N° d'ordre : ????

THESE

Vers l'utilisation des réseaux de Petri temporels étendus pour la vérification de systèmes temps-réel décrits en RT-LOTOS

Présentée pour obtenir

Le titre de Docteur de l'Institut National Polytechnique de Toulouse

École doctorale : **Systèmes**
Spécialité : **Systèmes Informatiques**

Par :

Tarek SADANI

Soutenue le 03 mai 2007 devant le jury composé de :

M.	*Jean-Pierre COURTIAT*	Directeur de thèse
M.	*Pierre de SAQUI-SANNES*	Co-Directeur de thèse
M.	*Richard CASTANET*	Rapporteur
M.	*Elie NAJM*	Rapporteur
M.	*Hubert GARAVEL*	Membre
M.	*Serge HADDAD*	Membre
M.	*Patrick SALLE*	Membre
M.	*François VERNADAT*	Membre

Remerciements

Je tiens à remercier ici Monsieur *Darrenougué*, directeur de l'ENSICA pour m'avoir accueilli en tant que doctorant au sein de cette Ecole. Mes remerciements vont de pair à Messieurs *Ghallab* et *Chatila* directeurs successifs du LAAS-CNRS pour m'avoir accueilli dans leur laboratoire.

Le partenariat de recherche qui lie l'ENSICA et le LAAS-CNRS trouve une déclinaison naturelle dans les relations qu'entretiennent le Département Mathématiques appliquées et Informatique de l'ENSICA et le groupe Outils et Logiciels pour la communication du LAAS-CNRS. Je remercie *Jean-Pierre Courtiat* responsable du groupe OLC du LAAS-CNRS et *Patrick Sénac*, chef du département DMI de l'ENSICA, pour m'avoir accueilli et offert un cadre privilégié pour le travail et ce depuis mon stage de DEA en 2003. Ces remerciements s'étendent à *François Vernadat* qui a pris les rênes du groupe OLC en septembre 2005.

Je tiens à remercier chaleureusement mes deux directeurs de thèse : *Jean-Pierre Courtiat* et *Pierre de Saqui-Sannes*. Leur bonne humeur, leurs encouragements m'ont permis de travailler dans les meilleures conditions. *Jean-Pierre Courtiat* n'a pas hésité à faire le déplacement depuis le Brésil pour assister à la soutenance, et ce malgré ses nouvelles obligations diplomatiques. Pendant toutes ces années *Pierre de Saqui-Sannes* m'a toujours accueilli dans son bureau avec le sourire. Je veux donc le remercier pour sa gentillesse et sa grande disponibilité.

Je remercie *Elie Najm* et *Richard Castanet* qui ont accepté la charge de rapporteurs. Ils ont aussi accepté d'être membres de mon jury, qu'ils trouvent ici le témoignage de ma reconnaissance.

Je remercie *Patrick Sallé* de m'avoir fait l'honneur de présider le jury.

Hubert Garavel, *Serge Haddad* et *François Vernadat* m'ont fait l'honneur d'examiner ce travail. Je suis très sensible à l'attention particulière qu'ils ont portée à la lecture de ce document. Leur présence dans le jury fut pour moi un grand plaisir.

Je remercie chaleureusement *Marc Boyer*, pour l'attention qu'il a portée à cette thèse. Ses observations judicieuses et ses remarques ont été dés plus déterminantes dans la conduite de ce travail.

J'adresse mes plus sincères remerciements à *Bernard Berthomieu*, qui a bien voulu répondre aux nombreuses sollicitations que je lui ai adressées pour réaliser ce travail.

Les personnels du DMI par leur gentillesse et la convivialité dont ils ont su faire montre ont rendu mon séjour très agréable parmi eux. Par ces quelques lignes je tiens à remercier toute l'équipe d'enseignants-chercheurs informaticiens du DMI : *Laurent Dairaine*, *Fabrice Frances*, *Jérome Lacan* et *Tanguy Perennou*. Par leurs grandes compétences et leurs remarquables qualités humaines, ils ont su rendre très chaleureuses et fructueuses

ces quelques années passées parmi eux. Je garde de très bons souvenirs des nombreuses discussions, souvent dans le couloir, avec *Yves Caumel*, pendant lesquelles j'ai pu profiter de ses grandes connaissances mathématiques et de ses idées enrichissantes qui m'ont conforté dans le sentiment que je ne savais rien. Je remercie aussi le personnel administratif du DMI dont la courtoisie et la disponibilité a été constante à mon égard. Je salue bien bas *René Espourteau*, chevalier de la Légion d'Honneur. Ma grande reconnaissance va également à *Bernard Jarlan* qui a grandement contribué à la préparation de la soutenance.

Je n'oublie pas de remercier ici les membres du groupe OLC avec qui j'ai passé des moments très agréables. Qu'ils trouvent ici collectivement et individuellement l'expression de toute ma gratitude.

Je ne saurais laisser passer cette occasion sans saluer les doctorants actuels et anciens que j'ai pu côtoyer au cours de ces quelques années. Je salue donc les vieux docteurs : *Ernesto*, *Laurent*, et *Florestan*. Les jeunes docteurs : *Mathieu* et *Manu* et les doctorants *Hervé*, *Jérome*, *Juan*, *Ahlem*, *Ali*, *Amine*, *Lei* et *Nounou*. Au moment d'écrire ces lignes, j'ai une pensée amicale pour *Ben*, mon collègue de bureau qui a su supporter dans un stoïcisme non dénué de grandeur, un humour dont je confesse le caractère parfois approximatif.

Je me dois de saluer comme il se doit tous ceux qui partagent avec moi cette passion pour le ballon rond. A tout ceux qui m'appellent *Tarekinho*, en club ou à l'ENSICA, je dis merci. Vous m'avez tous aidés à me sentir bien à Toulouse. A *Jérome Lacan*, je voudrais faire part de mes sentiments vibrants d'estime. Merci d'avoir donné du relief à mon palmarès Toulousain.

Il me semble opportun d'avoir une pensée à l'ensemble des étudiants Ensica que j'ai eu le plaisir d'avoir en cours tout au long de ces trois années et demie. Merci d'avoir supporté mes moments d'euphorie et mes changements d'humeurs occasionnés par ce travail. *Ernest*, *Julie* et *Sylvain* merci pour la jolie banderole que vous aviez brandie pendant et après la soutenance.

Je conclurai en exprimant toute ma gratitude à ma famille et à ma femme *Gaëlle*, dont le soutien a été essentiel tout au long de ces années de thèse. Ils savent tout ce qu'ils représentent pour moi.

Table des matières

Table des figures

Liste des tableaux

1 Introduction

"One of my most productive days was throwing away 1000 lines of code."
Ken Thompson

1.1 Systèmes temps-réel

Le temps réel touche des domaines très variés de la vie quotidienne. On le retrouve dans de multiples applications tels que le guidage ou la navigation (de bateaux, d'avions, de trains, d'automobiles), le contrôle de processus industriels, la robotique, les télécommunications et même les transactions bancaires.

De façon informelle, un système temps réel est défini comme un système dont l'exactitude du comportement dépend, non seulement de l'exactitude des traitements effectués, mais également du moment où les résultats de ces traitements sont produits. Au caractère interactif des relations système/environnement et au nécessaire respect des échéances temporelles, nombre de systèmes temps réel ajoutent des mécanismes de suspension et reprise sur contexte.

Dans le cadre de cette thèse nous convenons de mettre dans la catégorie *système temps-réel* toute entité dont le comportement est observé par rapport à une dimension temporelle. Par ailleurs nous appelons *système* un ensemble d'activités correspondant à un ou plusieurs traitements effectués en séquence ou en parallèle (en s'échangeant à l'occasion des informations).

Ces systèmes dits *temps-réel* ont souvent un caractère critique, car un dysfonctionnement de leur part peut entraîner la perte de vies humaines ou des dégâts matériels considérables. C'est pourquoi les éventuelles erreurs doivent être détectées le plus tôt possible dans le processus de conception. Le caractère fugitif de ces erreurs rajoute à la complexité du problème de vérification. L'utilisation de méthodes de description et vérification formelle, assistées par des outils informatiques adaptés devient de fait hautement recommandable si ce n'est indispensable. Les méthodes formelles peuvent être utilisées pour aider au développement de tels systèmes en fournissant un cadre formel dans lequel la vérification de la correction de leur comportement peut être effectuée.

Classiquement, on distingue deux grandes familles de techniques pour la vérification formelle.

1. Les techniques de preuve *"theorem proving"* qui sont des démonstrations mathématiques au sens usuel.

2. Les techniques de vérification de modèles, ou *model checking* qui, à partir d'une description formelle d'un système, construisent l'espace des états atteignables puis permettent de vérifier le respect des propriétés attendues du système en parcourant l'ensemble de ces états. Les contributions de cette thèse s'inscrivent dans le cadre de cette deuxième famille de techniques.

La technique du *model checking* dite « *presse bouton* » (pour son caractère automatique), présente l'avantage indéniable de permettre une détection rapide et économique des erreurs et ce dès les premières phases du processus de conception. Ce qui explique sans doute en partie la popularité grandissante dont elle jouit dans le monde industriel.

En élaborant un modèle du système, les concepteurs doivent veiller à offrir à la fois une précision suffisante pour préserver toutes les propriétés à vérifier et un niveau d'abstraction adéquat pour ne pas pénaliser la phase d'analyse. Il existe dans la littérature de multiples formalismes proposés pour la modélisation des systèmes temps-réel. Chacun possède ses caractéristiques spécifiques, plus ou moins pertinentes dans une conception spécifique et selon le type d'analyse envisagé.

Dans la suite de ce chapitre nous présentons deux techniques éprouvées pour la modélisation et la vérification des systèmes temps réel, techniques que nous utiliserons dans le cadre de cette thèse : les algèbres de processus et les réseaux de Petri.

1.2 Les algèbres de processus

Les algèbres de processus sont un formalisme mathématique pour la description et l'étude des systèmes concurrents. De manière générale, une algèbre de processus peut être considérée comme un langage formel doté d'une syntaxe et d'une sémantique. Cette dernière est définie formellement par le biais d'une axiomatique ou d'une sémantique opérationnelle. La syntaxe comporte quant à elle un ensemble restreint d'opérateurs (composition parallèle ou séquentielle, choix non déterministe) qui permettent la description de comportement complexes. Ces concepts fondamentaux des algèbres de processus, lorsqu'ils sont combinés avec des langages appropriés pour la description des types de données, fournissent des solutions techniques supérieures aux autres formalismes [36], en particulier de par l'expressivité offerte aux utilisateurs et le large spectre de leurs applications. Deux algèbres de processus ont été normalisées à l'ISO à savoir LOTOS [50] et E-LOTOS [51].

En définissant CCS [61], Milner a été le premier à isoler les concepts des algèbres de processus. Les recherches sur les algèbres de processus étendues avec une notion quantitative du temps ont commencé avec les travaux de Reed et Roscoe [76] sur une variante de CSP [46]. Les algèbres CCS, ACP [8] et ATP [65] ont aussi fait l'objet de travaux visant à les enrichir avec des informations temporelles. Le langage RT-LOTOS [28] qui fait l'objet de cette thèse appartient à cette famille d'algèbres temporisées. Pour une grande partie de ces algèbres, l'outillage a été réalisé par traduction vers les automates temporisés, ce qui a permis de réutiliser des outils comme KRONOS [89] ou UPAAL [13].

1.3 Les réseaux de Petri

Les réseaux de Petri offrent un support graphique et une base sémantique claire pour la description des systèmes concurrents. Ce formalisme pionnier, introduit par C.A. Petri en 1962, afin de modéliser la composition et la communication entre automates, a fourni les

premières approches de modélisation. Par le lien étroit avec les machines à états, les réseaux de Petri présentent un avantage indéniable en matière d'analyse de comportements.

Parmi la pléiade de modèles existants pour la modélisation des systèmes temps réel, nous en retenons qui ont été développés à partir des réseaux de Petri :

- Les réseaux de Petri temporisés introduits par Ramchandani [75] qui associent une durée de tir à chaque transition. Ces réseaux ainsi que divers modèles similaires sont essentiellement utilisés pour l'analyse de performances.
- Les réseaux de Petri temporels introduits par Merlin [60] que nous avons retenus dans le cadre de cette thèse pour leur caractère général sont quant à eux bien adaptés à l'analyse de comportements. Les réseaux temporels associent un intervalle de tir aux transitions.

1.4 Contributions

Ce travail de thèse est centré sur la vérification de descriptions RT-LOTOS. Les principales contributions sont :

- **Définition et formalisation des schémas de traduction d'un large sous-ensemble de la partie contrôle de RT-LOTOS vers les réseaux temporels (ou RdPT)** : Nous montrons à travers des schémas de traduction comment un RdPT peut être automatiquement construit à partir d'un comportement RT-LOTOS *régulier*. Nous démontrons la consistance de ces schémas en prouvant formellement que la traduction préserve la sémantique RT-LOTOS. Cette preuve a été possible parce que nous avons soigneusement évité l'introduction de toute transition auxiliaire. Aussi, la formalisation de ces schémas confère à RT-LOTOS une sémantique en termes de RdPT. Pour finir, ces schémas de traduction ont été implémentés dans *rtl2tpn,* un prototype qui prend en entrée une description RT-LOTOS et génère en sortie un RdPT équivalent. Il est ainsi devenu possible d'intégrer l'environnement *Tina* à notre plateforme de vérification.

- **Proposition d'un cadre formel pour la composition des RdPT** : Pour la mise en œuvre de notre approche, nous avons été amené à compenser une limitation des RdPT en matière de composition de comportements. Il est de notoriété que les RdPT n'offrent en natif aucun mécanisme pour composer (ou décomposer) de grands réseaux à partir de (ou en petits) réseaux. Ce constat nous a amené à augmenter le modèle des RdPT pour supporter la notion de composant. Les composants RdPT sont un modèle intermédiaire introduit avec l'idée de structurer les RdPT lors de leur génération à partir de descriptions RT-LOTOS. Notons que cette notion de composant reste assez générale pour être reprise dans d'autres contextes. Une contribution clé demeure la façon dont nous avons solutionné le problème de synchronisation temporelle dans les RdPT. Dans notre proposition de cadre pour la composition de RdPT, la synchronisation entre composants est obtenue sans manipulation explicite de l'information temporelle. Les composants RdPT gardent ainsi leur indépendance temporelle.

- **Extension de RT-LOTOS avec un mécanisme de *suspension/reprise*** : Rares sont les techniques de description formelle disposant d'un mécanisme de suspension avec sauvegarde du contexte temporel en vue d'une reprise. Ce constat vaut en particulier pour l'algèbre de processus RT-LOTOS et a motivé la proposition formulée dans cette

thèse d'étendre RT-LOTOS avec un opérateur *suspend/resume*. Notre proposition ne se limite pas aux aspects langage. En effet, nous proposons une traduction vers une classe simple de réseaux de Petri à chronomètres, qui permet via la méthode des classes d'états, la représentation exacte du comportement spécifié en RT-LOTOS étendu.

- **Traitement des données :** Nous avons étendu notre approche pour y inclure une partie *données*. Ce travail à nécessité la formalisation d'un nouveau modèle de réseaux de Petri temporels à *prédicats* et *actions*, supporté par une nouvelle version de *Tina* sous la forme de *Systèmes de Transitions Étiquetés* (composés de deux parties : la partie *contrôle* définie par un RdPT et la partie *données* définie par un ensemble de fonctions C). Nous avons par la même étendu le pouvoir de modélisation des composants introduits dans cette thèse en y ajoutant des variables, des actions, des offres et des ε-*transitions*.

- **Transposition de l'approche à la vérification de modèles UML temps-réel** : En dépit des qualités indéniables des langages formels de type RT-LOTOS, leur pénétration dans l'industrie reste limitée. L'un des obstacles demeure un apprentissage nécessitant un investissement certain. Les méthodes semi-formelles ont eu une fortune plus heureuse, favorisées en cela par des notations graphiques attrayantes et une sémantique moins contraignante. Dans un effort d'ouverture vers les standards métier en vigueur dans l'industrie, nous avons montré comment les travaux sur la traduction de RT-LOTOS vers les RdPT et RdPTPA ainsi que l'utilisation de l'environnement *Tina* qui en découle, peuvent être transposés et réutilisés à des fins de vérification formelle de modèles de systèmes temps-réel exprimés dans le profil UML TURTLE.

1.5 Organisation du mémoire

Le découpage de ce mémoire est à la fois logique et temporel. La chronologie des contributions s'est faite au rythme de l'évolution et des nouveautés intégrées dans environnement *Tina*.

Dans le chapitre 2, nous introduisons les deux modèles temporels qui font l'objet de cette thèse : RT-LOTOS et les réseaux de Petri temporels. Nous rappelons ainsi leurs sémantiques et principes de fonctionnement respectifs.

Le chapitre 3 expose la démarche que nous avons retenue pour traduire la partie contrôle de RT-LOTOS. Ainsi, pour chaque opérateur RT-LOTOS nous présentons un schéma de traduction vers les RdPT. La traduction est présentée de manière progressive par le biais de définitions formelles et illustrée par des exemples simples. Ce chapitre traite aussi du problème de la structuration des RdPT et propose une approche à base de composants. Nous y présentons des éléments de la preuve de consistance de la traduction. Pour clôturer ce chapitre et afin d'asseoir notre approche, nous présentons des études de cas étayées de comparaisons aux travaux du domaine.

Dans le chapitre 4 nous montrons que RT-LOTOS n'est pas adapté pour la description des systèmes préemptifs. Nous présentons une extension de RT-LOTOS avec un opérateur autorisant la suspension temporaire et la reprise ultérieure avec restauration du contexte. Nous montrons aussi comment le comportement d'une description RT-LOTOS étendue peut être construit, après traduction vers un réseau de Petri à chronomètres. Nous illustrons notre approche sur divers exemples en particulier un exemple illustrant l'intérêt de l'approche pour

des problèmes d'analyse d'ordonnançabilité.

Le chapitre 5 illustre l'extension de l'approche présentée dans le chapitre 2 par une partie *données*, et présente un nouveau modèle de réseaux de Petri temporels à *prédicats* et *actions* supporté par l'environnement *Tina*.

Le chapitre 6 présente les travaux sur la transposition de l'approche à des fins de vérification de modèles UML temps réel exprimés dans le profil TURTLE.

L'Annexe A illustre la preuve formelle de la consistance de la traduction des termes RT-LOTOS réguliers vers les RdPT 1-bornés.

L'Annexe B est consacrée au rappel de concepts et définitions des réseaux de Petri.

2 Modèles temporels considérés dans la thèse

Ce chapitre introduit les différents modèles temporels considérés dans cette thèse.

2.1 Systèmes de transitions étiquetées

Le comportement d'un système temporisé peut être représenté par un système de transitions étiquetées (ou $\mathcal{STE}$). Ce dernier représente l'ensemble des états possibles du système ainsi que les transitions entre ces mêmes états. Chaque transition est étiquetée par un label que nous appellerons l . Selon le label que l'on associe à une transition, cette dernière représente une occurrence d'action ou une progression du temps.

Définition 2.1 Un système de transitions étiquetées (ou $\mathcal{STE}$) est un tuple $\mathcal{LTS} = \langle Q, \rightarrow, Q_I \rangle$ avec :
Q un ensemble d'états,
$\rightarrow \subseteq Q \times Lab \times Q$ est une relation de transition entre états et,
$Q_I \subseteq Q$ est l'ensemble des états initiaux.
Nous notons $q \xrightarrow{l} q'$ si $\langle q, l, q' \rangle \in \rightarrow$.
De nombreuses relations d'équivalence existent dans la littérature. Ces relations diffèrent par leur pouvoir discriminateur et par les propriétés qu'elles permettent de vérifier. Ainsi, Milner a défini dans [62] les concepts de relations d'équivalence forte et de relations d'équivalence observationnelle basée sur la notion de bissimulation de Park [72].

Définition 2.2 Soient deux systèmes de transitions étiquetées $\mathcal{STE}_i = \langle Q_i, \rightarrow_i, q_i^0 \rangle$(pour $i = 1, 2$), nous définissons $R \subseteq Q_1 \times Q_2$ comme une bissimulation forte si $\forall (q_1, q_2) \in R$ et $l \in Lab$:

1. Si $\exists q_1^{'} \in Q_1$ avec $q_1 \xrightarrow{l} q_1^{'} \in \rightarrow_1$ alors $\exists q_2^{'} \in Q_2$ avec $q_2 \xrightarrow{l} q_2^{'} \in \rightarrow_2$ tel que $(q_1^{'}, q_2^{'}) \in R$.
2. Si $\exists q_2^{'} \in Q_2$avec $q_2 \xrightarrow{l} q_2^{'} \in \rightarrow_2$ alors $\exists q_1^{'} \in Q_1$ avec $q_1 \xrightarrow{l} q_1^{'} \in \rightarrow_1$ tel que $(q_1^{'}, q_2^{'}) \in R$.

Nous notons $q_1 \sim q_2$ s'il existe une bissimulation forte $R \subseteq Q_1 \times Q_2$ tel que $(q_1, q_2) \in R$.
Nous notons $\mathcal{STE}_1 \sim \mathcal{STE}_2$ si $q_1^0 \sim q_2^0$.

Un système de transitions temporisé (ou $\mathcal{STT}$) [44] est un système de transitions étiquetées où $Lab = Act \cup D$. L'ensemble Act représente les actions et D le domaine temporel. Ce dernier est restreint aux nombres positifs mais peut être discret ou dense. Enfin, notons qu'un $\mathcal{STE}$ peut être utilisé de différentes manières pour représenter un système temporisé. Par exemple, un étiquetage dans $Lab = Act \times D$ peut être utilisé pour représenter des actions temporelles.
Il existe dans la littérature plusieurs formulations de relation d'équivalence prenant en compte les informations temporelles. Dans la majeure partie de ces travaux, l'écoulement

du temps est traité comme l'occurrence d'une action. Ainsi, deux comportements bisimilaires doivent avoir exactement le même comportement temporel.

Les $\mathcal{STT}$ ne sont généralement pas considérés comme des modèles de spécification. Ils sont utilisés pour définir la sémantique d'autres modèles temporels.

2.2 Le langage RT-LOTOS

Le langage RT-LOTOS (Real-Time LOTOS) [28] est une algèbre de processus temporisée construite sur la technique de description formelle LOTOS [50] normalisée à l'ISO (cf. [57] pour une revue détaillée des principales extensions temporelles de LOTOS). RT-LOTOS propose essentiellement trois opérateurs pour décrire, de manière intuitive, l'expression du temps dans le comportement de processus LOTOS.

- L'opérateur de *délai* (noté ***delay***(d)) qui permet de retarder un processus d'une certaine quantité de temps d.
- L'opérateur de *Latence* (noté ***latency***(l)) qui permet de retarder un processus d'une certaine quantité de temps choisie de manière non-déterministe au sein de l'intervalle de latence [0,l]. Notons que cet opérateur porte sur la ou les premières actions du processus auquel il est appliqué. Par ailleurs, il n'a d'effet que s'il porte sur une action interne, l'occurrence d'une action observable étant, en tout état de cause, soumise à la date de l'offre faite par l'environnement.
- L'opérateur de restriction temporelle (noté $g\{t\}$) limite le temps pendant lequel une action observable g peut être offerte à son environnement. Le délai d'expiration commence à courir à partir du moment où l'action est offerte.

Considérons par exemple le processus $Q = delay(d)latency(l)g\{t\}; P$ qui combine ces trois opérateurs. La Figure 2.1 représente sur une ligne temporelle les contraintes spécifiées : dans l'intervalle [0,d], aucune action n'est offerte ; à une date choisie dans l'intervalle [d,d+l] l'action g est offerte ; cette dernière est offerte jusqu'à la date $d+t$, après laquelle elle n'est plus offerte à son environnement.

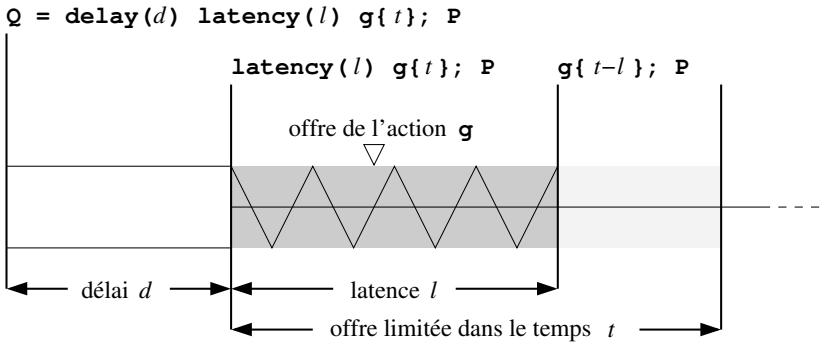


FIG. 2.1: Exemple de contraintes temporelles spécifiées avec RT-LOTOS

Présentation informelle des opérateurs de *Basic* RT-LOTOS

Nous appelons *Basic* RT-LOTOS le sous-ensemble de RT-LOTOS où les processus interagissent entre eux par synchronisation *pure* (sans échange de valeurs). Dans *Basic* RT-LOTOS, les actions sont identiques aux portes de synchronisation de processus.

Nous présentons d'abord les opérateurs RT-LOTOS qui ne font pas intervenir le temps (cf. Figure 2.2). Chaque processus P[a,b,c,d] est composé de deux processus P1[a,b] et P2[c,d] mis en relation avec différents opérateurs LOTOS et offrant chacun une synchronisation sur les portes a,b,c ou d à leur environnement :

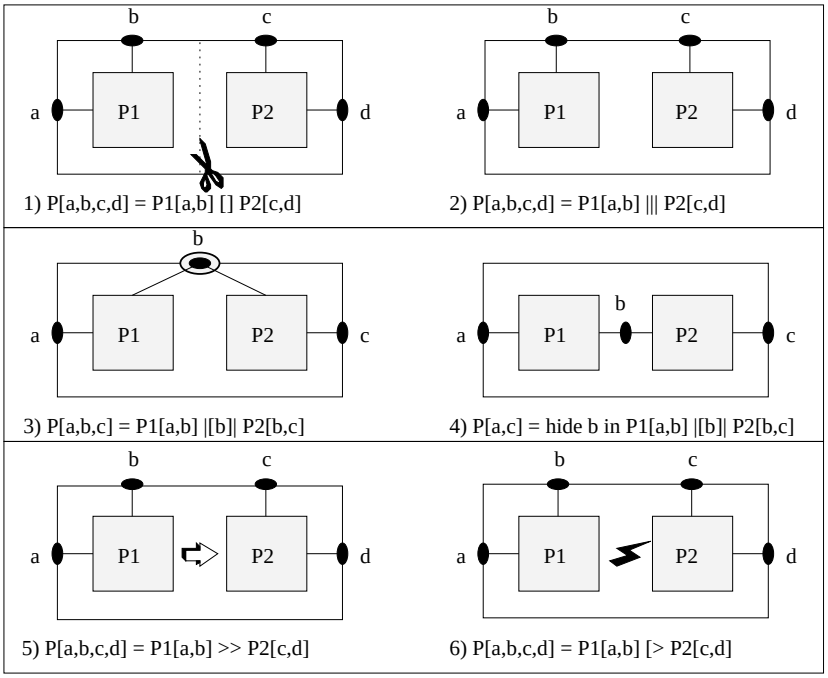


FIG. 2.2: Illustration des opérateurs de *Basic* LOTOS

1. **Choix** :P1[a,b] ou P2[c,d] suivant l'action qui se produira en premier.
2. **Parallélisme** : P1[a,b] et P2[c,d] s'exécutent en parallèle sans synchronisation.
3. **Synchronisation** : P1[a,b] et P2[c,d] s'exécutent en parallèle avec synchronisation sur la porte b.
4. **Intériorisation** : P1[a,b] et P2[c,d] s'exécutent comme dans (3) sauf que l'action b n'est plus disponible pour une synchronisation potentielle avec les processus appartenant à l'environnement du processus P.
5. **Séquence** : P1[a,b] s'exécute d'abord, à la terminaison de ce dernier P2[c,d] sera exécuté.
6. ***Préemption*** : à tout moment de l'exécution de P1[a,b], celui-ci peut être interrompu par P2[c,d].

Dans ce qui suit, les opérateurs temporels introduits par RT-LOTOS sont présentés de manière intuitive au moyen des graphiques décrits dans la Figure 2.3, où les axes temporels sont supposés avoir pour origine l'instant où le processus associé devient sensibilisé. Ces schémas illustrent les mécanismes offerts par les opérateurs temporels pour décrire les dates d'*offre* et d'occurrence des actions *internes* (cachées à l'environnement du processus par l'opérateur *hide*) et observables (offertes à l'environnement et par la même dépendantes de lui).

Les différents intervalles temporels y sont représentés comme suit :

- Une boite rectangulaire représente l'intervalle de délai, c'est-à-dire l'intervalle de temps pendant lequel il n'y a ni offre ni occurrence d'action ;
- Une zone grisée représente l'intervalle de temps dans lequel a lieu l'occurrence de l'action ;
- Un ressort représente l'intervalle de latence ; ie. l'intervalle de temps dans lequel est choisi une date pour l'offre de l'action ; cette zone est également grisée (gris foncé) car l'occurrence de l'action peut éventuellement y avoir lieu ;
- Un trait vertical épais représente l'instant à partir duquel l'action devient urgente (cas des schémas à gauche) ;
- Un triangle blanc marque un instant possible pour l'offre de l'action ;
- Un triangle noir marque un instant possible pour l'occurrence de l'action ; si le triangle noir est seul sur la ligne temporelle, alors l'offre et l'occurrence ont lieu au même instant.

1. La contrainte temporelle la plus simple consiste à retarder l'occurrence d'une action (observable ou interne) par une certaine quantité de temps. Dans le processus *P1*, l'occurrence possible de l'action a est retardée d'une durée d. En supposant que $\theta_{P1}(a)$ dénote la date à laquelle l'action a peut se produire, on a $\theta_{P1}(a) \in [d, \infty[$. En effet a est une action observable non urgente, l'occurrence effective de a dépend de la bonne volonté de son environnement.
2. Dans le processus *P2*, l'action a a été intériorisée et se comporte donc comme une action interne. Cette action est urgente puisqu'elle ne dépend plus de la disponibilité de son environnement. Donc, $\theta_{P2}(i_a) = d$, où i_a dénote l'action interne qui résulte de l'intériorisation de l'action a.
3. Dans le processus *P3*, une contrainte temporelle supplémentaire est associée à l'occurrence de a, qui dès qu'elle est sensibilisée, peut être offerte à son environnement par le processus *P3* pour une période limitée t. Nous avons donc $\theta_{P3}(a) \in [d, d+t]$. Si, pour une quelconque raison, l'action a ne peut se produire pendant cet intervalle de temps, une violation temporelle se produit conduisant à la transformation du processus *P3* en *stop*.
4. Dans le processus *P4*, l'action a a été intériorisée et par conséquent $\theta_{P4}(a) = d$ est d'une certaine manière similaire à la situation décrite pour le processus *P2*.
5. Dans le processus P5, la situation présente la synchronisation de deux processus. Si l'action a peut se produire, elle se produira à un temps appartenant à l'intersection des intervalles temporels caractérisant les contraintes temporelles associées à chaque processus de P5, soit P5_1 et P5_2. D'où $\theta_{P5}(a) \in [\theta_{P5-1}(x) + d1, \theta_{P5-1}(x) + d1 + t1] \cap [\theta_{P5-2}(y) + d2, \theta_{P5-2}(y) + d2 + t2]$. En cas d'intervalles temporels disjoints, une violation temporelle se produit (l'action a ne peut se produire) pour laquelle l'environnement n'est pas responsable. Comme mentionné précédemment, même si les intervalles temporels ne sont pas disjoints, une violation temporelle peut quand même se produire, si l'environnement n'accepte pas l'occurrence de l'action.
6. Dans le processus P6, l'action a a été intériorisée et par conséquent $\theta_{P6}(i_a) = max(\theta_{P5-1}(x) + d1, \theta_{P5-2}(y) + d2)$ si l'intersection des intervalles temporels est non vide.
7. Le processus P7 introduit l'opérateur de latence. Dans P7, l'action a est dans un premier temps retardée d'un temps d, ensuite éventuellement offerte à son environnement pendant une durée l puis, dans le cas où l'action a ne peut se produire, elle est continuellement offerte à son environnement. Comme pour le processus P1, on a $\theta_{P7}(a) \in [d, \infty[$, car il n'est pas possible de distinguer la raison pour laquelle une action observable ne

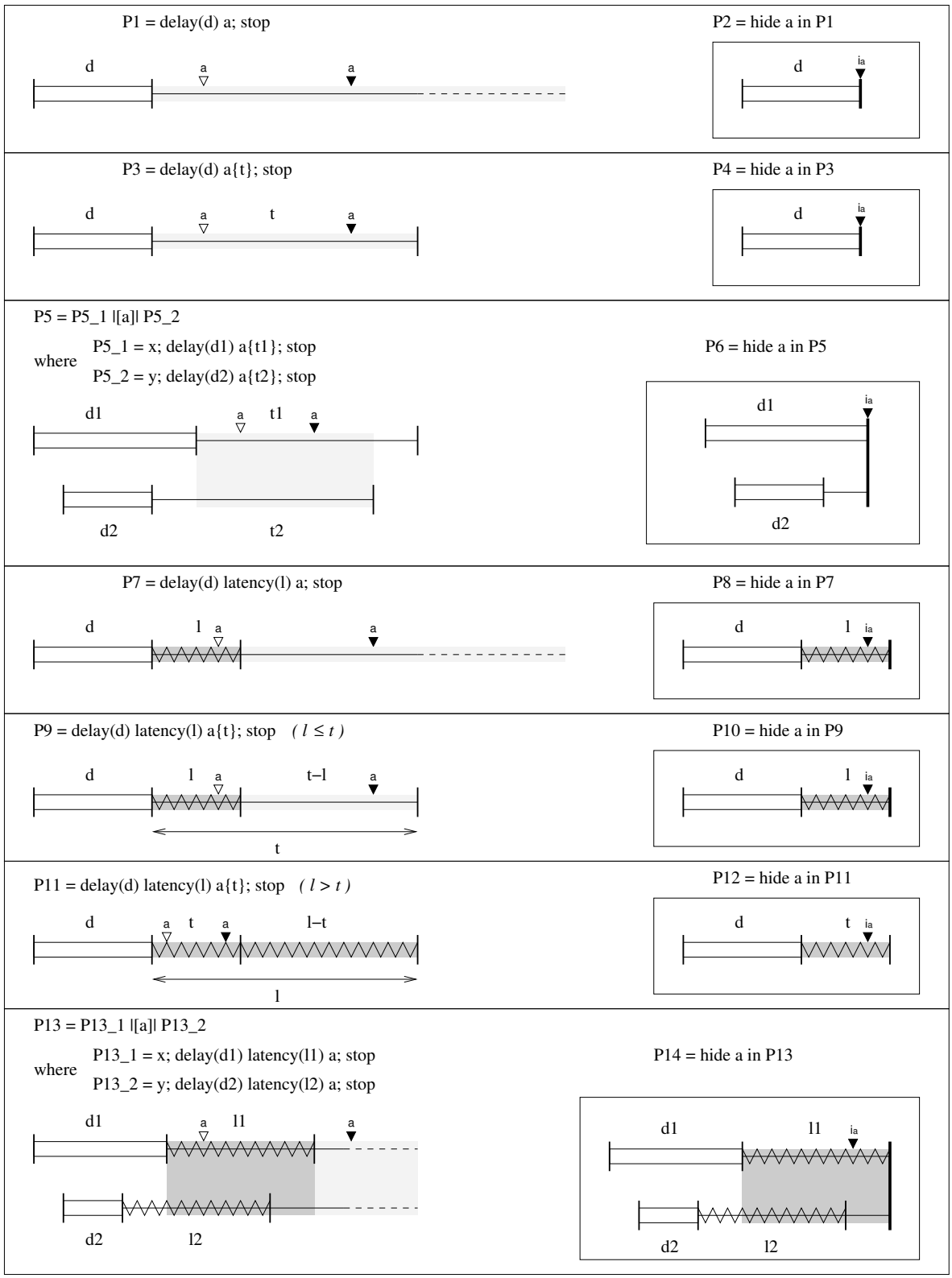


FIG. 2.3: Illustration des opérateurs temporels de RT-LOTOS

s'est pas produite pendant l'intervalle de latence $[d, d+l]$ (cela pourrait être dû au processus lui-même qui n'a pas offert l'action, ou à l'environnement qui n'a pas accepté l'action).

8. En intériorisant l'action a dans le processus P7, on obtient le processus P8. Puisqu'il y a un non-déterminisme temporel dans l'offre de l'action a pendant l'intervalle de latence $[d, d+l]$, et comme ce non-déterminisme temporel ne dépend pas exclusivement de l'environnement, on a $\theta_{P8}(i_a) \in [d, d+l]$. Ceci illustre bien, le fait que l'opérateur de latence a essentiellement pour objectif de relâcher au sein d'un intervalle temporel la condition d'urgence d'une action interne.
9. Le processus *P9* décrit la combinaison des opérateurs de latence et de délai avec une offre de l'action a limitée pour une période t, où $l \leq t$. D'où $\theta_{P9}(a) \in [d, d+t]$ ce qui est identique à la situation présentée dans le processus P3.
10. En intériorisant l'action a dans le processus P9 on obtient le processus P10 où $\theta_{P10}(i_a) \in [d, d+l]$, exactement comme pour le processus P8 (ceci est une conséquence de l'hypothèse $l \leq t$).
11. Le processus P11 est similaire au processus P9 sauf que l'on suppose maintenant que $l > t$. Donc $\theta_{P11}(a) \in [d, d+l]$.
12. En intériorisant l'action a dans le processus P11 on obtient le processus P12 où $\theta_{P12}(i_a) \in [d, d+t[$. Notons que cet intervalle de temps est ouvert à droite, ce qui implique que l'action interne i_a peut ne pas se produire.
13. Finalement, les processus P13 et P14 présentent des situations plus complexes pour lesquelles on pourra vérifier que : $\theta_{P13}(a) \geq max(\theta_{P13.1}(x) + d1, \theta_{P13.2}(y) + d2)$ et $\theta_{P14}(i_a) \in [tmin, tmax]$ (avec $tmin = max(\theta_{P13.1}(x) + d1, \theta_{P13.2}(y) + d2)$ et $tmax = max(\theta_{P13.1}(x) + d1 + l1, \theta_{P13.2}(y) + d2 + l2)$).

Syntaxe formelle de Basic RT-LOTOS

Soit $\mathcal{P}$ l'ensemble des identifiants de processus et soit $X \in \mathcal{P}$.

Soit $\mathcal{G}$ l'ensemble des portes définissables.

La syntaxe formelle de *Basic* RT-LOTOS est donnée par la Figure 2.4 ; la syntaxe de la déclaration d'un processus étant : ***process*** $X[g_1, ...g_n] := P$ ***endproc***

Des syntaxes alternatives, à savoir, $\triangle^u$, Ω^v et $\triangle^u\Omega^{v-u}$ ont été introduites avec respectivement pour signification *delay(u)*, *latency(v)* et *delay(u,v)*.

$$
\begin{array}{rcl}
P & ::= & \texttt{stop} \mid \texttt{exit} \mid X \texttt{ [} L \texttt{] } \mid \texttt{i ; } P \mid g \texttt{ ; } P \\
& \mid & P \texttt{ [] } P \mid P \texttt{ |[} L \texttt{]| } P \mid \texttt{hide } L \texttt{ in } P \\
& \mid & P \texttt{ >> } P \mid P \texttt{ [> } P \\
& \mid & \Delta^u\ P \mid \Omega^u\ P \\
& \mid & \texttt{i \{ } u \texttt{ \} ; } P \mid g \texttt{ \{ } u \texttt{ \} ; } P
\end{array}
$$

FIG. 2.4: Syntaxe de Basic RT-LOTOS

La sémantique opérationnelle de RT-LOTOS est présentée dans la Figure 2.5, elle est donnée dans le style SOS de Plotkin [74].

La présentation de la partie *données* de RT-LOTOS est quant à elle différée au chapitre 5 de cette thèse.

(1.a)	$exit \xrightarrow{\delta_s} stop$	(1.b)	$stop \xrightarrow{t} stop$
		(1.c)	$exit \xrightarrow{t} exit$
(2.a)	$g\{u\};P \xrightarrow{g_s} P\ (g \in \mathcal{G},\ u > 0)$	(2.b)	$g\{u+t\};P \xrightarrow{t} g\{u\};P\ (g \in \mathcal{G})$
		(2.c)	$g\{0\};P \xrightarrow{t} stop\ (g \in \mathcal{G})$
(3.a)	$i\{v\};P \xrightarrow{i_w} P\ (v > 0)$	(3.c)	$i\{u+t\};P \xrightarrow{t} i\{u\};P$
(3.b)	$i\{0\};P \xrightarrow{i_s} P$		
(4.a)	$\dfrac{P \xrightarrow{g} P'}{P[]Q \xrightarrow{g} P'}\ (g \in \mathcal{G}^{i,\delta})$	(4.b)	$\dfrac{P \xrightarrow{t} P' \quad Q \xrightarrow{t} Q'}{P[]Q \xrightarrow{t} P'[]Q'}$
(5.a)	$\dfrac{P \xrightarrow{g_x} P' \quad Q \xrightarrow{g_y} Q'}{P\|[L]\|Q \xrightarrow{g_{x\wedge y}} P'\|[L]\|Q'}\ (g \in L \cup \{\delta\})$	(5.c)	$\dfrac{P \xrightarrow{t} P' \quad Q \xrightarrow{t} Q'}{P\|[L]\|Q \xrightarrow{t} P'\|[L]\|Q'}$
(5.b)	$\dfrac{P \xrightarrow{g} P'}{P\|[L]\|Q \xrightarrow{g} P'\|[L]\|Q}\ (g \in \mathcal{G}^{i,\delta} \backslash L)$		
(6.a)	$\dfrac{P \xrightarrow{g} P'\ (g \in \mathcal{G}^{i,\delta} \backslash L)}{hide\ L\ in\ P \xrightarrow{g} hide\ L\ in\ P'}$	(6.c)	$\dfrac{P \xrightarrow{t} P' \quad P \overset{g_s}{\not\longrightarrow}\ (\forall g \in L)}{hide\ L\ in\ P \xrightarrow{t} hide\ L\ in\ P'}$
(6.b)	$\dfrac{P \xrightarrow{g_x} P'\ (g \in L)}{hide\ L\ in\ P \xrightarrow{i_x} hide\ L\ in\ P'}$		
(7.a)	$\dfrac{P \xrightarrow{g} P'}{P >> Q \xrightarrow{g} P' >> Q}\ (g \in \mathcal{G}^{i})$	(7.c)	$\dfrac{P \xrightarrow{t} P' \quad P \overset{\delta_s}{\not\longrightarrow}}{P >> Q \xrightarrow{t} P' >> Q}$
(7.b)	$\dfrac{P \xrightarrow{\delta_x} P'}{P >> Q \xrightarrow{i_x} Q}$		
(8.a)	$\dfrac{P \xrightarrow{g} P'}{P[> Q \xrightarrow{g} P'[> Q}\ (g \in \mathcal{G}^{i})$	(8.d)	$\dfrac{P \xrightarrow{t} P' \quad Q \xrightarrow{t} Q'}{P[> Q \xrightarrow{t} P'[> Q'}$
(8.b)	$\dfrac{Q \xrightarrow{g} Q'}{P[> Q \xrightarrow{g} Q'}\ (g \in \mathcal{G}^{i,\delta})$		
(8.c)	$\dfrac{P \xrightarrow{\delta} P'}{P[> Q \xrightarrow{\delta} P'}$		
(9.a)	$\dfrac{P_X[g_1/g'_1 \cdots g_n/g'_n] \xrightarrow{h} Q' \quad X[g'_1 \cdots g'_n] := P_X}{X[g_1 \cdots g_n] \xrightarrow{h} Q'}\ (h \in \{g_s, g_w \mid g \in \mathcal{G}^{i,\delta}\} \cup D)$		
(9.b)	$\dfrac{P \xrightarrow{g} P'}{P\phi \xrightarrow{\phi(g)} P'\phi}\ \begin{matrix}(\phi = [g_1/g'_1 \cdots g_n/g'_n]) \\ (g \in \mathcal{G}^{i,\delta})\end{matrix}$	(9.c)	$\dfrac{P \xrightarrow{t} P'}{P\phi \xrightarrow{t} P'\phi}\ (\phi = [g_1/g'_1 \cdots g_n/g'_n])$
(10.a)	$\dfrac{P \xrightarrow{g} P'}{\Delta^0 P \xrightarrow{g} P'}\ (g \in \mathcal{G}^{i,\delta})$	(10.b)	$\Delta^{u+t}P \xrightarrow{t} \Delta^u P$
		(10.c)	$\dfrac{P \xrightarrow{t} P'}{\Delta^0 P \xrightarrow{t} P'}$
(11.a)	$\dfrac{P \xrightarrow{g} P'}{\Omega^u P \xrightarrow{g_w} P'}\ (g \in \mathcal{G}^{\delta})$	(11.d)	$\dfrac{P \xrightarrow{t} P'}{\Omega^{u+t} P \xrightarrow{t} \Omega^u P'}$
(11.b)	$\dfrac{P \xrightarrow{i} P'}{\Omega^u P \xrightarrow{i} P'}$	(11.e)	$\dfrac{P \xrightarrow{t} P'}{\Omega^0 P \xrightarrow{t} P'}$
(11.c)	$\dfrac{P \xrightarrow{g} P'}{\Omega^0 P \xrightarrow{g} P'}\ (g \in \mathcal{G}^{i,\delta})$		

FIG. 2.5: Le modèle Basic RT-LOTOS

2.2.1 Comparaison avec E-LOTOS

Les derniers travaux de normalisation à l'ISO ont abouti à la norme E-LOTOS (pour Enhancements to LOTOS) [51]. Ce modèle étend la norme LOTOS [50]. E-LOTOS apporte des facilités en termes de manipulation de données, de structuration en fonctions, d'utilisation d'exceptions. Du point de vue temporel, E-LOTOS apporte un opérateur de délai déterministe "***wait***(d) ;P" et un opérateur de préfixe d'action :"$g(D)@T[SP]$;P" spécifie le processus qui propose une synchronisation sur la porte g lorsque SP est vrai, échange les valeurs D, enregistre dans une variable (par exemple $T='?t'$) le temps séparant l'offre et l'occurrence de l'action, ou l'impose (par exemple $T='!5'$), puis se comporte comme le processus P. L'urgence est rattachée à la notion d'action cachée. E-LOTOS permet également l'urgence sur les actions observables à conditions qu'elles soient définies comme *exception*.

La combinaison de ces fonctionnalités permet de spécifier un intervalle temporel pour offrir une action (par exemple "$g@?t\ [1<t<4]$"). Il est également possible d'introduire du non-déterminisme temporel en combinant l'opérateur d'assignation non-déterministe de variable et l'opérateur de délai déterministe mais employé sur cette même variable (par exemple *"var t : time in ?t := any time [1<t<4] ; wait(t) endvar ; P"*). Le non déterminisme temporel de E-LOTOS s'inscrit dans une démarche orientée *données* dans laquelle un temps non-déterministe est vu comme n'importe quelle donnée non-déterministe. Il se démarque, sur ce point, du non-déterministe temporel de RT-LOTOS par exemple, qui s'inscrit dans une démarche orientée *contrôle,* où la variable temporelle non-déterministe est une variable particulière introduite par un opérateur spécifique. Notons également que la combinaison de *l'offre limitée* dans le temps et le *délai non déterministe* diffère entre E-LOTOS et RT-LOTOS : dans l'expression RT-LOTOS "***latency***$(l)g\{t\}$" l'intervalle de latence et l'intervalle d'offre limitée dans le temps débutent au même instant, ce qui permet d'exprimer des violations temporelles potentielles dans le cas où $t < l$. Par contre, dans une construction E-LOTOS similaire, la contrainte d'offre d'une action dans un intervalle temporel n'entrera en vigueur qu'après l'expiration du délai non-déterministe, limitant ainsi la spécification de violation temporelle potentielle.

2.2.2 Validation de spécifications RT-LOTOS avec l'outil *rtl*

RT-LOTOS est supporté depuis huit ans par un outil de validation formelle. Appelé *rtl* : RT-LOTOS Laboratory [80], cet outil a été utilisé pour valider non seulement des systèmes directement modélisés en RT-LOTOS mais aussi des descriptions RT-LOTOS dérivées de modèles exprimés soit dans le langage SMIL [26], soit dans le profil UML temps réel TURTLE (Timed UML and RT-LOTOS Environnement) [6].

L'outil *rtl* prend en entrée une description RT-LOTOS et propose deux approches de validation : la *simulation* et la construction de l'espace des états du système qui nous intéressera plus particulièrement dans cette thèse. Cette construction de l'espace des états d'une description RT-LOTOS passe par la compilation de cette dernière en un DTA (Dynamic Time Automaton) [27]. Ce dernier est un automate temporisé étiqueté, qui opère une distinction entre actions urgentes et non urgentes. Le DTA sert de point de départ à la génération d'un graphe de régions préservant les propriétés de branchement. Dans ce dernier, un noeud (ou classe) définit à la fois un état de contrôle et une région représentée par un polyèdre convexe dont la dimension est égale au nombre d'horloges de l'état de contrôle. Un arc correspond soit à l'occurrence d'une action RT-LOTOS, soit à une progression du temps (arc étiqueté par t).

2.3 Réseaux de Petri temporels

Les réseaux de Petri temporels (ou RdPT) sont un formalisme éprouvé pour la spécification et la vérification des systèmes contraints par le temps.

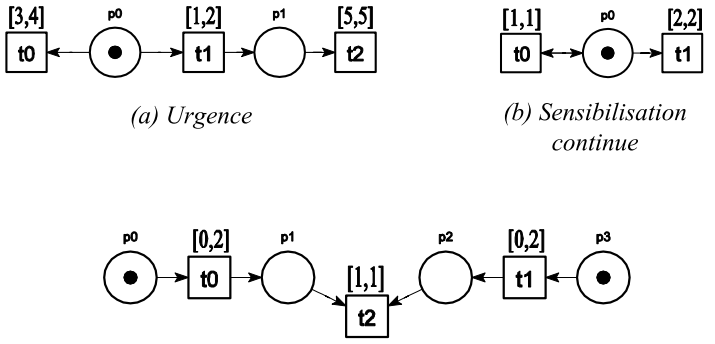
Les RdPT sont une extension des réseaux de Petri qui associe deux dates min et max à chaque transition. Si l'on suppose par exemple que la transition t soit devenue sensibilisée pour la dernière fois à la date θ, alors t ne peut être tirée avant la date $\theta + min$ et doit l'être au plus tard à la date $\theta + max$, à moins que t ne soit désensibilisée par le tir d'une autre transition. Le tir des transitions est de durée nulle.

De même que RT-LOTOS, les RdPT expriment naturellement des spécifications "*en délais*"[1].

Le problème de bornitude est indécidable [52] et les travaux sur ce modèle garantissent la décidabilité (comme l'accessibilité) moyennant l'hypothèse de bornitude du réseau. Rappelons qu'un réseau est borné si tout marquage accessible admet une borne supérieure.

Définition 2.3 Soit I^+ l'ensemble non vide des intervalles réels avec bornes rationnelles non négatives. Pour $i \in I^+, \downarrow i$ représente sa borne inférieure, et $\uparrow i$ sa borne supérieure (si elle existe) ou ∞ (sinon).

Un réseau de Petri temporel (RdPT) est un *tuple* $\langle P, T, Pre, Post, m_0, I_s \rangle$, tel que $\langle P, T, Pre, Post, m_0 \rangle$ est un réseau de Petri, et $I_s : T \rightarrow \mathbb{I}^+$est une fonction appelée intervalle statique. L'application I_s associe un intervalle temporel $I_s(t)$ à chaque transition du réseau. Les rationnels $Eft_s(t) = \downarrow i$ et $Lft_s(t) = \uparrow I_s(t)$ sont appelés date statique de tir au plus tôt de t, et date statique de tir au plus tard de t, respectivement. La Figure 2.6 illustre différentes notions d'importance quand il s'agit de modéliser des systèmes avec des RdPT.



(a) Urgence

(b) Sensibilisation continue

(c) Synchronisation

FIG. 2.6: Illustration des RdPT

Dans l'exemple du RdPT de la Figure 2.6(a), les transitions t_0 et t_1 sont sensibilisées par le marquage initial. Après une unité de temps, t_1 est tirable. Comme la borne supérieure de l'intervalle de t_1 est toujours atteinte avant que t_0 ne devienne tirable $(3 > 2)$, alors t_0 ne pourra jamais être tirée. La transition t_2 est quant à elle tirée 5 unités de temps après le tir de t_1. La Figure 2.6(c) décrit une synchronisation sur la transition t_2. t_0 (resp t_1) est tirée à une

[1] En explicitant débuts et fins d'actions, les RdPT peuvent aussi exprimer des spécifications "*en durées*".

date absolue $\theta_0 \leq 2$ (resp $\theta_1 \leq 2$), et t_2 est tirée à la date $max(\theta_0, \theta_1) + 1$. Pour finir, le RdPT de la Figure 2.6(b) illustre la propriété connue sous le nom de *sensibilisation continue*. Dans ce RdPT , la transition t_1 ne sera jamais tirée, a cause du fait qu'a chaque unité de temps, c'est la transition t_0 qui est tirée. Ce qui a pour effet d'enlever le jeton de la place p_0 et de le remettre immédiatement après. Par conséquent, t_1 ne sera jamais sensibilisée de manière continue pendant 2 unités de temps.

2.3.1 États et relation d'accessibilité

La définition qui suit est tirée de [17].

Définition 2.4 Un état d'un réseau temporel est un couple $e = (m, I)$ dans lequel m est un marquage et $I : T \longrightarrow I^+$une fonction qui associe un intervalle temporel à chaque transition sensibilisée par m.

L'état initial $e_0 = (m_0, I_0)$, où I_0 est une restriction de I_s aux transitions sensibilisées par le marquage initial m_0. Toute transition sensibilisée doit être tirée dans l'intervalle de temps qui lui est associé. Cet intervalle est relatif à la date de sensibilisation de la transition. Franchir t, à la date θ, depuis $e = (m, I)$, est donc permis ssi :

$m \geq Pre(t) \ \wedge \ \theta \in I(t) \ \wedge \ (\forall k \neq T)(m \geq Pre(k) \Rightarrow \theta \leq\uparrow I(k))$

l'état $e' = (m', I')$ atteint depuis e par le tir de t à θ est alors déterminé par :

1. $m' = m - Pre(t) + Post(t)$
2. pour chaque transition k sensibilisée par m' :

Si $K \neq t$ et $m - Pre(t) \geq Pre(k)$ alors $I'(k) = \{x - \theta | x \in I(k) \wedge x \geq \theta\}$ sinon $I'(k) = I_s(k)$.

On notera $\stackrel{t@\theta}{\longrightarrow}$ la relation d'accessibilité temporisée ainsi définie sur l'ensemble des états, et $e \stackrel{t}{\rightarrow} e'$ ssi $(\exists \theta)(e \stackrel{t@\theta}{\longrightarrow} e')$. Un *échéancier de tir* est une séquence de transitions σ associée à une séquence u de dates. L'échéancier (σ, u) est dit *réalisable* depuis l'état e si les transitions de la séquence σ sont successivement tirables depuis l'état e, aux dates relatives qui leur correspondent dans la séquence u. σ est le support de (σ, u) ; une séquence de transitions est tirable ssi elle est le support d'un échéancier réalisable. Le fonctionnement d'un réseau temporel est caractérisé par l'ensemble des états accessibles depuis son état initial, muni de la relation $\stackrel{t@\theta}{\longrightarrow}$.

2.3.2 Graphes de classes des réseaux temporels

Les états d'un réseau temporel admettent en général une infinité de successeurs. Ceci découle de la possibilité de tirer des transitions à n'importe quelle date dans leurs intervalles temporels. Ainsi une agglomération d'états en *classes d'états* [19][15] est nécessaire pour obtenir une représentation finie de l'espace d'états. Dans leur définition la plus générale, les espaces de classes d'états sont des recouvrements de l'ensemble des états munis d'une relation de transition $\stackrel{t}{\rightarrow}$ satisfaisant la propriété ci-dessus (c et c' désignent des ensembles d'états). Tous les états de chaque classe ont le même marquage.

$(\forall t \in T)(\forall c, c')(c \stackrel{t}{\rightarrow} c' \Leftrightarrow (\exists s \in c)(\exists s' \in c')(s \stackrel{t}{\rightarrow} s'))$

Les classes d'états permettent d'abstraire le temps du comportement d'un réseau temporel. Par conséquent, les transitions entre classes d'états ne portent plus d'information temporelle. Notons aussi que différents espaces de classes peuvent être définis, selon les propriétés de l'espace d'états qu'ils préservent.

2.3.3 Le problème de la synchronisation temporelle dans les réseaux de Petri temporels

Regardons de plus près l'exemple du RdPT de la Figure 2.6(c) . La transition t_2 possède deux places d'entrée. Il est important de comprendre que les comportements liés à ces deux places ne sont pas temporellement indépendants. Cette restriction provient du fait que la définition du tir des transitions dans les réseaux de Petri temporels :

- synchronise d'abord, (sans prendre en compte le temps), les comportements liés aux places d'entrée des transitions ;
- prend ensuite -et ensuite seulement- en considération le temps, par l'intervalle temporel associé à la transition (la contrainte [1,1] est ainsi commune à toutes les places d'entrée de la transition t_2).

Ainsi, dans les RdPT il n'y a pas de synchronisation temporelle *vraie* étant donné que les comportements modélisés ne sont pas temporellement indépendants. Ceci diffère de la synchronisation dans RT-LOTOS et les algèbres de processus temporisées en général. Rappelons que dans RT-LOTOS, les processus peuvent être synchronisés en tenant compte de leurs contraintes temporelles propres. Les réseaux de Petri temporels à flux indépendants [29] constituent une réponse à cette limitation des RdPT. Dans la section 3.7.1, nous présentons la solution que nous avons mis en œuvre pour contourner cette limitation des RdPT.

2.3.4 L'environnement *Tina*

Tina (TIme Petri Net Analyzer) [17] est un environnement logiciel développé par le LAAS-CNRS permettant l'édition et l'analyse de réseaux de Petri et réseaux temporels. Aux constructions classiques (graphe de marquages, arbre de couverture), *Tina* ajoute la construction d'espaces d'états abstraits, basés sur les techniques d'ordre partiel, préservant certaines classes de propriétés, comme l'absence de blocage, les propriétés de certaines logiques, ou les équivalences de test. Pour les RdPT, *Tina* propose différentes constructions de graphes de classes. Dans toutes ces constructions *Tina* permet de vérifier à la volée certaines propriétés *générales* telles que le caractère borné (dans certains cas), la présence de blocage ou encore la vivacité qui permet de s'assurer qu'une transition (ou une configuration du système) est atteignable à partir de tout état du comportement du système.

3 Traduction de la partie contrôle de RT-LOTOS vers les réseaux de Petri temporels

3.1 Introduction

La traduction d'un formalisme vers un autre est un sujet récurrent dans les revues et actes de conférences qui traitent de la vérification formelle de modèles. Si les transformations de modèles concernent essentiellement les formalismes et notations formelles (de telles traductions existent par exemple de SMV [59] vers PVS [71], de Murphi [49] vers PVS, de SMV vers Spin [47], des automates vers les réseaux de Petri et vice-versa). Ce type d'approches peut aussi s'appliquer aux langages de programmation (cf. l'outil Bandera [42] qui traduit du code java en SMV) ou aux notations dédiées au materiel (cf. les travaux sur les langages Verilog [84] ou VHDL [73]).

Dans un contexte prometteur pour les environnements "*open-source*", le projet *Topcased* (Toolkit in Open Source for Critical Application and Systems Development) [31] donne la part belle aux techniques de transformation de modèles. Facilitant ainsi la communication/ transformation entre les nombreux outils/langages de modélisation impliqués dans le développement d'un système critique.

La transformation de modèle pose un problème fondamental : celui de la préservation de propriétés. Un critère établi dans [53], évalue la qualité d'une traduction entre deux modèles à travers la capacité qu'a celle-ci à garantir une relation étroite entre les propriétés qui sont vraies dans le modèle d'origine et celles qui le sont aussi dans le modèle cible. La qualité d'une traduction est aussi liée à la modularité et la lisibilité du résultat obtenu dans le modèle cible. Ainsi, il est fortement souhaitable qu'une traduction maintienne la modularité du modèle d'origine et qu'elle n'induise pas une explosion du modèle cible. Identifier les différences entre modèles et minimiser leurs influences est crucial pour qu'une traduction soit efficace.

Les deux formalismes considérés dans cette thèse sont RT-LOTOS qui fait partie de la famille des algèbres de processus et les RdPT. Nombre de travaux dans la littérature traitent de la traduction d'algèbres de processus vers les RdP [41][83][66]. Le modèle Petri Box Calculus (PBC) [20] a été développé pour combiner algèbres de processus et RdP d'une manière consistante. Les auteurs prennent comme point de départ une algèbre de processus très proche de CCS. Les opérateurs de cette algèbre de processus sont définis en termes de RdP.

En ce qui concerne LOTOS, l'idée de réutiliser au profit de ce langage des techniques d'analyse éprouvées pour les RdP remonte à l'époque où LOTOS n'avait fait l'objet d'aucune extension temporelle rendue publique. Ainsi,[9][10][11][34][55] présentent différentes approche de traduction de LOTOS vers les réseaux de Petri. [82] traite de la traduction dans le sens contraire. [21] fut le premier article sur la traduction d'une extension temporelle de LOTOS vers les réseaux de Petri. [9] a proposé une sémantique en RdP pour un sous-ensemble de LOTOS restreint à une classe de comportements LOTOS qui ont une traduction vers les RdP finis. [10] a démontré qu'il est possible d'appliquer les méthodes de vérification développées

pour les RdP sans traduire explicitement les spécifications LOTOS en RdP. L'analyse est ainsi réalisée dans le monde LOTOS, la motivation des auteurs étant de contourner la complexité de la traduction et leur contribution se limitant à l'époque à une analyse d'accessibilité à la Karp et Miller.

Parmi toutes ces approches [34][38] est la seule opérant une traduction *complète* de LOTOS dans le sens où elle traite à la fois les aspects *contrôle* et *données*.

3.2 De RT-LOTOS vers les RdPT

Avant de présenter la traduction elle même, nous voudrions mettre l'accent sur une différence entre les algèbres de processus et les RdP quand il s'agit de faire le lien entre la structure et la sémantique d'un comportement donné.

Relation entre structure et comportement Les algèbres de processus et les RdP traitent la structure et la sémantique des systèmes concurrents de manières différentes. Pour illustrer ce constat intéressons nous à l'exemple ci-dessous. La Figure 3.1 montre trois réseaux N (un réseau de Petri non marqué N et deux réseaux de Petri marqués (N, M_0) et (N, M_1)). Chacun de ces réseaux décrit une séquence de deux actions a et b. Notons que le réseau sous-jacent est toujours le même dans ces trois cas, et que seuls les marquages diffèrent.

Intéressons nous maintenant à la question suivante :

Lequel des ces trois réseaux correspond au comportement RT-LOTOS suivant : *a ;b ; stop* ?

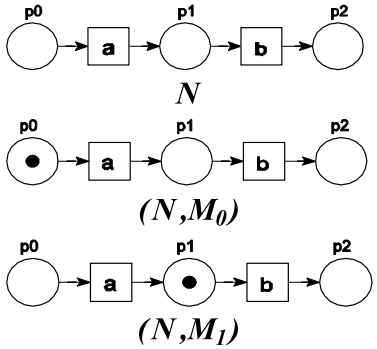


FIG. 3.1: Réseau de Petri avec différents marquages

La réponse va de soi, seul le réseau (N, M_0) peut exécuter l'action a suivie de l'action b. Dans le réseau (N, M_0), la transition a est sensibilisée par le marquage M_0 , elle est donc tirable. Après l'occurrence de a le marquage résultant est M_1 ($(N, M_0) \xrightarrow{a} (N, M_1)$).

Regardons maintenant ce qui se passe après l'exécution de l'action a dans les modèles RT-LOTOS et RdP. En RT-LOTOS l'expression *"a ; b ; stop"* se transforme en *"b ;stop"* ($a; b; stop \xrightarrow{a} b; stop$). Le réseau (N, M_0) se transforme quant à lui en (N, M_1). Cependant, seul le marquage change, alors que le réseau sous-jacent reste le même. Par contre, si nous voulions définir un réseau correspondant à l'expression *"b ;stop"* le choix du réseau (N, M_1) ne nous viendrait pas à l'esprit ; on opterait plutôt pour le réseau (N, M_1) expurgé des places p_0 et de la transition a.

Cet exemple montre bien qu'en matière de génération de comportement, RT-LOTOS et les algèbres de processus en général n'opèrent pas de la même manière que les RdP.

Dans RT-LOTOS, la structure de l'expression peut changer tandis que dans les RdP seul le marquage change alors que la structure reste la même pendant toutes les exécutions.

Ainsi dans les RdP, quand une action se produit, l'on pourrait toujours retrouver l'information de sa présence à partir de la structure statique du système, même si elle n'est plus exécutée dans le futur. Il existe donc une nette distinction entre les aspects statiques et dynamiques d'un système. A contrario, dans RT-LOTOS, une action passe aux oubliettes (à moins qu'elle ne fasse partie d'une boucle) après son occurrence, précisément parce qu'elle ne peut plus jamais être exécutée et que l'on peut donc changer -sans risque- la structure de l'expression du système. Il y a donc dans RT-LOTOS, une étroite relation entre les aspects statiques et dynamiques des processus.

Choix techniques pour la mise en œuvre de la traduction entre RT-LOTOS et les RdPT
Le processus de traduction entre RT-LOTOS et les RdPT peut être vu comme une opération qui vise à doter RT-LOTOS d'une sémantique à base de RdPT. Dans [66] l'auteur définit un critère d'*évaluation* de qualité pour toute sémantique à base de RdP. Dans notre contexte, la satisfaction de ce critère dit de *récupération* requiert qu'aucune transition *auxiliaire* ne soit introduite dans le graphe d'accessibilité du RdPT par rapport au graphe d'accessibilité du terme RT-LOTOS.

Un survol de la littérature a montré que peu de traductions entre algèbres de processus et RdP satisfont ce principe. Par exemple [41][83] ne le satisfont pas. Pour se plier à cette recommandation forte, nous définissons dans la suite de cette thèse une correspondance *une à une* entre les actions du terme RT-LOTOS et les transitions du RdPT correspondant. En s'interdisant de la sorte le recours à des transitions *auxiliaires*, nous garantissons de fait que la traduction ne génère aucun comportement additionnel. Le prix à payer est une complication, certes surmontable, des schémas de traduction entre RT-LOTOS et les RdPT. Ainsi, moyennant un effort supplémentaire dans la définition des schémas de traduction, nous nous assurons qu'une exécution du système spécifié peut être vue indifféremment comme une suite d'actions RT-LOTOS ou une séquence de tir d'un RdPT.

Une autre difficulté à surmonter réside dans le fait que les RdPT n'offrent en natif aucun mécanisme pour composer (ou décomposer) de grands réseaux à partir de (ou en) petits réseaux. Or, dans un tel processus de traduction, il est inévitable de considérer les RdPT comme des entités composables. Ceci nous a amené à augmenter le modèle des RdPT pour supporter la notion de *composant* avec l'idée de structurer les RdPT lors de leur construction à partir d'expressions RT-LOTOS. Ainsi, nous trouvons en la notion de *composant* RdPT un pendant à la notion de terme RT-LOTOS. Cette notion de *composant* fait partie de notre réponse à l'absence d'opérateurs de composition dans les RdPT.

3.3 Composant de réseaux de Petri temporels

Un *composant* est le bloc de construction de base de la procédure de traduction. Il encapsule un RdPT qui décrit son comportement. Par la même, il réalise une action en tirant une transition, et communique avec son environnement au travers de points d'interactions.

Nous utilisons des étiquettes pour représenter les actions d'un RdPT. Les transitions représentant différentes occurrences d'une même action portent la même étiquette. Un composant

possède deux ensemble d'étiquettes. *Act* qui désigne l'alphabet du composant et *Time* ={*tv, delay, latency*}[1] qui désigne les étiquettes introduites pour représenter le comportement temporel du composant. Ainsi, l'étiquette "*tv*" représente une violation temporelle dans une offre limitée dans le temps. Les étiquettes "*delay*" et "*latency*" représentent pour leur part un délai déterministe et un délai non déterministe.

La Figure 3.2 illustre un exemple de composant C_P correspondant au terme RT-LOTOS P. Durant son exécution C_P peut tirer l'action observable a qui est de ce fait rattachée à un point d'interaction (le rectangle noirci placé sur le périmètre du composant). Ses interfaces d'entrée et de sortie sont représentées respectivement par l'ensemble des places in_i (places initialement marquées) et la place *out* (représentée en gris). Quand la place de sortie d'un composant est marquée, ceci signifie que celui ci a terminé son exécution avec succès. Un composant est dit *actif* si au moins une de ses transitions est sensibilisée, sinon il est dit *inactif.*

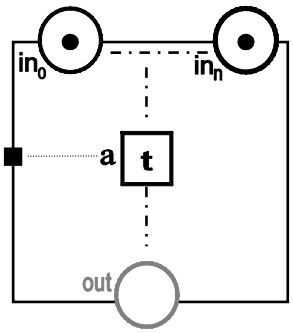


FIG. 3.2: Exemple de composant

Définition 3.1 (Composant RdPT) Soit $Act = A_O \cup A_h \cup \{exit\}$ un alphabet d'actions, où A_O est un ensemble d'actions observables (avec $i \notin A_O$, $exit \notin A_O$), $A_h = \{i\} \times A_O$ est l'ensemble des actions internes (i_a dénote l'action interne a).

Un composant RdPT est un tuple $C = \langle \sum, lab, I, O \rangle$ où :

- $\sum = \langle P, T, Pre, Post, M_0, IS \rangle$ est un RdPT.
- $Lab : T \rightarrow (Act \cup Time)$ est une fonction d'étiquetage qui associe à chaque transition de $\sum$ avec un label d'action (Act) ou un label d'évènement temporel dans $Time$ ($Time = \{tv, delay, latency\}$). Soit T^{Act}(resp. T^{Time}) l'ensemble des transitions avec des labels dans Act (resp. $Time$).
- $I \subset P$ est un ensemble non vide de places définissant l'interface d'entrée.
- $O \subset P$ est l'interface de sortie du composant. Un composant possède une interface de sortie s'il a au moins une transition étiquetée par $exit$. Auquel cas , O est la place de sortie de ces transitions. Sinon, $O = \emptyset$. $Card(O) \leq 1$.

De plus, les invariants suivants sont associés aux composants :

- **H1** Il n'y a pas de transition source dans un composant.
- **H2** Le RdPT encapsulé est 1-borné.
- **H3** Si toutes les places d'entrée sont marquées, alors toutes les autres places sont vides ($I \subset M \Rightarrow M = I$).

[1] Dans la suite le terme d'*évènement temporel* est utilisé pour désigner toute occurrence d'une transition dont le label apparitent à l'ensemble *Time= {tv, delay , latency}.*

- **H4** Si la place *Out* est marquée, alors toutes les autres places sont vides ($O \neq \emptyset \wedge O \subset M \Rightarrow M = O$).
- **H5** Pour chaque transition t tel que $Lab(t) \in Act$, si son label est celui d'une action observable ($Lab(t) \in A_O$), alors son intervalle temporel est $[0, \infty)$, sinon ($lab(t) \in A_h \cup \{exit\}$) c'est $[0, 0]$.

L'hypothèse H2 appelée propriété des *marquages saufs*, est essentielle pour la décidabilité de l'analyse d'accessibilité appliquée aux RdPT. Pour se conformer à cette propriété les spécifications RT-LOTOS devraient satisfaire le critère dit de *contrôle statique*. Dans [34] il est établi qu'une spécification LOTOS satisfait la propriété de contrôle statique si cette dernière ne contient pas de récursion à travers les opérateurs : $|[]|$, $[>$ et $\gg$. De plus [4] démontre qu'une spécification LOTOS qui satisfait le critère de contrôle statique est régulière si elle ne contient aucune *valeur* ou *sort*. Les opérateurs temporels introduits par RT-LOTOS n'ont pas d'effet sur la propriété de contrôle statique. Ces conditions sont donc suffisantes pour établir qu'un processus RT-LOTOS est régulier. Si c'est le cas, ce dernier peut donc être représenté par des RdPT finis.

H3 et H4 sont appelées propriétés de *marquages propres* dans [20].

3.4 Patterns de traduction

Notre intention est d'utiliser le graphe de classes d'états d'un RdPT pour représenter et analyser le comportement d'une spécification RT-LOTOS. Pour traiter ce passage de RT-LOTOS aux RdPT nous avons défini un ensemble de schémas de traduction (patterns). Ce choix vise à pallier le fait que ni les opérateurs de composition, ni les opérateurs temporels de RT-LOTOS n'ont de contrepartie directe en RdPT. Ainsi, les RdPT encapsulés dans les composants peuvent être combinés pour exprimer un comportement sous forme d'un RdPT composite. Pour ce faire, nous associons à chaque opérateur RT-LOTOS une opération spécifique sur les composants. Ces opérations sont décrites graphiquement dans la section suivante. A cette définition graphique -*facile à comprendre*-, nous ajoutons une définition formelle, ainsi que des exemples illustratifs.

La consistance de la traduction entre RT-LOTOS et les RdPT a fait l'objet d'une preuve formelle, décrite en section 3.9 et donnée intégralement en Annexe A de ce mémoire. Une procédure de validation expérimentale sur la base de comparaison entre les espaces des états d'un terme RT-LOTOS et le RdPT correspondant a été aussi mise en œuvre. Dans ce qui suit nous explicitons les détails de cette procédure.

Validation expérimentale des patterns de traduction : Cette opération vise à comparer les deux espaces des états engendrés respectivement par *rtl* et *Tina*. Cette comparaison des deux graphes d'accessibilité apporte une forme de validation expérimentale des patterns de traduction. [2]

Plus précisément *rtl* produit un graphe minimal d'accessibilité (cf. section 2.2.2) du système, préservant les propriétés de branchement. *Tina* dispose d'une construction qui préserve les propriétés de branchement sous le nom de graphe de classes atomiques (cf. section 2.3.2).

[2] Chronologiquement cette validation expérimentale a été entreprise avant que la consistance de la traduction n'ait été formellement prouvée, et visait à donner une certaine confiance dans les patterns proposés.

Nous avons recensé des différences potentielles entre les deux graphes. Ces différences sont dues à une procédure de minimisation implémentée dans *rtl* et adaptée de [87]. Cette dernière permet de considérer des régions plus grandes qu'il n'est requis d'un point de vue d'accessibilité stricte. De plus, avec *rtl*, une *latence*, un *délai déterministe* ou l'expiration d'une *offre temporelle* sont considérés comme une progression du temps, qui pourrait se produire à l'intérieur d'une région (on parle de progression implicite du temps). Ce qui n'est pas le cas avec *Tina* où l'occurrence de l'une de ces trois transitions est considérée comme un événement spécifique amenant nécessairement à une autre classe d'états.

Comment alors comparer le graphe issu de *Tina* à celui construit par *rtl* ?

La solution que nous proposons est de remplacer les étiquettes *Time* par t qui est le symbole utilisé par *rtl* pour représenter la progression du temps dans ses graphes d'accessibilité. Si les deux graphes d'accessibilité ne sont pas identiques, alors nous les minimisons tous deux en utilisant *l'équivalence observationnelle* de Milner [62] en compactons les chaînes de transitions étiquetées par t.

La Figure 3.3 décrit la procédure de validation expérimentale que nous avons mis en œuvre.

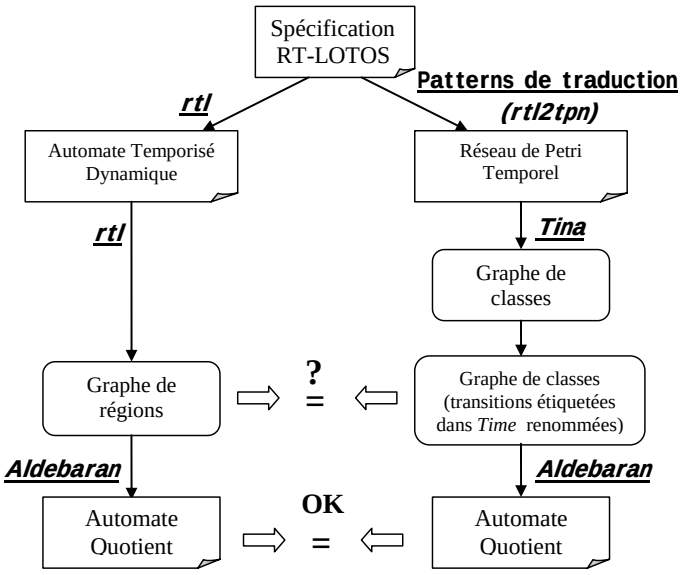


FIG. 3.3: Procédure de validation expérimentale des patterns

Les résultats de cette validation expérimentale sur des exemples illustratifs, vont accompagner la présentation des patterns. Le résultat est soit le graphe de classes du RdPT (s'il est confondu avec le graphe de région engendré par *rtl* pour le comportement RT-LOTOS correspondant), soit un automate quotient engendré par *Aldébaran* et qui est le même pour les deux espaces des états engendrés à la fois par *rtl* et par *Tina*.

3.4.1 Notations et définitions préliminaires

$f' = f \cup (a, b)$ définit la fonction $f' : A \cup \{a\} \longmapsto B \cup \{b\}$ tel que $f'(x) = f(x)$ si $x \in A$ et $f'(a) = b$ sinon.

Nommage des places et des transitions : La construction incrémentale d'un RdPT fait face au problème de l'unicité des noms : en substance l'opération d'ajout d'une nouvelle place/transition à un réseau devrait garantir l'unicité du nom de cette dernière. Afin de contourner ce problème, l'on pourrait mettre en exposant de la place/transition à rajouter le nom du terme RT-LOTOS correspondant. Prenons l'exemple du composant correspondant au comportement RT-LOTOS : $a;P$, et qui est obtenu en ajoutant une transition étiquetée par a à l'ensemble des transitions du composant C_P. Ainsi, au lieu d'écrire $T_{a;P} = \{t\} \cup T_P$, on écrirait $T_{a;P} = \{t^{a;P}\} \cup T_P$.

Opérations de bas niveau sur les RdPT

En vue d'une formalisation complète des patterns de traduction nous définissons les opérations suivantes sur les RdPT :

Définition 3.2 (Ajout d'une place ou d'une transition à un RdPT)
Soit $\sum = \langle P, T, Pre, Post, M_0, IS\rangle$ un RdPT.

Ajout d'une place : Soit p une nouvelle place ($p \notin P$), Pre_P et $Post_P$ deux ensembles de transitions dans T.
$\sum' = \sum \cup \langle Pre_P, p, Post_p\rangle$ définit le RdPT augmenté de la place p tel que ${}^\bullet p = Pre_p$ et $p^\bullet = Post_p$.

$$\sum^{'} = \left\langle P \cup \{p\}, T, Pre \cup \bigcup_{t \in Pre_p} (p,t), Post \cup \bigcup_{t \in Post_p} (t,p), M_0, IS \right\rangle$$

Ajout d'une transition : Soit t une nouvelle transition ($t \notin T$), I son intervalle temporel, Pre_t et $Post_t$ deux ensembles de places dans P. $\sum' = \sum \cup \langle Pre_t, (t,I), Post_t\rangle$ définit le RdPT augmenté de la transition t tel que ${}^\bullet t = Pre_t$ et $t^\bullet = Post_t$.

$$\sum^{'} = \left\langle P, T \cup \{t\}, Pre \cup \bigcup_{p \in Pre_t} (p,t), Post \cup \bigcup_{p \in Post_t} (t,p), M_0, IS \cup (t,I) \right\rangle$$

Définition 3.3 (Suppression d'un ensemble de places ou de transitions dans un RdPT)

Soit $\sum = \langle P, T, Pre, Post, M_0, IS\rangle$un RdPT.

Suppression d'un ensemble de places : Soit P' un ensemble de places ($P' \subset P$). Alors $\sum' = \sum \setminus P'$ est le RdPT où l'ensemble de places P' a été supprimé.

$$\sum^{'} = \langle P \setminus P', T, \{(p,t) \in Pre \mid p \notin P'\}, \{(t,p) \in Post \mid p \notin P'\}, M_0 \setminus P', IS\rangle$$

Suppression d'un ensemble de transitions : Soit T' un ensemble de transitions ($T' \subset T$). Alors $\sum' = \sum \setminus T'$ est le RdPT où l'ensemble de transitions T' a été supprimé.

$$\sum' = \langle P, T \setminus T', \{(p,t) \in Pre \mid t \notin T'\}, \{(t,p) \in Post \mid t \notin T'\}, M_0, \{(t,I) \in IS \mid t \notin T'\}\rangle$$

Définition 3.4 (Union de deux RdPT) Soient $\sum = \langle P, T, Pre, Post, M_0, IS\rangle$ et $\sum' = \langle P', T', Pre', Post', M_0', IS'\rangle$ deux RdPT.

L'union des deux RdPT, notée $\sum \biguplus \sum'$ est définie par :

$$\sum \biguplus \sum' = \langle P \uplus P', T \uplus T', Pre \uplus Pre', Post \uplus Post', M_0 \uplus M_0', IS \uplus IS'\rangle$$

Où $\uplus$ est l'opérateur de l'union disjointe d'ensembles, i.e. union avec renommage ad-hoc.

Ensembles de premières/dernières actions : Plusieurs comportements RT-LOTOS tels que ceux offrant une préemption ou un choix entre deux processus sont étroitement liés à l'occurrence de la première action dans l'un des deux processus. Pour définir formellement le composant correspondant à de tels comportements, nous définissons les ensembles suivants :

Définition 3.5 (Ensemble des premières actions)

Soit C_P un composant. L'ensemble des premières actions $\mathcal{FA}(\mathrm{C_P})$ est construit à partir des règles suivantes :

$\mathcal{FA}(\mathrm{C_{stop}}) = \emptyset$ — $\mathcal{FA}(\mathrm{C_{exit}}) = \{\mathrm{t_{exit}}\}$

$\mathcal{FA}(\mathrm{C_{a;P}}) = \{\mathrm{t_a}\}$ — $\mathcal{FA}(\mathrm{C_{delay(d)P}}) = \mathcal{FA}(\mathrm{C_P})$

$\mathcal{FA}(\mathrm{C_{a\{d\}P}}) = \{\mathrm{t_a}\}$ — $\mathcal{FA}(\mathrm{C_{P;Q}}) = \mathcal{FA}(\mathrm{C_P})$

$\mathcal{FA}(\mathrm{C_{latency(d)P}}) = \mathcal{FA}(\mathrm{C_P})$ — $\mathcal{FA}(\mathrm{C_{P \gg Q}}) = \mathcal{FA}(\mathrm{C_P})$

$\mathcal{FA}(\mathrm{C_{P|[A]|Q}}) = \mathcal{FA}(\mathrm{C_P}) \cup \mathcal{FA}(\mathrm{C_Q})$ — $\mathcal{FA}(C_{P[>Q}) = \mathcal{FA}(\mathrm{C_P}) \cup \mathcal{FA}(\mathrm{C_Q})$

$\mathcal{FA}(\mathrm{C_{P[]Q}}) = \mathcal{FA}(\mathrm{C_P}) \cup \mathcal{FA}(\mathrm{C_Q})$ — $\mathcal{FA}(\mathrm{C_{\mu X.(P;X)}}) = \mathcal{FA}(\mathrm{C_P})$

$\mathcal{FA}(\mathrm{C_{hide\ a\ in\ P}}) = \mathrm{h_a}(\mathcal{FA}(\mathrm{C_P}))$

Où $C_{\mu X.(P;X)}$ est le composant qui exécute les actions de P *ad infinitum*. $h_a(\alpha) = \alpha$ si $\alpha \neq a$ et $h_a(a) = i_a$.

Définition 3.6 (Ensemble des dernières actions)

Soit C_P un composant. L'ensemble des dernières actions $\mathcal{LA}(\mathrm{C_P})$ est construit à partir des règles suivantes :

$\mathcal{LA}(\mathrm{C_{stop}}) = \emptyset$ — $\mathcal{LA}(\mathrm{C_{exit}}) = \{\mathrm{t_{exit}}\}$

$\mathcal{LA}(\mathrm{C_{a;P}}) = \mathcal{LA}(\mathrm{C_P})$ — $\mathcal{LA}(\mathrm{C_{delay(d)P}}) = \mathcal{LA}(\mathrm{C_P})$

$\mathcal{LA}(\mathrm{C_{a\{d\}P}}) = \mathcal{LA}(\mathrm{C_P})$ — $\mathcal{LA}(\mathrm{C_{p;Q}}) = \mathcal{LA}(\mathrm{C_Q})$

$\mathcal{LA}(\mathrm{C_{latency(d)P}}) = \mathcal{LA}(\mathrm{C_P})$ — $\mathcal{LA}(\mathrm{C_{p \gg Q}}) = \mathcal{LA}(\mathrm{C_Q})$

$\mathcal{LA}(\mathrm{C_{P|[A]|Q}}) = \mathcal{LA}(\mathrm{C_P}) \cup \mathcal{LA}(\mathrm{C_Q})$ — $\mathcal{LA}(C_{p[>Q}) = \mathcal{LA}(\mathrm{C_P}) \cup \mathcal{FA}(\mathrm{C_Q})$

$\mathcal{LA}(\mathrm{C_{P[]Q}}) = \mathcal{LA}(\mathrm{C_P}) \cup \mathcal{LA}(\mathrm{C_Q})$ — $\mathcal{LA}(\mathrm{C_{\mu X.(P;X)}}) = \mathcal{LA}(\mathrm{C_P})$

$\mathcal{LA}(\mathrm{C_{hide\ a\ in\ P}}) = \mathrm{h_a}(\mathcal{LA}(\mathrm{C_P}))$

Où $C_{\mu X.(P;X)}$ est le composant qui exécute les actions de P *ad infinitum*. $h_a(\alpha) = \alpha$ si $\alpha \neq a$ et $h_a(a) = i_a$.

3.5 Composant de base

Le composant C_{stop} est représenté par un réseau vide (pas de places ni de transitions).

C_{exit} est le composant qui réalise une terminaison avec succès. Il contient une place d'entrée, une place de sortie et une seule transition étiquetée par *exit*[3] à laquelle est associé l'intervalle statique $[0, 0]$ (cf . Figure 3.4).

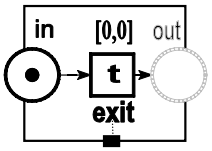


FIG. 3.4: Pattern du *exit*

3.6 Patterns applicables à un seul composant

Considérons le composant C_P de la Figure 3.2. Dans ce qui suit nous décrivons les différents patterns pouvant s'appliquer au composant C_P.

3.6.1 Pattern du préfixage par une action

Description informelle $C_{a;P}$ (cf. Figure 3.5) est le composant résultant du préfixage de C_P avec l'action a.

Ainsi, après l'exécution de l'action a par le composant $C_{a;P}$, le composant C_P est activé.

[3]Cette action *exit* correspond à l'action δ définie dans le modèle sémantique de LOTOS. La Figure 3.4 correspond à la règle SOS (1.a). Le comportement décrit par la règle SOS (1.c) est obtenu dans le cas d'une composition parallèle de composants.

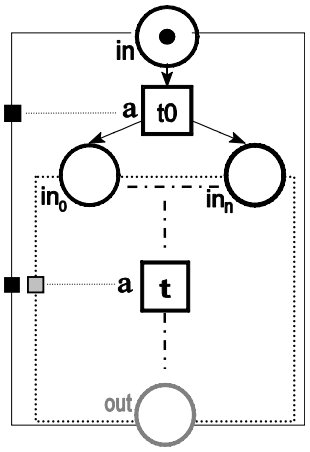


FIG. 3.5: Pattern du préfixage par une action

Définition formelle

$$C_{a;P} = \left\langle \sum_{a;P}, Lab_{a;P}, \{in\}, O_P \right\rangle$$

Le RdPT $\sum_{a;P}$ est obtenu en ajoutant la place in et la transition t_0 à $\sum_P$. $Lab_{a;P}$ étiquette la transition t_0 par a.

$$\sum_{a;P} = (\sum_P \cup \langle \emptyset, (t_0, [0, \infty)), I_P \rangle) \cup \langle \emptyset, in, t_0 \rangle$$

$$Lab_{a;P} = Lab_P \cup (t_0, a)$$

3.6.2 Pattern de l'offre limitée dans le temps

Description informelle $C_{a\{d\};P}$ (cf. Figure 3.6) est le composant résultant du préfixage du composant C_P, par une *offre limitée* dans le temps de d unités de temps sur l'action a.

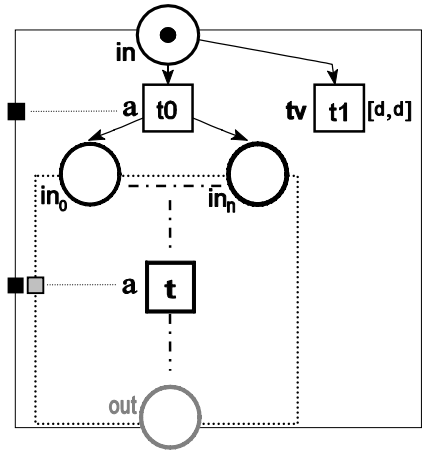


FIG. 3.6: Pattern de l'offre limitée dans le temps

Si pour une raison quelconque, l'occurrence de l'action a ne peut intervenir durant l'intervalle de temps $[0, d]$, la transition étiquetée par tv sera tirée et le composant $C_{a\{d\};P}$ deviendra *inactif* (i.e. qu'il n'y a aucune transition sensibilisée dans le composant $C_{a\{d\};P}$). Ce pattern est construit à partir du pattern précédent, en y introduisant une seule et unique nouvelle transition t_1. Par conséquent, nous utiliserons des éléments du composant $C_{a;P}$ dans la définition du composant $C_{a\{d\};P}$.

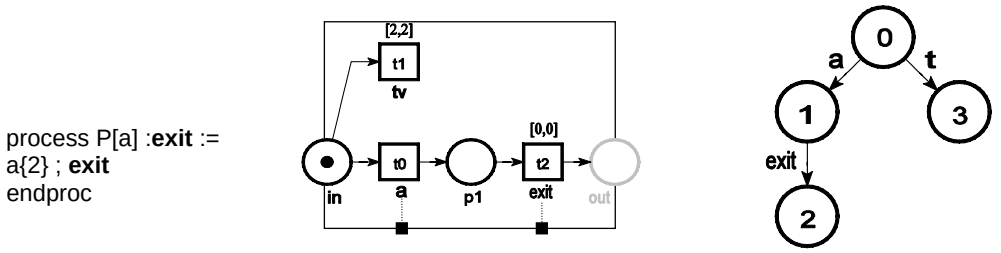
Définition formelle

$$C_{a\{d\};P} = \left\langle \sum_{a\{d\};P}, Lab_{a;P} \cup \{(t_1, tv)\}, \{in\}, O_p \right\rangle$$

$$\sum_{a\{d\};P} = \sum_{a;P} \cup \langle \{in\}, (t_1, [d, d]), \emptyset \rangle$$

Validation expérimentale Soit P un processus RT-LOTOS dont le comportement est décrit dans la Figure 3.7(a). Le processus P offre l'action a à son environnement pendant 2 unités de temps. Le composant correspondant (cf. Figure 3.7(b)) ainsi que le graphe de classes (cf. Figure 3.7(c)) obtenu selon la procédure[4] décrite dans la Section 3.4 suivent :

[4]Le graphe de régions engendré par *rtl* pour le processus P et le graphe de classes atomiques engendré par *Tina* pour le RdPT correspondant sont identiques après renommage de tv en t.



(a) Processus P (b) Composant C_P (c) Graphe de Classes

FIG. 3.7: Validation expérimentale du pattern de *l'offre limitée* dans le temps

3.6.3 Pattern du délai déterministe

Description informelle $C_{delay(d)P}$ (cf. Figure 3.8) est le composant résultant de l'application d'un *retard déterministe* de d unités de temps aux premières actions du composant C_p.

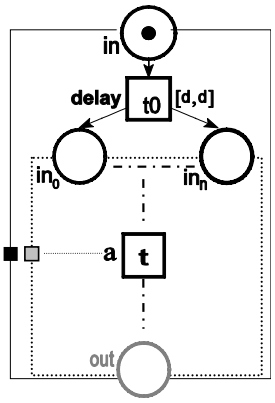


FIG. 3.8: Pattern du *délai déterministe*

Ce pattern est similaire à celui de $C_{a;P}$ à la différence que la transition rajoutée est étiquetée par *delay* et que l'intervalle statique de cette dernière est égal à $[d, d]$.

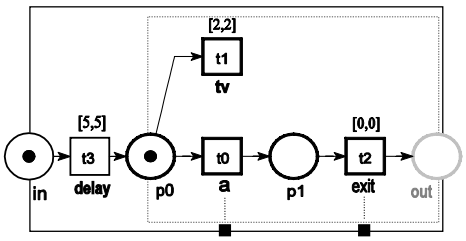
Définition formelle

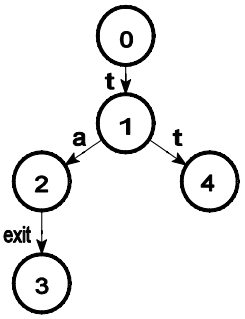
$$C_{delay\{d\};P} = \left\langle \sum_{delay\{d\};P}, Lab_P \cup \{(t_0, delay)\}, \{in\}, O_p \right\rangle$$

$$\sum_{delay\{d\};P} = (\sum_{P} \cup \langle \emptyset, (t_0, [d, d]), I_P \rangle) \cup \langle \emptyset, in, t_0 \rangle$$

Validation expérimentale Nous reprenons le processus P de la section précédente et nous lui appliquons maintenant un *délai déterministe* de 5 unités de temps.

process P_1[a] :**exit** :=
delay(5)a{2} ; **exit**
endproc





(a) Processus P (b) Composant C_P (c) Graphe de classes

FIG. 3.9: Validation expérimentale du pattern du délai déterministe

3.6.4 Pattern du délai non déterministe (*latence*)

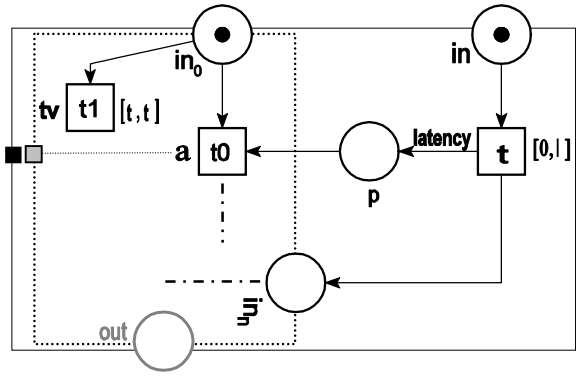


FIG. 3.10: Pattern de la latence

Description informelle $C_{latency(l)P}$ (cf. Figure 3.10) est le composant résultant de l'application d'un retard non déterministe de d unités de temps aux premières actions du composant C_p. Comme pour le pattern précédent, le composant $C_{latency(l)P}$ est construit en connectant une nouvelle transition à l'interface d'entrée de C_P, mais cette fois nous lui associons un intervalle statique égale à $[0, d]$.

La définition du composant $C_{latency(l)P}$ doit prendre en compte le cas subtil où une (ou plusieurs) des premières actions du composant C_P est (ou sont) contrainte(s) avec une offre limitée dans le temps. Par exemple, dans la Figure 3.10, l'action a est offerte à l'environnement de C_P durant t unités de temps. La sémantique RT-LOTOS stipule que dans pareil cas, la *latence* et l'offre dans le temps doivent démarrer en même temps, ce qui signifie que si la durée de la *latence* va au delà de t unités de temps, l'*offre temporelle* sur l'action a expirera. Pour obtenir le même comportement dans le RdPT correspondant, nous rajoutons la place in_0 (la place d'entrée de l'action a) dans l'interface d'entrée du composant $C_{latency(d)P}$.

Ainsi les transitions t_1 et t sont sensibilisées dès l'activation (toutes les places d'entrées marquées) du composant $C_{latency(d)P}$. Ce dernier a donc la possibilité d'exécuter a (tirer t_0) si t_0 est sensibilisée (i.e. tant que les places in_0 et p restent marquées) avant que t_1 ne soit tirée (à la date t). Par conséquent, l'action a n'est susceptible d'être offerte à l'environnement que pendant t unités de temps, ce qui est conforme à la sémantique de RT-LOTOS.

Définition formelle Soit $\mathcal{FA}_{\text{lo}}(C_P)$ l'ensemble des *premières* actions contraintes avec une offre limitée dans le temps.

$$\mathcal{FA}_{\text{lo}}(C_P) = \{t_a \in \mathcal{FA}(C_P) \,|\, tv \in ({}^\bullet t_a)^\bullet\}$$

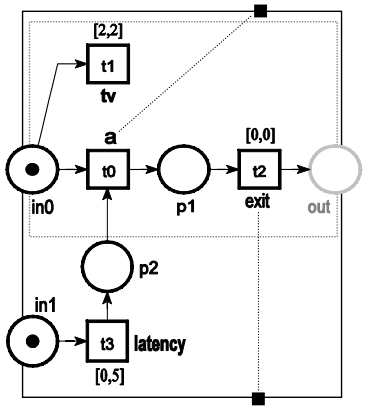
$$I_{lo} =^\bullet \mathcal{FA}_{lo}(C_P)$$

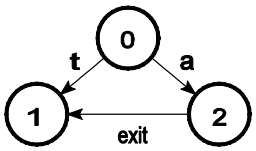
$$C_{latency(d)P} = \left\langle \sum_{latency(d)P}, Lab_a \cup \{t, latency)\}, I_P \cup \{in\}, O_P \right\rangle$$

$$\sum_{Latency(d)P} = \sum_P \cup \langle \emptyset, p, \mathcal{FA}(\mathrm{C_P}) \rangle \cup \langle \emptyset, (t, [0, d]), p \rangle \cup \langle \emptyset, in, t_0 \rangle$$

Validation expérimentale Appliquons maintenant au processus P un délai *non déterministe* de 5 unités de temps (cf. Figure 3.11(a)). La Figure 3.11(b) décrit le composant correspondant ainsi que l'automate quotient résultant (cf. Figure 3.11(c)).

process P_2[a] :**exit** :=
latency(5)a{2} ; **exit**
endproc





(a) Processus P (b) Composant C_P (c) Automate quotient

FIG. 3.11: Validation expérimentale du pattern de la latence

Commentons les résultats de cette validation expérimentale pour le pattern de *latence*. Cet exemple a nécessité le calcul de l'automate quotient pour établir l'équivalence des deux espaces des états engendrés par *rtl* et *Tina* respectivement. En effet, on peut observer dans le graphe de classes engendré par *Tina* (cf. Figure 3.12(b)) pour le réseau de la Figure 3.11(b) qu'à partir de la classe 0, l'expiration de latence mène vers une nouvelle classe 1. Ce qui n'est clairement pas le cas dans le graphe de région engendré par *rtl* (cf. Figure 3.12(a)). Ceci est dû au fait

que la *latence* est considérée dans ce cas précis comme une progression implicite du temps et qui peut donc se produire à l'intérieur de la classe 0.

(a) Graphe de régions (b) Graphe de classes

FIG. 3.12: Graphe de régions Vs graphe de classes

3.6.5 Pattern de l'intériorisation (*hiding*)

Description informelle $C_{hide\ a\ in\ P}$ (Figure 3.13) est le composant obtenu en intériorisant l'action a dans le composant C_P.

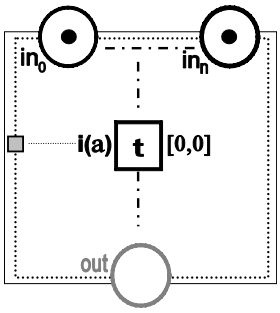


FIG. 3.13: Pattern du hide

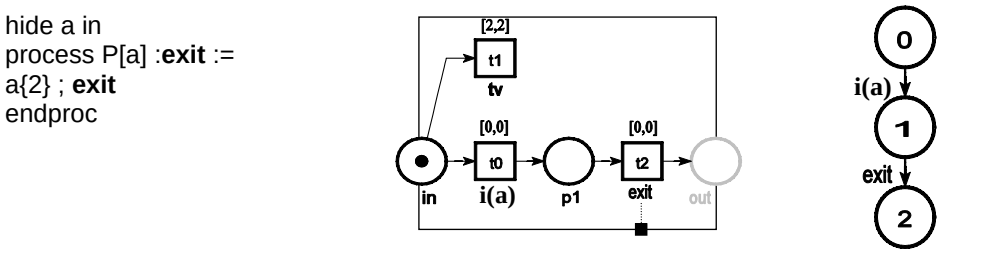
Cette opération vise à transformer des actions observables (externes) en actions internes. Rendant ainsi la synchronisation sur ces actions avec d'autres composants impossible. Dans RT-LOTOS, cacher une ou plusieurs actions induit une notion d'urgence dans l'occurrence de ces dernières. Pour obtenir un comportement similaire dans le RdPT correspondant, nous associons à toute transition représentant une action interne un intervalle statique de valeur $[0, 0]$. Une telle transition est donc tirable dès sa sensibilisation.

Définition formelle Soit la fonction $h_a(\alpha) = \alpha$ si $\alpha \neq a$ et $h_a(a) = i_a$.

$$C_{hide\,a\,in\,P} = \left\langle \sum_{hide\,a\,in\,P} , h_a \circ Lab_P, I_P, O_P \right\rangle$$

$$\sum_{hide\,a\,in\,P} = \langle P_P, T_P, Pre_P, Post_P, IS_{hide} \rangle\,, \quad IS_{hide}(t) = \begin{cases} [0,0]\ si\ Lab_P(t) = a \\ IS_P(t) \quad sinon \end{cases}$$

Validation expérimentale L'action a est intériorisée dans le processus P.



(a) Comportement RT-LOTOS (b) Composant $C_{hideainP}$ (c) graphe de classes

FIG. 3.14: Validation expérimentale du pattern du hide

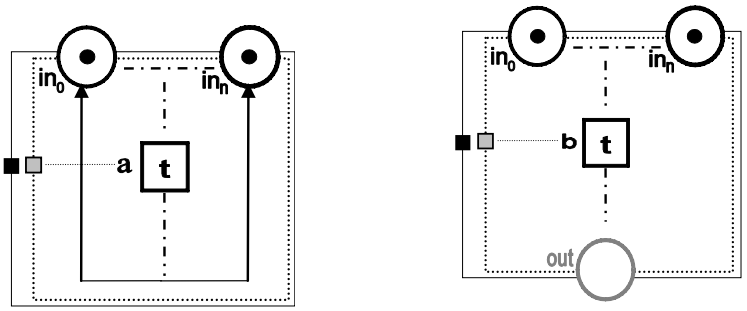
3.6.6 Patterns de la récursion et de l'instanciation

Description informelle

- $C_{\mu X.(P;X)}$ (cf. Figure 3.15(a)) est le composant qui exécute récursivement les actions du composant C_P. Nous modélisons la récursion par des RdPT cycliques. Pour ce faire nous connectons les dernières actions du composant C_P à l'interface d'entrée de ce dernier.

$$\forall t \in \mathcal{LA}(\mathrm{C_P}) : \mathrm{t}^\bullet = \mathrm{I_P}$$

  Les comportements récursifs peuvent être à l'origine d'une complexité supplémentaire ; cette problématique est au delà du cadre de cette thèse, qui traite essentiellement des aspects temporels. Le lecteur intéressé peut trouver dans [20] des détails sur le traitement du problème de la récursion dans un cadre combinant algèbre de processus et RdP.
- $C_{P[a/b]}$ (cf. Figure 3.15(b)) est le composant qui se comporte comme le composant C_P, mais où toutes les transitions étiquetées par a ont été renommées par b.



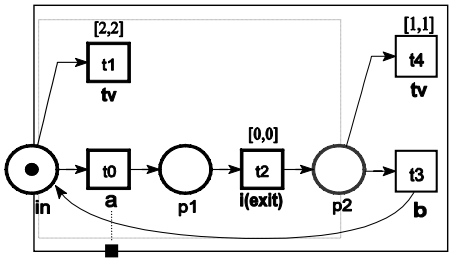
(a) Composant cyclique (b) Renommage de a par b

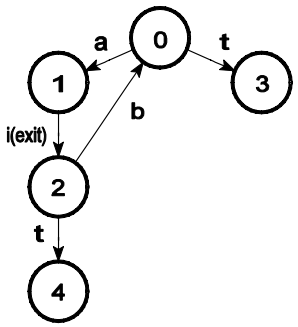
FIG. 3.15: Patterns de la récursion et de l'instanciation

Validation expérimentale Considérons le comportement RT-LOTOS de la Figure 3.16(a) qui exécute récursivement la séquence suivante : le processus P suivie d'une offre temporelle sur l'action b. (Notez l'usage de l'opérateur de composition séquentielle, cf. Section 3.7.2 pour les détails du traitement de cet opérateur).

.

```
process P3[a,b] :noexit :=
a{2} ; exit
>>
b{1}; P3[a,b]
endproc
```





(a) Comportement RT-LOTOS (b) Composant C_P (c) Graphe de classes

FIG. 3.16: Validation expérimentale du pattern de la récursion

3.7 Patterns s'appliquant à un ensemble de composants

Chacun des patterns suivants combine de différentes manières un ensemble de composants pour aboutir à un seul composant.

3.7.1 Pattern de la synchronisation parallèle

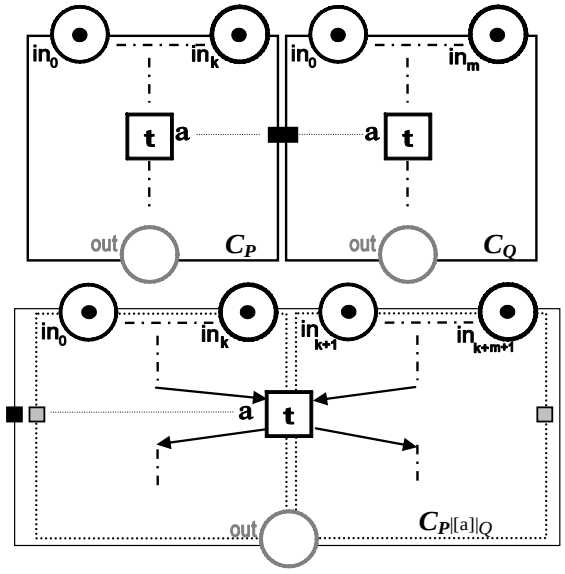


FIG. 3.17: Pattern de la synchronisation parallèle

Description informelle La communication par *rendez-vous* est un concept important des algèbres de processus. Elle consiste en une synchronisation symétrique par laquelle une action a partagée par n processus ne peut être exécutée que si tous ces derniers sont prêts (chacun de son coté) à exécuter cette action a. Dans la théorie des RdP, un tel scénario est représenté par une transition ayant n places d'entrées. Cette transition ne peut être tirée que si toutes ses places d'entrée sont marquées. L'opération de synchronisation est ainsi obtenue en opérant une fusion de transitions. Bien que cette opération de fusion de transitions soit simple dans les RdP (non temporels), cette dernière se trouve être un problème plutôt délicat dans les RdPT. En effet, une telle opération nécessite la manipulation explicite des intervalles temporels assignés aux transitions impliquées dans la synchronisation, et ceci afin d'exprimer la contrainte de temps globale sous forme de conjonctions d'intervalles. La possible incompatibilité de ces intervalles temporels, peut induire des inconsistances dans l'expression finale de la contrainte temporelle. Ce problème n'est d'ailleurs pas résolu dans [54] (cf. section 3.8 pour une comparaison détaillée avec [54]).

Pour éviter ce problème de composition d'intervalles, et préserver la compositionnalité de notre approche, nous évitons soigneusement d'assigner des intervalles temporels aux transitions représentants des actions. Au lieu de celà, nous associons l'information temporelle à des transitions dédiées (étiquetées par des labels dans *Time*). Ainsi la synchronisation peut être faite simplement sans nécessiter une prise en compte explicite des contraintes temporelles associées aux transitions. L'avantage de notre solution est double :

1. Permettre une construction incrémentale de la contrainte temporelle globale sur une action synchronisée. La Figure 3.17 illustre la synchronisation entre deux composants C_P et C_Q sur l'action a. Notre but est de définir un cadre compositionnel où, chaque

composant impliqué dans la synchronisation peut rajouter des contraintes temporelles sur l'occurrence de l'action a. De telle sorte que la contrainte temporelle globale sur l'action a dans le composant $C_{P|[a]|Q}$ soit la conjonction de plusieurs contraintes plus élémentaires. Cela implique que lorsque le composant C_P est prêt à l'exécution de l'action a, il est forcé dans l'absence d'une action alternative, à attendre que le composant C_Q puisse offrir l'action a. Ceci peut amener par exemple à l'expiration d'une *offre temporelle* sur a dans le composant C_P. Le comportement désiré est obtenu par une simple fusion des transitions étiquetées par a comme pour les RdP de base.

2. Le relâchement de la sémantique forte des RdPT : dans ces derniers, le tir des transitions est dirigé par nécessité. Ainsi, une action doit être exécutée dès que la valeur supérieure de son intervalle temporel est atteinte (sauf si elle est en conflit avec une autre transition). RT-LOTOS (et les algèbres de processus en général) favorise plutôt un point de vue basé sur l'interaction, où le comportement concret d'un système est influencé par son environnement. Ainsi, une transition sensibilisée peut être tirée dans la fenêtre temporelle qui lui est assignée mais ne peut être forcée à s'exécuter. Une large gamme de systèmes temps-réel fonctionnent selon ce principe. En particulier, les systèmes temps-réel souples sont des exemples typiques de systèmes qui ne peuvent être forcés à se synchroniser avec leur environnement. Un tel comportement serait impossible si les transitions représentant des actions étaient associées à des intervalles temporels. Pour modéliser la nécessité dans RT-LOTOS nous utilisons l'opérateur *hide*. La combinaison de ce dernier avec les opérateurs *offre limitée* et *latence* donne une flexibilité d'expression intéressante.

Au final, la synchronisation sur l'action a des composants C_P et C_Q est obtenue en fusionnant chaque transition étiquetée par a dans le composant C_P avec chaque transition étiquetée par a dans le composant C_Q, créant ainsi $n * m$ transitions étiquetées par a dans le composant $C_{P|[a]|Q}$(n et m représentent le nombre des transitions étiquetées par a dans les composants C_P et C_Q respectivement). Ainsi, les deux composants C_P et C_Q doivent s'attendre mutuellement pour tirer la transition a de manière synchrone.

Définition formelle Soit T_P^a l'ensemble des transitions étiquetées avec le label a dans le composant C_P. T_P^A est l'extension de cette définition aux ensembles de labels :

$$T_P^a = \{t \in T_P |\, Lab_P(t) = a\}$$

$$T_P^A = \bigcup_{a \in A} T_P^a$$

Le réseau $\sum_{P|[A]|Q}$ est obtenu en substituant toutes les transitions t_p dans le composant C_p et ayant un label a dans A par un ensemble de transitions (t_p, t_q) tel que t_q soit étiquetée par a dans le composant C_Q, avec ${}^\bullet(t_p, t_q) = {}^\bullet t_p \cup {}^\bullet t_q$ et $(t_p, t_q)^\bullet = t_p^\bullet \cup t_q^\bullet$. A cette transition est associé un intervalle statique $[0, \infty)$ (cf. H5).

La synchronisation sur la transition *exit* se fait de la manière suivante :

Si les deux composants C_P et C_Q possèdent une interface de sortie ($O_P \neq \emptyset \wedge O_Q \neq \emptyset$) , alors toutes les transitions étiquetées par *exit* sont fusionnées (même traitement que pour des actions observables), les deux places *out* des deux composants sont supprimées et remplacées par une seule place de sortie commune. Si au moins l'un des deux composants ne possède

pas de place de sortie, alors le composant $C_{P|[A]|Q}$ ne pourra pas terminer son exécution avec succès et nous ne lui associons donc pas d'interface de sortie. Les transitions étiquetées par *exit* (si elles sont présentes) seront supprimées.

Notons :

$$merge(t_p, t_q) = \langle {}^\bullet t_p \cup {}^\bullet t_q, ((t_p, t_q), IS(t_p)), t_p^\bullet \cup t_q^\bullet \rangle$$

$$A' = A \cup \{exit\}$$

$$O_{P|[A]|Q} = \begin{cases} \{out\} & si\ O_P \neq \phi \wedge O_Q \neq \phi \\ \phi & sinon \end{cases}$$

$$PreOut = \begin{cases} T_P^{exit} \times T_Q^{exit} & si\ O_P \neq \phi \wedge O_Q \neq \phi \\ \phi & sinon \end{cases}$$

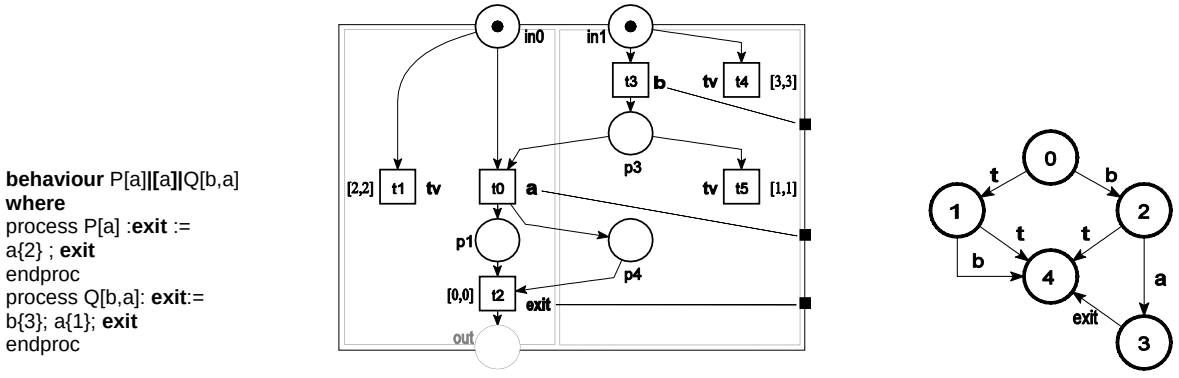
La définition de $C_{P|[A]|Q}$ suit :

$$C_{P|[A]|Q} = \left\langle \sum_{P|[A]|Q}, Lab_{P|[A]|Q}, I_P \cup I_Q, O_{P|[A]|Q} \right\rangle$$

$$\sum_{P|[A]|Q} = (\sum_P \backslash T_P^{A'} \setminus O_P) \uplus (\sum_Q \backslash T_Q^{A'} \setminus O_Q) \cup \bigcup_{t_p \in T_P^{A'}, t_q \in T_Q^{A'}} merge(t_p, t_q) \cup \langle PreOut, out, \phi \rangle$$

$$Lab_{P|[A]|Q}(t) = \begin{cases} Lab_P(t) & si\ t \in T_P \\ Lab_Q(t) & si\ t \in T_Q \\ a & si\ t = (t_p, t_q) \wedge t_p \in T_P^a \end{cases}$$

Validation Expérimentale Le processus P est maintenant synchronisé sur l'action a avec un nouveau processus Q. Ce dernier commence par offrir l'action b pendant 3 unités de temps ; ensuite il offre l'action a pendant une seule unité de temps.



(a) Comportement RT-LOTOS (b) Composant $C_{P|[a]|Q}$ (c) Automate quotient

FIG. 3.18: Validation expérimentale du pattern de la synchronisation

3.7.2 Pattern de la synchronisation séquentielle

Description informelle $C_{P \gg Q}$ (cf. Figure 3.19) est le composant résultant de la composition séquentielle des composants C_P et C_Q.

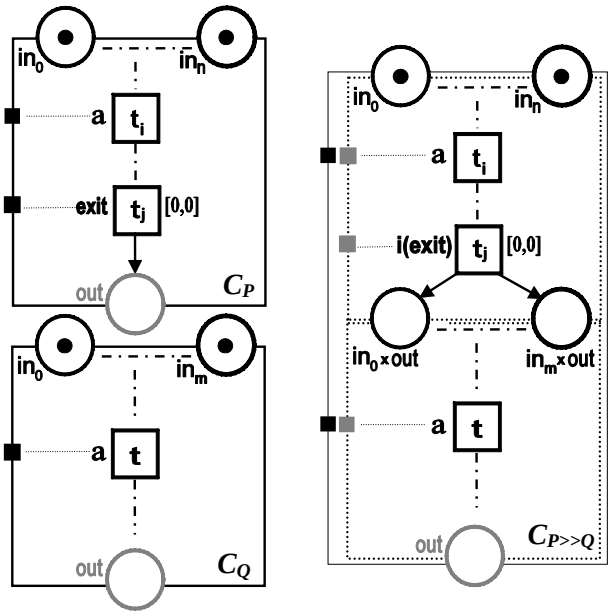


FIG. 3.19: Pattern de la composition séquentielle

Ce type de composition est possible seulement si le composant C_P possède une interface de sortie. Ainsi, si l'exécution du composant C_P se termine avec succès, le composant C_Q est alors activé. Le composant résultant $C_{P \gg Q}$ est obtenu par la fusion de l'interface de sortie du composant C_P et l'interface d'entrée du composant C_Q. De plus et pour se conformer à la sémantique de RT-LOTOS, le point d'interaction rattachée à la transition *exit* dans C_P est intériorisé dans le composant résultant $C_{P \gg Q}$.

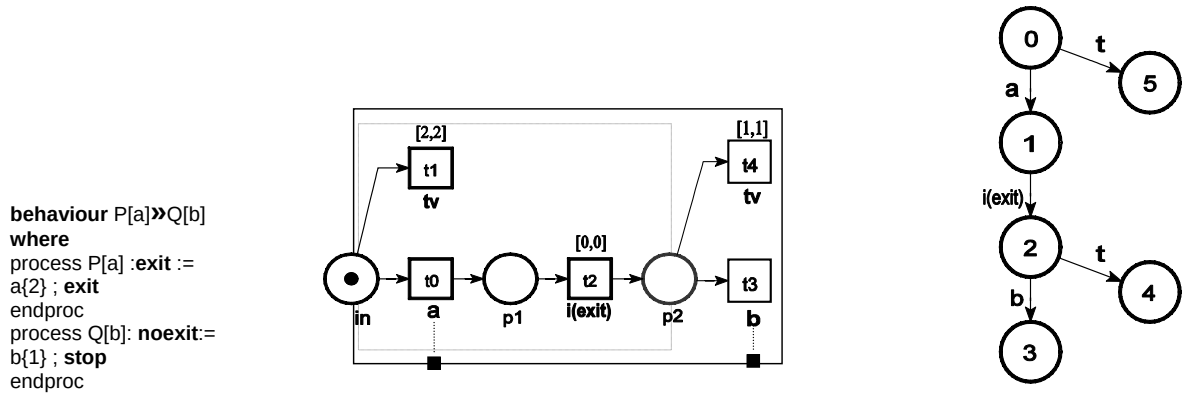
Définition formelle

$$C_{P \gg Q} = \left\langle \sum_{P \gg Q}, Lab_{hide\ exit\ in\ P} \cup Lab_Q, I_P, O_Q \right\rangle$$

$$\sum_{P \gg Q} = \langle P_P \setminus O_P \cup P_Q, T_{hide\ exit\ in\ P} \cup T_Q, Pre_P \cup Pre_Q, Post_{P \gg Q}, IS_P \cup IS_Q \rangle$$

$$Post_{P \gg Q} = (Post_P \setminus \{(t, O_P) | t \in^\bullet O_P\}) \cup \{(t, in_Q) | in_Q \in I_Q \wedge t \in^\bullet O_P\} \cup Post_Q$$

Validation expérimentale Dans ce qui suit le processus P est composé séquentiellement avec le processus Q.



(a) Comportement RT-LOTOS (b) Composant $C_{P|[a]|Q}$ (c) Graphe de classes

FIG. 3.20: Validation expérimentale du pattern de la composition séquentielle

3.7.3 Pattern du choix

$C_{P[]Q}$ (cf. Figure 3.21) est le composant qui se comporte comme le composant C_P ou comme le composant C_Q. Le temps peut progresser dans le composant $C_{P[]Q}$ s'il peut progresser à la fois dans C_P et dans C_Q.

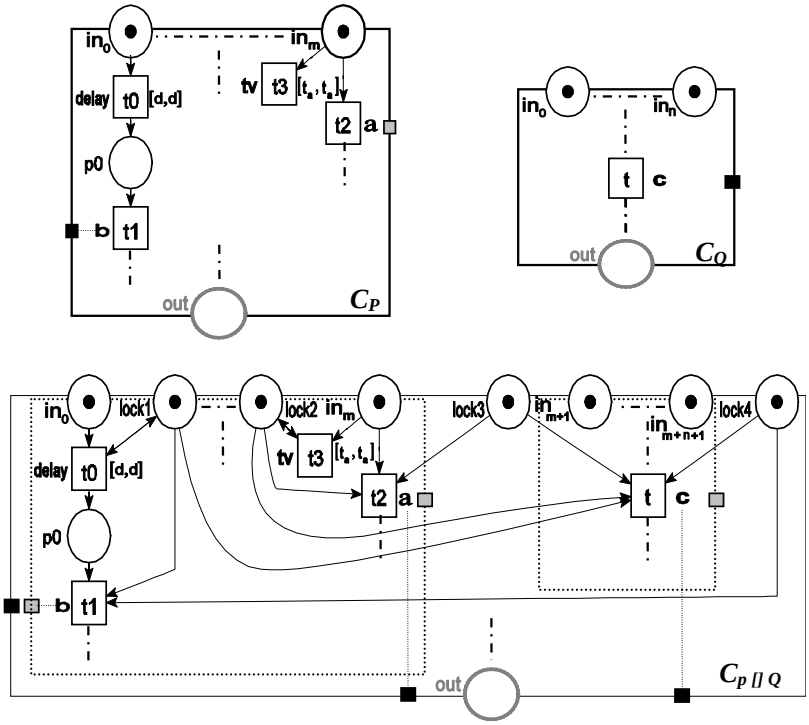


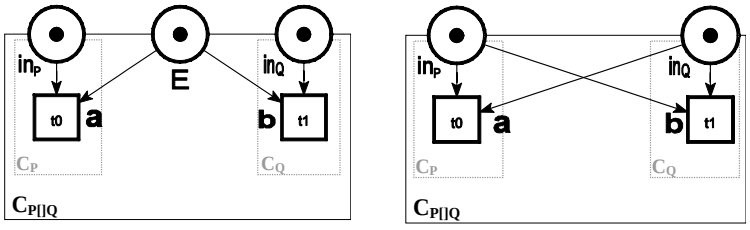
FIG. 3.21: Pattern du choix

Nous ne spécifions pas si le choix entre les deux alternatives (C_P et C_Q) est fait par le composant $C_{P[]Q}$ lui même, ou par son environnement. Néanmoins, ce choix doit être fait au niveau des premières actions dans $C_{P[]Q}$.

En d'autres termes, l'occurrence de l'une des premières actions dans l'un des composants C_P ou C_Q déterminera lequel des deux continuera son exécution et lequel sera désactivé. Ce problème peut être vu comme une compétition entre C_P et C_Q. En effet, ces deux composants sont en concurrence pour l'exécution de leur première action respective. Tant que l'occurrence de cette dernière ne s'est pas encore produite, C_P et C_Q vieillissent de manière similaire (le temps s'écoule de manière similaire dans les deux composants), ce qui signifie que des transitions étiquetées par des labels dans $Time$ peuvent être tirées sans aucune conséquence sur le choix du composant à exécuter. Une fois la première action exécutée, le contrôle est irréversiblement transféré au composant *gagnant* et l'autre composant est désactivé, dans le sens où il ne doit plus contenir des transitions sensibilisées, étant donné que son exécution n'est plus possible.
Ce schéma est le plus complexe parmi les schémas de traductions présentés dans cette thèse. Nous avons donc fait le choix de l'introduire en plusieurs étapes. Dans un premier temps, nous montrerons qu'une solution basée sur l'exclusion mutuelle telle qu'on la retrouve dans la littérature des RdP n'est pas suffisante dans le cadre de RT-LOTOS. Par la suite, nous expliquerons que le respect de la règle dite de *sensibilisation continue* (cf. Section 2.3) nécessite une attention particulière pour la mise en œuvre d'une solution d'exclusion mutuelle. Pour finir nous montrerons comment nous avons solutionné le problème qui se pose quand les alternatives du choix (C_P et/ou C_Q) contiennent du parallélisme initial (présence d'un opérateur parallèle au niveau des premières actions).

Dans ce qui suit nous considérons que a (resp b) est la première action du composant C_P (resp C_Q) .

Mise en oeuvre d'une solution d'exclusion mutuelle : l'idée de départ était de traduire le *choix* RT-LOTOS selon le pattern bien connu de l'exclusion mutuelle pour les RdP. Tel que présenté dans la Figure 3.22(a) (Une solution plus simple, utilisée pour LOTOS consiste à fusionner les places d'entrées des deux composants). Ce qui a pour effet que le tir de l'une des premières actions dans l'un des deux composants videra la place d'exclusion mutuelle E de son jeton, et ainsi désactivera la première action du composant concurrent. De la sorte, le contrôle est transféré, du composant $C_{P[]Q}$ vers l'un des composants C_P ou C_Q.



(a) Exclusion mutuelle (b) Choix PBC-like

FIG. 3.22: Choix entre C_P et C_Q

Une autre solution, présentée dans PBC [20], et décrite dans la Figure 3.22(b), consiste à faire en sorte que le tir de la première action dans un composant (C_P ou C_Q) vide directement la place d'entrée de la première action du composant concurrent et qui sera ainsi désactivé.

Ces deux schémas peuvent être appliqués pour LOTOS. Mais la présence des opérateurs temporels (*delay*, *latency* et *offre limitée* dans le temps) rend les choses plus compliquées et ces schémas se révèlent malheureusement insuffisants pour une réutilisation dans notre cadre.

Pour montrer l'inadéquation d'une réutilisation de la solution retenue dans [20] pour traiter le *choix* dans RT-LOTOS, considérons le cas décrit par la Figure 3.23 où les deux actions a et b sont contraintes par des offres limitées dans le temps ($a\{t_a\}$ et $b\{t_b\}$).

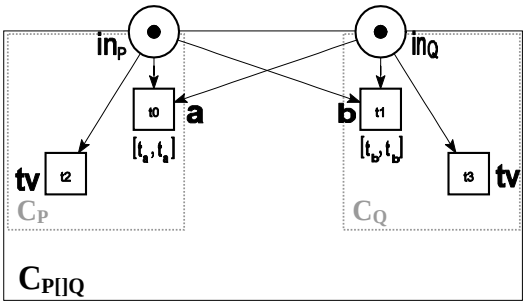


FIG. 3.23: Choix PBC-Like entre C_P et C_Q en présence d'offres temporelles

L'expiration de l'*offre limitée* sur l'action b dans le composant C_Q qui est représentée par le tir de la transition t_3, condamnera l'exécution de l'action a dans le composant C_P. Ce qui constitue un comportement erroné, étant donné que l'évolution temporelle d'un composant ne devrait pas interférer avec l'exécution de l'autre composant impliqué dans le choix. Rappelons que le tir des transitions dans *Time* ne doit avoir aucune incidence sur le choix du composant à exécuter, de même que l'expiration d'un opérateur temporel dans RT-LOTOS est traitée comme un écoulement du temps (cf. règles SOS de RT-LOTOS).

Exclusion mutuelle et respect de la règle de sensibilisation continue : Supposons toujours que les composants C_P et C_Q commencent par des offres temporelles sur les actions a et b respectivement, et regardons maintenant l'adéquation du pattern de l'exclusion mutuelle, décrit par la Figure 3.24, dans pareil cas.

Comme nous l'avions expliqué précédemment, l'occurrence de l'action a doit désactiver le composant C_Q. Ainsi l'exécution de l'action a (resp b) doit rendre impossible l'exécution de l'action b (resp a) , mais doit aussi interdire le tir de la transition t_3 (resp t_2) (t_2 et t_3 étiquetées par *tv*, représentent l'expiration des *offres temporelles* sur a et b respectivement*)* dans le composant C_Q (resp C_P). Étant donné que l'évolution temporelle (le tir de transitions avec des labels dans *Time*) d'un composant désactivé doit être stoppée, la place *d'exclusion* E doit être aussi une place d'entrée pour la transition t_3 ($E \in {}^\bullet t_3$). A contrario, l'expiration de l'offre limitée sur b dans C_Q ne doit pas interférer avec le comportement du composant C_P. Ainsi, le tir de la transition t_3 ne doit pas vider la place E de son jeton, car E est une

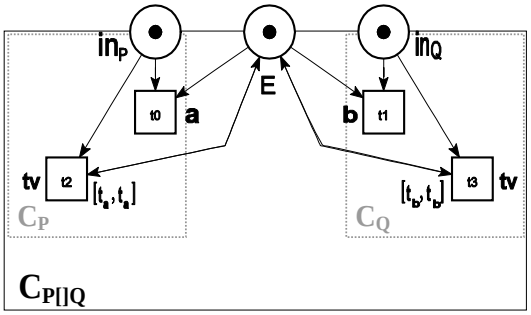


FIG. 3.24: Exclusion mutuelle entre C_P et C_Q en présence d'offres temporelles

place d'entrée pour l'action a ($E \in {}^\bullet t_0$). Dans le cas non temporisé, la solution à ce problème est de rajouter un arc d'entrée à la place E ($E \in {}^\bullet t_3 \cap t_3^\bullet$) tel qu'illustré dans la Figure 3.24. Cependant cette solution s'avère insuffisante dans le cas temporisé, et ce à cause de la règle de la *sensibilisation continue*. Ainsi, le tir de t_3 dans le composant C_Q, réinitialisera le timer associé à t_2 dans le composant C_P, ce qui constitue une violation de la sémantique RT-LOTOS.

Par conséquent, la place d'exclusion mutuelle E ne peut être partagée avec les transitions étiquetées par un label dans $Time$. Ainsi pour chaque transition $t \in T^{Time}$ dans C_P (resp C_Q), une place $E^t_{P[]Q}$ (appelée *lock-place* dans la suite) est rajoutée. Cette dernière est une place d'entrée/sortie pour la transition t, mais aussi pour toutes les premières actions dans C_P (resp C_Q). De plus ces places nouvellement introduites doivent appartenir à l'interface d'entrée du composant $C_{P[]Q}$.

Nous décrivons dans la Figure 3.25, le mécanisme de rajout des *lock-places* pour un composant C_P dont la première action a est contrainte par la somme des trois opérateurs temporels de RT-LOTOS ($delay(d)latency(l)a\{t\}$), ce qui est un maximum en pratique.

Notons que l'exécution de l'action b dans le composant C_Q videra toutes les places *lock* introduites de leurs jetons, ce qui bloquera l'évolution temporelle du composant C_P.

Pour des raisons de lisibilité nous ne représentons pas les arcs liant les places *lock* à l'action a dans le composant C_P. Notons pour finir, qu'en pratique, la seule place $E_{P[]Q}$ est suffisante pour gérer à la fois les transitions étiquetées par *delay* et *tv*.

Gestion de l'opérateur parallèle : La traduction de l'opérateur du choix est connue pour être un problème difficile en présence de parallélisme initial. Pour éviter cette complexité, [83] définit un opérateur de choix où chaque alternative a une et une seule place d'entrée, interdisant de fait la présence de parallélisme initial dans les alternatives du choix.

Dans le cadre de cette thèse, nous avons solutionné le problème posé par la présence de parallélisme initial sans imposer de restrictions sur les alternatives.

Pour illustrer le problème et décrire la solution que nous proposons, considérons le processus RT-LOTOS suivant : $P[](Q|||Q')$, a, b et c étant les premières actions des processus P, Q et Q' respectivement.

Pour traduire ce type de comportements, l'utilisation d'une seule place d'exclusion mutuelle se révèle insuffisante. On peut voir dans la Figure 3.26(a) que l'exécution de l'une des premières

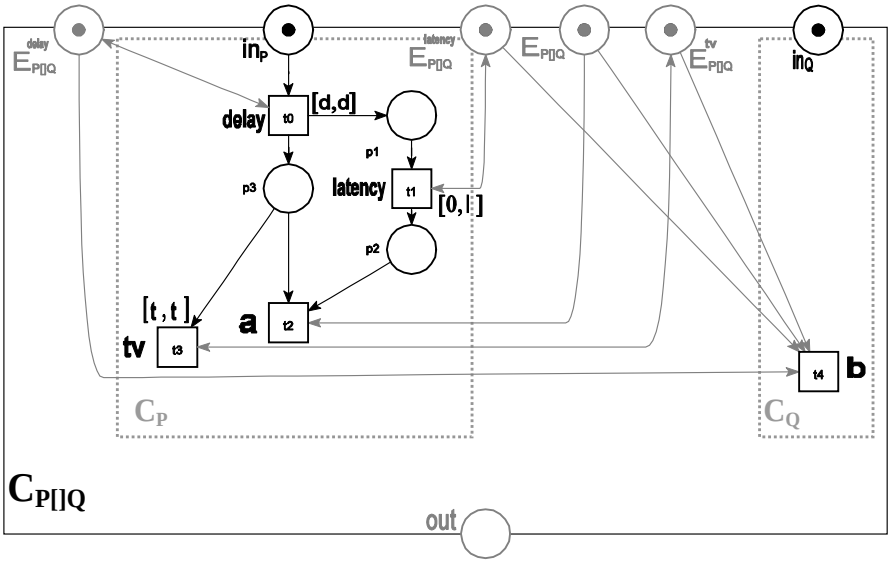


FIG. 3.25: Mécanisme de rajout des places *lock*

actions (action b du composant C_Q) dans le composant $C_{Q|||Q'}$ désactivera effectivement le composant C_P mais aussi le composant $C_{Q'}$, ce qui constitue un comportement erroné. Ainsi, pour garantir une exclusion mutuelle dans $C_{P[](Q|||Q')}$, nous introduisons une place d'exclusion mutuelle pour chaque première action dans le composant C_P et ceci par chaque première action dans le composant $C_{Q|||Q'}$. Ainsi, comme le décrit la Figure 3.26(b), pour la première action a dans C_P, une place d'exclusion mutuelle $E_{a,b}$ (resp $E_{a,b'}$) est introduite pour chaque première action b (resp b') dans $C_{Q|||Q'}$.

Arrivés a ce point, faisons un récapitulatif des places à introduire pour le pattern du choix :

Notons, d_t(resp l_t, tv_t) les transitions représentant un *délai déterministe* (resp. *latence*, *violation temporelle*) associée à une transition t.

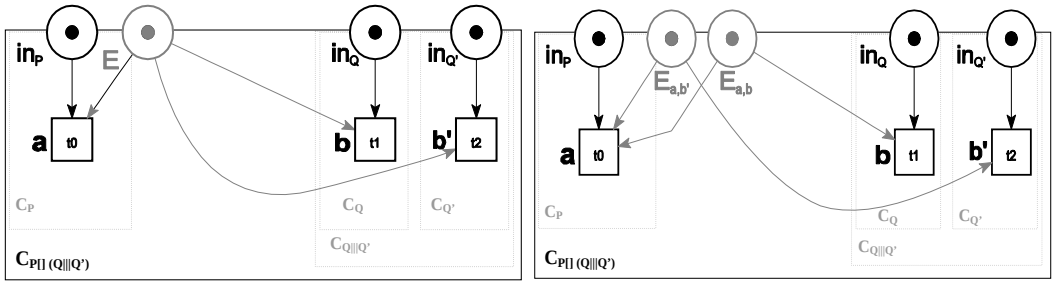
$\forall t \in \mathcal{FA}(\mathrm{C_P}), \forall t \in \mathcal{FA}(\mathrm{C_Q})$, les places nouvellement introduites sont :

$\langle \phi, E_{t,t'}, \{t, t'\} \rangle, \left\langle l_t, E^l_{t,t'}, \{l_t, t'\} \right\rangle, \left\langle tv_t, E^{tv}_{t,t'}, \{tv_t, t'\} \right\rangle$ *et* $\left\langle d_t, E^d_{t,t'}, \{d_t, t'\} \right\rangle$

Bien évidemment la même opération doit être effectuée pour le composant C_Q.

La purge des jetons restants dans le composant désactivé : Une dernière opération doit être mise en œuvre pour une définition complète du pattern de choix. Cette opération appelée *purge* vise à régler un problème posé par les places introduites par les opérateurs *délai déterministe* et *latence*. Ces places ont la particularité de rester marquées après la désactivation du composant auquel elles appartiennent (le fait que ces places là soient marquées ne suffit pas cependant pour sensibiliser une transition dans le composant). Le marquage de ces places pourrait poser des problèmes dans le cas de comportement réentrants (composants cycliques), en particulier le non respect de l'hypothèse des composants 1-bornés. Par conséquent, ces jetons doivent être purgés.

Cette tâche incombe à la dernière action du composant *gagnant*, qui devra donc purger le composant désactivé.



(a) (b)

FIG. 3.26: Choix entre C_P et C_Q en présence de l'opérateur parallèle

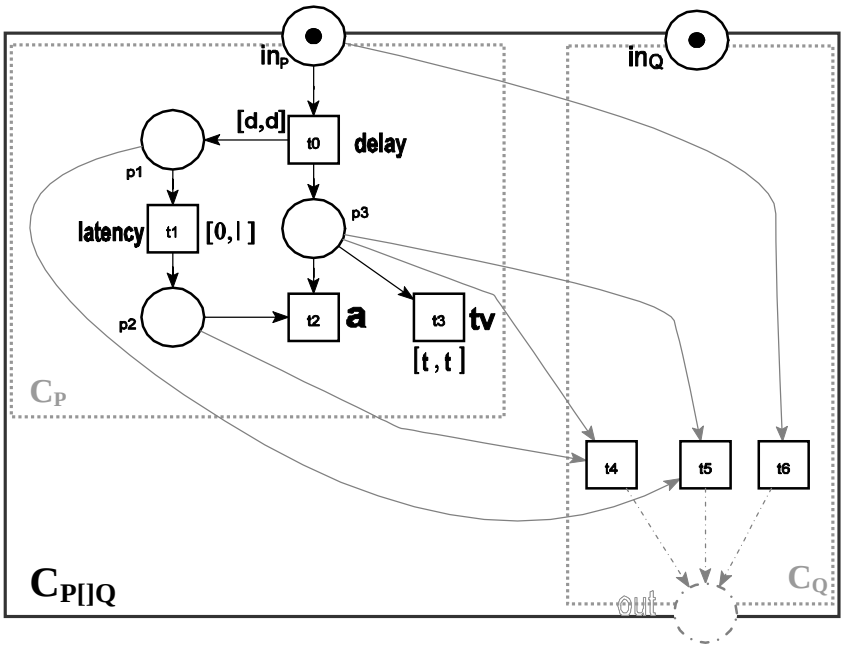
Soit $\mathcal{P}r_{\mathcal{FA}}(C_X)$: l'ensemble des marquages atteignables avant l'occurrence de la première action du composant C_X.

Pour chaque transition $t \in \mathcal{LA}(C_Q)$, et pour chaque marquage atteignable m pour les places dans $\mathcal{P}r_{\mathcal{FA}}(C_P)$, une transition t^m est crée ($t^{m\bullet} = \{out\}$) (la dernière action du composant qui fait la purge est dupliquée autant de fois qu'il existe de marquages atteignables).

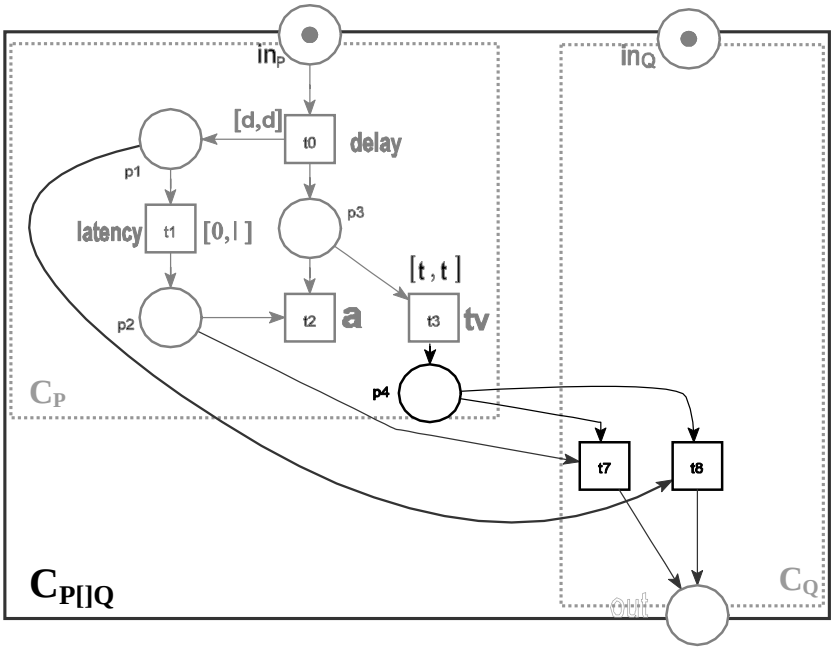
Par construction, une est une seule de ces transitions dupliquées est sensibilisée. Son tir videra le marquage m ($^\bullet t^m = {}^\bullet t \cup m$).

Il reste cependant un dernier détail à régler qui est posé par l'existence de marquages *m, m'* tels que $m \subset m'$. Les seules transitions ayant la faculté de faire évoluer un marquage m vers un marquage *m'* tel que $m \subset m'$ sont les transitions étiquetées par *tv,* précisément parce qu'elles n'ont pas de places de sortie. Pour régler ce problème nous rajoutons donc une place de sortie a ces transitions (cf. Figure 3.27(b) pour le traitement du cas de la transition t_3).

Le nombre des marquages accessibles avant l'occurrence de la première action est exponentiel en nombre des opérateurs *delay* et *latency*. En cas d'un *delai* et d'une *latence*, il y a trois marquages accessibles possibles : $\{^\bullet delay\}, \{delay^\bullet, ^\bullet latency\}, \{delay^\bullet, latency^\bullet\}$(cf. Figure 3.27(a)) ; en pratique c'est un maximum.



(a)



(b)

FIG. 3.27: Purge du composant C_P

Description informelle et récapitulatif : Pour obtenir le choix entre C_P et C_Q. Nous introduisons un ensemble de places appelées *Lock*. Ces dernières appartiennent à l'interface

d'entrée du composant $C_{P[]Q}$. Leur rôle est d'assurer le transfert de contrôle entre les deux composants. Pour chaque couple d'actions (première action dans C_P, action concurrente dans C_Q) on introduit une place *lock* (par exemple dans la Figure 3.21 l'action a possède une seule action concurrente dans C_Q qui est l'action c, alors que cette dernière en possède deux dans le composant C_P : a et b). Une place *lock* interagit seulement avec le couple de premières actions qui lui est associé ainsi qu'avec les transitons $Time$ présentes (*delay* pour a et tv pour b).

Les transitions $Time$ restaurent le jeton dans les places *lock* (puisqu'elles ne représentent pas une occurrence d'action mais une progression temporelle). L'occurrence d'une première action dans C_P (resp C_Q) bloquera l'exécution de C_Q (respectivement C_P) , en vidant de leurs jetons toutes les places *lock* rattachés au composant C_Q (resp C_P).

Définition formelle Nous introduisons les ensembles suivants :

- $Split(C_X)$: l'ensemble des transitions obtenus par la duplication de la dernière action de C_X.

$$Split(C_X) = \{t^m | t \in \mathcal{LA}(C_X), m \in \mathcal{P}r_{\mathcal{FA}}(C_P)\}$$

- $Lock_{C_X}$ est l'ensemble de places *lock* introduites dans le composant C_X.

$$Lock_{C_X} = \bigcup_{t \in \mathcal{FA}(C_X), t' \in \mathcal{FA}(C_Y)} \langle \emptyset, E_{t,t'}, \{t, t'\} \rangle$$

- Si t est une transition de C_X avec un label dans Act , nous appelons $Tev(t)$ l'ensemble des évènements temporels qui lui sont associés :

$$\begin{aligned} Tev(t) = &\{d_t \in T_X | Lab_X(d_t) = delay \wedge t \in d_t^{\bullet\bullet}\} \\ &\cup \{l_t \in T_X | Lab_X(l_t) = latency \wedge t \in d_t^{\bullet\bullet}\} \\ &\cup \{tv_t \in T_X | Lab_X(tv_t) = tv \wedge^\bullet t \cap^\bullet tv \neq \emptyset\} \end{aligned}$$

- $Lock_{C_X}^{Tev}$ est l'ensemble de places $Lock$ introduites dans le composant C_P afin de gérer les transitions étiquetées dans $Time$.

$$Lock_{C_P}^{Tev} = \bigcup_{t \in \mathcal{FA}(C_X), t' \in \mathcal{FA}(C_Y), v \in Tev(t)} \langle \{v\}, E_{t,t'}^v, \{v, t'\} \rangle$$

- L'ensemble de toutes des places *lock* introduite est ainsi défini par :

$$Lock = Lock_{C_P} \cup Lock_{C_P}^{Tev} \cup Lock_{C_Q} \cup Lock_{C_{PQ}}^{Tev}$$

La définition du pattern du choix suit :

$$C_{P[]Q} = \left\langle \sum_{P[]Q}, Lab_{P[]Q}, I_P \cup I_Q \cup Lock, O_{P[]Q} \right\rangle$$

avec :

$$\sum_{P[]Q} = (\sum_{P} \backslash O_P) \uplus (\sum_{Q} \backslash O_Q)$$

l'union disjointe des deux RdPT encapsulés dans les composants C_P et C_Q

$$\cup lock\ \{\langle\{t\}, p_t, \emptyset\rangle \mid t \in T_X, Lab_X = tv, X \in \{P, Q\}\}$$

les places Lock et les places de sortie pour les transitions étiquetées par tv

$$\backslash\{t \in T_X | Lab_X = exit, X \in \{P, Q\}\}$$

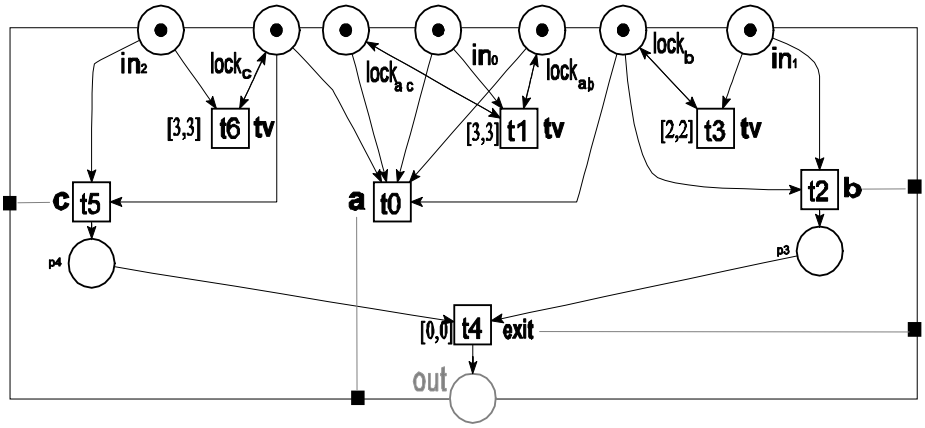
et la suppression des transitions étiquetées par $exit$

$$O_{P[]Q} = \begin{cases} \{out\} & si\ O_P \neq \phi\ ou\ O_Q \neq \emptyset \\ \emptyset & sinon \end{cases}$$

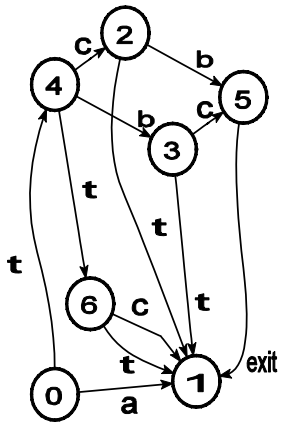
Validation expérimentale Soit le comportement RT-LOTOS décrit dans la Figure 3.28(a), le processus offre le choix entre deux alternatives. L'une d'entre elles contient du parallélisme initial. Le composant correspondant est décrit dans la Figure 3.28(b). Une fois encore nous obtenons le même automate quotient pour l'expression RT-LOTOS et le RdPT correspondant (cf. Figure 3.28(c)).

```
process P[a,b,c] :exit :=
a{2} ; stop
[]
(b{2};exit ||| c{3}; exit)
endproc
```

(a) Comportement RT-LOTOS



(b) Composant C_P



(c) Automate quotient

FIG. 3.28: Validation expérimentale du pattern du choix

3.7.4 Pattern de la préemption (*disrupt*)

Description informelle $C_{P[>Q}$ (cf. Figure 3.29) est le composant représentant le comportement où le composant C_Q peut interrompre le composant C_P, et ceci à n'importe quel moment durant l'exécution de ce dernier.

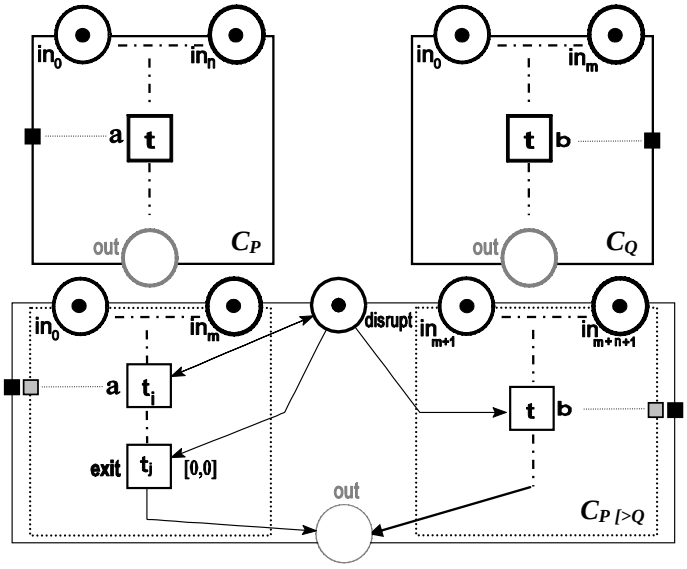


FIG. 3.29: Pattern du disrupt

Durant l'exécution du composant C_P, il y a un choix entre exécuter l'une des prochaines actions de C_P ou l'une des premières actions de C_Q. Pour obtenir le comportement voulu, C_Q vide la place *disrupt* de son jeton à l'occurrence de l'une de ses premières actions. Cette place nouvellement introduite appartient à l'interface d'entrée du composant $C_{P[>Q}$. Ainsi, le contrôle est irréversiblement transféré du composant C_P au composant C_Q. La place *disrupt* est une place d'entrée pour les premières actions de C_P et la transition *exit* de C_P (s'il en existe une) ; elle est aussi une place d'entrée/sortie pour toutes les autres transitions de C_P. Une fois une action du composant C_Q choisie, C_Q continuera son exécution, et les transitions de C_P ne seront plus sensibilisées.

Il y a des similitudes entre ce pattern et celui du choix présenté dans la section précédente. En effet, le tir de l'une des premières actions de C_Q désactivera le composant C_P. D'où le recours à une place d'exclusion mutuelle *disrupt*. En pratique l'introduction d'une seule place d'exclusion mutuelle ne suffit pas pour une modélisation correcte de l'opérateur [>. En voici l'explication.

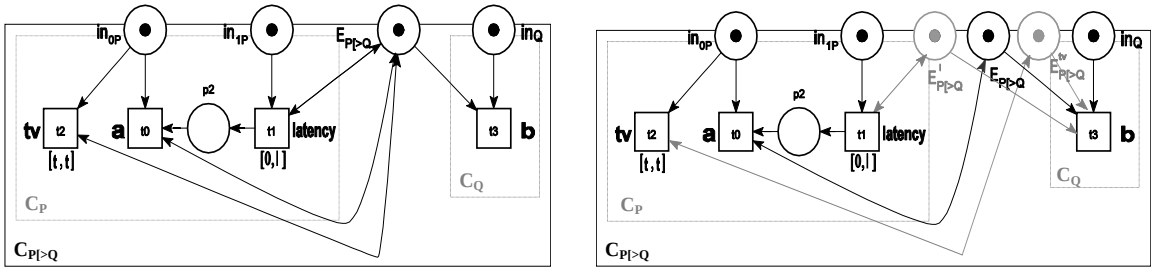
Soit $E_{P[>Q}$ cette place d'exclusion mutuelle.

$$(E_{P[>Q})^{\bullet} = \mathcal{FA}(\mathrm{C_Q}) \cup \bigcup_{\mathrm{t} \in \mathrm{T_P^{Act}} \setminus \mathcal{LA}(\mathrm{C_P})} \{\mathrm{t}\}$$

$$^{\bullet}(E_{P[>Q}) = \bigcup_{t \in T_P^{Act} \wedge Lab(t) = exit} \{t\}$$

Le respect de la règle de la *sensibilisation continue* dans les RdPT requiert aussi une attention particulière. En pratique le problème se pose pour une seule construction combinant à la fois les opérateurs de *latence* et d'*offre temporelle* sur une même action, comme dans l'exemple suivant :

Considérons un comportement RT-LOTOS dans lequel un processus $P = latency(l)a\{t\}P'$ peut être interrompu à tout moment de son exécution par un processus Q. L'ensemble des premières actions de ce dernier est réduit à la seule action b, nous avons ainsi $\mathcal{FA}(\mathrm{C_Q}) = \{\mathrm{b}\}$. Comme illustré dans la Figure 3.30(a), la place $E_{p[>Q}$ étant partagée par la transition t_2 représentant l'expiration de l'*offre temporelle* et étiquetée par tv, et la transition t_1 étiquetée par *latency*. L'expiration de la *latence* représentée par le tir de la transition t_1 provoquera la remise à zéro du timer associé à la transition t_2.



(a) (b)

FIG. 3.30: Introduction des places *disrupt*

Pour solutionner ce problème nous procédons de la même manière que pour le pattern du choix, en introduisant pour toute transition t étiquetée par un label dans $Time$ une place $E^t_{P[>Q}$ comme illustré dans la Figure 3.30(b).

$$\forall t \in T_P^{Time} (E^t_{P[>Q})^\bullet = \{t\} \cup \mathcal{FA}(\mathrm{C_Q})$$

$$^\bullet(E_{P[>Q}) = \{t\}$$

Comme pour le pattern du choix, le composant qui aura fini son exécution devra purger l'autre composant des jetons restants.

Nous distinguons deux cas de figure :

– C_P termine son exécution sans être interrompu par C_Q. Les jetons à purger se trouvent dans l'interface d'entrée de C_Q (cf. définition $\mathcal{P}r_{\mathcal{FA}}(C_Q)$ dans la section précédente), mais aussi dans les places introduites pour gérer l'occurrence des évènements temporels (tirs des transitions avec des labels dans $Time$) dans le composant C_P. Toute transition $t \in \mathcal{LA}(\mathrm{C_P})$ est dupliquée en un ensemble de transitions $\{t^m | m \in \mathcal{P}r_{\mathcal{FA}}(C_Q)\}$ avec $^\bullet t^m \cup m \cup \{E_{P[>Q}\} \cup \{E^v_{P[>Q} | v \in T_P^{Time}\}$

- C_Q termine son exécution, le composant C_p a donc été interrompu au cours de son exécution, mais il est impossible de savoir dans quel état, et donc de deviner l'emplacement des jetons à purger. Tous les marquages accessibles de C_P doivent être testés. Ainsi, pour tout marquage $M_P \in M(C_P)$, chacune des transitions parmi les dernières transitions de C_Q ($t \in \mathcal{LA}(\mathrm{C_P})$) doit être dupliquée en un ensemble $\{t^{M_P}\}$, de telle sorte que les places d'entrées de ces transitions augmentées du marquage M_P soient les places d'entrée pour la transition t .

Concernant la place de sortie du composant $C_{P[]Q}$, ce dernier en aura une si l'un des sous-composants C_P ou C_Q en a une.

Définition formelle Nous définissions les ensembles suivants :

$$Split(C_P) = \{t^m | t \in \mathcal{LA}(C_P), m \in Pre_{\mathcal{FA}}(C_P)\}$$

L'ensemble des transitions obtenues après le duplication de la dernière action de C_P.

$$Split(C_P) = \{t^m | t \in \mathcal{LA}(C_P), m \in \mathcal{M}(C_P)\}$$

Duplication de la dernière action de C_Q.

$$Disrupt^{Time} = \{E^v_{P[>Q} | v \in T_P^{Time}\}$$

Les places introduites pour désactiver les évènements temporels dans le composant C_P.

$$Disrupt = \{E_{P[>Q}\} \cup Disrupt^{Time}$$

$$O_{P[>Q} \quad = \quad \begin{cases} \emptyset & Si\ O_P = O_Q = \emptyset \\ \{out\} & sinon \end{cases}$$

$$PostSplit(C_X) = \begin{cases} \{(t^m, out) | t^m \in Split(C_X)\ Si\ O_X \neq \emptyset \\ \{(t^m, p) | t^m \in Split(C_X),\ p \in t^\bullet\ SiO_X = \emptyset \end{cases}$$

$$C_{P[>Q} = \left\langle \sum_{P[>Q}, Lab_{P[>Q}, I_P \cup I_Q \cup Disrupt, O_{P[>Q} \right\rangle$$

avec :

$$\sum_{P[>Q} = (\sum_P \backslash O_P \setminus \mathcal{LA}(C_P)) \uplus (\sum_Q \backslash O_Q \setminus \mathcal{LA}(C_Q))$$

l'union disjointes des deux RdPT encapsulés dans les composants C_P et C_Q, tronqués de leurs dernières actions et places de sorties.

$$\cup \left\langle T_P^{Act} \setminus \mathcal{LA}(C_P), E_{P[>Q}, \mathcal{FA}(C_Q) \cup T_P^{Act} \setminus \mathcal{LA}(C_P) \right\rangle$$

l'ensemble des places *disrupt* pour les actions de C_P

$$\cup \bigcup_{t \in T_P^{Time}} \left\langle \{t\}, E^t_{P[>Q}, \mathcal{FA}(C_Q) \cup \{t\} \right\rangle$$

l'ensemble des places *disrupt* pour les évènements temporels

$$\cup \bigcup_{t \in T_P^{Time} \wedge Lab(t)=tv} \langle \{t\}, p_t, \emptyset \rangle$$

rajout d'une place de sortie pour les transitions étiquetées par tv

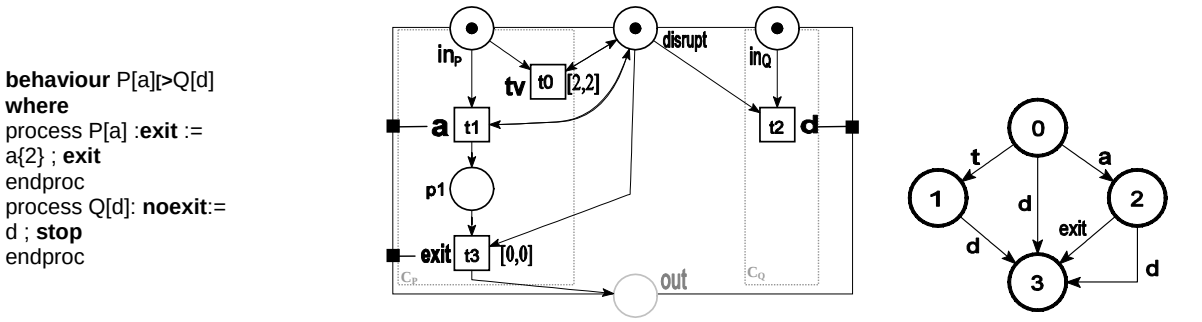
$$\cup \bigcup_{t^m \in Split(C_P)} \langle {}^\bullet t \cup m \cup Disrupt, t^m, O_{P[>Q} \rangle$$

la dernière action de C_P purge C_Q et les places *disrupt*

$$\cup \bigcup_{t^m \in Split(C_Q)} \langle {}^\bullet t \cup m, t^m, O_{P[>Q} \rangle$$

la dernière action de C_Q purge C_P.

Validation expérimentale Le processus P, peut être interrompu à tout moment de son exécution par le processus Q.



(a) Comportement RT-LOTOS (b) Composant $C_{P|[a]|Q}$ (c) Automate quotient

FIG. 3.31: Validation expérimentale du pattern du *disrupt*

3.8 Comparaisons avec des approches existantes

Petri Box Calculus temporisé (tPBC) : Une extension temporelle du Petri Box Calculus (PBC) [20] a été proposée dans [54]. Dans ce modèle les contraintes temporelles sont introduites en associant un intervalle de tir à chaque action. Cette façon de procéder est directement inspirée des RdPT , où le tir des transitons est dirigé par la nécessité. Cependant un problème connu avec cette stratégie (cf. section 2.3) est qu'elle n'est pas compositionnelle. La difficulté principale réside dans la composition d'intervalles (pas toujours compatibles) pour modéliser la synchronisation de transitons. La sémantique de [54] est basée sur l'intersection des intervalles

temporels afin d'obtenir un intervalle unique pour les transitions synchronisées. Comme l'on pouvait s'y attendre cette façon de faire n'est pas toujours satisfaisante, comme le montre l'exemple suivant.

Soit $E_1 = ((a[10,10]; b[2,2] \,||\, \hat{b}[12,12])\ sy\ b$, un terme tPBC.
E_1 décrit la synchronisation parallèle sur l'action b des deux termes suivants (la synchronisation des deux actions conjuguées b et $\hat{b}$ donne naissance à l'action interne i) :

- $a[10,10]; b[2,2]$: ce terme tPBC exécute a après 10 unités de temps ; b quant à elle est exécutée après 2 unités de temps après l'occurrence de a, et
- $\hat{b}[12,12]$: ce terme exécute $\hat{b}$ après 12 unités de temps.

Ainsi, d'après la description donnée ci-dessus, les occurrences des actions $\hat{b}$ et b ont lieu à la même date (12 unités de temps à compter de l'instant initial). La synchronisation sur l'action b est donc possible. Néanmoins, la règle définie pour l'opérateur de synchronisation dans [54], stipule que la synchronisation n'est possible que si elle conduit à une action bien définie i.e avec une information temporelle consistante. Ce qui n'est clairement pas le cas dans l'expression E_1, étant donné que l'intersection des deux intervalles temporels des actions $\hat{b}$ et b est vide ($[2,2] \cap [12,12] = \phi$).

Par conséquent, dans [54] le RdPT (appelé *ctbox*) correspondant à l'expression E_1 ne peut être construit.

Pour résoudre ce problème et éviter cette composition *problématique* d'intervalles, nous évitons soigneusement l'association d'intervalles temporels aux transitions représentant des actions. Dans le cadre de cette thèse, les contraintes temporelles sont assignées à des transitions dédiées. Ainsi la synchronisation des transitions peut être obtenue de manière simple en fusionnant directement les transitions concernées par la synchronisation sans avoir à se soucier des contraintes temporelles qui leurs sont associées.

Le comportement RT-LOTOS suivant est équivalent à l'expression E_1 :

```
P1= hide a, b in ( delay(12)b ; stop
                 |[b]| delay(10)a ; delay(2)b ; stop)
```

Le RdPT correspondant à *P1* (cf. Figure 3.32(a)) peut être construit de manière très simple en appliquant les patterns décrits dans ce chapitre. Notons que la synchronisation sur l'action b se produit effectivement après 12 unités de temps.

De plus, même si la synchronisation conduit à ce que les auteurs de [54] appellent une action *bien définie* (i.e. avec une contrainte temporelle consistante). Il leur est nécessaire d'enrichir la sémantique du tPBC avec des règles autorisant le tir d'actions dites *illégales*. Une action *illégale* est une action qui a été sensibilisée pendant un temps dépassant sa borne de tir supérieure.

Pour illustrer ce point particulier, considérons le terme tPBC suivant :
$E_2 = (a[0,0] \,||\, (b[1,1]; \hat{a}[0,1]))\ sy\ \{a\}$
E_2 ne peut exécuter l'action de synchronisation i, ce qui est consistant avec la sémantique du RdPT correspondant (cf. Figure 3.32(b)).

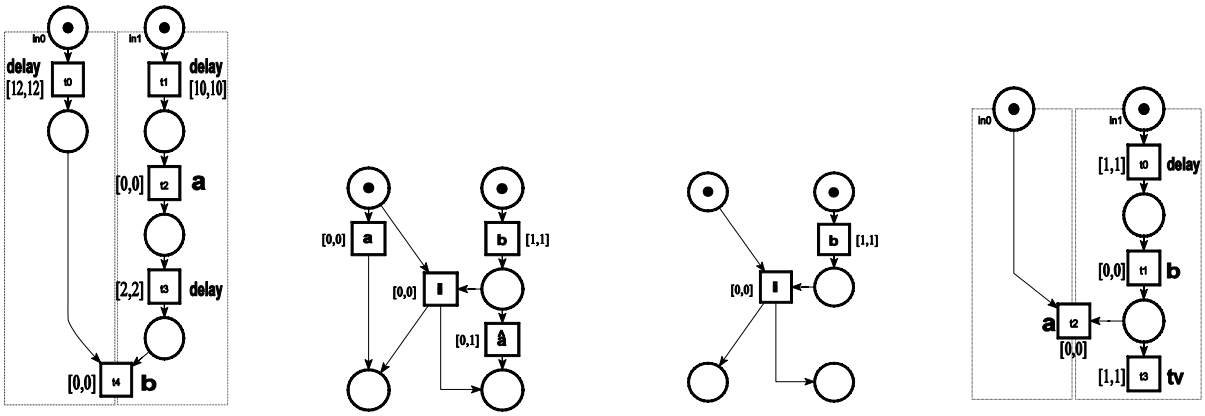
Cependant, les choses changent pour l'expression $E_2\ rs\{a\}$. Dans ce dernier terme, la synchronisation est forcée par l'opérateur de restriction. Le RdPT correspondant est alors obtenu en supprimant les transitions a et $\hat{a}$ (cf Figure 3.32(c)). Il est maintenant possible d'exécuter l'action b (après une unité de temps) immédiatement suivie de l'action i. Un tel résultat ne serait pas obtenu en considérant uniquement des actions dites légales, par exemple l'occurrence de l'action a (contrainte par un intervalle statique de [0,0]) n'est pas légale après l'écoulement

d'une unité de temps. Les auteurs introduisent à cet effet dans [54] une règle autorisant le tir de telles actions. Ces règles assurent la consistance entre les termes tPBC et leurs RdPT correspondants (appelés *ctbox*) mais nous pensons que ces règles supplémentaires ont un effet négatif sur la simplicité de la sémantique opérationnelle de [54] .

Le comportement RT-LOTOS suivant est équivalent à $E_2\, rs\{a\}$.

```
P2= hide a, b in
(a ; stop
|[a]|
delay(1)b ; a{1} ; stop)
```

La consistance entre *P2* est le RdPT correspondant décrit dans la Figure 3.32(d) est assurée sans s'éloigner des sémantiques originales de RT-LOTOS et des RdPT.



(a) composant C_{p1} (b) $ctbox_1$ (c) $ctbox_2$ (d) composant C_{P2}

FIG. 3.32: Composants Vs ctbox

***Caesar* [38] :** [34] est la seule approche opérant une traduction complète de LOTOS, traitant à la fois les parties *contrôle* et *données* de LOTOS. L'approche a été implémentée dans l'outil *Caesar* [38]. Mis à part les aspects temporels, une différence technique entre le travail présenté dans cette thèse et l'approche développée dans [34] réside dans la façon dont ces deux approches structurent les réseaux générés. Notre solution est basée sur les composants, alors que [34] utilise des unités dont chacune contient au maximum un jeton. Cet invariant limite la taille des marquages et permet des optimisations mémoire, mais rend nécessaire l'usage des $\varepsilon - transitions$, qui peut s'avérer inefficace dans certains cas.

Pour se positionner plus précisément par rapport à [34], abordons la question de l'opérateur *disrupt*. Pour traiter l'interruption d'un processus P par un processus Q, [34] propose d'introduire des $\varepsilon - transitions$ pour relier chaque état interruptible de P avec l'état d'entrée du processus Q. Nous avons testé une approche similaire pour RT-LOTOS en prenant non pas des $\varepsilon - transitions$ mais des transitions étiquetées par les premières actions du processus Q. L'inconvénient de cette solution est apparu lorsque nous avons donné à un processus Q la possibilité d'interrompre plusieurs processus *P1 ... Pn* placés en parallèle. La solution que nous avons proposée dans la Section 3.7.4 évite ce problème d'explosion combinatoire. A l'appui de nos dires, nous pouvons mentionner le banc d'essai réalisé sur la spécification LOTOS (non temporisée) suivante :

```
specification Sender2 : exit
behaviour
hide send_data, end_transfert, receive_ack, err in
  (sender[send_data, end_transfert, receive_ack]
|[end_transfert]|
  sender[send_data, end_transfert, receive_ack])
[>
 error[err]
where
process sender[send_data, end_transfert, receive_ack] : exit :=
send_data ; end_transfert ; receive_ack ; exit
endproc

process error[err] : noexit :=
err ; stop
endproc
endspec
```

Bien qu'artificiel, cet exemple a pour but d'accroître la complexité du comportement considéré et de comparer ainsi les performances de *rtl2tpn+Tina* et *Caesar* sur une construction comportement l'opérateur *disrupt*. Nous utilisons les options de *Tina* et de *Caesar* qui permettent de générer un automate au format *Aldebaran*, ce qui nous permet de vérifier par ailleurs que ces automates sont fortement bissimilaires.

Le tableau 3.1 permet de comparer les deux approches en termes de RdP engendré et de temps d'exécution.

	rtl2tpn+tina		***Caesar***	
#sender	RdP(p/t)	Automate (CPU)	RdP(p/t)	Automate (CPU)
2	13 / 7	$<$ 1s	11 / 16	$<$ 1s
5	28 / 13	$<$ 1s	32 / 43	1s
8	43 / 19	$<$ 1s	53 / 70	1s
10	53 / 23	$<$ 1s	67 / 88	6s
13	69 / 29	8s	88 / 115	1514s
15	78 / 33	23s	102 / 133	$>$24h

Tab. 3.1: Résultats comparatifs de *rtl2tpn+tina* Vs *Caesar*

3.9 Preuve de la consistance de la traduction

Dans cette section nous montrons que la traduction que nous venons de décrire préserve la sémantique RT-LOTOS, et que le cadre compositionnel que nous avons défini pour les composants préserve les bonnes propriétés (H1-H5) de ces derniers. Pour cela nous procèderons par induction structurelle sur les termes RT-LOTOS et sur les composants correspondants.
Nous supposons que les composants $C_1, \ldots, C_n$ sont respectivement *équivalents* aux termes RT-LOTOS $T_1, \ldots, T_n$ et étant donné un opérateur RT-LOTOS Op, nous prouvons que l'application du schéma de traduction (correspondant à Op) aux composants $C_1, \ldots, C_n$ donne un

composant *équivalent* au terme $Op(T_1, \ldots, T_n)$ (le comportement temporel doit évidemment être pris en compte).

Pour cela nous procéderons en deux étapes :

1. Dans un premier temps, nous définissons une sémantique plus *informative* pour RT-LOTOS. Cette dernière n'introduit aucune nouvelle construction, mais renseigne explicitement sur l'occurrence des *évènements temporels*. Un *évènement temporel* représente l'expiration de l'un des opérateurs temporels de RT-LOTOS. Illustrons ce point sur l'opérateur de *l'offre limitée* dans le temps. Considérons, pour cela la sémantique opérationnelle originale de cet opérateur décrite dans la Figure 2.5.

   $$a\{d\};\ P \xrightarrow{d'} stop$$

   Cette règle stipule, que tout délai $d' > d$ transformera le processus $a\{d\};P$ en *stop*. Mais dans la sémantique *informative* que nous introduisons, une action *tv* est rajoutée pour renseigner sur l'expiration de l'offre temporelle.

   $$a\{d\};\ P \xrightarrow{d} a\{0\};\ P \xrightarrow{tv} stop \xrightarrow{d'-d} stop$$

   Dans cette règle que nous substituons à l'ancienne, un délai $d' > d$ est evidemment toujours permis, mais il est divisé en trois étapes : un délai d, une *violation temporelle (tv)* et un délai $d' - d$.
   Il est relativement simple de définir une relation d'équivalence [5] qui abstrait ces évènements temporels nouvellement introduits et de montrer que cette sémantique informative de RT-LOTOS est ainsi équivalente à la sémantique originale de RT-LOTOS.

2. Dans un deuxième temps, nous prouvons que le modèle sémantique des composants est fortement bisimilaire à la sémantique *informative* de RT-LOTOS.
   Intuitivement un terme RT-LOTOS et un composant sont temporellement bisimilaires [88], ssi ils exécutent la même action au même moment et atteignent des états bisimilaires.
   A cet effet, nous prouvons -pour chaque opérateur et pour tout état atteignable-, que si un écoulement du temps (resp une occurrence d'action) est possible dans un terme RT-LOTOS, alors il (resp elle) est aussi possible dans le composant correspondant, et vice-versa. Il est intéressant de noter que durant l'exécution d'un composant, la structure du réseau reste la même et que seul le marquage change, alors que la structure d'un terme RT-LOTOS peut changer durant son exécution. Par conséquent, le réseau d'un composant peut ne pas correspondre directement à la traduction d'un terme RT-LOTOS mais il est fortement bisimilaire à un réseau qui correspond à la traduction de ce même comportement RT-LOTOS (même réseau, même état courant mais tronqué de la partie inatteignable).

Dans ce qui suit nous illustrons la preuve sur l'opérateur de synchronisation.

[5] Cette relation d'équivalence ressemble à la bisimulation faible de CCS qui abstrait les actions internes. Une différence notable est que les évènements temporels ne résolvent pas le choix (et le *disrupt* dans le cas de RT-LOTOS) contrairement aux actions internes de CCS.

Notation : un paragraphe commençant par $\underrightarrow{\mathbb{R}}$ a pour vocation de prouver que tout écoulement du temps possible dans le terme RT-LOTOS, l'est aussi (possible) dans le composant correspondant. $\underleftarrow{\mathbb{R}}$ dénote un paragraphe dédié à la preuve dans le sens opposé.

Similairement les notations $\underrightarrow{a}$ et $\underleftarrow{a}$ sont utilisées pour la preuve sur les occurrences des actions.

Preuve de l'opérateur de synchronisation

$\underrightarrow{\mathbb{R}}$: un écoulement du temps est possible dans $C_{P|[A]|Q}$ ssi cet écoulement est possible pour chaque transition sensibilisée.
Par construction, $T_{P|[A]|Q} = (T_P \backslash T_P^{A'}) \cup (T_Q \backslash T_Q^{A'}) \cup E$, E étant l'ensemble des transitions nouvellement crées.

$$E = \cup_{t_p \in T_P^{A'}, t_q \in T_Q^{A'}} merge(t_p, t_q)$$

Soit t une transition sensibilisée de $C_{P|[A]|Q}$. Nous distinguons les cas suivants :

1. $t \in (T_P \backslash T_P^{A'})$, alors cet écoulement du temps est acceptable pour t, étant donné que la contrainte temporelle associée à cette dernière dans C_P n'a pas été changée, et que t n'est pas concerné par la synchronisation. Le même raisonnement s'applique pour $t \in (T_Q \backslash T_Q^{A'})$.
2. $t \in E$, nous commençons par montrer que $Lab(t) \neq exit$. Supposons pour cela que $Lab(t) = exit$. Par construction, il existe alors t_p et t_q tel que $t = (t_p, t_q)$ avec $Lab(t_p) = Lab(t_q) = exit$. Rappelons que la transition *exit* est une action urgente spéciale (cf. H5), son exécution est forcée dès sa sensibilisation. Par construction, nous avons ${}^\bullet t =^\bullet t_p \cup {}^\bullet t_q$. Par conséquent pour que t soit sensibilisée, il faut que t_p le soit aussi dans C_P et que t_q le soit dans C_Q. La sensibilisation des tansitions t_p et t_q exclut toute possibilité d'écoulement du temps. Par induction, ceci exclut aussi tout écoulement du temps dans P et dans Q et par conséquent dans $P|[A]|Q$.
   De H5, nous avons que toute transition t avec $Lab(t) \neq exit$, est associée à un intervalle statique égal à $[0, \infty)$. Par conséquent un écoulement du temps est possible dans $C_{P|[A]|Q}$.

$\underleftarrow{\mathbb{R}}$: similairement à $\underrightarrow{\mathbb{R}}$.

$\underrightarrow{a}$: supposons que $P|[A]|Q \xrightarrow{a}$ et montrons que l'occurrence de l'action a est aussi possible dans $C_{P|[A]|Q}$.

Nous distinguons deux cas :

1. $a \in A'$: a est donc une action de synchronisation entre P et Q. Cette ation est possible dans $P|[A]|Q$ ssi elle l'est dans P et dans Q. Par induction, il existe deux transitions t_p et t_q étiquetées par a avec t_p tirable dans C_P et t_q tirable dans C_Q . Par construction, nous avons ${}^\bullet(t_p, t_q) =^\bullet t_p \cup^\bullet t_q$; ainsi le marquage $M_{P|[A]|Q}$ sensibilise (t_p, t_q). Après le tir de la transition, le marquage de $C_{P|[A]|Q}$ peut être vu comme l'union de ceux de $C_{P'}$ et $C_{Q'}$ car $(t_p, t_q)^\bullet = t_p^\bullet \cup t_q^\bullet$.
2. $a \notin A'$: si une action a est exécutée dans $P|[A]|Q$, alors cette dernière est soit une action P soit une action de Q. Par induction, elle est tirable soit dans C_P soit dans C_Q. Les ensembles *pre* et *post* de cette transition ne sont pas modifiées dans $C_{P|[A]|Q}$. L'action a est donc tirable dans $C_{P|[A]|Q}$.

$\underleftarrow{a}$:similaire à $\underrightarrow{a}$.

La preuve pour l'ensemble des opérateurs RT-LOTOS est présentée en Annexe A.

3.10 Expérimentations

3.10.1 Validation expérimentale

Les deux exemples suivants servent à valider expérimentalement l'approche proposée dans ce chapitre, et ceci sur la base de la comparaison des espaces des états engendrés d'une part par *rtl* et *rtl2tpn+Tina* d'autre part.

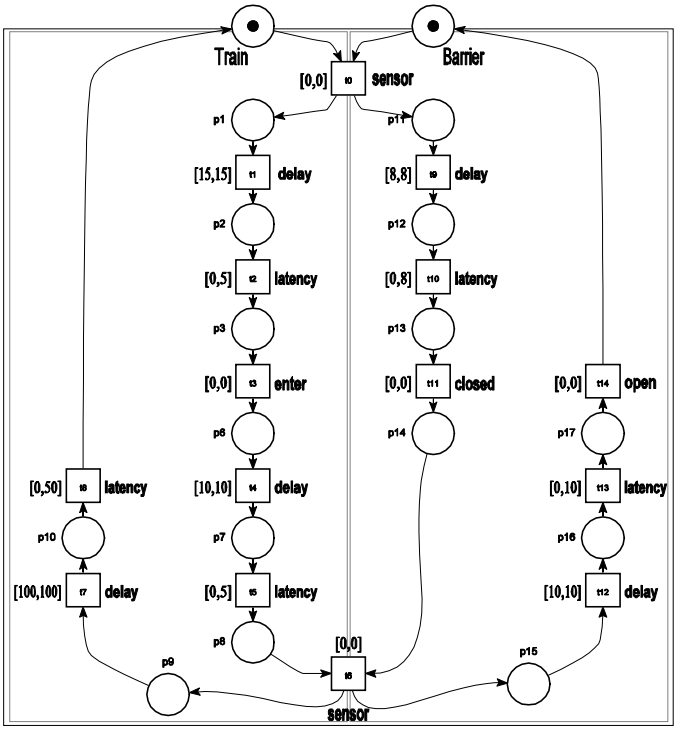
Passage à niveau

La spécification en langage naturel est tirée de [58]. Soit un système simple de contrôle ferroviaire sur lequel un train effectue des passages répétés. La barrière est contrôlée par un *sensor* : quand le train atteint le $sensor_1$ la barrière entame sa descente (action *down*) et elle est fermée (action *closed*) dans un délai de 8 à 16 secondes. Quand le train atteint le $sensor_2$, la barrière commence sa montée et elle est ouverte (action *open*) au bout d'un délai de 10 à 20 secondes. Après que le train ait passé le $sensor_1$, il lui faudra entre 15 et 20 secondes pour entrer (action *enter*) dans le passage à niveau et 10 à 15 secondes pour atteindre le $sensor_2$. Ensuite, le train mettra entre 100 et 150 secondes pour atteindre $sensor_1$ et boucler. La spécification RT-LOTOS du problème suit :

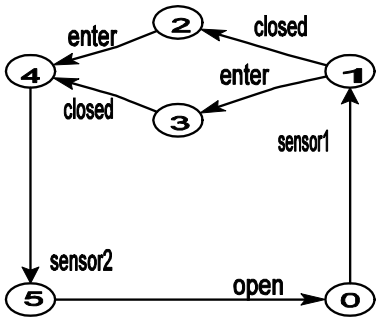
```
specification railway : noexit
behaviour
hide sensor1, sensor2, enter, closed, open  in

Train[sensor1,enter,sensor2]
|[sensor1,sensor2]|
Barrier[sensor1,closed,sensor2,open]

where

process Train[sensor1,enter,sensor2] :noexit :=
sensor1 ;  delay(15,20)enter ; delay(10,15)sensor2 ;
delay(100,150)Train[sensor1,enter,sensor2]
endproc

process Barrier[sensor1,closed,sensor2,open] :noexit :=
sensor1 ; delay(8,16)closed ; sensor2 ; delay(10,20)open ;
Barrier[snesor1,closed,sensor2,open]
endproc

endspec
```

La Figure 3.33(a) montre le RdPT engendré par *rtl2tpn* pour cet exemple. *Tina* engendre pour ce dernier un graphe de classes de 35 classes et 50 transitions.

Pour la spécification RT-LOTOS correspondante, *rtl* produit un graphe de 19 classes et 28 transitions. Ce petit nombre de classes résulte de l'algorithme de minimisation implémenté dans *rtl* et adapté de [87]. Après minimisation avec *Aldebaran* selon la procédure décrite dans la Section 3.4, nous obtenons le même automate quotient (cf. Figure 3.33(b)) pour les deux graphes engendrés par *rtl* et par *Tina*.



(a) $C_{Railway}$



(b) Automate quotient

FIG. 3.33: RdPT engendré par *rtl2tpn* et automate quotient

La vie simple

La spécification RT-LOTOS donnée ici, est inspirée d'une description de la vie donnée par Ken Turner dans [85].

A la différence de l'exemple précédent, nous ne donnons pas ici le RdPT final correspondant à cette spécification RT-LOTOS. A la place, nous donnons (cf. Figure 3.35) les deux composants correspondant aux deux processus *Biology* et *Family*.

rtl génère pour la spécification ci-dessus un graphe de régions de 7 classes et 11 transitions. *Tina* quand à lui génère pour la RdPT résultant de la synchronisation des deux composants

décrits dans la Figure 3.35 un graphe de classes de 13 classes et 24 transitions. Après minimisation avec *Aldebaran* [38], nous obtenons l'automate quotient de la Figure 3.34. Ce dernier est effectivement le même pour les deux graphes d'accessibilité.

```
specification Life[puberty, marriage, children] : noexit
behaviour hide birth, death in

Biology[birth,puberty,death]
|[puberty, death]|
Family[puberty, marriage, children, death]
where

process Biology[birth, puberty,death] :noexit :=
birth ;
(  (puberty{20} ; stop)
    [> (death ; stop)  )
endproc

process Family[puberty, marriage, children, death] :noexit :=
( (
    ( puberty ;
     (latency(30)marriage{20} ; stop) )
    |[puberty]|
    ( puberty ;
     (latency(20)children{30} ; stop) )
  )
  [> (death ; stop)
)
endproc

endspec
```

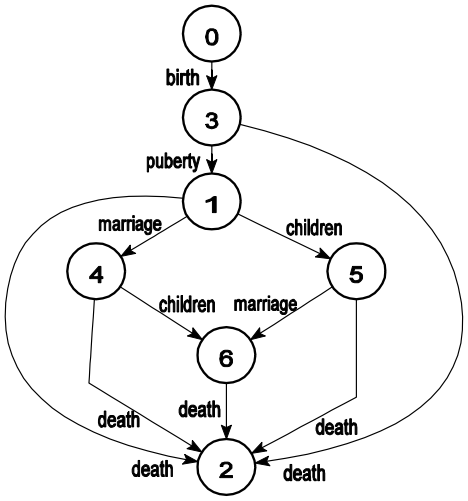


FIG. 3.34: Automate quotient

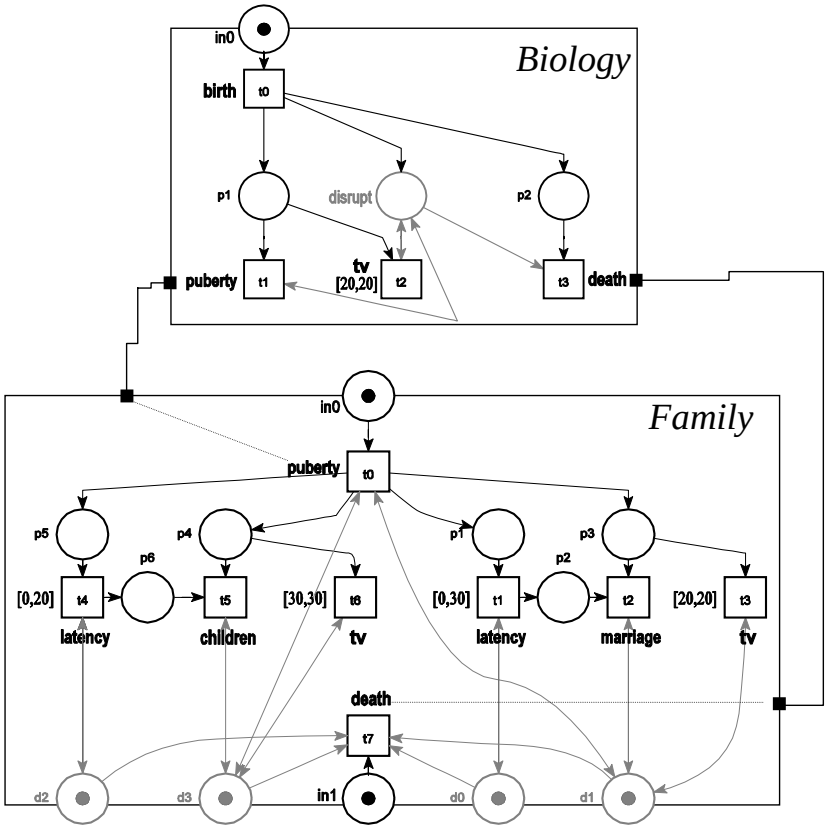


FIG. 3.35: Synchronisation entre les composants $C_{Biology}$ et C_{Family}

3.10.2 Ordonnancement d'un document multimédia interactif

Considérons l'exemple de la Figure 3.36. L'auteur de ce scénario multimédia souhaite présenter trois médias appelés respectivement A, B et C. Les durées de présentation de ces médias sont définies respectivement par les intervalles temporels suivants : [3,6], [3,5] et [3,8]. Ceci signifie que, par exemple, la présentation du média A doit durer au moins 3 secondes et au plus 6 secondes. Ainsi, du point de vue de l'auteur, n'importe quelle durée de présentation du média est acceptable du moment qu'elle appartienne à l'intervalle temporel spécifié. L'auteur exprime en outre les contraintes de synchronisation globales suivantes :

1. Les présentations des médias A et B doivent se terminer simultanément ;
2. Les présentations des médias B et C doivent commencer simultanément ;
3. Le début du scénario multimédia est déterminé par le début de A, et sa terminaison est déterminée par la fin de A et de B, ou par la fin de C.

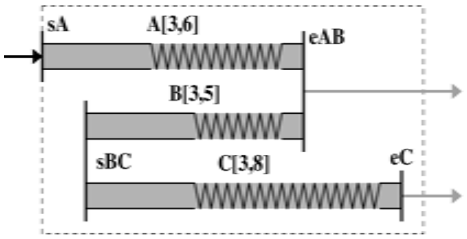


FIG. 3.36: Contraintes temporelles du scénario multimedia

La spécification RT-LOTOS de ce scénario suit :

```
Specification example : exit
behaviour
hide sA, SBC, eAB, eC in
((
   (sA ; delay(3,6)eAB{3} ;exit)
   |||
   (sBC ; delay(3,8)eC{5} ;exit)
 )
|[sBC,eAB]|
(sBC ; delay(3,5)eAB{2} ; exit)
)
|[sA ; latency(inf)sBC ; exit)
endspec
```

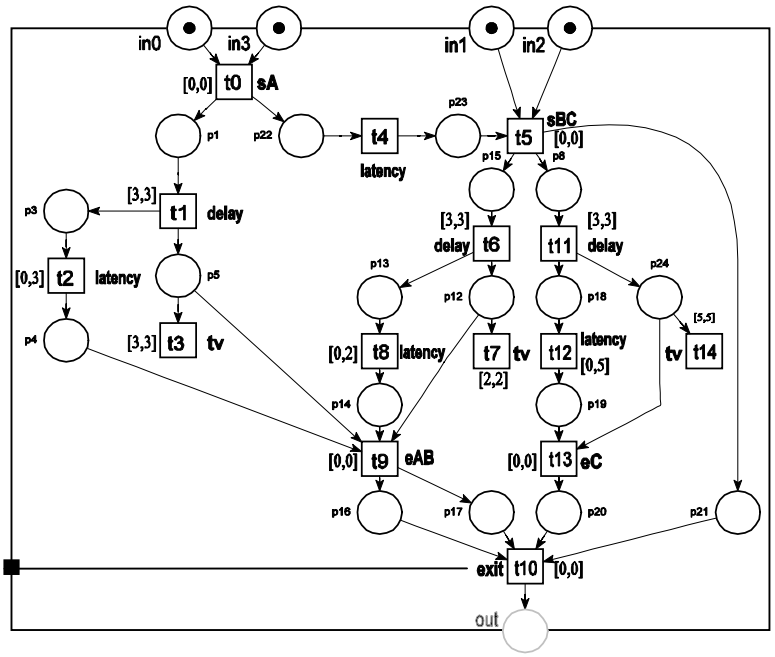


FIG. 3.37: RdPT engendré par *rtl2tpn*

Le RdPT engendré par *rtl2tpn* est décrit dans la Figure 3.37. *Tina* construit pour ce RdPT un graphe de classes de 230 classes et 598 transitions. Ce graphe permet d'analyser la cohérence temporelle du scénario multimédia. Nous disons que ce scénario est potentiellement cohérent s'il existe au moins un chemin commençant par l'action sA (caractérisant le début de la présentation du média A, et par conséquent, le début de la présentation du scénario) et menant à la fin de la présentation du scénario (l'occurrence des actions eAB ou eC). Effectivement, un tel chemin existe dans le graphe de classes, par conséquent, le scénario est potentiellement cohérent.

3.10.3 Benchmarking sur des exemples classiques

Les exemples suivants servent à tester la robustesse des solutions proposées dans ce chapitre face à une situation d'explosion combinatoire. Nous générons l'espace des états des systèmes décrits dans ce qui suit, d'une part avec *rtl* et, d'autre part, avec *rtl2tpn* et *Tina*.

Le dîner des philosophes

Nous utilisons le problème classique de synchronisation entre plusieurs processus connu sous le nom du *dîner des philosophes*. Nous proposons l'extension temporelle suivante du problème : Un certain nombre de philosophes, assis au tour d'une table ronde, passent leur vie à penser et à manger. Chacun d'entre eux a en face de lui un bol de riz. Entre chaque philosophe et son voisin il y a une baguette. Chaque philosophe pense pendant un laps de temps, après la faim finit par le gagner. Pour qu'un philosophe puisse manger, il faudrait qu'il ait en sa possession deux baguettes. Chaque philosophe commence par prendre possession de la baguette à sa gauche. Ensuite il mettra entre 0 et 10 secondes pour s'emparer de la baguette de droite. Une fois en possession des deux baguettes, il peut enfin commencer à manger. L'acte de manger durera entre 10 et 1000 secondes. Après avoir fini de manger, un philosophe pose la baguette gauche, puis la baguette droite dans un délai compris entre 0 et 10 secondes.

```
process Philoso-
pher[think,hungry,b_eat,e_eat, pick_left,put_left,pick_right,put_right] :noexit :=
think ; hungry ; pick_left ;  latency(10)pick_right ; b_eat ; delay(10,1000)e_eat ;put_left ;
latency(10)put_right ;Philosopher[think,hungry,b_eat,e_eat, pick_left,put_left,pick_right,put_right]
endproc
process  Chopstick[pick,put] : noexit :=
pick ; put ; Chopstick[pick,put]
endproc
```

La spécification RT-LOTOS du problème consiste en la synchronisation parallèle de différentes instances du processus *Philosopher* et des instances du processus *Chopstick*.

	rtl2tpn + Tina		*rtl*	
# philosophes	Classes/ Transitions	CPU	Classes/ Transitions	CPU
2	89/150	<1s	64/ 120	<1s
3	653/1536	<1s	530 /1395	3s
4	66 251/220 488	4s	?	>24h
5	876 090/3 668 235	103s	-	-

Tab. 3.3: Résultats de la génération de l'espaces des états

Scheduler de Milner

Le deuxième problème est une extension temporelle du scheduler de Milner [62]. L'intérêt de cet exemple réside dans le fait que les contraintes temporelles introduites ne provoquent pas l'explosion combinatoire du graphe de classes (comme pour l'exemple précédent). Bien au contraire, l'ajout de ces contraintes temporelles limite les comportements possibles, et le graphe de classes résultant est beaucoup plus petit que le graphe de marquages atemporel. Ceci est dû en particulier à la présence de nombreuses actions dont la contrainte temporelle est réduite à un point.

Le scheduler ordonnance N processus appelés "*Cyclers*". Ces derniers coopèrent en contrôlant chacun une tâche. Un *Cycler* C_i exécute indéfiniment le comportement suivant : C_i reçoit un signal g_i indiquant qu'il peut commencer son exécution, dès lors ; après un délai aléatoire compris entre 0 et 10 unités de temps il active la tâche i via l'action a_i.

Après un délai entre 10 et 100 unités de temps il reçoit le signal de terminaison de cette la tâche i qui lui sera notifiée à travers une action b_i. En parallèle, il informe le *Cycler* C_{i+1} qu'il peut commencer son exécution via l'action g_{i+1} et revient à son état initial.

```
process Cycler[gi,ai,bi,gi+1] : noexit :=
gi ; latency(10)ai ;  delay(10,100)
((bi ;  latency(10)gi+1 ;Cycler[gi,ai,bi,gi+1])
[]
(gi+1 ; latency(10)bi ; Cycler[gi,ai,bi,gi+1]))
endproc
Specification Scheduler : noexit
behaviour
hide g1..gn,a1..an,b1..bn in
(cycler[g1,a1,b1,g2]
|[g1,g2]|
(
...
Cycler[gi,ai,bi,gi+1]
|[gi+1]|
...
(cycler[gn, an, bn, g1]  |||  g1 ; stop) ...)
```

	rtl2tpn + tina		*rtl*	
# cyclers	Classes/Transitions	CPU	Classes Transitions	CPU
2	35/50	<1s	31/57	<1s
3	81/141	<1s	58/115	<1s
4	127/232	<1s	77/153	<1s
5	161/316	<1s	96/191	3s
6	219/414	<1s	115/229	18s
7	232/432	<1s	134/267	97s
8	311/596	<1s	153/305	490s
9	357/687	<1s	172/343	2363s
10	461/911	<1s	?	>24h

TAB. 3.4: Résultats de la génération de l'espace des états

Les résultats de la génération de l'espace des états pour ces deux exemples confirment que l'approche proposée dans ce chapitre et qui consiste à passer par le modèle des RdPT et l'utilisation de *Tina* est plus efficace que l'approche basée sur *rtl*.

3.11 Conclusion du chapitre

Nous avons décrit dans ce chapitre une approche transformationnelle efficace pour la vérification de spécifications RT-LOTOS. Notre but est d'utiliser le graphe de classes d'un RdPT pour représenter et analyser le comportement d'un système temps-réel décrit en RT-LOTOS. Après avoir examiné de près les sémantiques de ces deux modèles temporels, nous avons défini formellement le concept de composant RdPT ainsi qu'un ensemble de patterns de traduction. Ces derniers ont été implémentés dans *rtl2tpn*, un prototype qui prend en entrée une spécification RT-LOTOS et génère en sortie un RdPT dans le format accepté par *Tina*. Il est ainsi devenu possible d'intégrer le puissant outil *Tina* dans notre plateforme de validation. Les résultats expérimentaux confirment l'avantage de passer par les RdPT comme un modèle intermédiaire. Aussi, ce travail de traduction confère à RT-LOTOS une sémantique formelle en termes de RdPT. Ainsi, notre approche peut être vue comme une contribution pour une spécification *top-down* des systèmes temps-réel. En effet, au plus haut niveau nous utilisons RT-LOTOS pour décrire l'architecture du système ainsi que la communication entre les modules constituants. A un niveau plus bas, nous obtenons un RdPT qui décrit le comportement opérationnel du système.

Notons aussi que le passage par le modèle des RdPT a clarifié l'usage de certains opérateurs RT-LOTOS comme la *latence*.

Pour finir, les patterns de traduction présentés dans ce chapitre sont facilement adaptables pour d'autres extensions temporelles de LOTOS, en particulier ET-LOTOS [56]. RT-LOTOS se démarquant essentiellement de ces extensions par son opérateur de latence.

4 Extension de RT-LOTOS pour la vérification des systèmes préemptifs

La modélisation des systèmes complexes repose sur un triptyque (spécification des propriétés, définition d'une architecture, expression des comportements) que l'on peut destiner en particulier aux systèmes *temps-réel* en prenant explicitement en compte le temps et en gérant les interruptions et mécanismes de suspension/reprise (*suspend/resume*). Si nombre de techniques de description formelles permettent de décrire des mécanismes temporels et d'exprimer des contraintes de temps, plus rares sont celles qui disposent d'un mécanisme de suspension avec sauvegarde du contexte en vue d'une reprise. Ce constat vaut en particulier pour l'algèbre de processus temporisée RT-LOTOS et justifie la proposition formulée dans ce chapitre d'étendre RT-LOTOS avec un opérateur de suspension et reprise appelé *suspend/resume*.

Dans notre proposition d'une extension à RT-LOTOS, nous ne nous limitons pas aux aspects syntaxiques. Notre but est la vérification d'une spécification écrite en RT-LOTOS étendu et ce après traduction de la dite spécification vers un réseau de Petri temporel à chronomètres (RdPTC) [14]. Ces derniers étendent les RdPT utilisés au chapitre précédent, comme formalisme cible pour la traduction de la partie "contrôle" de RT-LOTOS.

Dans ce chapitre, nous proposons une traduction vers une classe simple de réseaux de Petri à chronomètres, qui permettra via la méthode des "classes d'états", la représentation exacte du comportement spécifié en RT-LOTOS étendu.

4.1 Motivations

En matière de description de systèmes *temps-réel*, il est important de pouvoir décrire des situations où un système peut voir son exécution interrompue puis reprendre sans perte du contexte temporel qui caractérisait l'état du système au moment de la suspension. Un tel mécanisme de suspension/reprise (ou *suspend/resume* en anglais) est-il réalisable en RT-LOTOS ? Pour répondre à cette question, prenons le cas d'un modèle simplifié d'une machine à laver le linge. Ce petit système est composé de deux processus nommés respectivement *Machine* et *Cover*. Ils modélisent le contrôleur de la machine et le couvercle.

```
process Machine[start, begin_wash, end_wash] : exit :=
start ; begin_wash ; end_wash ; exit
endproc

process Cover[open, close] : exit :=
open ; close ; Cover[open, close]
endproc
```

Le comportement global que l'on souhaite obtenir est le suivant : tout au long de son exécution, le processus *Machine* peut être suspendu par le processus *Cover* et reprendre après l'exécution complète du processus *Cover* (qui exécute pour sa part une séquence d'actions *open ; close*).

L'opérateur *disrupt* [> de RT-LOTOS n'est pas approprié pour décrire ce type de comportement. Voyons-le sur le même exemple :

```
Machine[start, begin_wash, end_wash]
[>
Cover[open, close]
```

Le comportement RT-LOTOS ci-dessus permet effectivement de suspendre le processus *Machine* mais en aucun cas d'assurer ultérieurement la reprise de ce même processus. Dans la description RT-LOTOS qui précède, la suspension de *Machine* sera suivie d'une exécution de la séquence infinie d'actions *'open ; close'*. Un recours à l'utilisation de l'opérateur de composition parallèle ||| ne résoudrait pour autant pas le problème. Le comportement obtenu serait celui d'un entrelacement entre les actions des processus *Machine* et *Cover,* ce qui ne garantit en aucune manière que *Cover* termine son exécution avant que *Machine* ne reprenne la sienne. Une autre solution serait d'adopter un style *orienté état*s [86], en utilisant les opérateurs de choix [] et de composition séquentielle ≫ pour définir explicitement tous les points de suspension dans le processus *Machine* (rappelons que les actions RT-LOTOS sont atomiques). La description *orientée états* de la machine à laver prendrait alors la forme suivante :

```
process Machine2[start, begin_wash, end_wash, open, close] : exit :=
((start ; exit)
[](Cover[open, close] >> (start ; exit)))
>>((begin_wash ; exit)
[](Cover[open, close] >> (begin_wash ; exit)))
>>((end_wash ; exit)
[](Cover[open, close] >> (end_wash ; exit)))
endproc
```

Le processus *Cover* est alors redéfini pour supprimer la récursivité. Cela vise à éviter les séquences d'actions infinies dans le processus *Cover* et à permettre la reprise effective du processus *Machine*.

```
process Cover [open, close] : exit :=
open ; close ; exit
endproc
```

Bien que le processus *Machine* soit réduit à une simple séquence d'actions, la contorsion introduite dans cette description *orientée états* rend cette dernière peu lisible et nuit à sa compositionabilité. De plus, cette réécriture de la description RT-LOTOS de la machine à laver ne permet que trois exécutions de la séquence d'actions *'open ; close'*.

Pour compléter cette discussion, considérons maintenant une version temporisée de cette machine à laver.

```
hide begin_wash, end_wash in
process Machine[start, begin_wash, end_wash] : exit :=
start ; delay(1,2)begin_wash ; delay(40,70)end_wash ; exit
endproc
```

Le comportement de cette version temporisée de la machine à laver s'interprète ainsi : après l'occurrence de l'action *start,* l'action *begin_ wash* est retardée de 1 à 2 unités de temps. Le lavage lui-même prend entre 40 et 70 unités de temps. Les actions *begin_ wash* et *end_ wash* sont intériorisées et deviennent de ce fait urgentes. Du fait de l'utilisation d'un modèle de

temps dense, il est matériellement impossible de décrire tous les points de suspension de cette nouvelle description RT-LOTOS. De plus, si l'on essaie de reproduire une description semblable à celle de *Machine2*, il ne serait plus possible de préserver les contraintes temporelles du processus Machine d'origine.

Considérons alors le comportement RT-LOTOS qui suit, exprimé dans un style orienté états :

```
(delay(1,2)begin_wash ; exit)
[](Cover[open, close] >> delay(1,2)begin_wash ; exit)
```

L'interaction avec l'environnement résout le choix affiché par le comportement RT-LOTOS ci-dessus. Le processus *Cover* est en effet exécuté si l'environnement offre l'action *open* à moins que l'action *begin_ wash* ne soit exécutée après un délai compris entre 1 et 2 unités de temps. Tant que le choix n'est pas fait, les deux branches du comportement vieillissent de manière similaire. Supposons maintenant que le processus *Cover* offre l'action *open* après une unité de temps, ce qui donne la séquence d'exécution suivante :

```
(delay(1,2)begin_wash...) [] (Cover[open, close] >> ...)
```
$\xrightarrow{1}$
```
(delay(0,1)begin_wash...) [] (Cover[open, close] >> ...)
```
$\xrightarrow{Cover[]}$
```
(delay(1,2)begin_wash ; exit)
```

Ainsi, après l'écoulement d'une unité de temps, l'action *begin_ wash* doit être sensibilisée au plus après une unité de temps. Le comportement résultant est tel qu'après la terminaison du processus *Cover*, l'action *begin_ wash* doit attendre entre une et deux unités de temps pour être sensibilisée (alors que le temps a déjà progressé d'une unité de temps).

On pourrait à nouveau modifier le comportement du processus *Machine2* pour restaurer son contexte temporel. Une variable t pourrait être introduite pour capturer le temps écoulé avant qu'une action *open* ne soit offerte par le processus *Cover*. On pourrait utiliser pour cela l'opérateur @ (présenté dans [57]) et écrire : *open@t*. Cette valeur de temps doit être soustraite de la contrainte temporelle de *Machine2*. Une soustraction directe n'est pas la bonne solution ; en effet, le risque existe d'aboutir à des domaines de temps négatifs. Pour obtenir un comportement global cohérent, il faudrait ajouter des gardes sur les différents intervalles valides pour t. Cet ensemble de modifications alourdit considérablement la spécification et nuit tant à sa lisibilité qu'à sa compositionnalité. Aussi, nous allons exploiter maintenant la piste de l'introduction dans RT-LOTOS d'un véritable opérateur de suspension/reprise.

4.2 Syntaxe et sémantique de l'opérateur *suspend/resume*

L'opérateur de composition que nous ajoutons à RT-LOTOS pour permettre de suspendre un processus et de reprendre ultérieurement l'exécution de celui-ci sera représenté par le symbole $[g \gg$ (où g représente l'action utilisée pour la reprise), combinaison des opérateurs *disrupt* $[>$ et de composition séquentielle $\gg$. L'extension apportée au *disrupt* vise à permettre la reprise. Ainsi, dans une description RT-LOTOS contenant un comportement Q susceptible de suspendre un processus P, il sera maintenant possible d'assurer la reprise de l'exécution de P, ce qui n'était pas possible avec l'opérateur *disrupt* qui déclenche une interruption définitive.

En définissant le nouvel opérateur *suspend/resume* $[g \gg$ comme une généralisation de l'opérateur *disrupt* $[>$, nous produisons de fait un opérateur binaire. Au niveau sémantique, nous

limitons ainsi à un le nombre de processus qui peuvent suspendre un même comportement. Notons bien que le fait d'avoir un opérateur binaire n'empêche nullement de représenter la suspension et la reprise de plusieurs processus concurrents (un exemple est donnée dans la section 4.4.4) .

Le comportement $P[g \gg Q$ est défini à travers une sémantique opérationnelle à la Plotkin [74] matérialisée par des règles *SOS*. Pour formaliser cette sémantique nous utiliserons les notations suivantes : $\mathcal{G}_\mathcal{P}$ représente l'ensemble des actions observables et cachées du processus P. Nous introduisons au passage l'opérateur sémantique $\ll g]$ (inspiré de [56]) et utilisons $P \ll g]Q$ pour exprimer qu'un processus P a été suspendu par Q. Cet opérateur apparaît uniquement au niveau sémantique. Il n'est pas utilisable dans une description RT-LOTOS.

$$1)\quad \frac{P \xrightarrow{a} P'}{P[g \gg Q \xrightarrow{a} P'[g \gg Q} \quad a \in G_P \backslash \{exit\}$$

$$2)\quad \frac{P \xrightarrow{exit} P'}{P[g \gg Q \xrightarrow{exit} P'}$$

$$3)\quad \frac{Q \xrightarrow{a} Q'}{P[g \gg Q \xrightarrow{a} P \ll g]Q'}$$

$$4)\quad \frac{Q \xrightarrow{a} Q'}{P \ll g]Q \xrightarrow{a} P \ll g]Q'} \quad a \in G_Q \backslash \{g\}$$

$$5)\quad \frac{Q \xrightarrow{t} Q'}{P \ll g]Q \xrightarrow{t} P \ll g]Q'}$$

$$6)\quad \frac{Q \xrightarrow{g} Q'}{P \ll g]Q \xrightarrow{g} P[g \gg Q'}$$

$$7)\quad \frac{P \xrightarrow{t} P', Q \xrightarrow{t} Q'}{P[g \gg Q \xrightarrow{t} P'[g \gg Q'}$$

La règle 1 définit l'exécution normale de P. Elle stipule qu'à la suite de toute action a de P, Q a encore une chance de suspendre P (Q restant actif).

La règle 2 stipule que si P a terminé son exécution avec succès alors Q ne peut plus être exécuté.

La règle 3 définit le transfert de contrôle de P vers Q, à l'occurrence de l'une des premières actions de Q.

Les règles 4 et 5 stipulent que Q est le seul comportement actif. Le processus suspendu P ne peut réaliser aucune action (règle 4). De plus, ce dernier ne vieillit pas (règle 5). Nous interprétons ainsi le passage du temps dans un processus comme un temps d'exécution (par exemple le délai entre les actions *begin_ wash* et *end_ wash* du processus *Machine* de la section 6.1) . Ce temps d'exécution doit être suspendu quand le processus est lui même suspendu (règle 5). De ce fait, un processus suspendu ne vieillit plus. Il peut dans certains cas s'avérer utile de considérer un temps global et par conséquent un processus qui bien que suspendu, vieillit. Implanter ce type de solution requiert des opérateurs *delay* et *latency* non préemptibles. La mise en œuvre d'une telle solution amènerait à des comportements non désirables, par exemple la sensibilisation d'une action urgente à l'intérieur d'un processus suspendu.

La règle 6 permet la reprise de P à l'exécution de l'action g de Q. Au lieu d'écrire $P \ll g]Q$, nous aurions pu écrire $P \ll g](Q, Q^0)$, où l'opérande Q^0 est utilisé pour sauvegarder l'état initial de Q. On écrirait alors $P \ll g](Q, Q^0) \xrightarrow{g} P[g \gg (Q^0, Q^0)$.

Enfin, la règle 7 stipule que du point de vue du temps, l'opérande $[g \gg$ se comporte comme le *disrupt* $[>$. En d'autres termes, le vieillissement de P induit celui de Q.

L'on pourrait croire que nous avons oublié une règle, pour définir le comportement de $P \ll g]Q$ quand Q termine son exécution avec succès (dans le but d'éliminer le comportement suspendu P). La terminaison de Q est prise en compte dans la règle 4, mais le comportement suspendu P apparaît toujours. Ceci ne pose pas de problème étant donné que le processus Q n'exécute aucune action après sa terminaison, et ne permettra donc pas la reprise de P. De plus, si cette terminaison est capturée par une composition séquentielle $\gg$, le comportement $P \ll g]Q$ dans son entier n'apparaît plus.

4.2.1 Propriétés de l'opérateur *suspend/resume*

Bisimulation forte et Congruence : Une des propriétés désirée est que la relation d'équivalence forte [62] soit une congruence. En effet, pour des besoins de vérification, il est important de pouvoir remplacer une partie de la spécification par une autre (qui lui soit fortement bisimilaire), sans pour autant changer la sémantique de toute la spécification (i.e. que la spécification obtenue soit fortement bisimilaire à l'originale).

C'est pourquoi nous montrons dans ce qui suit que la relation d'équivalence forte est aussi une congruence.

Définition 4.1 Soit $\mathcal{RTL}$ l'ensemble des expressions RT-LOTOS. Nous disons que deux expressions RT-LOTOS $P, Q \in \mathcal{RTL}$ sont fortement bisimilaires, et nous écrivons $P \sim Q$, ssi leurs systèmes de transitions étiquetés correspondants, $\mathcal{LTS}_\mathcal{P}$ et $\mathcal{LTS}_\mathcal{Q}$ sont fortement bisimilaires et nous écrivons $\mathcal{LTS}_\mathcal{P} \sim \mathcal{LTS}_\mathcal{Q}$ (cf. Définition 2.1).

Soit $\mathcal{RTL}^*$ l'ensemble des expressions RT-LOTOS étendu par l'opérateur *suspend/resume*.
Proposition 4.2 Si $P_1 \sim P_2$ avec $P_1, P_2 \in \mathcal{RTL}^*$ alors :
$P_1[g \gg Q \sim P_2[g \gg Q$ et $Q[g \gg P_1 \sim Q[g \gg P_2$ avec $Q \in \mathcal{RTL}^*$.
L'opérateur *suspend/resume* est congruent pour la bisimulation forte.

Preuve : Nous prouvons que les deux relations suivantes sont des bisimulations fortes :

$$R_1 = \{(P_1[g \gg Q, P_2[g \gg Q) \mid P_1 \sim P_2\} \cup \{(P_1 \ll g]Q, P_2 \ll g]Q) \mid P_1 \sim P_2\} \cup \sim$$
$$R_2 = \{(Q[g \gg P_1, Q[g \gg P_2) \mid P_1 \sim P_2\} \cup \{(Q \ll g]P_1, Q \ll g]P_2) \mid P_1 \sim P_2\} \cup \sim$$

Considérons d'abord le cas de la relation R_1.

1. Soit $(P_1[g \gg Q, P_2[g \gg Q) \in R_1$; nous devons montrer que tous ses comportements dérivés restent dans R_1, i.e., $P_1[g \gg Q \xrightarrow{l} P_1' \Rightarrow P_2[g \gg Q \xrightarrow{l} P_2'$ avec $(P_1', P_2') \in R_1$ et réciproquement.

– Si $P_1 \xrightarrow{l} P_1'$ alors il existe P_2' tel que $P_2 \xrightarrow{l} P_2'$ et $P_2 \sim P_2'$ puisque $P_2 \sim P_2'$.
Nous distinguons trois cas, selon que le comportement dérivé résulte de l'application de :

1. La règle 1, $l = a$, $a \in G_P \backslash \{exit\}$, $P_1[g \gg Q \xrightarrow{a} P_1'[g \gg Q$, $P_2[g \gg Q \xrightarrow{a} P_2'[g \gg Q$ et $(P_1'[g \gg Q, P_2'[g \gg Q) \in R_1$ puisque $P_1' \sim P_2'$

a) La règle 2, $l = exit$, $P_1[g \gg Q \xrightarrow{exit} P_1^{'}$, $P_2[g \gg Q \xrightarrow{exit} P_2^{'}$ et $(P_1^{'}, P_2^{'}) \in R_1$ puisque $P_1^{'} \sim P_2^{'}$

b) La règle 7, $l = t$, $P_1[g \gg Q \xrightarrow{t} P_1'[g \gg Q'$, $P_2[g \gg Q \xrightarrow{t} P_2'[g \gg Q'$ et $(P_1'[g \gg Q', P_2'[g \gg Q') \in R_1$ puisque $P_1^{'} \sim P_2^{'}$

– Si $P_1 \xrightarrow{l} P_1'$, ce cas est similaire au cas précédent.
– Si $Q \xrightarrow{l} Q'$, alors si le comportement dérivé Q' résulte de l'application de :

1. La règle 3 : $l = a$, $P_1[g \gg Q \xrightarrow{a} P_1 \ll g]Q'$, $P_2[g \gg Q \xrightarrow{a} P_2 \ll g]Q'$ et $(P_1 \ll g]Q', P_2 \ll g]Q') \in R_1$puisque $P_1 \sim P_2$.
2. La règle 7, le cas a déjà été traité plus haut.

1. On procède de la même manière pour $(P_1 \ll g]Q, P_2 \ll g]Q) \in R_1$

– Si $P_1 \ll g]Q \xrightarrow{l}$, alors si le comportement dérivé résulte de l'application de :

1. La règle 4, $l = a$, $a \in G_Q \backslash \{g\}$, $P_1 \ll g]Q \xrightarrow{a} P_1 \ll g]Q'$, $P_2 \ll g]Q \xrightarrow{a} P_2 \ll g]Q'$ et $(P_1 \ll g]Q', P_2 \ll g]Q') \in R_1$puisque $P_1 \sim P_2$.
2. La règle 6 : $P_1 \ll g]Q \xrightarrow{g} P_1[g \gg Q'$, $P_2 \ll g]Q \xrightarrow{g} P_2[g \gg Q'$ et $(P_1[g \gg Q', P_2[g \gg Q') \in R_1$puisque $P_1 \sim P_2$.
3. La règle 5, $l = t$, $P_1 \ll g]Q \xrightarrow{t} P_1 \ll g]Q'$,$P_2 \ll g]Q \xrightarrow{t} P_2 \ll g]Q'$ et $(P_1 \ll g]Q', P_2 \ll g]Q') \in R_1$puisque $P_1 \sim P_2$.

– Si $P_2 \ll g]Q \xrightarrow{l}$, ce cas est similaire au cas précédent.

Pour R_2 on procède de la même manière.

4.3 Réseaux de Petri temporels à chronomètres

Parmi les techniques de modélisation de systèmes temps réel qui permettent de modéliser les suspensions et reprises nous pouvons citer les automates à chronomètres ou *stopwatch automata* (SWA) [24] en anglais. Ces derniers sont une sous-classe des automates hybrides linéaires (LHA) [5].[24] a prouvé que les SWA avec délais cachés sont aussi expressifs que les LHA. Le problème d'accessibilité pour cette classe d'automates est indécidable [24]. Aucune classe décidable et suffisamment expressive des SWA n'ayant pu être identifiée, [24] propose un algorithme de sur-approximation pour le *model checking* des automates SWA. L'approche proposée repose sur un encodage basé sur les *DBM*[1] des classes d'états. Néanmoins les auteurs admettent que cette sur-approximation est trop grossière. Dans la famille des techniques de modélisation développées à partir des RdPT et supportant un mécanisme de suspension/reprise, nous pouvons citer les *scheduling*-TPN [78] qui ajoutent des ressources et des priorités aux RdPT. Les Inhibitor Hyperarc TPN (IHTPN) [79] ajoutent quant à eux des arcs inhibiteurs pour contrôler la progression des transitions. L'on trouve en [14] la preuve du fait que le problème de l'accessibilité de l'espace d'états de ces extensions est indécidable même pour des réseaux bornés. Des méthodes efficaces de sur-approximation d'états sont disponibles pour toutes ces extensions des RdPT. Les abstractions proposées capturent tous les états accessibles, parfois davantage. De telles sur-approximations fournissent des conditions suffisantes pour les propriétés de sûreté. Néanmoins, comme pour les SWA ; la sur-approximation reste trop grossière.

[1] Difference Bound Matrices

Dans ce chapitre nous considérons le modèle des réseaux de Petri temporels à chronomètres (ou RdPTC) [14] . Ces derniers étendent les RdPT avec des arcs activateurs qui permettent de contrôler l'évolution temporelle des transitions.

Soit I^+ l'ensemble non vide des intervalles réels avec bornes rationnelles non négatives. Pour $i \in I^+, \downarrow i$ représente sa borne inférieure, et $\uparrow i$ sa borne supérieure (si elle existe) ou ∞ (sinon). Pour tout $\theta \in \mathcal{R}^+$, $i \dot{-} \theta$ représente l'intervalle $\{x - \theta \mid x \in i \wedge x \geq \theta\}$.

Définition 4.3 : Un réseau de Petri temporel à chronomètres (RdPTC) est un *n-uplet* $\langle P, T, Pre, Post, Sw, m_0, I_s \rangle$, tel que $\langle P, T, Pre, Post, m_0, I_s \rangle$ est un réseau de Petri temporel est $Sw : T \rightarrow P \rightarrow \mathbb{N}$ est une fonction appelée fonction d'activation.

La fonction d'activation Sw associe un entier à chaque $(t, p) \in T \times P$. Les valeurs supérieures à 0 sont représentées par des arcs particuliers appelés arcs activateurs, orientés par un *losange*. Notons que ces arcs ne transmettent pas de jetons. Comme il est d'usage, une transition t est sensibilisée au marquage m ssi $m \geq Sw(t)$, sinon t est dite *suspendue*.

La Figure 4.1 décrit un RdPTC. L'arc reliant la place p_0 à la transition t_2 est un arc activateur. Le tir de la transition t_0 gèlera l'évolution temporelle de t_2. Cette dernière sera tirable quand son temps total de sensibilisation aura atteint 2 unités de temps.

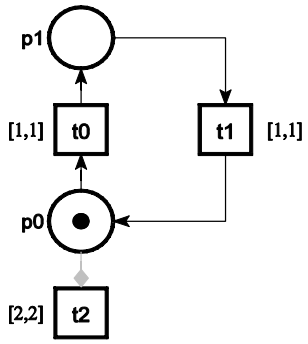


FIG. 4.1: Exemple de RdPTC

Une transition t est sensibilisée par le marquage m ssi $m \geq Pre(t)$. De plus, une transition sensibilisée par m est *active* ssi $m \geq Sw(t)$, sinon elle est dite *suspendue*.

Définition 4.4 Un état d'un RdPTC est un couple $s = (m, I)$ tel que m est un marquage et I est une fonction qui associe un intervalle temporel dans I^+ à chaque transition sensibilisée par m. Nous noterons $(m, I) \xrightarrow{t@\theta} (m', I')$ ssi $\theta \in \mathcal{R}^+$ et :

1. $m \geq Pre(t) \wedge m \geq Sw(t) \wedge \theta \geq \downarrow I(t) \wedge (\forall k \in T)(m \geq Pre(k) \Rightarrow \theta \leq \uparrow I(k))$
2. $m' = m - Pre(t) + Post(t)$
3. $(\forall k \in T)(m' \geq Pre(k) \Rightarrow I'(k) = si\ k \neq t \wedge m - Pre(t) \geq Pre(k)\ alors\ si\ m \geq Sw(k)\ alors\ I(k) \dot{-} \theta\ sinon\ I(k)\ sinon\ I_s(k))$

Nous avons $s \xrightarrow{t@\theta} s'$ si le tir de t à partir de s à la date relative θ conduit à s'. (1) assure que t soit tirée dans son intervalle temporel à moins qu'elle ne soit désensibilisée par le tir d'une autre transition, et qu'elle est active.

(2) est la règle de transformation de marquage classique.

(3) signifie que les transitions nouvellement sensibilisées sont associées à leurs intervalles de tir statiques alors que les transitions persistantes (celles restant sensibilisées lors du tir) ont leur intervalle inchangé si elles étaient suspendues, ou décalé de θ et tronqué à zéro si elles étaient actives. Les transitions qui restent sensibilisées lors de leur propre tir sont considérées comme nouvellement sensibilisées.

Le graphe d'états d'un RdPTC est l'ensemble des états accessibles depuis son état initial (m_0, I_0).

4.3.1 Classes d'états d'un RdPTC

Comme pour les RdPT, le nombre d'états d'un RdPTC est potentiellement infini. Les abstractions fournies par les constructions de graphes de classes des RdPT ont été adaptées dans [14] pour les RdPTC.

[14] prouve que le problème de l'accessibilité pour les RdPTC peut être réduit à celui de l'arrêt d'une machine à deux compteurs. Il s'ensuit que les problèmes d'accessibilité d'un marquage ou d'un état sont indécidables pour les RdPTC. Le caractère borné du RdPTC n'implique pas le caractère fini des graphes de classes. Pour assurer la terminaison de la méthode de classes, [14] propose une technique basée sur une quantification des polyèdres représentant l'information temporelle dans les classes. En ajustant un paramètre, le comportement exact du RdPTC peut être approché aussi étroitement que souhaité. Ainsi, lorsqu'il termine, l'algorithme proposé dans [14] produit une abstraction qui préserve les marquages et les propriétés *LTL* du réseau.

Cette technique produit des approximations plus précises que les techniques citées au début de cette section, et avec une précision ajustable [14]. Cependant, le recours à cette méthode d'approximation n'est pas nécessaire dans plusieurs cas pratiques. Les expérimentations présentées dans ce chapitre en sont des exemples.

4.4 Traduction en réseaux de Petri temporels à chronomètres

4.4.1 Composant à chronomètres

Un composant à chronomètres, appelé par la suite *Sw-composant* de par son nom anglais, ajoute la fonctionnalité *chronomètre* au concept de composant que nous avions introduit dans le Chapitre 2.

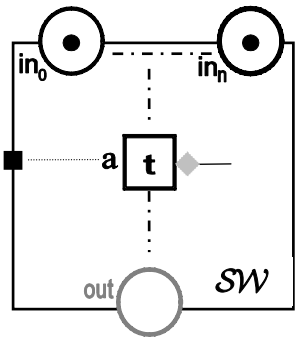


Fig. 4.2: Composant à chronomètres

Un *Sw-composant* est un tuple $C = \langle \sum, lab, I, O \rangle$ où :

$\sum = \langle P, T, Pre, Post, Sw, m_0, IS \rangle$ est un RdPTC. Les éléments (lab, I, O) sont identiques à ceux décrits en Section 3.3 pour les composants RdPT, de même que les invariants H1-H5 que l'on associe également aux composants à chronomètres.

4.4.2 Schéma de traduction pour l'opérateur *suspend/resume*

La Figure 4.3 décrit le comportement où le composant C_P peut être suspendu par C_Q et reprendre ultérieurement au point où il avait été interrompu. Pour modéliser ce comportement, nous introduisons une place partagée '*SR*', connectée à toutes les transitions de C_P via des arcs activateurs de poids 1, à l'exception de la transition *exit* (s'il y en a une), étant donné que le composant C_P ne peut plus être suspendu après sa terminaison. Ainsi, si la place *SR* est non marquée alors l'exécution du composant C_P est suspendue.

$$\forall t \in T_P \; lab(t) \neq exit \; \Rightarrow Sw(SR, t) = 1$$

Le composant C_P est suspendu à l'occurrence de la première action[2] de C_Q (règle 3), car le tir de cette dernière vide la place *SR* de son jeton. Après l'occurrence de l'action g dans le composant C_Q, ce dernier réactive le composant C_P en restituant le jeton dans la place *SR*. Cette dernière est donc une place d'entrée à toutes les premières actions du composant C_Q (cf. Définition 3.4.1 pour l'ensemble des premières actions : $\mathcal{FA}(C_Q)$). De plus, après la terminaison de C_P, ce dernier ne peut plus être suspendu, et C_Q doit être désactivé (règle 2). Ainsi, *SR* doit être une place d'entrée pour toutes les transitions *exit* du composant C_P.

$$SR^{\bullet} = \mathcal{FA}(\mathrm{C_Q}) \cup \bigcup_{\mathrm{t \in T_P \wedge lab(t) = exit}}$$

$$^{\bullet}SR = \{g\}$$

Pour finir, les interfaces de sorties des deux composants C_P et C_Q sont fusionnées.

$$O_{P[g \gg Q} = \{out\} \; si \; (O_P \neq \emptyset \, O_Q \neq \emptyset)$$

Notons que si l'alphabet du composant C_Q ne contient pas l'action spéciale g, alors l'opérateur se comporte exactement comme le *disrupt* $[>$.

[2] Nous complétons la définition de l'ensemble des premières actions donnée en Section 3.4.1 avec la règle suivante : $\mathcal{FA}(C_{P[g \gg Q}) = \mathcal{FA}(\mathrm{C_P}) \cup \mathcal{FA}(\mathrm{C_Q})$

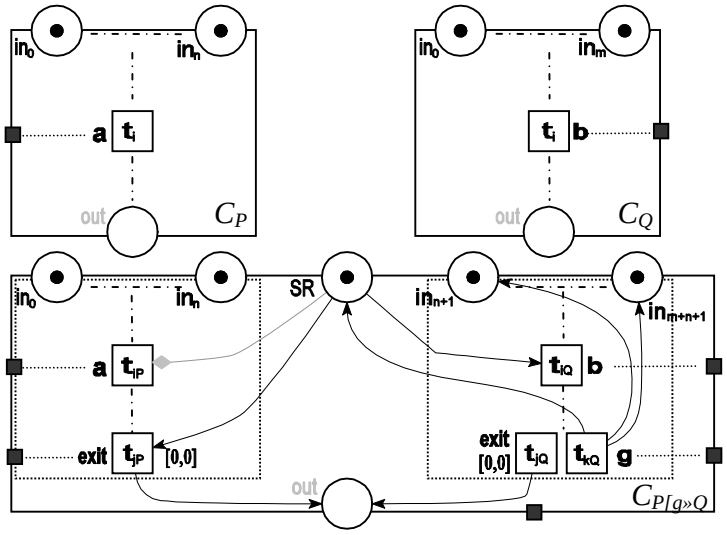


FIG. 4.3: Schéma de traduction du suspend/resume

La définition formelle de $C_{P[g\gg Q}$ suit :

$$C_{P[g\gg Q} = \left\langle \sum_{P[g\gg Q} , Act_P \cup Act_Q, Lab_{P[g\gg Q}, I_P \cup I_Q \cup \{SR\}, O_{P[g\gg Q} \right\rangle$$

avec :

$$\sum_{P[g\gg Q} = \langle P_{P[g\gg Q}, T_{P[g\gg Q}, Pre_{P[g\gg Q}, Post_{P[g\gg Q}, Sw, m_0, IS_{P[g\gg Q}\rangle$$

$$P_{P[g\gg Q} = P_P \cup P_Q \cup \{SR\} \cup \{out\} \backslash O_Q \backslash O_P$$

$$T_{P[g\gg Q} = T_P \cup T_Q$$

$$Pre_{P[g\gg Q} = Pre_P \cup Pre_Q \cup \bigcup_{t\in\mathcal{FA}(\mathrm{Q})} (SR,t) \cup \bigcup_{t\in T_P \wedge lab(t)=exit} (SR,t)$$

$$Post_{P[g\gg Q} = Post_P \cup Post_Q \cup \bigcup_{t\in T_Q \wedge lab(t)=g} (t,SR)$$

$$Sw_{P[g\gg Q} = Sw_P \cup Sw_Q \cup \bigcup_{t\in T_P \wedge lab(t)\neq exit} (SR,t)$$

$$IS_{P[g\gg Q} = IS_P \cup IS_Q$$

De plus, un ensemble d'arcs est introduit dans le composant $C_{P[g\gg Q}$ afin de connecter les places de l'interface d'entrée I_Q du composant C_Q avec la transition *exit* du composant C_P.

$$\bigcup_{t\in T_P \wedge lab(t)=exit \wedge p\in I_Q} (p,t)$$

Le rôle de ces arcs est de purger le composant C_Q (des jetons restants) après la terminaison de l'exécution de C_P. De la même manière, si le composant C_Q termine son exécution avec

succès, alors le composant C_P est purgé. Pour des raisons de lisibilité nous ne représentons pas les arcs dédiés à la purge dans le pattern de traduction de la Figure 4.3. Le principe est le même que celui utilisé pour le pattern du *choix* et repose sur une duplication de transitions.

4.4.3 Consistance de la traduction

Dans cette section, nous montrons que la traduction en RdPTC que nous venons de décrire est consistante, c'est-à-dire que le comportement du RdPTC obtenu est fortement équivalent à celui des processus décrits en RT-LOTOS étendu. Pour établir une équivalence forte entre le comportement $P[g \gg Q$ et le composant $C_{P[g \gg Q}$, nous suivons le même schéma que pour la preuve présentée dans la section 3.9.

Preuve :

$\underrightarrow{\mathbb{R}}$: un écoulement du temps dans $P[g \gg Q$ est soit :

1. Un écoulement du temps dans les deux processus P et Q (règle 7 de la sémantique opérationnelle). Si un écoulement du temps est possible dans les deux processus, alors la première action de Q (resp C_Q) est sensibilisée mais son occurrence n'a pas encore eu lieu. Par induction, cet écoulement du temps est aussi possible dans C_P et dans C_Q. La place SR (qui du reste est la seule place introduite par le pattern de traduction) est une place d'entrée pour les premières actions de C_Q. Par construction, la place SR est marquée (C_P n'est pas suspendu). Ainsi, cet écoulement du temps est possible dans les deux composants C_P et C_Q. Par conséquent cet écoulement du temps est aussi possible dans le composant $C_{P[g \gg Q}$.
2. Un écoulement du temps dans le processus Q (règle 5 de la sémantique opérationnelle). Le processus P a été suspendu à l'occurrence de la première action de Q. Par induction cet écoulement du temps est possible dans C_Q. La place SR n'interfère pas avec l'évolution de C_Q. Ainsi, dans $C_{P[g \gg Q}$ cet écoulement du temps est possible seulement dans C_Q.

$\underleftarrow{\mathbb{R}}$: similaire à $\underrightarrow{\mathbb{R}}$.

$\underrightarrow{a}$: une occurrence de l'action a dans $P[g \gg Q$ est soit :

1. L'occurrence de la première action de Q (règle 3 de la sémantique opérationnelle). Par induction, a est aussi possible dans C_Q. La place SR est une place d'entrée pour a. Par construction, SR est toujours marquée avant l'occurrence de la première action de C_Q. Par conséquent, l'occurrence de a est toujours dans $C_{P[g \gg Q}$.
2. L'occurrence d'une autre action de Q. Q est actif et P est suspendu (règle 4 de la sémantique opérationnelle). Par induction a est aussi possible dans C_Q. Par construction, SR n'interfère pas avec l'exécution du composant C_Q. Par conséquent a est aussi possible dans $C_{P[g \gg Q}$.
3. L'occurrence d'une action de P. Le processus P n'est pas suspendu. Par induction, a est aussi possible dans C_P. Par construction, la place SR est marquée. Par conséquent, a est aussi possible dans $C_{P[g \gg Q}$.

$\underleftarrow{a}$:similaire à $\underrightarrow{a}$.

4.4.4 Exemples

Machine à laver

Au dessous de la Figure 4.4 se trouve la description en RT-LOTOS étendu de la machine à laver introduite dans la section 4.1. La procédure de traduction définie dans la section précédente produit pour cette spécification le réseau de Petri à chronomètres décrit par la Figure 4.4.

```
specification WM[start, open, close] : exit
behavior hide b_wash, e_wash in

 Machine[start, b_wash, e_wash]
 [close>>
 Cover[open,close]
 where

process Machine[start, b_wash, e_wash] : exit :=
start ; delay(1,2)b_wash ; delay(40,70)e_wash ; exit
endproc

process Cover[open, close] : exit :=
open ; close ; Cover[open, close]
endproc
endspec
```

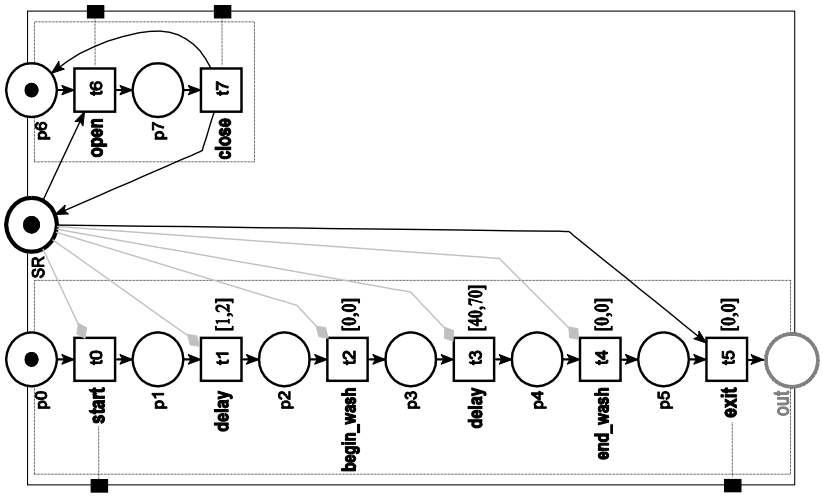


FIG. 4.4: Composant à chronomètres de la machine à laver

La construction proposée en [14] et implantée dans l'outil *Tina* produit en sortie le graphe de classes de la Figure 4.5. Le parcours de la description textuelle de ce graphe de classes nous permet de mettre en évidence plusieurs transitions *gelées* (dont la progression temporelle est stoppée) qui sont : *start* dans la classe 1, *begin_ wash* dans la classe 12, *end_ wash* dans la classe 9 et *delay* dans les classes 10 et 13.

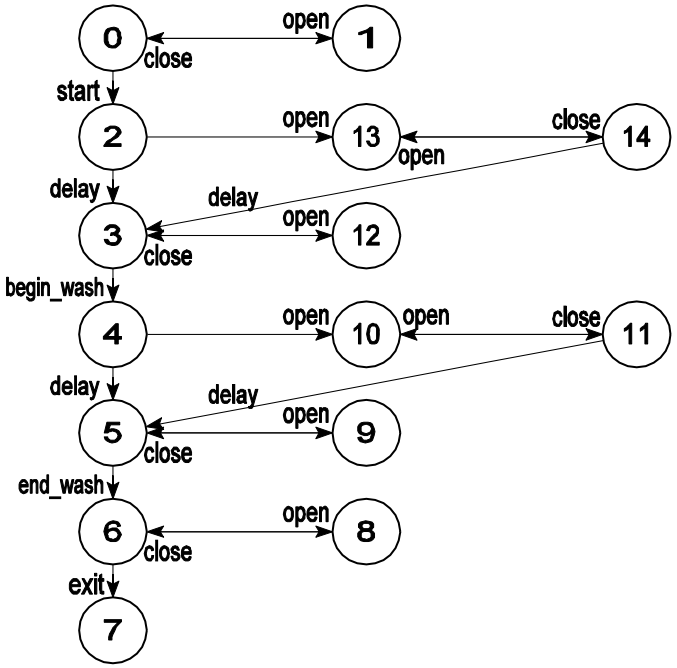


FIG. 4.5: Graphe de classes de la machine à laver

Analyse d'ordonnançabilité

L'opérateur *suspend/resume* proposé dans ce chapitre peut être utilisé dans des problèmes d'analyse d'ordonnançabilité. Ce type d'analyse vise à montrer qu'un ensemble de processus concurrents vont toujours satisfaire aux échéances dès lors qu'on les exécute selon une certaine politique d'ordonnancement sur un système particulier.

De nombreuses techniques ont été développées pour la modélisation formelle de problèmes d'ordonnancement, parmi lesquelles nous pouvons citer [48][1][12][33].[12][33] sont basées sur le formalisme des automates temporisés, tandis que[12][33] développent une approche algébrique.

Nous pensons que les approches basées sur les algèbres de processus, apportent un degré de lisibilité supplémentaire par rapport à celles basées sur des automates temporisés [48][1].

L'opérateur *suspend/resume* $P[g \gg Q$ apporte à RT-LOTOS le *'plus'* nécessaire à une analyse d'ordonnançabilité. Une discipline d'ordonnancement peut être décrite en RT-LOTOS en définissant un *scheduler* qui choisit parmi les processus en compétition ; l'analyse d'ordonnançabilité comporte deux étapes : en premier lieu, la traduction de la description RT-LOTOS en réseaux de Petri temporels à chronomètres que l'on va ensuite analyser avec l'outil *Tina*, dans le but de mettre en évidence les échéances non respectées. Si l'analyse ne révèle aucune échéance non respectée alors le système est déclaré ordonnançable.

A titre d'exemple, nous nous appuyons sur l'étude de cas proposée en [23].

Un contrôleur supervise trois types de tâches. Il démarre la procédure de supervision en envoyant une requête int_i. On suppose ici que les trois tâches s'exécutent sur un même et unique processeur. $Task_3$ et $Task_1$ sont des tâches périodiques de périodes respectivement égales à 150 et 50 unités de temps. $Task_2$ est pour sa part une tache sporadique avec un temps d'inter-arrivées minimal de 100 unités de temps. $Task_1$ a la priorité à la fois sur $Task_2$ et sur $Task_3$. $Task_2$ a la priorité sur $Task_3$.

La spécification en RT-LOTOS étendu de cette description informelle est donnée ci dessous. La Figure 4.6 décrit le réseau de Petri à chronomètres dérivé de cette même description.

```
specification Three_Tasks_System : noexit
behavior hide int1, int2, endT1, endT2, endT3 in
(Controller[int1, int2, int3]
|[int1, int2, int3]|
((Task3[int3,endT3]
  [endT2>> Task2[int2, endT2])
  [endT1>> Task2[int1, endT1]))
where
process Controller [int1, int2, int3] : noexit :=
   Launcher1[int1]
|||Launcher2[int2]
|||Launcher3[int3]
where
process Launcher1[int1] : noexit :=
delay(50)(int1 ; stop
          ||| Launcher1[int1])
endproc
process Launcher2[int2] : noexit :=
delay(150,INF)(int2 ; stop
              ||| Launcher2[int2])
endproc
process Launcher3[int3] : noexit :=
delay(150)(int3 ; stop
           ||| Launcher3[int3])
endproc
process Task1[int1, endT1] : noexit :=
int1 ; delay(10, 20)endT1 ; Task1[int1, endT1]
endproc
process Task2[int2, endT2] : noexit :=
int2 ; delay(18, 28)endT2 ; Task2[int2, endT2]
endproc
process Task3[int3, endT3] : noexit :=
int3 ; delay(20,28)endT3 ; Task3[int3, endT3]
endproc
endspec
```

Tina construit pour cet exemple un graphe de 615 classes et 859 transitions. Tous les marquages sont saufs (1-bornés), ce qui implique l'ordonnançabilité du système.

Les transitions t_0, t_2 et t_3 ne satisfont pas la propriété suivante : $(M \backslash {}^\bullet t) \cap t^\bullet = \phi$. Par conséquent le RdPTC de la Figure 4.6 pourrait ne pas être 1-borné (sauf) ; cependant, les contraintes temporelles empêchent l'insertion d'un nouveau jeton dans l'une des places p_1, p_3 ou p_5. Aucune des places citées ne contient plus d'un jeton et ce dans toutes les configurations possibles. Ceci signifie que le contrôleur ne lance jamais une nouvelle tâche alors qu'elle est en instance d'exécution. En d'autres termes, aucune des tâches ne manque son échéance. Notons que les propriétés quantitatives (du type pire cas) pourraient être vérifiées en rajoutant des observateurs à la description RT-LOTOS.

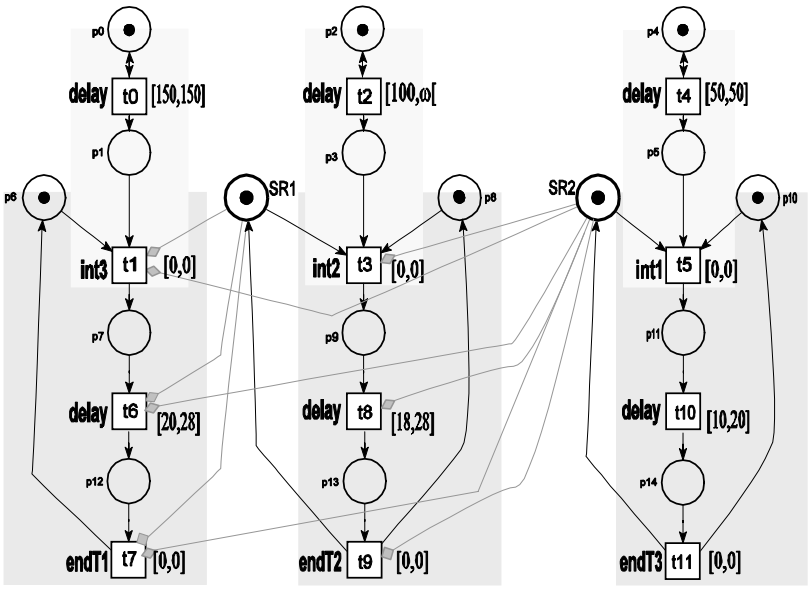


FIG. 4.6: RdPTC du système à trois taches

Un système de contrôle distribué avec *timeout* :

La description du système est tirée de [43]. Le système en question est composé de deux capteurs et un contrôleur. Ce dernier construit des commandes pour un robot en fonction des données lues par les capteurs (cf. Figure 4.7). Les deux capteurs se partagent un seul processeur et la priorité du 2ème capteur, pour utiliser le processeur, est plus élevée que la priorité du 1er capteur.

Chaque capteur construit un relevé et l'envoie au contrôleur. Chacun des capteurs met 1 à 2 millisecondes de temps du processeur pour construire le relevé en question. Une fois construit, le relevé du 1er capteur expire s'il n'est pas délivré dans les 4 millisecondes qui suivent ; le relevé du capteur 2 expire s'il n'est pas délivré dans les 8 millisecondes qui suivent.

Le contrôleur accepte un relevé de chaque capteur dans n'importe quel ordre, et envoie ensuite une commande au robot. Les relevés des deux capteurs qui sont utilisés pour construire une commande pour le robot doivent être reçus dans un intervalle de temps ne dépassant pas les 10 millisecondes. Dans le cas contraire, le relevé du 1er capteur est ignoré. Le contrôleur met 3 à 5 millisecondes pour synthétiser une commande.

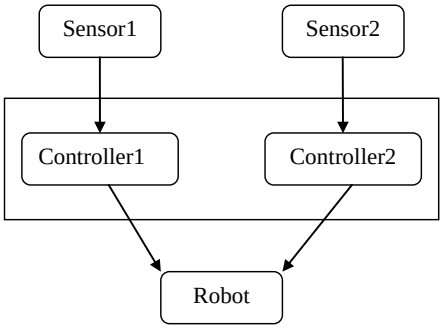


FIG. 4.7: Architecture du controleur de robot

```
Specification DisCtrlSystem : noexit
behavior
hide sR1, sR2, sC1, sC2, eC1, eC2, c1Ok, C2Ok, d1, d2,
  fail1, fail2, rE1,rE2, sig, eS1, eS2, err1, err2 in
((sensor1[sR1, C1Ok, rE1]
  ||| sensor2[sR2, C2Ok, rE2])
 |[sR1, sR2, C1Ok, C2Ok, rE1, rE2]|
 (timeout1[sR1, sC1, eC1, C1Ok, d1, rE1]
  ||| timeout2[sR2, sC2, eC2, C2Ok, d2, rE2]))
|[sC1, sC2, eC1, eC2, d1, d2]|
((controller1(sC1, eC1, err1, eS1]
  [eS2>> controller1(sC2, eC2, err2, eS2])
  ||| robotCtrl[d1, d2, sig, fail1, fail2])
where
process sensor1[sR1, C1Ok, rE1] : noexit :=
sR1 ; (C1Ok ; delay(6)sensor1[sR1, C1Ok, rE1]
      [] rE1 ; sensor1[sR1, C1Ok, rE1])
endproc
process sensor2[sR2, C2Ok, rE2] : noexit :=
sR2 ; (C2Ok ; delay(6)sensor2[sR2, C2Ok, rE2]
      [] rE2 ; sensor2[sR2, C2Ok, rE2])
endproc
process timeout1[sR1, sC1, eC1, C1Ok, d1, rE1] : noexit :=
sR1 ; sC1 ; (eC1 ; d1 ; C1Ok timeout1[sR2, sC1, eC1, C1Ok, d1, rE1]
          []delay(4)rE1 ; timeout1[sR2, sC1, eC1, C1Ok, d1, rE1])
endproc
process timeout2[sR2, sC2, eC2, C2Ok, d2, rE2] : noexit :=
sR2 ; sC2 ; (eC2 ; d2 ; C2Ok timeout2[sR2, sC2, eC2, C2Ok, d2, rE2]
          []delay(8)rE2 ; timeout2[sR2, sC2, eC2, C2Ok, d2, rE2])
endproc
process controller1[sC1, eC1, err1, eS1] : noexit :=
sC1 ; delay(1,2)( eC1 ; eS1 ;  controller1[sC1, eC1, err1, eS1]
               []err1 ; eS1 ; controller1[sC1, eC1, err1, eS1] )
endproc
process controller2[sC2, eC2, err2, eS2] : noexit :=
sC2 ; delay(1,2)( eC2 ; eS2 ;  controller2[sC2, eC2, err2, eS2]
               []err2 ; eS2 ; controller2[sC2, eC2, err2, eS2] )
endproc
process robotCtrl[d1, d2, sig, fail1, fail2] : noexit :=
d1 ; (d2 ; delay(3,5)sig ; robotCtrl[d1, d2, sig, fail1, fail2]
    [] delay(10)fail2 ;  robotCtrl[d1, d2, sig, fail1, fail2])
[]
d2 ; (d1 ; delay(3,5)sig ; robotCtrl[d1, d2, sig, fail1, fail2]
    [] delay(10)fail1 ;  robotCtrl[d1, d2, sig, fail1, fail2])
endproc
endspec
```

Le RdPTC corresspondant contient 38 places et 34 transitions. *Tina* construit pour ce dernier, en 80.59[3] secs un graphe de 40 723 classes et 76 807 transitions.

[3] Ces résultats ont été obtenus sur une machine dotée de 512 Mo de mémoire et d'un processeur cadencé à 1600Mhz

4.5 Comparaison avec les approches existantes pour E-LOTOS et ET-LOTOS

ET-LOTOS Un opérateur *suspend/resume* a été introduit dans ET-LOTOS [45] ; il consiste également en une généralisation du *disrupt*. La différence principale avec le *suspend/resume* présenté pour RT-LOTOS est qu'il permet l'autosuspension. Ainsi un processus P peut se suspendre lui même, sur une action précise, et donner le contrôle à un processus Q. Cependant, le processus P ne peut se suspendre que si le processus Q est prêt à interagir sur l'action de l'autosuspension. P et Q sont ainsi synchronisés sur cette action. Cette dernière est donc différente des autres actions de suspension. Pour la reprise, l'interprétation de l'occurrence de l'action g est la suivante : *austosuspension*, si P n'est pas suspendu. *Reprise* si P est suspendu. Cependant, une interprétation ambiguë de l'occurrence de l'action g peut provoquer des effets indésirables dans le cas de comportements récursifs où l'occurrence de l'action g forcera la reprise de tous les comportements suspendus.

E-LOTOS L'opérateur défini pour E-LOTOS utilise une exception (au lieu d'une action) pour la reprise. Une proposition alternative a été formulée dans [39]. Les auteurs de [39] proposent un opérateur plus général afin de permettre la modélisation de coroutines.

Dans E-LOTOS une exception est un événement observable urgent. Notons que cette définition n'est pas en concordance avec la philosophie de RT-LOTOS qui stipule que l'on ne peut pas forcer l'exécution des évènements observables. La seule manière d'introduire l'urgence dans RT-LOTOS est d'utiliser l'opérateur *Hide*. De plus, une utilisation systématique de l'urgence pour la reprise peut introduire des contraintes non nécessaires et ainsi provoquer des situations de blocage.

Pour conclure ce comparatif, nous soulignerons que notre approche se démarque essentiellement des deux approches citées plus haut par le fait que le problème de la suspension/reprise pour ET-LOTOS et E-LOTOS n'a été traité qu'au niveau langage. A notre connaissance, il n y a pas d'outils d'analyse pour ces deux extensions. Notre approche propose une traduction vers les RdPTC afin que les spécifications en RT-LOTOS étendu puissent être effectivement vérifiées en utilisant l'outil *Tina*.

4.6 Conclusion du chapitre

Dans ce chapitre nous avons montré que le langage RT-LOTOS n'est pas adapté pour décrire des situations où l'exécution d'un système est interrompue puis reprise sans perte du contexte temporel. Nous avons alors proposé d'enrichir RT-LOTOS par un opérateur de suspension/reprise. Nous avons ensuite proposé une traduction du langage RT-LOTOS ainsi étendu vers les réseaux de Petri temporels à chronomètres, tels que les supporte la dernière version de l'outil *Tina*. Ceci permet la construction des espaces des états de tels comportements. Cette traduction avec preuve à l'appui achève notre discussion sur la partie *contrôle* de RT-LOTOS. Le prochain chapitre va traiter de la partie *données* de RT-LOTOS.

5 Traitement des données par une traduction vers les réseaux de Petri temporels à Prédicats et Actions

Dans le chapitre 3 nous avons présenté et formalisé la traduction de la partie *contrôle* de RT-LOTOS (sans les données) vers les RdPT. Ce chapitre franchit une étape supplémentaire en permettant la prise en compte d'une partie *données*. Pour ce faire, nous considérons une classe de RdPT apte à traiter les données : les réseaux de Petri temporels à Prédicats et Actions. Ces derniers constituent une réponse aux problèmes de modélisation posés par certains systèmes temporisés pratiques.

La contribution présentée dans ce chapitre se limite au traitement des entiers et des booléens.

5.1 Le modèle *Full* RT-LOTOS

Nous appelons *Full* RT-LOTOS, la partie *contrôle* de RT-LOTOS étendue avec la possibilité d'échanger des valeurs lors de la synchronisation entre processus. Avec l'ajout des données dans les descriptions RT-LOTOS, les actions deviennent des entités qui adjoignent à une porte trois composantes de base de telle sorte qu'une action puisse :

1. Offrir une valeur x ($!x$),
2. Accepter une valeur y ($?y : type$),
3. Incorporer un prédicat qui conditionne l'acceptation d'une valeur.

Soit $\mathcal{S}$ l'ensemble des *sortes*. *Sorte* étant le nom donné à un domaine de valeurs. Un ensemble typique (et minimal) de sortes est $\mathcal{S} = \{bool, nat, int, string, time\}$ où $bool = \{false, true\}$, $nat = \mathbb{Z}$, $int = \mathbb{N}$, *string* est l'ensemble des chaînes de caractères affichables, et $time = D_0^\infty$.

Soit $\mathcal{V}$ l'ensemble des noms des variables. La fonction totale $sort : \mathcal{V} \longrightarrow S$ retourne pour chaque nom de variable la sorte $sort(v)$ de sa valeur. Soient $x, y \in \mathcal{V}$, $v_b \in bool$, $v_i \in int$ et $v_s \in string$. Soit $\mathcal{O}$ l'ensemble des noms d'*opérations,* et $op \in \mathcal{O}$. Le domaine de chaque *op* est une liste d'éléments de $\mathcal{S}$ et son image est aussi un élément de $\mathcal{S}$. Notons que ces sortes sont assimilées ici à des objets mathématiques (comme des ensembles, des listes, ...) à la place des types abstraits[1] Act-One [30].

En *Full* RT-LOTOS, seules les déclarations des variables (la signature des types abstraits) sont effectuées en utilisant la syntaxe de Act-One. La partie correspondant à la sémantique des types de données est elle implémentée en utilisant les langages C++ ou Java. Ceci a été motivé par des raisons pragmatiques qui rendent l'utilisation des types concrets plus facile que celle des types abstraits.

Par exemple, l'expression $+ : nat, nat \rightarrow nat$ correspond à la définition de l'opération d'addition qui utilise deux opérandes de type *nat* (correspondant aux entiers naturels), dont

[1] Un type abstrait est une structure qui n'est pas définie en termes d'implantation en mémoire ou par une explicitation de ses composantes, mais plutôt en termes d'opérations et de propriétés sémantiques.

le résultat produit un autre *nat*. L'implémentation de cette opération est faite en C++ au moyen d'un objet qui possède une méthode qui prend deux entiers naturels et en retourne un autre, dont le résultat correspond à l'addition des deux derniers. Dans ce mémoire, les données échangées entre processus sont restreintes à des valeurs de type entier ou booléen.

La syntaxe formelle de Full RT-LOTOS est donnée, dans la Figure 5.1 par la syntaxe des expressions des valeurs (E), celle des offres des portes (O), et finalement celle des expressions de comportements (P) ; la syntaxe de déclaration d'un processus étant :
process $X[g_1, ...g_n](x_1 : s_1, ...x_m : s_m) := P$ ***endproc***

$$
\begin{array}{rcl}
E & ::= & op(E_1, \dots E_n) \mid x \mid v_b \mid v_i \mid v_s \\
O & ::= & \texttt{?}\ x\ \texttt{:}\ s \mid \texttt{!}\ E \\
P & ::= & \texttt{stop} \mid \texttt{exit} \mid X\ \texttt{[}\ L\ \texttt{]}\ \texttt{(}\ E_1, \dots E_m\ \texttt{)} \\
 & \mid & \texttt{let}\ x\ \texttt{:}\ s\ \texttt{=}\ E\ \texttt{in}\ P \mid \texttt{hide}\ L\ \texttt{in}\ P \mid \Delta^u\ P \mid \Omega^u\ P \\
 & \mid & \texttt{i}\ \texttt{;}\ P \mid \mid \texttt{i}\ \texttt{\{}\ u\ \texttt{\}}\ \texttt{;}\ P \\
 & \mid & \texttt{i}\ \texttt{@}\ x\ \texttt{;}\ P \mid \texttt{i}\ \texttt{@}\ x\ \texttt{\{}\ u\ \texttt{\}}\ \texttt{;}\ P \\
 & \mid & g\ O_1 \cdots O_n\ \texttt{;}\ P \mid g\ \texttt{\{}\ u\ \texttt{\}}\ O_1 \cdots O_n\texttt{;}\ P \\
 & \mid & g\ \texttt{@}\ x\ \texttt{\{}\ u\ \texttt{\}}\ O_1 \cdots O_n\texttt{;}\ P \\
 & \mid & P\ \texttt{[]}\ P \mid P\ \texttt{|[}\ L\ \texttt{]|}\ P \mid P\ \texttt{>>}\ P \mid P\ \texttt{[>}\ P
\end{array}
$$

FIG. 5.1: Syntaxe de Full RT-LOTOS

Une présentation complète du modèle *Full* RT-LOTOS est donnée dans [57]. Toutes les règles qui nous intéressent dans le cadre ce chapitre se déduisent facilement des règles sémantiques de la Figure 2.5. Il suffit pour cela de remplacer l'offre des portes de *Basic* RT-LOTOS par celle de l'offre générale des portes pour prendre en compte l'échange des données.

La Table 5.1 liste les constructions RT-LOTOS traitées dans ce chapitre.

Constructions RT-LOTOS	**Description**
$a\,?y\,!x$	Réception de la valeur y et émission de la valeur x
$exit(v_1,...,v_n)$ $\gg$ $accept\ x_1 : int,...,x_n : int$ $in\ P$	Le processus qui a terminé son exécution avec succés transmet des valeurs $v_1,...,v_n$ au processus suivant qui peut les consulter à travers les variables $x_1 : int,...,x_n : int$ respectivement
$Let\ x : int = v$ in P	Instanciation de la variable $x$avec la valeur v dans P
$[pred] \rightarrow P$	P est exécuté si $pred$ est vrai

TAB. 5.1: Sous-ensemble de constructions RT-LOTOS avec utilisation des données

5.2 Réseaux de Petri temporels à Prédicats et Actions

Les réseaux de Petri temporels à Prédicats et Actions (ou RdPTPA) sont une extension des réseaux de Petri temporels (RdPT). Les RdPTPA sont obtenus à partir des RdPT en associant un prédicat et une action à chaque transition. Cette extension accroît le pouvoir de modélisation des RdPT, tout en conservant les principes de fonctionnement de ces derniers.

Nous énumérons ici les principes de fonctionnement des RdPTPA :

– Des variables globales sont associées au réseau. Ces variables peuvent être de type entier, booléen ou un enregistrement d'une quelconque complexité. La valeur initiale de ces

variables est indéfinie.
- Les conditions de sensibilisation d'une transition dépendent de la valeur des variables (en plus des conditions de sensibilisation dans les RdPT). Ainsi, un *prédicat* est associé à chaque transition. Ce *prédicat* est la condition nécessaire qui doit être satisfaite pour sensibiliser une transition (déjà sensibilisée d'un point de vue RdPT).
- Le tir d'une transition peut changer les valeurs des variables globales. A cet effet, une *action* est associée à chaque transition.

Dans la section qui suit, nous décrivons de façon formelle les concepts cités plus haut.

5.2.1 Définition formelle

Soit I^+ l'ensemble non vide des intervalles réels avec bornes rationnelles non négatives. Pour $i \in I^+, \downarrow i$ représente sa borne inférieure, et $\uparrow i$ sa borne supérieure (si elle existe) ou ∞ (sinon). Pour tout $\theta \in \mathcal{R}^+$, $i \dot{-} \theta$ représente l'intervalle $\{x - \theta \mid x \in i \wedge x \geq \theta\}$.

Définition 5.1 Un réseau de Petri temporel à Predicats et Actions (ou RdPTPA) est un *n-uplet* $\langle P, T, Pre, Post, m_0, I_s,\ Prec,\ A,\ D,\ d_0\rangle$, où $\langle P, T, Pre, Post, m_0, I_s\rangle$ est un RdPT, D est l'ensemble des vecteurs d'entiers de taille fixe. $A : T \rightarrow D \times D$ est une fonction qui associe une action A_t à chaque transition t du réseau, $Prec : T \rightarrow D \times Bool$ est une fonction qui associe un *prédicat* $Prec_t$ à chaque transition t du réseau. $d_0 \in D$ est l'environnement initial, dans lequel aucune variable n'est instanciée. Graphiquement, à chaque transition t on rattache une expression de la forme : $when\ \ Prec_t(d)\ \ do\ \ d' \leftarrow A_t(d)$.

Dans l'expression ci-dessus nous faisons référence à deux versions de l'environnement (d, d'). Nous utilisons d pour référencer l'environnement de l'état s, et d' pour référencer celui de l'état s'. Ainsi, en écrivant $x' = x + 1$ nous exprimons le fait que la valeur de la variable x dans l'état s' est plus grande de 1 que sa valeur dans l'état s.
Une transition t est dite *activable*, ssi elle est sensibilisée par le marquage m et son prédicat est satisfait :$activable(t, m, d) \Leftrightarrow m \geq Pre(t)\ \wedge\ Prec_t(d)$.

Definition 5.2 Un état d'un RdPTPA est un tuple $s = \langle m, I, d\rangle$ tel que m est un marquage et $I : T \rightarrow I^+$ est une fonction qui associe un intervalle temporel dans I^+ à chaque transition sensibilisée par m.

Nous noterons $\langle m, I, d\rangle \xrightarrow{t@\theta} \langle m', I', d'\rangle$ ssi $\theta \in \mathcal{R}^+$ et:

1. $m \geq Pre(t)\ \wedge\ \theta \geq \downarrow I(t)\ \wedge\ (\forall k \in T)(m \geq Pre(k) \Rightarrow \theta \leq \uparrow I(k))$
2. $m' = m - Pre(t) + Post(t)$
3. $(\forall k \in T)\ activable(k, m', d') \Rightarrow$
   $I'(k) =\ si\ k \neq t \wedge activable(k, m - Pre(t), d)\ \ alors\ \ I(k) \dot{-} \theta\ \ sinon\ I_s(k))$
4. $Prec_t(d) = true$
5. $d' \leftarrow A_t(d)$

L'état initial est défini par $s_0 = \langle m_0, I_0, d_0\rangle$.

(1) assure que la transition t tire dans son intervalle temporel à moins qu'elle ne soit désensibilisée par le tir d'une autre transition.

(2) est la règle classique de transformation de marquage.

(3) signifie que les transitions nouvellement sensibilisées sont associées à leurs intervalles de tir statique alors que les transitions persistantes (celles restant sensibilisées lors du tir) ont leur intervalle décalé de θ, vers l'origine des temps. Les transitions non sensibilisées par le nouveau marquage m' reçoivent des intervalles de tir vides. Soit t une transition sensibilisée du point de vue RdPT,

(4) signifie que les valeurs des variables de l'environnement d doivent satisfaire le prédicat associé à t.

(5) assure qu'après le tir de t, le nouvel environnement d' est obtenu en évaluant l'action associée à la transition t, avec les valeurs des variables de d comme arguments.

Dans la suite nous adoptons les conventions de notation suivantes:

- *when* $Prec_t(d)$ peut être omis, auquel cas la valeur de $Prec_t(d)$ est *true*.
- *do* $d' \leftarrow A_t(d)$ peut être omis, auquel cas le tir de la transition t ne change pas l'environnement d. Pour des raisons de clarté, nous expliciterons cette action vide en l'appelant *none*.

Exemple : La Figure 5.2 décrit un RdPTPA dont l'environnement est constitué d'une seule variable entière x. la transition t_1 incrémente la valeur de x, t_3 réinitialise x si la valeur de x est supérieure à 3. Le prédicat *true* est associé aux transitions t_0 et t_1. L'action *none* est associée à la transition t_2.

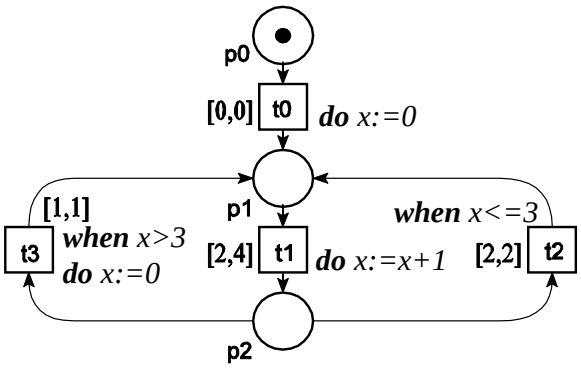


FIG. 5.2: Exemple de RdPTPA

5.3 Composants RdPTPA

De même que pour les RdPT, une vue modulaire des RdPTPA est nécessaire pour mener à bien le processus de traduction du sous-ensemble de full RT-LOTOS identifié dans la Table 5.1 vers les RdPTPA.

Nous généralisons ici la notion de composant introduite dans le chapitre précédent, pour y introduire des variables, des prédicats, des actions sur les variables ainsi que des offres structurées.

Nous exploiterons cette extension de telle sorte que les données puissent être :

1. offertes sur les points d'interactions d'un composant, et ainsi échangées entre composants (enrichissement de l'opérateur de préfixage par une action).
2. Utilisées pour exprimer des conditions à satisfaire préalablement par un composant afin que son exécution puisse se réaliser.
3. Utilisées pour instancier des variables (notion de composant paramétrable).
4. Passées par un composant qui a terminé son exécution avec succès à un autre composant (enrichissement des opérateurs de terminaison avec *succès* et de composition séquentielle).

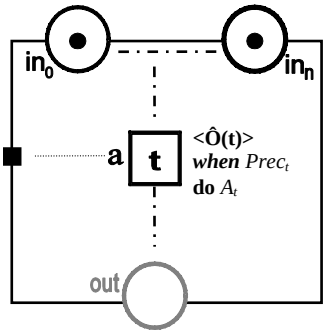


FIG. 5.3: Composant RdPTPA

Définition 5.3 Soit *Act* un alphabet d'actions.

Un composant RdPTPA (cf. Figure 5.3) est un tuple $C = \left\langle \sum, lab, I, O, \hat{O} \right\rangle$ où :

- $\sum = \langle P, T, Pre, Post, M_0, IS, Prec, A, D, d_0 \rangle$ est un RdPTPA.
- $Lab : T \rightarrow (Act \cup Time \cup \{\varepsilon\})$ est une fonction d'étiquetage qui étiquette chaque transition de $\sum$ avec un label d'action (Act), un label d'évènement temporel ($Time = \{tv, delay, latency\}$) ou ε. Une ε-transition[2] est une transition urgente (ayant un intervalle statique $[0,0]$). Cette dernière ne correspond pas à une action observable et de ce fait, elle est associée à une liste d'offres vide. Nous notons T^{Act}(resp. T^{Time}) l'ensemble des transitions avec des labels dans Act (resp. $Time$)
- $I \subset P$ est un ensemble non vide de places définissant l'interface d'entrée du composant.
- $O \subset P$ est l'interface de sortie du composant. Un composant possède une interface de sortie s'il a au moins une transition étiquetée par $exit$. Auquel cas, O est la place de sortie de ces transitions. Sinon, $O = \emptyset$. $Card(O) \leq 1$.
- $\hat{O}$ est une liste d'offres construite sur D. $\hat{O}(t) = \langle O_1, \ldots, O_n \rangle$ représente les offres associées à la transition t. Deux types d'offres sont à distinguer : une *déclaration* de valeur, qui a la forme $!e$ (où e est une expression de valeur) et une *déclaration* de variable, qui prend la forme $?x : t$ (où x est une variable et t son type,)[3]. Notons que dans un composant représentant un système complet (incluant son environnement), seules des offres de la forme $!e$ existent.

[2] Les ε-transitions peuvent être éliminées du graphe de classes par la fermeture transitive, comme dans l'outil *Caesar*.

[3] Les déclarations de variables, représentent les entrées acceptées par le composant (input), contrairement aux déclarations de valeurs qui représentent les sorties du composant (output).

De plus, les invariants H1-H5 de la Section 3.3 sont également associés aux composants RdPTPA.

Exemple : Dans ce qui suit nous illustrons l'utilisation des deux types d'offres possibles dans un composant RdPTPA.

La Figure 5.4 décrit un composant *'Sender'* qui reçoit un message de l'environnement et le transmet à son tour à un récepteur.

Le comportement temporel du *Sender* est le suivant : après que ce dernier ait transmis le message, il attend un acquittement pendant 5 unités de temps (concept d'une *offre limitée* dans le temps). Une fois ce délai écoulé sans réception d'acquittement (concept de *délai déterministe*) l'émetteur retransmet le message. Cette description informelle, peut être traduite littéralement en RT-LOTOS :

```
SENDER := request ?data :int ; Transmitter(data)
where
Transmitter(data) :=transmits !data ; (ack{5} ; SENDER
                                    []delay(5)timeout ; Transmitter(data))
```

Pour des raisons de concision nous avons pris quelques libertés avec la syntaxe RT-LOTOS.

Rappelons que l'opérateur du délai déterministe ne résout pas le choix, les détails du pattern du choix (représenté en gris dans la Figure 5.4) peuvent être trouvés dans la Section 3.7.3.

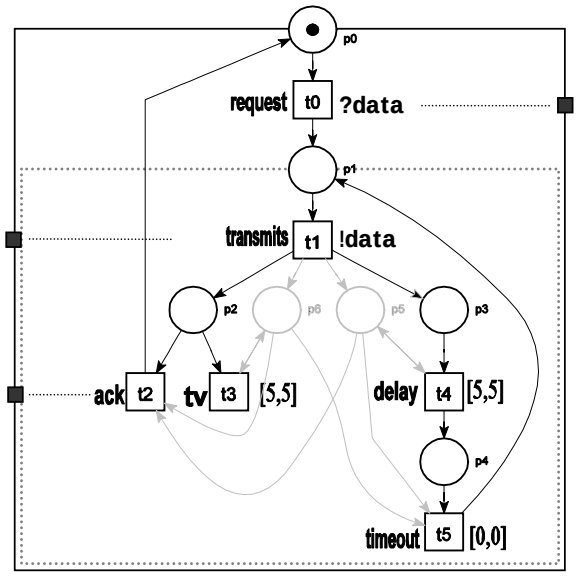


FIG. 5.4: Composant Sender

Notons que le composant de la Figure 5.4 ne représente pas un système complet étant donné qu'il contient une offre de type *déclaration de variable* (*?data*) associée à son point d'interaction *request*. Il est généralement impossible de construire l'espace des états d'un comportement si l'on ne connaît pas les valeurs des variables présentes dans le comportement en question[4].

[4]Une solution au problème serait d'avoir recours à une *évaluation symbolique*. La différence est que la variable *data* ne serait alors plus considérée comme une valeur numérique mais plutôt comme un nom symbolique de variable.

Notre but est d'utiliser le graphe de classes d'un RdPTPA engendré par *Tina* pour vérifier et raisonner sur une spécification RT-LOTOS. Par conséquent, nous travaillons sur des systèmes dits *clos* (système+environnement). A cette fin, nous connecterons au *Sender* un autre composant comme illustré dans la Figure 5.9.

Opérations de bas niveau sur les RdPTPA :

La construction incrémentale d'un RdPTPA se fait par rajouts successifs de places et/ou de transitions. Dans ce qui suit, nous formalisons ces différentes opérations :

Définition 5.4 (Ajout d'une place ou d'une transition à un RdPTPA)

Soit $\sum = \langle P, T, Pre, Post, m_0, IS, Prec, A, D, d_0 \rangle$ un RdPTPA.

Ajout d'une place à $\sum$: Soient p une nouvelle place à rajouter à $\sum$ ($p \notin P$), Pre_P et $Post_P$ deux ensembles de transitions dans T. $\sum' = \sum \cup \langle Pre_P, p, Post_p \rangle$ est le RdPTPA augmenté de la place p tel que ${}^\bullet p = Pre_p$ et $p^\bullet = Post_p$.

$$\sum' = \left\langle P \cup \{p\}, T, Pre \cup \bigcup_{t \in Pre_p} (p,t),\ Post \cup \bigcup_{t \in Post_p} (t,p), m_0, IS, Prec, A, D, d_0 \right\rangle$$

Ajout d'une transition à $\sum$: soient t ($t \notin T$) une nouvelle transition à rajouter à $\sum$, I_t l'intervalle statique de t, u le prédicat associé à t ($p \subseteq D \times Bool$), a l'action associée à t ($a \subseteq D \times D$) , Pre_t et $Post_t$ deux ensembles de places dans P.

$\sum' = \sum \cup \langle Pre_t, (t, I_t, u, a), Post_t \rangle$ est le RdPTPA augmenté de la transition t tel que ${}^\bullet t = Pre_t$ et $t^\bullet = Post_t$.

$$\sum' = \Big\langle P, T \cup \{t\}, Pre \cup \textstyle\bigcup_{p \in Pre_t}(p,t), Post \cup \bigcup_{p \in Post_t}(t,p) \\ m_0, IS \cup (t, I_t), Prec \cup (t,u),\ A \cup (t,a),\ D,\ d_0 \Big\rangle$$

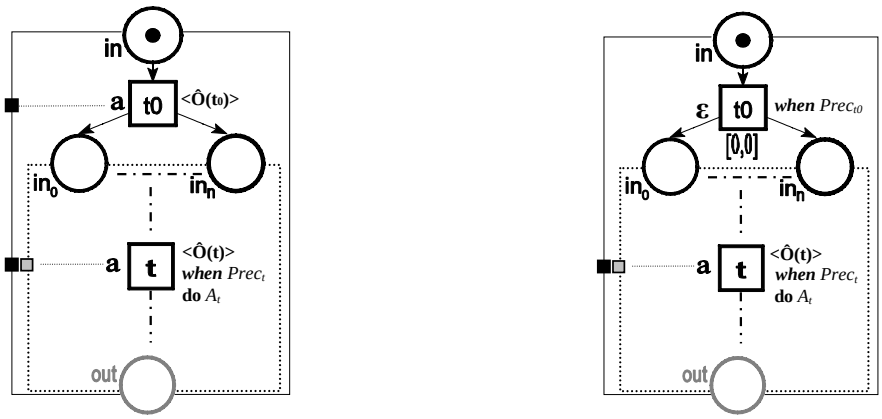
5.4 Patterns de traduction

Dans ce qui suit, nous présentons inductivement la traduction des constructions RT-LOTOS de la Table 5.1 vers les RdPTPA.

5.4.1 Pattern du préfixage par une action et composant gardé

Ce pattern (cf. Figure 5.5(a)) est utilisé pour la traduction des expressions du type $a\,?y : int\,!x$, où l'action a accepte une valeur y et offre une valeur x.

Dans le modèle de *composant* présenté dans le Chapitre 3, une action observable coïncidait avec un label de transition. Dans un composant RdPTPA, un label seul, ne suffit plus à représenter une action observable. Cette dernière est en effet formée d'un label de transition auquel on rajoute une liste de valeurs offertes par cette transition : $a < \hat{O}(t_a) >$. Rappelons que les valeurs offertes sont de type entier. Ainsi, un nombre infini d'actions observables est exprimable dans un composant RdPTPA.



(a) Préfixage par une action (b) Composant gardé

FIG. 5.5: Préfixage par une action et garde

$C_{a<\hat{O}(t_a)>;P}$ est le composant obtenu en préfixant le composant C_P par l'action $a < \hat{O}(t_a) >$. le composant résultant $C_{a<\hat{O}(t_a)>;P}$ exécute l'action a et active par la suite le composant C_P. La définition formelle du composant $C_{a<\hat{O}(t_a)>;P}$ suit :

Définition formelle

$$C_{a<\hat{O}(t_a)>;P} = (\sum_{a<\hat{O}(t_a)>;P}, lab_{a<\hat{O}(t_a)>;P}, \{in\}, O_P, \hat{O}_P \cup \hat{O}(t_a))$$

où le RdPTPA $\sum_{a<\hat{O}(t_a)>;P}$ est obtenu en rajoutant la place in et la transition t_0 à $\sum_P$. Pour finir , $Lab_{a<\hat{O}(t_a)>;P}$ associe le label a à la transition t_0.

$$\sum_{a<\hat{O}(t_a)>;P} = (\sum_P \cup \langle \phi, (t_0, [0, \infty), true, none), I_P \rangle \cup \langle \phi, in, t_0 \rangle)$$

$$Lab_{a<\hat{O}(t_a)>;P} = lab_P \cup (t_0, a)$$

Exemple : La Figure 5.6 décrit un composant qui offre la valeur 3 sur son point d'interaction *out*. Ce composant correspond à l'expression RT-LOTOS suivante : *out*!3; *stop*.

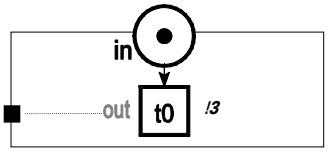


FIG. 5.6: Exemple de préfixage par une action

Notons qu'il est possible de communiquer plusieurs valeurs en même temps (un vecteur de valeur) sur la même action de synchronisation.

Un composant RdPTPA peut être préfixé par une $\varepsilon - transition$ et un prédicat (cf. Figure 5.5(b)). Un tel composant est dit gardé.

L'interprétation est la suivante : si le prédicat est satisfait, alors le composant est activé et son exécution est donc rendu possible. Un scénario typique est celui du choix entre plusieurs composants gardés. Ce pattern est utilisé pour la traduction de la construction $[pred] \rightarrow P$.

Les $\varepsilon - transitions$ s'avèrent aussi utiles lorsque la valeur d'une variable doit être modifiée ou testée sans que cela ne résulte en une action (observable ou non). Comme par exemple dans l'expression : $let\, x : nat = 0\, in\, P$.

5.4.2 Synchronisation et communication entre composants

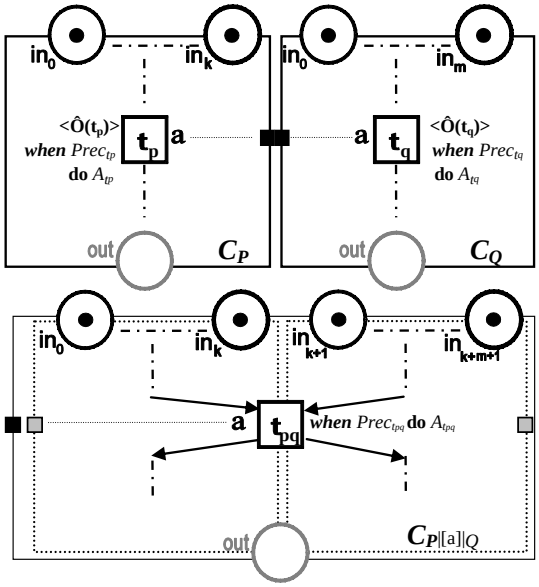


FIG. 5.7: Pattern de communication entre composants

La synchronisation parallèle de deux composants sur une action donnée, signifie qu'après avoir exécuté simultanément l'action en question, les deux composants reprennent leurs exécutions de manière concurrente.

Sur la Figure 5.7 la synchronisation des deux composants C_P et C_Q sur l'action a est obtenue en fusionnant chaque transition étiquetée par a dans C_P avec chaque transition étiquetée par a dans C_Q, créant ainsi $n * m$ transitions étiquetées par a dans le composant résultant $C_{P|[a]|Q}$ (n et m représentent le nombre de transitions étiquetées par a dans C_P et C_Q respectivement). La fusion des transitions concernées par la synchronisation (appartenant à la liste des actions auxquelles l'opérateur impose de synchroniser) se fait sans prendre en considération leurs intervalles temporels (cf. H5). Néanmoins, l'expression du rendez-vous dans les composants RdPTPA nécessite la prise en compte des offres associées aux transitions concernées par la synchronisation. Ainsi les attributs de la transition t_{pq} (issue de la fusion des deux transitions t_p et t_q) sont construits à partir des offres de t_p et t_q.

La table qui suit précise ces règles.

$O_i(t_p)$	$O_i(t_q)$	**type d'interaction**	$Prec_{t_{pq}}$	$A_{t_{pq}}$
$!e_1$	$!e_2$	assortiment de valeur	$e_1 = e_2$	none
$!e$	$?x : int$	passage de valeur	$true$	$x' := e$
$?x : int$	$?y : int$	génération de valeur	$true$	$(x', y') := (e, e)$

TAB. 5.2: Construction des attributs de la transition t_{pq}

Le cas où $O_i(t_p) = ?x : int$ et $O_i(t_q) = !e$, dual du 2ème cas de la table de la Figure 5.2, est traité de la même manière.

Pour résumer, la communication entre deux composants RdPTPA est possible quand ces deux composants exécutés en parallèle offrent la même action structurée (même label, même valeurs), et lorsque l'action fait partie des actions identifiées par l'opérateur de synchronisation parallèle.

Soit T_P^a l'ensemble des transitions étiquetées par a dans C_P.

$$T_P^a = \{t \in T_P |\, Lab_P(t) = a\}$$

$$T_P^A = \bigcup_{a \in A} T_P^A$$

Le réseau $\sum_{P|[A]|Q}$ est obtenu en remplaçant chaque transition t_p étiquetée par $a \in A$ du composant C_p par un ensemble de transitions (t_p, t_q), (t_q est aussi étiquetée par a dans C_Q) de telle sorte que l'ensemble des places d'entrée (resp sortie) de la transition nouvellement créee (t_p, t_q) soit égal à l'union des ensembles de places d'entrée (resp sortie) des transitions t_p et t_q. ${}^\bullet(t_p, t_q) = {}^\bullet t_p \cup {}^\bullet t_q$ (resp $(t_p, t_q)^\bullet = t_p^\bullet \cup t_q^\bullet$). La transition ainsi créée reçoit l'intervalle statique $[0, \infty)$(cf. H5).

Notons que les deux composants doivent aussi se synchroniser sur l'action *exit* pour se conformer à la sémantique de RT-LOTOS. En d'autres termes les composants C_P et C_Q doivent se terminer de manière synchrone. Ainsi, les interfaces de sortie des deux composants sont à leur tour fusionnées. $O_{P|[A]|Q} = \{out\}$ si $(O_P \neq \emptyset \wedge O_Q \neq \emptyset)$, $O_{P|[A]|Q} = \emptyset$ sinon.

Notons :

$$merge(t_p, t_q) = \langle {}^\bullet t_p \cup {}^\bullet t_q, ((t_p, t_q), IS(t_p),\ Prec_{t_{pq}},\ A_{t_{pq}}, t_p^\bullet \cup t_q^\bullet \rangle$$

$$A' = A \cup \{exit\}$$

$$O_{P|[A]|Q} = \begin{cases} \{out\} & si\ O_P \neq \emptyset \wedge O_Q \neq \emptyset \\ \emptyset & sinon \end{cases}$$

$$PreOut = \begin{cases} T_P^{exit} \times T_Q^{exit} & si\ O_P \neq \emptyset \wedge O_Q \neq \emptyset \\ \emptyset & sinon \end{cases}$$

Définition formelle

$$C_{P|[A]|Q} = \left\langle \sum_{P|[A]|Q}, Lab_{P|[A]|Q}, I_P \cup I_Q, O_{P|[A]|Q}, \hat{O}_{P|[A]|Q} \right\rangle$$

$$\sum_{P|[A]|Q} = (\sum_{P} \backslash T_P^{A'} \setminus O_P) \cup (\sum_{Q} \backslash T_Q^{A'} \setminus O_Q) \cup \bigcup_{t_p \in T_P^{A'}, t_q \in T_Q^{A'}} merge(t_p, t_q) \cup \langle PreOut, Out, \phi \rangle$$

$$Lab_{P|[A]|Q}(t) = \begin{cases} Lab_P(t) & si\ t \in T_P \\ Lab_Q(t) & si\ t \in T_Q \\ a & si\ t = (t_p, t_q) \wedge t_p \in T_P^a \end{cases}$$

Soit un composant C_X, $\hat{O}_X^A$ est l'ensemble des offres associées aux transitions dont le label appartient à l'ensemble A.$\hat{O}_X^A = \{\hat{O}_X(t)/t \in T_X^A\}$

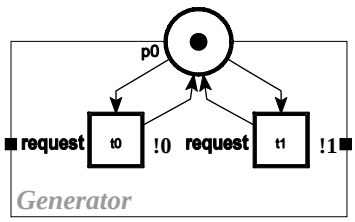
$$\hat{O}_{P|[A]|Q} = \hat{O}_P \cup \hat{O}_Q / \hat{O}_P^A / \hat{O}_Q^A$$

Exemple : Dans l'exemple suivant, le composant *'Sender'* de la section précédente est synchronisé avec un composant *'Generator'*. Ce dernier génère aléatoirement des valeurs *"finies"* (0 ou 1) pour la variable *data*.

La spécification RT-LOTOS du *Generator* et le composant correspondant (cf. Figure 5.8) suivent :

```
Generator := request!0; Generator
             []
             request!1; Generator
```

(a) Comportement RT-LOTOS



(b) Composant

FIG. 5.8: Comportement RT-LOTOS et composant *Generator*

Les composants *Sender* et *Generator* sont synchronisés sur l'action *request*. La Figure 5.9 décrit le composant résultant de cette synchronisation.

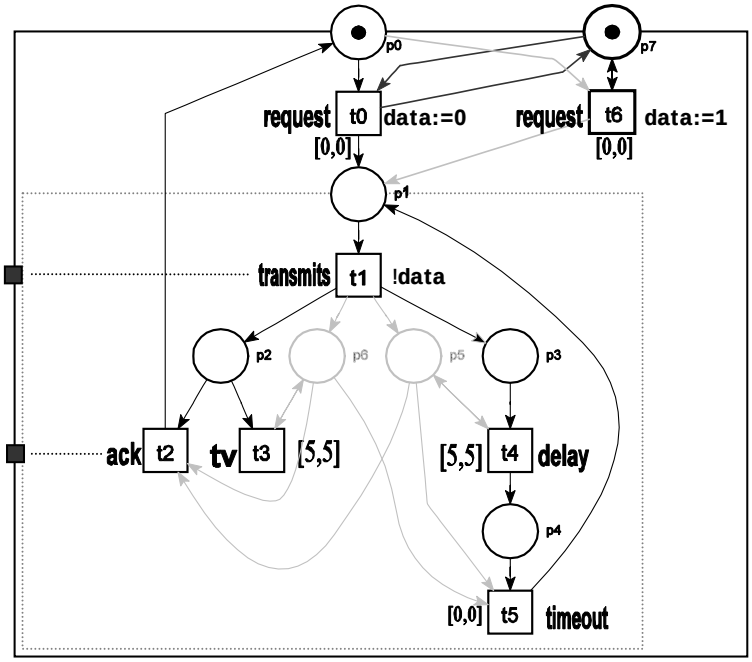


FIG. 5.9: Synchronisation des composants Sender et Generator

Après l'occurrence de la transition *request,* une valeur (soit 0 ou 1) est substituée à *data* dans le composant $C_{Transmitter(data)}$.

5.4.3 Composition séquentielle avec passage de valeurs

Ce pattern est utilisé pour la traduction de la construction suivante :

```
exit(v1,...,vn)
>>
accept x1 :int,...xn :int in P
```

La Figure 5.10 décrit la composition séquentielle des composants C_P et C_Q . Ainsi, si le composant C_P termine avec *succès* son exécution, il activera alors le composant C_Q. Ce type de composition est possible seulement si le composant C_P possède une interface de sortie.

Nous voulons exprimer le fait que le comportement du composant C_Q dépend des paramètres établis durant l'exécution du composant C_Q. Ainsi, un composant RdPTPA terminant son exécution avec *succès* peut transmettre des données à son successeur. Nous avons donc besoin d'un mécanisme pour permettre de passer ces paramètres entre ces deux composants.

A cette fin, nous devons généraliser la notion de terminaison avec succès d'un composant. Appelons *fonctionnalité* du composant C_P le tuple $func(C_P) = (e_1, ..., e_n)$. Ce dernier détermine les données transmises au composant C_Q. De même, nous appelons *admission* du composant C_Q l'ensemble des données que ce dernier s'attend à recevoir, $adm(C_Q) = (v_1, ..., v_n)$.

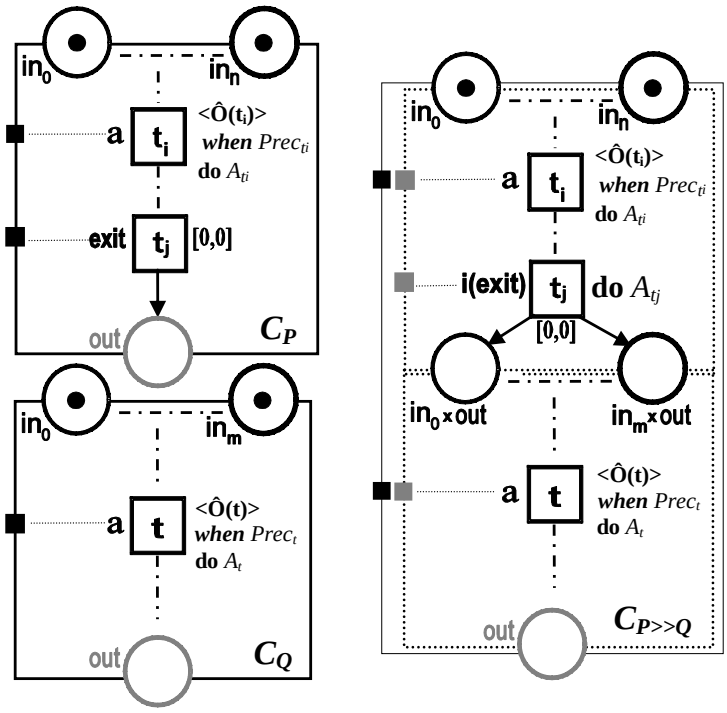


FIG. 5.10: Pattern de composition séquentielle avec passage de valeurs

Le composant résultant $C_{P \gg Q}$ est obtenu en fusionnant l'interface de sortie de C_P et l'interface d'entrée du composant C_Q , et en intériorisant le point d'interaction associé à la transition *exit* dans le composant C_P.

Une nouvelle *action* est associée aux transitions *exit* du composant C_P. Cette dernière actualise une liste finie de variables appartenant à l'admission du composant C_Q, en leur affectant les valeurs des variables dans la fonctionnalité du composant C_P $((v_1, ..., v_n) := (e_1, ..., e_n))$.

Définition formelle

$$C_{P \gg Q} = \left\langle \sum_{P \gg Q}, Lab_{hide\ exit\ in\ P} \cup Lab_Q, I_P, O_Q, \hat{O}_P \cup \hat{O}_Q \right\rangle$$

$$\sum_{P \gg Q} = \langle P_P \setminus O_P \cup P_Q, T_{hide\ exit\ in\ P} \cup T_Q, Pre_P \cup Pre_Q, Post_{P \gg Q}, IS_P \cup IS_Q$$

$$Prec_{hide\ exit\ in\ P} \cup Prec_Q, A_{P \gg Q}, d_0 \rangle$$

$$Post_{P \gg Q} = (Post_P \setminus \{(t, O_P) |\ t \in^\bullet O_P\}) \cup \{(t, in_Q) | in_Q \in I_Q \wedge t \in^\bullet O_P\} \cup Post_Q$$

$$A_{P \gg Q} = A_P / (t_{exit}, none) \cup A_Q \cup (t_{i(exit)}, A_{i(exit)})$$

Exemple : La spécification RT-LOTOS suivante décrit une procédure de *login*. L'action *prompt* représente la demande d'authentification par un utilisateur, qui doit se produire dans un délai qui ne dépasse pas les 10 unités de temps. Si ce délai est écoulé, le système est redémarré, sinon le système accepte la requête (si le nom de *login* et le mot de passe sont corrects), ou la refuse.

```
LOGIN_PROCEDURE := PHASE1 >> accept correct_login :Bool in PHASE2
where
PHASE1 := prompt ; (exit(true)[](exit(false))
          [>delay(10)timeout ; Phase1
PHASE2 :=([correct_login]-> exit)
         []
         ([not(correct_login)]-> stop)
```

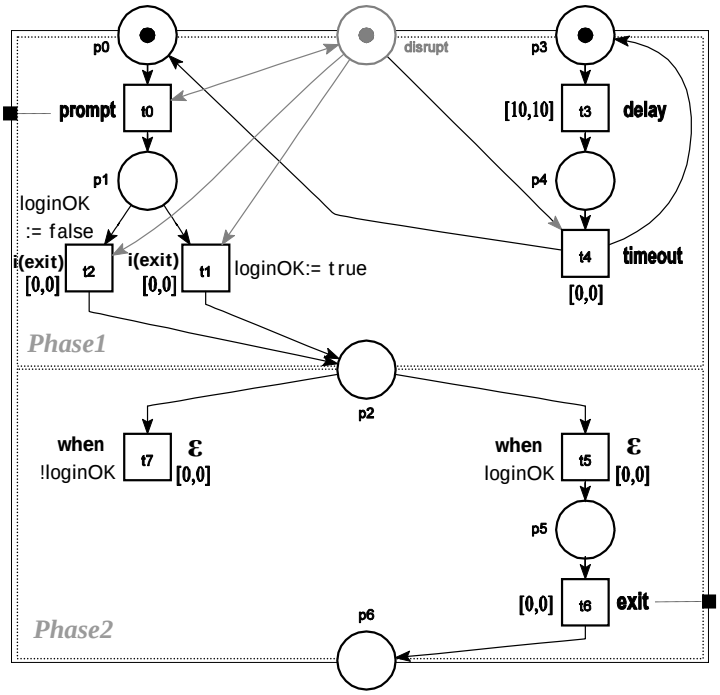


FIG. 5.11: Composant $C_{Phase1 \gg Phase2}$

Dans la Figure 5.11, le comportement du composant C_{Phase2} dépend de la variable *loginOK*, dont la valeur est établie durant l'exécution du composant C_{Phase1}.

5.5 Exemple : protocole *Stop&Wait*

Le protocole *Stop&Wait* est une technique simple pour une transmission fiable dans laquelle un émetteur transmet un message et se met à attendre une réponse. Avant d'émettre à nouveau un message, le processus émetteur attend l'arrivée de l'accusé de réception du message qu'il a précédemment envoyé. Pour corriger les pertes de messages et/ou d'accusés, une temporisation est lancée lorsqu'un message est émis. Si l'accusé de réception du message n'arrive pas avant

l'expiration de la temporisation, un timeout se produit et le message est alors retransmis. Nous donnons dans ce qui suit la spécification RT-LOTOS du protocole *Stop&Wait* :

```
specification StopWait : noexit
behaviour
hide send,rec_ack,timeout,receive,snd_ack,yes,no in

( Sender[send,rec_ack,timeout](false)
  |[send,rec_ack]|
  Medium[send,rec_ack,receive,snd_ack])
|[receive,snd_ack]|
Receiver[receive,snd_ack,yes,no]
where

process Medium[send,rec_ack,receive,snd_ack] :noexit :=
Slot[send,receive]
|||
Slot[snd_ack,rec_ack]
where

process Slot[in_slot,out_slot] :noexit :=
in_slot ;delay(1,6)out_slot ;
Slot[in_slot,out_slot]
endproc (* Slot *)

endproc (* Medium *)
process Sender[send,rec_ack,timeout](wait :bool) :noexit :=
[not(wait)]->(delay(1,3)send ;Sender[send,rec_ack,timeout](true))
[]
[wait]->(
  (rec_ack ; Sender[send,rec_ack,timeout](false))
  []
  (delay(19)timeout ; send ; Sender[send,rec_ack,timeout](true))
)
endproc (* Sender *)

process Receiver[receive,snd_ack,yes,no] :noexit :=
receive ;
delay(2,6) (
  (no ; Receiver[receive,snd_ack,yes,no])
  []
  (yes ; snd_ack ; Receiver[receive,snd_ack,yes,no])
)
endproc (* Receiver *)

endspec (* StopWait *)
```

La spécification ci-dessus décrit trois processus synchronisés entre eux.

Sender : le processus qui modélise l'émetteur et qui réalise les actions suivantes : *send*, *rec_ ack* : pour la synchronisation avec le médium (émissions et acquittements), *timeout* : pour caractériser l'expiration du délai d'acquittement pour le récepteur.

Medium : lui même composé de deux processus *Slot* au comportement identique, le premier gère le transfert des données et le deuxième celui des acquittements ; un processus *Slot* réalise en boucle deux actions *in_slot* et *out_slot* espacés d'un délai non déterministe compris entre 1 et 6 unités de temps.

Receiver : un processus qui modélise le récepteur et qui réalise les actions suivantes : *receive*, *snd_ack* : pour la synchronisation avec *Medium* (réceptions et acquittements) *yes*, *no* pour caractériser le fait que le récepteur a bien reçu la donnée ou pas.

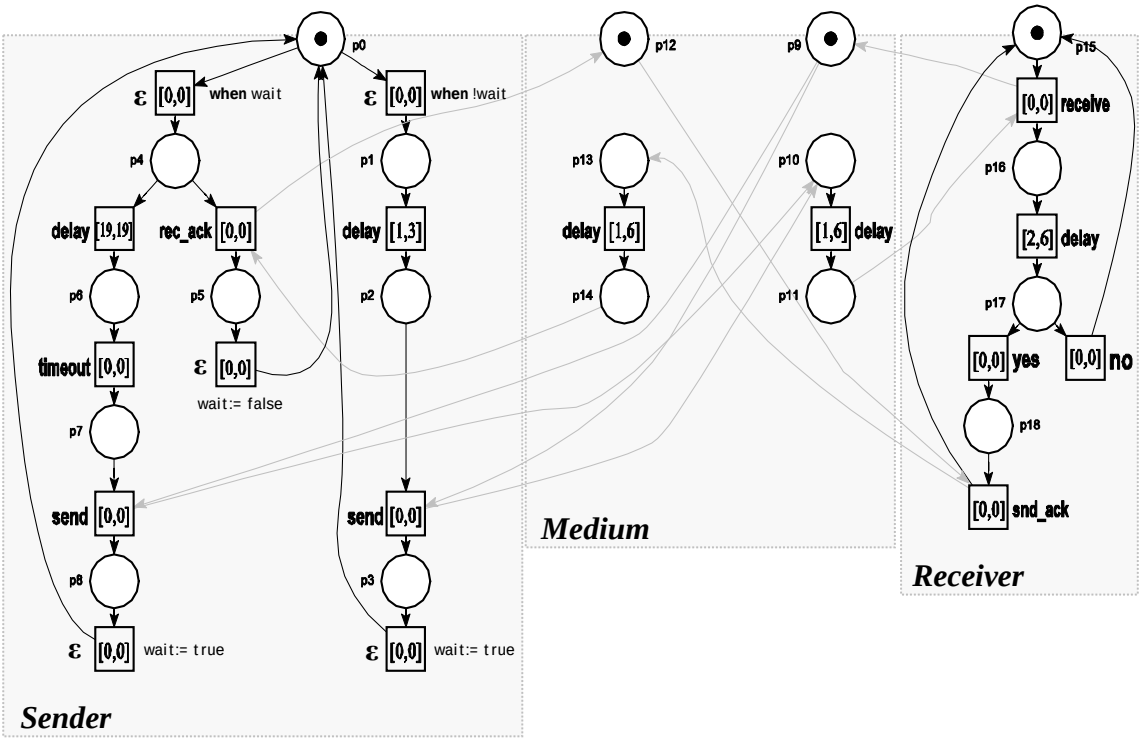


FIG. 5.12: RdPTPA du protocole *Stop&Wait*

Le RdPTPA correspondant est décrit dans la Figure 5.12.

Les RdPTPA sont supportés par *Tina* sous la forme de *Systèmes de Transitions Etiquetés* (composés de deux parties : la partie *contrôle* définie par un RdPT et la partie *données* définie par un ensemble de fonctions C).

Ainsi, pour construire l'espace des états de ce dernier nous devons fournir à *Tina* un fichier *.c* dans lequel nous décrivons la structure contenant l'ensemble des variables du RdPTPA (ici une seule variable *wait* de type booléen), une fonction de comparaison utilisée par *Tina* pour comparer chaque nouvel état avec ceux déjà rencontrés, ainsi que les fonctions représentants la totalité des préconditions et des actions du réseau.

Le RdPTPA de la Figure 5.12 est borné. Son graphe de classes, construit par *Tina* admet 140 classes et 224 transitions.

```
#include “avl.h“/* -- 1. API du stocker */
/* -- 2. déclarations locales -- */
typedef int bool ;
/* toutes les variables présentes dans le PATPN doivent etre déclarées dans cette structure */
struct value {
int wait ;
} value ;
struct value initval ={0} ; // état intial
/* -- 3. fonction de comparaison pour avl API (le stocker) */
bool compare_value(struct value *v1, struct value *v2) {
if(v1->wait > v2->wait) {return 1 ;}
if(v1->wait < v2->wait) {return -1 ;}
return 0 ;
}
/* cette fonction est appelée par le stocker quand une certaine valeur égale à v est déja stockée */
void free_value(struct value *v) {
free(v) ;
}
/* -- 4. fonctions pour l’API tina */
/* initialise la librairie (libavl.so) et retourne l’état initial (requise par tina) */
key initial() {
init_storage(comapre_value, free_value) ;
return store(&initval) ;}
/*table de transitions, utilisé pour mapper les triansitions ’tina’ et les transitions ’données’ */
char *transtable[5]={”t0”,”t3”,”t4”,”t9”,”t12”} ;
/*retourne le nombre des transitions ’données’ présentes dans le PATPN (requise par tina) */
char **transitions(int *sz){*sz=5 ; return transtable ;}
/* -- prédicats et actions -- */
key act_3(key s) {
struct value *v= lookup(s) ; // lookup(s) retourne l’état courant du système
struct value *new=(struct value *)malloc(sizeof(struct value)) ;
memcpy(new,v,sizeof(struct value)) ;
new->wait=1 ; // les actions associées à la transition sont exécutées
return store(new) ; // le nouvel état du système est stocké
}
key act_9(key s) {
struct value *old = lookup(s) ;
struct value *new = (struct value *)malloc(sizeof(value)) ;
memcpy(new,v,sizeof(struct value)) ;
new->wait=1 ;
return store(new) ;
}
key act_12(key s) {
struct value *old = lookup(s) ;
struct value *new = (struct value *)malloc(sizeof(value)) ;
memcpy(new,v,sizeof(struct value)) ;
new->wait=0 ;
return store(new) ;
}
bool pre_0(key s){
struct value *v= lookup(s) ;
return(v->wait==0) ;
}
bool pre_4(key s){
struct value *v= lookup(s) ;
return (v->wait==1) ;
}
```

Afin d'énoncer des propriétés que l'on cherche à vérifier, nous utilisons une variante de la logique temporelle à temps linéaire *SE-LTL* [25]. Cette dernière est une variante de *LTL* récemment introduite qui permet de traiter de façon homogène des propositions d'états et des propositions de transitions. Les modèles de cette logique sont des structures de Kripke étiquetées. Les structures de Kripke associées aux graphes de classes sont constituées de ces graphes, dont les noeuds (états de la structure de Kripke) sont étiquetés par les marquages (interprétés comme les propriétés d'états satisfaites) et les arcs par les transitions tirées (interprétées comme les propriétés de transitions satisfaites).

Exemple : Quelques formules de *SE-LTL :* (Pour tout chemin)

- $\Box P$ P vraie tout le long du chemin,
- $\Diamond P$ P vraie une fois au moins le long du chemin,
- $\Box\Diamond P$ P vraie infiniment souvent,

Les formules de cette logique sont préservées par les abstractions d'espaces d'états produites par *Tina* qui intègre dans sa dernière version un *model checker* pour la logique *SE-LTL* : *selt* [64].

selt permet de vérifier la satisfaction d'une propriété *SE-LTL* sur une structure de Kripke obtenue comme indiqué ci-dessus, depuis un graphe de classes. La vérification d'une propriété s'effectue en deux temps :

1. Construction d'un automate de *Buchi* acceptant les mots qui ne satisfont pas la formule *SE-LTL* à vérifier ; cette phase est réalisée de façon transparente pour l'utilisateur.
2. Construction de la composition de la structure de Kripke obtenue depuis le graphe des classes et de l'automate de Buchi, et recherche à la volée d'une composante fortement connexe contenant un état acceptant de l'automate de Buchi. Si aucune composante pareille n'est trouvée, alors la formule est satisfaite, sinon elle définit un contre-exemple.

En cas de non-satisfaction d'une formule, *selt* peut fournir une séquence contre-exemple en clair ou sous un format exploitable par le simulateur de *Tina,* et ce afin de pouvoir l'explorer pas à pas. Notons que, dans le cas d'un modèle temporisé, il faut au préalable associer à cette exécution un échéancier temporel.

Intéressons nous maintenant à une propriété sur le réseau de la Figure 5.12, exprimée dans le langage de *selt.*

$\phi_1 = \Box(t_{13} \Rightarrow \Diamond t_3)$

ϕ_1 exprime que tout message émis par le *Sender* sera accepté par le *Receiver.*

ϕ_1 n'est pas satisfaite.

Effectivement, tout message peut être rejeté et retransmis une infinité de fois. Le contre-exemple donné par *selt* exhibe une séquence contenant un circuit passant par l'état 80 (état 80 : $p_9p_4p_{16}p_{12}$) et dans lequel le message est rejeté (action *no*), retransmis (action *send*), puis reçu (action *receive*) pour y être à nouveau rejeté.

Une étude de cas plus conséquente, portant sur la vérification d'une unité de perçage de pièces métalliques peut être trouvée dans [81].

5.6 Conclusion du chapitre

Ce chapitre traite de la vérification d'un sous ensemble de *Full* RT-LOTOS et propose une traduction vers des réseaux de Petri temporels étendus avec des prédicats et des actions. Concernant les données exprimables en RT-LOTOS, la contribution de ce chapitre s'est limitée au traitement des entiers et des booléens (exprimés au moyen d'entiers). Les principales

difficultés à résoudre ont été d'identifier un bon modèle intermédiaire entre Full RT-LOTOS et les RdPTPA. Par ailleurs, l'étude de cas proposée dans la section 5.5 confirme l'intérêt de l'environnement *Tina* pour la vérification de *propriétés spécifiques* des descriptions RT-LOTOS.

Le modèle de composant proposé dans ce chapitre, devra maintenant évoluer pour offrir plus de flexibilité par rapport au schéma classique *(condition/action)* où une action n'est exécutée qu'après l'évaluation de la condition. A cet effet, le modèle de composant devra être étendu pour y introduire des mécanismes permettant de restreindre les valeurs qu'un composant reçoit sur ses points d'interaction et autoriser l'entrelacement des actions et des conditions.

6 Transposition au profil UML TURTLE

Les industriels montrent un intérêt croissant [31][18] pour l'intégration des techniques de description formelle dans leur processus de développement.

En dépit des qualités indéniables des langages de définition formelle de type RT-LOTOS, leur pénétration dans l'industrie reste limitée. L'un des obstacles demeure un apprentissage nécessitant un investissement certain. Les méthodes semi-formelles ont eu une fortune plus heureuse, favorisées en cela par des notations graphiques attrayantes et une sémantique moins contraignante, réduisant ainsi la distance entre les standards métier en vigueur dans l'industrie et les formalismes cibles pour la vérification.

L'objectif de ce chapitre est de montrer comment les travaux sur la traduction de l'algèbre de processus temporisée RT-LOTOS vers les RdPT et RdPTPA ainsi que l'utilisation de l'environnement *Tina* qui en découle, peuvent être transposés et réutilisés à des fins de vérification formelle de modèles de systèmes temps-réel exprimés dans le profil UML TURTLE (Timed UML and RT-LOTOS Environment [6]). Notons que les propositions de ce chapitre visent à substituer un traducteur TURTLE→RdPTPA au traducteur TURTLE →RT-LOTOS implanté dans l'outil *Ttool* et qui permettait jusqu'ici de vérifier des modèles TURTLE au moyen de l'outil *rtl*.

6.1 TURTLE : un profil *temps-réel* pour UML

Au travers de la notion de *profil*, les promoteurs d'UML à l'OMG [68] ont donné la possibilité de personnaliser le langage *généraliste* qu'est UML pour les besoins d'un domaine d'application particulier. Plusieurs profils *"UML temps réel"* ont ainsi vu le jour, au sein même de l'OMG (SPT [70] puis MARTE [69]) et dans plusieurs laboratoires de recherche (par exemple Accord/UML [2] ou OMEGA [67]).

TURTLE est également un profil UML temps-réel issu des laboratoires de recherche (LAAS-CNRS et ENSICA, puis ENST/SophiaAntipolis et Université Concordia à Montréal). Initialement centré sur la conception de systèmes temps-réel et la vérification formelle de modèles formés d'un diagramme de classes/objets décrivant la structure du système (architecture statique) et de diagrammes d'activités décrivant les comportements internes des objets (dynamique du système), le profil TURTLE a évolué pour prendre en compte les diagrammes de cas d'utilisation et de séquences nécessaires à couvrir la phase d'analyse d'un système temps-réel. Les développements les plus récents concernent le déploiement de composants sur des sites d'implantations représentées par les noeuds d'un diagramme de déploiement [7].

Dans le cadre de cette thèse, nous nous intéressons aux seuls diagrammes de conception c'est-à-dire aux diagrammes de classes/objets et aux diagrammes d'activité.

6.1.1 Diagramme de classes

Un diagramme de classes TURTLE se démarque d'un diagramme UML classique par l'ajout d'opérateurs de composition aux associations entre classes. Aux classes UML de base carac-

térisées par un nom, des attributs et des méthodes, le profil TURTLE ajoute des *Tclasses*[1] (cf. Figure 6.1) dotées d'attributs particuliers appelés *gates* qui sont autant des portes de communication. Ces *gates*, instances du type abstrait *Gate* sont bidirectionnelles.

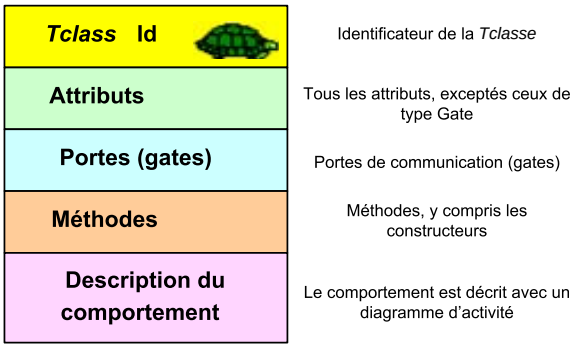
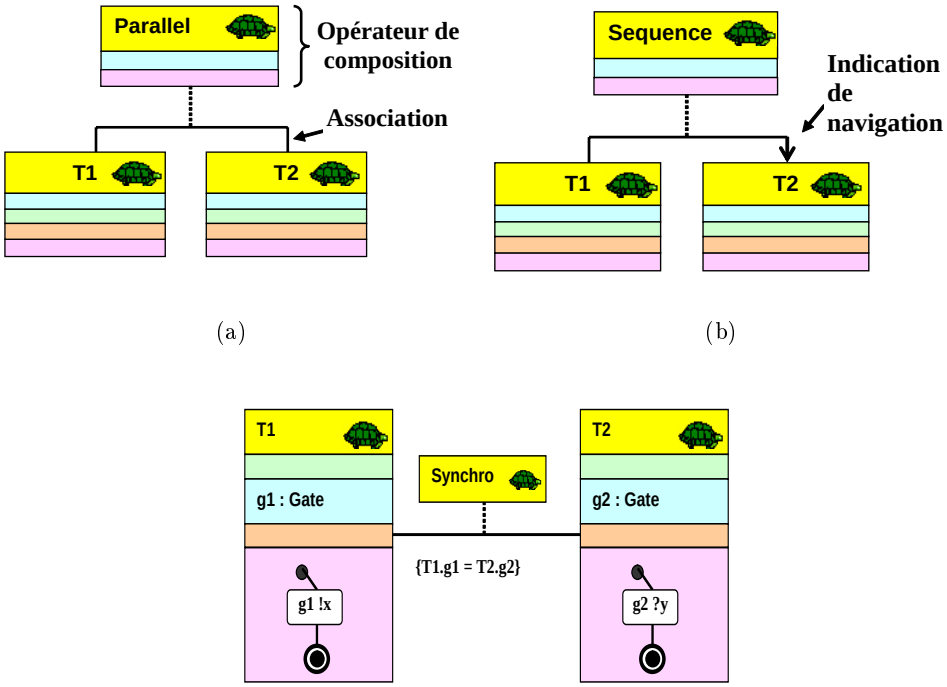


FIG. 6.1: Structure d'une *Tclasse*

TURTLE introduit aussi la notion d'opérateur de composition par l'entremise de classes associatives attachées aux associations (bipoints) entre *Tclasses*. Les opérateurs de TURTLE sont les suivants :

- ***Parallel*** (cf. Figure 6.2(a)) : Les deux *Tclasses* mises en relation par une association à laquelle est attribuée cet opérateur sont exécutées en parallèle et sans synchronisation
- ***Synchro*** (cf. Figure 6.2(c)) : Les deux *Tclasses* mises en relation par une association à laquelle est attribué cet opérateur réalisent des synchronisations entre elles dans deux flux d'exécution séparés. Cette synchronisation peut donner lieu à un échange de données dont le format est précisé dans le diagramme d'activité. Lorsque deux *Tclasses* doivent se synchroniser sur deux *gates,* ces dernières doivent être listées dans une formule *OCL* (*Object Constraint Language).* Par exemple, supposons que la porte g1 de la *Tclasse* T1 se synchronise avec la porte g2 de la *Tclasse* T2. Dans ce cas, la formule OCL qui accompagne la relation d'association entre T1 et T2 doit être {T1.g1 = T2.g2}. A chaque fois que T1 réalise une action sur g1, elle doit attendre que la classe T2 réalise une action sur g2 et réciproquement.
- ***Sequence*** (cf. Figure 6.2(b)) Les deux *Tclasses*, mises en relation par une association à laquelle est attribuée cet opérateur, sont exécutées l'une après l'autre, dans le sens donné par la navigation de l'association. T2 s'exécute à l'achèvement de T1.
- ***Preemption*** pour permettre à la *Tclasse* désignée par le sens de la navigation de l'association d'interrompre l'autre *Tclasse*.

[1] Les *Tclasses* sont construites par le mécanisme d'extension UML appelé *stéréotype*.



(a) (b)

(c) Synchronisation externe sur g1/g2

FIG. 6.2: Opérateurs de composition dans un diagramme de classes

6.1.2 Diagrammes d'activités

Toute *Tclasse* contient un diagramme d'activités modélisant son comportement. Un diagramme d'activité TURTLE (ou $\mathcal{AD}$) se démarque d'un diagramme d'activité UML 2.0 classique d'une part par la présence d'actions de synchronisation (avec possibilité de passage de valeurs) et des opérateurs temporels. Les objets TURTLE communiquent en effet par des offres de rendez-vous[2] à la LOTOS. Les opérateurs temporels (directement inspirés de ceux de RT-LOTOS) permettent d'exprimer un délai fixe, un délai non déterministe (*latence*) ou encore une offre de synchronisation limitée dans le temps. L'ensemble des constructions présentes dans les diagrammes d'activités TURTLE (accompagnés de leurs traduction en RDPTPA) est décrit dans les Figures 6.5, 6.6 et 6.7.

6.2 De TURTLE vers les RdPTPA

Comme pour les travaux de traduction de description RT-LOTOS présentés dans les chapitres précédents, le processus de traduction des modèles TURTLE nécessite de donner aux RdPTPA une capacité de modularité. A cet effet, nous reprenons dans ce chapitre le concept de composant RdPTPA présenté au chapitre 5.

[2]TURTLE supporte un rendez-vous à deux et pas un rendez-vous multiple.

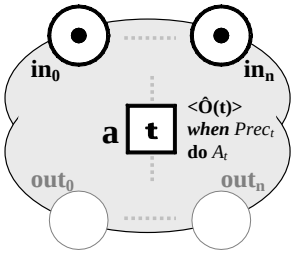


FIG. 6.3: Composant RdPTPA

Contrairement aux composants des chapitres précédents, les composants RdPTPA introduits dans ce chapitre peuvent avoir plusieurs places de sortie ($out_0, \ldots, out_n$) (cf. Figure 6.3).

La traduction des diagrammes TURTLE se passe en deux étapes :

1. Traduction des *Tclasses*.
   A chaque *Tclasse* correspond un composant RdPTPA, dont les transitions *observables* (non restreintes avec un intervalle temporel [0,0]) correspondent aux *gates* déclarées publiques. Les transitions *intériorisées* (auxquelles nous associons un intervalle temporel [0,0]) correspondent quant à elles aux *gates* déclarées privées. Les attributs (de type entier ou booléen) sont représentés par l'environnement (cf. Section 5.2) du RdPTPA.
2. Traduction des opérateurs de composition entre *Tclasses*.
   La composition entre *Tclasses* est traduite par une composition entre les composants RdPTPA construits lors de la phase précédente.

Exemple : L'exemple suivant décrit une *Tclasse Sender* dotée d'un attribut *data*, et des *gates* (*request*, *trasmits*, *ack*, *timeout*). Le comportement de la *Tclasse* est décrit par le diagramme d'activité de la Figure 6.4(b).

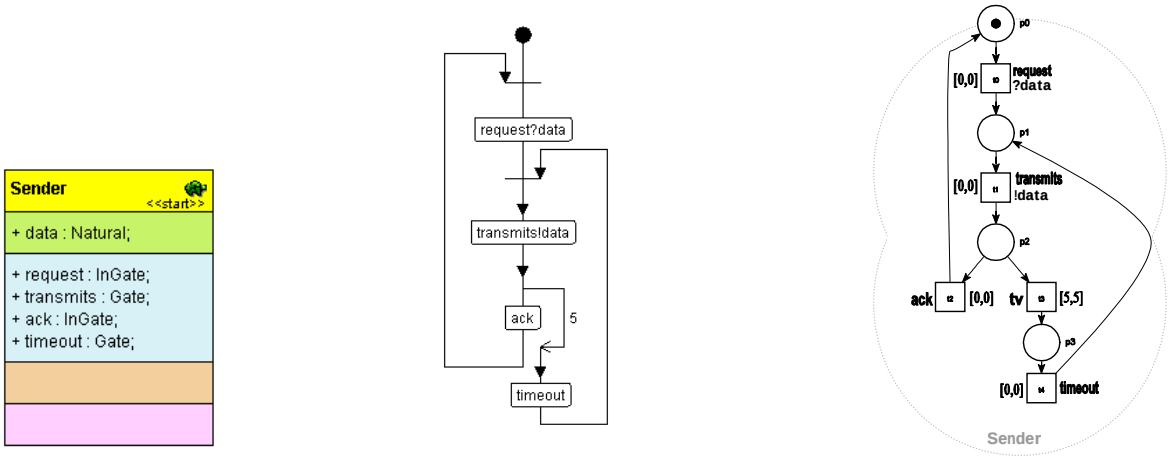
Un objet de type *Sender* reçoit un message de l'environnement et le transmet à son tour à un récepteur. Le comportement temporel d'un objet *Sender* est le suivant : après avoir reçu la donnée à transmettre de l'environnement et transmis le message, l'objet *Sender* attend un acquittement pendant 5 unités de temps (concept d'une offre limitée dans le temps). Une fois ce délai écoulé sans réception d'acquittement (concept de délai déterministe), l'objet déclare un *timeout* et retransmet le message.

Notons que le composant de la Figure 6.4(c) ne représente pas un système complet étant donné qu'il contient une offre de type *déclaration de variable* (*?data*) associée à la transition *request*.

6.2.1 Traduction des diagrammes d'activités TURTLE

Comme mentionné auparavant, chaque *Tclasse* contient un diagramme d'activité ($\mathcal{AD}$). Ce dernier combine actions de synchronisation et opérateurs temporels. Nous allons dans ce qui suit doter les composants RdPTPA des mêmes opérateurs que ceux présents dans TURTLE, ce qui va nous permettre de construire un réseau composite à partir de composants RdPTPA.

Dans ce qui suit, nous décrivons d'une manière inductive comment obtenir un nouveau composant à partir de composants correspondant aux diagrammes d'activités TURTLE ($\mathcal{AD}_1$,



(a) Classe Sender (b) Diagramme d'activité (c) Composant RdPTPA

FIG. 6.4: Diagrammes TURTLE et composant RdPTPA correspondant

$\mathcal{AD}_2$, ... $\mathcal{AD}_n$). Le composant résultant ($C_{O(\mathcal{T}(\mathcal{AD}_1),\mathcal{T}(\mathcal{AD}_2),...\mathcal{T}(\mathcal{AD}_n))}$) correspond au diagramme d'activité TURTLE obtenu par l'application de l'opérateur O aux opérandes. Ces derniers correspondent aux traduction des diagrammes d'activités TURTLE ($\mathcal{T}(\mathcal{AD}_1)$, $T(\mathcal{AD}_2)$, ... $\mathcal{T}(\mathcal{AD}_n)$).

Afin de simplifer la présentation des schémas de traduction, nous ne réprésentons qu'une seule place d'entrée sur le composant $C_{\mathcal{T}(\mathcal{AD})}$. Ce dernier est le composant sur lequel nous allons bâtir inductivement la traduction ($C_{\mathcal{T}(\mathcal{AD})}$ correspondant à la traduction du diagramme d'activité $\mathcal{AD}$).

6.2.2 Traduction des diagrammes de classes TURTLE

- **Traduction de l'opérateur *Parallel*** :(cf. Figure 6.5, ligne 8)

- **Traduction de l'opérateur *Synchro* :**(cf. Figure 6.6, ligne 1)

- **Opérateur de l'opérateur *Sequence* :** La Figure 6.8 décrit la composition séquentielle de $C_{\mathcal{T}(\mathcal{AD}1)}$ et $C_{\mathcal{T}(\mathcal{AD}2)}$ qui signifie que si $C_{\mathcal{T}(\mathcal{AD}1)}$ termine son exécution avec succès alors $C_{\mathcal{T}(\mathcal{AD}2)}$ sera exécuté. Nous distinguons deux cas :

1. Si $C_{\mathcal{T}(\mathcal{AD}1)}$ possède une seule place de sortie (i.e un seul point de terminaison possible) alors le composant correspondant à $C_{\mathcal{T}(\mathcal{AD}1)\gg\mathcal{T}(\mathcal{AD}2)}$ est obtenu en fusionnant l'interface de sortie de $C_{\mathcal{T}(\mathcal{AD}1)}$ avec l'interface d'entrée de $C_{\mathcal{T}(\mathcal{AD}2)}$ (voir la partie gauche de la Figure 6.8).
2. Dans le cas où plusieurs terminaisons sont possibles dans le composant $C_{\mathcal{T}(\mathcal{AD}1)}$, alors le composant résultant est obtenu comme suit : premièrement, nous rajoutons une ε-*transition* au composant $C_{\mathcal{T}(\mathcal{AD}2)}$ de telle sorte que l'ensemble des places de sortie de cette transition soit l'ensemble des places d'entrées du composant $C_{\mathcal{T}(\mathcal{AD}2)}$. Nous dupliquons ensuite cette ε-*transition* un nombre de fois égal au nombre de places de sorties dans $C_{\mathcal{T}(\mathcal{AD}1)}$. Pour finir, nous connectons chacune des ε-*transitions* ainsi crées aux places de sortie du composant $C_{\mathcal{T}(\mathcal{AD}2)}$ (cf. la partie droite de la Figure 6.8).

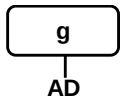
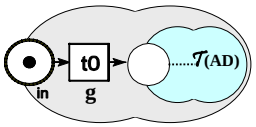
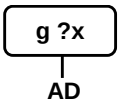
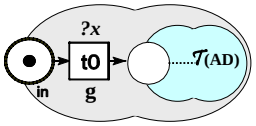
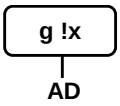
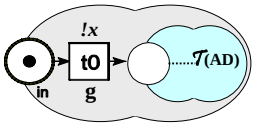
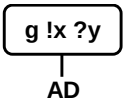
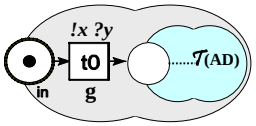
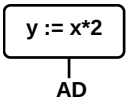
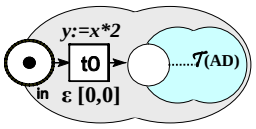
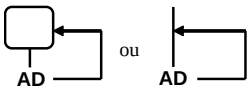
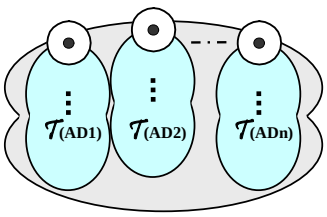
Diagramme d'activité TURTLE	Description	Traduction en RdPTPA
	Début d'un diagramme d'activité	
	Synchronisation sur la porte *g*	
	Synchronisation sur la porte *g* avec réception de la valeur *x*	
	Synchronisation sur la porte *g* avec émission de la valeur *x*	
	Appel sur la porte g avec émission de la valeur *x* et réception de la valeur *y*.	
	Assignation d'une valeur à un attribut	
	Structure itérative	
	Parallélisme entre *n* activités décrites par AD1, AD2 ...ADn	

FIG. 6.5: Opérateurs non temporels de TURTLE (1/2)

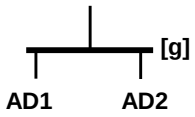
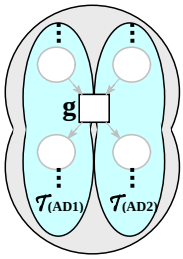
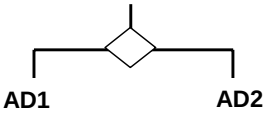
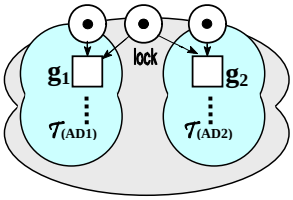
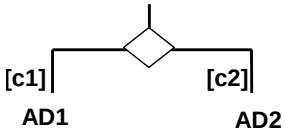
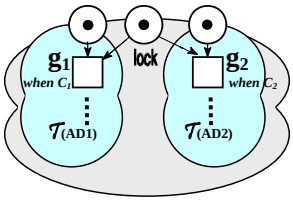
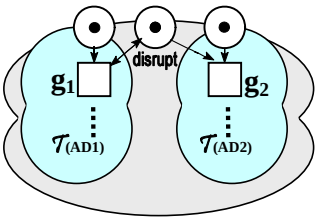
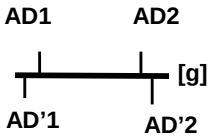
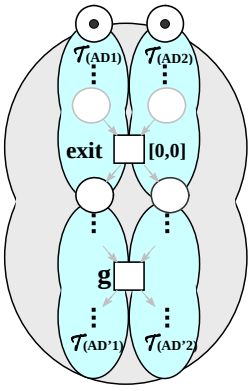
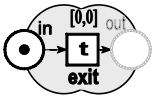
Diagramme d'activité TURTLE	Description	Traduction en RdPTPA
	Synchronisation sur la porte *g* entre deux activités décrites par AD1 et AD2.	
	Sélection de la première activité prête à s'exécuter entre deux activités décrites par AD1 et AD2	
	Sélection de la première activité prête à s'exécuter et dont la garde est vraie, entre les deux activités décrites par AD1 et AD2.	
	Préemption de l'activité décrite par AD1 par l'activité décrite par AD2.	
	Une fois que les deux activités décrites par AD1 et AD2 ont terminé leur exécution, les deux activités décrites par AD'1 et AD'2 sont exécutées en synchronisation sur la porte g.	
	Terminaison d'une activité	

FIG. 6.6: Opérateurs non temporels de TURTLE (2/2)

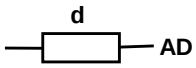
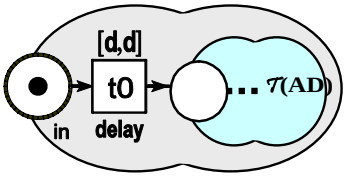
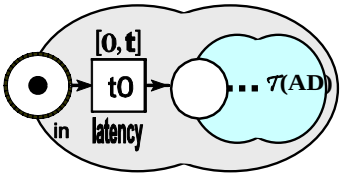
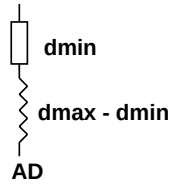
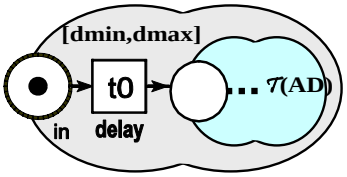
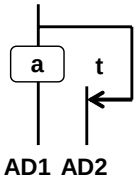
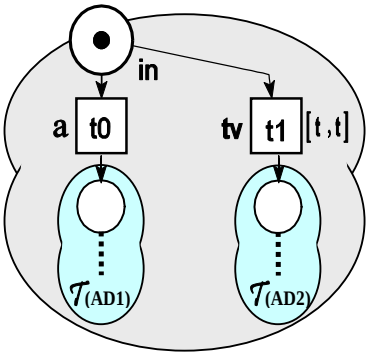
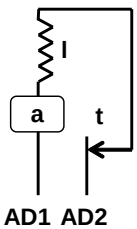
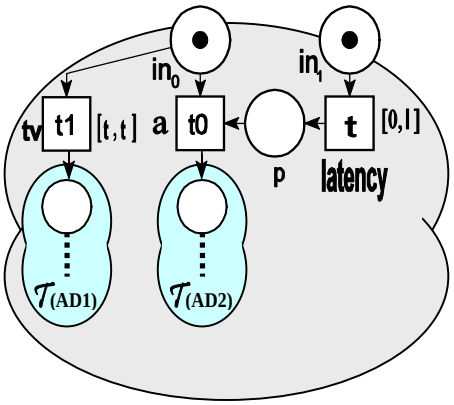
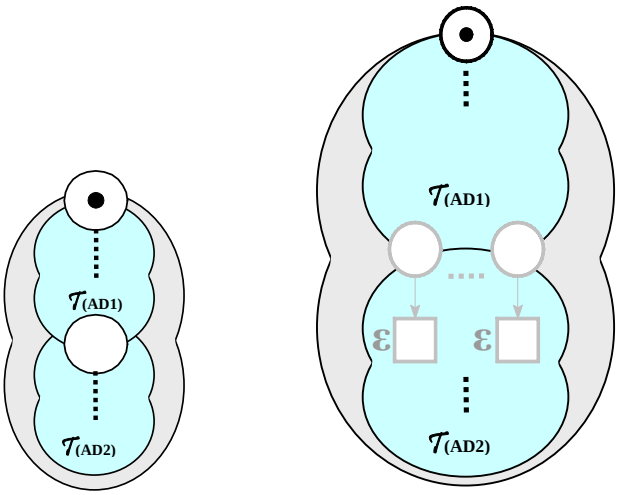
Opérateur temporel TURTLE	Description	Traduction en RdPTPA
	Retard déterministe de *d* unités de temps	
	Retard non déterministe entre 0 et *t* unités de temps	
	Retard non déterministe ente *dmin* et *dmax* unités de temps	
	L'action *a* est offerte pendant une période inférieure ou égale à t unités de temps. Si l'action *a* se produit, alors l'activité AD1 est exécutée, sinon AD2 est exécutée.	
	L'action *a* est offerte pendant une période inférieure ou égale à *t* unités de temps. Si l'action *a* se produit, alors l'activité AD1 est exécutée, sinon AD2 est exécutée. Notons que l'offre sur l'action *a* et le retard non déterministe démarrent en même temps	

FIG. 6.7: Opérateurs temporels de TURTLE



(a) (b)

FIG. 6.8: Traduction de l'opérateur de Séquence

– **Traduction de l'opérateur *Preemption* :** (cf. Figure 6.6 ligne 4).

6.3 Etude de cas

Système avionique simplifié

Nous empruntons à [22] une étude de cas issue du système de commandes de vol de l'Airbus A340. L'objectif des auteurs était de comparer différentes approches pour la vérification formelle de propriétés mixtes fonctionnelles et temps-réel au sein d'un même système embarqué. Nous ne nous intéressons pas ici à la vérification de telles propriétés. Notre but étant de générer l'espace des états du système spécifié avec TURTLE, en utilisant d'une part l'outil *rtl* et d'autre part *Tina*. Nous voulons en effet comparer les performances de ces deux approches respectives.

Le système en question asservit une gouverne en produisant périodiquement une commande à destination de celle-ci. De par son caractère critique, ce système est composé de trois fonctions redondantes, exécutées chacune sur un calculateur différent (cf. Figure 6.9).

– Une fonction maître F_R , périodique de période 20ms, exécutée sur le calculateur C_R et qui produit une commande CmdR. F_R est initialement en mode commande jusqu'à sa défaillance ;
– Une fonction secours F_L, périodique de période 20ms, exécutée sur le calculateur C_L et qui en cas de défaillance de F_R, produit la commande CmdL. F_R est considérée comme défaillante selon F_L, si celle-ci n'a reçu aucune commande CmdR pendant 2 cycles d'horloges (40ms). Dans ce cas F_L passe en mode commande (jusqu'à sa défaillance) et émet CmdL ;
– Une fonction de second secours F_B, périodique de période 20ms, exécutée sur le calculateur C_B et qui en cas de défaillance de F_R et de F_L, produit la commande CmdB. F_R et F_L sont considérées comme défaillantes, selon F_B, si celle-ci n'a pas reçu aucune commande CmdR ou CmdL pendant 5 cycles d'horloges (100ms). Dans ce cas F_B passe en mode commande et émet CmdB.

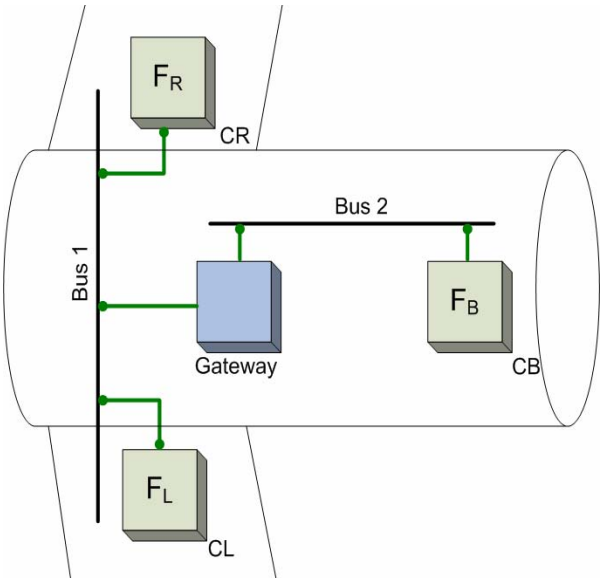


FIG. 6.9: Architecture simplifiée d'un système de commande

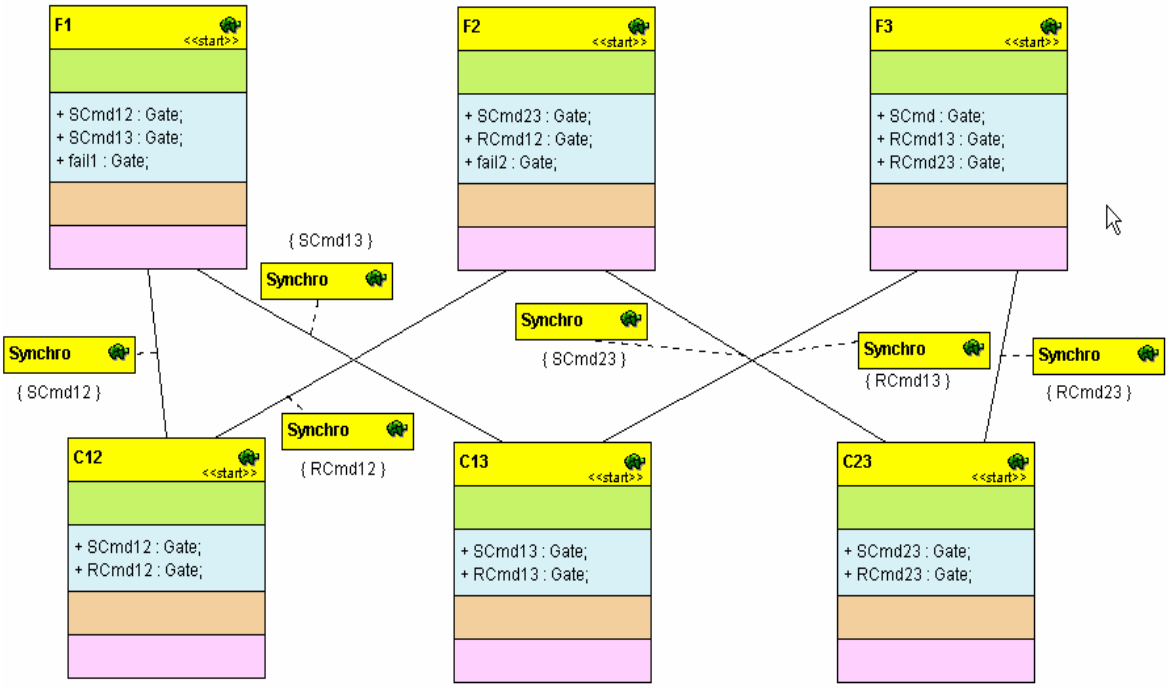
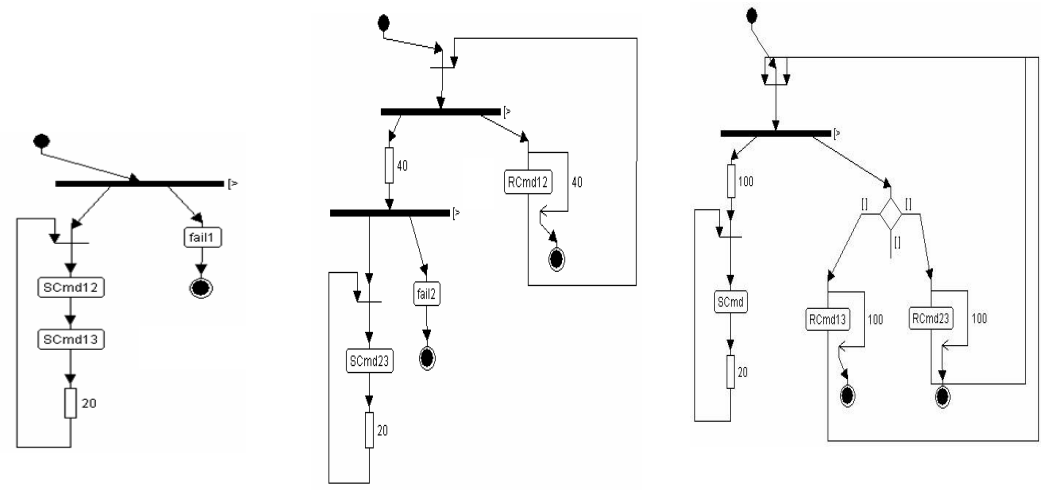


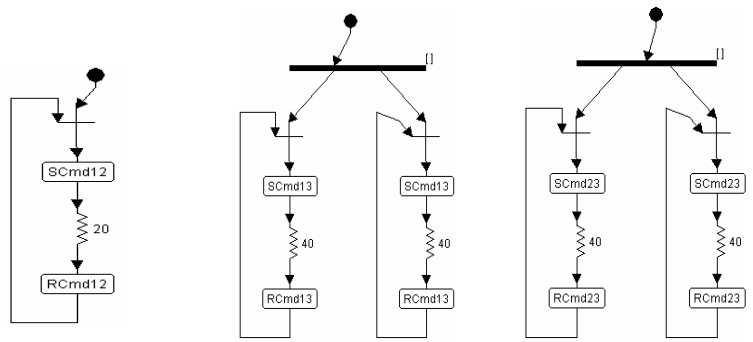
FIG. 6.10: Diagramme de classes du système



(a) F_R (b) F_L (c) F_B

FIG. 6.11: Diagrammes d'activités de F_R , F_L et F_B

Le système comporte aussi 3 canaux C_{RL}, C_{RB} et C_{LB} de latences respectives comprises dans les intervalles [0,20ms], [0,40ms] et [0,40ms]. Le canal C_{RL} peut stocker un seul message. Les canaux C_{RB} et C_{LB} peuvent en revanche en stocker deux.



(a) C_{RL} (b) C_{RB} (c) C_{LB}

FIG. 6.12: Diagrammes d'activités de C_{RL} , C_{RB} et C_{LB}

En appliquant les schémas de traductions présentés dans ce chapitre, nous dérivons un RdPT à partir du modèle TURTLE décrit dans les Figures 6.11 et 6.12.

Dans la table ci-dessous, nous comparons les résultats de la génération de l'espace des états pour le réseau obtenu en utilisant *Tina* avec l'approche antérieure à nos travaux qui consiste à passer par l'outil *rtl*.

Outils	Classes/Transitions	CPU(sec)
tina	566/ 1447	<1
rtl	104/ 152	430

TAB. 6.1: Résultats de la génération de l'espace des états

Ces résultats expérimentaux ont été obtenus sur une machine dotée de 512 Mo de mémoire et d'un processeur cadencé à 1600Mhz.

Extension1 : Une première extension du système consiste à augmenter les latences entre les fonctions. Soit un système S_2 composé de 3 fonctions F_1, F_2 et F_3 organisées selon le principe de redondance précédent, et tel que la latence de communication entre deux fonctions F_j et $F_i(j < i)$ est comprise entre 0 et L cycles.

	tina		*rtl*	
L	Classes/ Transitions	CPU	Classes/ Transitions	CPU
[0,1]	346/688	<1s	100/143	430s
[0,2]	1373/4096	<1s	104/152	430s
[0,5]	3208/10617	<1s	142/208	430s
[0,9]	5850/20641	<1s	146/214	430s

TAB. 6.3: Résultats pour l'extension 1

Les résultats de la génération de l'espace des états pour cette première extension sont données par la Table 6.3. Nous pouvons observer que l'augmentation du domaine de la latence a pour effet d'accroître la taille du graphe de classes généré par *Tina*. Néanmoins le temps de génération de l'espace des états reste négligeable. Pour *rtl*, le temps de génération reste le même et le nombre de classes ne croit pas beaucoup, ce qui est dû à l'algorithme de minimisation implanté dans rtl est adapté de [87] . La taille du graphe de région généré par *rtl* dépend directement du nombre d'horloges présentes dans le système ; or le fait d'élargir le domaine de la latence ne rajoute pas d'horloges supplémentaires.

Extension 2 : Une deuxième extension consiste à augmenter le nombre de fonctions du système. Considérons un système $S_3(n)$ composé de N fonctions F_1, $F_2 \ldots F_n$de période 20 ms. Toutes les fonctions ont la même période et tous les canaux peuvent stocker un et un seul message. La latence de communication entre deux fonctions F_j et (F_i $j < i$) est comprise entre 0 et 20 ms.

	Tina		*rtl*	
N	Classes/ Transitions	CPU	Classes/ Transitions	CPU
2	19/26	<1s	9/11	<1s
3	264/508	<1s	85 /148	3s
4	2064/4984	<1s	340/736	118mn
5	640 017/2 819 379	428s	?	>24h

TAB. 6.5: Résultats pour l'extension 2

Les résultats de la Table 6.5 pour cette deuxième extension confirment la robustesse de *Tina* face à une situation d'explosion combinatoire et montrent clairement que le passage par un modèle de type RdPT est avantageux en termes d'efficacité de la génération du graphe d'états. L'intérêt de l'approche proposée dans ce chapitre se voit confirmé.

6.4 Conclusion du chapitre

Le fait de doter le profil UML temps-réel TURTLE d'une sémantique formelle par traduction vers l'algèbre de processus temporisée RT-LOTOS a permis dans un premier temps de réutiliser au bénéfice de TURTLE l'outil de vérification formelle *rtl* développé au LAAS. Le profil TURTLE bénéficie désormais de la transposition des travaux effectués dans cette thèse et portant sur la traduction de spécifications RT-LOTOS vers les RdPTPA. Ainsi les modèles TURTLE peuvent maintenant être traduits en RdPTPA et être rigoureusement analysés avec l'environnement *Tina*. L'implantation complète de l'approche amènera à interfacer *Ttool* et l'environnement *Tina*.

7 Conclusion générale

Le recours aux techniques de définition formelle et aux algèbres de processus temporisées se justifie pleinement dès lors que l'on place la vérification formelle au cœur de la conception amont des systèmes temps réel. Dans cette optique, l'objectif de cette thèse a été la mise en œuvre d'une nouvelle approche pour la vérification des spécifications RT-LOTOS, l'approche existante ayant montré des limites en termes de performances et de type de propriétés vérifiées. A cet effet, nous avons proposé une traduction vers différentes classes de réseaux de Petri temporels (RdPT), des modèles intermédiaires, ainsi que des schémas de traduction permettant de traiter un sous ensemble du langage RT-LOTOS centré sur la partie *contrôle* avec une ouverture sur la partie *données*. Nos travaux ont ouvert la voie à l'utilisation de l'environnement *Tina* qui a montré son adéquation pour la vérification de propriétés générales et spécifiques des descriptions RT-LOTOS, ainsi que sa robustesse face à des situations d'explosion combinatoire. Les résultats expérimentaux confirment l'avantage de passer par les réseaux temporels comme modèle intermédiaire par rapport à l'approche précédente (outillée par *rtl*) qui consistait à compiler une spécification RT-LOTOS en une classe d'automates temporisés.

7.1 Bilan des contributions

Les contributions de cette thèse ont été développées au travers des chapitres 3 à 6 de ce mémoire.

Traduction de la partie *contrôle* de RT-LOTOS Ce point constitue l'objectif principal de cette thèse. La traduction se limite aux comportements RT-LOTOS dits réguliers i.e. représentables par des réseaux temporels finis. Ce sous-ensemble de RT-LOTOS (caractérisable par des restrictions quant à l'emploi de la récursion) s'avère suffisant pour couvrir une large gamme de systèmes temps-réel pratiques. Pour chaque opérateur RT-LOTOS, un schéma de traduction a été défini de manière complètement formelle et non ambiguë. La définition formelle est accompagnée de graphiques donnant l'intuition de chacun des schémas de traduction. Les points délicats ainsi que l'apport des schémas proposés par rapport à des approches similaires ont été détaillés dans un style incrémental. La formalisation de ces schémas confère à RT-LOTOS une sémantique en termes de RdPT et le caractère graphique des RdPT a clarifié l'usage de certains opérateurs RT-LOTOS tels que la *latence*. Dans un souci de rigueur, ces schémas ont été accompagnés à la fois d'une preuve formelle de correction et d'une procédure de validation expérimentale. Cette dernière est basée sur la comparaison des espaces des états des comportements RT-LOTOS et des RdPT correspondants. La preuve formelle, quant à elle, établit une équivalence forte (bissimulation temporelle) entre les termes RT-LOTOS réguliers et les RdPT 1-bornés. L'obtention de ce résultat de la préservation de la sémantique RT-LOTOS a été possible parce que nous avons soigneusement évité l'introduction de toute transition auxiliaire dans le graphe de classes des RdPT obtenus. Ainsi, nous nous assurons qu'une exécution du système spécifié peut être vue indifféremment comme une suite d'actions

RT-LOTOS ou une séquence de tir dans un RdPT. Un bénéfice immédiat est que les résultats d'analyse au niveau du réseau temporel sont facilement transposables au niveau de la spécification RT-LOTOS.

D'autre part, et pour mener à bien ce processus de traduction, il nous aura fallu compenser une limitation des RdPT en matière de composition de comportements. Les RdPT sont notoirement connus pour n'offrir « en natif » aucun mécanisme pour composer (ou décomposer) de grands réseaux à partir (ou en petits) réseaux. Ce constat nous a amené à augmenter le modèle des RdPT pour supporter la notion de composant. Une contribution clé réside dans la façon dont nous avons solutionné le problème de synchronisation temporelle dans les RdPT. Dans notre proposition de cadre pour la composition des RdPT, la synchronisation entre composants est obtenue sans manipulation explicite de l'information temporelle. Les composants RdPT gardent ainsi leur indépendance temporelle. Cette propriété est essentielle dans le cadre d'une approche compositionnelle comme la notre.

Pour finir, ces schémas de traduction ont été implémentés dans *rtl2tpn*, un prototype qui réutilise une partie du code de *rtl*. *rtl2tpn* prend en entrée une description RT-LOTOS et génère en sortie un RdPT équivalent. Ceci a ouvert la voie à l'intégration de l'environnement *Tina* à notre plateforme de vérification. L'outil doit être rendu plus robuste en s'affranchissant du code *rtl*, et des expérimentations doivent porter sur une gamme plus conséquente d'exemples.

Extension de RT-LOTOS pour la vérification de systèmes préemptifs Dans le cadre de cette thèse nous avons considéré RT-LOTOS comme une donnée immuable du problème, d'autant plus que les choix des concepteurs de RT-LOTOS nous paraissent théoriquement consistants et particulièrement bien motivés. Ce point constitue donc la seule proposition d'amélioration du langage RT-LOTOS, et donc l'unique transgression à la règle ni critique/ni suggestion pour RT-LOTOS. Nous avons dans un premier temps montré que le langage RT-LOTOS n'est pas adapté pour décrire des situations où l'exécution d'un système est interrompue puis reprise sans perte du contexte temporel. Nous avons alors proposé d'enrichir RT-LOTOS par un opérateur de suspension/reprise dont nous avons illustré l'utilité sur différents exemples. Nous avons aussi montré que la relation d'équivalence forte est une congruence pour ce dernier. Cependant, la richesse d'expressivité induite n'autorisait plus l'utilisation des RdPT pour la vérification des descriptions RT-LOTOS ainsi étendu. Nous avons alors proposé une traduction du langage RT-LOTOS étendu vers une classe de RdPT nouvellement introduite : les réseaux de Petri temporels à chronomètres, tels que les supporte la dernière version de l'environnement *Tina*. Cependant la vérification de tels systèmes n'est pas décidable ; en particulier, la bornitude des RdPTC n'implique pas le caractère fini du graphe de classes. Par conséquent, le recours à des méthodes d'approximations de l'espace des états s'avère parfois nécessaire. De telles approximations suffisent pour la vérification de propriétés de sûreté.

Traitement des données Nous avons étendu notre approche pour y inclure une partie *données*. Ce travail à nécessité la formalisation d'un nouveau modèle de réseaux de Petri temporels à prédicats et actions (RdPTPA), supporté par la dernière version de *Tina* sous la forme de Systèmes de Transitions Étiquetés (composés de deux parties : la partie contrôle définie par un RdPT et la partie données définie par un ensemble de fonctions C). Nous nous sommes appuyé sur les solutions développées pour la traduction de la partie contrôle de RT-LOTOS pour proposer des mécanismes pour la prise en compte des données. Nous avons par la même étendu le pouvoir de modélisation des composants introduits dans la première partie

de cette thèse, on y ajoutant des variables, des actions, des offres et des *ε-transitions*. Notons tout de même que notre contribution reste limitée aux entiers et aux booléens.

Transpositon de l'approche pour la vérification de modèles UML temps-réel Dans un effort d'ouverture vers les standards métier en vigueur dans l'industrie, nous avons montré comment les travaux sur la traduction de RT-LOTOS vers les RdPT et RdPTPA ainsi que l'utilisation de l'environnement *Tina* qui en découle, peuvent être transposés et réutilisés à des fins de vérification formelle de modèles de systèmes *temps-réel* exprimés dans le profil UML TURTLE. Ce dernier bénéficie désormais de la transposition des travaux effectués dans cette thèse. Les modèles TURTLE peuvent maintenant être traduits en RdPTPA et être ainsi rigoureusement analysés avec l'environnement Tina.

7.2 Perspectives

Pour clôturer cette étude, abordons maintenant les points qui nous semblent être des prolongements possibles et souhaitables à nos travaux.

L'enrichissement de RT-LOTOS pour l'analyse d'ordonnancement L'opérateur *Suspend/Resume* proposé dans cette thèse, apporte les qualités nécessaires pour l'utilisation de RT-LOTOS pour la spécification d'ordonnançeurs. Cependant, une technique de spécification basée sur l'emploi de l'opérateur *Suspend/Resume* seul n'est pas totalement satisfaisante. En effet, en dépit des qualités intrinsèques de RT-LOTOS, ce dernier reste un langage à spectre large et ne permet pas de modéliser des mécanismes tels que le partage de ressources ou l'assignation de priorités aux processus. Dans ce cadre, une extension de RT-LOTOS avec de tels mécanismes permettrait la spécification de politiques d'ordonnancement plus complexes et ainsi de traiter des exemples de taille plus conséquente. A plus court terme, une évolution souhaitée de l'opérateur *Suspend/Resume* serait de donner la priorité aux actions suspensives sur les actions ordinaires. Pour les besoins de l'analyse, une traduction sur la base du schéma présenté dans cette thèse vers un formalisme comme les PrTPN [16] (RdPT étendus avec des priorités) se substituerait à la traduction actuelle vers les RdPTC. La piste d'une traduction vers une classe qui reste à identifier des *scheduling*-TPN [78] ouvrirait quant à elle la voie à l'utilisation de l'outil *Romeo* [40] pour l'analyse d'ordonnançabilité, ainsi qu'à la vérification de propriétés quantitatives.

Un modèle intermédiaire adéquat pour la prise en compte des données Bien que le modèle intermédiaire des composants ait été introduit avec l'idée de structurer les RdPT lors de leurs générations à partir de spécifications RT-LOTOS, cette notion de composant reste assez générale pour être reprise dans d'autres contextes et pour d'autres langages de haut niveau. Dans l'immédiat, les patterns de traduction présentés dans ce chapitre sont facilement adaptables à d'autres extensions temporelles de LOTOS, en particulier ET-LOTOS [56]. RT-LOTOS se démarquant essentiellement de ces extensions par son opérateur de latence. Pour sa part, le modèle de composant avec données, devra maintenant évoluer pour offrir plus de flexibilité par rapport au schéma classique (condition/action) où une action n'est exécutée qu'après l'évaluation de la condition. A cet effet, le modèle de composant devra être étendu pour y introduire des mécanismes permettant de restreindre les valeurs qu'un composant reçoit sur ses points d'interaction, autoriser l'entrelacement des actions et des conditions, ainsi que

la mise en œuvre de techniques pour la réduction de l'espace des états comme par exemple la remise-à-zero des variables implantée par l'outil Caesar[38]. L'on peut légitimement se poser la question du coût de telles évolutions en terme de transitions auxiliaires dans le graphe de classes du réseau sous-jacent. D'autre part, la question de l'emploi de types de données algébriques dans le cadre de RT-LOTOS doit être reconsidérée. La disponibilité de l'outil Caesar.ADT [35] qui traduit automatiquement les types abstraits de LOTOS en langage C est un argument solide en faveur de la réhabilitation des types de données algébriques. En tout état de cause, les limitations des modèles condition/action sont discutées dans [37], et une réflexion plus approfondie doit être menée afin d'identifier le meilleur modèle intermédiaire pour un traitement efficace des données.

Conception *top-down* des systèmes temps-réel Abordons maintenant la vérification de modèles UML temps-réel «TURTLE». Notre approche peut naturellement évoluer vers une méthodologie de conception top-down des systèmes temps-réel. En effet, au plus haut niveau nous utilisons des modèles TURTLE pour décrire l'architecture du système ainsi que la communication entre les modules constituants. A un niveau plus bas, nous obtenons un RdPTPA qui décrit le comportement opérationnel du système. L'implantation complète de l'approche amènera à interfacer Ttool et l'environnement *Tina*, offrant ainsi une plateforme de validation dans laquelle les acquis de nos travaux entreront en synergie avec d'autres travaux sur l'ingénierie des exigences temporelles et la génération de séquences de test temporisées. L'utilisation de notre environnement de vérification formelle suppose que l'on ait clairement défini les exigences temporelles auxquelles l'on entend confronter une conception. Dans le cas d'une conception TURTLE, l'on pourra se reporter au traitement d'exigences temporelles proposé dans [32]. Les exigences étant formalisées, la vérification est un premier filtre d'erreurs qui ne dispense pas de tester les systèmes temps réel produits sur la base modèles TURTLE ou RT-LOTOS. Notons que [3] propose une approche de génération de séquences de tests pour les RdPTC et qu'il y a ainsi complémentarité avec nos travaux.

La vérification compositionnelle pour faire face à l'explosion combinatoire En dépit des acquis de la théorie des réseaux temporels, l'explosion de la taille du graphe de classes (à cause des contraintes temporelles ou pas) est une limite intrinsèque des méthodes énumératives utilisées dans le cadre de cette thèse. A cet effet, des techniques de sur-approximation du comportement temporel par le comportement a-temporel ont été explorées pour s'affranchir de la construction complète de l'espace des états du comportement temporel [77]. Pour faire face au problème d'explosion combinatoire, la tendance actuelle dans la communauté logicielle est d'exploiter la structure modulaire naturellement présente dans plusieurs systèmes à des fins de vérification compositionnelle. Les propriétés du système sont ainsi décomposées en propriété de ses composants et chaque composant est vérifié séparément dans un style «*assume-guarantee*», où un composant garantit de se comporter correctement s'il reçoit à son interface la séquence d'entrée supposée. Le modèle de composant introduit dans cette thèse confère aux réseaux temporels une capacité modulaire. Ce qui laisse à penser qu'un raisonnement compositionnel est également applicable à ces derniers, qu'ils soient engendrés à partir de spécifications RT-LOTOS ou pas. Cependant, l'une des limites d'une telle approche réside dans la difficulté de la génération des suppositions ou *"assumptions"* de l'environnement. Cette étape est traditionnellement effectuée à la main, ce qui constitue un effort non trivial. L'objectif final est le développement de techniques dites "*presse bouton*", en réduisant autant

que possible l'intervention de l'utilisateur. Et une avancée prometteuse dans cette direction est basée sur l'apprentissage via l'utilisation d'algorithmes pour la découverte automatique de suppositions. Une perspective de ce travail consiste dans l'adaptation des techniques utilisées pour les *"learning automata"* [63] dans le cadre des RdPT afin d'automatiser le processus de déduction des suppositions. L'apprentissage dans ce contexte produit les suppositions et les modifient en utilisant des contre-exemples obtenus par la vérification séparée des composants. Nous pensons que l'apprentissage automatique serait une bonne contribution pour la vérification formelle des réseaux de Petri temporels en général.

8 Publications de l'auteur

Revues avec comité de lecture

1. *T. Sadani, P. de Saqui-Sannes, J.-P. Courtiat*, Validation de spécifications RT-LOTOS : Une interface vers l'outil Tina, *Journal Européen des Systèmes Automatisés (JESA)*, Vol.39 N°1-2-3, pp.271-286, 2005.

Conférences internationales avec actes et comité de lecture

1. *T. Sadani, P. de Saqui-Sannes,* Une approche efficace pour le traitement des données dans RT-LOTOS, *Modélisation des systèmes réactifs (MSR'07).* Lyon, France, octobre 2007.
2. *T. Sadani, P. de Saqui-Sannes, J.-P. Courtiat*, Extended Real-Time LOTOS for preemptive Systems Verification, 1*5th International Conference on Real-Time and Network Systems (RTNS'07)*, Nancy, France, March 2007.
3. *T. Sadani, M.Boyer, P. de Saqui-Sannes, J.-P. Courtiat,* Mapping RT-LOTOS specifications into Time Petri Nets, *8th International Conference on Formal Engineering Methods (ICFEM 06)*, pp.360-379, Macao SAR, China, November 2006.
4. *T. Sadani, M. Boyer, P. de Saqui-Sannes, J.-P. Courtiat, Effective Representation of RT-LOTOS Terms by Finite Time Petri Nets, 26th IFIP WG 6.1 International Conference on Formal Methods for Networked and Distributed Systems (FORTE 06),* pp.404-419, Paris, France, September 2006.
5. *T. Sadani, P. de Saqui-Sannes, J.-P. Courtiat,* Recherche d'efficacité en vérification de modèles UML temps réel traduits en RT-LOTOS, *Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL'06).* Paris, France, mars 2006.
6. *T. Sadani, P. de Saqui-Sannes, J.-P. Courtiat,* Formal and Efficient Verification Techniques for Real-Time UML models, *3rd European Congress in Embedded Real Time Software (ERTS'06),* Toulouse, France, January 2006.
7. *T. Sadani, J.-P. Courtiat, P. de Saqui-Sannes,* From RT-LOTOS to Time Petri Nets : New Foundations for a Verification Platform*, 3rd IEEE International Conference on Software Engineering and Formal Methods (SEFM'05)* Koblenz, Germany, pp.250-260, September 2005.
8. *T. Sadani, P. de Saqui-Sannes, J.-P. Courtiat,* Formal Validation of RT-LOTOS Specifications : New Directions and Preliminary Results, *25th IEEE International Real-Time Systems Symposium (RTSS'04) Work in Progress Session,* Lisbon, Portugal, December 2004.

Conférences nationales avec comité de lecture

1. *T. Sadani, P. de Saqui-Sannes, J.-P. Courtiat, M. Boyer,* Extensions de RT-LOTOS pour la spécification et vérification de systèmes préemptifs, *Formalisation des Activités Concurrentes (FAC'06),* Toulouse, France, mars 2006.

2. *T. Sadani, J.-P. Courtiat, P. de Saqui-Sannes,* Formal Validation of RT-LOTOS Specifications using the Tina tool*, Formalisation des Activités Concurrentes (FAC'05),* Toulouse, France, mars 2005.

Bibliographie

[1] Y. Abdeddaim and O. Maler. Preemtive job-shop scheduling using stopwatch automata. In *Proc. of TACAS'02*, pages 113–126.

[2] Accord/UML. Accord/uml home page. http ://www-list.cea.fr/labos/fr/LLSP/accordUml/.

[3] N. Adjir, P. de Saqui-Sannes, and K.M. Rahmouni. Génération des séquences de test temporisées à partir des réseaux de petri temporels à chronomètres. In *Conférence internationale sur les nouvelles technologies de la répartition, NOTERE*, 2007.

[4] G. Ailloud. Verification in ECRINS of LOTOS programs. ESPRIT/SEDOS/C2/N89.2, November 1986.

[5] R. Alur, C. Courcoubetis, N. Hallbwachs, T. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. 138 :3–34, 1995.

[6] L. Apvrille, J.-P. Courtiat, C. Lohr, and P. de Saqui-Sannes. TURTLE : A real-time UML profile supported by a formal validation toolkit. *IEEE Transactions on Software Engineering*, 30(4), July 2004.

[7] L. Apvrille, P. de Saqui-Sannes, R. Pacalet, and A. Apvrille. Un environnement de conception de systèmes distribués basé sur uml. *Annals of Telecommunications*, 61 :1347–1368, 2006.

[8] J.C. Baeten and J.A. Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3(2) :142–188, 1991.

[9] M. Barbeau and G. Von Bochmann. Verification of LOTOS specifications : A Petri net based approach. In *Proc. of Canadian Conf. on Electrical and Computer Engineering*, 1990.

[10] M. Barbeau and G. Von Bochmann. Extension of the Karp and Miller procedure to LOTOS specifications. *Discrete Mathematics and Theoretical Computer Science*, 3 :103–119, 1991.

[11] M. Barbeau and G. Von Bochmann. A subset of LOTOS with the computational power of place/transition-nets. In *Proc. of the 14th Int. Conf. on Application and Theory of Petri Nets (ICATPN)*, volume 691 of *LNCS*, 1993.

[12] H. Ben-Abdallah, J. y. Choi, D. Clarke, Y.-S. Kim, I. Lee, and H.-L. Xie. A process algebraic approach to the schedulability analysis of real time systems. 15(3) :189–219, 1998.

[13] J. Bengtsson, K.G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Uppaal - a tool suite for automatic verification of real-time systems. In *Proc of the 4th DIMACS Workshop on Verification and Control of Hybrid Systems*, number 1066 in LNCS, pages 232–243, October 1995.

[14] B. Berthomieu, D. Lime, O.H. Roux, and F. Vernadat. Reachability problems and abstract state spaces for time petri nets with stopwatches. In *To appear*, 2007.

[15] B. Berthomieu and M. Menasche. Une approche par énumération pour l'analyse des réseaux de Petri temporels. In *Actes de la conférence IFIP'83*, pages 71–77, 1983.

[16] B. Berthomieu, F. Peres, and F. Vernadat. Bridging the gap between timed automata and bounded time petri nets. In *Formal Modeling and Analysis of Timed Systems, 4th International Conference, FORMATS 2006, Paris, France*, volume 4202. Springer, 2006.

[17] B. Berthomieu, P.-O. Ribet, and F. Vernadat. The TINA tool : Construction of abstract state space for Petri nets and time Petri nets. *Int. Journal of Production Research*, 42(14), 2004.

[18] B. Berthomieu, P.-O. Ribet, F. Vernadat, J. Bernartt, J.-M. Farines, J.-P. Bodeveix, M. Filali, G. Padiou, P. Michel, P. Farail, P. Gauffilet, P. Dissaux, and J. Lambert. Towards the verification of real-time systems in avionics : the cotre approach. In *Proc. of FMICS'2003*, pages 201–216.

[19] B. Berthomieu and F. Vernadat. State class constructions for branching analysis of time Petri nets,. In *Proc. of TACAS 2003*, number 2619 in LNCS, 2003.

[20] E. Best, R. Devillers, and M. Koutny. *Petri Net Algebra*. Monographs in Theoretical Computer Science : An EATCS Series. Springer-Verlag, 2001. ISBN : 3-540-67398-9.

[21] T. Bolognesi, F. Lucidi, and S. Trigila. From timed Petri nets to timed LOTOS. In *Protocol Specification, Testing and Verification X (PSTV), Proceedings of the IFIP WG6.1 Tenth International Symposium on Protocol*, pages 395–408, 1990.

[22] F. Boniol, G. Bel, and J. Ermont. Trois approches pour la modélisation et la vérification de systèmes embarqués. *Technique et science informatique*, 2003.

[23] G. Bucci, A. Fedeli, L. Sassoli, and E. Vicario. Time state space analysis of real-time preemptive systems. 30(2) :97–111, 2004.

[24] F. Cassez and K.G. Larsen. The impressive power of stopwatches. In *11th Int. Conf. on Concurrency Theory*, volume 1877 of *LNCS*, pages 138–152, University Park, P.A, USA, 2000. Springer-Verlag.

[25] S. Chaki, E.M. Clarke, J. Ouaknine, N. Sharygina, and N. Sinha. State/event-based software model checking. In *IFM*, pages 128–147, 2004.

[26] J.-P. Courtiat. Formal design of interactive multimedia documents. In A.Wolisz H.Konig, M.Heiner, editor, *Proc. of 23rd IFIP WG 6.1 Int Conf on Formal Techniques for Networked and distributed systems (FORTE'2003)*, volume 2767 of *LNCS*, 2003.

[27] J.-P. Courtiat and R.C. De Oliveira. A reachability analysis of RT-LOTOS specifications. In *Proc. of Eighth International Conference on Formal Description Techniques Protocol (FORTE)*, Montreal, Canada, 1995. Chapman & Hall.

[28] J.-P. Courtiat, C.A.S. Santos, C. Lohr, and B. Outtaj. Experience with RT-LOTOS, a temporal extension of the LOTOS formal description technique. *Computer Communications*, 23(12), 2000.

[29] M. Diaz and P. Sénac. Time stream petri nets : A model for timed multimedia information. In *Proceedings of the 15th International Conference on Application and Theory of Petri Nets*, pages 219–238. Springer-Verlag, 1994.

[30] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1*. Springer, Berlin, FRG, 1985.

[31] P. Farail, P. Gauffilet, M. Filali, P. Michel, and F. Vernadat. Vérifications dans un agl orienté modèles. *Génie Logiciel*, 69b :51–55, 2004.

[32] B. Fontan, L. Apvrille, P. de Saqui-Sannes, and J.-P. Courtiat. Real-time and embedded system verification based on formal requirements. In *IEEE Symposium on Industrial Embedded Systems*, 2006.

[33] A.N. Fredette and R. Cleaveland. Rtsl : A language for real-time schedulability analysis. In *Proc. of the Real-Time Systems Symposium*, pages 274–283, Durham, North Carolina, 1993. Computer Society Press.

[34] H. Garavel. *Compilation et vérification de programme LOTOS*. PhD thesis, Université Joseph Fourier – Grenoble I, 1989.

[35] H. Garavel. Compilation of LOTOS abstract data types. In S. T. Vuong, editor, *Formal Description Techniques II*, pages 147–162. Elsevier, 1990.

[36] H. Garavel. Défense et illustration des algèbres de processus. In *Actes de l'Ecole d'été Temps Réel ETR*, 2003.

[37] H. Garavel and F. Lang. NTIF : A general symbolic model for communicating sequential processes with data. In *IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE), LNCS*, volume 22, 2002.

[38] H. Garavel, F. Lang, and R. Mateescu. An overview of cadp 2001. *European Association for software science and technology (EASST) Newsletter*, 4, August 2002.

[39] H. Garavel and M. Sighireanu. A proposal for coroutines an suspend/resume in e-lotos. Technical Report (1.21.20.2.3), Input document to the ISO/IEC JTC1/SC21/WG7 Meeting on Enhancements to LOTOS, 1996.

[40] G. Gardey, D. Lime, M. Magnin, and O.H. Roux. Romeo : A tool for analyzing time petri nets. In *Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings*, volume 3576. Springer, 2005.

[41] U. Goltz. On representing CCS programs by finite Petri nets. In *Proc. of Int. Conf. on Math. Foundations of Computer Science*, volume 324 of *LNCS*, 1988.

[42] J. Hatcliff and M.B. Dwyer. Using the bandera tool set to model-check properties of concurrent java software. In *Proceedings of the 12th International Conference on Concurrency Theory*, pages 39–58. Springer-Verlag, 2001.

[43] T.A. Henzinger and P.-H. Ho. HYTECH : The cornell HYbrid TECHnology tool. In *Hybrid Systems*, pages 265–293, 1994.

[44] T.A. Henzinger, Z. Manna, and A. Pnueli. Timed transition systems. In *REX Workshop*, pages 226–251, 1991.

[45] C. Hernalsteen and A. Fevrier. Introduction of a suspend/resume operator in et-lotos. volume 1231 of *LNCS*, pages 400–414, 1997.

[46] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[47] G.J. Holzmann. *Design and validation of computer protocols*. Prentice-Hall, Inc., 1991.

[48] T. Hune, K.G. Larsen, and P. Pettersson. Guided synthesis of control programs using uppaal. *Nordic Journal of Computing*, 8 :43–64, 2001.

[49] C.N. Ip and D.L. Dill. Better verification through symmetry. *Formal Methods in System Design*, 9(1-2) :41–75, 1996.

[50] ISO - Information processing systems - Open Systems Interconnection. LOTOS - a formal description technique based on the temporal ordering of observational behaviour. ISO International Standard 8807 :1989, ISO, September 1989.

[51] ISO/IEC. Information technology - enhancements to LOTOS (E-LOTOS). Technical Report 15437 :2001, ISO/IEC, 2001.

[52] N.D. Jones, L.H. Landweber, and Y.E. Lien. Complexity of some problems in petri nets. *TCS : Theoretical Computer Science*, 4, 1977.

[53] S. Katz and O. Grumberg. A framework for translating models and specifications. In *Proc. of the 3d Int. Conf. on Integrated Formal Methods*, volume 2335 of *LNCS*.

[54] M. Koutny. A compositional model of time Petri nets. In *Proc. of the 21st Int. Conf. on Application and Theory of Petri Nets (ICATPN 2000)*, number 1825 in LNCS, pages 303–322, Aarhus, Denmark, 2000. Springer-Verlag.

[55] D. Larrabeiti, J. Quelmada, and S. Pavón. From LOTOS to Petri nets through expansion. In Gotzhein, R. and Bredereke, J., editors, *Proc. of Int. Conf. on Formal Description Techniques and Theory, application and tools (FORTE/PSV'96)*, 1996.

[56] L. Léonard and G. Leduc. An introduction to ET-LOTOS for the description of time-sensitive systems. *Computer Networks and ISDN Systems archive*, 29(3), 1997.

[57] C. Lohr. *Contribution à la conception de systèmes temps-réel s'appuyant sur la technique de description formelle RT-LOTOS.* PhD thesis, LAAS-CNRS, 2003.

[58] T. Massart, L. Van Begin, and E. Van Nuffel. Design of timed systems using a real-time process algebra. In *2nd Panhellenic logic symposium*, 1999.

[59] K.L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, 1993.

[60] P. Merlin. *A study of the recoverability of computer system.* PhD thesis, Dep. Comput. Sci., Univ. California, Irvine, 1974.

[61] R. Milner. *A calculus of communication systems*, volume 92 of *LNCS*. 1980.

[62] R.M. Milner. *Communications and Concurrency.* Prentice Hall, 1989.

[63] W. Nam and R. Alur. Learning-based symbolic assume-guarantee reasoning with automatic decomposition. In Susanne Graf and Wenhui Zhang, editors, *Automated Technology for Verification and Analysis, 4th International Symposium, ATVA*, volume 4218, pages 170–185. Springer, 2006.

[64] N. Navet, editor. *Systèmes temps-réel 1, techniques de description et de vérification.* Editions Hermes, Paris, 2006.

[65] X. Nicollin and J. Sifakis. The algebra of timed processes atp : Theory and application. *Information and Computation*, 114 :131–178, 1994.

[66] E. R. Olderog. *Nets, Terms, and formulas.* Cambridge University Press, 1991.

[67] OMEGA. Omega home page. http ://www-omega.imag.fr.

[68] OMG. Uml 2.0 superstructure specification. http ://www.omg.org/docs/ptc/03-08-02.pdf.

[69] OMG. Uml profile for modeling and analysis of real-time and embedded systems. http ://www.omg.org.

[70] OMG. Uml profile for schedulability, performance, and time, version 1.1. http ://www.omg.org.

[71] S. Owre, J. Rushby, N. Shankar, and F. von Henke. Formal verification for fault-tolerant architectures : Prolegomena to the design of pvs. In *IEEE Transactions on Software Engineering*, pages 107–125, 1995.

[72] D. Park. Concurrency and automata on infinite sequences. volume 104 of *LNCS*, pages 167–183. Springer-Verlag, 1981.

[73] D. Perry. *VHDL*. McGraw-Hill, New York, 1991.

[74] G.D. Plotkin. A structurel approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, 1981.

[75] C. Ramchandani. *Analysis of Asynchronous Concurrent Systems by Timed Petri Nets*. PhD thesis, MIT, Cambridge, Feb 1974.

[76] G.M. Reed and A.W. Roscoe. A timed model for communicating sequential processes. *Theoretical Computer Science*, 1988.

[77] P.-O. Ribet. *Vérification formelle de systèmes. Contribution à la réduction de l'explosion combinatoire*. PhD thesis, Institut National des Sciences Appliquées, Toulouse, juillet 2005.

[78] O.H. Roux and A.-M. Déplanche. A time petri net extension for real time task scheduling modeling. In *Eur. journal of Automation(JESA)*, 2002.

[79] O.H. Roux and D. Lime. Time petri nets with inhibitor hyperarcs, formal semantics and state space computation. In *Proc. Int. Conf. on Applications and Theory of Petri Nets*, Bologna, Italy, 2004.

[80] RT-LOTOS. Real-time LOTOS home page. http ://www.laas.fr/RT-LOTOS/.

[81] T. Sadani and P. de Saqui-Sannes. Une approche efficace pour le traitement des données dans rt-lotos. Technical report, LAAS-CNRS, February 2007.

[82] R. Sisto and A. Valenzano. Mapping Petri nets with inhibitor arcs onto basic LOTOS behavior expressions. *IEEE Transactions on computers*, 44(12) :1361–1370, December 1995.

[83] D. Taubner. *Finite Representations of CCS and TCSP Programs by Automata and Petri Nets*. Number 369 in LNCS. Springer-Verlag, 1989.

[84] D. Thomas and P. Moorby. *The Verilog Hardware Description Language - 2nd Edition -*. Kluwer Academic Publishers, Norwell, USA, 1995.

[85] K.J. Turner. The formal specification language LOTOS A course for users. 1993.

[86] C.A. Vissers, G. Scollo, M. van Sinderen, and E. Brinksma. Specification styles in distributed systems design and verification. *Theoretical Computer Science*, 89(1) :179–206, 1991.

[87] M. Yannakakis and D. Lee. An efficient algorithm for minimizing real-time transition system. In *Proc. of f the Conf. on Computer-Aided Verification (CAV)*, volume 697 of *LNCS*, Berlin, 1993.

[88] W. Yi. Real-time behaviour of asynchronous agents. In *Proc. of Int. Conf on Theories of Concurrency : Unification and Extension (CONCUR)*, volume 458 of *LNCS*, 1990.

[89] S. Yovine. Kronos : A verification tool for real-time systems. *Software Tools for Technology Transfer*, 1(123–133), 1997.

9 Annexe A : Preuve de la consistance de la traduction

Preuve de l'opérateur de préfixage par une action :
La sémantique RT-LOTOS stipule que dans le processus $a;P$ l'action a peut se produire à n'importe quel moment. Après l'occurrence de cette dernière, le processus se transforme en P.
$\underset{\rightarrow}{\mathbb{R}}$: A partir de H1 et H3, on a que t_0 est la seule transition sensibilisée dans le composant $C_{a;P}$. Or t_0 est une transition non temporisée (son intervalle statique est $[0,\infty)$ ainsi tout écoulement du temps est acceptable.
$\underset{\leftarrow}{\mathbb{R}}$: Par définition, tout écoulement du temps est acceptable.
$\underset{\rightarrow}{a}$: L'occurrence de l'action a est toujours possible quand sa place d'entrée est marquée.
$\underset{\leftarrow}{a}$: L'occurrence de l'action $a$dans le terme $a;P$ est toujours possible.
Il est évident que si C_P vérifie les hypothèses H1-H5, alors $C_{a;P}$ les vérifie aussi.

Preuve de l'opérateur de l'offre limitée dans le temps :
$\underset{\rightarrow}{\mathbb{R}}$:H1+H3$\Rightarrow$ t_0 et t_1 sont les seules transitions sensibilisées. Ainsi, l'écoulement de temps de d unités de temps acceptable dans le terme RT-LOTOS $a\{d\};P$ est aussi acceptable dans le composant $C_{a\{d\},P}$.
$\underset{\leftarrow}{\mathbb{R}}$:H1+H3$\Rightarrow t_1$ est la seule transition temporisée qui soit sensibilisée. Par conséquent, le composant $C_{a\{d\},P}$ accepte tout écoulement du temps jusqu'à la date d, tout comme le processus RT-LOTOS $a\{d\};P$.
$\underset{\rightarrow}{a}$: Évident pour l'occurrence de a. concernant l'occurrence de tv, cette dernière est possible seulement après une progression temporelle de d unités de temps, après cette date, l'occurrence de tv est aussi possible dans le composant $C_{a;\{d\}}$. Le processus RT-LOTOS $a\{d\};P$ se transforme en *stop*. Le composant $C_{a\{d\};P}$ devient inactif avec un marquage nul, étant donné l'absence de transitions sources (cf. H1), le composant $C_{a\{d\};P}$ se comporte comme le processus *stop*.
$\underset{\leftarrow}{a}$:similaire à $\underset{\rightarrow}{a}$.
Il est évident que si C_P vérifie les hypothèses H1-H5, alors le composant $C_{a\{d\};P}$ les vérifie aussi.

Preuve de l'opérateur de délai déterministe :

$\underset{\rightarrow}{\mathbb{R}}$ *delay(d)P* $\xrightarrow{d'}$ *delay(d-d')P* (avec $d' \leq d$). La transition t_0 a un intervalle temporel égal à *[d,d]*. H1+H3$\Rightarrow t_0$ est la seule transition sensibilisée. Par conséquent un écoulement du temps de d' unité de temps est acceptable dans le composant $C_{delay(d)P}$. Après l'écoulement de ce délai, le composant sera dans un état ou t_0 est toujuors l'unique transition sensibilisée avec un intervalle de tir égal à *[d-d',d-d']* qui est bisimilaire à l'état du terme *delay(d-d')P*.

$\underset{\rightarrow}{a}$ L'unique action possible pour un processus RT-LOTOS retardé d'un délai déterministe est l'expiration de ce même délai : delay(0)P $\xrightarrow{d'}$ P. Dans le composant équivalent $C_{delay(0)P}$, la

transition t_0 est sensibilisée et tirable. Le tir de t_0 active le composant C_p qui par induction est equivalent au processus P.

$\underset{\leftarrow}{\mathbb{R}}$:similaire à $\underset{\rightarrow}{\mathbb{R}}$.

$\underset{\leftarrow}{a}$:similaire à $\underset{\rightarrow}{a}$.

Preuve de l'opérateur de la latence :

La preuve est similaire à celle fournie pour l'opérateur de l'offre limitée dans le temps.

Preuve de l'opérateur de synchronisation :

La preuve est donnée dans la section 3.9.

Preuve de l'opérateur du choix :

Notons pour commencer que si les hypothèses H1 - H5 sont vérifiées dans les composants C_P et C_Q alors elles le sont aussi dans le composant $C_{P[]Q}$, étant donné qu'aucune transition source n'a été introduite est que les intervalles statiques des transitions n'ont pas été modifiés. La preuve de cet opérateur se fait en trois étapes : la première étape concerne le comportement avant l'occurrence de la première action. La deuxième traite de l'occurrence de la première action et la dernière étape vise à s'assurer que les transitions introduites n'interfèrent pas avec le comportement d'un composant, une fois le contrôle transférer à ce dernier.

Afin de simplifier les notations nous définissions l'opérateur de projection suivant $._{|X}$ (avec $X \in \{P, Q\}$) . Ce dernier opère sur des places, des transitions ou des marquages du composant $C_{P[]Q}$.

Ainsi :

$P_{|X} = P_X \backslash O_X$ est l'ensemble de places appartenant à C_X dans $C_{P[]Q}$.

$M_{|X} = M \cap P_{|X}$ est le marquage de C_X dans le marquage de $C_{P[]Q}$.

$T_{|X} = (T_X \backslash \mathcal{LA}(C_X)) \cup Split(C_X)$ est l'ensemble de transitions *non finales* de C_X augmenté de l'ensemble des transitions issuent de la duplication des transitions finales $\{t^m\}$ (introduites pour purger le composant C_Y $(Y \neq X)$).

Bien entendu, $W_{|P} \cap W_{|Q} = \phi$, avec W un ensemble de places , de transitions ou un marquage.

Aux hypothèses H1-H5 nous rajoutons les hypothèses suivantes :

H'1 Dans les composants C_P et C_Q et avant l'occurrence de la première action, seulement un seul marquage atteignable est marqué à la fois.

$\forall m, m' \in Pre_{\mathcal{FA}}(C_X)$, $m \neq m' : m \subseteq M \Rightarrow m' \nsubseteq M$ avec $X \in \{P, Q\}$

H'2 toutes les places *Lock* sont marquées avant l'occurrence de la première action $(M_{P[]Q}(E_{P[]Q}) = 1)$. Une fois la première action exécutée, toutes les places *lock* sont vidées de leurs jetons $(M_{P[]Q}(E_{P[]Q}) = 0)$.

$M_{P[]Q}(E_{P[]Q}) = 1 \Rightarrow \exists X \in \{P, Q\}, \forall pl \in Lock^{Tev}(C_X),\ M(pl) = 1$

$M_{P[]Q}(E_{P[]Q}) = 0 \Rightarrow \exists X \in \{P, Q\}, \forall pl \in Lock^{Tev}(C_X),\ M(pl) = 0$

De plus, les places *lock* sont continuellement sensibilisées dés le marquage initial et ce jusqu'au tir de la première action de C_P ou C_Q.

H'3 une fois la première action exécutée (et avant l'exécution de la dernière action), le marquage du composant $C_{P[]Q}$ peut être éclaté en deux ensembles disjoints : A le marquage du composant toujours actif et D celui du composant désactivé.

$(M_{P[]Q}(E_{P[]Q}) = 0 \wedge (M_{P[]Q}(exit) = 0) \Rightarrow \exists (A, D) \in \{P, Q\}, A \neq D$

$\exists m \in Pre_{\mathcal{FA}}(D), \exists M_A \in M(C_A) :\ M_{P[]Q} = m \cup M_A$

H'4 avant l'occurrence de la première action, le marquage du composant $C_{P[]Q}$ peut etre séparé en trois ensembles disjoints : celui des places *lock* et les deux ensembles $Pre_{\mathcal{FA}}(C_X)$,$X \in \{P, Q\}$.

$M_{P[]Q}(E_{P[]Q}) = 0 \Rightarrow \exists m_P \in Pre_{\mathcal{FA}}(P), \exists m_Q \in Pre_{\mathcal{FA}}(Q),\ M_{P[]Q} = m_P \cup m_Q \cup Lock$

Cœur de la preuve : $\underrightarrow{\mathbb{R}}$ il y a deux cas où l'écoulement du temps est possible en présence d'un choix :

1. $P[]Q \xrightarrow{d'} P'[]Q'$ ce qui implique $P \xrightarrow{d'} P'$ (et idem pour Q). Par induction $C_P \xrightarrow{d'} C_{P'}$ (et $C_Q \xrightarrow{d'} C_{Q'}$) . Mais cet écoulement du temps est il possible dans $C_{P[]Q}$? Le changement du comportement temporel de $C_{P[]Q}$, ne peut résulter que de l'introduction des places *lock* (plus précisément des places dans ${}^\bullet t$ pour une transition t). Or les places Lock sont initialement marquées (cf. H'2). Ainsi, si une transition t est sensibilisée dans un composant C_P, C_Q alors elle l'est aussi dans $C_{P[]Q}$.
   De plus ces places sont continuellement marquées depuis l'activation du composant $C_{P[]Q}$, et il n y a donc aucune différence entre la valeur de l'intervalle de tir associé à t dans $C_{P[]Q}$ est sa valeur dans C_P ou C_Q. Par conséquent tout écoulement du temps est possible dans $C_{P[]Q}$.

2. $A \xrightarrow{d'} A'$ avec $(A \in \{P, Q\})$. Le choix a été résolu ($(M(E_{P[]Q}) = 0$ dans $C_{P[]Q})$ et l'un des processus est toujours actif (A) et l'autre a été désactivé (D). De H'2 on a qu'aucune transition n'est sensibilisée dans D, il n' y a donc aucune restriction temporelle émanant de $T_{|D}$. Par induction, nous savons que $A \xrightarrow{d'} A'$ implique $C_A \xrightarrow{d'} C_{A'}$, Les éléments introduits par le pattern du choix dans le réseau du composant C_A n'interfèrent pas avec le comportement temporel de C_A. Par conséquent, un écoulement du temps de d unités de temps est toléré par toutes les transitions dans $T_{|D}$ et $T_{|A}$; cet écoulement du temps est donc possible dans le composant $C_{P[]Q}$(Les hypothèses H'1, H'2,H'3 et H'4 sont liées au marquage, l'écoulement du temps ne change pas ce dernier et ces hypothèses sont donc préservées) .

$\underleftarrow{\mathbb{R}}$: Un écoulement du temps possible dans le composant $C_{P[]Q}$, l'est-il aussi dans $P[]Q$?. Comme pour le précédent cas, nous distinguons deux possibilités : avant et aprés l'occurrence de la première action. A partir de H'2, nous faisons le même raisonnement que pour $\underrightarrow{\mathbb{R}}$, en se servant du fait que les places *Lock* n'interfèrent pas avec le comportement temporel des composants. Un écoulement du temps est possible dans $C_{P[]Q}$ ssi :

1. les places $E_{P[]Q}$ sont marquées est l'écoulement du temps est possible dans C_P et C_Q. Par induction, il est aussi possible dans P et Q est donc dans $P[]Q$.
2. les places $E_{P[]Q}$ sont vides, et l'écoulement du temps n'est possible que dans le composant toujours actif. Par induction cet écoulement du temps est toujours possible dans le processus actif (P ou Q).

$\underrightarrow{a}$ Il y' a quatre règles qui régissent l'évolution d'un comportement contenant un choix (Par symétrie nous ne considérons que les actions de P).

1. Transfert de contrôle à P ($P[]Q \xrightarrow{a} P'$: L'occurrence de la première action de P dans $P[]Q$ est possible ssi elle l'est aussi dans P ($P \xrightarrow{a} P'$). Par induction, le tir de t_a est aussi possible dans C_P. De plus $t_a \in \mathcal{FA}(C_P)$. De H'2 nous avons que toutes les places *lock* sont marquées (elles sont d'ailleurs continuellement marquées dés l'activation du composant), ainsi si t_a est tirable dans C_P alors elle l'est aussi dans $C_{P[]Q}$. Les hypothèses

H1 - H5 sont préservées par le tir de t_a.
H'1 est préservée étant donné que le tir de la première action vide le marquage atteignable.
H'2 est préservée par construction, $E_{P[]Q} \in^\bullet t_a$ par construction de $C_{P[]Q}$. Les places $E_{P[]Q}$ ainsi que les places *lock* sont marquées. Le tir de t_a vide toutes ces places.
H'3 est préservée, de H'4 nous savons qu'avant le tir de t_a le marquage M de $C_{P[]Q}$ est l'union disjointe de $M_p \in Pre_{\mathcal{FA}}(P)$, $M_Q \in Pre_{\mathcal{FA}}(Q)$ et $M_{Lock} = Lock(P[]Q)$. De ces ensembles, le tir de t_a vide M_p et M_{Lock} et crée des jetons seulement dans le composant C_P.
H'4 n'est applicable qu'avant le tir de t_a.

2. Occurrence d'un évènement temporel : un événement temporel possible dans *P[]Q*, est possible soit dans P ou dans Q. Par induction il l'est aussi soit dans C_P ou C_Q. Supposons qu'il se produise dans C_P, l'occurrence de cet évènement temporel est aussi possible dans $C_{P[]Q}$ car :
   - Soit toutes les places *lock* sont marquées (H'2) et nous savons que ses dernières n'interfèrent pas avec le comportement temporel de C_P.
   - Soit les places *lock* sont vides, et donc si cet évènement temporel est possible dans C_P il l'est aussi dans $C_{P[]Q}$.
   Toutes les hypothèses sont trivialement préservées.
3. Occurrence d'une dernière action : il n'est pas simple de caractériser cette notion de dernière action dans un processus RT-LOTOS. Nous pouvons par exemple utiliser la notation suivante $P[]\underline{Q} \equiv \underline{P[]Q}$. Ce que l'on doit prouver est que si l'un des deux composants termine son exécution, alors $C_{P[]Q}$ termine aussi la sienne.
   Supposons que $P[]Q$ puisse terminer son exécution. Ce qui signifie que le processus actif (noté A) peut aussi terminer son exécution, et idem pour C_A (par induction) . Ce qui signifie qu'il existe une transition t avec $t \in \mathcal{LA}(C_A)$ et $M_A \geq^\bullet t$ (le marquage de C_A est suffisant pour tirer t). De plus, nous avons $t^\bullet = \{out\}$ puisque t est une dernière action de C_A. Soit M'_Ale marquage obtenu après le tir de t ($M_A[t > M'_A$). Par H4, nous avons $M'_A = \{out\}$, ce qui implique $M_A = {}^\bullet t$, nous obtenons ainsi que $M_{|A} = {}^\bullet t$ (car $M_A = M_{|A}$).
   De H'3, nous avons que $\exists M_D \in Pre_{\mathcal{FA}}(D)$ tel que $M = M_D \cup M_{|A}$, ie. $M = M_D \cup {}^\bullet t$.
   Il existe une seule transition t^{M_D}qui soit sensibilisée à la fois (par construction des dernières action de $C_{P[]Q}$, cf. H'1), par conséquent la terminaison de $C_{P[]Q}$ est possible.
   La préservation des hypothèses H1, H2, H3, H5, et H'1-H'4 est trivial. En ce qui concerne H4, nous avons montré qu'avant le tir t^{M_D}, le marquage de $C_{P[]Q}$ est $M = M_D \cup {}^\bullet t = {}^\bullet t^{M_D}$. Ainsi, le marquage obtenu après le tir de t^{M_D} est $M' = M - {}^\bullet t^{M_D} + t^{M_D\bullet} = \{out\}$. H4 est donc préservée.
4. Toute autre action : toute action de ce type, possède les mêmes ensembles *Pre* et *Post* à la fois dans $C_{P[]Q}$ et dans C_P (ou C_Q) . L'occurrence d'une telle action est possible dans *P[]Q* si elle est possible dans le processus actif (A). Par induction elle est aussi possible dans C_A et puisque les ensembles *Pre* et *Post* sont les mêmes alors, son occurrence est aussi possible dans $C_{P[]Q}$.

$\underline{\leftarrow}$:similaire à $\underline{\rightarrow}$

Preuve de l'opérateur de la composition séquentielle :

La preuve est évidente par construction.

Preuve de l'opérateur du disrupt :

De même que pour l'opérateur du choix, nous avons besoin ici d'introduire les hypothèses suivantes :

H'1 : dans les composants C_P et C_Q et avant l'occurrence de la première action, seulement un seul marquage atteignable est marqué à la fois.

$\forall m, m' \in Pre_{\mathcal{FA}}(C_X)$, $m \neq m' : m \subseteq M \Rightarrow m' \nsubseteq M$ avec $X \in \{P, Q\}$

H'2 : Soit P est toujours actif, et donc toutes les places *disrupt* sont marquées ($M_{P[>Q}(E_{P[>Q}) = 1$).

Soit P a été interrompu et donc toutes les places disrupt sont vides ($M_{P[>Q}(E_{P[>Q}) = 0$).

$M_{P[>Q}(E_{P[>Q}) = 1 \Rightarrow \forall t \in T_P^{Time}$, $M(E^t_{P[>Q}) = 1$

$M_{P[>Q}(E_{P[>Q}) = 0 \Rightarrow \forall t \in T_P^{Time}$, $M(E^t_{P[>Q}) = 0$

De plus, les places *disrupt* ($E^t_{P[>Q}$) sont continuellement marquées et ceci à partir de l'activation du composant et jusqu'au tir de la première action de Q ou le tir de la transition t.

Cœur de la preuve : $\underrightarrow{\mathbb{R}}$: un écoulement du temps dans $P \lceil\!> Q$ est soit un écoulement du temps dans les deux processus, ou (aprés l'intérruption de P par Q) un écoulement du temps dans Q. Nous distinguons ces deux cas on se référant au marquage de la place $E_{P[>Q}$.

1. $M(E_{P[>Q}) = 1$, de H'2 nous avons que toutes les places *disrupt* sont marquées. En suivant un raisonnement similaire à celui du choix, nous pouvons affirmer que si un écoulement du temps est possible dans C_P et C_Q alors cet écoulement l'est aussi dans $C_{P[>Q}$ (Par induction si ce temps est possible pour P (resp Q) car il l'est aussi pour C_P (resp C_Q)).
2. $M(E_{P[>Q}) = 0$, par induction nous avons que cet écoulement du temps est possible dans C_Q. Par construction, toutes les transitions de C_P ont la place $E_{P[>Q}$ en entrée et sont par conséquent non sensibilisées. Cet écoulement du temps est donc acceptable pour $C_{P[>Q}$.

Cet écoulement du temps préserve trivialement toutes les hypothèses H1-H5 et H'1-H'4.

$\underleftarrow{\mathbb{R}}$ similaire à $\underleftarrow{\mathbb{R}}$

$\underrightarrow{a}$: Quatre différents cas doivent être distingués :

1. L'occurrence d'un évènement temporel x dans P : par induction x est aussi possible dans C_P. Donc P n'a pas encore été interrompu ($E_{P[>Q}$ est marquée). De H'2 nous avons que $E^v_{P[>Q}$ est aussi marquée. Par conséquent l'occurrence de l'évènement temporel x est aussi possible dans $C_{P[>Q}$.
2. L'occurrence d'une action de P : similaire au cas précédent.
3. Première action de Q : par induction cette action est possible dans C_Q. De H'2 nous avons que les places *disrupt* introduites sont marquées. Par construction, cette action est possible dans $C_{P[>Q}$.
4. Autre action de Q : par induction cette action est possible dans C_Q. Les places d'entrée de cette action sont les mêmes dans C_Q et dans $C_{P[]Q}$. Cette dernière est donc possible dans $C_{P[]Q}$.

$\underset{\leftarrow}{a}$:similaire à $\underset{\rightarrow}{a}$

Preuve des comportements de base :

stop : En RT-LOTOS, *stop* est le processus qui ne fait rien et qui accepte par conséquent tout écoulement du temps. Nous le représentons par un RdPT vide (qui ne contient ni places ni transitions). Il est évident que ce dernier ne tire aucune transition, accepte tout écoulement du temps et vérifie les hypothèses H1 -H5 de la section 3.3.

exit : preuve évidente par construction.

L'intériorisation : Cet opérateur se traduit simplement par le rajout d'un intervalle temporel égal à [0,0] à la transition concernée par l'intériorisation. Ce qui induit l'urgence de cette dernière.

10 Annexe B : Concepts et définitions des réseaux de Petri

Définition 10.1 Un réseau de Petri se définit par le tuple $\langle P, T, Pre, Post\rangle$avec :

1. $P = \{p_1, ..., p_i, ..., p_n\}$ un ensemble de places,
2. $T = \{t_1, ..., t_i, ..., t_n\}$ un ensemble de transitions, avec $P \cap T = \phi$,
3. $Pre : P \times T \rightarrow \mathbb{N}$ une application d'incidence avant,
4. $Post : P \times T \rightarrow \mathbb{N}$ une application d'incidence arrière.

– $Pre(p_i, t)$ contient la valeur entière n associée à l'arc allant de p_i à t ;
– $Post(p_i, t)$ contient la valeur entière associée à l'arc llant de t à p_i.

Si s est un élément de $P \cup T$, alors ${}^\bullet s$ désigne l'ensemble des prédécesseurs de s dans le réseau et $s^\bullet$ l'ensemble des successeurs de s dans le réseau. Autrement dit, si s est une transition, alors ${}^\bullet s = Pre(s)$ et $s^\bullet = Post(s)$, et si s est une place, alors ${}^\bullet s = Post(s)$ et $s^\bullet = Pre(s)$.

Cette notation s'étend naturellement aux sous-ensembles de noeuds, ie. ${}^\bullet S = \{t | \exists s \in S, t \in {}^\bullet s\}$ et $S^\bullet = \{t | \exists s \in S, t \in s^\bullet\}$.

Définition 10.2 Dans un réseau de Petri, toute transition tirable t peut être tirée et son tir conduit à un nouveau marquage m' défini, pour tout p par :

$$m'(p) = m(p) - Pre(p, t) + Post(p, t)$$

La règle de tir, liée à $\{m, Pre, Post\}$, signifie que le nouveau marquage $m'(p)$ sera obtenu, à partir du marquage précédent $m(p)$, en supprimant d'abord, dans les places d'entrée de t, le nombre de jetons indiqué sur les arcs entrants de t, par $-Pre(p, t)$, et en ajoutant ensuite, à chaque place de sortie p de t, le nombre de jetons correspondant au poids indiqué sur l'arc sortant de t vers p, par $+Post(p, t)$.

Le tir d'une transition t est noté $m[t > m'$ ou $m - t \rightarrow m'$.

Définition 10.3 L'ensemble des marquages accessibles A, se définit par le plus petit ensemble :

$m_0 \in A$ et, si $m \in A$ et $m[t > m'$ alors $m' \in A$.