Synthèse de contraintes temporisées pour une architecture d'automatisation en réseau

Étienne André* — **Thomas Chatain*** — **Olivier De Smet**** — **Laurent Fribourg*** — **Silvain Ruel****

* *LSV, ENS Cachan & CNRS, France*
{andre, chatain, fribourg}@lsv.ens-cachan.fr

** *LURPA, ENS Cachan, Cnam, France*
{de_smet, ruel}@lurpa.ens-cachan.fr

RÉSUMÉ. *Nous nous intéressons au problème de la synthèse de contraintes temporelles pour des systèmes concurrents. Ces systèmes sont modélisés par des réseaux d'automates temporisés dans lesquels certaines constantes, représentées par des paramètres, sont ajustables. Une valeur convenable de ces paramètres est supposée connue grâce à un processus de simulation. Nous présentons une méthode qui permet de définir une zone de bon fonctionnement autour de ce point de référence. Cette zone est définie par un système d'inégalités linéaires entre les paramètres. Cette méthode est appliquée à l'étude d'une architecture d'automatisation en réseau.*

ABSTRACT. *We deal with the problem of synthesis of timing constraints for concurrent systems. Such systems are modeled by networks of timed automata where some constants, represented as parameters, can be tuned. A suitable value of these parameters is assumed to be known from a preliminarily simulation process. We present a method which infers a zone of suitable points around this reference functioning point. This zone is defined by a system of linear inequalities over the parameters. This method is applied to the case study of a networked automation system.*

MOTS-CLÉS : *architecture d'automatisation en réseau, réseau d'automates temporisés, model-checking paramétré, synthèse de contraintes*

KEYWORDS: *Networked automation system, Network of timed automata, Parameteric model-checking, Constraints synthesis*

1. Introduction

On trouve aujourd'hui une grande variété de systèmes automatisés, par exemple dans les transports ou dans les systèmes industriels. De plus en plus souvent, la fonctionnalité de contrôle de ces systèmes a la particularité d'être distribuée sur plusieurs contrôleurs reliés par un réseau, lequel peut être soit un réseau spécialisé, soit, de plus en plus souvent, un réseau généraliste de type Ethernet. Ce réseau permet de séparer les modules d'entrée-sortie des contrôleurs qui effectuent les calculs (Thomesse, 1999). Il est indispensable d'analyser les performances temporelles de ces systèmes distribués, mais la construction de modèles satisfaisants pour ce type de systèmes est délicate et génère des modèles très complexes. Ainsi, les analyses de performances temporelles sont souvent effectuées par simulation à partir de scénarios d'entrée donnés (Jasperneite *et al.*, 2002; Vitturi, 2003). Une approche alternative consiste à utiliser des méthodes exhaustives comme le model-checking (Clarke *et al.*, 1999). Cependant, ces méthodes sont très coûteuses et ne conviennent pas aux grands systèmes, à moins de travailler sur une abstraction du système (Marsal *et al.*, 2005).

Récemment, une forme de model-checking temporisé paramétré a permis de déterminer les dépendances entre différents paramètres temporels de circuits matériels (Clarisó *et al.*, 2004; Chevallier *et al.*, 2009). Dans la lignée de ces travaux, nous proposons une approche pour le calage de paramètres temporels lors de la conception d'une architecture d'automatisation en réseau (Ruel *et al.*, 2008; Denis *et al.*, 2007).

Le système est modélisé par un réseau d'automates temporisés (Alur *et al.*, 1994) dans lesquels certaines constantes sont ajustables : elles sont donc représentées par des paramètres pour indiquer que le concepteur a une certaine liberté. De plus, une valeur convenable des paramètres est supposée préalablement connue grâce à un processus de simulation (Ruel *et al.*, 2008; Denis *et al.*, 2007). Nous présentons ici une méthode, schématisée sur la figure 1, qui permet de définir une zone de bon fonctionnement autour de ce point de référence. Cette zone est définie par un système d'inégalités linéaires entre les paramètres. Pour toute valeur des paramètres qui satisfait ces contraintes, nous garantissons que le système se comporte de la même façon que le système de référence. Notre méthode permet d'obtenir directement une zone dense de points de bon fonctionnement. Elle offre donc une alternative intéressante aux approches habituelles qui recherchent les points de bon fonctionnement par exploration dichotomique d'une zone supposée pertinente.

Nous décrivons tout d'abord l'architecture d'automatisation en réseau et sa modélisation (section 2). Nous présentons ensuite notre méthode pour rechercher une zone de bon fonctionnement autour d'un point de référence (section 3). Enfin, nous comparons les résultats obtenus par notre méthode à d'autres résultats obtenus par model-checking instancié (section 4).

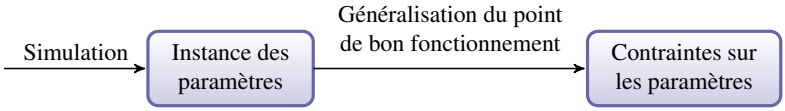

Figure 1. *Le principe de notre méthode*

2. Présentation du problème

2.1. *Définition du formalisme utilisé*

Automates temporisés. Un *automate temporisé* (Alur *et al.*, 1994) est un 5-uplet $\mathcal{A} = (\Sigma, Q, \mathcal{Z}, I, \mathcal{T})$ où Σ est un alphabet fini d'*actions* ; Q est un ensemble fini d'*états* ; $\mathcal{Z}$ est un ensemble fini d'horloges ; I est un *invariant* qui associe à chaque état $q \in Q$ une conjonction I_q d'inégalités de la forme $z \leq u$, z étant une horloge et u un entier ; la relation de *transition* $\mathcal{T}$ est un ensemble d'éléments de la forme (q, g, a, ρ, q') où q et q' sont des états, $a \in \Sigma$ est une action, la garde g est une conjonction de formules de la forme $z \geq l$, z étant une horloge et l un entier, et $\rho \subseteq \mathcal{Z}$ est l'ensemble des horloges réinitialisées par la transition.

Une *valuation des horloges* est une fonction $v : \mathcal{Z} \to \mathcal{R}_{\geq 0}$ et une *configuration* de l'automate est un couple (q, v) constitué d'un état et d'une valuation des horloges. Par abus, nous notons aussi ρ la fonction de réinitialisation des horloges définie par $\rho(v)(z) = 0$ si $z \in \rho$ et $\rho(v)(z) = v(z)$ sinon. Nous notons $v + d$ la valuation des horloges obtenue en ajoutant d à toutes les horloges, pour $d \in \mathcal{R}_{\geq 0}$.

L'évolution de l'automate se fait par alternance de mouvements discrets et de passage du temps décrits comme suit :

– *mouvement discret* : $(q, v) \xrightarrow{a} (q', v')$ avec une transition $(q, g, a, \rho, q') \in \mathcal{T}$ telle que v satisfait la garde g et $v' = \rho(v)$.

– *passage du temps* : $(q, v) \xrightarrow{d} (q, v + d)$ avec $d \in \mathcal{R}_{\geq 0}$ tel que $v + d$ satisfait l'invariant I_q.

Un pas de l'automate est un mouvement discret suivi d'un passage du temps $(q, v) \xrightarrow{a} (q', v') \xrightarrow{d} (q', v'')$, que l'on note $(q, v) \overset{a}{\Rightarrow} (q', v'')$. Une *exécution* ξ de l'automate depuis la configuration (q_0, v_0) est une suite finie de pas de la forme $(q_0, v_0) \overset{a_1}{\Rightarrow} (q_1, v_1) \overset{a_2}{\Rightarrow} \cdots \overset{a_k}{\Rightarrow} (q_k, v_k) \overset{a_{k+1}}{\Rightarrow} (q_k, v_k + d_{k+1})$.

Réseaux d'automates temporisés. Les interactions entre plusieurs automates sont décrites au moyen d'un alphabet commun d'actions Σ au sens de la théorie des traces (Diekert *et al.*, 1995). Soit $\Sigma^i \subseteq \Sigma$ l'alphabet d'actions local de chaque automate $\mathcal{A}^i = (\Sigma^i, Q^i, \mathcal{Z}^i, I^i, \mathcal{T}^i)$. La sémantique de la composition des automates impose que chaque $\mathcal{A}^i$ tel que $a \in \Sigma^i$ participe à toute transition globale étiquetée par

l'action a. L'automate temporisé global obtenu par la composition $\mathcal{A}^1 \| \mathcal{A}^2 \| \cdots \| \mathcal{A}^n$ est $\mathcal{A} = (\Sigma, Q, \mathcal{Z}, I, \mathcal{T})$ avec $\Sigma = \bigcup_{i=1}^{n} \Sigma^i$, $Q = \Pi_{i=1}^{n} Q_i$, $\mathcal{Z} = \bigcup_{i=1}^{n} \mathcal{Z}^i$, $I = \bigwedge_{i=1}^{n} I^i$ et $\mathcal{T}$ est l'ensemble des

$$((q^1, \ldots, q^n), \bigwedge_{\substack{i=1,\ldots,n \\ a \in \Sigma^i}} g^i, a, \bigcup_{\substack{i=1,\ldots,n \\ a \in \Sigma^i}} \rho^i, (q'^1, \ldots, q'^n))$$

tels que pour tout $i = 1, \ldots, n$, soit a est une action de l'automate $\mathcal{A}^i$ ($a \in \Sigma^i$) et $(q^i, g^i, a, \rho^i, q'^i) \in \mathcal{T}^i$ est une transition de $\mathcal{A}^i$, soit $a \notin \Sigma_i$ et $q^i = q'^i$.

Réseaux d'automates temporisés paramétrés. Dans cet article nous nous intéressons à des réseaux d'automates temporisés *paramétrés* (Alur *et al.*, 1993), c'est-à-dire que certaines des constantes entières qui apparaissent dans les gardes et les invariants peuvent être remplacées par des paramètres $\{p_1, \ldots, p_m\}$. Étant donné un automate temporisé paramétré $\mathcal{A}$ et une valuation π des paramètres du système, on note $\mathcal{A}[\pi]$ l'automate temporisé classique obtenu en remplaçant chaque paramètre par sa valeur numérique définie dans π. Nous désignons par $\mathcal{A}(K)$ le réseau d'automates temporisés paramétrés $\mathcal{A}$ dont les paramètres sont contraints par l'ensemble K d'inégalités linéaires entre les paramètres. Les comportements de $\mathcal{A}(K)$ sont la réunion des comportements des $\mathcal{A}[\pi]$ pour tous les valuations π qui respectent la contrainte K. Rappelons que le problème d'accessibilité est indécidable pour les réseaux d'automates temporisés paramétrés (Alur *et al.*, 1993).

2.2. *Étude de cas*

Nous nous intéressons aux architectures d'automatisation en réseau (AAR). La structure physique de ces architectures est illustrée sur la figure 2 (Ruel *et al.*, 2008; Denis *et al.*, 2007). Les contrôleurs (Automate Programmable Industriel (API)) sont modulaires et composés d'un processeur de calcul (PLC) qui traite de façon cylique le programme de commande, tandis que le processeur de communication (COM) interroge périodiquement les modules d'entrée/sortie déportées (MES). Ces deux cycles sont asynchrones ; les échanges de données entre ces processeurs se font par des mémoires partagées. Le réseau (NET) comprend des commutateurs Ethernet et des liens filaires. Il est dédié aux communications entre les API et MES. On considère qu'il n'y a pas de perte de trames dans ce réseau commuté. Les entrées et les sorties sont regroupées sur les MES.

Dans la suite, nous utiliserons une AAR composée d'un contrôleur (API et COM), d'un commutateur Ethernet (NET) et d'un MES. Nous nous restreignons à un contrôleur qui traite une seule entrée, et commande une seule sortie. Nous avons retenu un réseau basé sur Modbus-TCP, bien que la méthodologie présentée puisse s'appliquer à d'autres types de réseaux.

Lors du développement d'une AAR, le choix des composants a des conséquences sur les paramètres de réglages, en particulier les délais. Par exemple, quand un modèle

de MES est choisi, le paramètre « temps de réponse à une requête réseau » sera fixé, par exemple, à $750\,\mu s$. De même, le réglage de la séquence de scrutation des MES par un contrôleur fige le paramètre « période de rafraîchissement » à, par exemple, 1000 ms. Notons $(p_1, \ldots, p_n)$ le n-uplet des paramètres de l'AAR et $d = (d_1, \ldots, d_n)$ un n-uplet où d_i est la valeur du paramètre p_i. En réglant le paramètre p_i, le concepteur recherche une valeur satisfaisante de la durée entre une variation du signal d'entrée et sa répercussion sur le signal de sortie. Ce temps de réponse, noté $SIGrt$, doit être inférieur à une limite donnée pour garantir les performances du système complet.

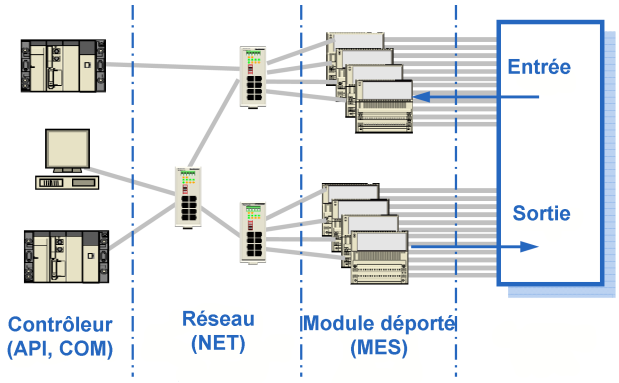

Figure 2. *Exemple d'architecture d'automatisation en réseau*

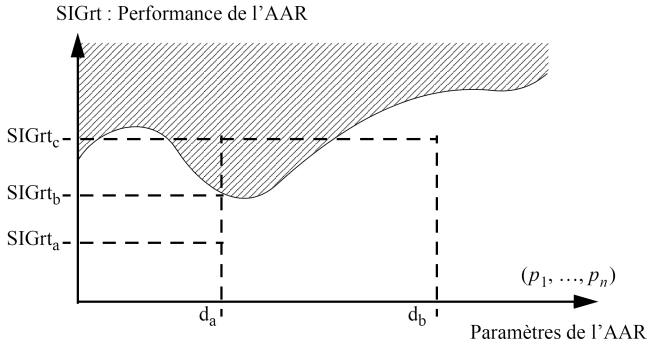

Figure 3. *Performance d'une AAR ($SIGrt = f(d_a)$)*

La qualification d'une AAR est difficile. Pour un jeu de paramètres $d = (d_1, \ldots, d_n)$, chaque occurrence du signal d'entrée peut produire un délai de réponse différent. Notons $SIGrt_b$ la borne supérieure de toutes les valeurs possibles de $SIGrt$ pour un jeu de paramètres d. La ligne horizontale $SIGrt_a$ sur la figure 3 représente la performance d'une AAR attendue par le concepteur. Dans ce cas, on ne peut pas respecter cette exigence avec l'AAR instanciée par d_a. On peut en effet atteindre une valeur $SIGrt_b$ de la performance attendue, supérieure à $SIGrt_a$. En revanche, une

exigence $SIGrt_c$ est valide pour l'AAR paramétrée par d_a. On peut remarquer que, avec d_b comme jeu de paramètres de l'AAR, la performance n'est pas respectée.

2.3. *Modélisation*

L'outil UPPAAL (Larsen *et al.*, 1997) est utilisé pour vérifier les modèles du système que l'on va construire.

Nous construisons un modèle formé de plusieurs composants, qui se synchronisent par un alphabet d'actions partagées. La synchronisation de deux composants sur une action a est représentée par l'exécution simultanée d'une action $a!$ sur l'un des composants, et d'une action $a?$ sur l'autre.

Comme on souhaite observer uniquement la propagation d'un évènement d'entrée (passage d'une entrée booléenne de 0 à 1), puis celle de l'évènement de sortie corrélé (passage d'une sortie booléenne de 0 à 1), les évolutions des signaux de 1 à 0 ne seront pas modélisées, ce qui simplifiera les modèles.

2.3.1. *Processeur de calcul*

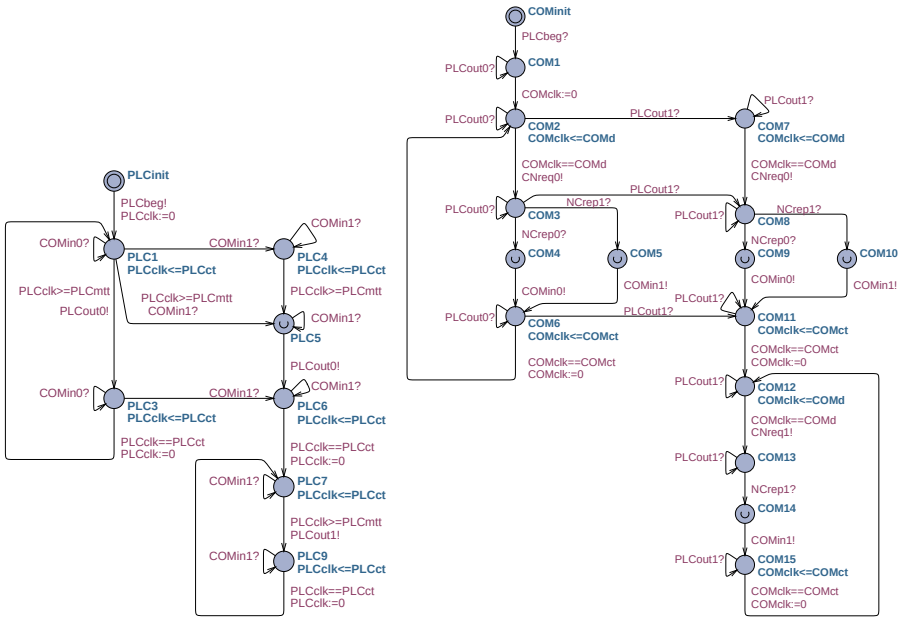

Figure 4. *Modèle de l'API (processeur de calcul à gauche, module de communication à droite)*

Le contrôleur choisi est de type API, constitué d'un processeur de calcul et d'un module de communication. Le processeur de calcul a un comportement périodique

composé de quatre tâches : lecture des entrées (durée fixe), exécution du programme (durée variable), écriture des sorties (durée fixe) et exécution d'autres traitements (durée variable). Dans un souci de simplification, le temps de cycle, somme de ces durées, est supposé constant. Nous considérons que les tâches de lecture des entrées et d'écriture des sorties sont ici à temps nul (leur durée est petite devant celle des exécutions du programme et des autres traitements).

Pour garder une cohérence lors de l'évolution du système complet, le processeur de calcul est le premier modèle a évoluer (pas de garde sur l'état $PLCinit$). L'horloge associée au modèle est $PLCclk$, le temps minimum d'exécution du programme est $PLCmtt$ et le temps de cycle de l'API est $PLCct$. Les transitions étiquetées $COMin0$, $COMin1$, $PLCout0$ et $PLCout1$ représentent les changements de valeur des entrées et des sorties.

2.3.2. *Module de communication*

Le module de communication a un comportement périodique : lecture des sorties de l'API, envoi de la trame au MES, attente de la réponse du MES, écriture vers les entrées de l'API et attente de la fin de la période. La durée de la période est modélisée par un délai fixe $COMct$. La réponse d'un MES après l'envoi de la trame et avant la fin du cycle est une hypothèse de modélisation correcte pour le type de composant utilisé. Le non-respect de cette hypothèse sera détecté par le blocage *(dead-lock)* du modèle. Les paramètres du modèle sont l'horloge du modèle ($COMclk$), le délai d'émission ($COMd$) et le temps de cycle ($COMct$).

2.3.3. *Réseau*

Le réseau est toujours disponible pour transmettre les trames, il n'y a donc pas d'initialisation. Les paramètres du réseau sont son horloge $NETclk$, et la durée de transmission d'une trame $NETd$.

Les travaux de (Marsal, 2006) et (Denis *et al.*, 2007) montrent que l'impact du réseau sur les performances temporelles d'une AAR sont faibles. On conserve seulement un délai de transmission constant ($NETd$) pour les deux sens de communication.

2.3.4. *Module d'entrée/sortie déportées*

Le MES a un comportement réactif, c'est-à-dire qu'il réagit immédiatement au changement de valeur d'une des entrées. $SIGin1?$ représente un passage de 0 à 1 du signal d'entrée. Pour les sorties, la réception d'une requête $NRreq0?$ ou $NRreq1?$ du module de communication via le réseau déclenche l'évolution du modèle. Après avoir attendu un délai fixe ($RIOd$) qui représente la durée de l'élaboration de la réponse, le MES émet la sortie vers le système ($SIGout0!$ ou $SIGout1!$), et émet simultanément la réponse vers le réseau ($RNrep0!$ ou $RNrep0!$). Cette simultanéité est modélisée avec un état « urgent » (décrit sur le schéma par un « U » dans l'état), c'est-à-dire qu'il n'est possible d'y rester que pendant un temps nul.

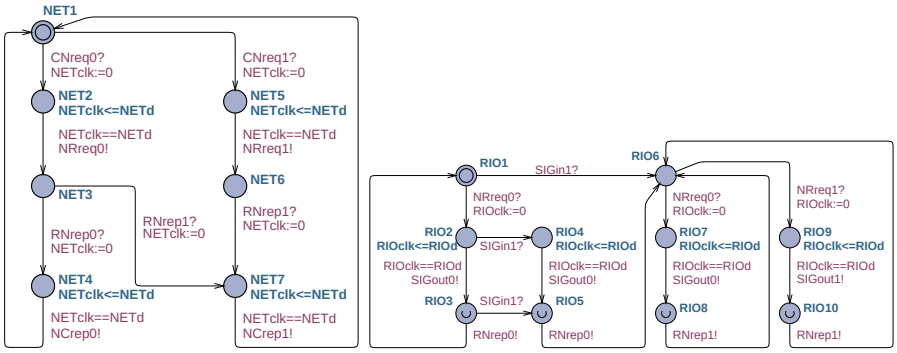

Figure 5. *Modèle du NET (à gauche) et du MES (à droite)*

2.3.5. *Environnement*

L'environnement génère d'abord l'évènement d'entrée, puis observe l'apparition de la réponse corrélée. L'évènement est émis lorsque tout le système est initialisé, c'est-à-dire après la première émission des sorties du MES ($SIGout0?$ sur $ENVinit$) mais sans synchronisation ultérieure (pas de garde sur la transition sortante de $ENV1$). Après l'émission de l'évènement d'entrée ($SIGin1!$), l'horloge $ENVclk$ mesure le délai jusqu'à l'apparition de $SIGout1?$. Le délai est ensuite comparé à $SIGrt$, et le système atteint $ENV6$, $ENV7$ ou $ENV8$ selon la valeur du délai mesuré par rapport au paramètre $SIGrt$.

On peut remarquer qu'il n'y a pas de synchronisation entre les modèles précédents et l'environnement, mais que l'on impose seulement un ordre lors des initialisations : le processeur de calcul démarre à n'importe quel moment, puis le module de communication démarre après le processeur, et ainsi de suite pour les autres composants. Cet ordonnancement entre les initialisations évitera d'avoir à parcourir un espace d'états trop important lors des analyses.

2.4. *Obtention d'un modèle* HYTECH

Nous cherchons à construire un modèle HYTECH pour en faire une analyse temporisée paramétrée (Alur *et al.*, 1993). L'outil HYTECH (Henzinger *et al.*, 1997) est principalement utilisé pour l'analyse de systèmes hybrides linéaires, dont les réseaux d'automates temporisés font partie. La fonctionnalité de HYTECH qui nous intéresse particulièrement ici est la possibilité de manipuler des délais paramétrés. Le model-checking paramétré est utilisé pour extraire les contraintes linéaires qui caractérisent les zones de bon fonctionnement.

Cette nécessité d'obtenir un modèle HYTECH nous interdit d'utiliser certaines constructions qui sont disponibles dans UPPAAL par exemple. En l'occurrence nous

avons dû éliminer les états urgents ou *committed* et écrire les synchronisations sous la forme de rendez-vous symétriques (pas d'émetteur et de récepteur).

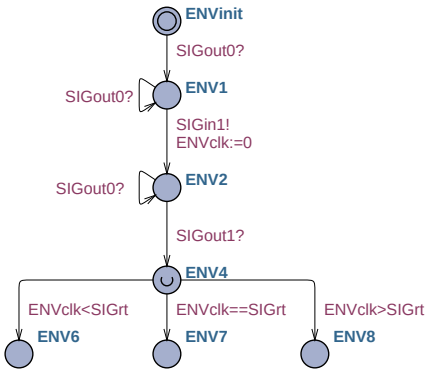

Figure 6. *Modèle de l'environnement*

3. Recherche d'une zone de bon fonctionnement autour d'un point de référence

3.1. *Généralisation d'un point de bon fonctionnement*

Un point de bon fonctionnement correspond à une *instanciation de chaque paramètre temporel* de la spécification. La vérification de la correction de ce point de référence est effectuée par un model-checking temporisé standard, à l'aide de l'outil UPPAAL comme indiqué dans la section 2.

Nous décrivons dans cette section une *méthode inverse* permettant de généraliser un bon point de fonctionnement dans le cadre des automates temporisés paramétrés (Alur *et al.*, 1993). Rappelons que les automates temporisés paramétrés sont une extension des automates temporisés classiques autorisant l'utilisation de *paramètres* dans les gardes et invariants en lieu et place des constantes (voir section 2.1). Étant donné un automate temporisé paramétré $\mathcal{A}$ et une valuation π des paramètres du système, on note $\mathcal{A}[\pi]$ l'automate temporisé classique obtenu en remplaçant chaque paramètre par sa valeur numérique définie dans π.

Un problème classique du cadre des automates temporisés paramétrés consiste à calculer une contrainte sur les paramètres du système, telle que, pour toute valuation des paramètres vérifiant cette contrainte, le système se comportera d'une façon définie comme *correcte*. Nous considérons ici le *problème inverse* (André *et al.*, 2009) : étant donné un automate temporisé paramétré $\mathcal{A}$ et une valuation de référence des paramètres π_0, nous cherchons à obtenir une contrainte K_0 sur les paramètres vérifiée par π_0 (noté $\pi_0 \models K_0$) telle que, pour toute valuation π des paramètres vérifiant K_0, le système $\mathcal{A}[\pi]$ se comportera de manière *équivalente* au système de référence $\mathcal{A}[\pi_0]$. Ainsi, si la valuation de référence π_0 correspond à un bon comportement, tel que l'im-

possibilité d'atteindre un mauvais état, alors la contrainte K_0 garantit elle-même ce bon comportement.

Cette notion d'équivalence entre $\mathcal{A}[\pi]$ et $\mathcal{A}[\pi_0]$ est en fait une *équivalence abstraction faite du temps* (Alur *et al.*, 1994), c'est-à-dire que les traces d'exécution de $\mathcal{A}[\pi]$ et $\mathcal{A}[\pi_0]$, vues comme des séquences alternantes d'états et de transitions, sont identiques.

L'algorithme *MéthodeInverse* (André *et al.*, 2009) résolvant ce problème inverse prend en entrée un système modélisé par un automate temporisé paramétré $\mathcal{A}$ d'état de contrôle initial q_0, ainsi qu'une valuation des paramètres π_0. Il calcule une contrainte K_0 sur les paramètres telle que :

1) $\pi_0 \models K_0$ et

2) pour tout $\pi \models K_0$, $\mathcal{A}[\pi]$ et $\mathcal{A}[\pi_0]$ sont *équivalents abstraction faite du temps*.

L'algorithme *MéthodeInverse* peut être résumé ainsi : en partant de $K_0 := \texttt{vrai}$, on calcule itérativement l'ensemble des états symboliques accessibles. Un état symbolique du système est un couple (q, C), où q est un état de contrôle de $\mathcal{A}$, et C une contrainte sur les paramètres du système[1]. Lorsqu'un état π_0-*incompatible* (q, C) est rencontré (c'est-à-dire lorsque $\pi_0 \not\models C$), K_0 est raffinée de la façon suivante : une inégalité π_0-*incompatible* J (c'est-à-dire telle que $\pi_0 \not\models J$) est sélectionnée dans C, et sa négation $\neg J$ est ajoutée à K_0. Cette procédure est alors relancée avec K_0 actualisée, et ainsi de suite, jusqu'à ce que l'ensemble complet des états accessibles (noté $Post^*$) soit atteint. Une version simplifiée de l'algorithme est donnée sur la figure 7, où les variables d'horloge ont notamment été omises dans un souci de simplicité. On note $Post^i_{\mathcal{A}}(S)$ l'ensemble des états symboliques accessibles depuis S en, au plus, i pas dans $\mathcal{A}$.

3.2. *Implémentation de la méthode inverse*

Une implémentation de cet algorithme — nommée IMITATOR, pour *Inverse Method for Inferring Time AbstracT behaviOR* (André, 2009) — a été réalisée sous la forme d'un programme écrit en Python qui appelle l'outil HYTECH pour le calcul du $Post$. Le programme Python contient environ 1500 lignes de code, et son développement a nécessité 4 hommes-mois. Ce programme prend en entrée un réseau d'automates temporisés paramétrés décrit sous la forme d'un fichier HYTECH, ainsi qu'une valuation π_0 des paramètres, et génère une contrainte K_0 telle que décrite ci-dessus (section 3.1). Son exécution est entièrement automatique.

---

1. En réalité, C est une contrainte sur les paramètres *et* les variables d'horloges, mais les variables d'horloges sont ici omises dans un souci de simplicité. Voir (André *et al.*, 2009) pour plus de détails.

ALGORITHME ***MéthodeInverse***($\mathcal{A}$, π_0)

Entrée $\mathcal{A}$: Automate temporisé paramétré
π_0 : Valuation des paramètres
Sortie K_0 : Contrainte sur les paramètres
Variables i : Itération courante
S : Ensemble courant d'états symboliques ($S = Post^i_{\mathcal{A}(K_0)}$)
S' : Ensemble précédent d'états symboliques

$i := 0$; $K_0 := \texttt{vrai}$; $S := \{(q_0, \texttt{vrai})\}$; $S' := \{(q_0, \texttt{vrai})\}$
FAIRE
FAIRE JUSQU'À S ne contient plus aucun état π_0-incompatible
Sélectionner un état π_0-incompatible (q, C) de S (c.-à-d. $\pi_0 \not\models C$)
Sélectionner une inégalité π_0-incompatible J de C (c.-à-d. $\pi_0 \not\models J$)
$S' := S$; $K_0 := K_0 \wedge \neg J$; $S := Post^i_{\mathcal{A}(K_0)}(\{(q_0, \texttt{vrai})\})$
FAIT
$S' := S$; $S := Post_{\mathcal{A}(K_0)}(S)$; $i := i + 1$
SI $S = S'$ **ALORS RETOURNE** K_0 **FSI**
FAIT

Figure 7. *L'algorithme* MéthodeInverse

3.3. *Application à l'exemple*

Appliquons IMITATOR au modèle de notre architecture d'automatisation en réseau. Dans ce cas, l'automate temporisé paramétré $\mathcal{A}$ est le produit des automates décrits dans la section 2.2, avec pour paramètres $COMct$, $COMd$, $NETd$, $PLCct$, $COMct$, $RIOd$, $SIGmrt$. La valuation de référence des paramètres π_0 est la suivante :

$$PLCct = 300 \quad COMct = 1000 \quad COMct = 2071 \quad PLCmtt = 100$$
$$RIOd = 70 \quad COMd = 25 \quad NETd = 10$$

L'application directe du programme IMITATOR à l'ensemble du modèle $\mathcal{A}$ de notre AAR cause un échec de HYTECH en raison d'un manque de mémoire. L'une des raisons de cette saturation de la mémoire est que HYTECH, au lieu d'effectuer un parcours à la volée des états symboliques accessibles, calcule d'abord le graphe complet non temporisé des états accessibles, puis effectue ensuite un parcours temporisé dans ce graphe. Le calcul *a priori* de ce graphe non temporisé conduit à l'explosion du graphe d'états, notamment en raison de la forte complexité et cyclicité du modèle de l'AAR.

Plutôt que de créer un nouvel outil ad hoc calculant le graphe des états accessibles à la volée[2], nous avons choisi de découper le modèle $\mathcal{A}$. En effet, dans le modèle de l'AAR, les transitions $\{PLC1 \rightarrow PLC4,\ PLC1 \rightarrow PLC5,\ PLC3 \rightarrow PLC6\}$, $\{COM2 \rightarrow COM7,\ COM3 \rightarrow COM8,\ COM6 \rightarrow COM11\}$, $\{RIO1 \rightarrow RIO6,\ RIO2 \rightarrow RIO4,\ RIO3 \rightarrow RIO5\}$ sont mutuellement exclusives. La restriction du

2. Cet outil est néanmoins en cours de développement.

modèle à l'une des trois transitions pour chacun de ces trois ensembles réduit l'explosion combinatoire. Nous obtenons ainsi 27 sous-modèles différents $\mathcal{A}_k$, où $k \in [1..27]$.

L'analyse est alors effectuée comme suit :

1) calcul de la profondeur du $Post^*_{\mathcal{A}[\pi_0]}$, c'est-à-dire le nombre d'itérations pour obtenir le point fixe pour $\mathcal{A}[\pi_0]$; dans notre cas, il est atteint après 51 itérations ;

2) sélection d'un sous-modèle $\mathcal{A}_l$, où $l \in [1..27]$, tel que $\mathcal{A}_l$ atteint également le point fixe $Post^*_{\mathcal{A}_l[\pi_0]}$ en 51 itérations ;

3) calcul, par application d'IMITATOR, de $K_l := \textit{MéthodeInverse}(\mathcal{A}_l, \pi_0)$;

4) vérification que, sous cette contrainte, l'ensemble des états accessibles dans le modèle paramétrique complet original $\mathcal{A}$, noté $Post^*_{\mathcal{A}(K_l)}$, respecte la condition de bon fonctionnement, définie par l'exclusion de mauvais états Bad.

La contrainte K_l générée par IMITATOR est la suivante[3] :

$$
\begin{array}{rrcl}
 & 2COMct + RIOd & < & SIGmrt \\
\wedge & 2COMct & < & PLCmtt + 6PLCct + COMd + NETd + RIOd \\
\wedge & COMd + 2COMct + NETd & < & 7PLCct \\
\wedge & 2COMct + 2NETd + RIOd & < & 7PLCct \\
\wedge & 3COMct & < & 10PLCct + NETd \\
\wedge & 2NETd + RIOd & < & PLCmtt \\
\wedge & COMct & < & PLCmtt + 3PLCct + NETd \\
\wedge & 10PLCct & < & 3COMct + NETd \\
\wedge & 3PLCct + 2NETd + RIOd & < & COMct \\
\wedge & COMd + NETd & < & PLCmtt \\
\wedge & 3PLCct + COMd + NETd & < & COMct \\
\wedge & 7PLCct & < & COMd + 2COMct + NETd + RIOd \\
\wedge & COMct & < & 3PLCct + COMd + NETd + RIOd
\end{array}
$$

Ce qui donne, après instanciation de tous les paramètres à l'exception de $COMct$, $PLCct$ et $SIGmrt$:

$$
\begin{array}{rrcl@{\quad}rrcl}
 & SIGmrt & > & 2COMct + 70 & \wedge & COMct & > & 3PLCct + 90 \\
\wedge & 10PLCct & < & 3COMct + 10 & \wedge & 7PLCct & > & 2COMct + 90 \\
\wedge & 2COMct & < & 6PLCct + 205 & \wedge & 3COMct & < & 10PLCct + 10 \\
\wedge & 7PLCct & < & 2COMct + 105 & & & & \\
\end{array}
$$

Ainsi, nous avons la garantie que, pour toutes les valeurs des paramètres du système vérifiant cette contrainte K_l, le système fonctionnera de la même manière que selon le point de bon fonctionnement π_0.

L'ensemble de ce processus demande environ 2 heures sur un Intel Xeon 3,20 GHz avec 2 Go de mémoire RAM.

4. Comparaison des résultats obtenus avec d'autres méthodes

La figure 8 présente les résultats obtenus par quatre techniques différentes. Chaque graphique illustre la relation entre les paramètres (seul $PLCct$ est indiqué) et la performance attendue pour le temps de réponse $SIGrt$. La zone au-dessus de la ligne brisée

3. Le programme HYTECH décrivant le modèle $\mathcal{A}_l$ utilisé par IMITATOR est disponible sur la page Web d'IMITATOR : http ://www.lsv.ens-cachan.fr/~andre/IMITATOR/.

représente la zone de bon fonctionnement (la demande du concepteur $SIGrt$ est supérieure au délai obtenu par l'AAR paramétré ici par $PLCct$), et la zone au-dessous représente le fait que le système n'atteint pas les performances attendues.

Les résultats de plusieurs simulations sur le même modèle instancié donnent l'histogramme de $SIGrt$ de la figure 8a. On suppose alors un domaine de bon fonctionnement trop optimiste (la valeur maximale observée en simulation n'est pas forcément la plus grande possible, ici la ligne brisée).

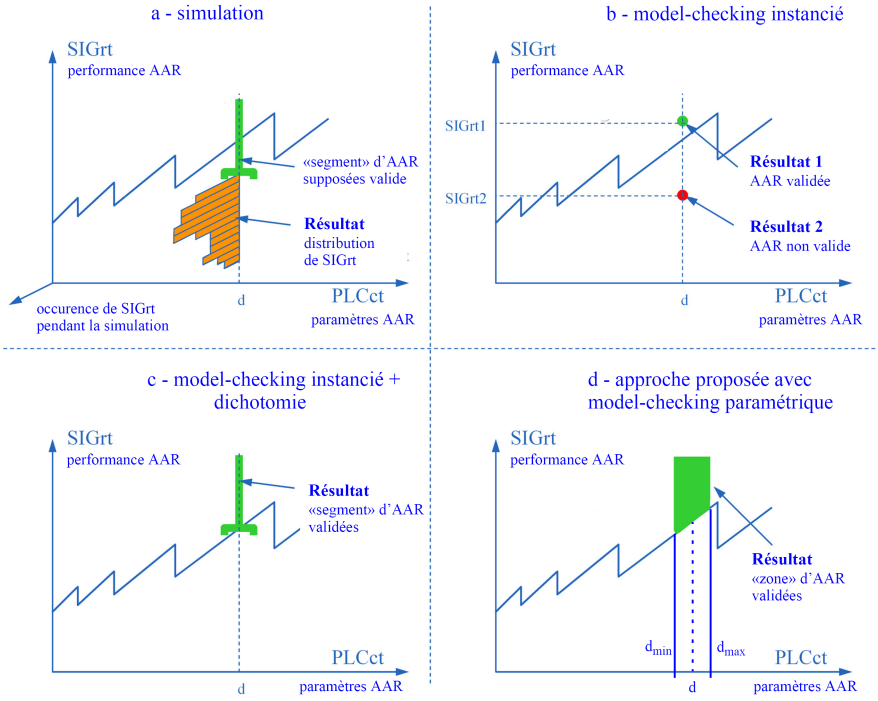

Figure 8. *Comparaison des résultats obtenus par quatre approches différentes*

La figure 8b montre les résultats obtenus par model-checking instancié. Ici, les valeurs instanciées des paramètres ainsi que la valeur instanciée de la performance ($SIGrt$) sont passées au model-checker UPPAAL. La réponse est soit « oui » si le système est validé, soit « non » s'il n'atteint pas la performance souhaitée. Contrairement aux résultats de simulation, les résultats du model-checking instancié sont garantis.

La figure 8c présente la forme (un segment) du résultat obtenu par une recherche dichotomique sur la figure 8b. Il s'agit du résultat le plus utile : il a été utilisé pour construire une vue échantillonnée des limites de la région, et a également été utilisé pour déterminer une valeur de $SIGrt$ et des paramètres proches de cette limite que nous avons pu donner en entrée à notre algorithme de model-checking paramétré.

Enfin, la figure 8d présente le résultat de notre approche. Le principal avantage est d'obtenir des régions *denses* (ici de d_{min} à d_{max}) dans toutes les directions, sans aucun échantillonnage.

	$PLCct$	$COMct$	$SIGmrt$	$PLCmtt$	$RIOd$	$COMd$	$NETd$	Volume
π_1	300	1000	2071	100	70	25	10	171192.7
π_2	600	500	2071	100	70	25	10	1512000.0
π_3	525	436	2071	100	70	25	10	737000.0
π_4	525	435	2071	100	70	25	10	1408000.0

Tableau 1. *Quatre points de fonctionnement*

Dans le tableau 1, nous présentons les régions obtenues à partir de quatre points de bon fonctionnement. Ces quatre points correspondent à quatre polyèdres en 7 dimensions, délimitant chacun un domaine paramétrique pour $PLCct$, $COMct$, $SIGmrt$, $PLCmtt$, $RIOd$, $COMd$ et $NETd$, dont le volume est donné dans le tableau.

Les zones délimitées par nos contraintes correspondent à des parties de la zone validée par simulation. Bien que la première soit nettement moins étendue que la dernière, il faut noter que la zone délimitée par les contraintes est un polyèdre en 7 dimensions ; c'est donc seulement sa projection sur les trois paramètres les plus significatifs qui est comparée à la zone trouvée par simulation.

5. Conclusion

L'analyse de cette architecture d'automatisation en réseau nous a conduits à une nouvelle méthode qui exploite une synergie entre les approches par simulation et par model-checking. En partant d'une valuation des paramètres temporels, déterminée par simulation, qui illustre un comportement acceptable du système, nous inférons par généralisation un ensemble de contraintes sur les paramètres qui garantissent que le système se comporte de manière similaire au point de référence. Notre méthode permet d'obtenir directement une zone dense de points de bon fonctionnement. Elle offre donc une alternative intéressante aux approches habituelles qui recherchent les points de bon fonctionnement par exploration dichotomique d'une zone supposée pertinente.

Cette zone délimitée par contraintes a été comparée à un ensemble de points obtenus par simulation à l'intérieur d'une zone balayée par une recherche dichotomique. Mais, alors que l'approche dichotomique se contente d'analyser le comportement en un grand nombre de points, ce qui est très coûteux et ne donne pas de garantie sur le comportement du systèmes entre les points explorés, notre méthode présente l'avantage de donner directement une zone dense de points où le bon fonctionnement du système est garanti.

Par ailleurs, cette comparaison indique que notre méthode peut être raffinée pour inférer de plus grandes zones de bon fonctionnement. Une piste serait d'itérer la méthode de généralisation en partant à chaque fois d'un point de bon fonctionnement qui

n'est pas dans la zone inférée (André *et al.*, 2009). Cette méthode incrémentale est actuellement en cours d'expérimentation.

Remerciements

Ce travail s'inscrit dans le cadre du projet SIMOP de l'institut Farman à l'ENS de Cachan.

6. Bibliographie

Alur R., Dill D., « A theory of timed automata », *Theor. Comp. Sci.*, vol. 126, n° 2, p. 183-235, 1994.

Alur R., Henzinger T. A., Vardi M. Y., « Parametric real-time reasoning », *STOC*, p. 592-601, 1993.

André E., « IMITATOR : A Tool for Synthesizing Constraints on Timing Bounds of Timed Automata », *ICTAC'09*, LNCS, Springer, August, 2009. To appear.

André É., Chatain Th., Encrenaz E., Fribourg L., « An Inverse Method for Parametric Timed Automata », *International Journal of Foundations of Computer Science*, 2009. To appear.

Chevallier R., Encrenaz E., Fribourg L., Xu W., « Timed Verification of the Generic Architecture of a Memory Circuit Using Parametric Timed Automata », *Formal Methods in System Design*, vol. 34, n° 1, p. 59-81, February, 2009.

Clarisó R., Cortadella J., « Verification of Timed Circuits with Symbolic Delays », *Proceedings of ASP–DAC'04*, p. 47-59, 2004.

Clarke E. M., Grumberg O., Peled D. A., *Model Checking*, MIT Press, 1999.

Denis B., Ruel S., Faure J.-M., Marsal G., Frey G., « Measuring the impact of vertical integration on response times in Ethernet fieldbuses », *Proc. of ETFA'07*, 2007.

Diekert V., Rozenberg G., *The book of traces*, World Scientific, 1995.

Henzinger T., Ho P., H. W.-T., « HYTECH : A Model Checker for Hybrid Systems », *Journal of Software Tools for Technology Transfer*, vol. 1(1-2), p. 110-122, 1997.

Jasperneite J., Neumann P., Theis M., Watson K., « Deterministic real-time communication with switched Ethernet », *Proceedings of the 4th IEEE International Workshop on Factory Communication Systems*, p. 11-18, 2002.

Larsen K. G., Pettersson P., Yi W., « UPPAAL in a nutshell », *Journal of Software Tools for Technology Transfer*, vol. 1(1-2), p. 134-152, 1997.

Marsal G., Evaluation of time performances of Ethernet-based automation systems by simulation of high-level Petri nets, PhD thesis, ENS Cachan (France) and Kaiserslautern University (Germany), December, 2006.

Marsal G., Witsch D., Denis B., Faure J., Frey G., « Evaluation of real-time capabilities of Ethernet-based automation systems using formal verification and simulation », *Proceedings of RJCITR'05*, p. 27-30, 2005.

Ruel S., De Smet O. Faure J.-M., « Efficient representation for formal verification of time performances of networked automation architectures », *Proc. of 17th IFAC World Congress*, p. 5119-5124, July, 2008.

Thomesse J.-P., « Fieldbuses and interoperability », *Control Engineering Practice*, vol. 7, n° 1, p. 81-94, 1999.

Vitturi S., « DP-Ethernet : the Profibus DP protocol implemented on Ethernet », *Computer Communications*, vol. 26, n° 10, p. 1095-1104, 2003.