Analyse de Réseau de Petri Temporels Exécutés de Façon Synchrone

Ibrahim Merzoug[1], Karen Godary-Dejean[1], and David Andreu[1]

Université de Montpellier, LIRMM,
161 Rue Ada, 34090 Montpellier, France
http://www.lirmm.fr/

Abstract

Lors de la conception de systèmes numériques complexes, le recours aux méthodes formelles est utile notamment pour valider les propriétés du système, avec certitude. Cependant, les processus de validation usuels font abstraction des propriétés non fonctionnelles, notamment celles issues des contraintes d'exécution sur la cible matérielle. En l'occurrence, l'analyse des réseaux de Petri temporels doit être étudiée avec attention lorsque ce formalisme, intrinsèquement asynchrone, est exécuté de façon synchrone sur un FPGA. Il faut alors considérer la synchronisation d'horloge, le parallélisme effectif et l'interprétation. Actuellement, aucune sémantique formelle et aucune méthode d'analyse ne s'attaquent à toutes ces problématiques en même temps. Ainsi, nous proposons une nouvelle méthode d'analyse pour les réseaux de Petri interprétés exécutés en synchrone, avec une sémantique formelle d'exécution et un graphe d'états spécifique : le Graphe de Comportement Synchrone.

1 Introduction

La validation formelle est une étape obligatoire du processus de conception des systèmes complexes pour les domaines critiques tel que le médical, le nucléaire ou l'avionique. Elle permet de garantir le comportement du système, ainsi que le respect des spécifications et des exigences réglementaires. Parmi les méthodes formelles, le *model checking* effectue une analyse exhaustive de tous les états possibles d'un modèle (formel) du système. A partir de ce modèle, à base de machine à états (réseaux de Petri, automates, etc.), il s'agit de construire le graphe d'états sur lequel sera effectuée la vérification des propriétés désirées, exprimées en logique temporelle. Cependant, les résultats de vérification sont insuffisants si les méthodes d'analyse ne considèrent pas la façon dont ce modèle est implémenté et exécuté. En effet, cela peut impacter directement le comportement réel du système, et mener à des comportements très différents selon la cible matérielle et les stratégies d'implémentation choisies.

Dans notre contexte, nous nous intéressons à la conception et au prototypage de dispositifs médicaux implantable actifs (DMIA), en particulier l'architecture numérique d'un stimulateur neural implantable [1]. Dans ce cadre, pour gérer la complexité et les besoins de fiabilité, une méthodologie a été développée en se basant sur les concepts d'ingénierie dirigée par les modèles [17]. Ainsi, une architecture numérique est composée d'un ensemble de composants inter-connectés. Le comportement des ces composants, ainsi que leur interconnexion, sont modélisés à l'aide d'un formalisme basé réseaux de Petri, ce qui permet l'analyse formelle du comportement global du système.

Le comportement du système est modélisé avec des réseaux de Petri temporels interprétés (*Interpreted Time Petri Nets* - ITPN), qui sont les réseaux de Petri temporels classiques [12] (c'est à dire avec un intervalle temporel associé aux transitions) auxquels l'ajout de l'interprétation permet de représenter l'interaction du système avec son environnement. Dans

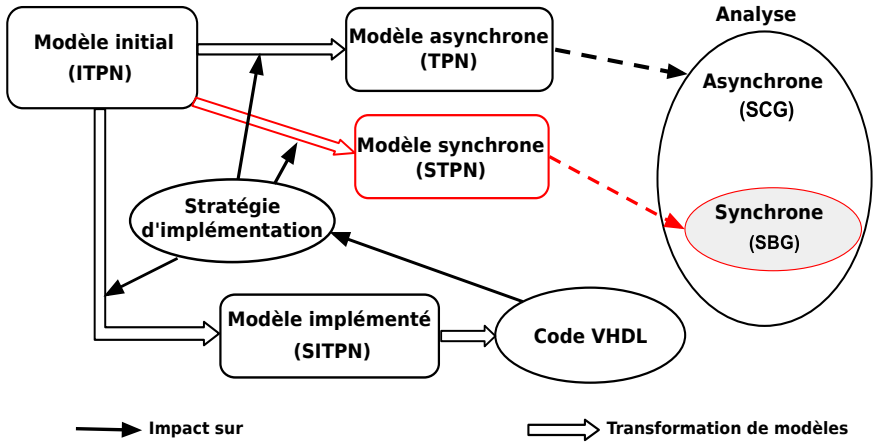


Figure 1: Méthodologie de conception

notre cas, cette interprétation est constituée de conditions, de fonctions et d'actions, qui interagissent avec les signaux externes (les entrées/sorties du système) ou qui manipulent des variables internes au système. Une condition est une expression logique, associée à une transition, qui influence le comportement du système en autorisant ou non (suivant leur valeur true ou false) le tir de la transition concernée. Les fonctions (aussi appelées actions impulsives) sont également associées aux transitions, et sont exécutées lors du tir de ces transitions. Les actions (aussi appelées actions continues) sont associées aux places et exécutées de façon continues tant que les places concernées sont marquées.

Ce modèle (initial) est utilisé pour contrôler DMIA. Il est implémenté sur un FPGA (*Field Programmable Gate Array*) et exécuté de façon synchrone : l'évolution des ITPN est dirigée par le signal d'horloge. Ce choix d'implémentation induit des contraintes très spécifiques liées à l'exécution, en particulier la simultanéité de tir des transitions (vrai parallélisme) et l'évolution discrète du temps. En intégrant toutes ces contraintes, le modèle ITPN initial est transformé en un modèle ITPN synchrones (*Synchronous* ITPN - SITPN), qui sera ensuite directement transformé en code VHDL pour exécution sur la cible FPGA [17], [9].

Les DMIA étant des systèmes critiques et temps réel, notre méthodologie de conception comprend également une étape d'analyse formelle afin d'obtenir des résultats de validation fiables, au plus tôt dans le processus de conception. Ainsi, initialement décrite dans [17], la méthodologie a été complétée par un processus d'analyse asynchrone [8] permettant d'exploiter les possibilités des méthodes et outils d'analyse existants des TPN classiques [3]. L'ensemble de la méthodologie existante est résumée en noir sur la Figure 1. Cependant, l'analyse asynchrone du comportement d'un modèle exécuté en synchrone introduit un écart entre le comportement réel et le comportement analysé, ce qui réduit significativement l'efficacité de l'analyse et la confiance dans les résultats. Par exemple, lors de l'analyse asynchrone, le tir des transitions parallèles (i.e. tirables au même instant) sont représentées par un entrelacement, alors qu'en synchrone elles sont toutes tirées simultanément en une seule étape. Le seul moyen d'analyser plus finement nos modèles de façon efficace et réaliste, tout en considérant les contraintes d'exécution, est de développer notre propre méthode d'analyse afin d'énumérer aussi bien que

possible l'espace d'état réel de notre système. Dans ce cadre, il est nécessaire que le formalisme utilisé pour représenter le modèle analysable du système permette de refléter la réalité de l'exécution. Toutefois, à notre connaissance, aucune des extensions de réseaux de Petri de la littérature (synchronisés [13], temporels [2], temporisés, [16], discrets [5]) ne réunit toutes les caractéristiques requises. Ainsi, nous avons défini une nouvelle méthode d'analyse des ITPN exécutés de façon synchrone. Cette contribution est représentée en rouge sur la Figure 1, et fait l'objet de cette article.

Une fois le contexte brièvement précisé, la section 2.1 présente les principes d'exécution et le formalisme ITPN du modèle initial. La section 3 introduit alors les règles de transformation du modèle initial ITPN en un modèle analysable, qui sera défini formellement ainsi que sa sémantique. Puis, dans la section 4, les principes et l'algorithme de construction du Graphe de Comportement Synchrone (Synchronous Behavior Graph, noté SBG) sont présentés et illustrés sur un exemple. Enfin, une brève comparaison entre les principes de notre approche vis-à-vis d'approches existantes est effectuée dans la section 5, avant de conclure et de mentionner les perspectives de nos travaux.

2 Contexte

2.1 Principes d'Exécution

Les principes d'exécution de notre système sont schématisés Figure 2 [8]. L'évolution synchrone du système est composée de 4 étapes synchronisées par son horloge. A chaque cycle d'horloge:

- Sur le front montant (1) : Mise à jour du marquage en fonction des transitions qui viennent d'être tirées. La modification du marquage peut durer la demi-période (2), mais elle sera obligatoirement terminée avant le front d'horloge suivant.

- Sur le front descendant (3) : Évaluation, à partir de l'état courant (c'est à dire le marquage des places et la valeur des conditions et des compteurs temporels), des transitions devant être tirées lors de la demi-période (4). C'est sur ce front descendant que sont gérés les compteurs temporels liés aux transitions temporelles: l'incrémentation de l'horloge d'1ut peut mener une transition à changer de statut (devenir tirable, ou être obligatoirement tirée). Il est également possible que le nouveau marquage entraîne l'initialisation du compteur d'une transition nouvellement sensibilisée.

- Les fonctions et les actions sont exécutées sur des états stables: l'exécution des fonctions est démarrée lorsque le tir des transitions est terminé, c'est à dire sur le front montant suivant le tir des transitions; et les actions sont démarrées lorsque la mise à jour du marquage est terminée, c'est à dire sur le front descendant suivant.

Une exécution synchrone sur FPGA permet d'obtenir un comportement déterministe du système. Ce type d'implémentation garantit que les fonctions exécutées lors du tir des transitions ont le temps de terminer leur exécution avant l'évaluation des conditions. C'est un élément important pour la cohérence du système, car les fonctions peuvent influer sur la valeur de variables internes, et donc sur la valeur des conditions.

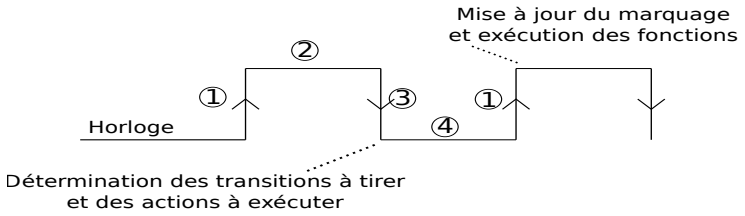


Figure 2: Principes d'exécution synchrone [8]

2.2 Le Modèle Initial: Réseau de Petri Temporels Interprétés

Le modèle initial (cf. Figure 1) est conçu en utilisant des ITPN. L'utilisation de réseaux de Petri associés à une interprétation est relativement courante en littérature, en particulier dans les domaines des systèmes à évenements discrets et systèmes logiques de contrôle (par exemple les *Signal Interpreted Petri Nets* - SIPN [18] ou les *Control Interpreted Petri Nets* - CIPN [7]). Mais l'interprétation est rarement associée à des éléments temporels. Dans notre cas, nous considérons l'extension temporelle classique des réseaux de Petri temporels [2], qui associe un intervalle temporel aux transitions. Et dans notre contexte, l'interprétation n'est pas réduite à la forme traditionnelle d'entrées/sorties, mais représentée par trois éléments différents: les actions, les fonctions et les conditions qui expriment, comme décrits précédemment, les interactions du système avec son environnement à des instants spécifiques de son évolution.

La définition formelle du formalisme ITPN que nous utilisons a été décrite dans [8]. Nous reportons ici les principales informations nécessaires à la compréhension de cet article [1]:

Soit $\mathbb{I}^*$ l'ensemble non vide des intervalles d'entiers positifs. Pour un intervalle $I \in \mathbb{I}^*$, $\downarrow I \in \mathbb{N}^*$ et $\uparrow I \in \mathbb{N}^* \cup \{+\infty\}$ représentent respectivement ses bornes minimale et maximale. Soit $\mathcal{C}$ l'ensemble des conditions, une condition étant une expression booléenne, et $\mathcal{B}$ l'ensemble des booléens $\{0, 1\}$. Soit $\mathcal{FC}$ l'ensemble des fonctions (actions impulsives), et $\mathcal{A}$ l'ensemble des actions (actions continues), décrites en VHDL.

Un ITPN est un n-uplet $< P, T, Pre, Post, m0, Is, C, Fc, A >$, tel que :

1. $< P, T, Pre, Post, m0 >$ est un réseau de Petri classique. P est l'ensemble des places, T l'ensemble des transitions, $m0$ le marquage initial. Pre, $Post$: $T \times P \rightarrow \mathbb{N}$ sont respectivement les fonctions pré-condition et post-condition ;

2. $I_s : T \rightarrow \mathbb{I}^* \cup \emptyset$ est la fonction associant les intervalles temporels statiques aux transitions;

3. $C : T \rightarrow \mathcal{C} \rightarrow \mathbb{B}$ est la fonction condition: une condition $c \in \mathcal{C}$ est associée à une transition $t \in T$ si $C(c, t) = 1$ (sinon $C(c, t) = 0$). La valeur instantanée d'une condition est définie par la fonction $Val : \mathcal{C} \rightarrow \mathbb{B}$.

4. $Fc : T \rightarrow \mathcal{FC}$ est la fonction des actions impulsives;

5. $A : P \rightarrow \mathcal{A}$ est la fonction des actions continues;

Comme montré sur la Figure 1, ce modèle initial, décrit en ITPN, est ensuite transformé soit en réseaux de Petri temporels interprétés synchrone (SITPN) puis en VHDL pour être exécuté sur une cible FPGA; ou alors il est transformé en un modèle analysable exprimé en réseaux

[1] Nous avons supprimé les priorités, qui servaient à gérer les conflits, puisque nous effectuons une analyse exhaustive.

de Petri temporels synchrone (STPN). Cette transformation dédiée à l'analyse synchrone est décrite ci-dessous.

3 Transformation de modèles pour l'analyse

3.1 Règles de transformation

L'étape de transformation de modèle doit considérer les propriétés non fonctionnelles, liées à la stratégie d'implémentation et les contraintes imposées par la cible matérielle. Le principe d'exécution de notre modèle est donné section 2.1. Si l'on désire que le comportement du modèle analysable soit le plus proche possible du comportement réel, la transformation vers le modèle analysable doit alors considérer très finement l'impact de la stratégie d'implémentation.

En effet, l'interprétation et l'exécution synchrone ne peuvent être analysées directement. Cependant, il est possible d'exprimer leur impact à l'aide des intervalles temporels. Ainsi, le modèle ITPN initial est transformé en un réseau de Petri temporels synchrone (STPN), qui sera le modèle analysable, en suivant les règles de transformation suivantes :

1. *Impact de l'exécution synchrone* :
   (a) Etant donné que sur le FPGA, les transitions ne sont tirées que sur les fronts du cycle d'horloge, alors le modèle évolue en temps discret. En effet, notre principe d'exécution (figure 2) a pour conséquence que les transitions sont obligatoirement tirées sur le front descendant de l'horloge, et que le tir prend obligatoirement $1ut$. Dans le STPN pour l'analyse, les intervalles temporels sont donc conservés mais doivent être considérés en temps discret.
   (b) Afin de garantir l'exécution déterministe, nous imposons pour l'implémentation sur la cible une sémantique (forte) de tir au plus tôt: une transition doit être tirée dès qu'elle est tirable [2].
   (c) Les deux contraintes précédentes mènent aux transformations suivantes : si une transition n'est associée ni à un intervalle temporel ni à une condition, cette transition sera alors tirée immédiatement, en 1 unité de temps ($1\ ut$), ce qui représente un cycle d'horloge; son intervalle temporel dans le STPN est alors égal à $[1, 1]$; de même, si un intervalle temporel $[a, b]$ est associé à une transition (sans condition), alors cet intervalle devient $[a, a]$.
   (d) De plus, l'exécution sur une cible matérielle avec du vrai parallélisme impose une sémantique de tirs simultanés: toutes les transitions tirables doivent être tirées en même temps.
2. *Impact de l'interprétation*:
   L'impact de l'interprétation peut être représentée en ne considérant que les conditions. En effet, les actions impulsives et continues influencent l'évolution du système en modifiant les valeurs des signaux et variables internes, qui sont ensuite pris en considération dans les conditions.
   Si une transition est associée à une condition, elle sera tirée dès que sa condition devient vraie: soit immédiatement (en $1ut$), soit dans le pire des cas elle ne sera jamais tirée si cette condition reste fausse. Ainsi son intervalle de tir sera $[1, +\infty[$.

[2] Remarque : attention, cette sémantique de tir au plus tôt est valable sur la cible, et donc sur le modèle ITPN, mais ce n'est pas le cas du modèle STPN qui doit au contraire pouvoir représenter tous les cas possibles, en particulier dans le cas d'une transition pouvant devenir vraie à des instants temporels différents.

3. *Impact de l'association de l'interprétation et des intervalles temporels* : Si un intervalle temporel et une condition sont associés à une même transition, cela peut mener à une situation de blocage: si la valeur de la condition value est toujours fausse alors que la borne supérieure de l'intervalle est atteinte, cette transition ne peut pas être tirée mais le temps ne peut pas non plus continuer à s'écouler. Dans cet article, nous ne considérerons pas ce cas comme possible (la sémantique de blocage est un travail en cours). Nous faisons l'hypothèse que lorsqu'une transition est associée à une transition possédant déjà un intervalle temporel, alors cette condition deviendra obligatoirement vraie pendant cet intervalle.

Exemple 1. *Les règles de transformation peuvent être illustrées sur le modèle de la figure 3. Ce modèle STPN est obtenu à partir d'un ITPN de même structure et même marquage initial, mais dont les transitions étaient respectivement:* t_0 *et* t_2 *des transitions normales, sans rien associé;* t_1 *une transition possédant une condition seule;* t_3*,* t_4 *et* t_5 *des transitions possédant un intervalle temporel et une condition;* t_6 *à* t_8 *des transitions temporelles sans condition, d'intervalles temporels* $[5, x]$*.*

3.2 Le Modèle Analysable: Définition des STPN

Un réseau de Petri temporels synchrone (STPN) est défini comme un n-uplet $< N, R_s >$, avec N un RdP classique et $R_s : T \rightarrow \mathbb{I}^*$ est la fonction de temps résiduel statique. A partir d'un modèle initial ITPN $< P, T, Pre, Post, m_0, I_s, C >$, le modèle analysable STPN $< P, T, Pre, Post, m_0, R_s >$ est construit ainsi :

- Le modèle STPN respecte la même structure que le modèle ITPN initial : il conserve les ensembles des places et transitions, les fonctions Pre and $Post$, et le marquage initial.
- Les intervalles résiduels R_s du modèle STPN sont définis par les règles de transformation décrites précédemment en considérant les intervalles et les conditions (fonctions I_s et C) du modèle initial.

Le marquage d'un STPN est défini par la fonction $m : P \rightarrow \mathbb{N}$. Une transition $t \in T$ est sensibilisée par un marquage m si ses places d'entrée possèdent tous les jetons permettant le tir de t, c'est à dire si $m \geq Pre(t)$. On note $Enable(m)$ l'ensemble des transitions sensibilisées par m. Un état d'un STPN est une paire $s = (m, R)$, avec m le marquage et R la fonction qui associe à chaque transition sensibilisée par un marquage m un intervalle de tir résiduel. On note respectivement $\downarrow R(t)$ et $\uparrow R(t)$ les bornes minimale et maximale de l'intervalle $R(t)$. L'état initial s_0 est (m_0, R_0), ou $R_0(t) = R_s(t)$, $\forall t \in Enable(m_0)$. On appelle $Firable(s)$ l'ensemble des transitions tirables à partir de l'état $s = (m, R)$. Une transition $t \in Firable(s)$ est tirable à un instant discret θ ($\theta \in \mathbb{N}^*$) ssi : $t \in Enable(m) \ \wedge \ \theta \in R(t)$

Deux transitions t and t' sensibilisés par un marquage m sont en conflit effectif dans un état $s = (m, R)$ ssi elles ont au moins une place en commun, et si le marquage de cette (ces) place(s) est insuffisant pour tirer les 2 transitions. Ainsi, deux transitions sont en conflit si le tir de l'une empêche le tir de l'autre. On note $Conflict(m, t)$ l'ensemble des transitions, dans l'état $s = (m, R)$, qui sont en conflit avec $t \in Firable(s)$:

$$Conflict(s, t) = \{t' \in Firable(s) \mid t' \notin Enable(m - Pre(t)) \wedge t' \neq t\} \quad (1)$$

3.3 Sémantique des STPN

Soit T^n l'ensemble des ensembles de transitions d'un STPN. On définit la sémantique des STPN comme un système de transitions temporisé $(S, s_0, \twoheadrightarrow)$, avec S l'ensemble des états,

$s_0 = (m_0, R_0) \in S$ l'état initial et $\twoheadrightarrow: S \times (T^n \times \mathbb{N}^*) \times S$ la relation de changement d'états, composée d'un changement d'états basé sur le tir de transitions, et d'un changement d'états basé sur l'évolution du temps. Soit $Fired$ l'ensemble des transitions tirées d'un état s à un instant discret non nul θ ($\theta \in \mathbb{N}^*$). La relation de changement d'états est définie, $\forall Fired \in T^n$ et $\theta \in \mathbb{N}^*$, comme suit :

1. Changement d'états basé sur le tir de transitions : $s = (m, R) \overset{Fired,\theta}{\twoheadrightarrow} s' = (m', R')$ ssi :
   (a) $\forall t \in Fired$, $t \in Firable(s) \wedge (Fired \cap Conflict(m, t) = \emptyset)$ (les transitions en conflit ne sont jamais tirées simultanément).
   (b) $\theta \leq\uparrow R(t)\ \forall t \in Enable(m)$ (il n'existe pas de transition devant être tirée à une date antérieure).
   (c) $m' = m - \sum_{t \in Fired} Pre(t) + \sum_{t \in Fired} Post(t)$
   (d) $\forall t' \in Enable(m')$, $R'(t') = R_s(t')$ ssi $t' \notin Enable(m)$, sinon $R'(t') = [max(1, \downarrow R(t') - \theta), \uparrow R(t') - \theta]$

2. Changement d'états basé sur l'évolution du temps seule : $s = (m, R) \overset{\theta}{\twoheadrightarrow} s' = (m, R')$ ssi :
   (a) $\forall t \in Enable(m), \theta <\uparrow R(t)$
   (b) $\forall t \in Enable(m), R'(t) = R(t) - \theta$

4 Le Graphe de Comportement Synchrone

Une fois la transformation de modèles effectuée, on obtient un modèle STPN. Ce modèle respecte nos contraintes d'implémentation, et peut être analysé, tout d'abord en construisant son graphe de comportement. L'approche la plus connue permettant de générer le graphe de comportement de réseaux de Petri temporels avec une sémantique discrète a été proposée par Popova-Zeugmann dans [14]. L'auteur représente un état par un marquage et un vecteur des instants de temps (valeurs courantes relatives en entiers) pour chacune des transitions sensibilisées ; cet état est appelé un *Integer-State*. Le nombre d'*Integer-State* dépend des bornes des intervalles temporels. Ces travaux étaient initiallement limités aux TPN finis, mais ont été étendus aux modèles infinis dans [15]. Bien que proche de nos besoins sur le plan de la gestion temporelle, ces travaux ne considèrent pas la possibilité du tir simultané de plusieurs transitions. C'est un problème important puisque dans notre contexte, l'exécution sur FPGA nous expose à du vrai parallélisme. Or, étant donné que les fonctions exécutées lors du tir des transitions manipulent des variables potentiellement partagées, il est important d'identifier de manière exacte quelles transitions peuvent être tirées en même temps.

Dans cette section, nous proposons une approche de génération du graphe de comportement d'un modèle STPN : le Graphe de Comportement Synchrone (*Synchronous Behavior Graph* - SBG). La sémantique du SBG est similaire à celle des STPN (section 3.3 précédente) : tous les changements d'états présentés dans la sémantique du STPN sont possibles dans le SBG. Pour éviter la confusion entre ces deux sémantiques, nous nommerons les états du SBG e (s étant réservé aux STPN). Par contre, la sémantique du SBG comporte un changement d'états supplémentaire qui exprime la possibilité d'une progression indéfinie/infinie, ce qui évite l'énumération infinie dans le cas d'un intervalle dont la borne supérieure est infinie :

3. Changement d'états basé sur l'évolution indéfinie/infinie du temps, $\theta = 1$, $e = (m, B, R) \overset{\theta}{\twoheadrightarrow} e = (m, B, R)$ ssi :

   (a) $\forall t \in T,\ t \in Enable(m) | R(t) = [1, +\infty[$

La suite de cette section décrit en détails la construction du graphe de comportement synchrone SBG.

4.1 Ensembles des Transitions Tirables

A partir d'un état $e = (m, R)$ du SBG, on définit divers ensembles de transitions tirables, en fonction des intervalles de temps résiduel :

1. Transitions Nécessairement Tirables (*Necessarily Firable Transitions* - NFT):
   - $1.1F(e)$ est l'ensemble des transitions qui doivent être immédiatement tirées après 1 unité de temps (1 ut): $t \in 1.1F(e)$ ssi $t \in enable(m)$ et $R(t) = [1, 1]$.
2. Transitions Probablement Tirables (*Probably Firable Transitions* - PFT):
   - $1.bF(e)$ est l'ensemble des transitions qui peuvent être tirées au moins après 1 ut et au plus tard après b unités de temps: $t \in 1.bF(e)$ ssi $t \in enable(m)$ et $R(t) = [1, b[$, $b \in \mathbb{N}^* - \{+\infty\}$, $b \neq 1$.
   - $1.\infty F(e)$ est l'ensemble des transitions qui peuvent être tirées au moins après 1 ut, mais qui au pire peuvent ne jamais être tirées : $t \in 1.\infty F(e)$ ssi $t \in enable(m)$ et $R(t) = [1, +\infty[$ [3].
3. Transitions Tirables Plus Tard (*Later Firable Transitions* - LFT) :
   - $a.aF(e)$ est l'ensemble des transitions qui doivent être tirées après a ut, avec a le plus petit temps résiduel supérieur à 1 des transitions de cet ensemble : $t \in a.aF(e)$ ssi $t \in enable(m)$, $R(t) = [a, a]$ et $\nexists t' \in enable(m) \mid R(t') = [b, b], b \neq 1 \wedge b < a$.
4. Transitions Tirables dans un Intervalle (*Interval Firable Transitions* - IFT) :
   - $c.dF(e)$ est l'ensemble des transitions qui peuvent être tirées au moins après c ut et au plus tard après d ut, avec c le plus petit temps résiduel supérieur à 1 des transitions de cet ensemble : $t \in c.dF(e)$ iff $t \in enable(m)$, $R(t) = [c, d]$ and $\nexists t' \in enable(m) \mid R(t') = [j, k], c \neq 1 \wedge j < c$.

Pour un ensemble de transitions $K \in T^n$, on appelle $PowerSet(K)$ l'ensemble non vide de tous les sous-ensembles de K: $PowerSet(K) = \{B \in T^n | \ B \neq \emptyset, B \subset K\}$. Par exemple, le power set de $K = \{t_1, t_2\}$ est $PowerSet(K) = \{\{t_1\}, \{t_2\}, \{t_1, t_2\}\}$. Cela sera utilisé lors de la construction du SBG, pour construire les ensembles de transitions tirables à partir d'un état donné, comme expliqué dans la section suivante.

4.2 Construction du SBG

La construction du SBG est présentée en détails dans l'algorithme (1). Pour simplifier, cet algorithme ne détaille pas la gestion des conflits (qui est présentée séparément dans la prochaine

[3] On rappelle que d'après les règles de transformation de la section 3.1, les intervalles $[a, +\infty[$ avec $a \neq 1$ sont impossibles.

Algorithm 1: Algorithme de construction du SBG

Data: $e_0 = (m_0, R_0)$
Result: SBG

```
1  Algorithm algo(e)
2      if 1.1F(e) ≠ ∅ then                                              // cas (1)
3          e --1.1F(e),θ=1--» e';   algo(e');
4          if 1.bF(e) ≠ ∅ ∨ 1.∞F(e) ≠ ∅ then                             // cas (2)
5              for (∀w_i ∈ PowerSet(1.bF(e) ∪ 1.∞F(e)) do
6                  e --1.1F(e)∪w_i,θ=1--» e';   algo(e');
7              end
8          end
9      else
10         if 1.bF(e) ≠ ∅ ∨ 1.∞F(e) ≠ ∅ then                             // cas (3)
11             for (∀w_i ∈ PowerSet(1.bF(e) ∪ 1.∞F(e)) do
12                 e --w_i,θ=1--» e';   algo(e')
13             end
14             if 1.bF(e) = ∅ ∧ a.aF(e) = ∅ ∧ c.dF(e) ≠ ∅ then        // sous-cas (3.1)
                   /* boucle sur le même état                                */
15                 e --θ=1--» e
16             else
17                 e --θ=1--» e';   algo(e');
18             end
19         else
20             if a.aF(e) ≠ ∅ ∧ c.dF(e) ≠ ∅ then                         // cas (4)
                   /* ∀t ∈ a.aF(e), R(t) = [a, a] ∀t ∈ c.dF(d), R(t) = [c, d]  */
21                 if a < c then                                        // sous-cas (4.1)
22                     e --a.aF(e),θ=a--» e';   algo(e');
23                 else
24                     if a > c then                                    // sous-cas (4.2)
25                         e --θ=c−1--» e';   algo(e');
26                     else                                             // sous-cas (4.3)
27                         e --a.aF(e),θ=a--» e';   algo(e')
28                         for (∀w_i ∈ PowerSet(c.dF(e)) do
29                             e --a.aF(e)∪w_i,θ=a--» e';   algo(e');
30                         end
31                     end
32                 end
33             else
34                 if a.aF(e) ≠ ∅ then                                  // cas (5)
35                     e --a.aF(e),θ=a--» e';   algo(e');
36                 else                                                 // cas (6)
37                     e --θ=c−1--» e';   algo(e');
38                 end
39             end
40         end
41     end
```

section). A partir d'un état $e = (m, R)$, il existe un ou plusieurs ensembles de transitions tirables comme décrit section 4.1. En fonction de la combinaison des ces ensembles, nous pouvons définir l'ensemble $Fired$ des transitions qui seront simultanément tirées lors d'un changement d'états, ainsi que le temps θ associé. Les différents cas sont tous traités en détails dans l'algorithme (1), et décrits globalement ci-dessous, en prenant comme exemple d'illustration le modèle STPN et son graphe SBG présentés Figure 3.

4.2.1 Gestion de *NFT* seul :

Si NFT est non vide et qu'il n'y a pas de transition probablement tirable ($PFT = \emptyset$, que LFT et IFT soient vides ou non), alors toutes les transitions de NFT doivent être tirées simultanément après $1tu$ (cas (1) de l'Algorithme 1).

Exemple 2. *La seule transition sensibilisée dans l'état e_0 (t_0) appartient à NFT. Elle est donc tirée en $1ut$, ce qui conduit au nouvel état e_1.*

4.2.2 Gestion de *PFT (1.bF et 1.∞F) seuls :*

Dans le cas où seulement les ensembles PFT sont non vides, alors le nombre de changements d'états possible est calculé en utilisant le power set. Pour chaque sous-ensemble de transitions du power set, les transitions sont tirées simultanément après $1ut$ (cas (3)). Il est également possible qu'aucune transition ne soit tirée, alors seul le temps évolue de $1ut$: $\theta = 1$. Dans le cas où $1.\infty F(e)$ est le seul non vide (NFT, LFT,IFT and $1.bF(e)$ étant donc vides) alors cela n'entraine pas de changement d'états (boucle sur le même état, sous-cas (3.1)), sinon cela entraine un changement d'états avec évolution des intervalles temporels.

Dans le cas particulier où seul l'ensemble 1.∞F est non vide, ces transitions peuvent ne jamais être tirées, ou peuvent être tirées à n'importe quel moment. Donc, un changement d'états qui boucle sur le même état avec $\theta = 1$ permets d'exprimer la possibilité d'une évolution du temps indéfinie et/ou infinie sans tir de cette transition.

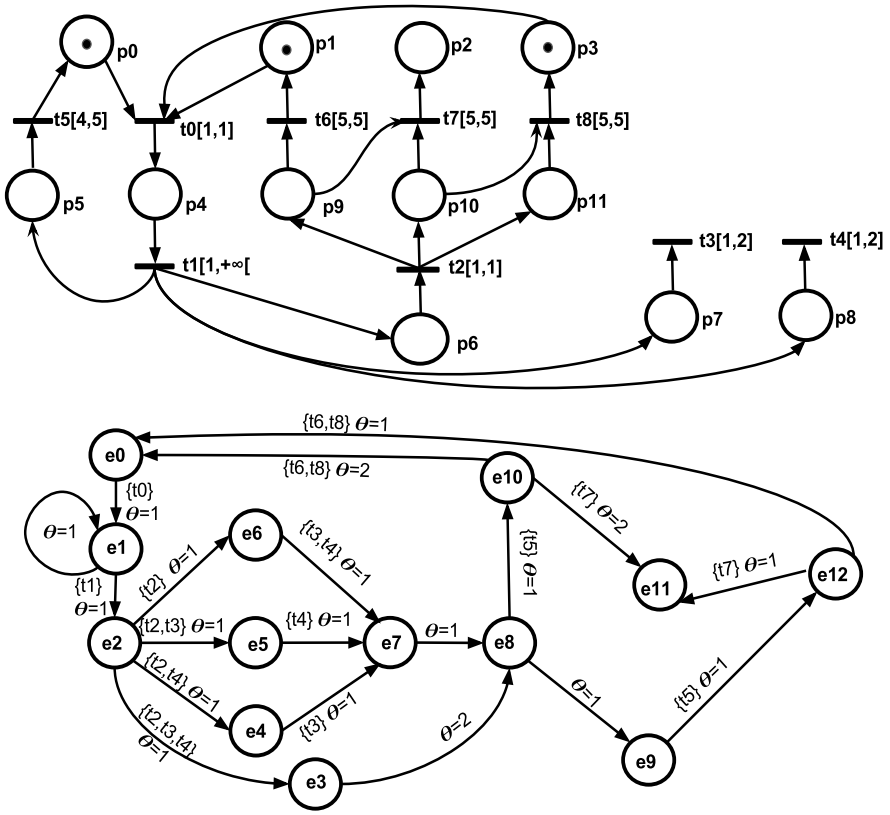
Exemple 3. *A partir de l'état e_1, deux changements d'états sont possibles: un tir de transition, lorsque t_1 est tirée avec $\theta = 1$ ce qui mène à e_2; ou une évolution temporelle de $\theta = 1$ sans changement d'état (sous-cas 3.1).*

4.2.3 Gestion de *NFT* et *PFT* ensemble :

La combinaison des cas précédents, c'est à dire lorsque que les deux ensembles NFT et PFT sont tous les deux non vides, est présentée dans le cas (2). Dans ce cas, les transitions de chacun des sous-ensembles de PFT seront tirées simultanément avec ceux de NFT, menant à chaque fois à un nouvel état. Si aucune transition de PFT n'est tirée, alors seules celles de NFT seront tirées simultanément.

Exemple 4. *Les transitions sensibilisées dans e_2 sont t_2, t_3, t_4 et t_5. La transition t_2 appartient à NFT, t_3 et t_4 appartiennent à PFT, et t_5 à ITF. Nous verrons plus tard comment sont gérées les transitions ITF, il suffit juste ici de savoir que t_5 n'est pas tirable. Ainsi, quatre changements d'états sont possibles, chacun menant à un nouvel état :*

- *Toutes les transitions sont tirées avec $\theta = 1$ et mènent à l'état e_3.*
- *Une seule transition entre t_3 et t_4 est tirée simultanément avec t_2, avec $\theta = 1$, chaque instance menant à nouvel état (e_5 et e_4, respectivement).*
- *t_2 est tirée seule avec $\theta = 1$, ce qui mène à l'état e_6.*



e0 p0,p1,p3 t0[1,1]	**e3** p5,p9,p10,p11 t5[3,4],t6[5,5], t7[5,5],t8[5,5]	**e6** p5,p7,p8,p9,p10,p11 t3[1,1],t4[1,1],t5[3,4] t6[5,5],t7[5,5],t8[5,5]	**e9** p5,p9,p10,p11 t5[1,1],t6[2,2], t7[2,2],t8[2,2]	**e12** p0,p9,p10,p11 t6[1,1],t7[1,1], t8[1,1]
e1 p4 t1[1,+∞[	**e4** p5,p7,p9,p10,p11 t3[1,1],t5[3,4], t6[5,5],t7[5,5],t8[5,5]	**e7** p5,p9,p10,p11 t5[2,3],t6[4,4], t7[4,4],t8[4,4]	**e10** p0,p9,p10,p11 t6[2,2],t7[2,2], t8[2,2]	
e2 p5,p6,p7,p8 t2[1,1],t3[1,2] t4[1,2],t5[4,5]	**e5** p5,p8,p9,p10,p11 t4[1,1],t5[3,4], t6[5,5],t7[5,5],t8[5,5]	**e8** p5,p9,p10,p11 t5[1,2],t6[3,3], t7[3,3],t8[3,3]	**e11** p0,p2,p11	

Figure 3: Exemple d'un STPN et de son SBG

4.2.4 Autre cas :

Les cas restants sont lorsque seuls LFT et IFT sont non vides. Dans ce cas, les transitions qui ont la plus petite borne inférieure des intervalles de temps résiduels seront tirées (cas (4.1), (4.3) et (5)). Si seulement IFT est non vide, ou si ce sont des transitions de IFT qui ont la plus petite borne inférieure c, alors seul le temps évolue de $c - 1$ tu (cas (4.2) et (6)) ce qui

mène à un nouvel état e'.

Exemple 5. *Cette situation peut être illustrée en étudiant l'état e_3. A partir de cet état, la transition sensibilisée possédant la plus petite borne inférieure est $t_5[3,4]$, qui appartient à IFT. Les autres transitions $t_6[5,5]$, $t_7[5,5]$ et $t_8[5,5]$ appartiennent à LFT. Ainsi, aucune transition n'est tirée à partir de e_3, et le temps évolue de $\theta = 2$ menant vers l'état e_8. En e_8, l'intervalle de la transition t_5 devient $[1,2]$, t_5 devient donc PFT.*

4.3 Gestion des conflits

Cette section explique notre méthode de gestion des conflits lors de la construction du SBG et en donne l'algorithme détaillé. Elle intervient dans l'algorithme 1, à chaque fois qu'un changement d'états basé sur le tir d'un ensemble de transitions est construit. L'Algorithme 2 de gestion des conflits prend alors en entrée cet ensemble de transitions tirables (que nous appellerons $Fired$). La première étape est de diviser cet ensemble en sous-ensemble de transitions ne contenant aucun conflit. Pour chaque couple de transitions en conflit, $Fired$ est divisé en sous-ensemble afin de séparer ces transitions. Ceci est appliqué récursivement sur chacun des sous-ensembles jusqu'à avoir séparé tous les conflits. Après suppression des doublons et des partitions, on obtient alors un ensemble WC de sous-ensembles de transitions tirables simultanément car libres de conflit. Chacun des sous-ensembles $Fired_i$ de WC mène alors à un nouvel état du graphe SBG : $\forall Fired_i \in WC(Fired), e \overset{Fired_i,\theta}{\twoheadrightarrow} e'_i$.

Algorithm 2: Algorithm Conflict Handling

```
Data: Fired ∈ T^n, e = (m, R)
Result: WC = {L ≠ ∅ | L ⊂ F}
WC ← ∅
Algorithm algo(Fired)
    if ∀t, t' ∈ F ∧ t' ∈ Conflict(m, t) then
        Fired' ← F − {t};
        Fired'' ← F − {t'};
        algo(Fired');
        algo(Fired'');
    else
        if ∀L ∈ WC ∧ F ⊄ L then
            WC ← Fired;
        else
            if L ⊂ Fired then
                WC − {L};
                WC ← Fired;
            end
        end
    end
    return WC
```

Exemple 6. *L'état e_{10} illustre la gestion des conflits : les transitions sensibilisées $t_6[2,2]$, $t_7[2,2]$ et $t_8[2,2]$ appartiennent à LFT, sont tirables mais sont en conflit (t_7 est en conflit avec t_6 et t_8, mais attention t_6 n'est pas en conflit avec t_8). En appliquant l'algorithme (2) sur $Fired = \{t_6, t_7, t_8\}$, nous obtenons $WC = \{\{t_7\}, \{t_6, t_8\}\}$. Chacun des sous-ensembles de WC est tiré avec $\theta = 2$, et ils mènent respectivement aux états e_{11} and e_0.*

5 Brève comparaison des principes d'analyse

Afin de prouver la validité de notre approche, les algorithmes exposés dans cet article ont été implémentés afin d'effectuer un ensemble de tests sur des exemples types. Il est cependant difficile d'estimer l'efficacité de notre méthode en la comparant à d'autre méthodes existantes, car à notre connaissance aucune méthode ne prend exactement en considération tous nos critères pour l'analyse. Ainsi, nous allons comparer, sur un exemple caractéristique, notre approche aux méthodes les plus proches : l'approche asynchrone proposée dans [8] qui utilise une transformation de modèles puis le graphe des classes d'états (SCG) [3]; et l'approche à temps discret basée sur les *Integer-State* (ISG) [14]. Cette comparaison se base, à partir des graphes d'états construits par ces 3 méthodes, sur la précision des analyses d'accessibilité et des traces temporelles.

L'approche SCG est adaptée aux réseaux de Petri temporels asynchrone à temps continu. Dans ce contexte, un état peut avoir une infinité de successeurs par le tir d'une transition tirable à tous les instants d'un intervalle continu. La méthode SCG regroupe tous ces successeurs en une seule classe d'équivalence. Une classe c est definie par un couple (m, D), avec m le marquage et D le domaine de tir des transitions, défini par l'ensemble des contraintes temporelles de tir des transitions sensibilisées par ce marquage.

Dans l'approche ISG, un état est un couple $z = (m, h)$, avec m le marquage et h le vecteur des horloges (relatives) des transitions. L'horloge d'une transition sensibilisée est définie par le temps écoulé depuis son instant de sensibilisation. Si la transition est désensibilisée, son horloge est désignée par #.

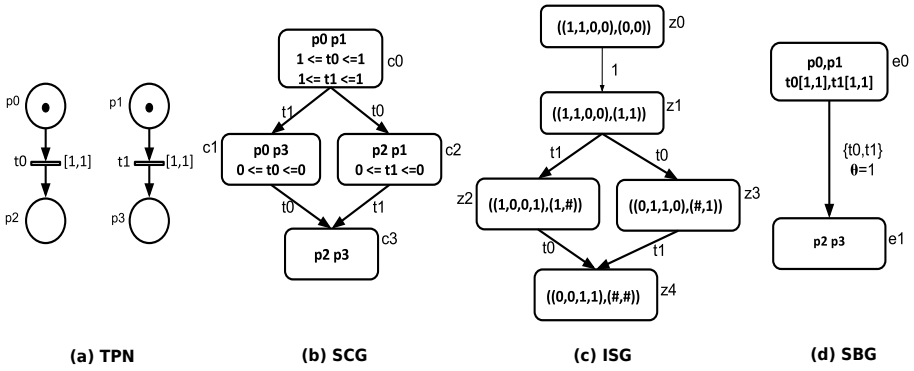


Figure 4: Illustration sur un exemple

5.1 Précision de l'atteignabilité de marquage

Considérons le modèle TPN donné Figure 4(a), son SCG Figure 4(b), son ISG Figure 4(c) et son SBG Figure 4(d). Nous voyons que certains marquages existent dans le SCG (par exemple dans les états c_1 et c_2) et dans l'ISG (z_2 et z_3) alors qu'ils ne sont pas présents dans le SBG. En effet, les deux premières méthodes traitent les transitions parallèles par entrelacement, c'est à dire qu'elles font la différentiation de l'ordre de tir de ces transitions parallèles. Cela génère des marquages qui n'existent pas dans notre contexte d'exécution synchrone, dans lequel les transitions parallèles sont tirées simultanément. Ainsi, si un état est atteignable dans le SCG

ou l'ISG, en termes de marquages, nous ne pouvons pas savoir a priori si c'est un état réel ou non. Par contre, si un marquage est inatteignable dans le SCG ou l'ISG, nous pouvons être sûr qu'il ne le sera pas non plus dans l'espace d'état réel. Ceci limite les possibilités d'analyse d'atteignabilité. Au contraire, notre SBG respecte exactement les contraintes de tirs synchrones, et donc par conséquence préserve les marquages réels.

5.2 Précision des traces temporelles

Dans les systèmes temps réel, l'analyse des traces (quelles transitions sont tirées et dans quel ordre) est important, principalement en considérant la dimension temporelle (à quel moment). Cependant, les propriétés temporelles ne peuvent pas être vérifiées précisément dans le SCG, puisque le graphe est une abstraction de l'espace d'états (qui est infini) en effectuant un sur-ensemble des intervalles temporels. De plus, la valeur absolue des horloges est non stockée dans le graphe, ce qui rend difficile une analyse portant sur la dimension temporelle des traces. Ces différentes affirmations sont présentées dans [11], [6]. Même si plusieurs méthodes ont été proposées pour remédier à ce problème [4] pour le SCG, les méthodes SBG et ISG étant énumeratives sur le temps discret, elles conservent de façon plus précise les valeurs des horloges ainsi que les traces temporelles.

6 Conclusion

Cet article présente une nouvelle méthode pour analyser les réseaux de Petri, en prenant en considération les contraintes d'exécution impliquées par l'implémentation sur la cible matérielle. Dans notre cas, nous considérons le contexte spécifique des systèmes numériques temps-réel critiques exécutés de façon synchrone sur un FPGA. Cela implique du parallélisme réel et une synchronisation sur l'horloge du système. Ce contexte nécessite donc une sémantique très spécifique des réseaux de Petri temporelles, puisque toutes les transitions sont tirées sur des fronts d'horloge (temps discret) et que plusieurs transitions tirables au même moment sont alors simultanément tirées. De plus, pour être au plus près de la réalité, nous ne considérons pas le tir des transitions comme instantané : le tir d'une (ou d'un ensemble de) transition(s) prend 1 unité de temps. Les méthodes et outils d'analyse existants ne prennent pas en considération toutes ces caractéristiques. Ainsi, nous avons proposé une nouvelle méthode d'analyse qui consiste en : 1) une transformation de modèles, qui modifie le modèle initial, exprimé en réseaux de Petri temporels interprétés, en un modèle exprimé en réseaux de Petri temporels synchrones; Cette transformation permet de prendre en compte les contraintes spécifiques liées à l'interprétation et à l'exécution synchrone; 2) un nouveau graphe d'états, appelé le graphe de comportement synchrone (*Synchronous Behaviour Graph* - SBG), qui construit l'espace d'états du modèle résultant de l'étape 1. Le SBG tient compte de toutes les contraintes d'exécution liées à notre contexte. Nous avons défini formellement ce graphe et sa sémantique, donné son algorithme de construction, et nous l'avons illustré sur un exemple simple.

Afin d'améliorer les performances de notre approche, nous nous intéressons à la réduction de notre graphe SBG, car l'énumération du temps discret peut mener à une taille importante de l'espace d'états. La littérature fournit de nombreuses techniques dont il est possible de s'inspirer, comme les classes d'équivalence ou le graphe de région des réseaux de Petri, mais ces méthodes devront être adaptées au temps discret et à notre sémantique de tir multiple. Il est également prévu de compléter notre approche avec la sémantique particulière de blocage de transition comme défini dans [10]. Cela permettra de gérer l'association sur un même transition d'un intervalle temporel et d'une condition, ce qui peut mener à un blocage si la condition est

toujours fausse lorsque la borne supérieure de l'intervalle est atteinte. Enfin, une perspective de nos travaux est de compléter le processus de conception de nos systèmes par l'ajout d'un outil de vérification formelle : nous voulons relier notre graphe de comportement synchrone à un *model checker*, ce qui permettra la vérification de propriétés exprimées en logique temporelles (CTL, LTL, ou TCTL..).

References

[1] D. Andreu, D. Guiraud, and G. Souquet. A distributed architecture for activating the peripheral nervous system. *Journal of neural engineering*, 6(2):026001, 2009.

[2] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time petri nets. *IEEE transactions on software engineering*, 17(3):259–273, 1991.

[3] B. Berthomieu, P-O Ribet, and F. Vernadat. The tool TINA–construction of abstract state spaces for petri nets and time petri nets. *Int. journal of production research*, 42(14):2741–2756, 2004.

[4] B. Berthomieu and F. Vernadat. State class constructions for branching analysis of time petri nets. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 442–457. Springer, 2003.

[5] R. David and H. Alla. *Discrete, continuous, and hybrid Petri nets*. Springer Science & Business Media, 2010.

[6] G. Gardey, O. H. Roux, and O. F. Roux. Using zone graph method for computing the state space of a time petri net. *Formal modeling and analysis of timed systems*, pages 246–259, 2004.

[7] I. Grobelna and M. Adamski. Model checking of control interpreted petri nets. In *18th International Conference Mixed Design of Integrated Circuits and Systems, (MIXDES 2011)*, Gliwice, Poland, 2011.

[8] H. Leroux, D. Andreu, and K. Godary-Dejean. Handling exceptions in petri net-based digital architecture: from formalism to implementation on fpgas. *IEEE Transactions on Industrial Informatics*, 11(4):897–906, 2015.

[9] H. Leroux, K. Godary-Dejean, and D. Andreu. Complex digital system design: A methodology and its application to medical implants. In *International Workshop on Formal Methods for Industrial Critical Systems*, pages 94–107. Springer, 2013.

[10] H. Leroux, K. Godary-Dejean, and D. Andreu. Integrating implementation properties in analysis of petri nets handling exceptions. *IFAC Proceedings Volumes*, 47(2):406–411, 2014.

[11] D. Lime and O. H. Roux. State class timed automaton of a time petri net. In *The 10th International Workshop on Petri Nets and Performance Models (PNPM'03)*, 2003.

[12] P. Merlin and D. Farber. Recoverability of communication protocols–implications of a theoretical study. *IEEE transactions on Communications*, 24(9):1036–1043, 1976.

[13] M. Moalla, J. Pulou, and J. Sifakis. Synchronized petri nets: A model for the description of non-autonomous sytems. *Mathematical Foundations of Computer Science 1978*, pages 374–384, 1978.

[14] L. Popova-Zeugmann. On time petri nets. *Elektronische Informationsverarbeitung und Kybernetik*, 27(4):227–244, 1991.

[15] L. Popova-Zeugmann. *Essential states in time Petri nets*. Humboldt-Univ. zu Berlin, 1998.

[16] J. R. Silva and P. M. G. Del Foyo. *Timed petri nets*. INTECH Open Access Publisher, 2012.

[17] G. Souquet, D. Andreu, and D. Guiraud. Petri nets based methodology for communicating neuroprosthesis design and prototyping. In *ISABEL'08: 1st International Symposium on Applied Sciences in Biomedical and Communication Technologies*, 2008.

[18] F. Wagner, P. Münch, S. Liu, and G. Frey. Development process for dependable high-performance controllers using petri nets and FPGA technology. In *1st IFAC Workshop on Dependable Control of Discrete Systems, (DCDS 2007)*, volume 40, pages 139–144, 2007.