sont à partager. L'enchaînement

et la continuité doivent garantir une certaine coordination afin de maintenir l'échange d'information selon des délais (généralement imposés).

Si l'un des plus grands défis dans les systèmes temps réels complexes est de pouvoir gérer des situations où un nombre variable de tâches doit être exécuté sous différentes contraintes de temps et de modèles de synchronisations. S'adapter, re-planifier, et synchroniser de tels systèmes en réponse à des propriétés inattendues, délais ou conditions techniques sont nécessaires pour garantir la fiabilité du système. Certains domaines critiques tels que la santé ne permettent aucune erreur, car le temps est compté et des vies humaines en dépendent de chaque fait et geste, et les décisions à prendre ne sont pas aussi simples avec les contraintes de temps et de délai (synchronisation) [8, 9]. Prenons en exemple :

En chirurgie, durant l'activité de transplantation d'organe (le cas général), le sang et l'organe pour le patient doivent arriver durant la même heure à l'hôpital, pour éviter toute dégradation fonctionnelle. Au niveau des services d'urgence, à l'arrivée de l'ambulance, il faut prévoir des processus en parallèle pour prendre en charge les cas et les orienter vers les services dédiés.

Lors d'un don d'organe (lorsque le donneur est un membre de la famille), il faut prévoir deux processus en parallèle, l'un qui permet de récupérer l'organe (par exemple un rein), du donneur, et l'autre pour la transplantation de l'organe au récepteur.

Un autre exemple, après une prescription d'ordonnance par un médecin, la prise d'un ensemble de médicaments (en même temps) (par exemple les antibiotiques, anti-inflammatoires et pansement gastrique), il est généralement recommandé de prendre les pansements un peu avant les autres médicaments afin d'éviter des problèmes d'estomac. Pour le traitement de l'hypertension artérielle, il est souvent prescrit un anti-hypertenseur et un diurétique avec un certain délai entre les deux prises (si les comprimés/gélules séparés). Parfois ces derniers médicaments sont combinés dans les mêmes gélules, dans ce cas aucune contrainte de délai entre les prises de comprimés n'est prise en considération.

À d'autres égards, dans le e-Commerce, la synchronisation doit assurer que tous les produits soient délivrés au consommateur durant les délais imposés pour éviter tout problème lié au stockage ou à la logistique sur le site (entrepôt). Prenant l'exemple de deux produits A et B correspondant à une commande qui doit être établie et préparée par deux agents. La livraison commence lorsque tous les produits sont disponibles et aucun des produits ne doit attendre plus que 5 unités de temps à l'entrepôt. Une politique stricte qui permet d'optimiser l'exécution et d'augmenter la rentabilité des entreprises de vente en ligne (e.g AMAZON, Alibaba, AliExpress, etc.). En réalité, c'est plus compliqué que ça en a l'air, en effet, de tels sites doivent gérer des commandes de clients du monde entier.

Le passage à niveau est un cas d'étude très répandu dans la littérature pour présenter à la fois l'aspect critique et les contraintes de temps et de synchronisation. Le sujet étant critique, car plusieurs vies humaines en dépendent, sans oublier les pertes financières qui en résultent. Dans [10], la sécurité d'un système ferroviaire contraint par des délais est mise en question. En effet, l'arrivée du train et le soulèvement de la barrière se fait en même temps. Selon l'auteur, un retard de l'ordre de 7 sec est permis entre ces deux actions. Plusieurs auteurs dans la littérature [4, 11, 12, 13, 14] ont considéré cet exemple avec différentes variantes de synchronisation en reliant n instances de trains en parallèle avec un contrôleur (ou plusieurs), et une copie du modèle de barrière.

Dans la littérature, nous trouvons par ailleurs des systèmes où les tâches n'ont pas la même priorité pour une synchronisation donnée (e.g. problème de *lip-synchronization*). En effet, dans ce processus, l'audio avec une priorité maître et la vidéo (esclave) sont connectés via une synchronisation du type maître [15]. Dans ce cas, les délais permis entre ces deux entités sont bien définies dans les travaux de *Blakowski*, (par exemple la vidéo peut être avancée par rapport à l'audio, mais pas plus que 80ms). D'autres exemples prenant en compte les délais entre entités multimédia sont présentés dans [16, 7].

Plus récemment, pendant la pandémie du *Covid-19*, les cours en ligne et les *webinaire* sont devenus indispensables. Lors de la présentation des diapositives, en parallèle, le professeur (ou l'intervenant) doit commenter chaque contenu dans la diapositive. Visualiser la diapo quelque instants avant de commencer à la commenter est permis, voire nécessaire pour que les étudiants (et participants) aient le temps d'avoir une idée d'abord sur le contenu et faciliter par la suite la compréhension des commentaires du prof/intervenant. Les délais peuvent être dus à une mauvaise connexion ou coordination.

Les exigences des systèmes actuels en termes de temps, de nouveaux mécanismes de synchronisation complexes et de contraintes de temps appellent le besoin de définir de nouveaux modèles permettant de prendre en considération ces exigences. En effet, les modèles existants et particulièrement ceux basés sur les RdP ne traitent qu'un sous ensemble de ces contraintes. Le modèle doit pouvoir :

— Prendre en compte le facteur temps ;
— Exprimer un large panel de mécanismes de synchronisations (en plus de celles déjà définies en littérature) ;

— Respecter la compositionnalité des sous-processus ;
— Prendre en considération l'aspect priorité entre processus (maître/esclave) ;
— Prendre en considération les délais temporels entre les évènements parallèles.

Par conséquent, la définition d'un modèle capable de synthétiser de manière cohérente ces différentes contraintes, serait un premier pas important pour la prise en charge complète des problématiques des systèmes temps réels complexes. Un tel modèle, une fois défini, pourrait être exploité à la vérification formelle et à l'analyse quantitative des systèmes temps réels complexes, grâce aux techniques d'analyse par énumération permettant de dériver le graphe d'accessibilité correspondant, décrivant les comportements possibles du système modélisé.

1.3 Objectif

Nous proposons dans cette thèse une méthodologie de vérification permettant la spécification et l'analyse des systèmes temps réel complexes. Un nouveau paradigme noté *rendez-vous* est introduit comme un mécanisme flexible afin de designer n'importe quel schéma de synchronisation entre tâches concurrentes de différentes priorités et contraintes par le temps.

Au vu de ce constat, nous introduisons une nouvelle extension de RdPT [17] appelée RTPN pour (*Time Petri Net with Rendezvous*) [18] dédiée à la modélisation des systèmes temps réels complexes, et utilisant comme modèle de base un RdPT augmenté de mécanismes spécifiques et pourvu de sémantiques adaptées pour cet objectif :

— Nous définissons des stratégies qui gouvernent les caractéristiques d'un rendez-vous, ensuite nous dérivons les modèles de synchronisations associées.
— Nous définissons les règles générées et leurs types (maîtres /esclave) introduisant ainsi toutes les possibilités de combinaisons qui peuvent être générées à partir du concept de *rendez-vous*.

À partir de là, nous proposons une méthode d'énumération de l'espace d'état du modèle défini par la construction du graphe des classes d'état. Une classe d'état regroupe un ensemble d'états ayant les mêmes caractéristiques, et est donnée par la paire (M, D) où M est un marquage et D correspond à un système de contraintes sous forme DBM (difference bound Marix) relatif aux transitions sensibilisées, à partir duquel nous pouvons calculer le système relatif aux rendez-vous noté DR. Ce dernier est aussi de forme DBM permettant de réaliser le test de franchissement de manière efficace et flexible. Le processus d'énumération munie d'une relation d'équivalence basée sur l'égalité des classes permet de construire le graphe des classes contractant l'espace d'état du $RTPN$. Cette contraction permet de préserver les propriétés linéaires du modèle.

Finalement, les graphes ainsi calculés pourraient être utilisés, si finis, pour l'analyse des propriétés générales du système (propriétés qualitatives). Concernant les propriétés temps réel, elles pourront être déduites du graphe grâce à l'application d'une procédure calculant des temps minimaux et maximaux de n'importe quelle séquence de franchissement. Ces informations permettent l'analyse quantitative et l'évaluation des performances du système modélisé.

1.4 Contribution

Les contributions scientifiques des travaux entrepris durant cette thèse, peuvent être énumérées comme suit :

- Proposition d'un modèle formel riche basé sur les RdPT (appelé RTPN), permettant de concentrer plusieurs sémantiques. Ce modèle rend possible la modélisation des systèmes temps réel complexes et en particulier des systèmes workflow.
- Proposition d'un processus de modélisation des systèmes temps réels en général et des applications workflow spécialement, par l'exploitation du modèle défini. Une proposition d'une sous classes du modèle a été définie $RTWF - nets$ permettant de modéliser le *healthcare* workflow
- Proposition d'une approche pour la contraction de l'espace d'état d'un RTPN. Cette contraction dite technique du graphe des classes nécessite pour ce cas la représentation de chaque classe E par une paire (M, D) où D est un système de contraintes impliquant des variables au nombre de transitions sensibilisées pour la classe.
- Proposition d'une approche implémentant la méthode de calcul de l'espace d'état d'un RTPN.

- Proposition d'une technique pour le calcul de la réponse du temps borné (i.e, temps minimal et maximal d'une séquence).
- Finalement, l'exploitation du graphe pour définir les propriétés d'un $RTPN$.

1.5 Organisation du document

La présente thèse est organisée comme suit : La présente introduction constitue le chapitre premier de cette thèse.

Le Chapitre 2 présente un état de l'art sur la modélisation, la spécification et la vérification des systèmes temps réel.

Le Chapitre 3 revient sur les définitions des réseaux de Petri, nous y énumérons ses principales extensions temporelles en soulignant celles étendues à la sémantique de *synchronisation*.

Dans le *Chapitre 4*, nous rappelons les principales méthodes permettant le calcul de l'espace d'états de ces modèles.

Le *Chapitre 5* s'intéresse à une nouvelle sémantique pour les RdPT dédiée à la spécification des applications temps réels complexes. Pour ce faire, nous définissons un nouveau paradigme appelé *Rendez-vous* qui sera décrit en détail tout au long du chapitre. Nous proposons un schéma et une description mathématique compacte générant un grand nombre de règles classées en stratégies et couvrant un large panel de la littérature.

Le *Chapitre 6* présente la syntaxe et la sémantique formelles du modèle proposé, et finit par le comparer avec des modèles existants. Nous montrons aussi comment exploiter ce modèle pour la modélisation des applications workflow avec un cas d'étude.

Le *Chapitre 7*, introduit une approche pour le calcul de l'espace d'état du modèle proposé (RTPN). Un exemple est donné pour expliciter l'approche proposée.

Le *Chapitre 8* montre comment exploiter le graphe obtenu pour l'analyse et la vérification des propriétés qualitatives et quantitatives du système modélisé.

Le *Chapitre 9* est constitué d'une conclusion et des perspectives.

Enfin, une annexe est rajoutée contenant les preuves formelles de nos propres propositions et théorèmes énoncés dans cette thèse.

Chapitre 2

Modélisation, spécification et vérification des systèmes temps réel

2.1 Introduction

Le présent chapitre présente un état de l'art sur les outils de modélisation en général et ceux basés sur le formalisme des réseaux de Petri en particulier. Nous donnons dans la suite l'essentiel pour permettre de vérifier et de spécifier les propriétés, qu'elles soient qualitatives ou quantitatives.

2.2 La modélisation des systèmes temps réel

Les systèmes temps réel constituent un domaine d'exploitation privilégié des méthodes formelles. Cela commence par la proposition d'un *modèle* mathématique capable de décrire fidèlement leurs comportements temporels et s'ensuit la proposition de techniques d'analyse adaptées à ce modèle, permettant de vérifier *qualitativement* et *quantitativement* les propriétés des systèmes modélisés.

Généralement, l'expressivité d'un modèle (nous entendons par là sa capacité à représenter un nombre important de caractéristiques du système) et sa simplicité en termes de vérification (décidabilité et complexité algorithmique) s'opposent. Néanmoins, il est souvent important de choisir un modèle suffisamment expressif, car nous évitons ainsi le pessimisme éventuel d'un modèle surévaluant le comportement réel du système. Ce pessimisme peut en effet avoir deux effets pervers : d'une part, conduire à considérer une propriété fausse sur le modèle alors qu'elle est vraie sur le système réel et, d'autre part, mener à un espace d'états infini alors que celui d'un modèle plus fin serait fini.

De nombreux modèles ont été proposés pour la modélisation, l'analyse et la vérification des systèmes temps réels ; nous pouvons citer : les automates temporisés (TA)[19], Le langage de réécriture (MAUDE)[20], UML[21], les langanges probabilistes/stochastiques tels que les chaînes de MARKOV (MC)[22, 23, 24], les langages ensemblistes tel que (EVENT B)[25], et le langage relationnel tel que (Z)[26, 27]. Tous ces formalismes ont fait leur preuve au fil du temps, néanmoins l'aspect synchronisation et délais entre processus restent non clairement définies et donc, sont inadaptés aux besoins attendus dans cette thèse qui se résument par : modéliser un plus grand schéma de synchronisation (plus large que celui déjà défini dans la littérature) avec des contraintes de délais.

Dans la présente thèse, nous portons un intérêt particulier aux réseaux de Petri et leurs extensions temporelles, un modèle où les mécanismes de bases, tels que (e.g. synchronisation, parallélisme, conflit, concurrence, etc.), sont représentés de manière naturelle. Les bonnes raisons de choisir les réseaux de Petri ont bien été décrites par *Van der Aalst*[28], citons : (i) sa sémantique formelle et sa nature graphique ; (ii) étant un modèle basé sur les états et pas sur les événements ; (iii) son pouvoir d'expression ; ses propriétés ; et enfin (iv) les techniques abondantes d'analyse de ses espaces d'état.

Les modèles temporisés pour lesquelles le temps est manipulé de manière explicite sont bien connus (e.g. les extensions au temps de : l'algèbre de processus, les automates finis (TA) [29] et les réseaux de Petri (RdPT) [17]). Ces derniers permettent de modéliser l'évolution d'un système suite à des actions discrètes ou à l'écoulement continu du temps. Avant de rentrer dans le détail des réseaux de Petri, nous proposons de faire un bref rappel sur les modèles temporels de synchronisation les plus connus et les plus proches de notre travail. Ces derniers sont basés sur des intervalles.

Concatenation $A \circ B$	B follows A
Union $A \cup B$	B follows A and common footage is not repeated
Intersection $A \cap B$	Only common footage of A and B is played
Difference $A - B$	Only footage of A that is not in B is played A
Parallel $A \parallel B$	A and B are concurrent and start simultaneously
Concatenation $A \wr\wr B$	A and B are concurrent and terminate simultaneously
Conditional (test) $?A_1 : A_2 ... A_i$	A is played if (test) evaluates to i
Loop loop A *time*	repetition of A for a duration of *time*
Stretch Stretch A factor	sets the duration equal to factor times duration of A
Limit limit A times	sets the duration equal to the minimum of time
Transition transition AB *type time*	*type* transition effect, *time* : its duration
Contains contains A *query*	a presentation contains A that match *query*

TABLE 2.1 – Modèle de Weiss

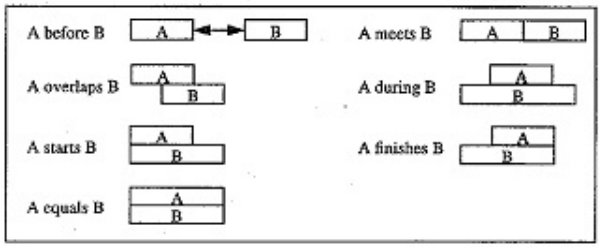

FIGURE 2.1 – Relations temporelles basiques d'Allen.

2.2.1 Les modèles temporels de synchronisation

Dans cette section, nous évoquons les schémas de synchronisation les plus proches de notre modèle décrit un peu plus loin dans cette thèse.

a)- Le modèle d'*Allen* [30] : Dans le modèle d'Allen, deux intervalles peuvent être reliés selon sept relations : : {before, meet, overlap, finish, during, start, equal}, et respectivement, leurs transpositions : {after, met by, overlapped by, finished by, contains, stared by, equal}. La Figure 2.1 représente un modèle réduit des 7 relations non-invertibles (voir Fig.2.1).

b)- Le modèle de *Wahl* [31] : Dans le modèle de Wahl, le nombre de relations est étendu à 29 en considérant un, deux puis trois paramètres de délai comme suit : { before, cobegin, coend, beforeendof}, {while,cross, delayed, startin, endin} and {overlaps}. La Figure 2.2 présente les 10 opérations réduites.

c)- Autres modèles : Généralement ces derniers sont dédiés au multimédia, nous citons le modèle de *Weiss* [32], qui définit quatre catégories d'opérations : creation {create, delay}, output {window,audio}, description {description, $hide - content$} et 12 opérateurs composées présentant des contenus vidéo, tels que : {Concatenation, union, intersection, difference, parallel, parallel-and, conditional, loop, stretch, limit, transition, contains}. Voir la Table 2.2.1 pour plus de détail. Le modèle de *Kermane*[33], quant à lui, décrit 4 catégories de compositions temporelles : sequential {seq, follow}, parallel {par-min,par-max, par-begin, par-end}, equality {equal, ident} et alternative. Les opérateurs sont représentés graphiquement dans la Figure 2.3.

2.2.2 Les réseaux de Petri et le temps...

Les deux principales extensions temporelles des réseaux de Petri sont les réseaux de Petri temporisés [34] et les réseaux de Petri temporels [17], généralement notés par le modèle de *Ramchandani*, respectivement le modèle de *Merlin*. Dans le premier, le temps est représenté par un délai minimal (ou exact, dans le cas d'un fonctionnement « au plus tôt » du réseau) permettant l'occurrence d'un événement. Les réseaux temporisés constituent une classe de réseaux incluse dans les réseaux de Petri temporels. Le temps est intégré au sein de ces derniers sous la forme d'un intervalle contraignant les instants de tir des transitions.

Différents types de réseaux temporels ont été proposés dans la littérature. Ils diffèrent par l'élément auquel est associée la notion du temps : il peut tout aussi bien s'agir des arcs (réseaux de Petri A-temporels

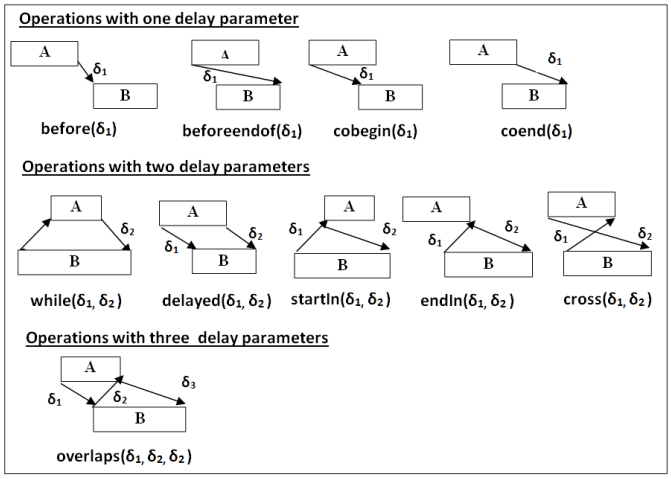

FIGURE 2.2 – Relations temporelles avec délais de Wahl

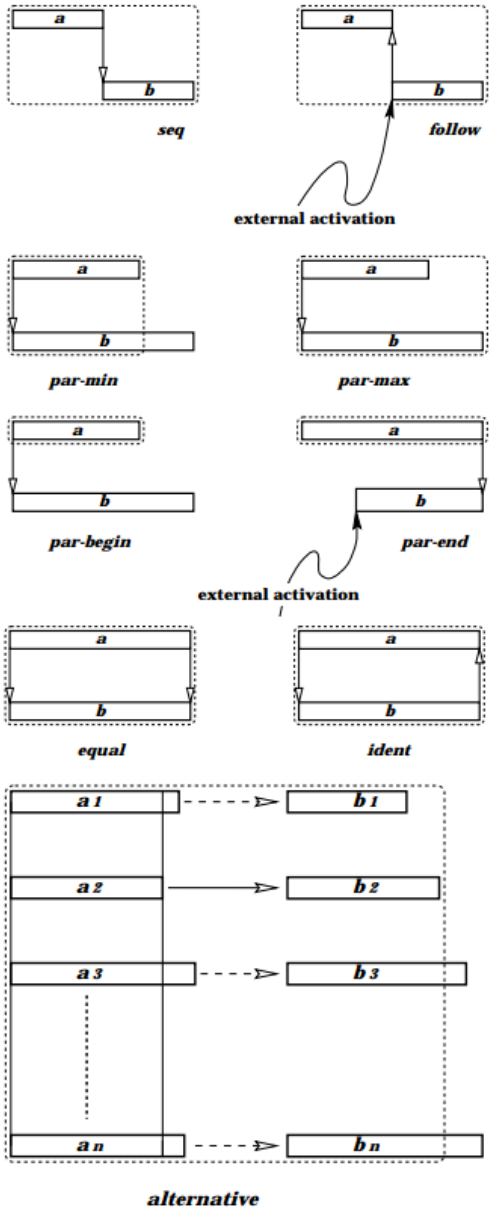

FIGURE 2.3 – Modèle de Kermane

[35] , [36]), des places (réseaux de Petri P-temporels [37]) ou des transitions (réseaux de Petri T-temporels [17]). Une étude intéressante de l'expressivité de ces modèles a été présentée dans [38]. Théoriquement, il s'avère qu'en sémantique forte (i.e. toute transition doit être tirée lorsque la borne temporelle supérieure de sa condition de tir est atteinte), les réseaux T-temporels et P-temporels sont incomparables, tandis que les réseaux A-temporels sont plus expressifs que les réseaux P-temporels et les réseaux T-temporels. En revanche, l'article met en évidence que la synchronisation par *rendezvous* nécessite des motifs complexes lorsqu'on travaille sur des réseaux A ou P-temporels, alors qu'elle est relativement facile à exprimer avec des réseaux T-temporels. C'est pourquoi ce modèle constitue un formalisme particulièrement adapté pour exprimer le parallélisme et la concurrence des applications temps réel.

De ce fait, au cours de cette thèse, nous nous focalisons sur ce modèle, que nous désignons simplement par le terme de réseau de Petri temporel, ou RdPT.

2.2.3 Les réseaux de Petri temporels : Modularité et synchronisation...

Pour gérer la complexité des systèmes temps réels, l'idée de les exprimer comme des assemblages de *composants* est apparue, d'où le nom *compositionnalité*. Lorsqu'on parle de composants, on parle de communication et de synchronisation. Par conséquent, la sémantique d'un assemblage de composants est obtenue depuis celles des composants individuels et des opérateurs d'assemblage.

Une technique d'assemblage des réseaux de Petri temporels et ceux étendus aux arcs inhibiteurs, est discutée dans [39] préservant la compositionnalité. Dans [40], l'auteur considère qu'un réseau de Petri peut être vu comme un ensemble de sous-réseaux, on parle d'un système de réseaux de Petri composés en parallèle (et évoluant en parallèle). Une technique de construction du graphe d'atteignabilité est développée pour ce modèle en mode synchrone (plusieurs transitions sont tirées) ou asynchrone (une transition est tirée). Le graphe obtenu est fini si le modèle est fini et les propriétés classiques peuvent être vérifiées (vivacité, blocages, etc.) ainsi que le calcul du temps maximal et minimale pour une séquence de tir d'un groupe de transitions est présenté.

Les réseaux de Petri communicants ($CmPN$) [41], sont étendus aux temps et à la préemption dans [42]. Les auteurs proposent une technique de construction de l'espace d'état des composants individuellement, de là une approche pour calculer l'espace d'état de leur assemblage est proposée.

Afin de partager des ressources entre un ensemble de systèmes temps réel, une technique de spécification et d'analyse est présentée dans [43]. Un réseau de Petri temporel étendu à la préemption est considéré pour spécifier chaque sous système composant. La synchronisation, la communication, les délais, le partage de ressources, la priorité et d'autres caractéristiques sont également pris en considération par la sémantique du modèle.

Dans [44] [45], les réseaux de Petri orientés objet sont étendus au temps et à la préemption (Real-time synchronised Petri nets). La sémantique du modèle est définie selon un ensemble de règles opérationnelles et structurées décrivant la synchronisation entre objets faisant référence chacun à un réseau de Petri. Dans [46], le formalisme *Time Recursive ECATNets* ($T-RECATNets$), est proposé pour modéliser et analyser les workflow reconfigurables et contraints par le facteur temps.

Plusieurs modèles reposant sur le formalisme des réseaux de Petri ont été proposés dans la littérature pour prendre en charge les contraintes de *synchronisation*. Dans tous ces modèles, une place est associée à la représentation d'une donnée ou ressource (image, séquence vidéo, échantillon de son mémoire, CPU, Machine...etc), alors que la structure du réseau et les transitions expriment les relations de dépendance intra-flux et inter-flux si l'on s'intéresse au domaine du multimédia. Deux objectifs fondamentaux sont recherchés :

- La validation des contraintes temporelles : L'absence d'inconsistance dans la description de la synchronisation spécifiée par l'utilisateur ;
- La vérification de la synchronisation vis-à-vis de la spécification de ces contraintes.

Dans ce cadre, parmi les modèles basés sur les RdPT étendus aux contraintes de synchronisations, nous pouvons citer de la littérature ceux jugés les plus proches de notre travail :

- Les RdPT (le modèle de Merlin) [17] rendent possible l'utilisation de temporisation imparfaite en attribuant à chaque transition un intervalle de temps dans lequel doit se situer le tir de la transition. Le modèle possède un bon pouvoir d'expression équivalent à celui d'une machine de *Turing*. Cependant, il induit une synchronisation forte entre flux, toujours conduite par le flux le plus en retard. Cette considération peut entraîner le non-respect des contraintes temporelles des flux les plus en avance. Le modèle RdPT ne permet donc pas une modélisation aisée d'un scénario de synchronisation inter-flux et présente une limite en termes de pouvoir de modélisation.

— Le modèle OCPN (*Object Composition Petri Net*)[47] étend le formalisme des réseaux de Petri en associant une temporisation aux places du réseau. Le modèle repose sur les sept schémas de synchronisation de base d'*Allen*, qui caractérisent toutes les positions relatives entre deux intervalles temporels [30]. Ces schémas définissent l'ensemble des règles de composition utilisable pour construire des modèles complexes à partir de modèles simples. Cependant, les temporisations des places représentent des durées nominales, et ne permettent de modéliser ni les variations de temps induites par les systèmes synchrones, ni la gigue admissible intrinsèque aux différents vecteurs d'un document multimédia. Le modèle OCPN présente donc une limite en termes de pouvoir d'expression. Le modèle a été étendu vers *XOCPN* (*Extended Object Composition Petri Net*)[48] pour garantir la QdS (Qualité de Service) et *DOCPN* (*Distributed Object Composition Petri Net*)[49] pour modéliser la synchronisation multimédia distribuée (en plus de la priorité et l'interaction des utilisateurs).
— Le modèle ATPN (*Arc Time Petri Net*) [50] comble les lacunes des RdPT en plaçant des intervalles de temps non plus sur les transitions, mais sur les arcs sortants des places. La spécification de la synchronisation inter-flux peut s'exprimer de façon simple par une fusion des transitions. Si les intervalles dynamiques associés aux arcs précédant une transition inter-flux ont une intersection vide, c'est qu'une désynchronisation inadmissible des flux considérés est intervenue, la représentation est alors interrompue. Toutefois, les règles de tir des transitions ne permettent pas de prendre en compte ce type de problème autrement que par une rupture de connexion. C'est donc en termes de pouvoir d'expression que se situe la limite du modèle ATPN.
— Le modèle TSPN (*Time Stream Petri Net*) ou réseaux de Petri à flux [51] a été proposé pour la modélisation des systèmes complexes dits *faiblement synchrones*, comme une extension des RdPT [17] et des OCPN [47]. Ces systèmes englobent un large panel allant des systèmes fortement synchrones aux systèmes asynchrones. Ce modèle offre le pouvoir d'expression des RdPTs et étend le modèle ATPN par l'ajout de règles de tir des transitions inter-flux, lorsque l'intersection des intervalles de validité des arcs entrants est vide. En cela, les TSPN prennent en compte les problèmes d'asynchronisme des systèmes distribués en tirant parti du degré de variation acceptable d'un processus. Concrètement, aux arcs sortants d'un TSPN sont attachés des triplets $[x, n, y]$ appelés intervalles de validité temporelle, où x, n et y désignent respectivement le temps minimal, nominal et maximal du traitement d'un processus. Neuf sémantiques de tir d'une transition inter-flux ont été définies de façon à contrôler précisément des dérives temporelles entre différents flux manipulés. Elles autorisent, par exemple, la spécification de mécanisme de synchronisation conduit par le flux le plus en avance (synchronisation de type Ou-fort), par le flux le plus en retard (synchronisation de type Et-faible), ou par un flux donné (synchronisation de type Maître). En cela, ces règles définissent les intervalles de tir couvrant tous les instants de synchronisation possibles, obtenus par combinaison complète et consistante des intervalles dynamiques de validité temporelle des arcs concernés. En conclusion, le modèle TSPN introduit une taxonomie de la synchronisation multimédia qui autorise la détection des fautes de synchronisation, tout en garantissant la continuité des flux de données par le respect des règles de tir des transitions inter-flux. En cela, le modèle TSPN est un modèle plutôt complet dans l'expression de la synchronisation temporelle d'une application multimédia. Le modèle étendu noté par $HTSPN$ pour (*Hierarchical Time Stream Petri Net*)[52] été proposé pour prendre en considération l'aspect hiérarchique des composants modélisés par une place particulière.
— Le modèle STPTPN (*Synchronizing Transitions Preemptive Time Petri Net*) [1] dédié à la modélisation des exigences multimédia est basé sur le modèle *RdPT* augmenté de mécanismes spécifiques tels que :(la préemption, la priorité et la synchronisation).

Discussion : Le Tableau 2.2 représente une comparaison établie entre les modèles les plus proches (selon nos objectifs) et basés sur les RdP. La comparaison donne une brève description de chaque modèle (la particularité du modèle actuel comparé aux autres), et une colonne qui exprime le pouvoir d'expression et de modélisation des modèles su-citer. Nous remarquons que malgré ce qu'apporte le modèle $TSPN$ comme avantages en matière de synchronisation, il ne permet pas forcément de couvrir tout type de synchronisation existant dans le monde réel (ce que nous verrons un peu plus loin). De plus, les délais entre actions ne peuvent être exprimés au travers des TSPN. Les RdPT même s'ils possèdent un pouvoir d'expression puissant, il sont très limités en matière de synchronisation et de délai entre actions. Le modèle STPTPN ayant un schéma basique couvrant un panel limité de synchronisation tout comme les TSPN et ne permettent pas de prendre en considération les délais. Enfin, nous constatons qu'aucun modèle parmi ceux cités plus haut ne permet de donner un large panel de synchronisation couvrant toutes les possibilités pouvant être rencontrées dans le monde réel. Par ailleurs, aucune considération des délais

TABLE 2.2 – Comparaison des modèles basées *Rdp* les plus proches

Approche	**Brève description**	**Pouvoir d'expression / modélisation**
$OCPN$	Temporisation sur les places, règles de synchronisation d'*Allen*	Modélisation des systèmes asynchrones et gigues entre média non possibles ; Pouvoir d'expression limité.
$ATPN$	Temporisation (intervalles temporels) sur les arcs et une seule règle de synchronisation	Présentation interrompue et une déconnexion si intersection des intervalles des arcs inter-flux vide ; Pouvoir d'expression limité
$TSPN$	Intervalles associées aux arcs et schéma de synchronisation d'*Allen*	L'ajout de règles permet d'éviter une déconnexion en cas d'intersection vide des intervalles des arcs entrants, et autorise des fautes de synchronisation et continuité des flux. Délai entre actions non pris en charge explicitement ; Deux arcs ne peuvent pas partager la même place et donc la même ressource ; Ne couvre pas tout type de synchronisation du monde réel.
TPN	Intervalles temporelles associés aux transition ; Synchronisation par le flux le plus en retard	Fort pouvoir d'expression (*machine de Turing*) ; Inconvénient : le modèle peut conduire au non respect des contraintes temporelles des flux les plus en avance car il induit une synchronisation par le flux le plus en retard ; le vrai parallélisme est non exprimé ; Pas de délai entre actions ; Pouvoir de modélisation limité.
$STPTPN$	Intervalles temporelles associés aux transitions ; Schéma de synchronisation des $TSPN$ sur les transitions et préemption	Inconvénient : schéma de synchronisation limité ; Délai entre actions non exprimé ; Restreint pour le multimédia ; Pouvoir de modélisation limité,

n'est prise en charge entre action, qu'elles soient du même privilège ou non (maître/esclave).

2.2.4 Réseaux de Petri et Workflow...

Les systèmes workflows sont d'un intérêt certain pour notre thèse, car faisant appel à des mécanismes de synchronisation complexes couplés à des contraintes de temps. Nous abordons ce type de systèmes comme modèle d'application au formalisme que nous proposons. Le modèle des réseaux de Petri et ses extensions ont été largement utilisés pour la spécification et l'analyse des systèmes de Workflow. En effet, si certaines extensions géraient l'aspect temporel [53], la modélisation des données [54] ou des structures hiérarchiques [55] d'autres sont proposés, combinant certains ou toute extension des modèles sus-cites [56].

Les auteurs dans [57] considèrent le modèle TSPN pour modéliser les workflow complexes. D'autres auteurs sont allés plus loin en proposant une sous-classe du modèle de base des réseaux de Petri dédiée essentiellement aux workflows, les *Workflow nets* ($WF-nets$)[28]. Dans [58], les $WF-nets$ sont considérés pour la validation et la vérification des *bisness procedures*. Le modèle $R/NT - WFNet$ est introduit dans [59] pour modéliser les workflow avec contraintes de ressources et un temps non déterministe. Une procédure est proposée pour calculer la date au plutôt et la date au plus tard pour commencer une activité. Dans [60], le modèle ($CTAPN$: *component-based timed-arc Petri Nets*) est présenté pour modéliser les *health-care workflow* collaboratifs.

Les auteurs, dans [61], introduisent le modèle $WFCS-nets$ (workflow with critical sections nets), pour traiter la synchronisation et les contraintes temporelles entre activités en section critique. Dans [62, 63], des modèles sont adaptés pour traiter les workflows inter-organisationnels ($IOWF$). Les auteurs, dans [64], utilisent les réseaux de Petri temporels pour modéliser et analyser l'aspect temporel des systèmes de workflow en utilisant TINA comme outil de vérification des deadlines des activités. L'information temporelle localisée dans les transitions des automates dérivés des RdPT permettent de détecter les propriétés temporelles quantitatives préservant la logique $TCTL$. Dans [65], les RdPT colorés ($CTPN$) sont considérés pour représenter les WAM : (*workflow authorization model*) pour désigner les décalages séparant plusieurs instances de workflow. Dans [66], les modèle $LTWN$ (*Logical Time Workflow nets*) et le modèle $ILTWN$ (*Inter-organizational* $LTWNs$) basés sur les réseaux de Petri temporels logiques $LTPN$ (*logical time Petri net*) sont présentés pour spécifier et vérifier les systèmes temps réels coopératifs. Plus encore, la correction logique ainsi que la réponse du temps borné sont étudiées. Dans [67], une approche basée sur la réduction d'ordre partiel décrivant comment l'analyse de propriétés qualitatives et quantitatives peuvent être perfectionnés en utilisant la logique $TCTL$. Finalement, dans [68], les auteurs introduisent le concept de *rendez-vous* pour spécifier les synchronisations complexes dans les systèmes de workflow temporisés. Des stratégies adoptant une variété de modèles de synchronisation sont alors dérivées. Nous portons un intérêt particulier aux travaux de *Van der last* incluant le modèle *Rdp*, pour modéliser et valider les workflows et les Web services grâce à un panorama de patterns présentés dans [69, 58, 28, 55, 70, 62]

2.2.5 Modèles Non basés sur les réseaux de Petri

Dans la littérature, nous trouvons également d'autres modèles formels pour spécifier et vérifier des systèmes de workflow complexes, certains sont basés essentiellement sur les graphes. Dans [71], les tâches associées à leur durée représentent les sommets du graphe, les arcs représentent leurs exécutions. Dans [72] les sommets sont associés aux instants de début et de fin des tâches correspondantes. Dans [8], les auteurs introduisent les graphes orientés où les tâches sont représentées comme dans [72] où la contrôlabilité d'un chemin d'un workflow est vérifiée pour toute combinaison de durée et délais. Dans [73], le (TWG) pour *Timed Workflow Graph* est défini pour représenter les contraintes temporelles entre activités comme les bornes supérieure et inférieure, cependant aucun délai entre activités n'est défini.

D'autres modèles encore ont été définis, pas forcément basés sur des graphes, nous pouvons citer ceux utilisant la logique (CTL) dans [74], le *event algebra* dans [75], *multi-agent theory* dans [76], *UML activity diagrams* dans [77], et *State Charts* [78].

2.3 Analyse et vérification des systèmes temps réel

La vérification des systèmes temps réel complexes passe non seulement par l'étude de propriétés *qualitatives*, telles que celles exprimées par « *les portes du métro se ferment toujours après que la sonnerie*

a retenti » mais aussi d'aspects temporels *quantitatifs* tels que « *les portes du métro se ferment toujours en moins de 3 secondes après que la sonnerie a retenti* ». Le système et les propriétés à vérifier nécessitent alors d'être exprimées dans des formalismes dédiés. Nous décrivons les propriétés qualitatives et quantitatives dans le paragraphe suivant.

2.3.1 Propriétés qualitatives et quantitatives

Selon le type du système à vérifier et les besoins attendus, nous classons les propriétés en deux grandes catégories :

les propriétés qualitatives qui concernent le fonctionnement du système à vérifier, citons :

— Sûreté : exprime '*quelque chose de mauvais ne doit pas se produire*'
— Vivacité : exprime '*quelque chose finira par avoir lieu*'
— Atteignabilité : exprime '*une certaine situation peut être atteinte*'
— Blocage : exprime '*le système ne peut plus évoluer*'. Cette propriété décrit le comportement qui met le système dans un état où il ne peut rien faire.
— Équité : exprime '*quelque chose aura lieu*' ou '*quelque chose n'aura pas lieu*', un nombre infini de fois.

Les propriétés quantitatives : recouvrent des questions de performance comme la durée minimale/maximale d'attente d'un service ou le temps de la réponse bornée.

2.3.2 Et la logique dans tous ça ?

Une fois le système décrit sous la forme d'un modèle (dans notre cas un RdPT), il importe de formaliser les spécifications de sa correction. L'expression d'une propriété peut prendre la forme d'une logique dédiée ou autres (des observateurs). *Amir Pnueli* est le premier fondateur induisant la logique temporelle linéaire LTL(Linear Temporel Logic) en 1977 [79], permettant d'exprimer les comportements linéaires d'un système. Ensuite, la logique temporelle arborescente CTL (Computing Temporal Logic), a été proposée par Clark et Emerson [80], permettant d'exprimer les propriétés sur les arbres d'exécution. Tardivement CTL^* vient en 1986 [81], constituant une classe plus générale des deux dernières. Nous pouvons écrire :$CTL^* = LTL + CTL$. La logique CTL a été étendue ensuite avec la notion du temps donnant naissance à $TCTL = Temps + CTL$ [82] .

Une propriété est vérifiée en explorant l'espace d'état du modèle, totalement ou en partie. Cette exploitation est, cependant, soumise au problème bien connu de *l'explosion combinatoire.*

Nous voyons dans la suite de cette thèse que l'espace d'état peut être représenté en un nombre fini de groupes d'états partageant une ou plusieurs propriétés intéressantes, par exemple, un même marquage pour un RdPT ou une accessibilité par la même séquence de transitions discrètes.

2.3.3 Espace d'états des réseaux de Petri temporels

La méthode classique d'analyse des réseaux de Petri temporels est *le graphe des classes d'états* [83] développée en [83] [2]. Dans cette méthode, l'espace d'états est partitionné selon la relation d'équivalence suivante : deux états sont en relation si et seulement s'ils sont accessibles par une même séquence de transitions discrètes. En particulier, ils partagent le même marquage. Le graphe des classes d'états préserve les marquages et le langage discret du graphe des états accessibles. Il est donc adapté à la vérification de l'accessibilité, de marquage et de propriétés LTL. Par contre, il ne préserve pas la structure de branchement du graphe des états et ne permet pas la détermination des contraintes temporelles quantitatives, ce qui rend impossible la vérification de propriétés CTL* ou TCTL du modèle.

Pour remédier au premier problème (propriétés CTL*), *Yoneda et Ryuba* proposent dans [84] un autre partitionnement de l'espace d'états en *classes atomiques.* Dans une classe atomique, tout état a un successeur dans chacune des classes atomiques obtenues par tir d'une transition. Les classes atomiques de *Yoneda et Ryuba* sont obtenues en découpant les classes d'états (définies de façon différente de [2]), par l'ajout de contraintes linéaires. Les auteurs prouvent que le graphe des classes atomiques permet la vérification de propriétés CTL*. Cependant, cette technique n'est pas applicable aux réseaux pour lesquels les intervalles de temps ne sont pas bornés.

Par ailleurs, *Lilius* a proposé une technique plus améliorée [85] permettant l'utilisation de techniques d'ordre partiel développées pour les systèmes non temporisés. Dans [4], *Berthomieu et Vernadat* proposent une construction alternative des classes d'états atomiques, qui fournit un graphe plus compact. Le graphe obtenu se calcule plus rapidement et est applicable aux réseaux dont les intervalles de temps ne sont pas forcément bornés.

Pour remédier au second problème (propriétés $TCTL$), *Hanifa Boucheneb* et *Rachid Hadjidj* [13, 14] ont proposé une sous-classe de cette logique notée : ($TCTL_{TPN}$), ajoutant un RdPT au RdPT initial pour aider à capturer et relever le temps d'un événement, une exploration à la volée est alors appliquée sur le graphe construit afin de vérifier les propriétés temporelles. D'autres méthodes de contraction de l'espace d'état sont proposées dans [86, 12, 11]. Par ailleurs, *E.Vicario* a proposé dans [87] un algorithme pour le calcul de la réponse du temps borné pour n'importe quelle séquence du graphe. Une fois la séquence de franchissement de transitions identifiée, un algorithme récursif permet de calculer le temps minimum et maximum de la réalisation de cette séquence est proposé.

Une méthode plus efficace en matière de rapidité et de performance est proposées par *Boucheneb and al* dans [88] avec une complexité de $O(n^2)$ au lieu de $O(n^3)$ (n étant le nombre de transitions sensibilisées). Un algorithme est également proposé pour le calcul des temps minimaux et maximaux d'exécution d'une séquence du chemin, sa complexité est de l'ordre de $O(n\text{x}m)$, où n est le nombre de transitions et m la longueur du chemin. Récemment, cette approche a été adaptée pour le calcul de l'espace d'état des $TSPN$ par *Abdelli*, cette dernière permet d'avoir un graphe permettant de vérifier les propriétés LTL du modèle [3].

D'autres techniques d'abstraction de l'espace d'états d'un RdPT ont vu le jour. Ces dernières sont basées sur la notion de *régions géométrique ou zones*. Initialement, ces méthodes ont été proposées pour les automates, ensuite sont adaptées pour les RdPT. Nous ne développons pas plus ces derniers, mais nous en retenons en l'occurrence l'approche basée sur la vérification des propriétés $TCTL$ en établissant une traduction des réseaux de Petri vers les automates temporisés. Les traductions se divisent en deux catégories : d'une part, les *traductions structurelles* d'un réseau de Petri temporel non nécessairement borné à dates de tir au plus tard finies ou infinies vers un automate temporisé [89] ; d'autre part les traductions avec calcul de l'espace d'état qui consiste à obtenir l'espace d'états d'un réseau de Petri temporel sous la forme d'un automate temporisé [90].

C'est dans ce contexte général que se situe cette thèse. Nous y développons une nouvelle technique pour une vérification plus efficace et un nouveau modèle permettant de prendre en compte l'évolution de la complexité des systèmes temps réel.

2.4 Conclusion

Nous avons présenté tout au long de ce présent chapitre un état de l'art sur la modélisation, l'analyse et la vérification des systèmes temps réel. Nous avons donné un intérêt particulier au formalisme des réseaux de Petri étendus à des mécanismes plus complexes tels que la synchronisation et la modularité. Nous allons développer ces derniers modèles dans le chapitre qui suit.

Chapitre 3

Les réseaux de Petri : état de l'art et extensions

3.1 Introduction

Introduits par *Carl Adam Petri* dans sa thèse intitulée : *"Communication avec Automates"* en 1962 [91], les réseaux de Petri *RdP* sont des outils graphiques et mathématiques permettant de modéliser le comportement dynamique des systèmes à événement discrets comme les systèmes manufacturiers, les systèmes de télécommunications, les réseaux de transport, etc.

Si leur représentation graphique permet de visualiser naturellement le parallélisme, la synchronisation, le partage de ressources, les choix (conflit), leur représentation mathématique, quant à elle, permet d'établir les équations d'états, à partir desquelles il est possible d'apprécier les propriétés du modèle et de les comparer avec le comportement du système modélisé. Le modèle de base des réseaux de Petri [92] permet donc d'étudier les propriétés logiques des systèmes, c'est-à-dire les propriétés qui dépendent de l'ordre des événements, mais sans faire référence aux instants où ils occurrent.

De nombreuses extensions ont été apportées au modèle de base des *RdP*. Ces dernières permettent soit de manipuler les données de façon plus concise (réseaux de Petri colorés[93]), soit d'augmenter la puissance de spécification avec la notion du temps (réseaux de Petri temporels [17], réseau de Petri temporisés [34] et réseaux de Petri stochastiques [94]).

Dans notre cas, nous nous intéressons aux extensions temporelles déterministes du modèle, à savoir les réseaux de Petri temporels *RdPT* [17] ou (TPN pour *Time Petri Net* en anglais), souvent appelé le modèle de *Merlin*. Ces extensions ont été établies dans le but de prendre en compte le facteur temps dans l'évolution du modèle.

La première partie de ce chapitre sera consacrée aux réseaux de Petri autonomes ; leur définition et leurs propriétés logiques et dynamiques les plus importantes sont rappelées. Une discussion sur les méthodes pour vérifier et étudier leurs comportements et leurs propriétés sera menée.

La deuxième partie sera consacrée aux extensions temporelles du modèle *RdP*, plus précisément au modèle de Merlin où le temps associé aux transitions prend la forme d'un intervalle. Un rappel des définitions et des règles de fonctionnement de cette extension des *RdP* fera l'objet de cette seconde partie. Enfin, nous terminons le présent chapitre par présenter quelques extensions de *RdPT* élargies à certains mécanismes permettant de capturer des sémantiques plus complexes. Nous nous focalisons notamment sur des modèles introduisant des mécanismes de synchronisation [51, 3][95, 1] .

3.2 Les réseaux de Petri

3.2.1 Présentation informelle

Les Réseaux de Petri (*RdP*) représentent le comportement d'un système en termes de conditions ou ressources (modélisées par les *places*) et d'événements ou actions (modélisés par les *transitions*). Une transition, notée t, est liée par des arcs (avec des poids $w \in \mathrm{N}$) aux places d'entrée p_i (arcs d'entrée $w(p_i, t)$) et aux places de sortie p_j (arcs de sortie $w(t, p_j)$). Une transition *source* est une transition qui ne comporte aucune place d'entrée, une transition *puits* est une transition qui ne comporte aucune place de sortie.

Places d'entrée	Transition	Places de sortie
Pré-condition	événement	post-condition
Donnée d'entrée	traitement	donnée de sortie
Signal d'entrée	processus de signal	signal de sortie
Ressource demandée	tâche ou activité	ressource acquise
Conditions	clause logique	conclusions
Buffers d'entrée	processus	buffers de sortie

TABLE 3.1 – Interprétation des Places /Transitions dans un *RdP* [5]

La véracité d'une condition est exprimée par la présence d'un nombre de jetons égal ou supérieur à $w(p,t)$. Les jetons font référence à des objets physiques, des informations, des structures de données ou des ressources et leur distribution dans les places est appelée *marquage*.

Les transitions sont la base du fonctionnement des *RdP* et donc plusieurs notions leur sont associées. Une transition t est dite *sensibilisée* quand le nombre de jetons dans chaque place d'entrée est supérieur ou égal au poids de l'arc la liant à t. Lors du *tir* ou *franchissement* d'une transition t, $w(p_i,t)$ jetons sont enlevés de chaque place d'entrée pi et $w(t,p_j)$ jetons sont mis dans les places de sortie p_j. A un instant donnée, plusieurs transitions peuvent être sensibilisées. Quelques interprétations typiques des places et transitions dans un réseau de Petri sont représentées dans la Table 3.1.

3.2.2 Présentation formelle

Un réseau de Petri peut être assimilé à un système composé de deux parties distinctes : Une partie statique (structurelle), qui se constitue d'un graphe orienté, comportant :

— Un ensemble fini de places, symbolisées par des cercles.
— Un ensemble fini de transitions, symbolisées par des tirets ou rectangles.
— Un ensemble d'arcs orientés qui assurent la liaison d'une place à une transition ou d'une transition à une place.

Définition 3.2.1. ***(Réseau de Petri)***. *Un RdP est un quadruplet* $Quad = (P,T,Pre,Post)$, *où :*

- P, *est un ensemble fini non vide de places ;*
- T, *est un ensemble fini non vide de transitions ;*
- $Pre : P \times T \longrightarrow \mathbb{N}$, *est l'application d'incidence avant (pré-condition), correspondant aux arcs directs reliant les places aux transitions ;*
- $Post : P \times T \longrightarrow \mathbb{N}$, *est l'application d'incidence arrière (post-condition), correspondant aux arcs directs reliant les transitions aux places ;*

Graphiquement, un arc relie une place p à une transition t(resp. une transition t à une place p) si $Pre(p,t) \neq 0$ (resp. $Post(p,t) \neq 0$).

Remarque 3.2.1. *Le quadruplet* $Quad = (P,T,Pre,Post)$ *est un réseau de Petri sans aucun marquage initial (réseau de Petri non marqué). Un réseau de Petri (ou réseau de Petri marqué) est défini un peu plus loin.*

La Figure 2.1 représente un exemple de réseaux de Petri avec p_1,p_2,p_3 (resp. t_1,t_2,t_3) les places (resp. les transitions) du réseau. p_1, est la seule place d'entrée de la transition t_1 ; p_2 et p_3 sont les places d'entrées de la transition t_3 ; p_2 est aussi place de sortie de la transition t_1 ; p_3 est une place de sortie de la transition t_2 ; t_2 est une transition source et t_3 est une transition puits.

3.2.3 Marquage d'un RdP

Le marquage d'un *RdP* est précisé par la présence à l'intérieur des places d'un nombre fini (positif ou nul), de marques ou de jetons. Une place est donc vide ou marquée. Lorsque la place représente une condition logique (e.g. : machine au repos, . . .), la présence d'un jeton indique que cette condition est vraie ; fausse dans le cas contraire. Lorsque la place représente une ressource (au sens le plus large, e.g. un stock, ...), elle peut contenir plusieurs jetons (e.g. le nombre de pièces en stock). Ainsi, le marquage initial d'un *RdP* correspond à la distribution initiale des jetons dans chacune des places du *RdP*, précisant l'état initial du système retenu pour l'analyse.

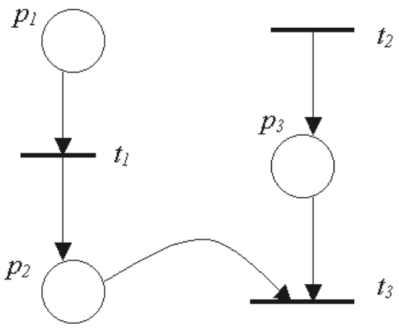

FIGURE 3.1 – Exemple d'un RdP

Définition 3.2.2. ***(Réseau de Petri marqué)****. Un RdP marqué est un couple* $< Quad, M >$*, où :*

- *Quad, est un réseau de Petri (voir plus haut) ;*
- *M, est une application qui associe à chaque place p de Quad un nombre de marques p à* $M(p)$*.* $M : P \rightarrow N$*. On désigne par* $M(p)$ *le nombre de marques (jetons) contenues dans la place p.*

Cette partie statique est complétée par le marquage initial noté M_0. La partie dynamique du réseau consiste alors à faire évoluer ce marquage, c'est ce que nous allons voir dans le paragraphe qui suit.

3.2.4 Règles de fonctionnement

Une transition t est dite sensibilisée (franchissable ou tirable), par le marquage M, ssi : $\forall p \in P$, $M(p) \succeq Pre(p,t)$. Par la suite, nous notons par $E(M)$ l'ensemble des transitions sensibilisées pour M. Le tir (le franchissement) d'une transition t a pour conséquences :

— De retirer $Pre(p,t)$ jetons de chaque places d'entrée p de la transition t.

— De rajouter $Post(p,t)$ jetons à chaque place de sortie p de la même transition t.

Si le marquage avant le tir de la transition t est M son tir conduira donc au marquage M' vérifiant : $\forall p \in P$, $M'(p) = M(p) - Pre(p,t) + Post(p,t)$.

L'ensemble des marquages accessibles en partant du marquage M_0 en franchissant une séquence de transitions S sera noté $Acc(Quad)$. Si $M_n \in Acc(Quad)$, alors il existe au moins une séquence de transitions $S = (t_1, ..t_n)$ franchissable qui permet de passer du marquage M_0 au marquage M_n.

3.2.5 Transitions nouvellement sensibilisées

Soit le tir de la transition t à partir du marquage M pour aboutir au marquage M'. Une transition t' sensibilisée pour M' est dite *nouvellement sensibilisée* pour M' si elle est sensibilisée seulement dans M', ou bien elle est sensibilisée pour M et M' et en conflit avec la transition franchie t pour le marquage M. Nous notons $New(M')$ l'ensemble des transitions nouvellement sensibilisées pour M'. Formellement,

$t' \in New(M')$ ssi,

$$\begin{cases} (t' \in E(M')) \wedge (t' \notin E(M)) & \vee \\ (t' \in E(M) \cap E(M')) \wedge (\exists p \in P, \quad Pre(p,t) + Pre(p,t') \succ M(p)) \end{cases}$$

Autrement, la transition t' est dite *anciennement sensibilisée* ou *persistante*.

3.2.6 Sémantique du parallélisme

A un instant donné, plusieurs transitions peuvent être sensibilisées (pour le même marquage) et donc tirables (franchissables). la *sémantique du parallélisme* répond à la question suivante : *Comment l'état suivant sera t-il définit dans le cas où plusieurs transitions sont sensibilisées en même temps ?*

Plusieurs possibilités existent pour définir le marquage suivant, exemple [51] :

1. Toutes les transitions sont tirées à la fois ;
2. Une seule transition est tirée à la fois ;
3. Un sous ensemble de transitions sensibilisées est tiré ;
4. Tous les sous-ensembles possibles de l'ensemble des transitions sont tirés à la fois.

Il est clair que le premier (1) et le dernier (4) cas sont extrêmes :
— Le premier cas, un seul marquage est obtenu ;
— Le dernier, 2^{n-1} marquages sont obtenus, un pour chaque sous-ensemble (pour chaque transition, puis pour chaque 2 parmi les n, ensuite pour chaque 3 parmi les n, ..., etc.)

Le deuxième (2) cas est le plus important, chaque transition sensibilisée mène à un marquage (avec n marquages suivants pour n transitions sensibilisées). Ce dernier choix est le plus usuel pour définir la *sémantique d'entrelacement* ou *interleaving semantic*.

Dans la *sémantique d'entrelacement*, les marquages suivants sont obtenus en considérant tous les tir possibles de toutes les transitions, une après l'autre, en partant du même état, et menant à n marquages suivants avec n transitions sensibilisées en parallèle. Ceci simplifie l'étude de toutes les propriétés intéressantes du modèle. Cependant, le problème de l'explosion combinatoire du nombre de marquages ne sera pas résolu.

Remarque 3.2.2. *Lorsqu'une transition est validée, cela n'implique pas qu'elle sera immédiatement franchie ; cela ne représente qu'une possibilité de franchissement ou d'évolution du RdP. Pour les RdP il y a un seul franchissement à la fois. Ces remarques impliquent que lorsque plusieurs transitions sont sensibilisées dans un même marquage (les transitions sont franchissables en parallèle), l'ensemble des marquages suivants sera obtenu en considérant toutes les possibilités de franchissement des transitions les unes après les autres. Ainsi, n transitions sensibilisées conduiront à n marquages permettant ainsi d'étudier l'ensemble des comportements possibles. Cependant, ce mode de fonctionnement conduit à une explosion combinatoire des marquages pour les RdP complexes.*

Enfin, le troisième cas constitue un cas général du modèle proposé dans le chapitre suivant (le sous-ensemble des transitions à tirer définit un *Rendezvous*).

3.2.7 Conflit et parallélisme

Conflit structurel : Deux transitions t_1 et t_2 sont en conflit structurel si et seulement si elles ont au-moins une place d'entrée en commun : $\exists p \quad pre(p, t_1).pre(p, t_2) \neq 0$

Conflit effectif : Elles sont en conflit effectif pour un marquage M si et seulement si t_1 et t_2 sont en conflit structurel et que :

- $M \succeq pre(., t_1)$
- $M \succeq pre(., t_2)$
- $pre(., t_1) + pre(., t_2) \succeq M$

Parallélisme structurel : Deux transitions t_1 et t_2 sont parallèle structurellement, si elles n'ont aucune place d'entrée commune (le produit scalaire de leurs vecteurs pre est nul) : $(pre(., t_1))^T \times pre(., t_2) = 0$

Parallélisme effectif : Elles sont en parallèlisme effectif pour un marquage M, si et seulement si t_1 et t_2 sont parallèles structurellement et :

- $M \succeq pre(., t_1)$
- $M \succeq pre(., t_2)$

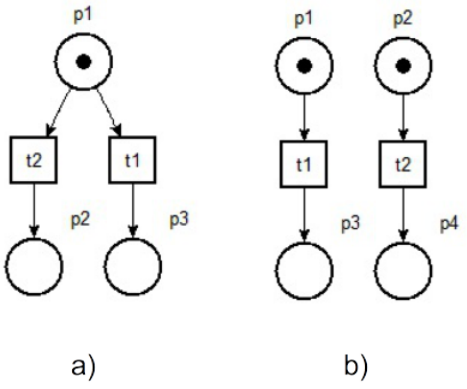

FIGURE 3.2 – Transitions sensibilisées : a) conflit, b) parallélisme

Des exemples de conflit et de parallélisme sont représentés en Figure. 3.2.a) et b) Si le *RdP* est implémenté dans une machine parallèle (multiprocesseur), toutes les transitions parallèles peuvent être tirées au même instant (dit *vrai* parallélisme). Si la machine est séquentielle (mono-processeur), nous implémentons le parallélisme par *entrelacement* : une seule transition est tirée à la fois (voir plus haut).

3.2.8 Co-begin/Co-end et parallélisme

Dans la littérature, on trouve plusieurs structures basiques des RdPs présentant la synchronisation (e.g. Cobegin/Coend, Parallélisme, par signal, par rendez-vous, Exclusion Mutuelle, le mécanisme de lecteurs/écrivains, Buffer, les philosophes, ... ,ect.). Nous nous intéressons plus particulièrement aux trois premiers, pour plus de détail sur les autres synchronisations, le lecteur est invité à consulter les travaux de *Diaz* [96].

Généralement, le terme *Co-begin* représente le commencement en parallèle tandis que le *Co-end* représente la terminaison synchronisée d'un ensemble d'actions ou d'événements.

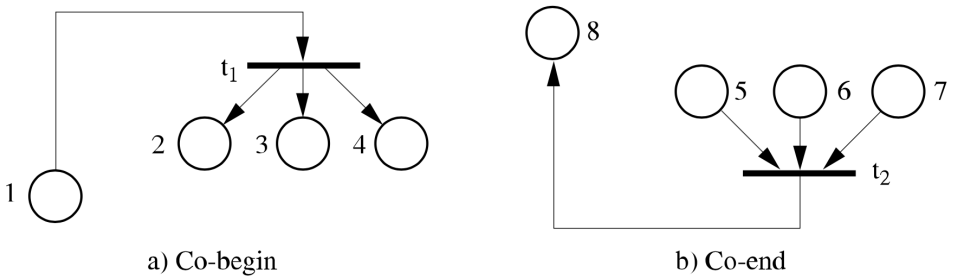

FIGURE 3.3 – Co-begin / Co-end

La Figure 3.3.a) représente le commencement en parallèle et en même temps de trois processus ou actions, e.g. P_a, P_b et P_c. Le modèle indique que ces actions sont lancées par l'état du type *Co-begin*(P_a, P_b, P_c) durant le tir de la transition t_1. Tirer t_1 correspond à l'exécution de l'ordre. Ces actions seront ensuite exécutées lorsque les trois places 2, 3 et 4 sont marquées.

De la même manière, la Figure.3.3.b) définit l'état du type *Co-end*(P_a, P_b, P_c). Le tir de t_2 représente l'évènement qui se déroule à la fin de toutes ces trois actions, les place 5, 6 et 7 respectivement représentent la fin des actions P_a, P_b et P_c.

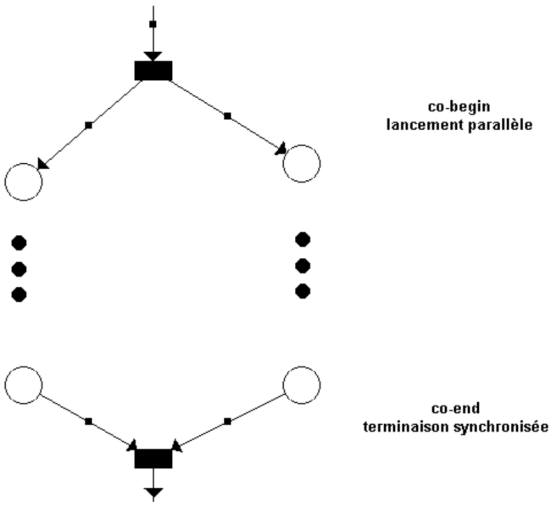

FIGURE 3.4 – Co-begin / Co-end et le parallélisme

3.2.9 Propriétés qualitatives des *RdP*

Les principales propriétés des *RdP* peuvent être classées en deux groupes :

1. Les propriétés structurelles.
2. Les propriétés comportementales ou dynamiques.

Les propriétés structurelles sont indépendantes du marquage initial et sont liées à la topologie du réseau. Leur analyse repose essentiellement sur les techniques d'algèbre linéaire et leur but est de bâtir une passerelle entre la structure du réseau étudié et son comportement. Les propriétés dynamiques, quant à elles dépendent du marquage initial et sont liées à l'évolution de ce dernier. Leur vérification nécessite bien souvent la construction du graphe de marquages accessibles. Elles permettent d'apporter des réponses aux questions concernant l'accessibilité d'un marquage particulier, la bornitude, la vivacité,...,etc.

Bornitude :

Cette propriété permet de caractériser la possibilité pour une place d'accumuler une quantité bornée ou non de marques au cours de l'évolution du réseau.

- Une place p est bornée ou *k-bornée* pour un marquage initial M_0, s'il existe un entier naturel k tel pour tout marquage accessible à partir de M_0, le nombre de marques de p reste inférieur ou égal à k, i.e. : $\exists k \in \mathbb{N}$ / $\forall\ M \in Acc(Quad)$, $M(p) \leq k$.
- Une place p est dite binaire si elle est 1-bornée.
- Un *RdP* est k-borné pour un marquage initial M_0, si toutes les places sont k-bornées pour M_0, et on note : $\exists k \in \mathbb{N}$ / $\forall\ M \in Acc(Quad)$, $\forall\ p \in P$, $M(p) \leq k$.
- Un *RdP* est dit *Sauf*, s'il est *1-borné*.

Vivacité :

Un *RdP* marqué est vivant si et seulement si pour tout marquage accessible M ($M \in Acc(Quad)$), et pour toute transition t, il existe une séquence de franchissement partant de M et contenant t. La vivacité d'un réseau garantit le franchissement de toute transition quelque soit le marquage atteint. La propriété de vivacité est une propriété forte, souvent difficile à vérifier.

Blocage :

Un *RdP* est bloqué si aucune transition n'est tirable pour un marquage donné M ; M est appelé *marquage puits*.

Réseau Réinitialisable :

Un *RdP* est réinitialisable pour un marquage initial M_0, si pour tout marquage accessible M, il existe une séquence de franchissement de transitions telle que l'on puisse accéder à M_0 (M_0 est alors appelé *état d'accueil*).

3.2.10 Graphe de couverture (de marquage)

La vérification des propriétés données ci-dessus se fait généralement en établissant le *graphe de couverture* qui est composé de nœuds qui correspondent aux marquages accessibles, et d'arcs correspondant aux franchissements de la transition qui fait passer d'un marquage à un autre. Le nombre de nœuds dans ce graphe est fini. La construction du graphe de couverture permet de décider si un *RdP* est borné ; on dit que la propriété de réseau borné est décidable. Dans ce cas, le graphe sera appelé le graphe des marquages accessibles et noté $GAcc(Quad, M_0)$. A partir de ce graphe, on peut vérifier toutes les autres propriétés (vivacité, accessibilité d'un marquage, ...). Cependant, pour un *RdP* non borné, établir l'arbre de couverture ne suffit pas pour résoudre le problème d'accessibilité et le problème de vivacité. Ces deux problèmes sont également décidables, mais par des algorithmes plus compliqués. Malgré ceci, le graphe de couverture reste le seul outil pour vérifier les propriétés dynamiques d'un *RdP* pour un marquage initial donné. Cette approche d'analyse est générale, elle est applicable à toute classe de réseaux. Mais, ses limites sont liées à l'explosion combinatoire du nombre d'états. De plus, à partir de ce graphe nous pouvons savoir s'il existe des fonctionnements spécifiques comme par exemple l'existence d'un fonctionnement répétitif stationnaire. L'approche qui est basée sur le graphe de couverture pour étudier les *RdP* constitue l'analyse énumérative.

Les graphes de marquages des RdPs de la Figure 3.2.a) et la Figure 3.2.b) sont représentés respectivement sur la Figure 3.5.a) et la Figure3.5.b).

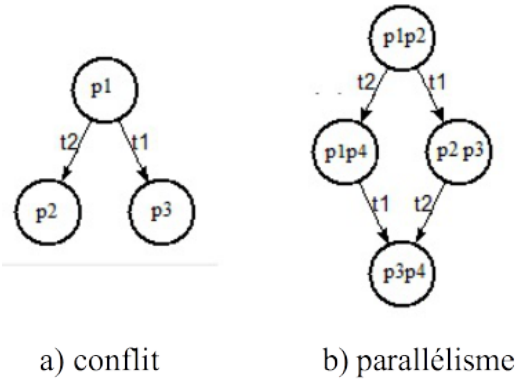

FIGURE 3.5 – Graphe de marquage : a) Conflit, b) Parallélisme

Exemple :

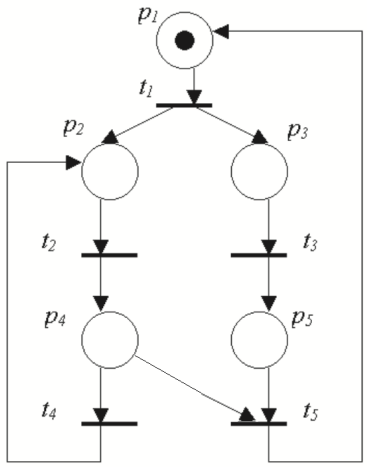

FIGURE 3.6 – Evolution d'un Rdp [1]

La Figure 3.6 représente un RdP $Quad = (P, T, Pre, Post, M_0)$ tel que : $P = p_1, p_2, p_3, p_4, p_5$, $T = t_1, t_2, t_3, t_4, t_5$, $n = |T| = 5$, $m = |P| = 5$. Le marquage initial M_0 est donné par le vecteur $M_0 = [1, 0, 0, 0, 0]$, où seule t_1 est sensibilisée par M_0.

- Le franchissement de la transition t_1 à partir de M_0 conduit au marquage M_1, on note : $M_0 \xrightarrow{t_1} M_1$ avec $M_1 = [0, 1, 1, 0, 0]$, où t_2, t_3 sont sensibilisés par M_1.
- Le franchissement de la transition t_2 à partir de M_1 conduit au marquage M_2, on note :$M_1 \xrightarrow{t_2} M_2$ avec $M_2 = [0, 0, 1, 1, 0]$, où t_3, t_4 sont sensibilisées par M_2.
- Le franchissement de la transition t_3 à partir de M_1 conduit au marquage M_3, on note :$M_1 \xrightarrow{t_3} M_3$ avec $M_3 = [0, 1, 0, 0, 1]$, où t_2 est sensibilisée par M_3.
- Le franchissement de la transition t_3 à partir de M_2 conduit au marquage M_4, on note : $M_2 \xrightarrow{t_3} M_4$ avec $M_4 = [0, 0, 0, 1, 1]$, où t_4, t_5 sont sensibilisées par M_4, et ainsi de suite,...

L'ensemble des marquages accessibles obtenu pour le RdP de la Figure 3.6 est donné par : $Acc(Quad) = \{M_0, M_1, M_2, M_3, M_4\}$ avec, $M_0 = [1, 0, 0, 0, 0]$, $M_1 = [0, 1, 1, 0, 0]$, $M_2 = [0, 0, 1, 1, 0]$, $M_3 = [0, 1, 0, 0, 1]$, $M_4 = [0, 0, 0, 1, 1]$. Le graphe de couverture est illustré par la Figure 3.7.

3.3 Extensions temporisées des RdP

Plusieurs extensions ont été apportées aux RdPs pour pouvoir représenter implicitement le temps. En associant le temps aux places et/ou transitions et/ou arcs. Nous considérons ici le cas où le temps est

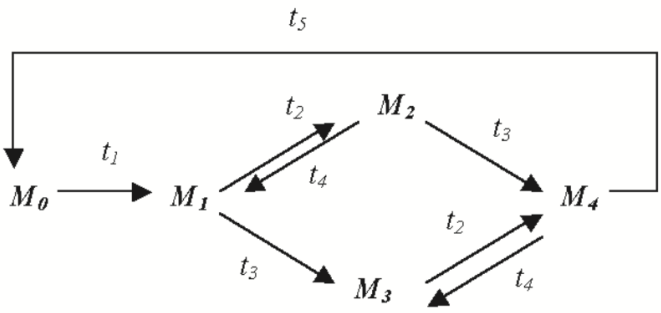

FIGURE 3.7 – Graphe de couverture

associé à chaque transition sous la forme d'un intervalle de temps compris entre une valeur : $x(t)$ et une valeur $y(t)$ qui représentent respectivement la borne minimum et la borne maximum du domaine du tir où la transition sensibilisée peut être tirée, le modèle est appelé réseau de Petri temporel ou modèle de Merlin ($RdPT$) [17].

3.3.1 Les réseaux de Petri t-Temporels : *RdPT*

Les réseaux de Petri temporels $RdPT$ (TPN pour *Time Petri Nets* en anglais) [17] sont une extension des RdPs introduit par Merlin en 1974. Le modèle de Merlin a été conçu pour l'étude des problèmes de recouvrement pour les protocoles de communication. Dans ce modèle, à chaque transition est associée une contrainte temporelle de type intervalle. L'intervalle associé à la transition t est relatif au moment où la transition devient sensibilisée. Soit $[EFT(t), LFT(t)]$. Supposons que t soit validée a l'instant θ, alors t peut être tirée seulement entre $EFT(t) + \theta$ et $LFT(t) + \theta$, sauf si elle est désensibilisée par le tir d'une autre transition avec laquelle elle était en conflit.

Il existe également deux sémantiques usuelles pour les réseaux de Petri temporels : la sémantique dite *forte* et celle dite *faible*. Dans le cas de la sémantique faible, les contraintes de temps constituent des instants possibles d'occurrences d'événements (autrement dit, les bornes temporelles supérieures peuvent être dépassées) alors que pour la sémantique forte, l'événement doit nécessairement se produire dans l'intervalle donné.

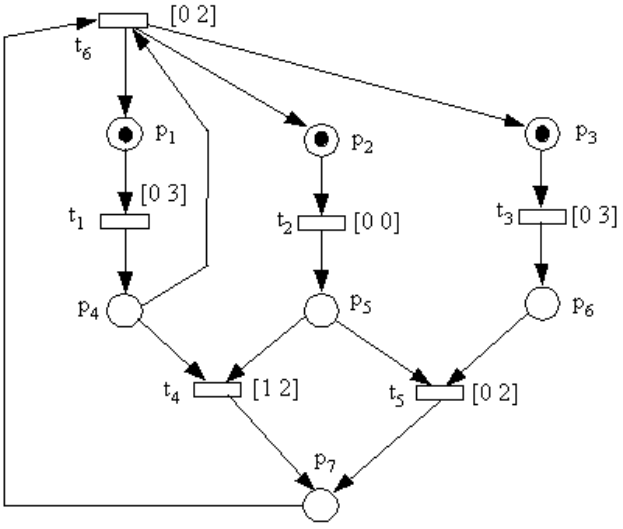

FIGURE 3.8 – Un réseau de Petri T-temporel

Considérons le RdPT de la Figure 3.8. Les transitions t_1, t_2 et t_3 sont sensibilisées au sens classique des RdPs.

Lors du franchissement de t_2 à une date comprise entre 0 et 0 (c-à-dire tout de suite), le jeton de la place p_2 est consommé et un jeton est généré dans la place p_5. Les transitions t_4 et t_5 ne sont pas encore sensibilisées car les places p_4 et p_6 ne contiennent toujours pas de jetons.

Nous allons maintenant définir plus formellement *RdPT*.

Définition 3.3.1. ***(Réseau de Petri t-Temporel)***. *Un RdPT est un doublet* $QuadT = (Quad, FI)$ *où :*

- *Quad est un RdP marqué ;*
- $FI : T \longrightarrow \mathbb{Q}^+ \times \mathbb{Q}^+ \cup \{\infty\}$, *est la fonction intervalle statique ou* $\mathbb{Q}^+$ *représente l'ensemble des nombres rationnels non négatifs. L'application* FI *associe à chaque transition* t *du réseau son intervalle statique de franchissement* $FI(t) = [EFT(t), LFT(t)]$ *tel que :*
  1. $EFT(t) \in \mathbb{Q}^+$,$LFT(t) \in \mathbb{Q}^+ \cup \{\infty\}$ *et* $0 \preceq LFT(t) \preceq LFT(t)$.
  2. $EFT(t)$ *(resp.* $LFT(t)$ *) représente l'instant de tir au plus tôt (resp. au plus tard) de la transition* t.
  3. *Une transition* t *doit être sensibilisée et atteindre son délai minimum* $EFT(t)$ *avant de pouvoir être tirée (franchie) et ne peut le rester au-delà du délai maximum* $LFT(t)$ *sans être tirée. Le tir d'une transition est de durée nulle.*

Remarque 3.3.1. *Soit* $QuadT := (Quad, FI)$ *un RdPT , la sémantique forte impose dans un RdPT, lorsque plusieurs transitions sont franchissables, de franchir l'une de ces transitions avant la fin du domaine de tir des autres transitions. Pour plus de détail sur le fonctionnement du modèle, la sémantique formelle d'un RdPT est formulée dans la section suivante.*

Règle de fonctionnement

A partir de l'instant initial, le modèle séjourne dans son marquage initial jusqu'au franchissement d'une transition, ce franchissement conduit vers un nouveau marquage (i.e., les jetons utilisés disparaissent et ceux produits apparaissent). Le modèle va ensuite séjourner dans le nouveau marquage jusqu'au prochain franchissement et ainsi de suite. Donc le modèle évolue et son évolution est due :

- soit à la progression du temps,
- soit au franchissement d'une transition.

Soit la Figure 3.9 présentant un RdPT et $S = 1; t_2; 1; t_4$ une séquence de transition de sa sémantique. L'ensemble des marquages accessibles du réseau est : $\{(1,1,0,1);(1;0;1;1),(1;0;1;0),(1;0;0;1),(1;0;0;0),(0;0;0;0)\}$. Soit $\overline{t}_i$ la valuation de l'horloge correspondant à la transition t_i (au départ toutes les horloges correspondantes aux transitions sensibilisées sont à 0). Le vecteur représentant l'état du RdPT (i.e. son marquage et les valuations des différentes horloges) est :

$(P_1; P_2; P_3; P_4)$
$\overline{t_1}$
$\overline{t_2}$
$\overline{t_3}$
$\overline{t_4}$

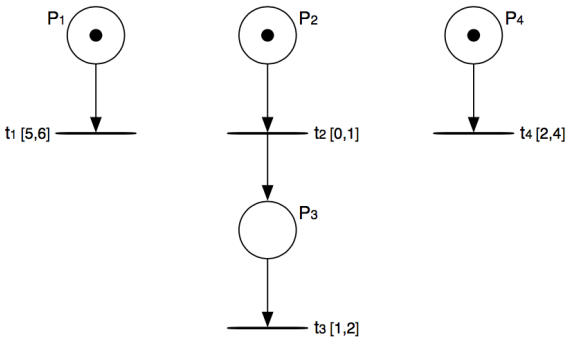

FIGURE 3.9 – Evolution d'un *RdpT*

x signifie que la transition associée n'est pas sensibilisée : la valuation de l'horloge n'a pas d'influence sur le comportement du réseau.

Remarque 3.3.2. *Dans ce qui suit, nous excluons le cas de la multi-sensibilisation des transitions*[1] *(nous considérons des réseaux modélisant les systèmes mono-serveur). i.e pour tout marquage acces-*

1. Ceci permet de modéliser les systèmes multiserveur ou multiprocesseur.

(1 ;1 ;0 ;1)		(1 ;1 ;0 ;1)		(1 ;0 ;1 ;1)		(1 ;0 ;1 ;1)		(1 ;0 ;1 ;0)
0		1		1		2		2
0	$\xrightarrow{1}$	1	$\xrightarrow{t_2}$	x	$\xrightarrow{1}$	x	$\xrightarrow{t_4}$	x
x		x		0		1		1
0		1		1		2		x

sible M, aucune transition du réseau ne peut être sensibilisée simultanément plus d'une fois : ($\forall t \in T, 2Pre(p,t) \succ M$). *Cette hypothèse peut être levée en considérant les transitions multi-sensibilisées pour un même marquage comme étant des transitions différentes (chaque sensibilisation de la transition a sa propre identification et ses propres paramètres temporels).*

Ensuite, la multi-sensibilisation peut être traitée selon les stratégies dites « non déterministe » ou « première sensibilisée première tirée » [97, 98] . Dans la première stratégie, les transitions multi-sensibilisées sont considérées comme indépendantes (choix aléatoires de celle à franchir et de celle à désensibiliser en cas de conflit). Par contre, dans la seconde stratégie, les transitions multi-sensibilisées sont ordonnées de la plus récente à la plus vieille. La plus vieille transition est franchie en premier. En cas de conflit, par exemple, la transition la plus récente est désensibilisée.

Le comportement d'un *RdPT* peut être caractérisé par la notion d'état. De manière intuitive un état sera représenté par un couple composé :

— du marquage courant ;

— de l'intervalle de tir pour chaque transition sensibilisée (au sens des *RdP* classiques).

De manière plus formelle, nous avons la définition suivante.

Définition 3.3.2. ***(Un état)****. L'état d'un RdPT est défini par une paire $e = (M, I)$, telle que :*

— *M est l'application marquage, assignant à chaque place p du réseau un certain nombre de marques ;*

— *I est l'application intervalle dynamique de tir, associant à chaque transition sensibilisée du réseau, l'intervalle de temps dynamique dans lequel elle peut être tirée : $I : E(M) \rightarrow \mathbb{Q}^+ \times (\mathbb{Q}^+ \cup \{\infty\}), I(t) = [EFT_e(t), LFT_e(t)]$*

Remarque 3.3.3. *Les intervalles de tir dans l'état courant e peuvent différer de ceux assignés initialement (les intervalles statiques) aux transitions du réseau. Pour cela ils seront appelés intervalles dynamiques. Mais nous avons toujours $EFT_e(t) \preceq EFT(t)$ et $LFT_e(t) \preceq LFT(t)$.*

Condition de Franchissement

La définition de l'état nous donne la possibilité de déterminer les conditions de tir des transitions depuis un état. Sachant que l'origine du temps pour θ est l'instant où l'état e a été atteint, une transition t est tirable à un instant θ depuis un état $e = (M, I)$ si et seulement si les deux conditions suivantes sont satisfaites :

- La transition t est sensibilisée par le marquage M au sens des RdP classiques, $\forall p \in P, M(p) \succeq Pre(p,t)$;

- θ est compris entre la date de tir au plus tôt de t et la plus petite des dates de tir au plus tard des autres transitions sensibilisées dans l'état e : $(EFTe(t) \preceq \theta \preceq LFTe(t) \wedge \forall t' \in E(M), \theta \preceq LFTe(t')$.

A partir de la notion d'état et des conditions de tir d'une transition depuis un état, l'état suivant peut être calculé.

L'état suivant

Le tir d'une transition t franchissable depuis un état $e = (M, I)$ à un instant θ (θ étant relatif à l'instant d'apparition de l'état e), conduit au nouvel état $e' = (M', I')$, déterminé par :

— Le nouveau marquage $M' : \forall p \in P, M'(p) = M(p) - Pre(p,t) + Post(p,t)$;

— Les nouveaux intervalles de tir I' : pour toutes les transitions t' sensibilisées par le marquage M', nous distinguons :

1. Pour toutes les transitions persistantes ; t' est sensibilisée par les marquages M' et M et non en conflit avec la transition t : $I'(t') = [MAX(0, EFT_e(t') - \theta), LFT_e(t') - \theta]$, avec $I(t') = [EFTe(t'), LFTe(t')]$
2. les intervalles associés aux autres transitions (transitions nouvellement sensibilisées) sont les intervalles statiques, $I'(t') = [EFT(t'), LFT(t')]$.

Les ensembles d'états peuvent être représentés par des paires (M, D), dans lesquelles M est un marquage et D un ensemble de vecteurs de dates appelé domaine de tir. La i-éme projection de l'espace D est l'intervalle de tir $I(t_i)$ associé à la i-éme transition sensibilisée. Les domaines de tir seront décrits par des systèmes d'inéquations linéaires avec une variable par transition sensibilisée (notée comme les transitions). Cette approche est décrite dans le chapitre suivant.

3.4 Autres extensions...

Dans le but d'étendre les *RdPT* aux contraintes de synchronisation et de modularité, plusieurs modèles ont vu le jour. Nous proposons un bref rappel des deux extensions les plus proches de notre travail, les réseaux de Petri à flux ou $TSPN$ et l'extension des *RdpT* pour la modularité.

3.4.1 Réseau de Petri à Flux ($TSPN$)

Les réseaux de Petri à Flux ou (*Time Stream Petri Net*) [51] ont été introduits pour la modélisation des contraintes de synchronisation dans les systèmes faiblement synchrones. Cette notion de faiblement synchrone offre un cadre générique qui couvre une grande palette de systèmes allant des systèmes synchrones aux systèmes purement asynchrones. Ce pouvoir d'expression des réseaux à flux a été exploité notamment dans la spécification des systèmes multimédia. Ce modèle étend les Timed Link Petri Nets [50] aux sept schémas de synchronisations définis dans $OCPN$ (*Object Composition Petri Net*) [47] tels que, les processus sont représentés comme des places, et leurs caractéristiques temporelles sont données par des intervalles associées aux arcs. Ces intervalles appelés intervalles de validité temporelle (IVT), sont définis comme un triplet $[x, n, y]$, où x, n et y sont respectivement, *la durée minimale*, *nominale* et *maximale* admissibles relatives au processus. Il y a trois stratégies fondamentales de synchronisation dans un $TSPN$ impliquant neuf règles obtenues à partir d'une combinaison consistante et complète des intervalles temporels de validité absolus des arcs associés à une transition marquée :

- Stratégies de synchronisation dynamiques $\{'And', 'WeakAnd', 'And - Master\}$ régies par le dernier processus, (i.e., Le dernier arc qui atteint la borne minimum de son IVT, autorise le franchissement de la transition).
- Stratégies de synchronisation dynamiques, $\{'Or', 'StrongOr', 'OrMaster'\}$ régies par le premier processus, (i.e., le premier arc qui atteint la borne minimum de son IVT autorise le franchissement de la transition).
- Stratégies de synchronisation statiques $\{'Master', 'StrongMaster', 'Weak - Master'\}$ régies par un processus sélectif, (i.e., la transition peut être franchie seulement si son arc maître ait atteint la borne minimale de son IVT).

En conclusion, les $TSPN$ offrent neuf stratégies de synchronisation (voir Figure 3.10) qui peuvent être associées à une transition, introduites grâce à la fonction $Syn(t)$. En particulier, il existe deux stratégies pour régir respectivement une synchronisation par le flux au plus tard et au plus tôt. Enfin, la synchronisation peut être régie par le flux maître dont les caractéristiques sont prédéfinies statiquement dans le modèle.

Fonctionnement

Soit $A_{in}(t_i)$, l'ensemble des arcs entrants de la transition t_i. On désigne par la notation a_{ij} l'arc (p_i, t_j). Supposons $\tau(a_{ij})$ la date absolue de la sensibilisation de l'arc a_{ij}, i.e, l'instant où le nombre de jetons dans la place p_i satisfait la pré-condition de l'arc a_{ij}. Par conséquent, l'intervalle $[x(a_{ij}), y(a_{ij})]$ associé à l'arc a_{ij} dénote l'intervalle des dates relatives (à l'instant $\tau(a_{ij})$), durant lesquelles les jetons dans la place p_i pourront être consommés lors du tir de la transition t_j. L'arc a_{ij} modélise ici un processus autonome dont les contraintes temporelles sont spécifiées par son intervalle de validité. La modélisation d'une synchronisation entre n processus (arcs) (i.e, a_{ij} $i = 1..n$), est réalisée par la transition t_j . La synchronisation est possible lorsque la transition t_j est sensibilisée (i.e, lorsque tous ses arcs le sont) et sa date de sensibilisation notée $\tau(t_j)$ est donnée par $\underset{\forall a_{ij} \in A_{in}(tj)}{MAX}\{\tau(a_{ij})\}$. Par exemple, une évolution synchrone de tous les processus (Stratégie And) exigerait de consommer tous les jetons en respectant les contraintes des n processus. Plus formellement, t_j peut être tirée à l'instant relatif δ, si :
$\tau(t_j) + \underset{\forall a_{ij} \in A_{in}(tj)}{MAX}\{x(a_{ij})\} \preceq \tau(t_j) + \delta \preceq \tau(t_j) + \underset{\forall a_{ij} \in A_{in}(tj)}{MIN}\{y(a_{ij})\}$.

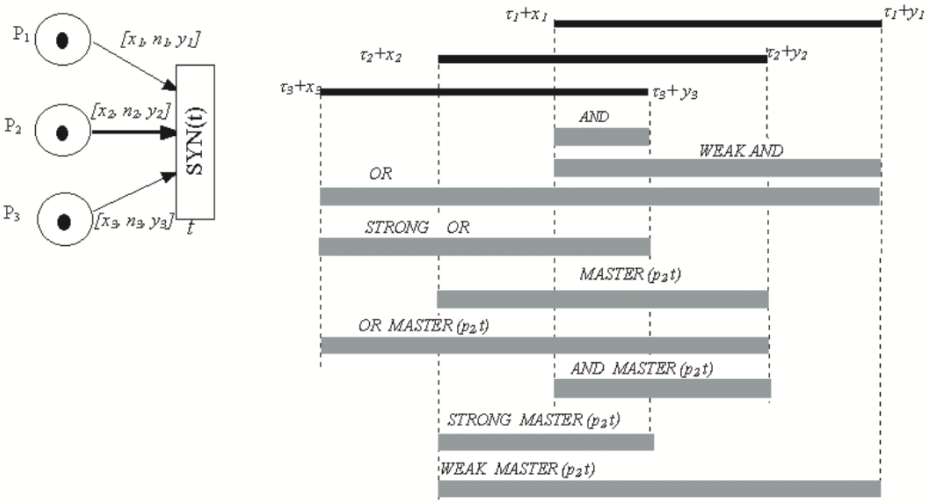

FIGURE 3.10 – Synchronisations dans un Réseau de Petri à Flux

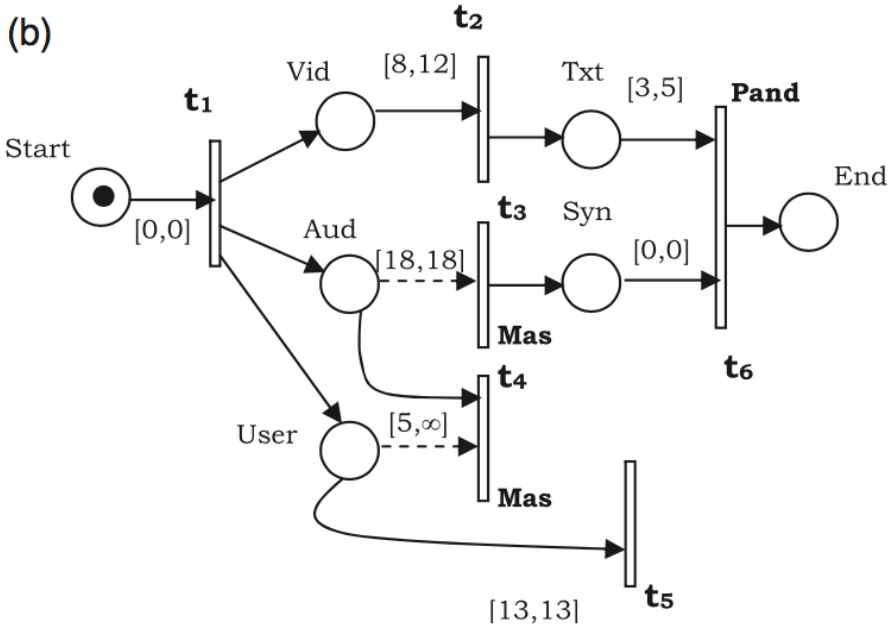

FIGURE 3.11 – Exemple d'un $TSPN$

Une évolution asynchrone (Stratégie Or) exigerait qu'au moins l'un des processus ait ses contraintes satisfaites. Donc, la transition t_j peut être tirée à l'instant δ, si : $\tau(t_j) + \underset{\forall a_{ij} \in A_{in}(tj)}{MIN}\{x(a_{ij})\} \preceq \tau(t_j) + \delta \preceq \tau(t_j) + \underset{\forall a_{ij} \in A_{in}(tj)}{MAX}\{y(a_{ij})\}$.

Par ailleurs, une évolution contrôlée (Stratégie Master) exigerait que le processus sélectionné (appelé *Maître ou Master*) ait ses contraintes satisfaites. Donc si a_{mj} est l'arc représentant le processus maître, la transition t_j peut être tirée à l'instant δ, si : $\tau(t_j)+x(a_{mj}) \preceq \tau(t_j)+\delta \preceq \tau(t_j)+y(a_{mj})$. De même, les autres schémas de synchronisation permettent de définir des variantes des trois stratégies décrites précédemment, permettant ainsi de modéliser une grande partie des relations de synchronisations introduites par l'algèbre d'*Allen* [30].

La Figure 3.11 représente un exemple du modèle $TSPN$.

Le modèle $TSPN$ est souvent utilisé pour modéliser les objets multimedia, étant un outil puissant en matière de synchronisation et de validation des contraintes temporelles (absence d'inconsistance) .

3.4.2 Réseaux de Petri modulaires

La *modularité* est l'une des approches qui permet de répondre au problème de l'explosion combinatoire de l'espace d'états des systèmes temps réels complexes. Cette approche permet de représenter et de vérifier des systèmes de grande taille sous forme de modules.

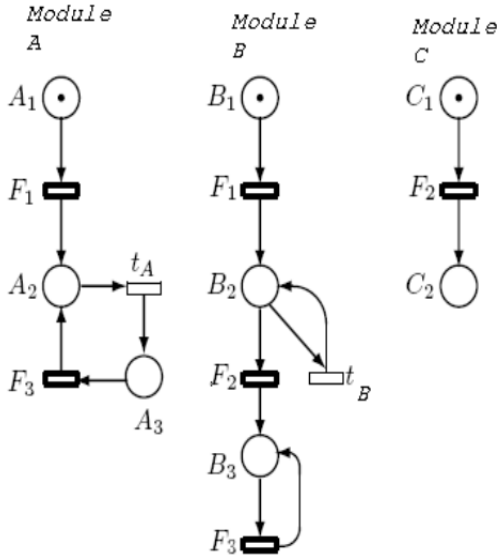

FIGURE 3.12 – Exemple de réseau de Petri modulaire

La Figure 3.12 illustre un réseau de Petri modulaire qui consiste en trois modules A, B et C. Les modules A et B comportent tous les deux les transitions étiquetés dans l'ensemble { F1, F3 }, tandis que B et C ont en commun la transition F2. Ces transitions sont alors considérés comme étant des transitions de fusion et forment l'ensemble des transitions de fusion. Dans la suite, nous allons montrer la problématique de compositionnalité, dans notre cas , on parle de *rendez-vous* entre des modules de RdPT. Nous allons voir que cette propriété n'est pas toujours évidente à conserver.

Considérons les deux *RdpT* de la Figure 3.13, présentés initialement dans [39]. Les deux réseaux sont identiques sauf en ce qui concerne les noms des places et transitions. Nous voulons composer les deux en une action A, qui sera visible en composant C_1 (voir Figure 3.13.a) lorsque t_{12} est tirée et lorsque t_{21} est tirée pour C_2(voir Figure 3.13.b). Selon la définition donnée dans [39] les deux réseaux sont non composables car l'action A ne peut être réalisée dans temps puisque t_{12} et t_{21} ne sont pas sensibilisés et franchissables pour les mêmes instants ; Et donc leur produit ne peut être défini.

En plus, l'idée de prendre l'intersection des deux intervalles des transitions convoquées dans le rendez-vous présente deux problèmes majeurs : le premier est que l'intersection peut être vide, cela se traduit par une transition *jamais tirable*. Le deuxième est que le produit n'est pas composable (les caractéristiques d'un composant individuel une fois composé est une restriction de ses caractéristiques avant composition),

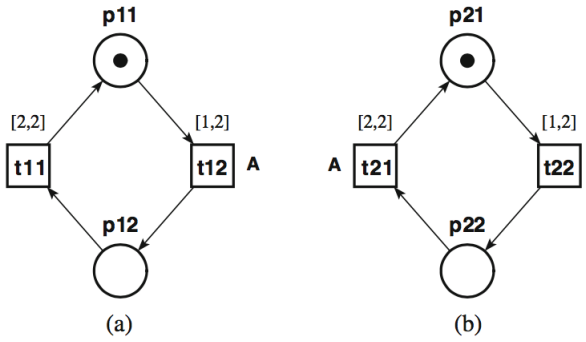

FIGURE 3.13 – Exemple de deux composants (a) : Composant C_1, (b) : Composant C_2

ou (chaque séquence d'exécution dans le système composé est la synchronisation de deux séquences appartenant respectivement aux deux composants (pris séparément)). Cela explique en particulier que la propriété de sûreté des composants est préservée par composition.

Ce qui se passe c'est que cette extension de produit n'obéit pas à cette propriété. Pour illustrer ceci, construisons le produit des deux réseaux C_1 et C_2 en utilisant le produit modifié. Le résultat du produit est illustré dans la Figure 3.14 ; transition t_3 est le résultat de la fusion de t_{12} de C_1 et la transition t_{21} de C_2 ($t_3 = t_{12} \oplus t_{21}$) laquelle est assignée l'intersection des deux intervalles $[1, 2]$ et $[2, 2]$, qui vaut [2, 2]. Considérons la séquence temporelle de transition $\sigma = 2; t_{22}; 2; t_3$ du réseau C_{1x2}. En projetant cette séquence dans le réseau C_1 nous obtenons $\sigma_1 = 2; 2; t_3$, ou $4; t_{12}$ alors le temps de passage de transition peut être rajouté et $t_3 = t_{12} \oplus t_{21}$. De la même manière, sa projection sur le sous réseaux C_2 est $\sigma_2 = 2; t_{22}; 2; t_{21}$. En clair, σ_2 appartient au comportement de C_2 (avant composition), mais σ_1 n'appartient pas au comportement de C_1 , car dans C_1 (pris séparément), pas plus de 2 unités de temps peut passer avant que la transition t_{12} ne soit tirée. Donc ce produit est faux sémantiquement.

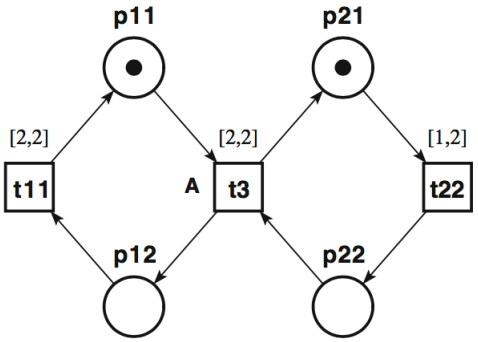

FIGURE 3.14 – Composants NON-Composables $C_{1x2} = C_1 \parallel C_2$

Dans le cas non temporisé, la composition des deux éléments présentent un *rendez-vous*. Après l'ajout de l'information temporelle, nous voulons nous attendre à ce que ce rendez-vous ait lieu dans un intervalle de temps spécifié localement dans les composants. Si nous prenons l'exemple précédent, si nous considérons l'exécution concurrente entre les deux composants, le temps sera vide : t_{12} et t_{21} ne pourront jamais être tirées en même temps, et donc t_3 ne pourra jamais être tirée. Cependant, l'intervalle de tir dans un *RdpT* est relative au temps à partir duquel la transition a été sensibilisée la dernière fois. Donc, après composition, le temps durant lequel les transitions synchronisées peuvent être tirées ne correspond pas au temps local des transitions à synchroniser.

Le modèle ($STPTPN$)

Un exemple de modèle modulaire basé sur les RdPT est le $STPTPN$ pour (*Synchronizing Transitions Preemptive Time Petri Net*) [95]. Ce denier a été défini pour la modélisation des exigences multimédias augmentant les *RdPT* avec entre autres les concepts de modularité, de prémption et de synchronisation.

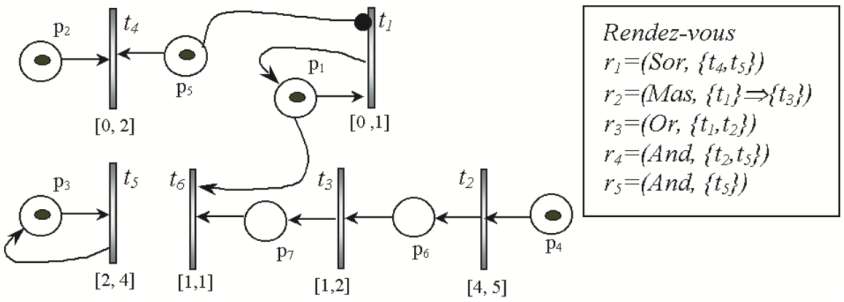

FIGURE 3.15 – Exemple d'un $STPTPN$ [1]

Fonctionnement

Un $STPTPN$ progresse à chaque pas par le franchissement d'un rendez-vous ; dénotant l'occurrence d'une synchronisation d'événements. Par conséquent, l'occurrence d'un rendez-vous est conditionnée par sa règle de synchronisation. Cette dernière implique le tir de toutes ses transitions, prenant en compte leurs contraintes de temps et la disponibilité des ressources nécessaires pour déterminer quels événements doivent être générés suite à la réalisation du rendez-vous. Un $STPTPN$ reprend les mêmes règles de synchronisation définies dans un $TSPN$ mais en les associant cette fois ci aux rendez-vous décrivant les schémas de synchronisations entre modules.

La Figure 3.15 représente un exemple du modèle $STPTPN$ [1].

3.5 Conclusion

Les RdP ont largement démontré leur efficacité et leur facilité d'usage pour la modélisation et l'analyse des systèmes à évènements discrets, offrant ainsi un modèle puissant pour la modélisation des systèmes parallèles et concurrents. Rapidement, la nécessite de la prise en compte du facteur temps comme une composante explicite de l'application modélisée, a conduit à l'émergence des différents modèles temporisés. Ces derniers ont été intensivement utilisés dans une optique d'évaluation des performances. Néanmoins, la modélisation de certains systèmes temps réel complexes nécessite d'exprimer plusieurs de ces sémantiques dans un même modèle. Partant de là, la définition d'un tel cadre de spécification permettrait d'envisager une caractérisation exhaustive des contraintes imposées et une analyse plus correcte des propriétés. Pour répondre à ces besoins, nous explorons dans les prochains chapitres la possibilité d'étendre les $RdPT$ à des mécanismes de synchronisation complexes dans le but de capturer la majorité des comportements observés dans les systèmes temps réel à forte synchronisation comme les systèmes des workflow.

Chapitre 4

Espace d'états des RdPT

4.1 Introduction

Les techniques de vérification de propriétés (e.g. accessibilité, vivacité, bornitude,...,etc) reposent sur le calcul et l'exploration de *l'espace d'états* du modèle. Or, L'espace d'états d'un RdPT borné étant (en général) infini, l'idée est de regrouper des états caractérisés par une même propriété. Il est donc nécessaire de recourir à une méthode permettant de calculer une *abstraction finie* de cet espace d'états. Nous présentons dans ce chapitre un panorama des méthodes appliquées sur les RdPTs, ces dernières sont essentiellement basées sur la notion de « graphe de classes d'états » et permettent de préserver un ensemble de propriétés intéressantes. La définition des classes d'états, des transitions entre classes d'états et la construction du graphe sont alors rappelées. Enfin, les précédentes méthodes et ses variantes sont discutées et comparées selon différents critères (tableau comparatif).

4.2 La méthode du graphe des classes d'états de *Berthomieu et al*

La méthode la plus courante de calcul de l'espace d'état d'un *RdPT* est le graphe des classes d'états [83][2]. Puisque l'espace d'états est infini, il a fallu rassembler les états en un nombre fini de groupes. Dans cette méthode, les groupes sont appelés *classes d'états*. L'ensemble des états d'un RdPT peut être infini pour deux raisons :

1. Parce qu'un état peut admettre une infinité de successeurs,
2. Parce qu'un réseau peut admettre des échéanciers de longueurs infinies passant par des états dont les marquages sont tous différents.

Pour gérer le premier cas, on regroupe certains ensembles d'états en classes d'états. Plusieurs regroupements sont possibles (voir la suite du chapitre). Une classe d'états regroupe donc tous les états atteints par une même séquence de tir réalisable. En particulier, tous les états d'une même classe ont alors le même marquage.

La Figure 4.1 présente la classe C_i qui regroupe l'ensemble des états (S_j), atteints après le tir de la transition t_i à partir de la classe C_k et à des instants différents. Nous décrivons dans la section suivante la méthode classique de construction de l'espace d'état d'un *RdpT* basée sur les classes d'états et ses différentes versions (variantes), classées chronologiquement ensuite, selon la classe de propriétés préservées.

4.2.1 Le graphe des classes d'états

La méthode de construction du graphe des classes d'états ou (*State Class Graph : SCG*) est initialement proposée par *Berthomieu et al* [83, 2]. Cette dernière préserve le marquage et les propriétés linéaire (LTL) du modèle. Nous présentons dans cette section la définition de la classe d'état. Nous montrons comment construire le graphe des classes d'états pas à pas avec un exemple illustratif. Enfin, nous mettons le point sur les différentes variantes de constructions du graphe et terminons par citer les avantages et les limites liés à ces approches.

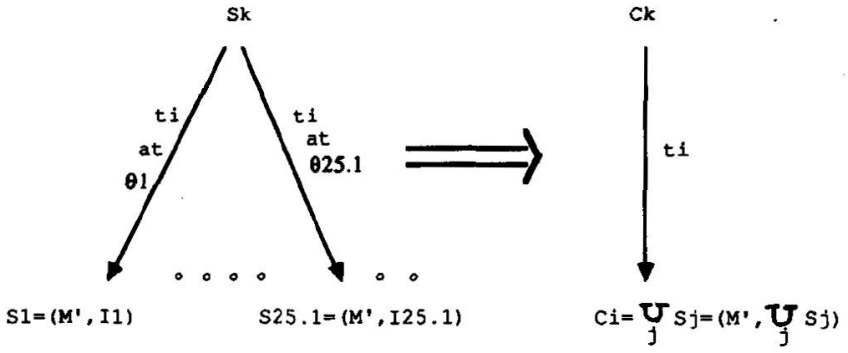

FIGURE 4.1 – Etat et classe d'états [2]

Classe d'état

Formellement, les états d'une même classe sont tous les états accessibles après franchissement de la même séquence de tir depuis l'état initial, mais à des dates différentes.

Définition 4.2.1. ***(Classe d'états)****. Une classe d'états est un couple $E = (M, D)$ dans lequel :*

- *M, est le marquage de la classe (marquage commun à tous les états de la classe) ;*
- *D, est le domaine de tir de la classe, défini par l'union des intervalles de tir associées à chaque état de la classe. La classe initiale E_0 contient seulement l'état initial e_0. D peut être représenté par l'ensemble des solutions d'un système d'inéquations linéaires, comportant une variable pour chaque transition sensibilisée par le marquage M de la classe. Les inéquations de D sont de deux types ; on parle alors d'un système mis sous forme DBM (Difference Bound Matrix) :*

  - $\forall t_i \in E(M), 0 \preceq MIN_{e\in E}\{EFT_e(t_i)\} \preceq \underline{t_i} \preceq MAX_{e\in E}\{LFT_e(t_i)\}$;
  - $\forall t_j, t_i \in E(M) \wedge (t_i \neq t_j), \underline{t_j} - \underline{t_i} \preceq MAX_{e\in E}\{LFT_e(t_j) - EFT_e(t_i)\}$. *Où $\underline{t_j}$ et $\underline{t_i}$ sont des variables associées respectivement aux transitions t_j et t_i.*

Transition de classes d'états (Condition de tir d'une transition)

Une transition t_i est tirable depuis la classe $E = (M, D)$, ssi :

(i) t_i est sensibilisée par M au sens des RdP ;

(ii) $D \wedge (\forall t_i \in E(M), \underline{t_i} \preceq \underline{t})$ est consistant.

La condition (ii) interdit le tir de toute transition validée (au sens des RdP) dont l'intervalle de tir (courant) est strictement précédé par celui d'une autre transition validée. Le tir de t_i conduit à la classe $E' = (M', D')$ suivante, caractérisée par :

- Le nouveau marquage $M' : \forall p \in P, M'(p) = M(p) - Pre(p, t) + Post(p, t)$;
- Le nouveau domaine D' est obtenu selon la procédure suivante (4 étapes) :

  a) Ajouter au système D, les conditions de tir de t_i exprimant qu'elle soit la première tirée parmi l'ensemble des transitions validées, i.e. $D \wedge (\forall t_j \in E(M), \underline{t_i} \preceq \underline{t_j})$;

  b) Supprimer dans le système obtenu les inéquations incluant les variables $\underline{t_j}$ associées aux transitions en conflit avec t_i, ces transitions sont désensibilisées par le tir de t_i ;

  c) Dans le système ainsi réduit, effectuer le changement de variables suivant : $\forall i \neq j$, $\underline{t_j} = \underline{t'_j} + \underline{t_i}$, et éliminer par substitution toutes les occurrences de la variable $\underline{t_i}$, pour ne garder que les nouvelles $\underline{t'_j}$;

  d) Compléter ce dernier système par une variable supplémentaire pour chaque transition nouvellement sensibilisée. Associer à ces transitions leurs intervalles de tir statiques.

L'ensemble de solutions du système déterminé à l'étape (c) peut être vu comme le domaine de tir des transitions distinctes de t_i qui sont restées sensibilisées pendant le tir de t_i, exprimé avec la nouvelle origine du temps ; la date à laquelle la transition t_i a été tirée. Les éliminations effectuées aux étapes (b) et (c) préservent les contraintes temporelles induites sur les variables restantes.

À partir de cette représentation finie des états accessibles et de la relation de transition entre classes, on peut aborder la notion de graphe des classes d'états.

Construction du graphe des classes d'états

La racine de ce graphe est la classe initiale E_0 ; pour chaque transition tirable dans une classe E, une nouvelle classe E' est créée. Afin de diminuer le risque d'explosion combinatoire, chaque nouvelle classe est comparée avec celles déjà produites pour tester une possible égalité. Si c'est le cas, l'exploration de la branche concernée est abandonnée (existence de séquences répétitives). L'existence d'un chemin dans cet arbre, reliant la classe initiale à la classe E, prouve la possibilité d'existence d'un échéancier de tir réalisable, supporté par ce chemin. Deux points restent à préciser pour pouvoir construire efficacement cet arbre :

- Comment tester l'égalité de deux classes ?
- Comment savoir si l'énumération se termine ?

Condition d'égalité entre classes d'états

Deux classes d'états $E = (M, D)$ et $E' = (M', D')$ sont égales par définition si et seulement si : $M = M'$ et $D = D'$. Il a été démontré dans [4] que le système d'inégalités particulières qui définit les domaines de tir, admet une représentation sous forme DBM unique. La complexité de calcul de la forme est polynomiale [2], précisément, égale à $o(l^3)$ où l étant le nombre de transitions sensibilisées pour M. Nous montrons plus loin dans ce chapitre que cette complexité de calcul d'une classe peut être réduite à $o(l^2)$ [87] [88].

Lorsque la forme DBM des domaines de tir est établie, vérifier l'identité de leurs formes suffit pour conclure à l'égalité de deux domaines. Au vu des résultats précédents, l'égalité de deux classes d'états pourra donc être construite de manière incrémentale, en même temps que sont énumérées les classes. L'énumération est achevée lorsque toutes les branches de l'arbre ont été explorées ; cela suppose que le nombre de classes soit fini. Mais nous avons vu que la finitude de l'ensemble des marquages accessibles est indécidable. Comme la définition des classes d'états inclut le marquage, la finitude de l'ensemble des classes est aussi indécidable, d'où l'objet du paragraphe suivant.

Quelques résultats

Finitude du graphe des classes : En assumant un $RdPT$ borné, toute classe a un nombre fini de successeurs (au plus un par transition sensibilisée). Il reste à examiner les conditions sous lesquelles l'ensemble des classes est fini.

Théorème 4.2.1. *[2] Le nombre de classes d'un RdPt est fini si et seulement si le réseau est borné.*

Par conséquent, ce problème est indécidable car le problème de bornitude d'un $RdPT$ est indécidable. *Berthomieu et autres* [2] ont démontré que si les bornes des intervalles initiaux sont des rationnels, le nombre des domaines de tirs possibles est fini. Puisqu'un état est déterminé par le marquage et le domaine de tir, à partir de ce résultat, il est très facile de montrer que si le réseau est borné alors le graphe est borné et vice-versa.

Enfin, pour un $RdPT$, l'accessibilité de marquages est un problème indécidable [99], ce qui implique que la bornitude, l'accessibilité d'états et la vivacité sont également des problèmes indécidables. En revanche, la k-bornitude est décidable et peut être décidée par le calcul d'une abstraction finie de l'espace d'états telle que le graphe des classes d'états vu dans ce présent chapitre [83, 2].

Exemple illustratif

A titre d'illustration, construisons les classes du réseau temporel représenté en Figure 4.2. Initialement seule la transition t_1 est sensibilisée.

La classe initiale $E_0 = (M_0, D_0)$ avec : $M_0 : \{p_1, p_2\} \to 1$. $D_0 : \{4 \preceq \underline{t_1} \preceq 5$.

- Le franchissement de t_1 depuis E_0 conduit à la classe $E_1 = (M_1, D_1)$, avec :

$M_1 : \{p_3, p_4\} \to 1$

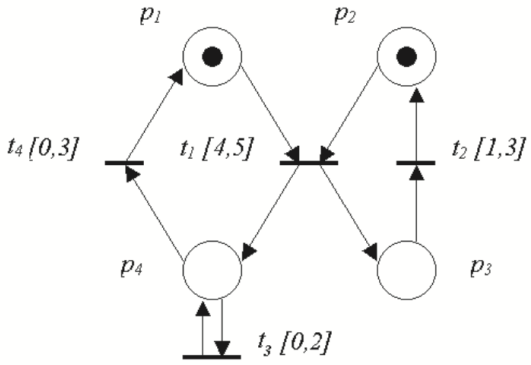

FIGURE 4.2 – Exemple illustratif

$$D_1 : \begin{cases} 1 \preceq \underline{t_2} \preceq 3 \\ 0 \preceq \underline{t_3} \preceq 2 \\ 0 \preceq \underline{t_4} \preceq 3 \end{cases}$$

- Les transitions sensibilisées par le marquage M_1 sont : t_2, t_3,t_4. Franchir t_2 depuis E_1 dans $[1, 2]$ (2= Min(3 ;2 ;3)), conduit à la classe $E_2 = (M_2, D_2)$, avec $M_2 : \{p_2, p_4\} \rightarrow 1$; et D_2 est calculée en quatre étapes, selon la règle définie dans le paragraphe précédent :

Etape (a) : $D_2(a)$ est obtenue en ajoutant à D_1 les conditions de tirabilité de t_1 : $\begin{cases} 1 \preceq \underline{t_2} \preceq 3 & \underline{t_2} \preceq \underline{t_3} \\ 0 \preceq \underline{t_3} \preceq 2 & \underline{t_2} \preceq \underline{t_4} \\ 0 \preceq \underline{t_4} \preceq 3 \end{cases}$

Etape (b) : Aucune transition n'étant en conflit avec t_2, on a $D_2(b) = D_2(a)$.
Etape (c) : Le changement d'origine produit le système suivant :

$$\begin{cases} 1 \preceq \underline{t_2} \preceq 3 & 0 \preceq \underline{t_2} + \underline{t_4} \preceq 3 \\ 0 \preceq \underline{t_2} + \underline{t_3} \preceq 2 & \underline{t_2} \preceq \underline{t_2} + \underline{t_4} \\ \underline{t_2} \preceq \underline{t_2} + \underline{t_3} \end{cases}$$

Depuis lequel $D_2(c)$ est obtenu par élimination de t_2 : $\begin{cases} 0 \preceq \underline{t_3} \preceq 2 \\ 0 \preceq \underline{t_4} \preceq 3 \\ \underline{t_4} - \underline{t_3} \preceq 1 \end{cases}$

Etape (d) : Aucune transition n'étant nouvellement sensibilisée, nous obtenons alors $D_2 = D_2(c)$.

- Franchir t_3 depuis E_1 dans $[0, 2]$ conduit à une nouvelle classe $E_4 = (M_4, D_4)$, avec : $M_4 : \{p_3, p_4\} \rightarrow 1$

$$D_4 : \begin{cases} 0 \preceq \underline{t_2} \preceq 3 \\ 0 \preceq \underline{t_3} \preceq 2 \\ 0 \preceq \underline{t_4} \preceq 3 \end{cases}$$

- Franchir t_4 depuis E_1 dans $[0, 2]$ conduit à la classe E_5 , avec : $M_5 : \{p_1, p_3\} \rightarrow 1$; $D_5 : \{0 \leq t_2 \leq 3$

Les transitions sensibilisées par le marquage M_2 sont : t_3, t_4.
- Franchir t_3 depuis E_2 dans $[0, 1]$ conduit à la classe $E_3 = (M_3, D_3)$ avec : $M_3 : \{p_2, p_4\} \rightarrow 1$.

$$D_3 : \begin{cases} 0 \preceq \underline{t_3} \preceq 2 \\ 0 \preceq \underline{t_4} \preceq 1 \end{cases}$$

- Franchir t_4 depuis E_2 dans $[0, 1]$ conduit à la classe initiale E_0.

Les transitions sensibilisées par M_3 sont : t_3, t_4, telles que :
- Franchir t_3 depuis E_3 dans $[0, 2]$ conduit à la même classe E_3.
- Franchir t_4 depuis E_3 dans $[0, 2]$ conduit à la classe initiale E_0.

Depuis E_4 :
- Franchir t_2 depuis E_4 dans $[0, 2]$ conduit à E_3 .
- Franchir t_3 depuis E_4 dans $[0, 2]$ conduit à la même classe E_4.
- Franchir t_4 depuis E_4 dans $[0, 2]$ conduit à une nouvelle classe $E_5 = (M_5, D_5)$.

Seule la transition t_2 est sensibilisée par M_5 :

- Franchir t_2 depuis E_5 dans $[0, 3]$ conduit à la classe initiale E_0.

L'énumération précédente des classes accessibles produit le graphe des classes de la Figure 4.3. Ce dernier à 6 classes d'états et 11 transitions d'états (arcs).

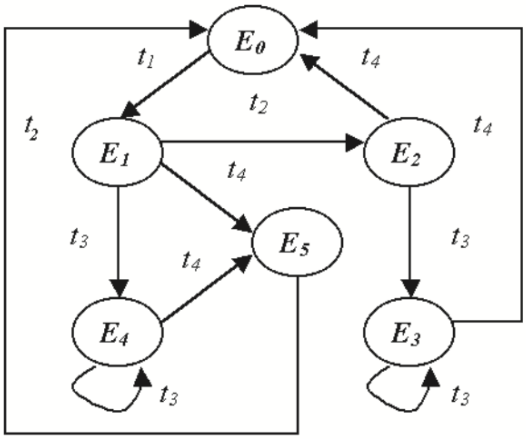

FIGURE 4.3 – Graphe de classes d'états du *RdPT* Fig.4.2

Variantes et extensions

- Nous pouvons construire une version plus compacte du graphe des classes d'états en ne mémorisant pas les classes incluses dans des classes déjà calculées [100] (e.g. une classe est incluse dans l'autre si elle a le même marquage et que l'ensemble des solutions de son système d'inégalités est inclus dans la première). Des variantes d'inclusion pour l'abstraction du graphe des classes d'états ont été proposées et comparées dans [101] telles que :(inclusion, convex union, convex hull), notées *(Inclusion State Class Graph : ISCG)*. Cependant, Les graphes obtenus par inclusion ne préservent plus les séquences de tir du graphe d'états, donc les propriétés LTL , mais seulement les marquages. *Boucheneb et al* ont proposé un sous ensemble de *RdpT* pour lequel le graphe des classes compactes, noté *(Compact State Class Graph : CSCG)*, préserve, marquage et propriétés LTL selon certaines conditions. On peut résumer ce cas comme suit : puisque le tir d'une transition désensibilise toutes les transitions en conflit avec cette dernière et lorsque la classe successeur ne contient que des transitions nouvellement sensibilisées (et donc initialisés par leurs intervalles statiques), cela nous permet de regrouper les classes (précurseurs) de cette dernière si la condition d'équivalence selon [12], est vérifiée. Le graphe de classe d'états relaxé de [101], noté *(Relaxed State Class Graph : RSCG)* consiste à calculer le graphe de classe standard sus-cité (SCG), et de relaxer la classe obtenue au fur et à mesure (la classe initiale également). Ceci permet d'avoir un graphe de classes étendues par progression du temps. Cette propriété rend l'inclusion plus puissante [1]
- La préemption : La méthode de construction de l'espace d'états est appliquée aux systèmes à chronomètres. Cependant, le système d'inéquation ne garde pas sa forme DBM et prend une forme générale dite *polyédrique*. Pour remédier à ce problème, les auteurs proposent un autre système sous forme d'une conjonction de deux, l'un prend la forme DBM et l'autre non. Le lecteur est renvoyé à [102].
- Synchronisation : l'adaptation de la méthode de calcul de l'espace d'état d'un $RdPT$ au modèle $TSPN$ est présentée dans [3]. Comme pour un $RdpT$, elle consiste à regrouper dans la même classe les états accessibles après le franchissement de la même séquence non temporisé. Cependant, dans un $TSPN$, les contraintes temporelles ne sont plus associées aux transitions, mais plutôt aux arcs entrants, en plus le tir d'une transition nécessite une condition plus complexe que celle requise dans un $RdPT$ classique. En effet, seulement les contraintes temporelles associées aux arcs sensibilisés sont présents dans le système D. Plus encore, le système de contraintes associé aux arcs ne donne pas une condition suffisante de tir et de consistance de classes, alors, un autre système équivalent est rajouté pour les contraintes des transitions sensibilisées. La Figure 4.4 représente le graphe de classe du modèle $TSPN$ de la Figure 4.4 du chapitre précédant. Le graphe de classe contient 9

1. la puissance de l'inclusion exprime la capacité d'une classe à inclure d'autres.

classes et 8 arcs [3]. Dans [1], l'application de la méthode du graphe des classes pour l'énumération de l'espace d'état d'un $STPTPN$ est proposée en prenant en compte plusieurs facteurs :

- la sémantique des *chronomètres* ;
- Le mécanisme de *synchronisation* ;
- Le mécanisme de *la priorité en temps réel*.

Cette dernière produit un *graphe sur-approximé*, si le modèle de base contient des arcs inhibiteurs et un *graphe exact* dans le cas contraire.

- Composition et Modularité : L'analyse et la vérification des systèmes temps réel avec RdPT composés (en parallèle), est proposée dans [40]. Cette dernière n'est qu'une extension de l'approche initialement proposée pour les *RdpT* dans [103]. L'outil RT-studio a été étendu pour l'analyse des systèmes avec modularité, un système avec un ensemble de RdPT étendus à des gardes et des actions associés aux transitions permettant de communiquer et de se synchroniser.
- Sensibilisation multiple : La construction de l'espace d'états des RdPT avec des transitions multi-sensibilisées (i.e. Une transition t est multi-sensibilisée par un marquage m s'il existe un entier $k \succ 1$ tel que $m \succeq k.Pre(t)$) est une variante proposée dans plusieurs travaux et selon différentes sémantiques (e.g. age semantics, threshold semantics,...,etc). Le lecteur est renvoyé à [97, 98].

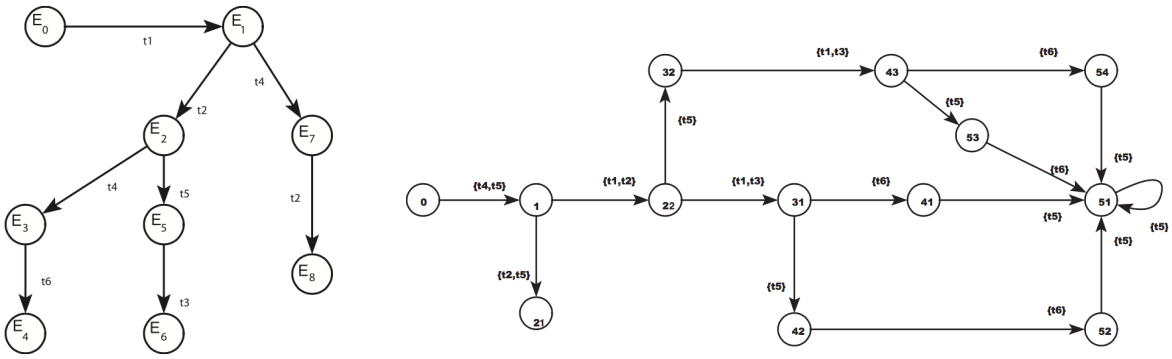

FIGURE 4.4 – Les graphes de classes du $TSPN$ Fig. 3.11 [3] et $STPTPN$ Fig.3.15 [1]

Les limites de l'approche

Le graphe des classes d'états permet d'obtenir l'ensemble des marquages accessibles d'un RdPT borné. Il préserve le langage non temporisé du réseau (marquage). Il permet ainsi de vérifier des propriétés de logique temporelle linéaire (Linear Temporal Logic - LTL) [79], c'est-à-dire des propriétés sur les exécutions d'un RdPT. Cette construction est supportée par un bon nombre d'outil incluant TINA [100, 104], ROMEO [105] et RT-studio [106].

Cependant, le graphe des classes ne préserve pas les propriétés de branchement comme CTL(Computation Tree Logic) [79]. En effet, lorsque la classe $E^{\uparrow}$ est le successeur de la classe E par la transition t, cela signifie qu'il existe au moins un état de E qui possède un successeur par t dans $E^{\uparrow}$, mais cela n'implique pas forcément que tous les états de E ont un successeur par t dans $E^{\uparrow}$. Pour remédier à cela, il est nécessaire de considérer des raffinements du graphe des classes d'états, comme ceux proposés par *Yoneda et Berthomieu* en graphe des classes d'états atomiques [84, 4] (voir plus loin).

Les propriétés quantitatives telles que la *réponse bornée* est une propriété très importante pour les systèmes temps réel. En effet, elle permet de définir le temps d'exécution de séquence de transitions. Cependant, cette propriété n'est pas donnée explicitement et des calculs supplémentaires doivent être effectués comme proposé dans [2, 83], avec le calcul des échéanciers relatif à une séquence de tir donnée.

Ce n'est plus la cas dans [88], car un algorithme est proposé permettant de définir des propriétés quantitatives (temps des chemins), avec une complexité de calcul inférieure.

4.2.2 Graphe des classes d'état fortes

Une construction plus fine de l'espace d'états des RdPT est proposée dans [4]. Le graphe connu sous le nom le graphe des classes d'état fortes ou *(Strong State Class Graph : SSCG)*, préserve les états et les propriétés linéaires (LTL).

Classe d'état

Afin de construire ce dernier graphe, il est nécessaire de représenter ces ensembles d'états de façon canonique. Une représentation adéquate est fournie par les domaines d'horloges.

Définition 4.2.2. ***(Classe d'état forte)***. *Une classe d'états forte est un couple $S = (M, H)$ dans laquelle :*

- *M, est le marquage de la classe (marquage commun à tous les états de la classe) ;*
- *H, un domaine d'horloges décrit par un système d'inéquations où chaque variable $\underline{t_i}$, est associée à une transition sensibilisée t_i qui définit le temps écoulé depuis la dernière sensibilisation de t_i. La classe initiale S_0 contient seulement l'état initial s_0. H peut être représentée par :*
  - $\forall t_i \in E(M), \quad 0 \preceq \underline{t_i} \preceq 0$;
  - $\forall t_j, t_i \in E(M), \quad 0 \preceq MIN_{e\in E}\{EFT_e(t_i) - \underline{t_i}\} \preceq \theta \preceq MAX_{e\in E}\{LFT_e(t_j) - \underline{t_j}\}$;
    Où $\underline{t_j}$ et $\underline{t_i}$ sont des variables associées respectivement aux transitions t_j et t_i.

Transition de classes d'états fortes

Une transition t_i est tirable depuis la classe $S = (M, H)$, ssi :

(i) t_i est sensibilisée par M au sens des RdP ;

(ii) $H \quad \wedge \quad 0 \preceq \theta \quad \wedge \quad 0 \preceq MIN_{e\in E}\{EFT_e(t_i) - \underline{t_i}\} \preceq \theta \preceq MAX_{e\in E}\{LFT_e(t_j) - \underline{t_j}\}$; est consistant. (θ est une nouvelle variable).

Le tir de t_i conduit à la classe $S' = (M', S')$, caractérisée par :

- Le nouveau marquage $M' : \forall p \in P, M'(p) = M(p) - Pre(p, t) + Post(p, t)$;
- Le nouveau domaine d'horloge H' est obtenu selon la procédure suivante (4 étapes) :

  a) Ajouter au système H, les conditions de tir (ii) (une nouvelle variable θ est ajoutée),

  b) Supprimer dans le système obtenu les inéquations incluant les variables $\underline{t_j}$ associées aux transitions en conflit avec t_i, ces transitions sont désensibilisées par le tir de t_i ;

  c) Dans le système ainsi réduit, effectuer le changement de variable suivant : $\forall i \neq j$, $\underline{t'_j} = \underline{t_j} + \theta$, et éliminer par substitution toutes les occurrences de la variable $\underline{t_j}$ et la variable θ, pour ne garder que les nouvelles $\underline{t'_j}$;

  d) Compléter ce dernier système par une variable supplémentaire pour chaque transition nouvellement sensibilisée t_i, tel que $0 \preceq \underline{t_i} \preceq 0$.

La variable θ décrit les dates de tir possibles de t_i depuis les états de la classe de départ. A partir de cette représentation des états accessibles et de la relation de transition entre classes, on peut aborder la notion de graphe des classes d'états fortes.

Construction du graphe

Le graphe des classes d'états fortes est construit à partir de la classe initiale de la même manière que le graphe des classes (plus haut). Il est à noter que, dans le cas des RdPT avec intervalles statiques bornés, l'ensemble des systèmes d'horloges distincts que l'on peut construire est fini. Dans ce cas, deux classes d'états fortes $S = (M, Q)$ et $S' = (M', Q')$ sont égales par définition [4], si et seulement si : $M = M'$ et $Q = Q'$.

Dans le cas où le nombre des transitions concurrentes est important, une méthode alternative (spécification de relaxation), est proposée : *Normalisation* [107] ; elle permet de préserver la convexité du système d'horloge. Moins coûteuse que la relaxation, elle permet de calculer le plus grand ensemble d'horloge préservant les états et peut être décrite sous forme de système DBM. Cette étape est appliquée à toutes les classes générées par la démarche citée ci-dessus.

Exemple Illustratif

A titre d'illustration, construisons le graphe des classes d'état fortes du RdPT représenté en Figure 4.5, initialement présenté dans [4]. Au départ, seule la transition t_1 est sensibilisée.

La classe initiale $S_0 = (M_0, H_0)$ avec : $M_0 : \{p_1, 2p_2\} \rightarrow 1$. $H_0 : \{\underline{t_1} = 0$.

- Le franchissement de t_1 depuis S_0 conduit à la classe $S_1 = (M_1, H_1)$ (toutes les transitions sont nouvellement sensibilisées), avec :

$M_1 : \{p_3, p_4, p_5\} \rightarrow 1$

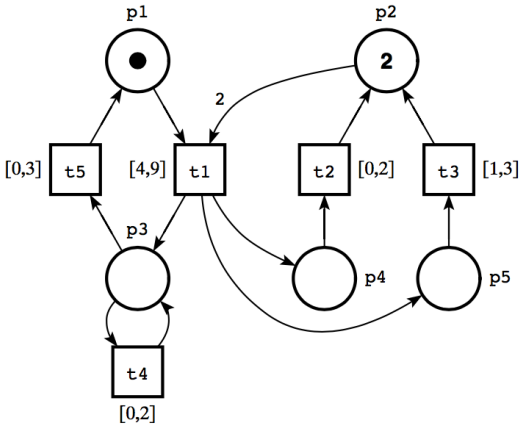

FIGURE 4.5 – Exemple illustratif [4]

$$H_1 : \begin{cases} \underline{t_2} = 0 \\ \underline{t_3} = 0 \\ \underline{t_4} = 0 \end{cases}$$

- Le franchissement de t_2 depuis S_1 conduit à la classe $S_2 = (M_2, H_2)$, avec :
$M_2 : \{p_2, p_3, p_5\} \rightarrow 1$ et H_2 obtenu comme suit :

$$H_2 : \begin{cases} 0 \preceq \theta \\ 0 \preceq \underline{t_2} + \theta \preceq 2 \\ 1 \preceq \underline{t_3} + \theta \preceq 3 \\ 0 \preceq \underline{t_4} + \theta \preceq 2 \\ 0 \preceq \underline{t_5} + \theta \preceq 3 \end{cases}$$

- Aucune transition n'est nouvellement sensibilisée par le tir de t_2, les transitions t_3, t_4, t_5 ne sont pas en conflit avec t_2. Nous allons simplement ajouter les contraintes suivantes :

$$\begin{cases} \underline{t_3}' = \underline{t_3} + \theta \\ \underline{t_4}' = \underline{t_4} + \theta \\ \underline{t_5}' = \underline{t_5} + \theta \end{cases}$$

- éliminer les variables $\underline{t_3}; \underline{t_4}; \underline{t_5}$ et θ ; nous obtenons :

$$\begin{cases} 0 \preceq \underline{t_3}' \preceq 2 \\ 0 \preceq \underline{t_4}' \preceq 2 \\ 0 \preceq \underline{t_5}' \preceq 2 \end{cases}$$

Le graphe des classes d'état fortes du RdPT de la Figure 4.5 admet 18 classes et 48 transitions.

Variante

Enfin, comme pour le graphe des classes (vu plus haut), et comme proposé dans [108], nous pouvons construire une version généralement plus compacte du graphe des classes d'état fortes ne préservant que les états (mais ne préservant pas les séquences de tir). Il suffit pour cela de ne pas mémoriser une classe contenue (par inclusion) dans une classe déjà construite. L'autre variante par (relaxation), revient à noter l'ensemble d'états S par le plus grand ensemble d'horloges décrivant cet ensemble d'états. Ce plus grand ensemble d'horloge n'est pas en général convexe, mais il est toujours constitué d'une réunion finie d'ensembles convexes. La normalisation est une autre variante du graphe et présente une spécialisation de la relaxation, le lecteur est envoyé vers [4] et [107].

Les limites de la méthode

C'est important de noter que le graphe de classes d'état fortes dans sa version compacte préserve seulement les états (*l'accessibilité* d'états), alors que le graphe de classes d'état fortes standard préserve états et propriétés LTL comme le graphe de classes d'état. Cependant, il est moins compact et son calcul est plus complexe. La construction du graphe des classes d'état fortes ne présente que peu d'intérêt par

elle-même. En fait, elle n'est fournie que parce qu'elle constitue le point de départ de la construction préservant les propriétés de branchement, décrite dans la section suivante.

4.2.3 Graphes de classes d'états atomiques

La première construction pour un graphe de classes atomiques, ou (*Atomic Stat Class graph : ASCG*) a été proposée par *Yoneda et al* [84] pour les RdPT avec intervalle statique borné supérieurement. Cependant, le graphe obtenu peut être infini même si le modèle est borné. Ultérieurement, la technique a été étendue pour tout type de *RdPT* dans [4] (le *RdpT* n'est pas forcément borné supérieurement), que nous rappelons dans cette sous section. Les variantes de cette technique ne sont rien d'autres que celles proposées pour la technique du (SSCG) étant donné que cette méthode se repose sur la construction du graphe des classes d'états fortes d'abord (voir la suite). Selon [84], nous appelons *atomique* une classe stable par rapport à toutes ses classes suivantes, c'est-à-dire dont chaque état a un successeur dans chacune de ses classes suivantes. Notons que les états dans une classe d'états forte (horloges) peut être distinguée une à une, alors que c'est impossible pour les états dans une classe d'état (délai exprimant le domaine de tir). La raison principale est que le domaine de tir d'une classe d'état est une union des domaines de tir de tous ses états, et l'*union* est connue comme étant une opération *irréversible*.

Les classes fortes sont adéquates comme partition initiale (contrairement aux classes du graphe des classes). Toutefois, le fait que cet ensemble soit un recouvrement plutôt qu'une partition, implique que le résultat final sera généralement non minimal en nombre de blocs, et non unique.

Construction du graphe des classes atomiques

Le graphe des classes atomiques est obtenu par raffinement du graphe des classes fortes. La technique de partition des classes est expliquée en détail dans [4]. Intuitivement, pour chaque transition du graphe courant, nous calculons depuis la classe destination $S^{\uparrow}$ l'ensemble des (horloges possibles) des états ayant un successeur dans $S^{\uparrow}$. L'intersection de cet ensemble avec celui capturé par la classe source S définit une partition de S. Le graphe des classes atomiques se construit comme suit : A partir du graphe des classes fortes : Tant qu'une certaine classe est non stable vis à vis de ses successeurs après le tir d'une transition t faire :

— Partitionner (ou éclater) la classe courante vis à vis de sa successeur par t
— Collecter toutes les classes accessibles à partir de la classe initiale.

Partitionner une classe remplace la classe courante par l'ensemble des classes résultantes du partitionnement de cette dernière.

Exemple illustratif

Afin de comprendre la constructions précédente, nous considérons le RdPT de la Figure 4.6.a), avec son graphe des classes LTL 4.6.b) et son graphe des classes CTL 4.6.c). La Figure 4.7 présente les valuation d'horloges correspondantes.

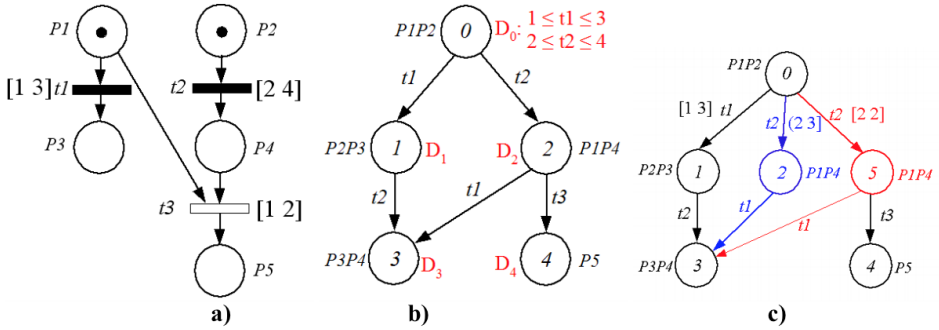

FIGURE 4.6 – a) RdPT b) graphe des classes LTL c) graphe des classes CTL

Le problème du branchement correspond au fait qu'une séquence du graphe peut ne pas être effectivement franchissable pour tous les états d'une classe. C'est le cas de la séquence t_2t_3 sur le graphe : elle n'est pas toujours franchissable pour tous les états de la classe 2, dans le RdPT. En effet, si t_2 est franchie à l'instant 3, t_3 ne pourra être franchie que dans l'intervalle $3 + [1, 2] = [4, 5]$. Or, d'après la

sémantique forte, c'est le temps qui commande et t_1 avec $[1,3]$ doit être franchie avant t_3. Par contre, si t_2 est franchie à l'instant 2, t_3 sera en conflit avec t_1 et pourra être franchie. Dans ce graphe des classes atomiques préservant les propriétés CTL 4.6.c), le problème du branchement est résolu : la classe 2 de la Figure 4.6.b) est partitionnée en deux classes 2 et 5. Si t_2 est franchie à l'instant 2 ; t_3 peut être franchie (elle est en conflit avec t_1), mais elle ne sera jamais franchie si t_2 est franchie entre $(2,3]$ (voir les valuations d'horloge dans la Figure. 4.7).

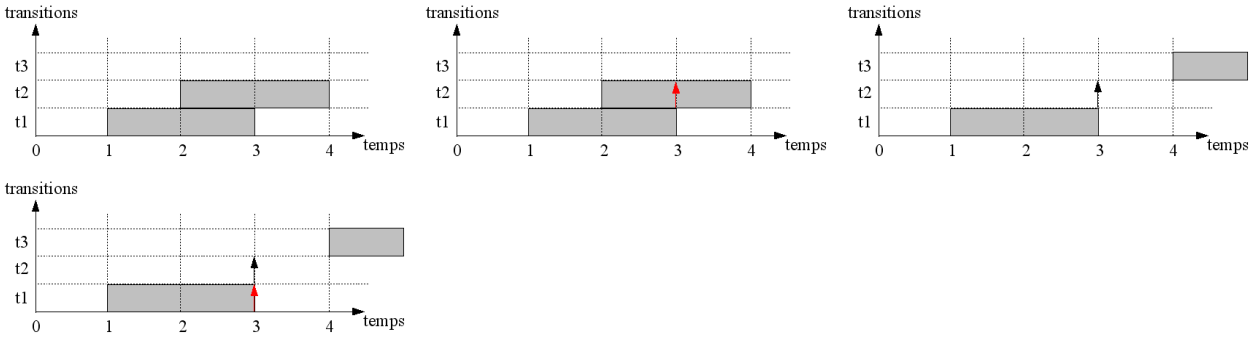

FIGURE 4.7 – Valuations d'horloges

Limites de l'approche

- Ce qui compte dans le graphe des classes atomiques c'est la différentiation entre les états pour lesquels il y a un conflit entre deux transitions de ceux pour lesquels seule l'une des deux transitions est franchissable. Dans l'exemple précédent, à partir de la classe 2 on peut tirer seulement t_1, alors qu'à partir de la classe 5, t_1 et t_3 sont en conflit et tirables.

- La mémoire du passé : Le graphe ne garde pas des informations dans le passé cela s'explique du fait qu'une classe est atteinte par plusieurs tirs de transitions. En effet, sur l'exemple précédent nous ne pouvons pas distinguer les états de la classes 3 successeurs de la classe 2 des états successeurs de la classe 5. Ce problème a été résolu par d'autres techniques dans la littérature mais qui ne sont pas le sujet de nos investigations actuelles.

- Il y a une classe de propriétés d'un grand intérêt pratique, pour laquelle la construction dédiée n'est pas proposée dans [84, 4, 86], celle des propriétés *quantitatives*, comme exprimées dans les logiques temporelles *temporisées* ou la réponse du temps borné.

4.3 La méthode des matrices de distances de *Boucheneb* pour le calcul du graphe des classes

Nous présentons, dans ce qui suit, la méthode basée sur *les matrices de distances* proposée par *Boucheneb et al* [88]. En appliquant cette méthode, il est possible de réduire d'avantage la complexité de calcul des classes d'états [2], en évitant d'appliquer systématiquement l'algorithme de *FLOYD-WARSHALL* [109]. L'idée est d'exploiter les calculs déjà réalisés sur les classes précédentes pour simplifier ceux des classes successeurs. Les auteurs proposent aussi un algorithme qui calcule, en parcourant un chemin du graphe des classes, les temps minimal et maximal d'exécution de la séquence de transitions du chemin. La complexité de l'algorithme est $O(mn)$, où m est la longueur du chemin et n est le plus grand nombre de transitions sensibilisées dans les classes du chemin.

Classe d'état

Soit $\bullet$ l'origine temporelle de la classe. $D[t,\bullet], D[\bullet,t]$ et $D[t,t']$ sont respectivement l'opposé du délai résiduel minimal de franchissement de t, le délai résiduel maximal de franchissement de t, et l'écart algébrique maximal, dans la classe E, entre les délais résiduels de franchissement des transitions t et t'. La classe d'états initiale $E_0 = (M_0, D_0)$ tel que M_0 est le marquage initial et la matrice D_0 définie sur $(E(M_0) \cup \bullet)^2$, comme suit :

$D_0[t,\bullet] = -EFT(t)\,; \quad D_0[t,t] = 0\,; \quad D_0[\bullet,\bullet] = 0\,; \quad D_0[\bullet,t] = LFT(t), \quad D_0[t,t'] = D_0[t,\bullet] + D_0[\bullet,t']$

Transition d'état

Partant d'une classe $E = (M, D)$ (sous forme canonique), le calcul des transitions franchissables peut être réalisé sans utiliser l'algorithme de *FLOYD-WARSHALL* ou tout autre algorithme de résolution de systèmes d'inéquations. En effet, une transition t_f est franchissable à partir de la classe E uniquement si elle est sensibilisée pour le marquage M et la distance maximale qui la sépare de chaque transition sensibilisée t est positive ou nulle, formellement :

$(\forall t \in E(M), \quad Max(t - t_f) \succeq 0$ ou encore $D[t_f, t] \succeq 0)$.

Le tir de t_f conduit à la classe $E' = (M', D')$ suivante, caractérisée par :

— Le nouveau marquage $M' : \forall p \in P, M'(p) = M(p) - Pre(p,t) + Post(p,t)$;

— La matrice D' est obtenu selon la procédure suivante :

a) $D[\bullet, \bullet] = 0, \quad \forall t \in E(M) \quad D_0[t,t] = 0$

b) $\forall t \in E'(M)$;

- Si t Persistante :

$D'[\bullet, t] = D[t_f, t] \qquad D'[t, \bullet] = Min\{D[t,t']\}; (\forall t' \in E(M))$

- Si t nouvellement sensibilisée :

$D'[\bullet, t] = LFT(t) \qquad D'[t, \bullet] = -EFL(t)$

c) $\forall (t,t') \in (E'(M))^2 \wedge t \neq t'$;

- Si (t,t') persistantes :

$D'[t,t'] = Min(D[t,t'], D'[t,\bullet] + D'[\bullet,t']$.

- Si (t,t') nouvellement sensibilisées :

$D'[t,t'] = D'[t,\bullet] + D'[\bullet,t']$.

Construction du graphe des classes d'état

La construction du graphe des classes d'état est réalisée en appliquant la règle de franchissement établie précédemment. Le graphe obtenu est semblable à celui du SCG plus haut à la différence cette approche réduit la complexité en temps qui passe de $O(n^3)$ à $O(n^2)$. Les auteurs ont ensuite développé un algorithme qui calcule, en explorant un chemin du graphe des classes, l'intervalle temporel de franchissement de la séquence du chemin. La complexité de cet algorithme est de $O(mn)$, où , est la longueur du chemin et n est le nombre de transitions sensibilisées dans la plus grande classe du chemin (en nombre de transitions sensibilisées).

Exemple illustratif

L'application de l'approche de calcul des classes au modèle de la Figure 4.8 a)- produit le graphe des classes illustré en Figure 4.8b)-. Il est à noter que le graphe obtenu est le même comparé avec la méthode des graphes de classes [83, 2]. Initialement seule la transition t_1 est sensibilisée.

La classe initiale $E_0 = (M_0, D_0)$ avec : $M_0 : \{p_0, p_2\} \rightarrow 1$.

D_0 :

	$\bullet$	t_1
$\bullet$	0	2
t_1	-1	0

- Le franchissement de t_1 depuis E_0 conduit à la classe $E_1 = (M_1, D_1)$, avec :

$M_1 : \{p_0, p_1, p_2\} \rightarrow 1$

D_1 :

	$\bullet$	t_1	t_2
$\bullet$	0	2	1
t_1	-1	0	0
t_2	-1	1	0

- Vérifier si la transition t_1 est franchissable ? t_1 est franchissable car $D_1[t_1, t_2] \succeq 0$. La classe successeur E_2 de E_1 par la transition t_1 a aussi deux transitions sensibilisées, i.e. : $E(M_2) = t_1, t_2$. La transition t_1 est nouvellement sensibilisée alors que la transition t_2 est persistante. La matrice des distances D_2 de la classe E_2 est calculée comme suit :

$D_2[\bullet, t_2] = D_1[t_1, t_2] = 0 \qquad D_2[t_2, \bullet] = MIN(D_1[t_2, t_2], D_1[t_2, t_1]) = 0$
$D_2[\bullet, t_1] = LFT(t_1) = 2 \qquad D_2[t_1, \bullet] = -EFT(t_1) = -1$
$D_2[t_1, t_2] = D_1[\bullet, t_2] + D_1[t_1, \bullet] = -1 \qquad D_2[t_2, t_1] = D_1[\bullet, t_1] + D_1[t_2, \bullet] = 2.$

$E_2 = (M_2, D_2)$, tel que :
$M_2 : \{p_0, 2p_1, p_2\} \rightarrow 1$

D_2 :

	•	t_1	t_2
•	0	2	0
t_1	-1	0	-1
t_2	0	2	0

-De la même manière, nous poursuivons le processus pour obtenir la classe E_3 :
$E_3 = (M_3, D_3)$, tel que :
$M_3 : \{p_0, p_1, p_2\} \rightarrow 1$

D_3 :

	•	t_1
•	0	1
t_1	0	0

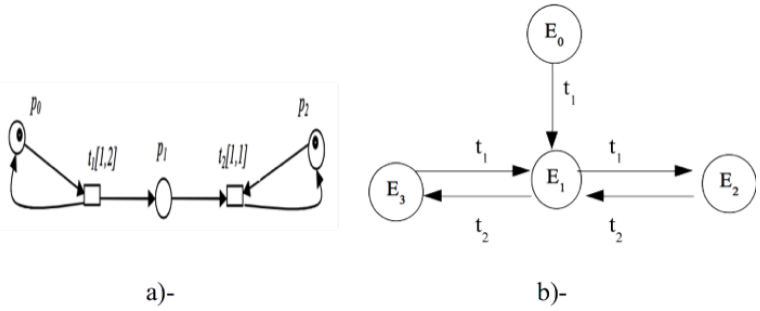

FIGURE 4.8 – a) Exemple d'un *RdpT* et b) son graphe de classes.

Le graphe a quatre classes d'états et cinq transitions d'états.

4.4 Discussion et Comparaison

Cette section est dédiée à la comparaison des constructions de graphes vu plus haut. La première partie de cette section est consacrée à la comparaison des deux classes de construction celle proposée par Berthomieu (SCG) [83, 2] et celle proposée par *Boucheneb* [88] et basée sur les matrices de distances. En général, l'algorithme de construction du graphe basé matrices de distances n'est qu'une mise en œuvre de celle du graphe (SCG) et produit le même graphe conservant les même propriétés de marquage et linéaires (LTL), mais avec certaines différences :

- La complexité en temps de calcul d'une classe successeur est de $O(n^2)$, où n est le nombre de transitions sensibilisées pour le marquage de la classe. Comparativement à la méthode de calcul utilisant l'algorithme de SCG, l'approche réduit la complexité en temps qui passe de $O(n^3)$ à $O(n^2)$.

- La complexité de l'algorithme de calcul des temps de chemin ou de cycle : est $O(mn)$, où m est la longueur du chemin et n est le plus grand nombre de transitions sensibilisées dans les classes du chemin. Comparativement à la méthode du SCG , en utilisant l'algorithme de FLOYD-WARSHALL, la complexité de calcul serait $O(mn^3)$. L'algorithme de FLOYD-WARSHALL est appliqué au moins une fois pour chaque arc du chemin.

La seconde partie de cette section est consacrée à la comparaison des variantes de constructions des graphes de classe de Berthomieu à savoir les graphes : SCG, SSCG et ASCG (On peut facilement reprendre cette comparaison sans rien changer et remplacer le SCG par le graphe de Boucheneb). La comparaison se porte sur les aspects les plus intéressants comme suit :

— La définition de la classe : pour toutes les constructions, une classe est définie par le couple : *marquage* (tous les états de la même classe ont le même marquage) et un *système d'inéquations* différent pour ces constructions. En effet, si pour le SCG et ses variantes le système représente des *délai* ou *intervalles* de tir, pour SSSCG et ASCG le système présente des valuations *d'horloges*.
— Les propriétés préservées : La propriété non temporisée (marquage) est préservée par le SCG, l'accessibilité d'état est préservée par le SSCG et les deux préservent les propriétés linéaires comme celles décrites en logique LTL. Cependant, leurs variantes (par inclusion ou relaxation) ne permettent pas de garder les propriétés linéaires des deux constructions.

Construction/graphe	**Propriétés**	**Les variable / système**	**taille et complexité**	Classe initiale
Construction standard : SCG de [83, 2]	marquage et LTL	Délai (dates de début et de fin de tir)	$O(n^3)$ pour le graphe et $O(mn^3)$ pour les chemins	$M_0 \wedge EFT(t) \preceq \underline{t} \preceq LFT(t)(\forall \quad t \in M_0)$
Construction compacte : SCG de [83, 2]	marquage	Délai (dates de début et de fin de tir)	tailles de graphe réduites par rapport à SCG	$M_0 \wedge EFT(t) \preceq \underline{t} \preceq LFT(t)(\forall \quad t \in M_0)$
Construction compacte : CSCG de [12]	marquage et (LTL ? selon des conditions)	Délai (dates de début et de fin de tir)	tailles de graphe réduites par rapport à SCG	$M_0 \wedge EFT(t) \preceq \underline{t} \preceq LFT(t)(\forall \quad t \in M_0)$
Construction relaxée : RSCG de [101]	marquage	Délai relaxés	tailles de graphe réduites par rapport à SCG et SCG compacte	$M_0 \wedge EFT(t) \preceq \underline{t} \preceq LFT(t)(\forall \quad t \in M_0)$
Construction : SSCG de [4]	états et LTL	Valuations d'horloge (le temps écoulé depuis la dernière sensibilisation de la transition)	temps de calcule et tailles de graphe plus important par rapport à SCG	$M_0 \wedge 0 \preceq \overline{t} \preceq 0(\forall \quad t \in M_0)$
Construction par inclusion SSCG de [86]	états	Valuation d'horloge	tailles de graphe réduite par rapport à SSCG	$M_0 \wedge 0 \preceq \overline{t} \preceq 0(\forall \quad t \in M_0)$
Construction relaxée : RSSCG de [107]	états	Valuation d'horloge relaxés	tailles de graphe réduite par rapport à SSCG	$M_0 \wedge 0 \preceq \overline{t} \preceq 0(\forall \quad t \in M_0)$
Construction : ASCG de [4]	états et branchement CTL	Valuation d'horloge	temps de calcule et tailles de graphe plus important par rapport à SSCG	$M_0 \wedge 0 \preceq \overline{t} \preceq 0(\forall \quad t \in M_0)$
Construction basée distance : [88]	marquage et LTL	distances de temps	$o(n^2)$ pour le graphe et $O(mn)$ pour les chemins	$M_0 \wedge EFT(t) \preceq \underline{t} \preceq LFT(t)(\forall \quad t \in M_0)$

TABLE 4.1 – Tableau comparatif

— La taille du graphe obtenu : généralement, l'algorithme de construction du SSCG fourni un graphe plus large que celui de SCG. Le graphe ASCG est obtenu après le partitionnement de certaines classes du SSCG donc sa taille devrait augmenter. Pour les variantes de chaque construction (SCG,SSCG et ASCG) ; par inclusion ou relaxation, le graphe obtenu est plus compacte car il regroupe les classes incluses dans des classes plus grandes donc la taille du graphe obtenu devrait diminuer (Voir tableau suivant pour plus de détail).
— Complexité : la complexité des calculs pour le SSCG est souvent plus élevée que celles pour le SCG.

Il est à noter que les variantes des constructions par relaxation permettent d'avoir des graphes plus compacts et plus petits et c'est un avantage, car cela permet de vérifier les propriétés de l'accessibilité d'états ou de marquage en un temps réduit. Cependant, les propriétés linéaires ne sont généralement plus conservées.

Remarque 4.4.1. *Aucune des constructions précédentes ne permet de vérifier des propriétés temporelles quantitatives de type TCTL (Timed Computation Tree Logic) car elles ne fournissent pas d'informations quantitatives implicitement sur les instants de tir de transitions. Un algorithme supplémentaire doit être établi [2, 88], d'autres auteurs ont ont proposé une sous-classe des RdpT vérifiant ces propriétés [110, 111].*

4.5 Conclusion

Dans ce chapitre, nous avons rappelé les principales méthodes permettant de calculer l'espace d'états des *RdPT*. Nous avons aussi rappelé les définitions de classe d'états de transition d'états et de constructions du graphe pour chaque approche. Les variantes les plus intéressantes sont discutées en se focalisant sur l'avantage de ces dernières. Enfin un tableau comparant les différentes méthodes sus-citées et leurs variantes est présenté. Les constructions sont comparés selon un ensemble de critères citons : les propriétés préservées ; le type de variables ou de systèmes de contraintes temporelles considéré ; la complexité de calcul ; la taille du graphe obtenu ; et enfin la définition de la classe, particulièrement la classe initiale.

Nous explorons dans le prochain chapitre une nouvelle sémantique qui permet d'étendre les *RdPT* à différents mécanismes dans le but de capturer la majorité des comportements observés dans les systèmes temps réel complexes, plus particulièrement on s'intéresse aux mécanisme de synchronisation et délai.

Chapitre 5

Synchronisation et contraintes temporelles : Notion de *Rendez-vous*

5.1 Introduction

Nous présentons tout au long de ce chapitre, notre modèle de synchronisation pour les systèmes temps réel complexes. En effet, le modèle permet à un ensemble de processus parallèles, avec le même privilège ou non, à un point donné, de se synchroniser conjointement. Le nouveau modèle basé sur le concept de *Rendez-vous Temporel* ou simplement : *Rendez-vous* , définit des stratégies et des patterns couvrant un large panel des règles connues en littérature notamment celles d'*Allen* [30] et *Wahl* [31] . Enfin, un tableau récapitulatif de toutes les règles et stratégies est présenté.

5.2 Présentation du concept *Rendez-vous*

Nous présentons dans cette section notre approche en décrivant les différents paradigmes et notations utilisés.

Dans notre modèle, nous considérons un ensemble de processus qui participent au déroulement (ou fonctionnement) d'un système. Un processus peut avoir un privilège (*Maître*), ou non (*Esclave*) et être associé à un intervalle temporel qui définit la date au plus tôt pour commencer et la date au plus tard pour terminer son exécution. Un *Rendez-vous* est le point durant lequel un ensemble de processus doit attendre les autres, pour se synchroniser et s'exécuter conjointement.

Formellement, un *rendez-vous*, noté R est le tuple $(T_m, T_s, (Loc^-, Loc^+), (\alpha^-, \alpha^+, \delta^-, \delta^+))$, où :

— T_m est un ensemble fini non vide de processus maîtres ;

— T_s est un ensemble fini de processus esclaves. $T_m \cap T_s = \emptyset$ et on note $T_r = T_m \cup T_s$. Chaque processus t est caractérisé par des contraintes temporelles de la forme $[x(t), y(t)]$. $x(t)$ (resp. $y(t)$) dénote la date au plus tôt pour commencer (resp. la date au plus tard pour terminer) un processus t. $y(t) - x(t)$ définit alors la durée maximale d'un processus t. La différence entre un processus maître et un esclave est que le début ou la fin d'un rendez-vous doit être décidé exclusivement par l'exécution d'un processus maître. Cependant, le processus esclave ne doit pas nécessairement influencer la synchronisation sauf si un certain privilège lui serait attribué.

— Loc^-, Loc^+ sont les localités considérées dans un rendez-vous ; Loc^- est appelée *localié de début : (STL : Start Time locality)* et Loc^+ est la *localité de fin : (ETL : End Time Locality)*. Loc^- introduit des règles pour calculer le temps préférable pour commencer un rendez-vous, tandis que Loc^+ définit les règles pour calculer la date préférable pour le terminer. Les valeurs de localités Loc^- resp Loc^+ sont encodées par le couple (Set^-, Op^-) resp. (Set^+, Op^+). En conséquence, nous introduisons les expressions suivantes pour calculer les dates des deux localités :

$$Loc^- = \underset{\forall t \in Set^-}{Op^-} \{x(t)\}, \quad Loc^+ = \underset{\forall t \in Set^+}{Op^+} \{y(t)\}$$

où :

$Set^+, Set^- \in \{T_r, T_m\}$ et $Op^-, Op^+ \in \{MIN, MAX\}$.

Selon le type de synchronisation qu'on veut spécifier, la date de début Loc^- peut se référer à une des valeurs suivantes :

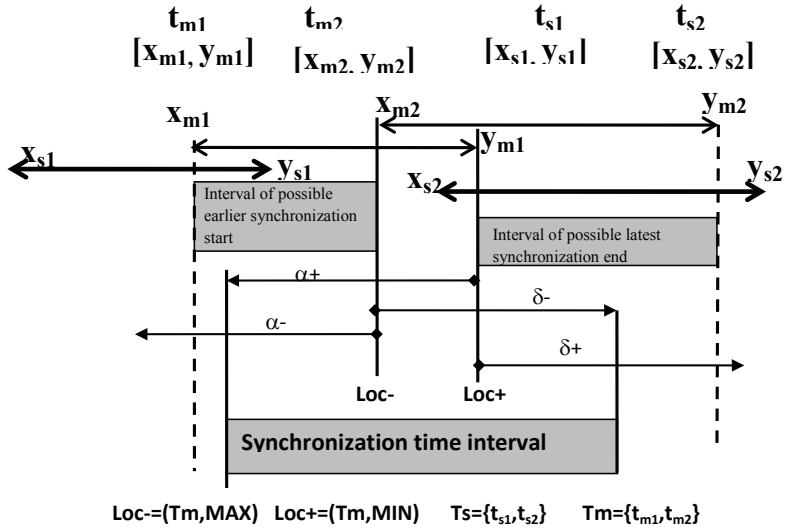

FIGURE 5.1 – Schéma de synchronisation : retard / avance

— (T_r, MAX) : Signifie que le début de la synchronisation est déterminé par le dernier processus ayant commencé son exécution. Dans ce cas tous les processus devraient avoir commencé, conjointement, leur exécution pour perfectionner la synchronisation.
— (T_m, MAX) : Signifie que le début de la synchronisation est déterminé par le dernier processus maître ayant commencé son exécution. Dans ce cas, tous les processus maîtres devraient synchroniser leur exécution. (voir Figure 5.1).
— (T_m, MIN) : Signifie que le début de la synchronisation est déterminé par l'exécution du premier processus maître. Une telle valeur de localité doit avoir au-moins un processus maître présent pour commencer la synchronisation.
— (T_r, MIN) : Signifie que le début de la synchronisation est décidé par l'exécution du premier processus. Dans cette synchronisation, au-moins un processus est valable pour commencer la synchronisation

De la même manière, Loc^+ est associé à quatre valeurs de localité :
— (T_r, MIN) : Signifie que la fin de la synchronisation est déterminée lorsque le premier processus termine son exécution. Ceci dit tous les processus ont déjà commencé leurs exécution et doivent terminer, conjointement, la synchronisation.
— (T_m, MIN) : Signifie que la fin de la synchronisation a lieu lorsque le premier processus maître finit son exécution. Dans ce cas, tous les processus maîtres ont déjà commencé leur exécution afin de pouvoir terminer, conjointement, le rendez-vous (voir Figure 5.1).
— (T_m, MAX) : Signifie que la fin de la synchronisation est définie par le dernier processus maître ayant fini son exécution. Dans ce cas au-moins un processus maître devrait être en train de s'exécuter pour terminer la synchronisation.
— (T_r, MAX) : Signifie que la synchronisation se termine lorsque le dernier processus termine son exécution. Dans ce cas, au moins un processus est en cours d'exécution pour terminer le rendez-vous.

Pour être plus cohérent avec la sémantique des rndez-vous maîtres, nous avons besoin dans notre modèle, qu'au-moins une de ses localités se réfère à un processus maître, soit $Set^- = T_m$ ou bien $Set^+ = T_m$. Ceci force le début ou la fin de la synchronisation pour qu'elle soit exclusivement décidée par l'exécution d'un processus maître.

— Les paramètres $\alpha^-, \alpha^+, \delta^+$ et δ^- sont des rationnels positives de $Q^+ \cup \{+\infty\}$ qui déterminent l'*avance* ou le *retard* sur une localité. Concrètement, afin de relaxer les contraintes déterminées par les valeurs de localités prédéfinies plus haut, nous utilisons des paramètres, en plus, pour exprimer

le décalage de temps relativement aux points de la synchronisation définis par : Loc^- et Loc^+.

— α^- (resp. α^+) représente l'avance permis sur la localité Loc^- (resp. Loc^+), pour commencer la synchronisation. Concrètement, α^- dénote que la date au plus tôt pour commencer la synchronisation peut être avancée de α^- unités de temps avant Loc^-, dans ce cas au-moins un processus de Set^- a déjà commencé son exécution. De l'autre coté, α^+ dénote que la date au plus tôt pour commencer le rendez-vous peut être avancé de α^+ unités de temps avant l'occurrence Loc^+ mais pas plus tôt que Loc^-, dans ce cas au moins un processus de Set^- a déjà commencé son exécution (voir Figure 5.1).

— δ^- (resp. δ^+) représente le retard permis sur la localité Loc^- (resp. Loc^+) pour terminer la synchronisation. δ^- dénote que la synchronisation doit se terminer δ^- unités de temps après Loc^-, et avant Loc^+, ce qui signifie qu'au moins un processus de Set^+ est entrain de s'exécuter. δ^+ dénote que la fin de la synchronisation peut être retardée, pas plus de δ^+ unités de temps après Loc^+ ce qui signifie qu'au moins un processus de Set^+ n'a pas encore fini son exécution (voir Figure 5.1).

5.3 Les stratégies d'un *Rendez-vous*

Soit $R = (T_m, T_s, (Loc^-, Loc^+), (\alpha^-, \alpha^+, \delta^-, \delta^+))$ un rendez-vous. Selon la valeur des paramètres temporels et des deux localités, nous définissons et discutons par la suite un panel de modèles de synchronisation pouvant être dérivé du concept de *rendez-vous*. D'abord, en combinant les valeurs de localité Loc^- et Loc^+, nous obtenons 12 règles générales pouvant être regroupées en 4 stratégies principales comme suit :

— $(Op^- = MAX)$: Dans cette stratégie, le début de la synchronisation est décidé lorsque tous les processus de Set^- sont prêts à commencer leur exécution. Cette stratégie est appelée, *Co-Begin*

— $(Op^- = MIN)$: Dans cette stratégie, la date au plus tôt pour joindre la synchronisation est décidée par le premier processus de l'ensemble Set^- qui a commencé son exécution. Cette stratégie est appelée, *One-Begin*

— $(Op^+ = MIN)$: Dans cette stratégie, la fin d'un rendez-vous est déterminée lorsque tous les processus de l'ensemble Set^+ sont prêts pour achever leur exécution. Cette stratégie est appelée, *Co-End* .

— $(Op^+ = MAX)$: Dans cette stratégie, la date au plus tard pour finir la synchronisation est asynchrone et est décidée lorsque tous les processus de l'ensemble Set^+ ont terminé leur exécution de façon asynchrone. Cette stratégie est appelée, *One-End* .

Les stratégies *One-Begin* et *CoBegin* montrent la façon avec laquelle les processus devront synchroniser leur commencement tandis que les stratégies *One-End* et *CoEnd* spécifient comment les processus devront se synchroniser pour terminer leur exécution.

Chaque stratégie définit une classe de pattern. Par restriction des conditions, nous pouvons définir un sous-pattern dans chaque classe comme décrit plus loin dans ce chapitre. Ensuite, en considérant des valeurs spécifiques des paramètre temporels, chaque classe de pattern peut être relaxée pour couvrir partiellement ou totalement le pattern d'une autre classe.

Pour mieux comprendre le concept de rendez-vous, considérons l'exemple d'une *Agence de voyage* décrit ci-après :

Exemple "*Agence de voyage*"

Pour organiser des voyages nationaux et internationaux, une agence de voyage doit effectuer l'ensemble des tâches suivantes.

Après l'enregistrement de la requête du client, l'opérateur doit procéder afin de trouver les propositions qui devront être présentées au client. Si le client est satisfait et qu'une proposition répond à ses attentes ; la réservation est établie.

L'enregistrement est constitué de différentes sous tâches parallèles : Réservation du vol (*Book-flight*), réservation d'une chambre d'hôtel (*Book-hotel*), et réservation d'une voiture (*Book-car*), ainsi que le payement d' une assurance de voyage dans le cas d'une annulation ou perte de bagages ; les sous-tâches *Insur-cancel* et *Insu-loss*) sont alors effectuées. Selon la demande du client, la réservation peut inclure toutes les tâches, ou non. En effet, nous pouvons exclure l'assurance voyage et considérer un sous ensemble des tâches précédentes.

Les contraintes temporelles associées à chaque tâche déterminent la date au plus tôt de commencer et la date au plus tard d'achever une tâche. Chaque tâche a ses propres contraintes de temps selon la disponibilité du client et les besoins administratifs. Plus encore, des tâches particulières peuvent être

avancées ou retardées relativement à d'autres, afin de trouver le bon vol, la chambre désirée ou pour une éventuelle requête de visa.

5.3.1 Les patterns *One-Begin*

Formellement, les patterns *One-begin* sont obtenus lorsque la condition ($Op^- = MIN$) est satisfaite. Cela signifie que le commencement (début) du rendez-vous est décidé par le premier processus parmi l'ensemble Set^- ayant commencé son exécution. Suivant la cas où T_s est vide ou non, deux sous-classes du *One-Begin* peuvent être introduites :

- Si $T_s = \emptyset$: Nous avons nécessairement $Set^- = T_m$. Tous les processus considérés dans le pattern *One-Begin* ont le même privilège. Nous appelons cette sous-classe, *Close One-begin*(COB).
- Si $T_s \neq \emptyset$: Dans cette sous-classe, la présence d'au moins un processus esclave est requise dans le pattern *One-Begin*. Nous appelons cette sous-classe, le *Open One-Begin*. Plus encore, suivant la valeur de Set^- deux types de *Open One-Begin* sont définis :
  - Si $Set^- = T_m$: Le début de la synchronisation est décidé par l'exécution du premier processus maître. Nous appelons ce type de pattern, le *Master Open One-Begin* (MOOB).
  - Si $Set^- = T_r$: Le début de la synchronisation est décidé par l'exécution du premier processus. Ce type de pattern est appelé *Extended Open One-begin* (EOOB).

La différence entre le COB et le MOOB est que la fin de ce dernier peut être décidée par la fin d'un processus esclave, tandis que la fin du premier est nécessairement décidée par un processus maître. Cependant, le début des deux est décidé par l'exécution du premier processus maître.

Tous les patterns définis dans le *One-Begin* ne peuvent pas être restreins ou relaxés en prenant en considération les paramètres (α^-, α^+). Ceci est dû au fait que le début de la synchronisation ne peut pas être avancé plus tôt que l'exécution du premier processus parmi l'ensemble Set^-.

La Figure 5.2 illustre la classe du pattern *One-Begin*. Nous avons $T_m = \{t_{m1}, t_{m2}\}$ et $T_s = \{t_{s1}, t_{s2}\}$. Le rectangle gris foncé représente la durée nominale de la synchronisation, le rectangle gris clair représente l'extension de la durée de la synchronisation.

Dans la représentation du pattern COB, les processus esclaves $\{t_{s1}, t_{s2}\}$ sont exclus. La synchronisation a lieu lorsque le premier processus maître est exécuté (t_{m1}) ; tel que x_{tm1} dénote la date de Loc^-. La fin de l'exécution de COB dépend de la valeur de Loc^+. Ceci couvre l'intervalle $[y_{tm1}, y_{tm2}]$ du premier processus maître qui termine son exécution (t_{m1}, si $Loc^+ = (T_m, MIN)$) au dernier processus maître qui s'achève (t_{m2} si $Loc^+ = (T_m, MAX)$).

Dans le cas où les processus esclaves sont convoqués dans le pattern, le MOOB est réalisé lorsque le premier processus maître est exécuté (t_{m1}), cependant le EOOB est la stratégie requise lorsque le première processus commence (t_{s1}). Selon la valeur de Loc^+, l'exécution du pattern MOOB peut finir dans le meilleur des cas lorsque le premier processus termine son exécution (t_{s1} si $Loc^+ = (T_r, MIN)$). La durée du pattern peut être étendue au dernier processus qui termine son exécution (t_{s2} si $Loc^+ = (T_r, MAX)$).

Pour EOOB, l'achèvement de la synchronisation doit être décidé par la fin d'un processus maître. Pour cela, la durée nominale dure jusqu'au premier processus maître qui termine son exécution (t_{m1} si $Loc^+ = (T_m, MIN)$) et peut être étendue jusqu'à la fin d'exécution du dernier processus maître (t_{m2} si $Loc^+ = (T_m, MAX)$).

Exemple : Revenons à l'exemple de l'*agence de voyage* dans lequel nous considérons l'ensemble des tâches suivantes pour le processus d'une réservation nationale (sans visa) : T_m= { *Book-hotel, Book-Flight, Book-Car* } et T_s= { *Insu-trip, Insu-Loss* }. Les tâches concernant une réservation s'exécutent en parallèle et ont besoin de synchroniser leur exécution.

Si toutes les options de réservation sont traitées séparément nous appliquons alors, la synchronisation par *One-Begin*. Le date de réservation est alors définie par la date de la première tâche réalisée. Dans le cas,où le client n'a pas besoin de payer l'assurance de voyage proposée par l'agence, nous avons $T_s = \emptyset$, le pattern COB est considéré. Cependant, s'il n'est pas sûr de payer son assurance car il prévoit des procédures et donc du temps additionnel, alors nous appliquons la synchronisation MOOB. Par contre, dans le cas où il voudrait réserver une assurance comprise pour prévenir toute annulation ou vol raté, nous appliquons le pattern EOOB.

5.3.2 Les patterns *One-End*

Formellement, les patterns *One-End* sont définis lorsque ($Op^+ = MAX$). Cela signifie que la synchronisation se termine lorsque le dernier processus maître de Set^+ a achevé son exécution. Selon le cas où

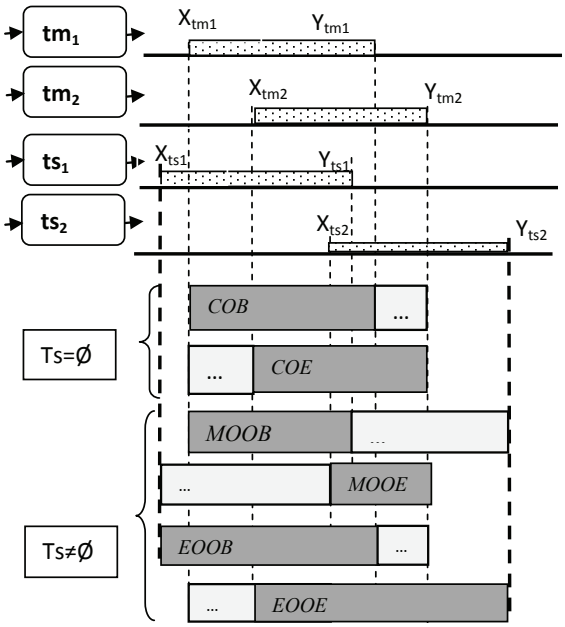

FIGURE 5.2 – Les patterns : One-Begin et One-End

T_s est vide ou non, deux sous-classes du *One-End* peuvent être dérivés :

- Si $T_s = \emptyset$: Nous avons nécessairement $Set^+ = T_m$. Tous les processus considérés dans le pattern *One-End* ont le même privilège. Nous appelons cette sous classe de pattern *Close One-End* (COE).
- Si $T_s \neq \emptyset$: Dans cette sous-classe, la présence d'au moins un processus esclave est requise. Nous appelons cette sous classe *Open One-End*. Selon la valeur de Set^+, deux types de *Open One-End* sont définis :
  - Si $Set^+ = T_m$: La fin de la synchronisation est décidée par le premier processus maître ayant fini son exécution. Nous appelons ce type de pattern, *Master Open One-End* (MOOE).
  - Si $Set^+ = T_r$: La synchronisation s'achève lorsque le dernier processus finit son exécution. Ce type de pattern est appelé, *Extended Open One-End* (EOOE).

Bien que la synchronisation dans le COE et le MOOE se termine de la même manière, le début peut être différent. En effet, le début de MOOE peut être décidé par le début du processus esclave, alors que le début de COE est nécessairement décidé par le début d'un processus maître.

Comme pour la classe *One-Begin*, tous les patterns définis dans la classe *One-End* ne peuvent pas être relaxés en variant les valeurs des paramètres (δ^-, δ^+). Ceci est dû au fait que la fin de la synchronisation ne peut pas être retardée plus que la date d'exécution du dernier processus parmi l'ensemble Set^+.

Dans la Figure 5.2, la sémantique du pattern *One-End* est représentée. Dans le cas de COE, les processus $\{t_{s1}, t_{s2}\}$ sont ignorés. La synchronisation se termine lorsque le dernier processus maître est achevé (t_{m2}) ; y_{tm2} dénote la date de Loc^+. Le début du pattern est décidé par le processus maître et dépend de la valeur de Loc^-. Ceci couvre l'intervalle $[x_{tm1}, x_{tm2}]$ du premier processus maître qui commence son exécution (t_{m1} si $Loc^- = (T_m, MIN)$) jusqu'à la fin du dernier processus maître (t_{m2} si $Loc^- = (T_m, MAX)$).

Dans le cas du pattern MOOE, ce dernier se termine lorsque le dernier processus maître termine son exécution (t_{m2}) alors que le EOOE se termine lorsque le dernier processus se termine (t_{s2}). La date du début de MOOE dépend de la valeur de Loc^-, et couvre l'intervalle allant de la date du premier processus qui commence son exécution (t_{s1} si $Loc^- = (T_r, MIN)$) jusqu'à la date du dernier processus qui commence son exécution (t_{s2} si $Loc^- = (T_r, MAX)$).

En ce qui concerne le pattern EOOE, son commencement doit être décidé par le commencement d'un processus maître. Par conséquent, soit le dernier processus qui commence son exécution (t_{m2} si $Loc^- = (T_m, MAX)$), ou peut être avancée au début de l'exécution du premier processus maître (t_{m1}) si $Loc^- = (T_m, MIN)$).

Exemple : Revenons à notre exemple. Si nous considèrons les tâches suivantes pour une réservation nationale, T_m= { *Book-hotel, Book-Flight, Book-Car* } et T_s= {*Insu-trip, Insu-Loss* }. Dans le cas où le client n'a pas besoin d'assurance ($T_s = \emptyset$), la réservation se termine lorsque toutes les tâches maîtres sont exécutées, donc nous appliquons le pattern COE. Cependant, s'il a besoin d'une assurance, deux cas peuvent être observés : s'il veut contracter de lui même une assurance plus tard, nous appliquons le pattern EOOE. Autrement, s'il a déjà contracté une assurance, et a besoin seulement de compléter la réservation, nous considérons le pattern MOOE.

5.3.3 Les patterns *Co-Begin*

La classe des patterns *CoBegin* regroupe tous les patterns, tels que le début de la synchronisation a lieu lorsque tous les processus de l'ensemble Set^- sont prêts à commencer, conjointement, leur exécution.

Formellement, nous avons ($Op^- = MAX$). Au premier niveau, deux sous-classes du *CoBegin* peuvent être introduites, selon le cas où T_s est vide ou non :

- Si $T_s = \emptyset$: Dans cette sous classe, tous les processus considérés dans le *CoBegin* ont le même privilège. Nous appellons cette sous classe de pattern, le *close CoBegin*.
- Si $T_s \neq \emptyset$: Dans cette sous classe, la présence d'au moins un processus esclave est requise. Nous appelons cette sous classe de patterns le *Open CoBegin*. Pour cette variante, nous distinguons d'autres types de patterns selon la valeur de Set^- :
  - $Set^- = T_m$: Dans ce cas particulier, la synchronisation est décidée lorsque tous les processus maîtres commencent conjointement leur exécution. Nous appelons ce type de pattern le *Master Open CoBegin*.
  - $Set^- = T_r$: Dans ce cas, la synchronisation commence lorsque tous les processus sont prêts pour commencer leur exécution. Nous appelons ce type de pattern, le *Extended Open CoBegin*.

Lorsque nous considérons, en plus, les paramètres temporels associés (α^-, α^+), la sous-classe de *CoBegin* peut être relaxée pour couvrir des sous variantes.

- Si ($\alpha^- = 0$) : Cela signifie que le début de la synchronisation ne peut pas être avancé relativement à la date Loc^- du pattern *CoBegin*. Ce type de pattern est appelé *Synchronous CoBegin*. En conséquent, selon la sous classe du pattern avec laquelle nous travaillons, nous obtenons *Synchronous Close CoBegin*(SCCB), le *Synchronous Master Open CoBegin* (SMOCB) ou le *Synchronous Extended Open CoBegin* (SEOCB).
- Si ($\alpha^- \neq 0$) : Dans ce cas, d'autres variantes de classes de patterns sont couvertes :
  - Si ($\alpha^+ \in [0, +\infty[$) ou ($\alpha^- \in]0, +\infty[$) : Dans ce cas, le *CoBegin* est relaxé pour permettre le commencement au plus tôt de la synchronisation et avancer dans le temps relativement au *synchronous CoBegin* associé, mais pas plus tôt que le *One-Begin* associé. Selon la sous classe de pattern, nous obtenons deux variantes : le *Advanced Close CoBegin* (ACCB), le *Advanced Master Open CoBegin* (AMOCB), et le *Advanced Extended Open CoBegin* (AEOCB).
  - Si ($\alpha^- = +\infty$) et ($\alpha^+ = +\infty$) : En supposant ces conditions, le *CoBegin* est relaxé au maximum en avançant son exécution et se comporte comme le *One-Begin* associé.

Les différentes variantes du pattern *CoBegin* sont présentées dans la Figure 5.3.a. Le SCCB commence lorsque tous les processus maîtres sont prêts à commencer leur exécution (t_{m2} est le dernier processus maître qui commence son exécution). Sa durée nominale est atteinte lorsque tous les processus maîtres sont prêts à terminer leur exécution (lorsque t_{m1} se termine), et peut être étendue au dernier processus maître qui achève son exécution (t_{m2} si $Loc^+ = (T_m, MAX)$).

Le SCCB peut être avancé mais pas plus tôt que le début de COB (début de t_{m1}). Le pattern SMOCB est similaire au pattern SCCB dans la façon de commencer. Cependant, la durée nominale du SMOCB est définie lorsque le premier processus se termine (t_{s1} si $Loc^+ = (T_r, MIN)$) et peut être étendue au dernier processus qui se termine (t_{s2} si $Loc^+ = (T_r, MAX)$).

En ce qui concerne le SEOCB, ce pattern commence lorsque tous les processus sont prêts pour commencer leur exécution (lorsque t_{s2} commence) et peut être avancé mais pas plus tôt que le premier processus qui commence son exécution (t_{s1}) (début de EOOB). Sa durée nominale est fixée lorsque le premier processus maître se termine (t_{m1} si $Loc^+ = (T_m, MIN)$) et peut être étendue au dernier processus qui se termine (t_{m2} si $Loc^+ = (T_m, MAX)$).

Il est à remarquer que si $Set^- = T_r$, alors nous avons nécessairement $Set^+ = T_m$. Cependant, si $Set^- = T_m$, alors Set^+ fait référence à T_r ou T_m.

Exemple : Revenons à notre exemple de réservation. Si nous considèrons l'ensemble des tâches suivantes pour un processus de réservation au national T_m= { *Book-hotel, Book-Flight, Book-Car* } et T_s=

{*Insu-trip, Insu-Loss* }. Dans le cas où le client n'a pas besoin d'assurance ($T_s = \emptyset$), et a une préférence pour l'hôtel et un type de voiture, l'opérateur peut faire une recherche pour voir les disponibilités de voiture, de chambre d'hôtel et de vol avec la même requête, afin de matcher la disponibilité des dates selon les besoins du client. Dans ce cas, nous appliquons le pattern SCCB. S'il y a une grande demande par exemple pour une chambre d'hôtel, l'opérateur peut décider d'avancer le début de la réservation en réservant en premier n'importe quelle chambre disponible de l'hôtel. Ensuite il regarde plus tard avec les autres tâches une fois la réservation du vol et de la voiture établie. Dans ce cas, nous appliquons le pattern ACCB. Cependant dans le cas où le client veut rajouter une assurance au processus, deux cas peuvent se présenter : S'il veut contracter plus tard l'assurance via l'agence de voyage, nous procédons avec le pattern SMOCB. Cependant, nous considérons le pattern AMOCB dans le cas où la réservation de la chambre doit être fixée avant les autres tâches.

Par ailleurs, si le client est sûre de payer l'assurance, l'opérateur peut inclure cette tâche lors du commencement du processus avec toutes les tâches ensembles, nous procédons dans ce cas avec le pattern SEOCB. Le AEOCB est considéré dans le cas de réservation de chambre précédé bien avant.

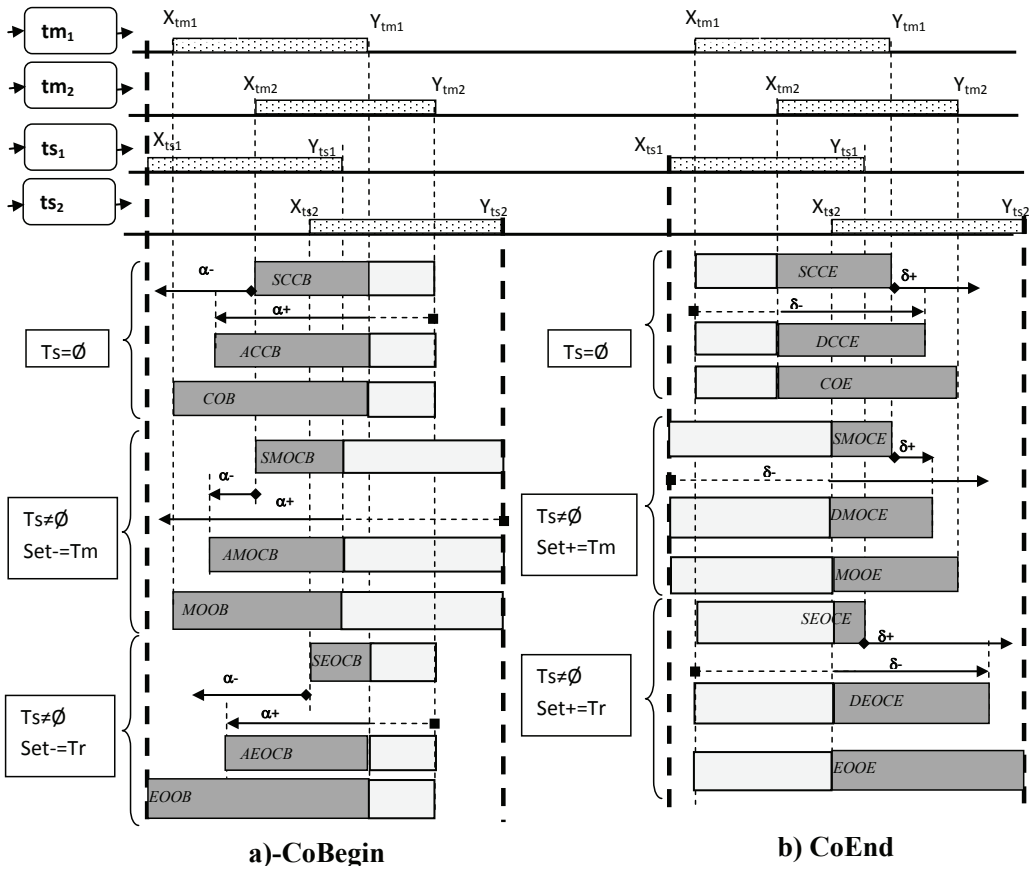

FIGURE 5.3 – Les patterns : a)-CoBegin et b)- CoEnd.

5.3.4 Les patterns *CoEnd*

Cette classe inclut tous les patterns tels que la date au plus tard pour terminer la synchronisation est décidée lorsque tous les processus parmi Set^+ terminent, conjointement, leur exécution. Formellement, nous avons ($Op^+ = MIN$). Au premier niveau, deux sous classes de *CoEnd* peuvent être considérées selon le cas où T_s est vide ou non.

- Si $T_s = \emptyset$: Nous avons nécessairement $Set^+ = T_m$. tous les processus considérés dans le pattern *CoEnd* ont le même privilège. Nous appelons cette sous classe de pattern, le *Close CoEnd*.
- Si $T_s \neq \emptyset$: Dans cette sous classe, la présence d'au moins un processus esclave est requise dans le *CoEnd*. Nous appelons cette sous classe, le *Open CoEnd*. Suivant la valeur de Set^+, nous définissons

encore deux sous variantes :

- $Set^+ = T_m$: Dans ce cas, la synchronisation se termine lorsque le premier processus maître achève son exécution alors que tous les processus maître ont déjà commencé leur exécution. Nous appelons ce type de pattern le *Master Open CoEnd*.
- $Set^+ = T_r$: La fin d'un rendez-vous est décidée lorsque le premier processus se termine, alors que tous les processus ont déjà commencé leur exécution. Nous appelons ce sous pattern le *Extended Open CoEnd*.

En considérant, en plus, les paramètres temporels associés (δ^-, δ^+), différentes sous classes du *CoEnd* peuvent être relaxées en couvrant de nouveaux patterns ou des patterns de classes déjà énoncées plus haut.

- Si $(\delta^+ = 0)$: Cela signifie que la synchronisation se termine lorsque le dernier processus se termine. Nous appelons ce pattern le *synchronous CoEnd*. Suivant la sous classe du pattern, nous obtenons : le *Synchronous Close CoEnd* (SCCE), le *Synchronous Master Open CoEnd* (SMOCE) et le *Synchronous Extended Open CoEnd* (SEOCE).
- Si $(\delta^+ \neq 0)$: Dans ce cas, différentes sous variantes du pattern sont définies :
  - Si $(\delta^- \in [0, +\infty[)$ ou $(\delta^+ \in]0, +\infty[)$: Dans ce cas , le *CoEnd* est relaxé pour permettre que le pattern *Synchronous CoEnd* soit retardé dans le temps, mais pas plus tard que le pattern *One-End* associé. Nous appelons ce pattern le *delayed CoEnd*. Selon la sous classe à laquelle le pattern appartient, Nous obtenons le *Delayed Close CoEnd* (DCCE), le *Delayed Master Open CoEnd* (DMOCE) et le *Delayed Extended Open CoEnd* (DEOCE).
  - Si $(\delta^- = +\infty)$ et $(\delta^+ = +\infty)$: Dans ce cas, le *Synchronous CoEnd* est relaxé au maximum en retardant la fin de l'exécution et obtenir le pattern *One-End* associé.

Les différentes variantes *CoEnd* sont illustrées dans la Figure 5.3.b. Le SCCE se termine lorsque tous ses processus maîtres sont prêts à se terminer conjointement (t_{m1} est le premier processus maître qui se termine en forçant t_{m2} à se terminer). Sa durée nominale dépend du processus qui le commence, soit le dernier processus commençant son exécution (t_{m2}), ou étendue au premier processus qui commence (t_{m1}). Le SCCE peut être retardé mais pas plus tard que le COE, pour spécifier le pattern DCCE.

Les patterns *Master Open CoEnd* (i.e, SMOCE, DMOOE et MOOE) sont similaires aux patterns *Close CoEnd* (i.e, SCCE, DCCE, COE) associés dans la façon de se terminer. Cependant, ils diffèrent par le processus qui les commencent, un Master pour les premiers et un processus quelconque pour les seconds.

Le SEOCE s'achève lorsque tous les processus peuvent se terminer conjointement (lorsque t_{s1} se termine), et sa terminaison peut être retardée mais pas plus tard que l'achèvement du dernier processus (t_{s2}), pour simuler le pattern DEOCE. Sa durée nominale est délimitée par le dernier processus maître qui commence son exécution (t_{m2} si $Loc^- = (T_m, MAX)$) et peut être avancée au début de t_{m1}, si $Loc^- = (T_m, MIN)$.

Exemple : Dans note exemple de réservation, nous considèrons les tâches suivantes pour une réservation nationale, T_m= { *Book-hotel, Book-Flight, Book-Car* } et T_s= {*Insu-trip, Insu-Loss* }.

Dans le cas où le client ne veut pas contracter d'assurance ($T_s = \emptyset$), mais préfère seulement réserver un hôtel et une voiture durant son séjour, l'opérateur peut décider de procéder avec toutes les tâches, en même temps et voir les disponibilités. Donc, la réservation se termine lorsque toutes les taches sont achevées conjointement. Dans ce cas on applique le pattern SCCE. Dans le cas où la date de réservation de la chambre ne matche pas avec le séjour, l'opérateur peut retarder la fin de réservation et attendre la libération d'une chambre. Ce processus peut être désigné comme un pattern DCCE. Cependant, dans le cas où le client veut rajouter une assurance à son voyage, deux cas peuvent se produire : S'il n'est pas sûr de payer une assurance via l'agence de voyage, il aura besoin de confirmer sa décision avant la fin de toutes les tâches. L'opérateur pourra procéder a ces tâches en une seule requête sans inclure une assurance, elle pourra être prise séparément en cas où le client confirme sa décision avant la fin du temps nécessaire à cette procédure. Ce processus peut être spécifié par le pattern SMOCE.

Si en plus, les dates de réservation de chambre sont modifiables pour matcher avec la date du séjour, nous procédons avec le pattern DMOCE. D'un autre coté, si le client a besoin d'une assurance via l'agence de voyage, l'opérateur peut décider d'inclure cette dernière dans la même requête que les autres tâches. La réservation est alors achevée lorsque toutes les tâches sont procédés conjointement selon les besoins du client ; le pattern SEOCE est considéré dans ce cas. Si le processus a besoin d'être retardé pour attendre la libération d'une chambre, alors le pattern DEOCE est considéré.

Rules	Loc^-	Loc^+	(Loc^-, Loc^+)	$(\alpha^-, \alpha^+, \delta^-, \delta^+)$	**Cons**
Non master rules	$T_s = \emptyset$				
AND	SCCB	SCCE	$((T_m, MAX), (T_m, MIN))$	$(0, ?, ?, 0)$	No
Delayed AND	SCCB	DCCE	$((T_m, MAX), (T_m, MIN))$	$(0, ?, ?, > 0)$	Yes
Weak-AND	SCCB	COE	$((T_m, MAX), (T_m, MAX))$	$(0, ?, ?, ?)$	Yes
Open-AND	ACCB	DCCE	$((T_m, MAX), (T_m, MIN))$	$(> 0, ?, ?, > 0)$	Yes
Advanced And	ACCB	SCCE	$((T_m, MAX), (T_m, MIN))$	$(> 0, ?, ?, 0)$	Yes
Advanced WeakAnd	ACCB	COE	$((T_m, MAX), (T_m, MAX))$	$(> 0, ?, ?, ?)$	Yes
Or	COB	COE	$((T_m, MIN), (T_m, MAX))$	$(?, ?, ?, ?)$	Yes
Or-AND	COB	SCCE	$((T_m, MIN), (T_m, MIN))$	$(?, ?, ?, 0)$	Yes
Delayed Or-AND	COB	DCCE	$((T_m, MIN), (T_m, MIN))$	$(?, ?, ?, > 0)$	Yes
Master rules	$T_s \neq \emptyset$				
Master-AND	SMOCB	SMOCE	$((T_m, MAX), (T_m, MIN))$	$(0, ?, ?, 0)$	No
Del Master-AND	SMOCB	DMOCE	$((T_m, MAX), (T_m, MIN))$	$(0, ?, ?, > 0)$	Yes
Strong Weak Master-AND	SMOCB	EOOE	$((T_m, MAX), (T_r, MAX))$	$(0, ?, ?, ?)$	Yes
Del Strong Master-AND	SMOCB	DEOCE	$((T_m, MAX), (T_r, MIN))$	$(0, ?, ?, > 0)$	Yes
Weak Master-AND	SMOCB	MOOE	$((T_m, MAX), (T_m, MAX))$	$(0, ?, ?, ?)$	Yes
Strong Master-AND	SMOCB	SEOCE	$((T_m, MAX), (T_r, MIN))$	$(0, ?, ?, 0)$	No
Master-Or-AND	MOOB	SMOCE	$((T_m, MIN), (T_m, MIN))$	$(?, ?, ?, 0)$	Yes
Del Master-Or-AND	MOOB	DMOCE	$((T_m, MIN), (T_m, MIN))$	$(?, ?, ?, > 0)$	Yes
Strong Master-Or-AND	MOOB	SEOCE	$((T_m, MIN), (T_r, MIN))$	$(?, ?, ?, 0)$	No
Master-Or	MOOB	MOOE	$((T_m, MIN), (T_m, MAX))$	$(?, ?, ?, ?)$	Yes
Strong Master-Or	MOOB	EOOE	$((T_m, MIN), (T_r, MAX))$	$(?, ?, ?, ?)$	Yes
Del Strong Master-Or-AND	MOOB	DEOCE	$((T_m, MIN), (T_r, MIN))$	$(?, ?, ?, > 0)$	Yes
Del Strong AND-Master	SEOCB	DMOCE	$((T_r, MAX), (T_m, MIN))$	$(0, ?, ?, > 0)$	Yes
Strong AND-Master	SEOCB	SMOCE	$((T_r, MAX), (T_m, MIN))$	$(0, ?, ?, 0)$	No
Strong Weak AND-Master	SEOCB	MOOE	$((T_r, MAX), (T_m, MAX))$	$(0, ?, ?, ?)$	No
Strong Open AND-Master	AEOCB	DMOCE	$((T_r, MAX), (T_m, MIN))$	$(> 0, ?, ?, > 0)$	Yes
Adv Strong AND-Master	AEOCB	SMOCE	$((T_r, MAX), (T_m, MIN))$	$(> 0, ?, ?, 0)$	Yes
Adv Strong Weak AND-Master	AEOCB	MOOE	$((T_r, MAX), (T_m, MAX))$	$(> 0, ?, ?, ?)$	Yes
Del Strong OR-AND Master	EOOB	DMOCE	$((T_r, MIN), (T_m, MIN))$	$(?, ?, ?, > 0)$	Yes
Strong OR-AND Master	EOOB	SMOCE	$((T_r, MIN), (T_m, MIN))$	$(?, ?, ?, 0)$	Yes
Strong Or-Master	EOOB	MOOE	$((T_r, MIN), (T_m, MAX))$	$(?, ?, ?, ?)$	Yes
Adv Master-AND	AMOCB	SMOCE	$((T_m, MAX), (T_m, MIN))$	$(> 0, ?, ?, 0)$	Yes
Master Open-AND	AMOCB	DMOCE	$((T_m, MAX), (T_m, MIN))$	$(> 0, ?, ?, > 0)$	Yes
Adv Strong Master-AND	AMOCB	SEOCE	$((T_m, MAX), (T_r, MIN))$	$(> 0, ?, ?, 0)$	Yes
Adv Weak Master-AND	AMOCB	MOOE	$((T_m, MAX), (T_m, MAX))$	$(> 0, ?, ?, ?)$	Yes
Adv Strong Weak Master-AND	AMOCB	EOOE	$((T_m, MAX), (T_r, MAX))$	$(> 0, ?, ?, ?)$	Yes
Strong Master Open-AND	AMOCB	DEOCE	$((T_m, MAX), (T_r, MIN))$	$(> 0, ?, ?, > 0)$	Yes

TABLE 5.1 – Règles de synchronisation .

5.4 Les Règles génériques d'un *Rendez-vous*

Comme expliqué, un rendez-vous suit une stratégie pour commencer sa synchronisation et une autre pour la terminer. En combinant les patterns vus plus haut, nous introduisons des règles génériques qui fixent et ajustent toutes les sémantiques d'un rendez-vous.

La Table.5.1 énumère l'ensemble des combinaisons de patterns avec les règles associées. Il est claire que certaines stratégies ne peuvent pas être combinées, comme par exemple *extended One-Begin* avec *extended One-End*. Ceci est dû au fait que nous avons besoin qu'au moins une des deux stratégies soit conduite par des processus maîtres.

Concrètement, si tous les processus ont le même privilège ($T_s = \emptyset$), alors nous pouvons obtenir 9 règles. Ces dernières sont obtenues en considérant la stratégie Loc^- de { SCCB, ACCB, COB } avec la stratégie Loc^+ de { SCCE, DCCB, COE }. Nous appelons ces règles, les *non master rules.*

Par ailleurs, si les processus n'ont pas le même privilège ($T_s \neq \emptyset$), nous définissons, ce qu'on identifie comme les *Master rules*. Les stratégies Loc^- de { SMOCB, AMOCB, MOOB } peuvent être obtenues en combinant avec n'importe quelle des 6 autres stratégies Loc^+ ; ceci mène à 18 règles maîtres possibles.

Cependant, la stratégie Loc^- de { SEOCB, AEOCB, EOOB } peut être combinée seulement avec { SMOCE, DMOCE, MOOE }, incluant 9 règles maîtres en plus. En tout, nous générons 27 règles du type maître parmi les 36 règles identifiables. Cependant, certaines règles peuvent se simplifier en considérant des combinaisons différente des paramètres $(\alpha^-, \alpha^+, \delta^-, \delta^+)$.

5.5 Discussion et Comparaison

Les règles du type *AND* dénotent un rendez-vous où les stratégies *Cobegin* ou *CoEnd* sont considérées. Les règles du type *Or* sont ceux qui adoptent la stratégie *One-begin*. Par ailleurs, les règles du type *Weak* appliquent la stratégie *One-End*, alors que les règles du type *Strong* sont celles conduites par des processus esclaves dans l'une des deux stratégies combinées.

Notre modèle	**Le modèle d'ALLEN**	**Le modèle de Wah**	**Le modèle basé** $TSPN$
AND	Oui	Oui	Oui
Weak-AND	Oui	Oui	Oui
OR	Oui	Oui	Oui
Advanced-AND	NON	Oui	Non
Open-AND	NON	Oui	Non
Delayed-AND	NON	Oui	Non
Master-Or	NON	NON	Oui
Master-AND	NON	NON	Oui
Strong Master	NON	NON	Oui
Master Open-AND	NON	NON	NON
Master Or-AND	NON	NON	NON
Master Open-AND	NON	NON	NON

TABLE 5.2 – Comparaison .

Les règles identifiées comme *Advanced*, sont celles avec ($\alpha^- > 0$). Ceci dénote le début de la synchronisation qui est avancé par rapport au début de la règle originale (e.g. le *advanced AND* commence avant le *AND*). D'un autre coté, les règles notées comme *Delayed* sont celles avec ($\delta^+ > 0$). Ceci spécifie que la fin de la synchronisation est retardée relativement à la fin de la règle originale (e.g The *Delayed AND* se termine après le *AND*).

Cependant, les règles du type OPEN sont celles avec ($\alpha^- > 0$ et $\delta^+ > 0$), c-à-dire le début de la synchronisation est avancé et sa terminaison est retardée relativement à la règle originale du début (e.g le *Open AND* commence avant et se termine après *AND*).

Finalement, l' exécution d'une règle est dite critique ou *Critical* si le commencement au plus tôt et la terminaison au plus tard de la synchronisation se réfère à la même date. La durée de synchronisation dans ce cas est atomique et ne peut pas être étendue. Plus encore, l'exécution des règles inconsistantes se fait si la date au plus tôt de commencer ne peut avoir lieu avant la fin de la date au plus tard de la synchronisation. Dans ce cas, la synchronisation ne peut avoir lieu. Ceci apparaît par exemple dans la règle AND lorsque les intervalles de temps associés aux processus maîtres ne se chevauchent pas. La colonne *Const* de la Table.5.1 identifie les règles qui sont toujours consistantes quelque soit la contrainte du temps des processus maîtres.
Par ailleurs, le symbole

"?"

dans la colonne paramètres temporels désigne n'importe quelle valeur rationnelle positive ou nulle.

La table 5.2 montre quelques règles générées par notre modèle et leur position dans les autres modèles de la littérature. La mention 'Oui' dénote que la règle en question existe, la mention 'Non' dénote que cette règle n'existe pas. Exemple, les règles AND, Weak-AND et Or sont des règles qui existent dans chacun des modèles : Allen, Wah et basé $TSPN$. D'autre part, les règles Advanced AND, Open AND existent dans Wah mais pas pour les modèles d'Allen et celui basé $TSPN$. Pour les règles de type master, certaines sont générées par le modèle $TSPN$ (les modèles d'Allen et Wah ne sont pas concernées), on peut citer : Master Or, Master AND et d'autres sont nouvelles comme Master Open-AND et Master Or-AND.

Finalement, cette comparaison nous permet de dire que notre modèle présente un schéma de synchronisation qui couvre un grand nombre de règles de synchronisation définies dans la littérature. En effet, notre modèle couvre les 10 règles du modèle $TSPN$ [57], les 7 règles issues de la théorie d' *Allen* [30] et les règles avec des paramètres du temps de [31].

5.6 Conclusion

Nous avons introduit dans ce chapitre notre modèle de synchronisation avec contraintes temporelles, appelé *rendez-vous*.

Nous introduisons formellement dans le chapitre qui suit, une nouvelle extension des réseaux de Petri temporels qui exploite la sémantique du concept de *rendez-vous*. Nous montrons comment exploiter ce nouveau cadre formel pour la spécification des systèmes Workflow.

Chapitre 6

Les réseaux de Petri Temporels avec Rendez-vous : $RTPN$

6.1 Introduction

Nous introduisons dans ce chapitre une nouvelle extension des *RdpT*, notée $RTPN$, pour (*Time Petri Net with Rendezvous*). Ce nouveau formalisme est dédié à la spécification et l'analyse des systèmes temps réel complexes. Cette proposition synthétise l'étude présentée dans le chapitre précédent, et tente de proposer un modèle général permettant de traiter une large frange d'applications englobant entre autres, celles dédiées aux systèmes nécessitant des schémas de synchronisations complexes à l'image des systèmes *workflow*. Nous présentons d'abord la syntaxe du modèle, suivie de sa sémantique formelle.

6.2 Syntaxe formelle d'un $RTPN$

Le modèle $RTPN$ (*Time Petri Net with Rendezvous*), est une nouvelle extension des *RdPT*s que nous avons proposée dans [112, 68] incluant la notion de synchronisation et de délai. Cette extension est basée sur la notion de *rendez-vous* présentée et détaillée tout au long du chapitre précédent.

Formellement, la syntaxe d'un $RTPN$ est définie comme suit :

Définition 6.2.1. ***(Réseau de Petri temporel avec Rendez-vous)****[112, 18]. Un RTPN est un doublet* $(QuadT, RDV_s)$ *où :*

- $QuadT$ *est un RdPT marqué,* $(P, T, Pre, Post, M_0, FI)$ *;*
- RDV_s *est l'ensemble des rendez-vous. Un rendez-vous* R *de* RDV_s *est un schéma de synchronisation sous la forme suivante :* $(T_m, T_s, (Loc^-, Loc^+), (\alpha^-, \alpha^+, \delta^-, \delta^+))$*, où :*
  - T_m *et* T_s *sont deux sous-ensembles de transitions de* T*, tels que* $T_m \cap T_s = \emptyset$ *et nous notons* $T_r = T_m \cup T_s$*.* T_m *est un ensemble fini non vide de transitions Masters, et* T_s *un ensemble fini de transitions esclaves.*
  - Loc^-, Loc^+ *sont les localités considérées dans un rendez-vous. Nous avons* $Loc^- = (Set^-, Op^-)$ *et* $Loc^+ = (Set^+, Op^+)$*, tel que* $Set^-, Set^+ \in \{T_m, T_r\}$ *et* $Op^-, Op^+ \in \{MIN, MAX\}$*, avec au moins un des paramètres* Set^- *et* Set^+ *soit égale à* T_m*.*
  - *Finalement,* $\alpha^-, \alpha^+, \delta^+$ *et* δ^- *sont des paramètres de délai qui prennent leurs valeurs dans* $Q^+ \cup \{+\infty\}$*.*

Dans un $RTPN$, une transition peut contribuer au tir de plusieurs rendez-vous. Cependant, la sélection du rendez-vous à franchir se fait de manière indéterminée. Cependant, un mécanisme de priorité sur les rendez-vous pourrait être rajoutée comme paramètre pour résoudre le problème du non déterminisme. Dans le cas où une transitions n'est évoquée par aucun rendez-vous, la transition évolue de manière asynchrone car elle n'est sujette par aucun schéma de synchronisation prédéfinie.

Considérons le *RdpT* de la Figure.6.1, étendu à l'ensemble des rendez-vous suivants : $RDV_s = \{R_1, R_2, R_3, R_4\}$, tel que :
$R_1 = (\{t_0, t_1\}, \emptyset, ((\{(t_0, t_1\}, MAX), (\{t_0, t_1\}, MIN)), (0, 0, 0, 0))$
$R_2 = (\{t_2\}, \{t_1\}, ((\{t_1, t_2\}, MIN), (\{t_2\}), MIN), (+\infty, +\infty, +\infty, 0))$
$R_3 = (\{t_6\}, \{t_2\}, ((\{t_6\}, MAX), (\{t_2, t_6\}, MAX)), (0, +\infty, +\infty, +\infty))$

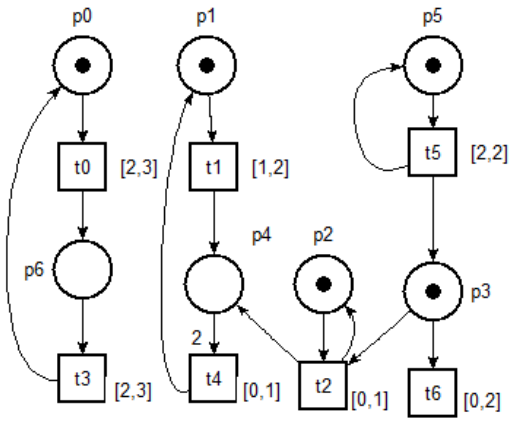

FIGURE 6.1 – Exemple d'un RTPN

$R_4 = (\{t_3, t_4\}, \emptyset, (\{t_3, t_4\}, MIN), (\{t_3, t_4\}, MIN)), (+\infty, +\infty, +\infty, 0))$
$R_5 = (\{t_3\}, \emptyset, (\{t_3\}, MAX), (\{t_3\}, MIN)), (0, 0, 0, 0))$
$R_6 = (\{t_4\}, \emptyset, (\{t_4\}, MAX), (\{t_4\}, MIN)), (0, 0, 0, 0))$

D'après la sémantique des règles génériques des rendez-vous, les rendez-vous R_1, R_5, R_6 suivent la règle *And* et R_2 et R_3 considèrent respectivement, les règles *Strong Or-AND-Master* et *Strong Weak AND-Master* . Finalement, R_4 présente un *Or-AND*.

Dans l'exemple, les transitions t_1 et t_2 sont évoquées dans deux rendez-vous différents. t_1 peut progresser suivant la synchronisation définie par R_1 ou R_2, tandis que t_2 peut progresser suivant R_2 ou R_3. De la même manière les transitions t_3 et t_4 peuvent synchroniser leur exécution suivant le rendez-vous R_4 ou progresser de façon asynchrone suivant respectivement, les rendez-vous R_5 et R_6. Finalement, comme t_5 n'est évoquée dans aucun rendez-vous prédéfini, elle doit progresser de manière asynchrone.

6.3 Sémantique formelle d'un $RTPN$

Un $RTPN$ évolue par le tir d'un *rendez-vous* en chaque étape. Ceci implique le tir de toutes les transitions du rendez-vous indiquant que certaines conditions sont satisfaites. Le tir d'un rendez-vous dépend du marquage et des règles de synchronisation qui mènent vers les contraintes temporelles dynamiques du modèle.

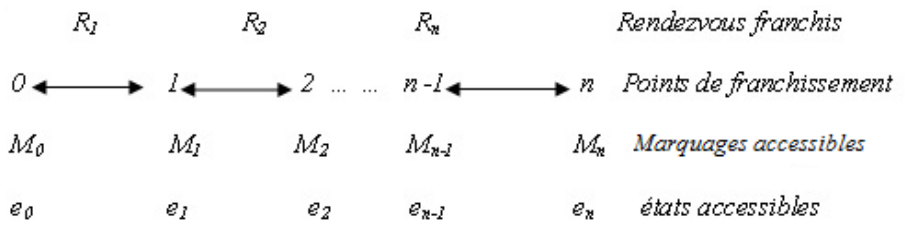

FIGURE 6.2 – Evolution d'un $RTPN$

Avant de poser le sémantique formelle du modèle. Nous présentons d'abord, quelques notations :

Soit $RT := (P, T, Pre, Post, FI_s, M_0, RDV_s)$ un $RTPN$ et
$R := (T_m, T_s, (Loc^-, Loc^+), (\alpha^-, \alpha^+, \delta^-, \delta^+))$ un rendez-vous de RDV_s.

- Nous notons par $Tr(R) = T_r$ l'ensemble des transitions définies dans le rendez-vous R.
- Soit l'ensemble des rendez-vous asynchrones, noté RDV_a. $R_a \in RDV_a$ sous forme : $(\{t\}, \emptyset, ((\{t\}, MAX), (\{t\}, MIN)), (0, +\infty, +\infty, 0))$ tel que t est une transition de T qui n'est évoquée par aucun rendez-vous prédéfini, $\forall R \in RDV_s, t \notin Tr(R)$.
- L'ensemble, noté RDV, définit tous les rendez-vous potentiellement tirables du modèle, nous notons : $RDV = RDV_s \cup RDV_a$.

— Pour rappel, $E(M)$ désigne l'ensemble des transitions *sensibilisées* pour un marquage M au sens classique des *RdPT*.
— Un *rendez-vous* R est dit sensibilisé pour un marquage M, si toutes ses transitions sont sensibilisées pour M et nous notons par $ER(M)$ l'ensemble de tous les *rendez-vous* de RDV sensibilisés pour M. $ER(M) = \{R \in RDV \mid Tr(R) \subseteq E(M)\}$.
— Soit M un marquage, et soient t_i et t_j deux transitions sensibilisées pour M. t_i et t_j sont en *conflit* pour M, si $\exists p \in P, \quad pre(p,t_i) + pre(p,t_j) > M(p)$. Nous notons par $Conf(M)$ la relation construite sur $E(M)^2$ telle que $(t_i,t_j) \in Conf(M)$, ssi t_i et t_j sont en conflit pour le marquage M. Un rendez-vous R est en conflit avec lui-même pour M, s'il existe deux transitions t_i, $t_j \in Tr(R)$ en conflit pour M.
— Soit M un marquage, deux *rendez-vous* R_i et R_j sensibilisés par M sont en *conflit* pour M, s'il existe $t_i \in Tr(R_i)$ et $t_j \in Tr(R_j)$ tel que, $(t_i,t_j) \in Conf(M)$.

Il est à noter que si deux transitions (au moins), d'un même rendez-vous R sont en conflit , alors R ne peut pas être tiré. C'est le cas de deux processus en conflit pour une même ressource et qui ont besoin de synchroniser leur exécution.

Pour illustrer les définitions précédentes, revenons à notre exemple de la Figure 6.1. L'ensemble des rendez-vous RDV_s est étendu avec l'ensemble des rendez-vous asynchrones :
$R_7 := (\{t_5\}, \emptyset, ((\{t_5\}, MAX), (\{t_5\}, MIN)), (0, +\infty, +\infty, 0))$.
Le marquage initiale est donné par M_0 : $p_0, p_1, p_2, p_3, p_5 \mapsto 1; p_4, p_6 \mapsto 0$. L'ensemble des transitions sensibilisées est $E(M_0) = \{t_0, t_1, t_2, t_5, t_6\}$. L'ensemble des rendez-vous sensibilisés est $ER(M_0) = \{R_1, R_2, R_3, R_7\}$.

Comme nous pouvons le remarquer, la transitions t_2 et t_6 sont en conflit pour M_0, et le rendez-vous R_3 est en conflit avec lui même et ne peut être tiré à partir de M_0. De plus, les rendez-vous R_2, R_3 sont en conflit pour M_0, le tir de l'un désensibilise l'autre .

Comme pour les $RDPT$, des sémantiques différentes peuvent être considérées dans un $RTPN$:

(i) La sémantique *mono-serveur* : Quelque soit le marquage, une seule instance de transition et par extension, de rendez-vous peut être sensibilisée pour un marquage ;

(ii) La sémantique *multi-serveur* : Qui considère qu'une transition et donc, un rendes-vous, peuvent être sensibilisées plus d'une fois pour un marquage donné [98][97].

Dans la suite de notre travail, nous considérons la sémantique mono-serveur au lieu de la sémantique multi-serveur.Concrètement, si le marquage actuel peut sensibiliser plusieurs instances d'une même transition et d'un même rendez-vous, seulement une instance est considérée. La sémantique que nous formulons peut être adaptée pour considérer d'autres politiques de tir comme celle du multi-serveur. Assumant ceci, nous introduisons ci-après l'expression formelle d'un état accessible.

Définition 6.3.1. ***(Etat accessible)****. Un état accessible, noté e, d'un RTPN est le tuple $e = (M, V)$, où M est le marquage accessible et V la fonction qui associe à chaque transition sensibilisée son intervalle dynamique de tir $V(t) := [x(t), y(t)]$. Nous notons $e_0 = [M_0, V_0]$ l'état initial, où :$V_0(t) := IF_s(t)$.*

L'état initial de notre exemple $RTPN$ est donné par (M_0, V_0), tel que $V_0 : t_0 \mapsto [2,3]; t_1 \mapsto [1,2]; t_2 \mapsto [0,1]; t_5 \mapsto [2,2]; t_6 \mapsto [0,2]$.

Soit l' état accessible e d'un $RTPN$, l'intervalle de tir d'un rendez-vous sensibilisé est donné comme suit :

Définition 6.3.2. ***(L'intervalle de tir d'un rendez-vous)****.*

Soit $e = (M, V)$ un état accessible dans un RTPN. Pour chaque rendez-vous R, tel que $R = (T_m, T_s, (Loc^-, Loc^+), (\alpha^-, \alpha^+, \delta^-, \delta^+))$ où : $Loc^- = (Set^-, Op^-)$ et $Loc^+ = (Set^+, Op^+)$. Nous définissons l'intervalle de tir de R, comme suit :

$$[Low(R), Up(R)] :=$$
$$\left[MAX \left\{ \begin{array}{l} \underset{\forall t \in Set^-}{Op^-} \{x(t)\} \quad - \alpha^- \\ MIN \left(\underset{\forall t \in Set^+}{Op^+} \{y(t)\} - \alpha^+, \quad \underset{\forall t \in Set^-}{Op^-} \{x(t)\} \right) \\ \underset{\forall t \in Set^-}{MIN} \{x(t)\} \end{array} \right. , \right.$$
$$\left. MIN \left\{ \begin{array}{l} \underset{\forall t \in Set^+}{Op^+} \{y(t)\} \quad + \delta^+ \\ MAX \left(\underset{\forall t \in Set^-}{Op^-} \{x(t)\} + \delta^-, \underset{\forall t \in Set^+}{Op^+} \{y(t)\} \right) \\ \underset{\forall t \in Set^+}{MAX} \{y(t)\} \end{array} \right. \right]$$

$[Low(R), Up(R)]$ détermine l'intervalle de temps relatif durant lequel le *rendez-vous* R doit être tiré en respectant la synchronisation considérée. Cet intervalle est calculé à partir des intervalles dynamiques de tir des transitions $Tr(R)$ selon la valeur des localités et leurs paramètres de délais qui lui sont associés. A titre d'exemple, si nous appliquons cette règle générale sur notre $RTPN$:

- R_1 qui adopte la règle *AND* peut être tiré dans $\left[\underset{\forall t\in\{t_0,t_1\}}{MAX}\{x(t)\}, \underset{\forall t\in\{t_0,t_1\}}{MIN}\{y(t)\}\right]$ =[2,2].

- Pour R_2, qui adopte un *Strong Or-AND-Master* peut être tirée dans $\left[\underset{\forall t\in\{t_1,t_2\}}{MIN}\{x(t)\}, y(t_2)\right]$=[0,1].

- Ensuite , R_3 qui considère un *Strong Weak AND-Master* peut être tiré dans $\left[x(t_6), \underset{\forall t\in\{t_2,t_6\}}{MAX}\{y(t)\}\right]$ =[0,2].

- Finalement, le rendez-vous asynchrone R_7 peut être tiré dans $[x(t_5), y(t_5)] = [2, 2]$.

Parmi les rendez-vous R_3, R_1, R_2 et R_7 sensibilisés pour M_0, un seul peut être franchi. Nous formulons dans ce qui suit, une condition suffisante et nécessaire pour le tir d'un rendez-vous.

Définition 6.3.3. ***(Condition de tir d'un rendez-vous)***.

Un rendez-vous sensibilisé R_f peut être tiré à partir d'un état accessible $e = (M, V)$, à une date relative d, ssi :

(i) R_f n'est pas en conflit avec lui même ;

(ii) R_f a atteint sa borne inférieure : $0 \leq Low(R_f) \leq d$;

(iii) Aucun autre rendez-vous sensibilisé n'a dépassé sa borne supérieur : $d \leq \underset{R\in ER(M)}{MIN}\{Up(R)\}$;

Dans notre $RTPN$, seulement R_2 peut être tiré à partir de l'état initiale e_0. En effet, R_3 est en conflit avec lui même et ne peut être tiré . Par ailleurs, le temps ne peut progresser pour atteindre la borne inférieure de R_1 et R_7 sans surpasser celle de R_2.

Nous introduisons maintenant la sémantique formelle d'un $RTPN$ donnée sous forme d'un système de transitions d'états.

Définition 6.3.4. ***(Sémantique d'un** $RTPN$**)*** *[112, 18] La sémantique d'un $RTPN$ est définie comme un système de transition $\boldsymbol{ST} = (\Gamma, e_0, \rightarrow)$, tel que :*

- Γ *est l'ensemble des états accessibles ;*
- $e_0 = (M_0, V_0)$ *est l'état initial ;*
- $\rightarrow: \Gamma\times(T\times\mathbb{Q}^+)\times\Gamma$ *est la relation de transition entre état tel que :* $((M, V), (R_f, d), (M^\uparrow, V^\uparrow)) \in \rightarrow$, *si et seulement si, les conditions de tir de la définition 6.3.3 sont satisfaites, et nous avons :*
  - $(\forall p \in P, \forall t \in Tr(R))$, $M^\uparrow(p) := M(p) - Pre(p, t) + Post(p, t)$
  - *La fonction $V^\uparrow$ est calculée, comme suit :*
    1. *éliminer les transitions désensibilisées dans $M^\uparrow$*
    2. *Pour chaque transition t sensibilisée pour M et qui reste sensibilisée pour $M^\uparrow$, décaler $V(t)$ de la valeur du temps écoulé d, vers la nouvelle origine temporelle, et tronquer si nécessaire au niveau de sa borne inférieure à une valeur non négative $V^\uparrow = [MAX(0, x(t) - d), y(t) - d]$.*
    3. *Ajouter les transitions nouvellement sensibilisées pour $M^\uparrow$ en leur assignant leur intervalle statique : $V^\uparrow(t) = IF_s$.*

Le tir d'un rendez-vous R_f implique le tir de toutes ses transitions (i.e., $t \in Tr(R_f)$), ce qui mène à un nouvel état accessible $e^\uparrow$, où le marquage correspondant $M^\uparrow$ est obtenu à partir de M après le tir des transitions de $Tr(R_f)$.

Il est à noter que la sémantique d'un $RTPN$ autorise que la progression du temps puisse dépasser la borne supérieure de l'intervalle dynamique d'une transition sensibilisée ($y(t) < 0$), ce qui n'est pas le cas d'un *RdpT*. Ceci dénote que le processus en question a terminé son exécution alors que la synchronisation n'a pas encore eu lieu. Concrètement, dans un $RTPN$ les contraintes temporelles des transitions sensibilisées suivent une sémantique de temps faible *weak semantics*, tandis que celle des rendez-vous obéit à une sémantique de temps forte *strong semantics*.

Considérons le tir du rendez-vous R_2 dans notre exemple à l'instant $d = 1$ à partir de l'état e_0 pour accéder au nouvel état $e_1(M_1, V_1)$, tel que le nouveau marquage est : $M_1 : p_0, p_2, p_4, p_5, \mapsto 1; p_1, p_3, p_6 \mapsto 0$. Par conséquent, l'ensemble des transitions sensibilisées pour M_1 est $E(M_1) = \{t_0, t_5\}$ et nous avons :
$V_1 : t_5 \mapsto [2 - 1, 2 - 1] = [1, 1]; t_0 \mapsto [2 - 1, 3 - 1] = [1, 2]$.
Seul R_5 est sensibilisé et peut être tiré à l'instant 2 à partir de l'état e_1.

Jusqu'à lors nous nous sommes investis dans la définition d'un nouveau modèle permettant d'étendre les RdPT aux sémantiques permettant de capturer les mécanismes de synchronisation. Nous explorons maintenant, comment exploiter ce nouveau modèle pour la spécification des applications complexes de workflow en exploitant les mécanismes nouvellement introduits.

6.4 Application : les $RTPN$ et les systèmes Workflow

Dans cette section, nous introduisons une sous-classe des $RTPN$, notée $RTWFN$ (*Timed Workflow nets with rendezvous*). Le modèle est dédié à la spécification des systèmes de workflow. Nous présentons ensuite, la base de notre approche de modélisation pour les systèmes de workflow.

Définition 6.4.1. ***(Réseau de Workflow temporel avec Rendez-vous)***. *Soit, RT_w un $RTWFN$. RT_w est un tuple (RT, p_b, p_e), tel que :*

- *$RT = (P, T, Pre, Post, M_0, IF_s, RDV_s)$ est un $RTPN$.*
- *p_b est une place particulière de P notée place de départ (début) du réseau de workflow, et nous avons :*
  $\bullet p_b = \emptyset$ et $M_0(p_b) \neq 0$;
- *p_e est une place particulière de P notée place d'arrivée (fin) du réseau de workflow, et nous avons : $p_e\bullet = \emptyset$ et $M_0(p_e) = 0$, où $\bullet p$ dénote l'ensemble des transitions entrantes de la place p tandis que $p\bullet$ représente l'ensemble des transitions sortantes de p.*

La place p_b dénote la source du réseau tandis que la place p_e représente le puits du réseau. Le $RTWFN$ doit vérifier qu'il existe un chemin du marquage initial incluant la place p_b vers le marquage final incluant la place p_e ; le réseau est dit *fortement connexe*.

Le modèle $RTWFN$ du workflow entier contraint par la synchronisation et les délais peut être obtenu selon l'approche suivante :

(a) Premièrement, nous créons les places p_e et p_b.

(b) *Tâche unique/individuelle :* Chaque unité élémentaire (*tâche*) est traduite par une transition $t \in T$ et une place d'entrée $p \in P$. Pour les contraintes temporelles, un intervalle de temps est associé à chaque transition représentant la tâche $I(t) = [x(t), y(t)]$. Ceci défini la date au plus tôt et la date au plus tard d'une tâche. Si aucune contrainte temporelle n'est imposée ; nous avons $I(t) = [0, +\infty]$. Dans le cas d'une tâche critique $I(t) = [0, 0]$, t ne peut être retardée et doit avoir lieu aussitôt que sa place d'entrée est marquée (Voir Figure 6.3.x). Si la tâche est la première dans le processus alors sa place d'entrée est p_b. Si c'est la dernière dans le réseau du workflow alors sa place de sortie est p_e ;

(c) *Séquence* : Dans l'exemple de Figure 6.3.a, les tâches t_1 et t_2 sont exécutées en séquence, représentant la contrainte de précédence lors de l'exécution des tâches dans le workflow ;

(d) *Choix* : Dans Figure 6.3.b, t_1 et t_2 sont en conflit et ne peuvent pas s'exécuter en même temps ;

(e) *Concurrence* : Dans Figure 6.3.c, les tâches sont en concurrence ; ils peuvent s'exécuter en parallèle et ne sont pas en conflit. Leur exécution peut être gérée par des règles de synchronisation exprimées sous forme de rendez-vous.

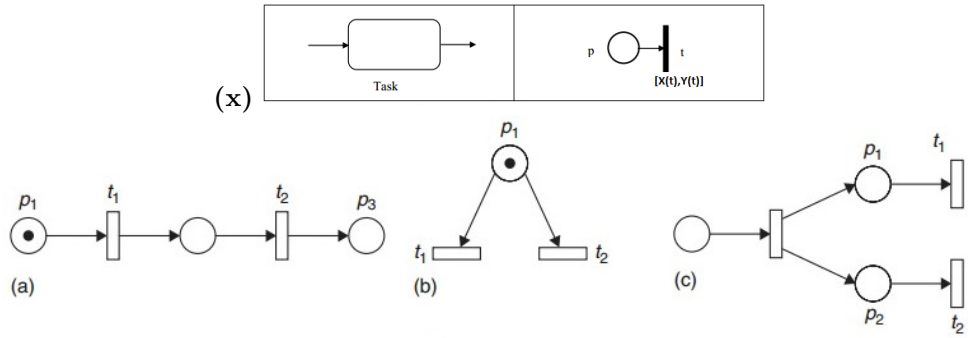

FIGURE 6.3 – L'approche de modélisation pour le $RTWFN$

6.5 Exemple : Le Workflow d'un patient

Dans cette section, nous présentons un exemple montrant la puissance d'expression du modèle $RTWFN$ dédié à la modélisation des systèmes workflow, en prenant en considération des schémas de synchronisation assez complexes (voir Fig. 6.4).

Nous considérons l'exemple du *Patient workflow*, initialement présenté dans [113, 9, 8, 18]. Ce dernier est une combinaison et adaptation d'un ensemble de workflow dans le domaine de la santé. La problématique est la suivante :

Toutes les 5 minutes (transition *patient-arrival*), un patient se présente à l'hôpital (ou une clinique privée). Nous assumons qu'en moyenne Np patients arrivent tous les jours ce qui correspond au marquage initial de la place P_b.

A la réception, il est demandé au patient de présenter sa carte d'identité et sa carte d'assurance médicale ensuite, il est invité à prendre une chaise à la salle d'attente. Suite à la réception des cartes, celles-ci sont introduites dans le système d'authentification et de vérification via les transitions (*Check-Insurance, Check-Id*). L'exécution des deux tâches doit être coordonnée car les données peuvent être partagées entre différents systèmes. Cependant, dans le cas où une tâche est en retard la coordination peut être retardée mais pas plus tard que 2 unités de temps après le temps maximal nécessaire. Le temps de latence des deux systèmes pour répondre à la requête sont respectivement $[2, 6]$ et $[3, 4]$. Pour spécifier ces besoins, nous considérons les rendez-vous suivants :

$R_0 := (T_m = \{Check - insurance, Check - id\}, \emptyset, ((T_m, MIN), (T_m, MIN)), (\infty, \infty, \infty, 2)))$.

Les règles de synchronisation associées à R_0 se réfère à une règle non master *Delayed Or-And*. Si l'assurance et la carte d'identité sont valides, le patient est admis et appelé pour être re-dirigé vers une deuxième chambre afin d'être diagnostiqué (transition *Admission*). Autrement, son admission est rejetée (transition *Reject*) et doit quitter l'hôpital. Nous assumons que le déplacement d'une chambre à l'autre dure entre 2 et 5 unités de temps.

Une fois le patient est examiné lors du premier diagnostic, une infirmière prend sa tension artérielle (transition *Obtain-vital*) pendant que le praticien interroge le patient et reporte des notes sur son dossier, (transition *Exam-Report*). Selon les notes, le médecin donne un premier diagnostic et assigne le service suivant qui prendra en charge le patient. Les deux tâches commencent en même temps et l'examen se termine lorsque la dernière tâche est achevée. Pour spécifier les contraintes précédentes, nous considérons le rendez-vous suivants associé à une règle *Weak-And* :

$R_1 := (T_m = \{Exam - Report, Obtain - vital\}, \emptyset, ((T_m, MAX), (T_m, MAX)), (0, \infty, \infty, 0)))$

Une fois l'examen achevé, le patient est orienté vers le service d'admission pour lui attribuer une chambre libre (transition *assign-bedroom*). Nous assumons que l'hôpital possède k chambres notées par le marquage de la place *bedrooms*. Si aucune chambre n'est disponible, le patient peut attendre jusqu'à ce qu'une chambre soit libérée.

Une fois le patient est admis, il prend une chambre en attendant le staff du service vers lequel il a été re-dirigé. Nous considérons dans notre cas trois services : cardio-vasculaire (ou STEMI : Segment Elevation Myocardial Infarctus), dermatologie et chirurgie. Les places Ms_1,Ms_2 et Ms_3 dénotent le nombre de patients pris en charge en parallèle dans les trois services. Nous assumons que la capacité des trois services sont respectivement s_1, s_2 et s_3 et dépendent de la disponibilité de l'équipe médicale de chaque service. Le patient ne peut être pris en charge que si l'équipe médicale est libre (transitions *SETMI-handling, Derma-Handling, Surgery-handling*).

La prise en charge STEMI nécessite deux actions thérapeutiques : En présence d'un infarctus du myocarde, un traitement fibrinolytique par une perfusion est donné (tâche Master désignée par la transition *Rep-fibr*). L'action complémentaire consiste à l'administration de béta-bloquants (dénoté par la transition esclave *Oral-beta*). Nous voulons exprimer que le traitement oral ne peut commencer plus de 1 minute avant le début de la perfusion ni finir plus de 2 unités de temps après la fin de la perfusion. Afin de modéliser ces dernières synchronisations, nous considérons le rendez-vous associé suivant avec la règle Master *Strong-Open-AND-Master*.

$R_2 := (T_m = \{rep - fibr\}, T_s = \{oral - beta\}, ((T_m, MAX), (T_m, MIN)), (1, \infty, \infty, 2)))$.

Concernant le service de dermatologie, trois actions thérapeutiques parallèles et indépendantes sont prescrites : une référence en soins primaires, médicaments anti-douleurs et actions de référence en dermatologie. Suivant le cas, une, deux ou les trois actions peuvent être demandées. Pour modéliser ces besoins, nous adoptons le rendez-vous suivant avec la règle *OR*.

$R_3 := (T_m = \{Prim - care, derma - ref, pain - medic\}, \emptyset, ((T_m, MIN), (T_m, MAX)), (\infty, \infty, \infty, \infty)))$.

Le dernier service traite des soins chirurgicaux suite à un diagnostic de fracture du fémur. Après un CT-scan, l'action suivante consiste à réaliser deux tâches principales en parallèle : planifier une chirurgie ou prise de médicaments. La dernière tâche nécessite l'administration de deux médicaments obligatoires $p - m - A$ et $p - m - B$ et un optionnel $p - m - C$. A et B doivent être pris ensemble, alors que C doit

être pris 5 unités de temps avant. Exemple : Administration d'un antibiotique avec un anti-inflammatoire et des bandages gastriques. Pour ce besoin, nous considérons le rendez-vous suivant avec la règle Master *Adv Weak-Master-AND*.

$R_4 := (T_m = \{p-m-A, p-m-B\}, (T_s = \{p-m-C\}, ((T_m, MAX), (T_m, MAX)), (5, \infty, \infty, \infty)))$.

Parallèlement à la prise de médicaments, la chirurgie est en cours de planification (transition *Surg-plan*). Après l'action chirurgicale, une période de récupération est observée.

Quelque soit le service que nous considérons, le patient est maintenu par la suite en observation dans sa chambre un certain temps une fois la prise en charge effectuée. En parallèle, son état de santé est reporté (transitions *Doc1, Doc2, Doc3*). A la fin de sa prise en charge, il est invité en caisse puis il doit quitter l'hôpital (transition *checkout*), ainsi la chambre est libérée pour l'admission d'un nouveau patient.

6.6 Comparaison

Dans cette section, nous comparons le modèle $RTPN$ vu dans la première partie du chapitre, avec d'autres modèles de la littérature (voir Fig. 6.5). Chronologiquement nous avons :

Le modèle *RdpT* [17] : Ce modèle permet l'utilisation de temporisations imparfaites en attribuant à chaque transition un intervalle de temps dans lequel doit se situer le tir de la transition, le modèle possède un bon pouvoir d'expression équivalent à celui d'une machine de Turing. Cependant, il induit une synchronisation forte entre flux, toujours conduite par le flux le plus en retard, cette considération peut entraîner le non respect des contraintes temporelles des flux les plus en avance. Le modèle *RdPT* ne permet donc pas une modélisation aisée d'un scénario de synchronisation inter-flux et présente une limite en terme de pouvoir de modélisation.

Les modèles $TSPN$[51][52] et $HTSPN$: Le modèle $TSPN$ défini pour la modélisation des systèmes faiblement synchrones semble être un modèle apparenté au $RTPN$. Par conséquent, la comparaison entre les deux modèles nous permet de faire valoir quelques différences de fond : dans un $TSPN$, les intervalles de temps sont associés aux arcs entrants. De plus, une place dans un $TSPN$ modélise un processus du système appelé flux ou stream (voir Figure.3.10). Chaque schéma de synchronisation entre les différents processus se présente par une règle associée à la transition correspondante. De là, un $TSPN$ ne peut être considéré comme une extension directe d'un *RdPT*. Aussi, il est mal adapté à la modélisation des spécifications orientées événements puisque chaque action y est modélisée comme une place. Plus encore, certains comportements ne peuvent être modélisés intuitivement avec un $TSPN$, menant à des spécifications ambigues [114]. Comme cas particulier, la modélisation des ressources d'un processus dans un $TSPN$ est assez floue, car une place y est utilisée pour la représentation d'un processus ou d'une action. Par conséquent, comme un $RTPN$ permet de prendre en compte les schémas de synchronisation définis dans un $TSPN$, et ce en plus d'autres règles et mécanismes de délai (non supportés par un $TSPN$), il offre ainsi un outil plus puissant et plus adapté pour modéliser un large panel dépassant le cadre des systèmes dits faiblement synchrones, et cela tout en préservant l'expressivité intuitive qui a permis aux *RdPT* d'être ce modèle pratique si largement utilisé.

Finalement, le modèle $STPTPN$ [1][95] est assez proche de notre modèle $RTPN$ dans le sens où les deux sont une extension des *RdpT* avec les contraintes temporelles associées aux transitions.Cependant le $RTPN$ défini un large panel de synchronisation et prend en compte les délai alors que les $STPTPN$ sont plutôt étendu vers d'autres mécanismes tels que la suspension du temps et les mécanismes de préemption des ressources (non supportés par les $RTPN$). En plus, si les $RTPN$ sont applicables pour tout domaine spécialement les workflow (une sous classe est définie pour cela, voir plus haut), les $STPTPN$ sont proposés pour les applications multimédia.

Toutefois, les langages précédents sont dotés pour certains d'entre eux d'outils et de techniques permettant leurs éditions ainsi que leurs analyses, alors que le $RTPN$ (et donc les $RTWFN$) est toujours en cours de validation. Etant conscient de ces lacunes, et dans le but de rendre ce modèle pratique, nous devons doter ce dernier de techniques permettant son analyse et un environnement réalisant les différents stages de la vérification, ceci implique :

- La proposition d'un algorithme permettant de dériver le graphe d'accessibilité du $RTPN$.
- Un module d'édition doté d'une interface graphique ;
- *Un interpréteur* permettant de traduire un $RTPN$ vers son graphe d'accessibilité correspondant et d'y vérifier les propriétés de la spécification ;

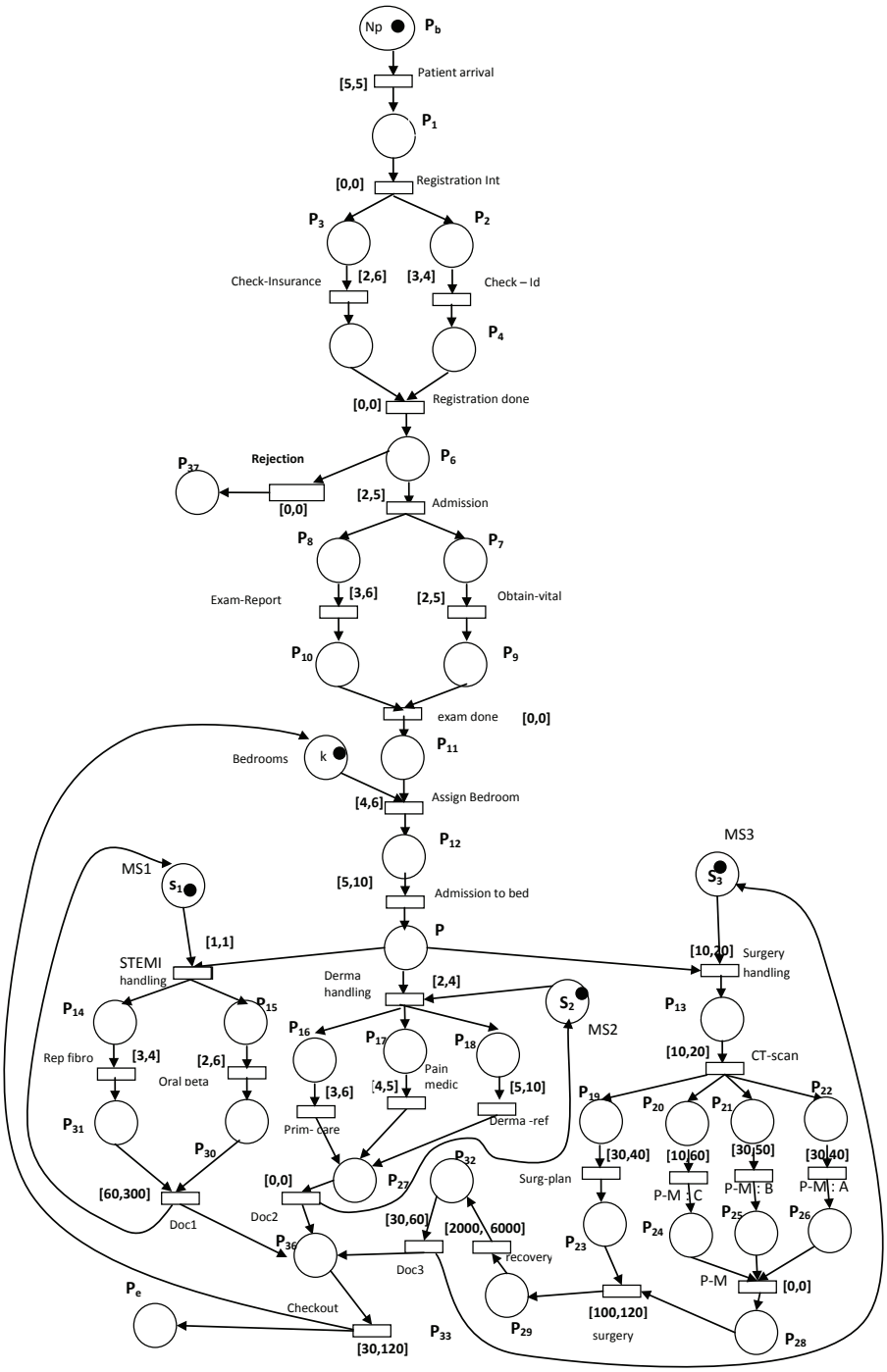

FIGURE 6.4 – Le *RTWFN* représentant : Patient workflow

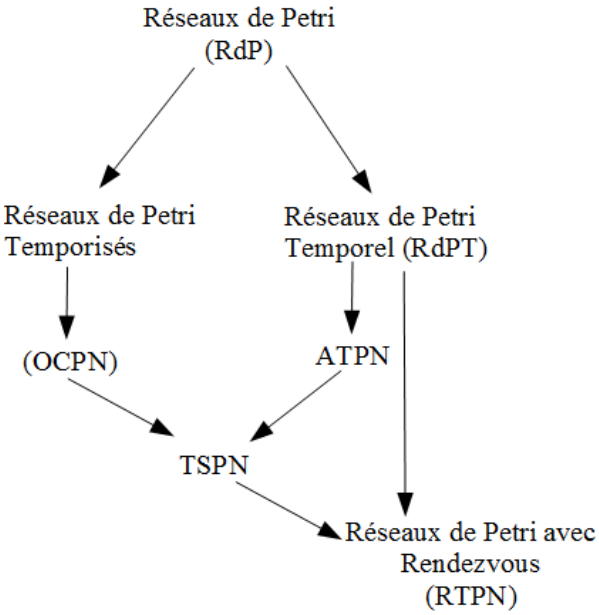

FIGURE 6.5 – *RTPN* vs *RdPT* et *TSPN*

- *Un simulateur* dont le fonctionnement est basé sur la sémantique formelle du modèle, réalisant des tests de simulation en générant les traces d'un $RTPN$.

Comme on peut le voir sur la figure 6.5, notre modèle étend le modèle de base des *RdpT* en associant un schéma de synchronisation aux transitions. Cependant, pour les $TSPN$ notre modèle propose un schéma plus important que celui basé sur le modèle d'ALLEN. En effet, on a la possibilité d'exprimer la notion de délai (retard et avance).

6.7 Conclusion

Nous avons présenté dans ce chapitre, à travers sa syntaxe et sa sémantique formelles un modèle permettant d'étendre les *RdPT* à différents mécanismes, et ce, dans le but de modéliser un large panel de systèmes temps réel . Nous avons explicité la puissance d'expression de cette nouvelle proposition à travers la spécification des contraintes et exigences des *Workflow* . Néanmoins, l'utilisation opérationnelle de ce modèle ne pourrait être productive que par la dotation de ce dernier de techniques et d'algorithmes permettant la dérivation d'un graphe d'accessibilité capturant les comportements de la spécification de base. D'ailleurs, ce besoin fera l'objet d'une proposition abordée dans le prochain chapitre, permettant l'énumération exhaustive de l'espace d'état d'un $RTPN$ ainsi que son analyse.

Chapitre 7

Construction du graphe des classes d'un $RTPN$

7.1 Introduction

Dans les chapitres précédents, nous avons vu que la syntaxe et la sémantique formelles d'un $RTPN$ (comme celle d'un $RdPT$), est exprimée en utilisant les systèmes de transitions étiquetées induisant un graphe infini en raison de la densité des instants de tir. Dans le présent chapitre, nous présentons une approche d'énumération de l'espace d'état du modèle $RTPN$, cette dernière consiste à regrouper des états accessibles après le franchissement de la même séquence non temporisée. Ce regroupement est appelé *classe d'état* et permet de regrouper le graphe des classes d'états.

7.2 Graphe des classes d'un $RTPN$

Problématique :
Pour un $RTPN$, s'ajoutent les contraintes liées à l'introduction des mécanismes de synchronisation et de délai. En effet, l'addition du concept de *rendez-vous* dans le modèle, en plus du concept standard de transition rend complexe la formalisation des contraintes de franchissement. L'expression de ce système devrait inférer aussi bien les contraintes des transitions que celles des rendez-vous. En clair, nous avons besoin de déterminer le plus large système capturant toutes les solutions des systèmes concernant les transitions et les rendez-vous sensibilisés associés aux états accessibles dans la même classe.

Proposition :
Pour un $RdPT$ [17], la méthode du graphe des classes [83, 2] permet de calculer un graphe symbolique préservant principalement les propriétés linéaires du modèle. La définition des classes d'états pour un $RTPN$ ne change pas par rapport à celle donnée pour les $RdPT$. Cette dernière consiste à regrouper, dans la même classe, les états accessibles après le franchissement de la même séquence non temporisée, mais de rendez-vous. Les états de la même classe ont tous le même marquage M ; et la classe initiale E_0 contient seulement l'état initiale e_0.

Dans un $RdPT$, le calcul des classes accessibles est obtenu en appliquant un algorithme comme celui du plus court chemin de *Floyd-Warshall*. Une complexité de $o(m^3)$. Où (m) est le nombre d'horloges dans le système D [2]. Ceci fait appel à des substitutions de variables par élimination et intersection, représentant la progression du temps.

Comme pour un $RdpT$, dans un $RTPN$, les contraintes temporelles sont associées aux transitions. Aussi une classe d'état est exprimée en fonction des contraintes temporelles associées aux transitions sensibilisées. Cependant, Nous ne parlons plus du tir de transition mais plutôt du tir de rendez-vous. En effet, le tir de rendez-vous nécessite des conditions plus complexes que celles du tir des transitions d'un $RdPT$. Nous allons voir, un peu plus loin que les contraintes temporelles associées aux rendez-vous sensibilisées sont obtenu grâce à des formules induites des contraintes temporelles des transitions.

Finalement, une classe d'états d'un $RTPN$ est définie par le couple (M, D) où M est le marquage commun à tous ses états et D est un système d'inéquations. Chaque variable présente dans D, notée $\underline{t_i}$, est associée à une transition sensibilisée t_i de même nom. Formellement, une classe d'état est définie comme suit :

Définition 7.2.1. *Soit $ST = (\Gamma, e_0, \rightarrow)$ le système de transition associé à un RTPN. Une classe d'état d'un RTPN, noté E, est l'ensemble des états de Γ accessible après le tir d'une même séquence non temporisée S, à partir de l'état initial e_0. La classe E est définie par (M, D), où M est le marquage accessible après le tir de S, et D est le système d'inéquations normalisé*[1]*, définit comme suit :*

$$D := \begin{cases} \bigwedge_{t_i,t_j \in E(M)} & (\underline{t_j} - \underline{t_i} \ \leq \ d_{ij}) \\ \bigwedge_{\forall t_i \in E(M)} & (d_{i\bullet} \leq \ \underline{t_i} \ \leq d_{\bullet i}) \end{cases}$$

tel que :

$d_{ij},\ d_{\bullet i} \in \mathbb{Q} \cup \{\infty\}, d_{i\bullet} \in \mathbb{Q}^+$

$d_{\bullet i} := \underset{\forall e \in E}{MAX}\{y(t_i)\} \qquad d_{i\bullet} := \underset{\forall e \in E}{MIN}\{x(t_i)\}$

$d_{ij} := \underset{\forall e \in E}{MAX}\{y(t_j) - x(t_i)\} \qquad d_{\bullet\bullet} := 0$

Nous notons par l'élément $\{\bullet\}$ l'instant où la classe E est atteinte. Par conséquent, la variable $\underline{t_i}$ exprime la date relative à l'instant $\bullet$, pour la transition t_i. Les coefficients $d_{\bullet i}, d_{i\bullet}$ et d_{ij} sont respectivement, le délai résiduel maximum, le délai résiduel minimum de la transition t_i et la distance maximale entre les temps résiduels de t_i à t_j. Nous présentons le système D comme une matrice carrée, où chaque ligne et sa colonne correspondante sont indexées par un élément de $E(M) \cup \{\bullet\}$. Concrètement, nous avons :

$\forall (t_i, t_j) \in E(M)^2 \ D[\bullet, t_i] := d_{\bullet i}\ ;\ D[t_i, \bullet] := -d_{i\bullet}\ ;\ D[t_i, t_j] := d_{ij}\ ;\ D[\bullet, \bullet] := 0.$

Le système D_0 relative à la classe initiale E_0 est définit comme suit :

$\forall t_i \in E(M_0), \quad D_0[\bullet, t_i] = LFT(t_i), \quad D_0[t_i, \bullet] = -EFT(t_i).$

$\forall (t_i, t_j) \in (E(M_0))^2 \ D_0[t_i, t_j] = LFT(t_j) - EFT(t_i),$

Nous présentons dans ce qui suit, notre approche de construction du graphe des classe d'états. Tout d'abord, il faudrait déterminer une condition suffisante pour le tir d'un rendez-vous. En effet, le système D définit plus haut n'est pas suffisant (en sa forme actuelle), pour le tir d'un rendez-vous. En fait, nous avons besoin de calculer un autre système équivalent, noté β, exprimant les contraintes temporelles des rendez-vous sensibilisés, ce dernier est défini comme suit :

Définition 7.2.2. *Soit $ST = (\Gamma, e_0, \rightarrow)$ le système de transition associé au RTPN. Soit $E = (M, D)$ le classe accessible après le tir de la séquence S dans ST. Pour $\underline{R}$ la variable associée à un rendez-vous sensibilisé R, nous définissons le système suivant :*

$$\beta := \begin{cases} \bigwedge_{\forall t_i \in E(M), \forall R_j \in ER(M)} & \underline{R_j} - \underline{t_i} \ \leq \ b_{ij} \\ \bigwedge_{\forall t_i \in E(M) - Tr(R), \forall R_j \in ER(M)} & \underline{t_i} - \underline{R_j} \ \leq \ b_{ji} \end{cases}$$

avec :

$b_{ij},\ b_{\bullet j} \in \mathbb{Q} \cup \{\infty\}.$

$b_{ij} := \underset{\forall e \in E}{MAX}\{Up(R_j) - x(t_i)\}\ .$

$b_{ji} := \underset{\forall e \in E}{MAX}\{y(t_i) - Low(R_j)\}\ .$

La variable $\underline{R}$ mesure le temps résiduel du rendez-vous R. Par conséquent, les coefficients b_{ij} et b_{ji} expriment la distance de temps maximale entre les temps résiduels de t_i et R_j lorsque nous considérons tous les états accessibles dans E. Nous voyons plus loin que les contraintes de type : $\underline{t_i} - \underline{R_j} \ \leq \ b_{ji}$, tel que $t_i \in Tr(R_j)$ sont d'aucune utilité pour le processus d'énumération des classes. C'est pour cela qu'il ne sont pas considérés dans le système β (plus haut).

Le système β_0 relatif à la classe initiale E_0 est défini comme suit :

$\forall R \in ER(M_0), \forall t \in E(M_0)$

$\beta_0[t, R] = Up_0(R) - EFT(t),$

$\forall R \in ER(M_0), \forall t \in E(M_0) - Tr(R)$

$\beta_0[R, t] = LFT(t) - Low_0(R),$

Comme pour D, le système β peut être représenté par une matrice (voir section Exemple plus loin).

Afin de formaliser le test de tir, nous avons besoin de formules nécessaires pour permettre le calcul, avec un coût minimal, des coefficients des systèmes β à partir de ceux de D.

Dans ce qui suit, nous notons par $-Op^+$ la fonction inverse de Op^+, c-à-dire : Si $-Op^+ = MAX$ alors $Op^+ = MIN$ et vice versa. La même remarque s'applique pour la fonction $-Op^-$.

1. D satisfait la forme normal : $\forall (x, y, z) \in E(M), (d_{xy} \leq d_{xz} + d_{zy}) \wedge (d_{xy} \leq d_{\bullet y} - d_{x\bullet})$

Proposition 7.2.1. *Le système β associé à la classe E peut être calculé comme suit :*

$$\forall R' \in ER(M),\ \forall x \in E(M)$$
$$\beta[x,R'] := MIN \begin{cases} \underset{\forall t' \in Set^+}{Op^+} \{D[x,t']\} & +\delta^+ \\ MAX \begin{cases} \underset{\forall t' \in Set^+}{Op^+} \{D[x,t']\} \\ \underset{\forall t' \in Set^-}{Op^-} \{D[x,t'] + \varepsilon(t')\} & +\delta^- \\ D[x,\bullet] + \delta^- \end{cases} \\ \underset{\forall t' \in Set^+}{MAX} \{D[x,t']\} \end{cases}$$

Nous avons aussi :

$$\forall R' \in ER(M),\ \forall x \in E(M) - Tr(R')$$
$$\beta[R',x] := MIN \begin{cases} \underset{\forall t' \in Set^-}{-Op^-} \{D[t',x]\} & +\alpha^- \\ MAX \begin{cases} \underset{\forall t' \in Set^+}{-Op^+} \{D[t',x] - \varepsilon(t')\} & +\alpha^+ \\ \underset{\forall t' \in Set^-}{-Op^-} \{D[t',x]\} \end{cases} \\ \underset{\forall t' \in Set^-}{MAX} \{D[t',x]\} \end{cases}$$

Avec $\varepsilon(t) = EFT(t) - LFT(t)$.

Preuve : La preuve est définie en remplaçant chaque coefficient par son expression formelle de la définition 7.2.2. Ensuite, en utilisant les définitions 6.3.2 et 7.2.1, nous prouvons les formules précédentes. Le détail des preuves est donné en annexe.

Maintenant que la méthode de calcul des coefficients du système β est bien définie, nous pouvons à présent l'utiliser dans le processus d'énumération des classes. Formellement, le graphe des classes d'un $RTPN$ est calculé comme suit :

Proposition 7.2.2. *Soit RT un $RTPN$. Le graphe des classes de RT, noté GR, est le tuple $(CE, E_0, \mapsto)$, avec :*

- *CE est l'ensemble des classes accessibles.*
- *$E_0 = (M_0, V_0) \in CE$ est la classe initiale.*
- *$\mapsto$: est la relation de transition entre classes définie sur $CE \times T \times CE$, nous avons, $((M,D), \underline{R_f}, (M^\uparrow, D^\uparrow)) \in \mapsto$, si et seulement si : $(R_f \in ER(M)) \wedge (DR[R_f] \geq 0)$; tel que :*

$$DR[R_f] := MIN \begin{cases} \underset{\forall t \in Set^-}{-Op^-} \{\lambda[t]\} & +\alpha^- \\ MAX \begin{cases} \underset{\forall t \in Set^+}{-Op^+} \{\lambda[t] - \varepsilon(t)\} & +\alpha^+ \\ \underset{\forall t \in Set^-}{-Op^-} \{\lambda[t]\} \end{cases} \\ \underset{\forall t \in Set^-}{MAX} \{\lambda[t]\} \end{cases}$$

$$\lambda[t] = \underset{\forall R \in EM}{MIN} \beta[t,R]$$

Et, nous avons :

1. $(\forall p \in P, \forall t \in Tr(R_f))$, $M^\uparrow(p) := M(p) - Pre(p,t) + Post(p,t)$.
2. *Les coefficients du système $D^\uparrow$ sont calculés en appliquant l'algorithme suivant :*

   $\forall t \in E(M^\uparrow)$

   Si $t \notin New(M^\uparrow)$ (t est persistante)

   $D^\uparrow[\bullet,t] := \beta[R_f,t]$; $D^\uparrow[t,\bullet] := Min(0, \lambda[t])$.

   Si $t \in New(M^\uparrow)$ (t est nouvellement sensibilisée.

   $D^\uparrow[\bullet,t] := LFT(t)$; $D^\uparrow[t,\bullet] := -EFT(t)$.

   $\forall (t_1,t_2) \in (E(M^\uparrow))^2$

   Si t_1 ou t_2 sont nouvellement sensibilisés.

   $D^\uparrow[t_1,t_2] := D^\uparrow[\bullet,t_2] + D^\uparrow[t_1,\bullet]$.

Si t_1 et t_2 sont persistantes.
$D^{\uparrow}[t_1, t_2] := MIN(D[t_1, t_2], \quad D^{\uparrow}[\bullet, t_2] + D^{\uparrow}[t_1, \bullet])$.

Selon la proposition 7.2.2, un rendez-vous sensibilisé R_f est franchissable si la distance de tir entre R_f et tout autre rendez-vous sensibilisé est positive ; ceci est déterminé en calculant $DR[R_f]$. Autrement dit, il existe au moins un état dans E tel que le rendez-vous R_f soit franchissable avant tout autre R sensibilisé. $DR[R_f]$ mesure le temps maximal relatif que l'on peut laisser écouler avant le tir de R_f. Par ailleurs, $\lambda[t]$ dénote le temps maximal que l'on peut laisser écouler avant le tir de la transition t. Si $DR[R_f]$ est négatif, cela veut dire que le temps ne peut pas progresser pour permettre le tir de R_f.

Une fois R_f tiré, la nouvelle classe est obtenue en calculant le nouveau marquage comme dans le cas des *Rdp* classiques. Le nouveau système $D^{\uparrow}$ est déterminé en calculant en premier lieu les coefficients $D^{\uparrow}[\bullet, t]$ et $D^{\uparrow}[t, \bullet]$ avant de calculer les distances $D^{\uparrow}[t, t']$. La complexité de calcul de la classe $E^{\uparrow}$ en utilisant la proposition 7.2.2 est quadratique et égale approximativement à $\mathrm{O}(2n^2 + 2mn)$, où n est le nombre des transitions sensibilisées et m est le nombre de rendez-vous sensibilisés.

Si nous exprimons l'espace d'état sous forme de résolution de système d'inéquations comme dans la méthode classique de construction du graphe des classes d'un *RdPT* [2], le tir d'un rendez-vous sensibilisé R_f revient à vérifier si le système (F) : $D \wedge \beta \wedge (\wedge_{\forall R \in ER(M)}(0 \preceq \underline{R} - \underline{R_f}))$ soit consistant. Cela implique de vérifier si R_f est franchissable avant tous les autres rendez-vous sensibilisés en respectant les contraintes de temps de tous les rendez-vous et transitions sensibilisés, encodé par le système $D \wedge \beta$. Ce qui est équivalent à tester $\underset{\forall R \in ER(M)}{MIN}(\underline{R}) - \underline{R_f} \preceq DR[R_f]$.

Concrètement, la formule définie par 7.2.2 pour calculer $DR[R_f]$ est obtenue en remplaçant la valeur de $Low(R_f)$ et en faisant une intersection avec les contraintes suivantes : $\forall R \in ER(M), \quad \underline{R} - \underline{t} \preceq \beta[t, R]$. Ceci détermine :
$\underset{\forall R \in ER(M)}{MIN}(\underline{R}) - \underline{t} \preceq \underset{\forall t \in E(M)}{MIN}\beta[t, R]$ par conséquent nous avons :

$\underset{\forall R \in ER(M)}{MIN}(\underline{R}) - \underline{t} \preceq \lambda[t]$.
Ensuite en considérant toutes les contraintes :
$\forall t \in R_f, \underset{\forall R \in ER(M)}{MIN}(\underline{R}) - \underline{t} \preceq \lambda[t]$.

Le rendrez-vous R_f est tirable, seulement lorsque toutes ses transitions le soient aussi. Nous avons également la contrainte $\forall t \in R_f, (\underline{t}) \preceq \underline{R_f}$ et $\underset{\forall R \in ER(M)}{MIN}(\underline{R}) - \underline{R_f} \preceq DR[R_f]$. Nous obtenons la formule de calcul de $DR[R_f]$ dans 7.2.2, La preuve détaillée est présenté en annexes.

Pour résumer, le calcul du système $D^{\uparrow}$ n'est qu'une optimisation de l'adaptation de algorithme de calcul d'une classe d'un *RdPT* qui revient à suivre les étapes :

1. Dans le système F, remplacer chaque variable $\underline{R}$ associée à un rendez-vous persistent dans $M^{\uparrow}$ par : $\underline{R} := \underline{R_f} + \underline{R}$, et chaque variable $\underline{t}$ associée à une transition persistante dans $M^{\uparrow}$ par $\underline{t} := \underline{R_f} + \underline{t}$, exprimant ainsi la progression dans le temps ;
2. Éliminer par substitution et intersection la variable $\underline{R_f}$ et les variables $\underline{t_f}$, telles que $t_f \in Tr(R_f)$ ainsi que toutes les variables associées aux rendez-vous et aux transitions désensibilisées par le tir de R_f ;
3. Ajouter au système obtenu les contraintes des transitions nouvellement sensibilisées pour $M^{\uparrow}$: $\forall t \in New(M^{\uparrow}), EFT(t) \leq \underline{t} \leq LFT(t)$
4. Garder dans le nouveau système les contraintes liées au variables $\underline{t}$ sensibilisées dans $M^{\uparrow}$.

7.2.1 Complexité

Selon les étapes de 'algorithme précédent, le calcul de la classe $E^{\uparrow}$ est d'une complexité cubique au nombre des variables présentes dans F. Même si cette complexité est polynomiale, elle reste importante pour pouvoir définir un algorithme efficace pour obtenir le graphe des classes d'un *RTPN*. L'algorithme proposé dans la Proposition 7.2.2 permet de réduire cette complexité au carré du nombre des variables. En effet, nous pouvons ignorer (éliminer) toutes les manipulations redondantes issues de l'algorithme précédent. Exemple, pour déterminer le coefficient $D^{\uparrow}[\bullet, t]$ tel que t est une transition sensibilisée et persistante dans $M^{\uparrow}$, nous avons besoin seulement besoin des contraintes du système F suivantes :
$\underline{t} - \underline{R_f} \preceq \beta[R_f, t]$ en remplaçant $\underline{t}$ par $\underline{t} + \underline{R_f}$, nous obtenons : $\underline{t} \preceq \beta[R_f, t] = D^{\uparrow}[\bullet, t]$

De la même manière, nous pouvons déterminer les coefficients $D^{\uparrow}[t, \bullet]$ en considérant seulement les contraintes suivantes :

$\forall R \in ER(M) \underline{R} - \underline{t} \preceq \beta[R, t]$ et $0 \preceq \underline{R} - R_f$.

En remplaçant $\underline{t}$ par $\underline{t} + R_f$ et $\underline{R}$ par $\underline{R} + \overline{R_f}$, nous obtenons :

$\forall R \in ER(M) \underline{R} - \underline{t} \preceq \beta[\overline{R}, t]$ et $0 \preceq \underline{R}$. Ensuite par intersection, nous avons :

- $MIN(0, \underset{\forall R \in ER(M)}{MIN} \beta[R, t] \preceq \underline{t}$, concrètement :

$-MIN(0, \lambda(t)) \preceq \underline{t} \Leftrightarrow -D^{\uparrow}[t, \bullet] \preceq \underline{t}$

Finalement, pour calculer le coefficient $D^{\uparrow}[t, t']$ lorsque t et t' sont deux transitions persistantes dans $M^{\uparrow}$, nous avons besoin de considérer seulement la contrainte : (*) : $\underline{t'} - \underline{t} \preceq D[t, t']$.

Après avoir remplacé $\underline{t'}$ et $\underline{t}$ par, respectivement, $\underline{t'} - \underline{R_f}$ et $\underline{t} - \underline{R_f}$ et faire l'intersection de cette dernière contrainte (*) avec les deux contraintes de la formule, nous obtenons :

$\underline{t'} - \underline{t} \preceq MIN(D[t, t'], D^{\uparrow}[\bullet, t'] + D^{\uparrow}[t, \bullet]) = D^{\uparrow}[t, t']$.

Résultats, l'intersection avec les autres contraintes de F devient inutile. La formule exprimée par la Proposition 7.2.1 offre une manipulation optimale pour le calcul des coefficients de $D^{\uparrow}$ à partir du système augmenté de la condition de tir F. Par conséquent, la complexité de l'algorithme donné par la Proposition 7.2.2 est de loin la meilleur (minimale) comparée à l'algorithme primaire ;

Le graphe des classes d'un $RTPN$ est construit de manière incrémentale par énumération de toutes les classes accessibles à partir de la classe initiale. En regroupant dans le même nœud du graphe celles qui sont équivalentes (voir les définitions suivantes).

7.2.2 Équivalence entre classes d'état

Soit $\rceil D \lceil$ (resp. $\rceil D' \lceil$) l'ensemble des solutions qui satisfait les systèmes D (rep.D').

Théorème 7.2.3. ***(Équivalence par inclusion)****. Soit $GR = (CE, E_0, \mapsto)$ le graphe de classe associé au $RTPN$. Deux classes $E = (M, D)$ et $E' = (M', D')$ sont équivalentes par inclusion si :*

- $M' = M$;
- $\rceil D' \lceil \subseteq \rceil D \lceil$.

Théorème 7.2.4. ***(Équivalence par égalité)****. Soit $GR = (CE, E_0, \mapsto)$ le graphe de classe associé au $RTPN$. Deux classes $E = (M, D)$ et $E' = (M', D')$ sont équivalentes par égalité si :*

- $M' = M$;
- $D' = D$.

7.2.3 Finitude du graphe des classes d'un $RTPN$

Théorème 7.2.5. *Le graphe des classes GrR d'un RTPN R, construit selon les Propositions 7.2.1, 7.2.2 et les Théorèmes 7.2.3, 7.2.4 est fini, si R est borné.*

Preuve : Le nombre de classes est borné si le nombre de marquages accessibles est borné, et que le nombre de systèmes D différents soit fini. Ce dernier est prouvé fini si les D est de forme DBM et que ses coefficients sont des rationnels [2] [83]. Cette dernière propriété étant vérifiée pour le graphe GR, donc sa finutude est décidé par sa bornitude.

7.3 Algorithme d'énumération du graphe des classes d'un $RTPN$

Pour construire le graphe de classes d'un $RTPN$, nous appliquons le test de franchissement donné en 7.2.2 sur la classe initiale E_0, ensuite sur toutes les classes accessibles. Les conditions suffisantes exprimées dans le Théorème.7.2.4 permettent de regrouper dans un même noeud toutes les classes équivalentes par égalité. Pour chaque classe à explorer, l'algorithme doit procéder d'abord au calcul des matrices, $\beta[R, t]$ et $\beta[t, R]$ avant celles de $\lambda[t]$ et $DR[R]$. Comme les deux premières matrices sont utilisées dans le calcul des classes directement accessibles à partir de E, il est nécessaire de les sauvegarder en mémoire jusqu'à l'exploration totale des successeurs voisins de E, autrement il faudrait les recalculer à chaque exploration. Par conséquent, pour une gestion optimale de l'espace mémoire, il est nécessaire d'adopter une stratégie d'exploration en largeur puis en profondeur ; ceci permet de ne garder en mémoire ces matrices que le

temps de l'exploration des successeurs. L'algorithme d'énumération adoptant cette stratégie décrit ci-après utilise une procédure récursive *Explorer- classe* dont le rôle est d'explorer les successeurs voisins de la classe courante en entrée de la procédure. Des structures de pile PE et de liste LE permettent de sauvegarder temporairement respectivement les classes calculées et les classes à explorer. Le graphe des classes est calculé de manière incrémentale jusqu'à qu'il n y ait plus de classes à explorer ou que la profondeur du graphe ait surpassé une valeur prédéfinie. Cette dernière condition permet de mettre fin au processus d'énumération dans le cas d'un processus infini dû par exemple à un réseau non borné.

Remarque 7.3.1. *Le critère d'arrêt pour l'exploitation du graphe peut correspondre tout aussi bien à la profondeur maximale du graphe, au marquage maximal du graphe, au nombre maximal de classes, au temps de calcul permis.*

7.4 Exemple illustratif

Considérons le $RTPN$ de la Figure 8.3. Ce réseau permet de décrire des concepts complexes telle que la sémantique de synchronisation (rendez-vous) avec délais. En effet, un ensemble de rendez-vous est défini dont certains évoluent de manière synchrone ($RDV_s = \{R_1, R_2, R_3, R_4\}$) et d'autres de manière asynchrone ($RDV_a = \{R_5, R_6\}$). Le détail des rendez-vous est comme suit :
$R_1 = (\{t_4, t_5\}, \emptyset, ((\{(t_4, t_5\}, MIN), (\{t_4, t_5\}, MIN)), (+\infty, +\infty, 0, 0))$
$R_2 = (\{t_1\}, \{t_3\}, ((\{t_1\}, MAX), (\{t_1, t_3\}), MIN), (0, 0, 0, 0))$
$R_3 = (\{t_1, t_2\}, \emptyset, ((\{t_1, t_2\}, MIN), (\{t_1, t_2\}, MAX)), (+\infty, +\infty, +\infty, +\infty))$
$R_4 = (\{t_2, t_5\}, \emptyset, (\{t_2, t_5\}, MAX), (\{t_2, t_5\}, MIN)), (0, +\infty, 0, 0))$
$R_5 = (\{t_5\}, \emptyset, (\{t_5\}, MIN), (\{t_5\}, MAX)), (+\infty, 0, +\infty, 0))$
$R_6 = (\{t_6\}, \emptyset, (\{t_6\}, MAX), (\{t_6\}, MAX)), (0, +\infty, +\infty, 0))$

En appliquant la technique d'énumération développée précédemment, nous sommes en mesure de recenser toutes les classes accessibles dans le graphe de classes GR représenté dans la Figure 7.2. Au départ, La classe E_0 correspond à l'état initiale et est caractérisée par le marquage initiale M_0 et le système d'inégalités D_0 (voir Table 7.4).

Ensuite, à partir de la classe E_0, nous déterminons le / ou les rendez-vous parmi l'ensemble $ER(M_0)$ qui peuvent être tirés ? La valeur de DR des rendez-vous permet de répondre à cette question (si DR est positif ou nul le rendez-vous est tirable, dans le cas contraire, il ne l'est pas). Nous commençons par calculer les deux matrices représentant le système β_0, à partir de la matrice D_0 en appliquant les formules de la Proposition 7.2.1. Nous devons calculer également $\varepsilon(t_i)$ tel que : $\varepsilon(t_i) = -LFT(t_i) + EFT(t_i)$, comme suit :
$\varepsilon(t_1) = -LFT(t_1) + EFT(t_1) = -2 + 1 = -1$;
$\varepsilon(t_2) = -5 + 4 = -1$;
$\varepsilon(t_3) = -2 + 1 = -1$;
$\varepsilon(t_4) = -2 + 0 = -2$;
$\varepsilon(t_5) = -4 + 2 = -2$,
$\varepsilon(t_6) = -1 + 1 = 0$.

Nous avons :
Classe E_0 **:** $M_0 : p_1, p_2, p_3, p_4, p_5 \mapsto 1; p_6, p_7 \mapsto 0; E(M_0) = \{t_1, t_2, t_4, t_5\}; ER(M_0) = \{R_1, R_3, R_4, R_5\}$.

D_0	•	t_1	t_2	t_4	t_5
•	0	1	5	2	4
t_1	0	1	5	2	4
t_2	-4	-3	1	-2	0
t_4	0	1	5	2	4
t_5	-2	-1	3	0	2

TABLE 7.1 – La matrice représentant le système D_0

Dans la matrice $\beta_0[t, R]$, la première ligne à partir du haut correspond à l'ensemble des rendez-vous sensibilisés dans E_0 ($ER(M_0)$) et la première colonne à gauche définit l'ensemble des transitions sensibilisées ($E(M_0)$). Pour la matrice $\beta_0[R, t]$ les éléments de la ligne et de la colonne sont inversés par rapport à la matrice $\beta_0[t, R]$. Nous calculons le premier élément ($\beta_0[t_1, R_1]$) de la matrice comme suit :

Algorithm 1 Construction du graphe des classes d'un *RTPN*

Déclaration
1- PE : Pile de type classe ;{ contient les classes non encore explorées}
2- GE : Structure graphe ;{ contient le graphe à calculer}
3- LE : Liste de type classe ;{ contient les classes calculées}
4- $D, \beta_{[T,R]}, \beta_{[R,T]}, \lambda, DR$: Matrices ;
5- $ChoixArret[i]$:Tableau pour les paramètres d'arrêt ; {Paramètre d'arrêt ; exp : $ChoixArret[1] == true$ le marquage a été choisi comme paramètre d'arrêt}
6- $Arret[i]$: Tableau de valeurs logiques constante ; {détermine le critère d'arrêt d'exploration du graphe (exp :$Arret[0] == true$; profondeur maximale atteinte, voir Remarque 7.3.1) }
{**Procédure Explorer classe** (E : type classe}
début
7- Calculer ($TransSens$) ;{ calculer les transitions sensibilisées}
8- Calculer ($RdvSens$) ;{ calculer les rendez-vous sensibilisés}
9- Calculer (D) ;{ qui permet de calculer les deux matrice qui suivent }
10- Calculer les deux matrice($\beta[t, R]$ et $\beta[R, t]$) ;{ selon Proposition 7.2.1}
11- Calculer la matrice (λ) ;{ Permet de calculer la matrice DR selon Proposition 7.2.2}
12- Calculer matrice (DR) ;{ Déterminer les rendez-vous franchissables selon Proposition 7.2.2}
for $R \in RdvSens$ (Pour tout rendez-vous sensibilisé) **do**
  if $R \in RdvFranch$ (R est franchissable ?) **then**
    13- $E^{\uparrow}$:= Classe accessible(E, R) ;{ Selon proposition 7.2.2 }
    if $E^{\uparrow} \notin LE$ (pas d'équivalence selon Théorème 7.2.4) **then**
      13- *Empiler*($PE, E^{\uparrow}$) ;
      14- $LE \leftarrow E^{\uparrow} + LE$;
      15- Créer arc graphe$(GR, E^{\uparrow}, L)$;
    end if
  end if
end for
16- E := Dépiler PE ;
while $(Arret[i] \neq 1) \wedge (E^{\uparrow} \neq E)$ (critère d'arrêt non atteint et la classe est nouvelle càd pas d'équivalence) **do**
  17- Décrémenter paramètre d'arrêt ;
  18- Explorer classe(E) ;
  19- E := Dépiler PE ;
  20- Incrémenter paramètre d'arrêt ;
  21- $prof := prof + 1$;
end while
———————- Fin Déclaration ———————
22- **DEBUT**
23- *Calculer la classe initiale* $E_0 = (M_0, D_0)$;
24- *Créer structure graphe*(GR, E_0) ;
25- $LE \leftarrow E_0 + LE$;
26- *Empiler*(PE, E_0) ;
27- *Explorer classe*(E_0) ;
28- E := *Dépiler* (PE) ;
if GE est non vide **then**
  29- Le graphe GR n'est pas complètement exploré
end if
FIN

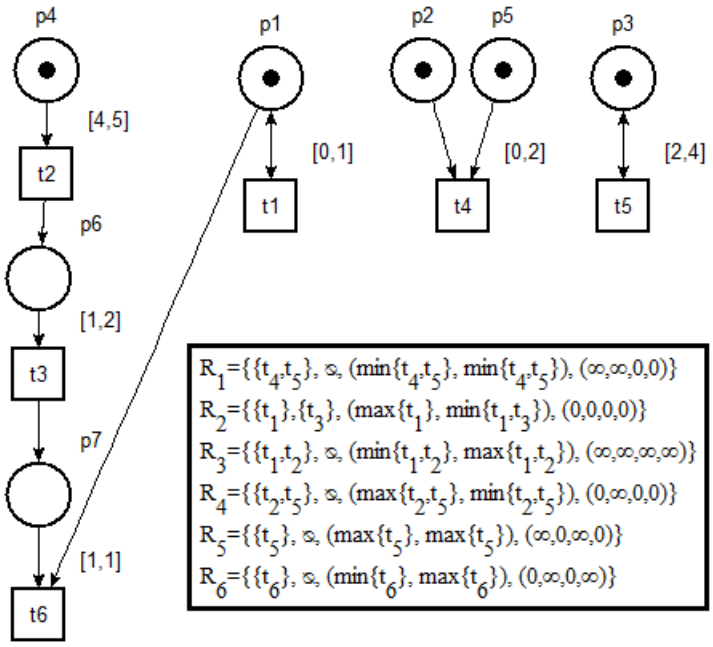

FIGURE 7.1 – Exemple d'un *RTPN*

$\beta_0[t,R]$	R_1	R_3	R_4	R_5
t_1	2	5	4	4
t_2	-2	1	0	0
t_4	2	5	4	4
t_5	0	3	2	2

$\beta_0[R,t]$	t_1	t_2	t_4	t_5
R_1	1	5	x	x
R_3	x	x	2	4
R_4	-3	x	-2	x
R_5	-1	3	0	x

TABLE 7.2 – Les matrices représentant le système β_0

$\lambda_0[t]$	...
t_1	2
t_2	-2
t_4	2
t_5	0

$DR_0[R]$	...
R_1	0
R_3	2
R_4	-2
R_5	0

TABLE 7.3 – $\lambda_0[t]$ et $DR_0[R]$

$$\beta_0[t_1,R_1] := MIN \begin{cases} \underset{\forall t' \in Set^+}{Op^+} \{D_0[t_1,t']\} + \delta^+ \\ MAX \begin{cases} \underset{\forall t' \in Set^+}{Op^+} \{D_0[t_1,t']\} \\ \underset{\forall t' \in Set^-}{Op^-} \{D_0[t_1,t'] + \varepsilon(t')\} + \delta^- \\ D_0[t_1,\bullet] + \delta^- \end{cases} \\ \underset{\forall t' \in Set^+}{MAX} \{D_0[t_1,t']\} \end{cases}$$

Avec :
$\underset{\forall t' \in Set^+}{Op^-} = MIN$, $\underset{\forall t' \in Set^+}{Op^+} = MIN$, $Set^- = \{t_4, t_5\}$, $Set^+ = \{t_4, t_5\}$ $(Tr(R_1) = Tm(R_1))$ et $t' \in \{t_4, t_5\}$.
Les paramètres temporelles $(\delta^- = \infty, \delta^+ = 0)$. Nous remplaçons et nous obtenons :

$$\beta[t_1, R_1] := MIN \begin{cases} \underset{\forall t' \in \{t_{4,t_5}\}}{Min} \{D_0[t_1, t']\} \quad + 0 \\ MAX \begin{cases} \underset{\forall t' \in \{t_{4,t_5}\}}{Min} \{D_0[t_1, t']\} \\ \underset{\forall t' \in \{t_{4,t_5}\}}{Min} \{D_0[t_1, t'] + \varepsilon(t')\} \quad + \infty \\ D_0[t_0, \bullet] + \infty \end{cases} \\ \underset{\forall t' \in \{t_{4,t_5}\}}{Max} \{D_0[t_1, t']\} \end{cases}$$

Ensuite,

$$\beta[t_1, R_1] := MIN \begin{cases} \underset{\forall t' \in \{t_{4,t_5}\}}{Min} \{(2,4)\} \quad + 0 \\ MAX \begin{cases} \underset{\forall t' \in \{t_{4,t_5}\}}{Min} \{2,4)\} \\ \underset{\forall t' \in \{t_{4,t_5}\}}{Min} \{2-2, 4-2)\} \quad + \infty \\ 0 + \infty \end{cases} \\ \underset{\forall t' \in \{t_{4,t_5}\}}{Max} \{2,4)\} \end{cases}$$

Après simplification et suppression de certaines lignes inutiles, nous obtiendrons une formule plus simple comme suit :
$\beta_0[t_1, R_1] := MIN\{Min(2,4)Max(2,4)\} = Min(2,4) = 2.$
$\beta_0[t_2, R_1] := MIN\{Min(-2,0)Max(-2,0)\} = Min(-2,0) = -2.$
$\beta_0[t_4, R_1] := MIN\{Min(2,4)Max(2,4)\} = Min(2,4) = 2.$
$\beta_0[t_5, R_1] := MIN\{Min(0,2)Max(0,2)\} = Min(0,2) = 0.$

Le reste est calculé de la même manière en remplaçant R_1 par R_3, R_4 et R_5.

Une fois la matrice $\beta_0[t, R]$ calculée, nous passons à la matrice $\beta_0[R, t]$, le symbole x dénote que le coefficient n'est pas défini et donc n'est pas utile dans le processus de calcul. En appliquant la même définition, nous déterminons les éléments de la matrice, comme suit :

$$\beta_0[R_1, t_1] := MIN \begin{cases} \underset{\forall t' \in Set^-}{-Op^-} \{D_0[t', t]\} \quad + \alpha^- \\ MAX \begin{cases} \underset{\forall t' \in Set^+}{-Op^+} \{D_0[t', t] - \varepsilon(t')\} \quad + \alpha^+ \\ \underset{\forall t' \in Set^-}{-Op^-} \{D_0[t', t]\} \end{cases} \\ \underset{\forall t' \in Set^-}{MAX} \{D_0[t', t]\} \end{cases}$$

Avec :
$\underset{\forall t' \in Set^+}{-Op^-} = MAX$, $\underset{\forall t' \in Set^+}{-Op^+} = MAX$. Les paramètres temporelles ($\alpha^- = \infty, \alpha^+ = 0$). Nous remplaçons et nous obtenons :

$$\beta_0[R_1, t_1] := MIN \begin{cases} \underset{\forall t' \in \{t_4, t_5\}}{Max} \{D_0[t', t_1]\} \quad + \infty \\ MAX \begin{cases} \underset{\forall t' \in \{t_4, t_5\}}{Max} \{D_0[t', t_1] - \varepsilon(t')\} \quad + 0 \\ \underset{\forall t' \in \{t_4, t_5\}}{Max} \{D_0[t', t_1]\} \end{cases} \\ \underset{\forall t' \in \{t_4, t_5\}}{Max} \{D_0[t', t_1]\} \end{cases}$$

Ensuite,

$$\beta_0[R_1, t_1] := MIN \begin{cases} \underset{\forall t' \in \{t_4, t_5\}}{Max} \{(-1,1)\} \quad + \infty \\ MAX \begin{cases} \underset{\forall t' \in \{t_4, t_5\}}{Max} \{(-1+2, 1+2)\} \quad + 0 \\ \underset{\forall t' \in \{t_4, t_5\}}{Max} \{(-1,1)\} \end{cases} \\ \underset{\forall t' \in \{t_4, t_5\}}{Max} \{(-1,1)\} \end{cases}$$

Après simplification, nous obtenons :
$\beta_0[R_1, t_1] := MIN\{Max(-1,1), MAX(max(-1,1), max(-1+2, 1+2) + 0\} = 1.$
$\beta_0[R_1, t_2] := MIN\{Max(5,3), MAX(max(5,3), max(5+2, 3+2) + 0\} = 5.$
$\beta_0[R_1, t_4] := x$.
$\beta_0[R_1, t_5] := x$.

Une fois le système β_0 défini, nous calculons les éléments $\lambda_0[t]$ pour les transitions sensibilisées et les éléments $DR_0[R]$ pour tout rendez-vous sensibilisé dans E_0.

$$\lambda_0[t_1] = \underset{\forall R \in E(M_0)}{MIN} \beta_0[t_1, R] =$$
$$\underset{\forall R \in \{R_1,R_3,R_4,R_5\}}{MIN} \beta_0[t_1, R] = MIN\{2, 5, 4, 4\} = 2$$
$$\lambda_0[t_2] = \underset{\forall R \in E(M_0)}{MIN} \beta_0[t_2, R] =$$
$$\underset{\forall R \in \{R_1,R_3,R_4,R_5\}}{MIN} \beta_0[t_1, R] = MIN\{-2, 1, 0, 0\} = -2$$
$$\lambda_0[t_4] = \underset{\forall R \in E(M_0)}{MIN} \beta_0[t_4, R] =$$
$$\underset{\forall R \in \{R_1,R_3,R_4,R_5\}}{MIN} \beta_0[t_4, R] = MIN\{2, 5, 4, 4\} = 2$$
$$\lambda_0[t_5] = \underset{\forall R \in E(M_0)}{MIN} \beta_0[t_5, R] =$$
$$\underset{\forall R \in \{R_1,R_3,R_4,R_5\}}{MIN} \beta_0[t_5, R] = MIN\{0, 3, 2, 2\} = 0$$

Nous déterminons les valeurs de $DR[R_f]$ en utilisant les formules de la Proposition 7.2.2, comme suit :

$$DR_0[R_1] := MIN \begin{cases} \underset{\forall t \in Set^-}{-Op^-}\{\lambda_0[t]\} & + \alpha^- \\ MAX \begin{cases} \underset{\forall t \in Set^+}{-Op^+}\{\lambda_0[t] - \varepsilon(t)\} & + \alpha^+ \\ \underset{\forall t \in Set^-}{-Op^-}\{\lambda_0[t]\} \end{cases} \\ \underset{\forall t \in Set^-}{MAX}\{\lambda_0[t]\} \end{cases}$$

$$DR_0[R_1] := MIN \begin{cases} \underset{\forall t \in \{t_4,t_5\}}{MIN}\{\lambda_0[t]\} & + \infty \\ MAX \begin{cases} \underset{\forall t \in \{t_4,t_5\}}{MIN}\{\lambda_0[t] - \varepsilon(t)\} & + 0 \\ \underset{\forall t \in \{t_4,t_5\}}{MIN}\{\lambda_0[t]\} \end{cases} \\ \underset{\forall t \in \{t_4,t_5\}}{MAX}\{\lambda_0[t]\} \end{cases}$$

$DR_0[R_1] = MIN\{MAX(Max(0, 2), Max(0 + 2, 2 + 2) + 0), Max(0, 2)\} = MIN(4, 2) = 2.$
De la même manière, nous calculons $DR_0[R_3], DR_0[R_4]$ et $DR_0[R_5]$:

Par conséquent :
$DR[R_1] \geq 0$ et donc : R_1 tirable
$DR_0[R_3] = MIN\{Max(2, -2)\} = 2$; nous avons : $DR[R_3] \geq 0$. (mais t_2 non tirable donc R_3 non tirable)
$DR_0[R_4] = MIN\{Min(-2, 0)\} = -2$; nous avons : $DR[R_4] \not\geq 0$. (R_4 non tirable)
$DR_0[R_5] = 0$; nous avons : $DR[R_5] \geq 0$. tirable)

Nous remarquons qu'à partir de la classe E_0, nous pouvons tirer les rendez-vous suivants :
R_1 pour atteindre la classe $E_1 = (M_1, D_1)$
R_5 pour atteindre la classe $E_0 = (M_0, D_0)$ de nouveau.

Reste à calculer les marquages M et les systèmes D pour les classes accessibles E_1, E_2 suivant la Proposition 7.2.2. Nous prenons l'exemple de la classe E_1

Classe E_1 : $M_1 : p_1, p_3, p_4 \mapsto 1; p_2, p_5, p_6, p_7 \mapsto 0$, l'ensemble des transitions sensibilisées est $E(M_1) = \{t_1, t_2, t_5\}$ et l'ensemble des rendez-vous sensibilisés est $ER(M_1) = \{R_3, R_4, R_5\}$. La matrice D_1 est calculée comme suit :
$D_1[\bullet, t] = \beta[R_f, t]$ pour $t \in \{t_1, t_2\}$ et persistante, R_f est le rendez-vous franchi pour atteindre E_1 (R_1)
$D_1[\bullet, t_1] = \beta[R_1, t_1] = 1$
$D_1[\bullet, t_2] = \beta[R_f, t_2] = 5$
$D_1[\bullet, t_5] = LFT(t_5) = 4$ (t_5) ; nouvellement sensibilisée.

Nous avons ensuite :
$D_1[t, \bullet] = Min(0, \lambda[t])$ pour $t \in \{t_1, t_2\}$ et persistante.
$D_1[t_1, \bullet] = Min(0, 2) = 0$

$D_1[t_2, \bullet] = Min(0, -2) = -2$
$D_1[t_5, \bullet] = EFT(t_5) = -2$ (t_5) ; nouvellement sensibilisée.

Le détail des autres classes accessibles est le suivant :

D_1	$\bullet$	t_1	t_2	t_5
$\bullet$	0	1	5	4
t_1	0	1	5	4
t_2	-2	-1	1	2
t_5	-2	-1	3	2

$\beta_1[t, R]$	R_3	R_4	R_5
t_1	5	4	4
t_2	3	1	2
t_5	3	2	2

$\beta_1[R, t]$	t_1	t_2	t_5
R_3	x	x	4
R_4	-1	x	x
R_5	-1	3	x

TABLE 7.4 – Les matrices représentant les système D_1 et β_1

$\lambda_1[t]$	...
t_1	4
t_2	1
t_5	2

$DR_1[R]$	...
R_3	1
R_4	2
R_5	-1

TABLE 7.5 – $\lambda_1[t]$ et $DR_1[R]$

Classe E_2 **:** $M_2 : p_1, p_3, p_6 \mapsto 1; E(M_2) = \{t_1, t_3, t_5\}; ER(M_2) = \{R_2, R_5\}$.

D_2	$\bullet$	t_1	t_3	t_5
$\bullet$	0	1	2	4
t_1	0	1	2	4
t_3	-1	0	1	3
t_5	0	1	2	2

$\beta_2[t, R]$	R_2	R_5
t_1	1	4
t_3	0	3
t_5	1	2

$\beta_2[R, t]$	t_1	t_3	t_5
R_2	x	x	4
R_5	1	2	x

$\lambda_2[t]$	...
t_1	1
t_3	0
t_5	1

$DR_2[R]$	...
R_2	1
R_5	1

TABLE 7.6 – Les matrices représentant les système $D_2, \beta_2, \lambda_2[t]$ et $DR_2[R]$

Classe E_3 **:** $M_3 : p_1, p_3, p_7 \mapsto 1; E(M_3) = \{t_1, t_5, t_6\}; ER(M_3) = \{R_5, R_6\}$.

D_3	$\bullet$	t_1	t_5	t_6
$\bullet$	0	1	4	1
t_1	0	1	4	1
t_5	0	1	2	1
t_6	-1	0	3	0

$\beta_3[t, R]$	R_5	R_6
t_1	4	1
t_5	2	1
t_6	3	0

$\beta_3[R, t]$	t_1	t_5	t_6
R_5	1	x	1
R_6	0	3	x

$\lambda_3[t]$	...
t_1	1
t_5	1
t_6	0

$DR_3[R]$	...
R_5	1
R_6	0

Classe E_4 **:** $M_4 : p_1, p_3, p_7 \mapsto 1; E(M_4) = \{t_1, t_5, t_6\}; ER(M_4) = \{R_5, R_6\}$.

D_4	•	t_1	t_5	t_6
•	0	1	4	1
t_1	0	1	4	1
t_5	-2	-1	2	-1
t_6	0	0	4	0

$\beta_4[t,R]$	R_5	R_6
t_1	4	1
t_5	2	-1
t_6	4	0

$\beta_4[R,t]$	t_1	t_5	t_6
R_5	-1	x	-1
R_6	0	4	x

$\lambda_4[t]$	...
t_1	1
t_5	-1
t_6	0

$DR_4[R]$	...
R_5	-1
R_6	0

D_5	•	t_5
•	0	4
t_5	-1	2

$\beta_5[t,R]$	R_5
t_5	2

$\beta_5[R,t]$	t_5
R_5	0

$\lambda_5[t]$	...
t_5	2

$DR_5[R]$	...
R_5	2

Classe E_5 **:** $M_5 : p_3 \mapsto 1; E(M_5) = \{t_5\}; ER(M_5) = \{R_5\}$.
Classe E_6 **:** $M_6 : p_3 \mapsto 1; E(M_6) = \{t_5\}; ER(M_6) = \{R_5\}$.

D_6	•	t_5
•	0	4
t_5	-2	2

$\beta_6[t,R]$	R_5
t_5	2

$\beta_6[R,t]$	t_5
R_5	0

$\lambda_6[t]$	...
t_5	2

$DR_6[R]$	...
R_5	2

Classe E_7 **:** $M_7 : p_3 \mapsto 1; E(M_7) = \{t_5\}; ER(M_7) = \{R_5\}$.

D_7	•	t_5
•	0	3
t_5	0	2

$\beta_7[t,R]$	R_5
t_5	2

$\beta_7[R,t]$	t_5
R_5	0

$\lambda_7[t]$	...
t_5	2

$DR_7[R]$	...
R_5	2

Classe E_8 **:** $M_8 : p_1, p_3, p_6 \mapsto 1; E(M_8) = \{t_1, t_3, t_5\}; ER(M_8) = \{R_2, R_5\}$.

D_8	•	t_1	t_3	t_5
•	0	1	2	4
t_1	0	0	1	4
t_3	0	1	2	1
t_5	-2	1	2	2

$\beta_8[t,R]$	R_2	R_5
t_1	1	4
t_3	0	4
t_5	-1	2

$\beta_8[R,t]$	t_1	t_3	t_5
R_2	x	x	4
R_5	1	2	x

$\lambda_8[t]$	...
t_1	1
t_5	1
t_6	0

$DR_8[R]$	...
R_2	0
R_5	-1

Classe E_9 **:** $M_9 : p_1, p_3, p_7 \mapsto 1; E(M_9) = \{t_1, t_5, t_6\}; ER(M_9) = \{R_5, R_6\}$.
Classe E_{10} **:** $M_{10} : p_1, p_3, p_7 \mapsto 1; E(M_{10}) = \{t_1, t_5, t_6\}; ER(M_{10}) = \{R_5, R_6\}$.

D_9	•	t_1	t_5	t_6
•	0	1	4	1
t_1	0	1	4	1
t_5	-1	0	2	0
t_6	-1	0	4	0

$\beta_9[t,R]$	R_5	R_6
t_1	4	1
t_5	2	0
t_6	3	0

$\beta_9[R,t]$	t_1	t_5	t_6
R_5	2	x	1
R_6	1	0	x

$\lambda_9[t]$	...
t_1	1
t_5	-1
t_6	0

$DR_9[R]$	...
R_5	-1
R_6	0

D_{10}	•	t_1	t_5	t_6
•	0	0	4	0
t_1	0	0	4	0
t_5	-2	-2	2	-2
t_6	0	0	4	0

$\beta_{10}[t,R]$	R_5	R_6
t_1	4	0
t_5	2	-2
t_6	4	0

$\beta_{10}[R,t]$	t_1	t_5	t_6
R_5	-2	x	2
R_6	0	-2	x

$\lambda_{10}[t]$	...
t_1	0
t_5	-2
t_6	0

$DR_{10}[R]$	...
R_5	-2
R_6	0

D_{11}	•	t_5
•	0	3
t_5	-1	2

$\beta_{11}[t,R]$	R_5
t_5	2

$\beta_{11}[R,t]$	t_5
R_5	0

$\lambda_{11}[t]$	...
t_5	2

$DR_{11}[R]$	...
R_5	2

Classe E_{11} **:** $M_{11} : p_3 \mapsto 1; E(M_{11}) = \{t_5\}; ER(M_{11}) = \{R_5\}$.
Classe E_{12} **:** $M_{12} : p_1, p_3, p_6 \mapsto 1; E(M_{12}) = \{t_1, t_3, t_5\}; ER(M_{12}) = \{R_2, R_5\}$.

D_{12}	•	t_1	t_3	t_5
•	0	1	2	4
t_1	0	1	2	4
t_3	0	0	1	4
t_5	-2	-1	0	2

$\beta_{12}[t,R]$	R_2	R_5
t_1	1	2
t_3	0	1
t_5	-1	0

$\beta_{12}[R,t]$	t_1	t_3	t_5
R_2	x	x	4
R_5	-1	0	x

$\lambda_{12}[t]$	...
t_1	1
t_3	0
t_5	-1

$DR_{12}[R]$	...
R_2	1
R_5	-1

Finitude du graphe des classe : Le graphe de classe est borné comportant 13 classes et 17 arcs.

7.5 Conclusion

Durant ce chapitre, une technique de construction des classes d'états d'un *RTPN* est développée. Une sémantique plus complexe est prise en considération induite par le mécanisme de synchronisation. La représentation des classes n'est pas différente de celle d'un *RdpT*, exprimée par un marquage et un système de contraintes temporelles. Cependant, les conditions de tir ne correspondent plus à une transition mais plutôt au tir d'un rendez-vous sensibilisé. Plus encore, une condition suffisante est proposée afin de calculer l'ensemble des classes accessibles à partir de la classe initiale et un nouveau système noté β est associé au système D. Pour assurer la finitude de la construction du graphe pour un *RTPN* borné, deux

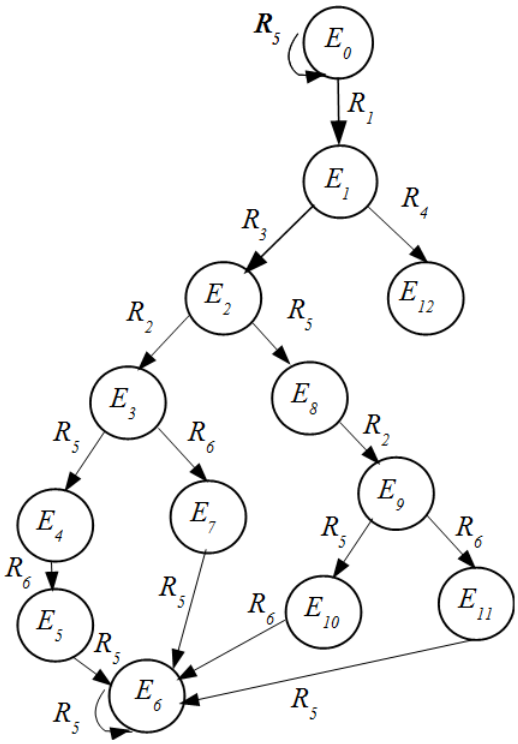

FIGURE 7.2 – Le graphe de classes correspondant au *RTPN* 8.3

test d'égalité sont proposés. Enfin, nous discutons dans le chapitre suivant les propriétés que préserve cette construction et comment les déterminer.

Chapitre 8

Analyse de l'espace d'état d'un $RTPN$

8.1 Introduction

Nous discutons dans ce chapitre comment exploiter le graphe des classes construit dans le Chapitre précédent pour l'analyse des propriétés linéaires et temps réel du système modélisé. Nous développons ensuite un algorithme qui calcule, en parcourant un chemin (ou un cycle) du graphe des classes, les temps minimaux et maximaux d'exécution de la séquence du chemin.

8.2 Les propriétés qualitatives et quantitatives du graphe des classes d'un $RTPN$

Dans cette section, nous voyons comment vérifier les propriétés dites générales d'un $RTPN$. Par "général", nous faisons référence aux deux principales familles de propriétés (e.g. propriétés qualitatives et propriétés quantitatives).

Le but de construire le graphe des classes d'un $RTPN$ est de déduire les propriétés les plus importantes du modèle (e.g. accessibilité, blocage, vivacité,...etc). En considérant le graphe GR, toutes les propriétés linéaires du modèle (celles qui peuvent être vérifiées en utilisant des logiques linéaires telles que LTL) sont préservées. Par conséquent, pour vérifier si un $RTPN$ satisfait une propriété donnée ; il suffit de la vérifier sur son graphe GR. Ainsi, les propriétés rattachées à l'atteignabilité peuvent être immédiatement déduites en explorant le graphe de la même manière que pour les réseaux de Petri standards. Les propriétés de sûreté i.e, '*quelque chose de mauvais n'arrive jamais*', sont alors vérifiables.

D'autres propriétés peuvent être vérifiées (discrètes ou quantitatives) sur le graphe. Dans ce cas, nous pouvons étendre le modèle $RTPN$ en amont avec des places ou des transitions (rendez-vous asynchrone) d'observation permettant de modéliser la propriété à analyser. Ensuite la vérification de cette propriété sur le graphe GR se fait en vérifiant par exemple l'atteignabilité ou non du marquage des places de l'observateur [115, 116].

Par ailleurs, concernant les propriétés temps réel (e.g. propriété du temps borné) requièrent la détermination des temps minimum et maximum d'une séquence de franchissement, elles peuvent être calculées à partir du graphe des classes GR. Pour ce faire, *Bucci et autres* [117] ont démontré que pour un $RdPT$ à chronomètre, le calcul du temps maximum ou minimum d'une séquence de franchissement nécessite la résolution d'un système linéaire dont la complexité est exponentielle à la longueur de la séquence et au nombre de transitions sensibilisées. *Boucheneb et Mullins*[88] proposent un algorithme plus simple qui calcule, en parcourant un chemin (ou un cycle), les temps minimaux et maximaux d'exécution de la séquence du chemin. La complexité de l'algorithme est de $O(m \text{ x } n)$, tel que m est la longueur du chemin et n le nombre de transitions sensibilisées.

Ce coût de calcul étant encore plus important pour un $RTPN$ [1]. Nous montrons ci-après comment calculer, en parcourant le long d'un chemin du graphe, les temps minimum et maximum de n'importe quelle sous séquence du chemin exploré.

1. La complexité dépend en plus du nombre de rendez-vous sensibilisés.

8.3 Temps minimal et maximal d'une séquence de franchissement d'un *RTPN*

Afin de formaliser le processus de calcul, nous considérons une séquence de franchissement $S_i^n = (R_{i+1}, .., R_n)$. S_i^n décrivant un chemin dans le graphe GR partant du noeud représentant la classe E_i au noeud représentant la classe E_n. Par conséquent, les temps minimum et maximum de la séquence $(R_{i+1}, ..., R_n)$ sont donnés par la somme $\underline{R_{i+1}} + .. + \underline{R_n}$, et dont les valeurs optimales sont prises dans l'espace de franchissement capturant toutes les solutions admissibles de $(Q^+)^{n-i}$, et défini comme suit :

Définition 8.3.1. *Soit E_n une classe accessible dans GR à partir de la classe E_i après le franchissement de la séquence $S_i^n = (R_{i+1}, .., R_n)$. Nous dénotons par space(S_i^n) l'espace de franchissement défini sur $(Q^+)^{n-i}$ des vecteurs $(\theta_{i+1}, .., \theta_n)$ telle que la séquence S_i^n peut être franchie aux dates relatives $(\underline{R_{i+1}}, .., \underline{R_n}) := (\theta_{i+1}, .., \theta_n)$:*

$$space(S_i^n) := \left\{ \begin{array}{l} (\underline{R_{i+1}}, .., \underline{R_n}) := (\theta_{i+1}, .., \theta_n) \in (Q^+)^{n-i} \mid \\ \forall e_n \in E_n, \forall j \in \{i, ..., n-1\} \quad \exists e_j \in E_j, \\ \quad e_i \overset{L(R_{i+1})}{\rightarrow} e_{i+1} ... e_j \overset{L(R_{j+1})}{\rightarrow} ... e_{n-1} \overset{L(R_n)}{\rightarrow} e_n \end{array} \right\}.$$

Notons que les variables $\underline{R_{i+1}}, .., \underline{R_n}$ sont dépendantes temporellement. De là, le vecteur $(\underline{R_{i+1}}, .., \underline{R_n}) := (\theta_{i+1}, .., \theta_n)$ est une solution admissible, s'il y a un état e_i dans E_i qui peut franchir consécutivement : le rendez-vous R_{i+1} à la date relative $\underline{R_{i+1}} := \theta_{i+1}$, R_{i+2} à la date relative $\underline{R_{i+2}} := \theta_{i+2}$, .., et finalement le rendez-vous R_n à la date relative $\underline{R_n} := \theta_n$ pour atteindre l'état e_n accessible dans E_n.

De plus, puisque une classe contient tous les états accessibles après franchissement de la même séquence, alors chaque vecteur $(\theta_{i+1}, .., \theta_n)$ de $space(S_i^n)$ dénotera un état accessible dans E_n.

Définition 8.3.2. *L'espace space(S_i^n) peut être exprimé en fonction de space(S_i^{n-1}), comme suit :*
- Si space$(S_i^{n-1}) \neq \emptyset$ alors

$$space(S_i^n) := \left\{ \begin{array}{l} (\underline{R_{i+1}}, .., \underline{R_n}) \mid (\underline{R_{i+1}}, .., \underline{R_n}) \in space(S_i^{n-1}) \\ \wedge \\ 0 \preceq Low_{n-1}(R_n) \preceq R_n \preceq \underset{\forall R \in e_{n-1}}{MIN} \{Up_{n-1}(R)\} \end{array} \right\}.$$

En d'autres termes, si R_n est franchissable dans $space(S_i^{n-1})$, alors $space(S_i^n)$ représente tous les vecteurs de $space(S_i^{n-1})$ (états de E_{n-1}), satisfaisant la condition de franchissement donnée en Définition 5.3.3, et la restriction de l'espace $space(S_i^n)$ aux variables $(R_{i+1}, .., R_{n-1})$ est un sous ensemble de $space(S_i^{n-1})$. Par conséquent, le calcul des temps de performance est déterminé récursivement en faisant des projections sur les espaces de franchissement calculés antérieurement. Nous calculons les temps de performances relatifs à la séquence S_i^n grâce à la fonction DS_n définie comme suit :

Définition 8.3.3. *(Fonction des distances de temps)*
Soit E_n une classe accessible dans GR, après le franchissement de la séquence $S_i^n = R_{i+1}, .., R_n$. Nous définissons au point (n) la fonction DS_n appelée fonction des distances de temps calculant les distances minimales et maximales de tout chemin allant du point $(i) \in \{0, .., n\}$ au point (n).

$DS_n : (\{0, .., n\} \cup E_n(M) \cup ER_n(M))^2 \rightarrow Q \cup \{\infty\}$
$\forall i \in \{0, .., n-1\}, \forall R \in ER_n(M), \forall t \in E_n(M)$
$DS_n[i, n] := \underset{space(S_i^n)}{MAX} \{\underline{R_{i+1}} + .. + \underline{R_n}\}$
$DS_n[n, i] := \underset{space(S_i^n)}{-MIN} \{\underline{R_{i+1}} + .. + \underline{R_n}\}$ $DS_n[i, R] := \underset{space(S_i^n)}{MAX} \{\underline{R_{i+1}} + .. + R_{\underline{n}} + Up_n(R)\}$
$DS_n[R, i] := \underset{space(S_i^n)}{-MIN} \{\underline{R_{i+1}} + .. + \underline{R_n} + Low_n(R)\}$
$DS_n[i, t] := \underset{space(S_i^n)}{MAX} \{\underline{R_{i+1}} + .. + \underline{R_n} + y_n(t)\}$
$DS_n[t, i] := \underset{space(S_i^n)}{-MIN} \{\underline{R_{i+1}} + .. + \underline{R_n} + x_n(t)\}$
$DS_n[i, i] := 0$
$DS_n[n, n] := 0$

Le calcul des précédentes distances est réalisé en considérant le vecteur optimal $(\theta_{i+1}, .., \theta_n)$ dans l'espace $space(S_i^n)$ tel que il existe un état de E_n accessible à partir d'un état de E_i après franchissement de la séquence $S_i^n = (R_{i+1}, .., R_n)$ aux dates relative $(\underline{R_{i+1}}, .., \underline{R_n}) := (\theta_{i+1}, .., \theta_n)$ par conséquent, si

t est une transition sensibilisée pour E_n, alors $DS_n[i,t]$ représente l'écart maximum entre le point (i) et la borne supérieure de t. $DS_n[t,i]$ dénote la valeur opposée de l'écart minimum entre le point de franchissement (i) et la borne inférieure de t. De même, si R est un rendez-vous sensibilisé pour E_n, alors $DS_n[i,R]$, représente l'écart maximum entre le point (i) et la borne supérieure de R. $DS_n[R,i]$ dénote la valeur opposée de l'écart minimum entre le point (i) et la borne inférieure de R. Finalement, $DS_n[i,n]$ (respectivement $DS_n[n,i]$), calcule la distance de temps maximum (respectivement, la valeur opposée de la distance de temps minimum), entre les points (i) et (n).

Les formules de calcul de la fonction DS_n sont données par la proposition suivante :

Proposition 8.3.1. *Soit $E_n = (M_n, D_n)$ une classe accessible dans GR, à partir de la classe $E_{n-1} = (M_{n-1}, D_{n-1})$ après le franchissement du rendez-vous R_n. La fonction DS_n associée à E_n est calculée récursivement à partir de DS_{n-1} comme suit :*

$DS_n[n,n] := 0.$

$DS_n[n,R] := \beta[\bullet, R]$

$$DS_n[n,R] := MIN \begin{cases} \underset{\forall t \in Set^+}{Op^+} \{D[\bullet,t\} \quad + \delta^+ \\ MAX \begin{cases} \underset{\forall t \in Set^+}{Op^+} \{D[\bullet,t]\} \\ \underset{\forall t \in Set^-}{Op^-} \{D[\bullet,t] + \varepsilon(t)\} \quad + \delta^- \\ \delta^- \end{cases} \\ \underset{\forall t \in Set^+}{MAX} \{D[\bullet,t]\} \end{cases}$$

$DS_n[R,n] := \beta[R, \bullet]$

$$DS_n[R,n] := \quad MIN \begin{cases} \underset{\forall t \in Set^-}{-Op^-} \{D[t,\bullet]\} \quad + \alpha^- \\ MAX \begin{cases} \underset{\forall t \in Set^+}{-Op^+} \{D[t,\bullet] - \varepsilon(t)\} \quad + \alpha^+ \\ \underset{\forall t \in Set^-}{-Op^-} \{D[t,\bullet]\} \end{cases} \\ \underset{\forall t \in Set^-}{MAX} \{D[t,\bullet]\} \end{cases}$$

$DS_n[n,t] := D_n[\bullet,t].$

$DS_n[t,n] := D_n[t,\bullet].$

$\forall i \in \{0,..,n-1\}$

$DS_n[i,n] := \underset{\forall R \in ER_{n-1}(M)}{MIN} \{DS_{n-1}[i,R]\}$

$DS_n[n,i] := DS_{n-1}[R_n, i]$

$\forall i \in \{0,..,n-1\}, \forall R \in ER_n(M)$

$DS_n[i,R] := \beta[i,R]$

$$DS_n[i,R] := MIN \begin{cases} \underset{\forall t \in Set^+}{Op^+} \{DS_n[i,t]\} \quad + \delta^+ \\ MAX \begin{cases} \underset{\forall t \in Set^+}{Op^+} \{DS_n[i,t]\} \\ \underset{\forall t \in Set^-}{Op^-} \{DS_n[i,t] + \varepsilon(t)\} \quad + \delta^- \\ D[i,\bullet] + \delta^- \end{cases} \\ \underset{\forall t \in Set^+}{MAX} \{DS_n[i,t]\} \end{cases}$$

$DS_n[R,i] :=$

$$MIN \begin{cases} \underset{\forall t \in Set^-}{-Op^-} \{DS_n[t,i]\} \quad + \alpha^- \\ MAX \begin{cases} \underset{\forall t \in Set^+}{-Op^+} \{DS_n[t,i] - \varepsilon(t)\} \quad + \alpha^+ \\ \underset{\forall t \in Set^-}{-Op^-} \{DS_n[t,i]\} \end{cases} \\ \underset{\forall t \in Set^-}{MAX} \{DS_n[t,i]\} \end{cases}$$

$\forall i \in \{0,..,n-1\}, \forall t \in E_n(M)$

- Si t est persistante

$DS_n[i,t] := MIN(DS_{n-1}[i,t], DS_n[i,n] + D_n[\bullet,t]).$

$DS_n[t,i] := MIN(DS_{n-1}[t,i], DS_n[n,i] + D_n[t,\bullet])$.

- Si t nouvellement sensibilisée
$DS_n[i,t] := DS_n[i,n] + LFT(t)$
$DS_n[t,i] := DS_n[n,i] - EFT(t)$

Preuve : La preuve est donnée en Annexe A.

Le calcul des temps de performance de la séquence S_i^i est entrepris en parcourant le chemin du graphe, partant du noeud représentant la classe (i) en réalisant à chaque noeud intermédiaire le calcul des éléments de la fonction $DS_j(j = i..n)$ jusqu'à atteindre le noeud final (n). A ce point, nous aurons calculé les coefficients $DS_n[n,i]$ et $DS_n[i,n]$ dénotant respectivement, la valeur opposée de la distance minimum de temps, et la distance maximum de temps entre les noeuds (i) et (n).

Remarque 8.3.1. *Le calcul de l'élément $DS_n[n-1,n]$ a été déjà réalisé lors du calcul de chaque classe du graphe (voir Proposition 7.2.2), et peut être sauvegardé comme paramètre de la classe pour éviter un calcul redondant.*

Pour illustrer la méthodologie de vérification présentée auparavant, nous considérons l'exemple présenté dans le Chapitre 6. (n). Les coefficients du système DS_0 sont définis comme suit :
- Nous avons le point $n = 0$ et $DS_0[0,0] := 0$
- Pour toute transitions sensibilisée dans E_0 : $DS_0[0,t] := LFT(t)$
$DS_0[t,0] := -EFT(t)$

Nous codons le système DS_0 par quatre matrices. Par exemple, les coefficient du système DS_0 de l'exemple du $RTPN$ du chapitre 6 figure est représenté par la table suivante :

$DS_0[i,t]$	t_1	t_2	t_4	t_5
0	1	5	2	4

$DS_0[t,i]$	t_1	t_2	t_4	t_5
0	0	-4	0	-2

$DS_0[i,n]$	0
0	0

$DS_0[n,i]$	0
0	0

TABLE 8.1 – Distance de temps au point (0)

Le calcul de la fonction DS pour les autres points se fait récursivement.

Pour simplifier les choses, nous considérons le chemin qui mène vers la classe E_{12} (voir Figure 7.2) du chapitre précédent. les Tables suivantes reproduisent les temps de performances de la séquence du chemin $(E_0 \to E_1 \to E_{12})$. La séquence de franchissement correspondante est $S_0^{12} = (R_1, R_4)$. Le temps minimal et maximal de cette séquence sont $[2,4]$.

$DS_1[i,t]$	t_1	t_2	t_5
0	1	5	4
$n = 1$	1	5	2

$DS_1[t,i]$	t_1	t_2	t_5
0	0	-4	-2
$n = 1$	0	-2	-2

$DS_1[i,n]$	0
$n = 1$	2

$DS_1[n,i]$	0
$n = 1$	0

TABLE 8.2 – Distance de temps au point (1)

$DS_2[i,t]$	t_1	t_3	t_5
0	1	4	4
$n = 1$	1	4	2
$n = 2$	0	2	0

$DS_2[t,i]$	t_1	t_3	t_5
0	0	-3	-2
$n = 1$	0	-3	-2
$n = 2$	0	-1	-2

$DS_2[i,n]$	0	1
$n = 12$	4	2

$DS_2[n,i]$	0	1
$n = 12$	-2	-2

TABLE 8.3 – Distance de temps au point (2)

8.4 Conclusion

Nous avons montré à travers ce chapitre comment exploiter le graphe des classes (obtenu à partir d'une spécification *RTPN*), pour la vérification de ses propriétés linéaires, et pour le calcul des temps minimum et maximum de n'importe quel chemin du graphe.

Chapitre 9

Conclusion Générale

9.1 Bilan

La vérification et l'analyse des propriétés *qualitatives* et *quantitatives* des systèmes temps réel est une tâche ardue du fait des sémantiques complexes qu'elles induisent. Les méthodes formelles ont longtemps été utilisées pour l'analyse de ces systèmes, parmi lesquelles celles basées sur les réseaux de Petri [92]. Principalement, les techniques d'analyse basées sur l'exploration de l'espace d'état de ces systèmes sont des solutions exhaustives, mais coûteuses ; néanmoins elles permettent de répondre aux propriétés de sûreté.

Nous avons tenté à travers cette thèse d'apporter un certain nombre de solutions à des problématiques différentes, mais néanmoins participant à la proposition d'une méthodologie formelle permettant de modéliser et d'analyser les systèmes temps réel complexes. La motivation vient du fait que, parmi les modèles basés (ou non), sur le principe des réseaux de Petri, aucun n'avait la puissance de spécification suffisante pour modéliser de tels systèmes.

Une fois les besoins bien définis, s'ensuit la proposition d'un modèle appelé *RTPN (Time Petri Nets with Rendezvous)*[112, 68, 18] a été élaborée permettant d'étendre les *RdPT* à différents mécanismes dont certains ont été précédemment introduits dans la littérature, tels que :

- la synchronisation : permettant le tir simultané et indivisible d'un ensemble de transitions appelé *rendezvous* selon un schéma de synchronisation prédéfini ; les stratégies de *Senac* [51], de *Little* [47] et autres ont été considérés ;

- les Délais : permettant de modéliser le retard et l'avance d'une transition par rapport aux autres transitions appartenant au même rendezvous. Ces paramètres temporels ont été déjà élaborés dans les travaux suivants : les modèles temporelles dd'*Allen* et *Wahl* [30, 31], en multimédia avec *Blakowski*[7] et *Owezarski*, les patterns de synchronisation de *Van Der Last* [70] .

Grâce à ce cadre formel, il est devenu possible de capturer et de modéliser avec plus de fidélité et de précision les comportements les plus complexes induits par les systèmes temps réel. D'ailleurs, nous avons montré comment notre modèle pouvait facilement spécifier de manière naturelle et implicite les exigences des workflow [68, 18].

Toutefois, le pouvoir d'expression de ce modèle ne pourrait être exploitable si aucune approche d'analyse de ses propriétés n'aurait été pourvue. Pour ce faire, nous avons développé une approche pour la construction du graphe des classes d'un $RTPN$.

L'approche en question permet de construire le graphe des classes d'un $RTPN$. Nous avons formulé l'expression d'une classe E au moyen du couple (M, D) ; où M est le marquage accessible et D est un système de contraintes utilisant les variables transitions. Cependant, le test de franchissement nécessite l'extension du système D aux contraintes relatives aux rendezvous sensibilisés. La complexité du calcul d'une classe est d'ordre exponentiel, facteur du nombre de variables présentes dans le système étendu ; augmentant la complexité de la construction du graphe lorsque ce dernier devient de taille importante. Nous avons montré que cette complexité de calcul, peut être réduite en ignorant certains calculs qui sont généralement redondants. En effet, le calcul de la classe E avec l'algorithme de FLOYD WARSHALL est de l'ordre de $O((n+m)^3)$ alors que notre algorithme a une complexité de l'ordre de $O(2n^2 + 2mn)$ où n (rep. m) représente les transitions (rep. les rendezvous) sensibilisées.

Le graphe ainsi construit préserve toutes les propriétés linéaires du modèle (e.g. accessibilité, vivacité, blocage,...etc), notamment celles inhérentes à la sûreté.

Par ailleurs, nous avons développé une technique permettant de déterminer à partir du graphe obtenu

les temps minimaux et maximaux de n'importe quelle sous séquence de franchissement. Ceci peut être exploité pour l'estimation quantitative des propriétés temps réel du modèle.

Enfin, tout au long de notre thèse, nous avons motivé nos différentes propositions par leur application dans la spécification des contraintes des systèmes de workflow.

9.2 Perspectives

Plusieurs travaux futures inhérents à cette thèse sont à envisager :

a) L'implémentation de l'outil d'édition graphique d'un $RTPN$ et mise en œuvre de l'approche proposée ;

b) Exploitation de notre technique pour la réduction de la taille des graphes des classes en supprimant les comportements induits par la sémantique d'interleaving ; Comparaison avec les technique de réduction par ordre partiel [85] .

c) Adaptation de l'approche d'énumération pour la génération de l'automate temporisé équivalent, voire du graphe des classes atomiques préservant les propriétés de branchement[84, 4] .

d) Etudes de cas de systèmes temps réels complexes incluant entre autres les applications des workflow.

e) Implémenter l'approche et la tester sur un cas d'étude.

Bibliographie

[1] A Abdelli. *Sémantique et validation d'un modèle basé sur les RdPT pour la spécification et l'analyse des systèmes temps réel complexes : Application aux systèmes multimédias.* PhD thesis, PhD thesis, Universite ? des Sciences et Technologies Houari Boumedienne, 2007.

[2] Bernard Berthomieu and Michel Diaz. Modeling and verification of time dependent systems using time petri nets. *IEEE transactions on software engineering*, 17(3) :259, 1991.

[3] Abdelli Abdelkrim and Nadjib Badache. Towards building the state class graph of the TSPN model. *Fundam. Informaticae*, 86(4) :371–409, 2008.

[4] Bernard Berthomieu and François Vernadat. State class constructions for branching analysis of time petri nets. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 442–457. Springer, 2003.

[5] Tadao Murata. Petri nets : Properties, analysis and applications. *Proceedings of the IEEE*, 77(4) :541–580, 1989.

[6] John A. Stankovic. Misconceptions about real-time computing : A serious problem for next-generation systems. *Computer*, 21(10) :10–19, 1988.

[7] G Blakowski and R Steinmetz. A media synchronization survey : reference model, specification, and case studies. *IEEE Journal on Selected Areas in Communications*, 14(1) :5–35, 2006.

[8] Carlo Combi and Roberto Posenato. Controllability in temporal conceptual workflow schemata. In *International Conference on Business Process Management*, pages 64–79. Springer, 2009.

[9] Khodakaram Salimifard, Seyed Yaghoub Hosseini, and Mohammad Sadegh Moradi. Improving emergency department processes using coloured petri nets. In *PNSE+ ModPE*, pages 335–349. Citeseer, 2013.

[10] François Defossez. *Modélisation discrète et formelle des exigences temporelles pour la validation et l'évaluation de la sécurité ferroviaire.* PhD thesis, 2010.

[11] Hanifa Boucheneb and Rachid Hadjidj. Ctl* model checking for time petri nets. *Theoretical Computer Science*, 353(1-3) :208–227, 2006.

[12] Hanifa Boucheneb and Hind Rakkay. A more efficient time petri net state space abstraction useful to model checking timed linear properties. *Fundamenta Informaticae*, 88(4) :469–495, 2008.

[13] Rachid Hadjidj and Hanifa Boucheneb. Much compact time petri net state class spaces useful to restore ctl* properties. In *Fifth International Conference on Application of Concurrency to System Design (ACSD'05)*, pages 224–233. IEEE, 2005.

[14] Rachid Hadjidj and Hanifa Boucheneb. Improving state class constructions for ctl* model checking of time petri nets. *International Journal on Software Tools for Technology Transfer*, 10(2) :167–184, 2008.

[15] Abdelli Abdelkrim and M. Daoudi. Towards SMIL document analysis using an algebraic time net. In Kiyoharu Aizawa, Yuichi Nakamura, and Shin'ichi Satoh, editors, *Advances in Multimedia Information Processing - PCM 2004, 5th Pacific Rim Conference on Multimedia, Tokyo, Japan, November 30 - December 3, 2004, Proceedings, Part III*, volume 3333 of *Lecture Notes in Computer Science*, pages 273–281. Springer, 2004.

[16] Philippe Owezarski and Marc Boyer. Modeling of multimedia architectures : the case of videoconferencing with guaranteed quality of service. *Petri Nets : Fundamental Models, Verification and Applications*, pages 501–525, 2009.

[17] Philip Meir Merlin. A study of the recoverability of computing systems. 1975.

[18] Abdia Hamdani and Abdelkrim Abdelli. Towards modelling and analyzing timed workflow systems with complex synchronizations. *Journal of King Saud University-Computer and Information Sciences*, 32(4) :491–504, 2020.

[19] Patricia Bouyer. Model-checking timed temporal logics. *Electron. Notes Theor. Comput. Sci.*, 231 :323–341, 2009.

[20] José Eduardo Rivera, Francisco Durán, and Antonio Vallecillo. Formal specification and analysis of domain specific models using maude. *Simul.*, 85(11-12) :778–792, 2009.

[21] Quan Z. Sheng and Boualem Benatallah. Contextuml : A uml-based modeling language for model-driven development of context-aware web services. In *2005 International Conference on Mobile Business (ICMB 2005), 11-13 July 2005, Sydney, Australia*, pages 206–212. IEEE Computer Society, 2005.

[22] William J Anderson. *Continuous-time Markov chains : An applications-oriented approach*. Springer Science & Business Media, 2012.

[23] Willliam Feller. *An introduction to probability theory and its applications, vol 2*. John Wiley & Sons, 2008.

[24] Olle Häggström et al. *Finite Markov chains and algorithmic applications*, volume 52. Cambridge University Press, 2002.

[25] Eerke A. Boiten. *Modeling in Event-B - System and Software Engineering* jean-raymond abrial cambridge university press, may 2010 ISBN-10 : 0521895561. *J. Funct. Program.*, 22(2) :217–219, 2012.

[26] J. Michael Spivey. *Z Notation - a reference manual (2. ed.)*. Prentice Hall International Series in Computer Science. Prentice Hall, 1992.

[27] Jonathan P. Bowen, Andreas Fett, and Michael G. Hinchey, editors. *ZUM '98 : The Z Formal Specification Notation, 11th International Conference of Z Users, Berlin, Germany, September 24-26, 1998, Proceedings*, volume 1493 of *Lecture Notes in Computer Science*. Springer, 1998.

[28] WMP van der Aalst. Information and process integration in enterprises : Rethinking documents, chapter 10 : Three good reasons for using a petri-net-based workflow management system, 1998.

[29] R. Alu and D.L. Dill. A theory for timed automata. *Theoretical Computer Science*, 126(2) :183–235, 1994.

[30] James F Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11) :832–843, 1983.

[31] Thomas Wahl and Kurt Rothermel. Representing time in multimedia systems. In *ICMCS*, pages 538–543, 1994.

[32] Ron Weiss, Andrzej Duda, and David K. Gifford. Composition and search with a video algebra. *IEEE Multim.*, 2(1) :12–25, 1995.

[33] Chérif Keramane and Andrzej Duda. Interval expressions - A functional model for interactive dynamic multimedia presentations. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems, ICMCS 1996, Hiroshima, Japan, June 17-23, 1996*, pages 283–286. IEEE Computer Society, 1996.

[34] Chander Ramchandani. *Analysis of asynchronous concurrent systems by timed Petri nets*. PhD thesis, Massachusetts Institute of Technology, 1973.

[35] Parosh Aziz Abdulla and Aletta Nylén. Timed petri nets and bqos. In *International Conference on Application and Theory of Petri Nets*, pages 53–70. Springer, 2001.

[36] David de Frutos Escrig, Valentín Valero Ruiz, and Olga Marroquín Alonso. Decidability of properties of timed-arc petri nets. In *International Conference on Application and Theory of Petri Nets*, pages 187–206. Springer, 2000.

[37] Wael Khansa. *Réseaux de Petri P-temporels : Contribution à l'étude des systèmes à événements discrets*. PhD thesis, Université Savoie Mont Blanc, 1997.

[38] Marc Boyer and Olivier H Roux. Comparison of the expressiveness of arc, place and transition time petri nets. In *International Conference on Application and Theory of Petri Nets*, pages 63–82. Springer, 2007.

[39] Florent Peres, Bernard Berthomieu, and François Vernadat. On the composition of time petri nets. *Discrete Event Dynamic Systems*, 21(3) :395–424, 2011.

[40]

[41] Giacomo Bucci and Enrico Vicario. Rapid prototyping through communicating petri nets. In *[1992 Proceedings] The Third International Workshop on Rapid System Prototyping*, pages 58–75. IEEE, 1992.

[42] Giacomo Bucci and Enrico Vicario. Compositional validation of time-critical systems using communicating time petri nets. *IEEE transactions on software engineering*, 21(12) :969–992, 1995.

[43] Laura Carnevali, Alessandro Pinzuti, and Enrico Vicario. Compositional verification for hierarchical scheduling of real-time systems. *IEEE Transactions on Software Engineering*, 39(5) :638–657, 2012.

[44] Didier Buchs and Nicolas Guelfi. A formal specification framework for object-oriented distributed systems. *IEEE Transactions on Software Engineering*, 26(7) :635–652, 2000.

[45] Giovanna Di Marzo Serugendo, Dino Mandrioli, Didier Buchs, and Nicolas Guelfi. Real-time synchronised petri nets. In *International Conference on Application and Theory of Petri Nets*, pages 142–162. Springer, 2002.

[46] Kamel Barkaoui, Hanifa Boucheneb, and Awatef Hicheur. Modelling and analysis of time-constrained flexible workflows with time recursive ecatnets. In *International Workshop on Web Services and Formal Methods*, pages 19–36. Springer, 2008.

[47] Thomas D. C. Little and Arif Ghafoor. Synchronization and storage models for multimedia objects. *IEEE journal on selected areas in communications*, 8(3) :413–427, 1990.

[48] Miae Woo, Naveed U. Qazi, and Akif Ghafoor. A synchronization framework for communication of pre-orchestrated multimedia information. *IEEE Netw.*, 8(1) :52–61, 1994.

[49] Sheng Uei Guan, Hsiao-Yeh Yu, and Jen-Shun Yang. A prioritized petri net model and its application in distributed multimedia systems. *IEEE Trans. Computers*, 47(4) :477–481, 1998.

[50] B Walter. Timed net for modeling and analysing protocols with time. In *Proceedings of the IFIP Conference on Protocol Specification Testing and Verification*, 1983.

[51] Michel Diaz and Patrick Sénac. Time stream petri nets a model for timed multimedia information. In *International Conference on Application and Theory of Petri Nets*, pages 219–238. Springer, 1994.

[52] Patrick Sénac, Michel Diaz, Alain Leger, and Pierre de Saqui-Sannes. Modeling logical and temporal synchronization in hypermedia systems. *IEEE journal on selected areas in communications*, 14(1) :84–103, 1996.

[53] JianQiang Li, YuShun Fan, and MengChu Zhou. Timing constraint workflow nets for workflow analysis. *IEEE Transactions on Systems, Man, and Cybernetics-Part A : Systems and Humans*, 33(2) :179–193, 2003.

[54] Nick Russell and Arthur HM Ter Hofstede. newyawl : Towards workflow 2.0. In *Transactions on Petri Nets and Other Models of Concurrency II*, pages 79–97. Springer, 2009.

[55] Jens Bæk Jørgensen, Kristian Bisgaard Lassen, and Wil MP van der Aalst. From task descriptions via colored petri nets towards an implementation of a new electronic patient record workflow system. *International Journal on Software Tools for Technology Transfer*, 10(1) :15–28, 2008.

[56] Franciny M Barreto and Stéphane Julia. Modeling of video games using workflow nets and state graphs. In *Proceedings of the 31st Brazilian Symposium on Software Engineering*, pages 261–266, 2017.

[57] Franco Cicirelli, Angelo Furfaro, and Libero Nigro. Using time stream petri nets for workflow modelling analysis and enactment. *Simulation*, 89(1) :68–86, 2013.

[58] Wil MP Van der Aalst. The application of petri nets to workflow management. *Journal of circuits, systems, and computers*, 8(01) :21–66, 1998.

[59] Huaiqing Wang and Qingtian Zeng. Modeling and analysis for workflow constrained by resources and nondetermined time : An approach based on petri nets. *IEEE Transactions on Systems, Man, and Cybernetics-Part A : Systems and Humans*, 38(4) :802–817, 2008.

[60] Cristiano Bertolini, Zhiming Liu, and Jiří Srba. Verification of timed healthcare workflows using component timed-arc petri nets. In *International Symposium on Foundations of Health Informatics Engineering and Systems*, pages 19–36. Springer, 2012.

[61] Yehia Thabet Kotb and Essameddin Badreddin. Synchronization among activities in a workflow using extended workflow petri nets. In *Seventh IEEE International Conference on E-Commerce Technology (CEC'05)*, pages 548–551. IEEE, 2005.

[62] Wil MP van der Aalst and Mathias Weske. Reflections on a decade of interorganizational workflow research. In *Seminal contributions to information systems engineering*, pages 307–313. Springer, 2013.

[63] Saida Boukhedouma, Zaia Alimazighi, and Mourad Oussalah. Adaptation and evolution frameworks for service based inter-organizational workflows. *International Journal of E-Business Research (IJEBR)*, 13(2) :28–57, 2017.

[64] Pedro M Gonzalez Del Foyo and José Reinaldo Silva. Using time petri nets for modelling and verification of timed constrained workflow systems. In *ABCM Symposium series in mechatronics*, volume 3, pages 471–478, 2008.

[65] Vijayalakshmi Atluri and Wei-Kuang Huang. A petri net based safety analysis of workflow authorization models 1. *Journal of Computer Security*, 8(2-3) :209–240, 2000.

[66] Yuyue Du and Changjun Jiang. Towards a workflow model of real-time cooperative systems. In *International Conference on Formal Engineering Methods*, pages 452–470. Springer, 2003.

[67] Hanifa Boucheneb and Kamel Barkaoui. Partial order reduction for checking soundness of time workflow nets. *Information Sciences*, 282 :261–276, 2014.

[68] Abdia Hamdani and Abdelkrim Abdelli. Using the rtpn model for the modeling of complex workflow systems. In *ICAASE*, pages 76–83, 2018.

[69] Wil M. P. van der Aalst. Everything you always wanted to know about petri nets, but were afraid to ask. In Thomas T. Hildebrandt, Boudewijn F. van Dongen, Maximilian Röglinger, and Jan Mendling, editors, *Business Process Management - 17th International Conference, BPM 2019, Vienna, Austria, September 1-6, 2019, Proceedings*, volume 11675 of *Lecture Notes in Computer Science*, pages 3–9. Springer, 2019.

[70] Wil MP Van Der Aalst and Arthur HM Ter Hofstede. Yawl : yet another workflow language. *Information systems*, 30(4) :245–275, 2005.

[71] Olivera Marjanovic and Maria E Orlowska. On modeling and verification of temporal constraints in production workflows. *Knowledge and Information Systems*, 1(2) :157–192, 1999.

[72] Claudio Bettini, X Sean Wang, and Sushil Jajodia. Temporal reasoning in workflow systems. *Distributed and Parallel Databases*, 11(3) :269–306, 2002.

[73] Johann Eder, Euthimios Panagos, and Michael Rabinovich. Time constraints in workflow systems. In Janis A. Bubenko Jr., John Krogstie, Oscar Pastor, Barbara Pernici, Colette Rolland, and Arne Sølvberg, editors, *Seminal Contributions to Information Systems Engineering, 25 Years of CAiSE*, pages 191–205. Springer, 2013.

[74] Marek Rusinkiewicz, Amit Sheth, and George Karabatis. Specifying interdatabase dependencies in a multidatabase environment. *Computer*, 24(12) :46–53, 1991.

[75] Elsa L Gunter, Ayesha Yasmeen, Carl A Gunter, and Anh Nguyen. Specifying and analyzing workflows for automated identification and data capture. In *2009 42nd Hawaii International Conference on System Sciences*, pages 1–11. IEEE, 2009.

[76] Javier Alfonso-Cendón, José M Fernández-de Alba, Rubén Fuentes-Fernández, and Juan Pavón. Implementation of context-aware workflows with multi-agent systems. *Neurocomputing*, 176 :91–97, 2016.

[77] Dennis M Riehle, Sven Jannaber, Arne Karhof, Oliver Thomas, Patrick Delfmann, and Jörg Becker. On the de-facto standard of event-driven process chains : How epc is defined in literature. *Modellierung 2016*, 2016.

[78] W Zhang, T Beaubouef, and H Ye. Statechart : A visual language for software requirement specification. *International Journal of Machine Learning and Computing*, 2(1) :52, 2012.

[79] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57. IEEE, 1977.

[80] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In Dexter Kozen, editor, *Logics of Programs, Workshop, Yorktown Heights, New York, USA, May 1981*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981.

[81] Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2) :244–263, 1986.

[82] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking for real-time systems. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science (LICS '90), Philadelphia, Pennsylvania, USA, June 4-7, 1990*, pages 414–425. IEEE Computer Society, 1990.

[83] Bernard Berthomieu and Miguel Menasche. An enumerative approach for analyzing time petri nets. In *Proceedings IFIP*. Citeseer, 1983.

[84] Tomohiro Yoneda and Hikaru Ryuba. Ctl model checking of time petri nets using geometric regions. *IEICE Transactions on Information and Systems*, 81(3) :297–306, 1998.

[85] J.Lilius. Efficient state space search for time petri nets. In *In MFCS Workshop on concurrency ?98 Vol 18 of Electronic Notes in Theoretical Computer Science. Elsevier*, 1999.

[86] Hanifa Boucheneb and Rachid Hadjidj. Towards optimal *ctl**model checking of time petri nets. *IFAC Proceedings Volumes*, 37(18) :459–464, 2004.

[87] Enrico Vicario. Static analysis and dynamic steering of time-dependent systems. *IEEE Trans. Software Eng.*, 27(8) :728–748, 2001.

[88]

[89] Franck Cassez and Olivier H Roux. Structural translation from time petri nets to timed automata. *Journal of Systems and Software*, 79(10) :1456–1468, 2006.

[90] Didier Lime and Olivier H. Roux. Model checking of time petri nets using the state class timed automaton. *Journal of Discrete Events Dynamic Systems - Theory and Applications (DEDS)*, 16(2) :179–205, 2006.

[91] Carl Adam Petri. Communication with automata [kommunikation mit automaten]. 1962.

[92] James L Peterson. Petri nets. *ACM Computing Surveys (CSUR)*, 9(3) :223–252, 1977.

[93] Kurt Jensen. *Coloured Petri nets : basic concepts, analysis methods and practical use*, volume 1. Springer Science & Business Media, 2013.

[94] Y Atamna. *Réseaux de Petri temporisés stochastiques classiques et bien formés : définition, analyse et application aux systèmes distribués temps réel.* PhD thesis, PhD thesis, Universit e Paul Sabatier de Toulouse, 1994.

[95] A Abdelli and Nadjib Badache. Synchronized transitions preemptive time petri nets : A new model towards specifying multimedia requirements. In *IEEE International Conference on Computer Systems and Applications, 2006.*, pages 17–24. IEEE, 2006.

[96] Michel Diaz Petri Nets. Fundamental models, verification and applications.–2010.

[97] Bernard Berthomieu. La méthode des classes d'états pour l'analyse des réseaux temporels. In *3e congres Modélisation des Systemes Réactifs (MSR'2001)*, pages 275–290, 2001.

[98] Hanifa Boucheneb, Didier Lime, and Olivier H Roux. On multi-enabledness in time petri nets. In *International Conference on Applications and Theory of Petri Nets and Concurrency*, pages 130–149. Springer, 2013.

[99] Neil D Jones, Lawrence H Landweber, and Y Edmund Lien. Complexity of some problems in petri nets. *Theoretical Computer Science*, 4(3) :277–299, 1977.

[100] Bernard Berthomieu and François Vernadat. Time petri nets analysis with TINA. In *Third International Conference on the Quantitative Evaluation of Systems (QEST 2006), 11-14 September 2006, Riverside, California, USA*, pages 123–124. IEEE Computer Society, 2006.

[101] Rachid Hadjidj and Hanifa Boucheneb. Efficient reachability analysis for time petri nets.

[102] Bernard Berthomieu, Didier Lime, Olivier H Roux, and François Vernadat. Problèmes d'accessibilité et espaces d'états abstraits des réseaux de petri temporels à chronomètres. *Journal européen des systemes automatisés*, 39(1/3) :223, 2005.

[103] Hanifa Boucheneb and Gérard Berthelot. Towards a simplified building of time petri nets reachability graph. In *Proceedings of the 5th International Workshop on Petri Nets and Performance Models, PNPM 1993, Toulouse, France, October 19-22, 1993*, pages 46–47. IEEE Computer Society, 1993.

[104] Bernard Berthomieu*, P-O Ribet, and François Vernadat. The tool tina–construction of abstract state spaces for petri nets and time petri nets. *International journal of production research*, 42(14) :2741–2756, 2004.

[105] Guillaume Gardey, Didier Lime, Morgan Magnin, and Olivier H. Roux. Romeo : A tool for analyzing time petri nets. In Kousha Etessami and Sriram K. Rajamani, editors, *Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings*, volume 3576 of *Lecture Notes in Computer Science*, pages 418–423. Springer, 2005.

[106] Rachid Hadjidj and H. Boucheneb. Rt-studio : A tool for modular design and analysis of realtime systems using interpreted time petri nets. In *PNSE+ModPE*, 2013.

[107] Rachid Hadjidj. *Analyse et validation formelle des systemes temps réel.* ProQuest, 2006.

[108] Hanifa Boucheneb and Rachid Hadjidj. Using inclusion abstraction to construct atomic state class graphs for time petri nets. *Int. J. Embed. Syst.*, 2(1/2) :128–139, 2006.

[109] Robert W Floyd. Algorithm 97 : shortest path. *Communications of the ACM*, 5(6) :345, 1962.

[110] Hanifa Boucheneb, Guillaume Gardey, and Olivier H. Roux. TCTL model checking of time petri nets. *J. Log. Comput.*, 19(6) :1509–1540, 2009.

[111] Mohammad Esmail Esmaili, Reza Entezari-Maleki, and Ali Movaghar. Improved region-based TCTL model checking of time petri nets. *J. Comput. Sci. Eng.*, 9(1) :9–19, 2015.

[112] Abdia Hamdani and Abdelkrim Abdelli. Time petri net with rendezvous. In *2017 4th International Conference on Control, Decision and Information Technologies (CoDIT)*, pages 0126–0131. IEEE, 2017.

[113] Andreas Lanz, Barbara Weber, and Manfred Reichert. Workflow time patterns for process-aware information systems. In *Enterprise, Business-Process and Information Systems Modeling*, pages 94–107. Springer, 2010.

[114] Marc Boyer. *Contribution à la modélisation des systèmes à temps contraint et application au multimédia.* PhD thesis, Toulouse 3, 2001.

[115] Olivier H Roux and Didier Lime. Time petri nets with inhibitor hyperarcs. formal semantics and state space computation. In *International Conference on Application and Theory of Petri Nets*, pages 371–390. Springer, 2004.

[116] Joel Toussaint, Francoise Simonot-Lion, and J-P Thomesse. Time constraint verification methods based on time petri nets. In *Proceedings of the Sixth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, pages 262–267. IEEE, 1997.

[117] Giacomo Bucci, Andrea Fedeli, Luigi Sassoli, and Enrico Vicario. Timed state space analysis of real-time preemptive systems. *IEEE transactions on software engineering*, 30(2) :97–111, 2004.

Chapitre 10

Annexe A : Preuves

Nous présentons ci-après les preuves des propositions et théorèmes énoncées dans cette thèse.

10.1 Preuve de la Proposition 7.2.1

Pour déterminer les formules de $\beta[t, R']$ et de $\beta[R', t]$ nous devons remplacer $Low(R')$ et $Up(R')$ par leurs expressions formelles respectives de la Définition 6.3.2.

(i) Cas de $\beta[t, R']$: Par rapport à la Définition 7.2.2, nous avons :
$\beta[t, R'] := \underset{\forall e \in E}{MAX} \{Up(R') - x(t)\}$.

Pour obtenir la formule de $\beta[t, R']$, nous avons besoin de remplacer $Up(R')$ par son expression formelle (Définition 6.3.2), nous obtenons alors :

$$\beta[t, R'] := MIN \begin{cases} \underset{\forall t' \in Set^+}{Op^+} \underset{\forall e \in E}{MAX} \{y(t') - x(t)\} \quad + \delta^+ \\ MAX \begin{cases} \underset{\forall t' \in Set^+}{Op^+} \underset{\forall e \in E}{MAX} \{y(t') - x(t)\} \\ \underset{\forall t' \in Set^-}{Op^-} \underset{\forall e \in E}{MAX} \{x(t') - x(t)\} \quad + \delta^- \end{cases} \\ \underset{\forall t' \in Set^+}{MAX} \underset{\forall e \in E}{MAX} \{y(t') - x(t)\} \end{cases}$$

Rappelons que $x(t') = MAX(0, y(t') + \varepsilon(t'))$, avec : $\varepsilon(t') = EFT(t') - LFT(t')$.

$$:= MIN \begin{cases} \underset{\forall t' \in Set^+}{Op^+} \underset{\forall e \in E}{MAX} \{y(t') - x(t)\} \quad + \delta^+ \\ MAX \begin{cases} \underset{\forall t' \in Set^+}{Op^+} \underset{\forall e \in E}{MAX} \{y(t') - x(t)\} \\ \underset{\forall t' \in Set^-}{Op^-} \underset{\forall e \in E}{MAX} \{y(t') - x(t) + \varepsilon(t')\} + \delta^- \\ \underset{\forall e \in E}{MAX} \{-x(t)\} + \delta^- \end{cases} \\ \underset{\forall t' \in Set^+}{MAX} \underset{\forall e \in E}{MAX} \{y(t') - x(t)\} \end{cases}$$

Et enfin :

$$\beta[t, R'] := MIN \begin{cases} \underset{\forall t' \in Set^+}{Op^+} \{D[t, t']\} \quad + \delta^+ \\ MAX \begin{cases} \underset{\forall t' \in Set^+}{Op^+} \{D[t, t']\} \\ \underset{\forall t' \in Set^-}{Op^-} \{D[t, t'] + \varepsilon(t')\} \quad + \delta^- \\ D[t, \bullet] + \delta^- \end{cases} \\ \underset{\forall t' \in Set^+}{MAX} \{D[t, t']\} \end{cases}$$

(ii) Cas de $\beta[R', t]$: de la même manière nous avons : $\beta[R', t] := \underset{\forall e \in E}{MAX} \{y(t) - Low(R')\}$, en remplaçant $Low(R')$ par son expression formelle (Définition 6.3.2), nous obtenons :

$$\beta[R',t] := MIN \begin{cases} \underset{\forall t' \in Set^-}{-Op^-}\ \underset{\forall e \in E}{MAX}\{y(t) - x(t')\} \quad + \alpha^- \\ MAX \begin{cases} \underset{\forall t' \in Set^+}{-Op^+}\ \underset{\forall e \in E}{MAX}\{y(t) - y(t')\} \quad + \alpha^+ \\ \underset{\forall t' \in Set^-}{-Op^-}\ \underset{\forall e \in E}{MAX}\{y(t) - x(t')\} \end{cases} \\ \underset{\forall t' \in Set^-}{-MIN}\ \underset{\forall e \in E}{MAX}\{y(t) - x(t')\} \end{cases}$$

$$:= MIN \begin{cases} \underset{\forall t' \in Set^-}{-Op^-}\ \underset{\forall e \in E}{MAX}\{y(t) - x(t')\} \quad + \alpha^- \\ MAX \begin{cases} \underset{\forall t' \in Set^+}{-Op^+}\ \underset{\forall e \in E}{MAX}\{y(t) - x(t') - \varepsilon(t')\} \quad + \alpha^+ \\ \underset{\forall t' \in Set^-}{-Op^-}\ \underset{\forall e \in E}{MAX}\{y(t) - x(t')\} \end{cases} \\ \underset{\forall t' \in Set^-}{MAX}\ \underset{\forall e \in E}{MAX}\{y(t) - x(t')\} \end{cases}$$

Et enfin :

$$\beta[R',t] := MIN \begin{cases} \underset{\forall t' \in Set^-}{-Op^-}\ \{D[t',t]\} \quad + \alpha^- \\ MAX \begin{cases} \underset{\forall t' \in Set^+}{-Op^+}\ \{D[t',t] - \varepsilon(t')\} \quad + \alpha^+ \\ \underset{\forall t' \in Set^-}{-Op^-}\ \{D[t',t]\} \end{cases} \\ \underset{\forall t' \in Set^-}{MAX}\ \{D[t',t]\} \end{cases}$$

10.2 Preuve de la Proposition 7.2.2

(i) Démontrons les formules de calcul du système $DR[R_f]$:

- Nous avons $DR[R_f] = \underset{\forall R \in ER(M)}{MIN}\ \underset{\forall e \in E}{MAX}\{Up(R) - Low(R_f)\}$, en remplaçant $Low(R_f)$ par son expression formelle (Définition 6.3.2), nous obtenons :

$$DR[R_f] := \underset{\forall R \in ER(M)}{MIN}\ \underset{\forall e \in E}{MAX}\{Up(R) - MAX \begin{cases} \underset{\forall t \in Set^-}{Op^-}\ \{x(t)\} \quad - \alpha^- \\ MIN \left(\underset{\forall t \in Set^+}{Op^+}\ \{y(t)\} - \alpha^+, \quad \underset{\forall t \in Set^-}{Op^-}\ \{x(t)\} \right) \\ \underset{\forall t \in Set^-}{MIN}\ \{x(t)\} \end{cases}$$

}

Ensuite, nous simplifions l'expression comme suit :

$$DR[R_f] := MIN \begin{cases} \underset{\forall R \in ER(M)}{MIN}\ \underset{\forall e \in E}{MAX}\{Up(R) - (\underset{\forall t \in Set^-}{Op^-}\ \{x(t)\} \quad - \alpha^-)\} \\ MAX \begin{cases} \underset{\forall R \in ER(M)}{MIN}\ \underset{\forall e \in E}{MAX}\{Up(R) - (\underset{\forall t \in Set^+}{Op^+}\ \{y(t)\} \quad - \alpha^+)\} \\ \underset{\forall R \in ER(M)}{MIN}\ \underset{\forall e \in E}{MAX}\{Up(R) - (\underset{\forall t \in Set^-}{Op^-}\ \{x(t)\})\} \end{cases} \\ \underset{\forall R \in ER(M)}{MIN}\ \underset{\forall e \in E}{MAX}\{Up(R) - (\underset{\forall t \in Set^-}{MIN}\ \{x(t))\}\} \end{cases}$$

Ensuite :

$$:= MIN \begin{cases} \underset{\forall t \in Set^-}{-Op^-}\ \underset{\forall R \in ER(M)}{MIN}\ \underset{\forall e \in E}{MAX}\{Up(R) - (\{x(t)\} \quad - \alpha^-)\} \\ MAX \begin{cases} \underset{\forall t \in Set^+}{-Op^+}\ \underset{\forall R \in ER(M)}{MIN}\ \underset{\forall e \in E}{MAX}\{Up(R) - (\{y(t)\} \quad - \alpha^+)\} \\ \underset{\forall t \in Set^-}{-Op^-}\ \underset{\forall R \in ER(M)}{MIN}\ \underset{\forall e \in E}{MAX}\{Up(R) - (\{x(t)\})\} \end{cases} \\ \underset{\forall t \in Set^-}{-MIN}\ \underset{\forall R \in ER(M)}{MIN}\ \underset{\forall e \in E}{MAX}\{Up(R) - (\{x(t))\}\} \end{cases}$$

Ensuite :

$$:= MIN \left\{ \begin{array}{l} -Op^- \underset{\forall t \in Set^-}{} \underset{\forall R \in ER(M)}{MIN} \underset{\forall e \in E}{MAX} \{Up(R) - (\{x(t)\} \quad - \alpha^-)\} \\ MAX \left\{ \begin{array}{l} -Op^+ \underset{\forall t \in Set^+}{} \underset{\forall R \in ER(M)}{MIN} \underset{\forall e \in E}{MAX} \{Up(R) - (\{x(t) + \varepsilon(t)\} \quad - \alpha^+)\} \\ -Op^- \underset{\forall t \in Set^-}{} \underset{\forall R \in ER(M)}{MIN} \underset{\forall e \in E}{MAX} \{Up(R) - (\{x(t)\})\} \end{array} \right. \\ -MIN \underset{\forall t \in Set^-}{} \underset{\forall R \in ER(M)}{MIN} \underset{\forall e \in E}{MAX} \{Up(R) - (\{x(t)\})\} \end{array} \right.$$

Ensuite :

$$:= MIN \left\{ \begin{array}{l} -Op^- \underset{\forall t \in Set^-}{} \underset{\forall R \in ER(M)}{MIN} \underset{\forall e \in E}{MAX} \{Up(R) - x(t)\} \quad + \alpha^- \\ MAX \left\{ \begin{array}{l} -Op^+ \underset{\forall t \in Set^+}{} \underset{\forall R \in ER(M)}{MIN} \underset{\forall e \in E}{MAX} \{Up(R) - x(t) - \varepsilon(t)\} \quad + \alpha^+ \\ -Op^- \underset{\forall t \in Set^-}{} \underset{\forall R \in ER(M)}{MIN} \underset{\forall e \in E}{MAX} \{Up(R) - x(t)\} \end{array} \right. \\ \underset{\forall t \in Set^-}{MAX} \underset{\forall R \in ER(M)}{MIN} \underset{\forall e \in E}{MAX} \{Up(R) - x(t)\} \end{array} \right.$$

$$:= MIN \left\{ \begin{array}{l} -Op^- \underset{\forall t \in Set^-}{} \underset{\forall R \in ER(M)}{MIN} \{\beta[R,t]\} \quad + \alpha^- \\ MAX \left\{ \begin{array}{l} -Op^+ \underset{\forall t \in Set^+}{} \underset{\forall R \in ER(M)}{MIN} \{\beta[R,t]\} - \varepsilon(t)\} \quad + \alpha^+ \\ -Op^- \underset{\forall t \in Set^-}{} \underset{\forall R \in ER(M)}{MIN} \{\beta[R,t]\} \end{array} \right. \\ \underset{\forall t \in Set^-}{MAX} \underset{\forall R \in ER(M)}{MIN} \{\beta[R,t]\} \end{array} \right.$$

Et enfin :

$$DR[R_f] := := MIN \left\{ \begin{array}{l} -Op^- \underset{\forall t \in Set^-}{} \{\lambda[t]\} \quad + \alpha^- \\ MAX \left\{ \begin{array}{l} -Op^+ \underset{\forall t \in Set^+}{} \{\lambda[t] - \varepsilon(t)\} \quad + \alpha^+ \\ -Op^- \underset{\forall t \in Set^-}{} \{\lambda[t]\} \end{array} \right. \\ \underset{\forall t \in Set^-}{MAX} \{\lambda[t]\} \end{array} \right.$$

(ii) Soit $E = [M, D]$ une classe de GR et β et DR les systèmes associés. Nous devons vérifier si l'accessibilité des états est préservée, ce qui revient à prouver que :

$$DR[R_f] = \underset{\forall R \in ER(M)}{MIN} \underset{\forall e \in E}{MAX} \{Up(R) - Low(R_f)\}$$

On a :

$$DR[R_f] := MIN \left\{ \begin{array}{l} -Op^- \underset{\forall t \in Set^-}{} \{\lambda[t]\} \quad + \alpha^- \\ MAX \left\{ \begin{array}{l} -Op^+ \underset{\forall t \in Set^+}{} \{\lambda[t] - \varepsilon(t)\} \quad + \alpha^+ \\ -Op^- \underset{\forall t \in Set^-}{} \{\lambda[t]\} \end{array} \right. \\ \underset{\forall t \in Set^-}{MAX} \{\lambda[t]\} \end{array} \right.$$

$$:= MIN \left\{ \begin{array}{l} -Op^- \underset{\forall t \in Set^-}{} \underset{\forall R \in ER(M)}{MIN} \{\beta[R,t]\} \quad + \alpha^- \\ MAX \left\{ \begin{array}{l} -Op^+ \underset{\forall t \in Set^+}{} \underset{\forall R \in ER(M)}{MIN} \{\beta[R,t]\} - \varepsilon(t)\} \quad + \alpha^+ \\ -Op^- \underset{\forall t \in Set^-}{} \underset{\forall R \in ER(M)}{MIN} \{\beta[R,t]\} \end{array} \right. \\ \underset{\forall t \in Set^-}{MAX} \underset{\forall R \in ER(M)}{MIN} \{\beta[R,t]\} \end{array} \right.$$

$$:= MIN \left\{ \begin{array}{l} -Op^- \underset{\forall t \in Set^-}{} \underset{\forall R \in ER(M)}{MIN} \underset{\forall e \in E}{MAX} \{Up(R) - x(t)\} \quad + \alpha^- \\ MAX \left\{ \begin{array}{l} -Op^+ \underset{\forall t \in Set^+}{} \underset{\forall R \in ER(M)}{MIN} \underset{\forall e \in E}{MAX} \{Up(R) - x(t) - \varepsilon(t)\} \quad + \alpha^+ \\ -Op^- \underset{\forall t \in Set^-}{} \underset{\forall R \in ER(M)}{MIN} \underset{\forall e \in E}{MAX} \{Up(R) - x(t)\} \end{array} \right. \\ \underset{\forall t \in Set^-}{MAX} \underset{\forall R \in ER(M)}{MIN} \underset{\forall e \in E}{MAX} \{Up(R) - x(t)\} \end{array} \right.$$

$$:= MIN \left\{ \begin{array}{l} -Op^- \underset{\forall t \in Set^-}{} \underset{\forall R \in ER(M)}{MIN} \underset{\forall e \in E}{MAX}\{Up(R) - (\{x(t)\} \quad - \alpha^-)\} \\ MAX \left\{ \begin{array}{l} -Op^+ \underset{\forall t \in Set^+}{} \underset{\forall R \in ER(M)}{MIN} \underset{\forall e \in E}{MAX}\{Up(R) - (\{x(t) + \varepsilon(t)\} \quad - \alpha^+)\} \\ -Op^- \underset{\forall t \in Set^-}{} \underset{\forall R \in ER(M)}{MIN} \underset{\forall e \in E}{MAX}\{Up(R) - (\{x(t)\})\} \end{array} \right. \\ -MIN \underset{\forall t \in Set^-}{} \underset{\forall R \in ER(M)}{MIN} \underset{\forall e \in E}{MAX}\{Up(R) - (\{x(t)\})\} \end{array} \right.$$

Ensuite :

$$:= MIN \left\{ \begin{array}{l} -Op^- \underset{\forall t \in Set^-}{} \underset{\forall R \in ER(M)}{MIN} \underset{\forall e \in E}{MAX}\{Up(R) - (\{x(t)\} \quad - \alpha^-)\} \\ MAX \left\{ \begin{array}{l} -Op^+ \underset{\forall t \in Set^+}{} \underset{\forall R \in ER(M)}{MIN} \underset{\forall e \in E}{MAX}\{Up(R) - (\{y(t)\} \quad - \alpha^+)\} \\ -Op^- \underset{\forall t \in Set^-}{} \underset{\forall R \in ER(M)}{MIN} \underset{\forall e \in E}{MAX}\{Up(R) - (\{x(t)\})\} \end{array} \right. \\ -MIN \underset{\forall t \in Set^-}{} \underset{\forall R \in ER(M)}{MIN} \underset{\forall e \in E}{MAX}\{Up(R) - (\{x(t)\})\} \end{array} \right.$$

$$DR[R_f] := MIN \left\{ \begin{array}{l} \underset{\forall R \in ER(M)}{MIN} \underset{\forall e \in E}{MAX}\{Up(R) - (\underset{\forall t \in Set^-}{Op^-} \{x(t)\} \quad - \alpha^-)\} \\ MAX \left\{ \begin{array}{l} \underset{\forall R \in ER(M)}{MIN} \underset{\forall e \in E}{MAX}\{Up(R) - (\underset{\forall t \in Set^+}{Op^+} \{y(t)\} \quad - \alpha^+)\} \\ \underset{\forall R \in ER(M)}{MIN} \underset{\forall e \in E}{MAX}\{Up(R) - (\underset{\forall t \in Set^-}{Op^-} \{x(t)\})\} \end{array} \right. \\ \underset{\forall R \in ER(M)}{MIN} \underset{\forall e \in E}{MAX}\{Up(R) - (\underset{\forall t \in Set^-}{MIN} \{x(t)\})\} \end{array} \right.$$

Et ceci, nous donne :
$:= \underset{\forall R \in ER(M)}{MIN} \underset{\forall e \in E}{MAX}\{Up(R) - Low(R_f)\}$

Reste à prouver maintenant que la condition de tir de la Proposition 7.2.2, est nécessaire et suffisante.
$\Longrightarrow$) Il existe au moins un état e accessible dans E qui permet le tir de R_f, cela veut dire qu'il satisfasse la condition de tir de la Définition 6.3.3 :
$Low(R_f) \preceq \underline{R_f} \preceq \underset{\forall R \in ER(M)}{MIN} Up(R) \Leftrightarrow \underset{\forall R \in ER(M)}{MIN} \{Up(R) - Low(R_f)\} \succeq 0.$

Par conséquent, s'il existe qu'un seul état qui satisfait la formule ci-dessus il doit maximiser : $Up(R) - Low(R_f)$ et donc nous écrivons : $\underset{\forall R \in ER(M)}{MIN} \underset{\forall e \in E}{MAX}\{Up(R) - Low(R_f)\} = DR[R_f] \succeq 0.$ et la condition de tir de la proposition 7.2.2 est déterminée.

$\Longleftarrow$) La condition de tir est satisfaite si $DR[R_f] \succeq 0 \Longleftrightarrow$
$\forall R \in ER(M) \underset{\forall e \in E}{MAX}\{Up(R) - Low(R_f)\} \succeq 0$. Par conséquent, il existe un état e accessible dans E qui satisfait $\forall R \in ER(M)\{Up(R) - Low(R_f)\} \succeq 0 \Longleftrightarrow \forall R \in ER(M), Low(R_f) \preceq Up(R) \Longleftrightarrow Low(R_f) \preceq \underset{\forall e \in E}{MIN}\{Up(R)\}$.
En conséquence, la condition de tir de la Définition 6.3.3 est déterminée et R_f est tirable à partir de e.

(iii) Démontrons les formules de calcul du système $D^{\uparrow}$
(1) Cas de $D^{\uparrow}[\bullet, t]$:
Nous remplaçons $D^{\uparrow}[\bullet, t]$ par son expression formelle $\underset{\forall e \in E^{\uparrow}}{MAX}\{y^{\uparrow}(t)\}$. Delà deux cas peuvent être observés selon l'âge de la sensibilisation de la transition t :

(a) Si t est persistante, alors nous avons selon la Définition 6.3.3, $y^{\uparrow}(t) := y(t) - \underline{R_f}$; nous écrivons : $D^{\uparrow}[\bullet, t] := \underset{\forall e \in E^{\uparrow}}{MAX}\{y(t) - \underline{R_f}\}$.

En conséquence, pour maximiser l'expression $y(t) - \underline{R_f}$ dans l'espace de $E^{\uparrow}$, nous devons minimiser la valeur de $\underline{R_f}$ dans E. Rappelons que les états appartenant à $E^{\uparrow}$ sont des états de E qui permettent le tir de R_f. Nous avons alors :
$(*) : Low(R_f) \preceq \underline{R_f} \preceq \underset{\forall R \in ER(M)}{MIN}\{Up(R)\}$ satisfaite. A cet effet, et pour maximiser $(y(t) - \underline{R_f})$, on a besoin de remplacer l'occurrence : $\underline{R_f}$ par :$Low(R_f)$ en maximisant l'expression de l'espace de E. Nous écrivons alors :
$D^{\uparrow}[\bullet, t] := \underset{\forall e \in E^{\uparrow}}{MAX}\{y(t) - Low(R_f)\} = \beta[R_f, t]$.

(b) Si t est nouvellement sensibilisée, alors selon la Définition 6.3.3,
$D^{\uparrow}[\bullet, t] := \underset{\forall e \in E^{\uparrow}}{MAX}\{y(t)^{\uparrow}\}$

$:= \underset{\forall e \in E^{\uparrow}}{MAX}\{LFT(t)\} = LFT(t).$

(2) Cas de $D^{\uparrow}[t, \bullet]$:
Remplaçons $D^{\uparrow}[t, \bullet]$ par son expression formelle, nous obtenons $D^{\uparrow}[t, \bullet] := \underset{\forall e \in E^{\uparrow}}{MAX}\{-x^{\uparrow}(t)\}$.

(a) Si t est persistante, alors nous avons selon la Définition 6.3.3.
$x^{\uparrow}(t) := MAX(0, x(t) - \underline{R_f})$. On écrit alors :
$D^{\uparrow}[t, \bullet] := \underset{\forall e \in E^{\uparrow}}{MAX}\{-MAX(0, x(t) - \underline{R_f})\}$
$D^{\uparrow}[t, \bullet] := MIN(0, \underset{\forall e \in E^{\uparrow}}{MAX}\{\underline{R_f} - x(t)\})$.

Selon la relation $(*)$, pour maximiser la valeur $\underline{R_f} - x(t)$ en $E^{\uparrow}$, nous avons besoin de remplacer $\underline{R_f}$ par $\underset{\forall R \in ER(M)}{MIN}\{Up(R)\}$ en maximisant l'expression dans l'espace E :
$D^{\uparrow}[t, \bullet]$
$:= MIN(0, \underset{\forall e \in ER(M)}{MIN} \underset{\forall e \in E}{MAX}\{Up(R) - x(t)\})$
$:= MIN(0, \underset{\forall e \in ER(M)}{MIN}\{\beta[t, R]\})$.

Ensuite , nous reconnaissons l'expression de $\lambda[t]$ et nous écrivons la formule suivante :
$D^{\uparrow}[t, \bullet] := MIN(0, \lambda[t])$.

(b) Si t est nouvellement sensibilisée, alors :
$D^{\uparrow}[t, \bullet] := \underset{\forall e \in E^{\uparrow}}{MAX}\{-x^{\uparrow}(t)\}$.
$:= \underset{\forall e \in E^{\uparrow}}{MAX}\{-EFT(t)\} = -LFT(t)$.

(3) Cas de $D^{\uparrow}[t_1, t_2]$:
Nous avons $D^{\uparrow}[t_1, t_2] := \underset{\forall e \in E^{\uparrow}}{MAX}\{y^{\uparrow}(t_2) - x^{\uparrow}(t_1)\}$.

(a) Si t_1 et t_2 sont persistantes, nous pouvons écrire :
$D^{\uparrow}[t_1, t_2]$
$:= \underset{\forall e \in E^{\uparrow}}{MAX}\{y^{\uparrow}(t_2) - MAX(0, x(t_1) - \underline{R_f})\}$.
$:= \underset{\forall e \in E^{\uparrow}}{MAX}\{y^{\uparrow}(t_2) + MIN(0, \underline{R_f} - x(t_1))\}$.
$:= \underset{\forall e \in E^{\uparrow}}{MAX}\{MIN(y^{\uparrow}(t_2), y^{\uparrow}(t_2) + \underline{R_f} - x(t_1))\}$.
$:= MIN(\underset{\forall e \in E^{\uparrow}}{MAX}\{y^{\uparrow}(t_2)\}, \underset{\forall e \in E^{\uparrow}}{MAX}\{y^{\uparrow}(t_2) + \underline{R_f} - x(t_1)\})$.

Remplaçons maintenant : $y^{\uparrow}(t_2)$ par :$y(t_2) - \underline{R_f}$, nous obtenons :
$D^{\uparrow}[t_1, t_2] := MIN(\underset{\forall e \in E^{\uparrow}}{MAX}\{y^{\uparrow}(t_2), \underset{\forall e \in E^{\uparrow}}{MAX}\{y^{\uparrow}(t_2) - \underline{R_f} - x(t_1) + \underline{R_f}\})$.
$D^{\uparrow}[t_1, t_2] :=$
$$MIN\begin{pmatrix} \underset{\forall e \in E^{\uparrow}}{MAX}\{y^{\uparrow}(t_2)\} \\ \underset{\forall e \in E^{\uparrow}}{MAX}\{y^{\uparrow}(t_2) - x(t_1))\} \end{pmatrix}$$

Puisque l'espace de tir de $E^{\uparrow}$ est inclut dans E, nous avons :
$$D^{\uparrow}[t_1, t_2] \preceq MIN\begin{pmatrix} \underset{\forall e \in E^{\uparrow}}{MAX}\{y^{\uparrow}(t_2)\} \\ \underset{\forall e \in E^{\uparrow}}{MAX}\{y^{\uparrow}(t_2) - x(t_1))\} \end{pmatrix}$$
$$D^{\uparrow}[t_1, t_2] \preceq MIN\begin{pmatrix} D[t_1, t_2] \\ D^{\uparrow}[\bullet, t_2] \end{pmatrix},$$
d'un autre coté, nous avons :
$D^{\uparrow}[t_1, t_2] := \underset{\forall e \in E^{\uparrow}}{MAX}\{y(t_2) - x(t_1))\}$
$\preceq \underset{\forall e \in E^{\uparrow}}{MAX}\{y(t_2)\} + \underset{\forall e \in E^{\uparrow}}{MAX}\{-x(t_1))\}$
$D^{\uparrow}[t_1, t_2] \preceq D^{\uparrow}[\bullet, t_2] + D^{\uparrow}[t_1, \bullet]$.
Nous prouvons prouver par induction que :
Si $D[t_1, t_2] \preceq D^{\uparrow}[\bullet, t_2] + D^{\uparrow}[t_1, \bullet]$
$D^{\uparrow}[t_1, t_2] := D[t_1, t_2]$

Sinon
$D^{\uparrow}[t_1, t_2] := D^{\uparrow}[\bullet, t_2] + D^{\uparrow}[t_1, \bullet]$
En conclusion, nous écrivons :
$D^{\uparrow}[t_1, t_2] := MIN(D[t_1, t_2], D^{\uparrow}[\bullet, t_2] + D^{\uparrow}[t_1, \bullet])$

(b) Si t_1 ou t_2 sont nouvellement sensibilisée, alors nous avons : $y^{\uparrow}(t_2) = LFT(t_2)$ ou $x^{\uparrow}(t_1) = EFT(t_1)$. Dans tout les cas, nous avons :
$D^{\uparrow}[t_1, t_2] := D^{\uparrow}[\bullet, t_2] + D^{\uparrow}[t_1, \bullet]$.

10.3 Preuve de la Proposition 8.3.1

1. Cas où $i = n$
   (a) $DS_n[n, n] := 0$; évident.
   (b) $DS_n[n, R] := \underset{space(S_i^n)}{MAX} \{Up_n(R)\}$

   En se basant sur les Définitions 8.3.1 et 8.3.2, il apparaît clairement que $space(S_i^n)$ détermine l'espace de E_n.
   $DS_n[n, R] := \underset{space(S_i^n)}{MAX} \{Up_n(R)\} = \underset{\forall e_n \in E_n}{MAX} \{Up_n(R)\} = \beta_n[\bullet, R]$.
   (c) $DS_n[R, n] := \underset{space(S_i^n)}{-MIN} \{Low_n(R)\} = \underset{\forall e_n \in E_n}{-MIN} \{Low_n(R)\} = \beta_n[R, \bullet]$.
   (d) $DS_n[n, t] := \underset{space(S_i^n)}{MAX} \{y_n(t)\} = \underset{\forall e_n \in E_n}{MAX} \{y_n(t)\} = D_n[\bullet, t]$
   (e) $DS_n[t, n] := \underset{space(S_i^n)}{-MIN} \{x_n(t)\} = \underset{\forall e_n \in E_n}{-MIN} \{x_n(t)\} = D_n[t, \bullet]$
2. Cas de $DS_n[i, n]$:
   Remplaçons $DS_n[i, n]$ par son expression formelle de la Définition **??** :
   $DS_n[i, n] = \underset{space(S_i^n)}{MAX} \{\underline{R_{i+1}}, ..., \underline{R_n}\}$

   Ensuite, en utilisant la Définition 8.3.2 , nous nous projetons sur l'espace $space(S_i^{n-1})$ en maximisant la valeur de $\underline{R_n}$ en lui substituant la valeur $\underset{\forall R \in E R_{n-1}}{MIN} \{Up_{n-1}(R)\}$

   Nous obtenons ensuite la valeur de $DS_n[i, n]$ comme suit :
   $$DS_n[i, n] := \underset{space(S_i^{n-1})}{MAX} \left\{ \underline{R_{i+1}}, ..., \underline{R_{n-1}} + \underset{\forall R \in E R_{n-1}}{MIN} Up_{n-1}(R) \right\}$$
   $$DS_n[i, n] := \left\{ \underset{\forall R \in E R_{n-1}}{MIN} \underset{space(S_i^{n-1})}{MAX} \underline{R_{i+1}}, ..., \underline{R_{n-1}} + Up_{n-1}(R) \right\}$$
   $$DS_n[i, n] := \underset{space(S_i^{n-1})}{MIN} \{DS_{n-1}[i, R]\}$$
3. Cas de $DS_n[n, i]$:
   Nous pouvons écrire : $DS_n[n, i] = \underset{space(S_i^n)}{-MIN} \{\underline{R_{i+1}}, ..., \underline{R_n}\}$.

   En utilisant la Définition 8.3.2 , nous nous projetons sur l'espace $space(S_i^{n-1})$ en minimisant la valeur de $\underline{R_n}$, ce qui revient à lui substituer $Low_{n-1}(R_n)$:
   $$DS_n[n, i] := \underset{space(S_i^{n-1})}{-MIN} \left\{ \underline{R_{i+1}}, ..., \underline{R_{n-1}} + Low_{n-1}(R_n) \right\}$$
   $$DS_n[n, i] := DS_{n-1}[R_n, i]$$
4. Cas de $DS_n[i, R]$:
   Nous écrivons : $DS_n[i, R] := \underset{space(S_i^n)}{MAX} \{\underline{R_{i+1}}, ..., \underline{R_n} + Up_n(R)\}$.

   Remplaçons $Up_n(R)$ par son expression formelle, nous obtenons :
   $DS_n[i, R] := \underset{space(S_i^n)}{MAX} \{\underline{R_{i+1}}, ..., \underline{R_n} +$
   $$MIN \left\{ \begin{array}{l} \underset{\forall t \in Set^+}{Op^+} \{y_n(t)\} \quad + \delta^+ \\ MAX \left\{ \begin{array}{l} \underset{\forall t \in Set^+}{Op^+} \{y_n(t)\} \\ \underset{\forall t \in Set^-}{Op^-} \{x_n(t)\} \quad + \delta^- \end{array} \right. \\ \underset{\forall t \in Set^+}{MAX} \{y_n(t)\} \end{array} \right. \}$$

$$DS_n[i,R] := MIN \begin{cases} \underset{\forall t \in Set^+}{Op^+} \underset{space(S_i^n)}{MAX} \{\underline{R_{i+1}}, ..., \underline{R_n} + y_n(t)\} & + \delta^+ \\ MAX \begin{cases} \underset{\forall t \in Set^+}{Op^+} \underset{space(S_i^n)}{MAX} \{\underline{R_{i+1}}, ..., \underline{R_n} + y_n(t)\} \\ \underset{\forall t \in Set^-}{Op^-} \underset{space(S_i^n)}{MAX} \{\underline{R_{i+1}}, ..., \underline{R_n} + x_n(t)\} & + \delta^- \end{cases} \\ \underset{\forall t \in Set^+}{MAX} \underset{space(S_i^n)}{MAX} \{\underline{R_{i+1}}, ..., \underline{R_n} + y_n(t)\} \end{cases}$$

Rappelons que $x(t) = MAX(0, y(t) + \varepsilon(t))$, avec : $\varepsilon(t) = EFT(t) - LFT(t)$.

$$DS_n[i,R] := MIN \begin{cases} \underset{\forall t \in Set^+}{Op^+} \{DS_n[i,t]\} & + \delta^+ \\ MAX \begin{cases} \underset{\forall t \in Set^+}{Op^+} \{DS_n[i,t]\} \\ \underset{\forall t \in Set^-}{Op^-} \{DS_n[i,t] + \varepsilon(t)\} & + \delta^- \\ D[i,\bullet] + \delta^- \end{cases} \\ \underset{\forall t \in Set^+}{MAX} \{DS_n[i,t]\} \end{cases}$$

5. Cas de $DS_n[R,i]$:
   $DS_n[R,i] = - \underset{space(S_i^n)}{MIN} \{\underline{R_{i+1}}, ..., \underline{R_n} + Low_n(R)\}$.

   Remplaçons $Low_n(R)$ par son expression formelle, nous obtenons :
   $DS_n[R,i] := - \underset{space(S_i^n)}{MIN} \{\underline{R_{i+1}}, ..., \underline{R_n} +$

$$MAX \begin{cases} \underset{\forall t \in Set^-}{Op^-} \{x_n(t)\} & - \alpha^- \\ MIN \begin{cases} \underset{\forall t \in Set^+}{Op^+} \{y_n(t)\} & - \alpha^+ \\ \underset{\forall t \in Set^-}{Op^-} \{x_n(t)\} \end{cases} & \} . \\ \underset{\forall t \in Set^-}{MIN} \{x_n(t)\} \end{cases}$$

$$DS_n[R,i] := MIN \begin{cases} \underset{\forall t \in Set^-}{-Op^-} \underset{space(S_i^n)}{-MIN} \{\underline{R_{i+1}}, ..., \underline{R_n} + x_n(t)\} & + \alpha^- \\ MAX \begin{cases} \underset{\forall t \in Set^+}{-Op^+} \underset{space(S_i^n)}{-MIN} \{\underline{R_{i+1}}, ..., \underline{R_n} + y_n(t)\} & + \alpha^+ \\ \underset{\forall t \in Set^-}{-Op^-} \underset{space(S_i^n)}{-MIN} \{\underline{R_{i+1}}, ..., \underline{R_n} + x_n(t)\} \end{cases} \\ \underset{\forall t \in Set^-}{-MIN} \underset{space(S_i^n)}{-MIN} \{\underline{R_{i+1}}, ..., \underline{R_n} + x_n(t)\} \end{cases}$$

$$DS_n[R,i] := MIN \begin{cases} \underset{\forall t \in Set^-}{-Op^-} \{DS_n[t,i]\} & + \alpha^- \\ MAX \begin{cases} \underset{\forall t \in Set^+}{-Op^+} \{DS_n[t,i] - \varepsilon(t)\} & + \alpha^+ \\ \underset{\forall t \in Set^-}{-Op^-} \{DS_n[t,i]\} \end{cases} \\ \underset{\forall t \in Set^-}{MAX} \{DS_n[t,i]\} \end{cases}$$

6. Cas de $DS_n[i,t]$:
   (a) Si t est persistante au point $(n-1)$:
   $DS_n[i,t] = \underset{space(S_i^n)}{MAX} \{\underline{R_{i+1}}, ..., \underline{R_n} + y_n(t)\}$

   Nous remplaçons $y_n(t)$ par $y_{n-1}(t) - R_n$; nous éliminons ensuite la variable $\underline{R_n}$
   $DS_n[i,t] = \underset{space(S_i^n)}{MAX} \{\underline{R_{i+1}}, ..., \underline{R_n} + (y_{n-1}(t) - \underline{R_n})\}$
   $DS_n[i,t] = \underset{space(S_i^n)}{MAX} \{\underline{R_{i+1}}, ..., \underline{R_{n-1}} + y_{n-1}(t)\}$

   D'un coté nous avons, $DS_n[i,t] \preceq \underset{space(S_i^{n-1})}{MAX} \{\underline{R_{i+1}}, ..., \underline{R_{n-1}} + y_{n-1}(t)\} = DS_{n-1}[i,t]$

   De l'autre coté : $DS_n[i,t] \preceq \underset{space(S_i^n)}{MAX} \{\underline{R_{i+1}}, ..., \underline{R_n}\} + \underset{space(S_i^n)}{MAX} \{y_n(t)\}$

   Et nous prouvons par induction que :
   $DS_n[i,t] = DS_{n-1}[i,t] \qquad DS_{n-1}[i,t] \preceq DS_n[i,n] + D[\bullet,t]$
   $DS_n[i,t] = DS_n[i,n] + D[\bullet,t]$ sinon
   d'où la formule :
   $DS_n[i,t] = MIN(DS_{n-1}[i,t]; \qquad DS_n[i,n] + D[\bullet,t])$.

(b) Si t est nouvellement sensibilisée :
$DS_n[i,t] = \underset{space(S_i^n)}{MAX}\{\underline{R_{i+1}}, ..., \underline{R_n} + y_n(t)\}$

Tenant en compte que $y_n(t) = Max(t)$, nous écrivons :
$DS_n[i,t] := \underset{space(S_i^n)}{MAX}\{\underline{R_{i+1}}, ..., \underline{R_n}\} + max(t) = DS_n[i,n] + max(t)$

7. Cas de $DS_n[t,i]$:
(a) Si t est persistante au point $n-1$:
$DS_n[t,i] := -\underset{space(S_i^n)}{MIN}\{\underline{R_{i+1}}, ..., \underline{R_n} + x_n(t)\}$.

Nous pouvons constater que :
$DS_n[t,i] \preceq \underset{space(S_i^n)}{-MIN}\{\underline{R_{i+1}}, ..., \underline{R_n}\}\ \underset{space(S_i^n)}{-MIN}\{x_n(t)\}$.
$DS_n[t,i] \preceq DS_n[n,i] + D_n[t,\bullet]$
Par ailleurs, nous remplaçons $x_n(t)$ par $MAX(0, x_{n-1}(t) - \underline{R_n})$ et nous obtenons :
$DS_n[t,i] = -\underset{space(S_i^n)}{MIN}\{\underline{R_{i+1}}, ..., \underline{R_n} + MAX(0, x_{n-1}(t) - \underline{-R_n})\}$.
$DS_n[t,i] :=$
$$MIN\begin{cases} \underset{space(S_i^n)}{-MIN}\{\underline{R_{i+1}}, ..., \underline{R_{n-1}} + x_{n-1}(t)\} \\ \underset{space(S_i^n)}{-MIN}\{\underline{R_{i+1}}, ..., \underline{R_n}\} \end{cases}$$
Comme l'espace $space(S_{n-1})$ inclut celui de $space(S_n)$.
$$DS_n[t,i] \preceq MIN\begin{cases} \underset{space(S_i^n)}{-MIN}\{\underline{R_{i+1}}, ..., \underline{R_{n-1}} + x_{n-1}(t)\} \\ DS_n[n,i] \end{cases}$$
$$DS_n[t,i] \preceq MIN\begin{cases} DS_{n-1}[t,i] \\ DS_n[n,i] \end{cases}$$
Ensuite, nous prouvons par induction que :
$$DS_n[t,i] \preceq MIN\begin{cases} DS_{n-1}[t,i] & \text{si } DS_{n-1}[t,i] \preceq D_n[\bullet,i] + DS_n[n,i] \\ D_n[\bullet,i] + DS_n[n,i] & \text{sinon} \end{cases}$$
De là nous déduisons la formule :
$DS_n[t,i] :=$
$$MIN\begin{cases} DS_{n-1}[t,i] \\ D_n[\bullet,i] + DS_n[n,i] \end{cases}$$
(b) Si t est nouvellement sensibilisée :
$DS_n[t,i] := -\underset{space(S_i^n)}{MIN}\{\underline{R_{i+1}}, ..., \underline{R_n} + x_n(t)\}$.

Nous avons, $x_n(t) := min(t)$; ce qui revient à écrire :
$DS_n[t,i] = -\underset{space(S_i^n)}{MIN}\{\underline{R_{i+1}}, ..., \underline{R_n}\} - min(t) = DS_n[n,i] - min(t)$.