Propriétés de latence, fraîcheur et réactivité dans un programme synchrone multi-périodique

Rémy Wyss, Frédéric Boniol, Julien Forget, Claire Pagetti

Propriétés de latence, fraîcheur et réactivité dans un programme synchrone multi-périodique

Rémy Wyss[1], Frédéric Boniol[1], Claire Pagetti[1], Julien Forget[2]

[1] ONERA-Toulouse, France
{frederic.boniol,claire.pagetti,remy.wyss}@onera.fr
[2] LIFL/Polytech'Lille, France
julien.forget@lifl.fr

Abstract. Nous avons proposé dans [WBPF12] un cadre formel pour l'analyse de propriétés de latence dans un programme synchrone multi-périodique. Ce cadre ne permettait que le calcul d'une sur-approximation de la latence. Le présent article reprend ce cadre pour proposer un calcul exact, et l'étend à deux autres propriétés : la fraîcheur et la réactivité.

1 Introduction

Contexte. Dans le domaine des systèmes embarqués il est important de garantir que les logiciels sont exempts d'erreurs. Pour ce faire, les concepteurs utilisent des langages formels de haut niveau. Parmi ceux-ci, les langages synchrones et en particulier ceux dits flots de données (tels que Lustre [HCRP91]), fournissent un bon niveau d'abstraction et s'appuient sur des générateurs de code éprouvés. A partir de ces spécifications formelles, plusieurs outils ou méthodes permettent (ou tentent) de vérifier ou prouver des exigences fonctionnelles ou logiques, au niveau modèle ou au niveau code.

Cet article s'intéresse à la vérification d'une autre classe de propriétés : les propriétés temporelles, plus particulièrement les propriétés de *latence*, *fraîcheur* et *réactivité*. La *latence* est la durée nécessaire pour qu'une entrée (e.g. un ordre de pilotage) se propage jusqu'à la sortie du système (e.g. un ordre d'inclinaison de gouverne). La *fraîcheur* à l'inverse caractérise l'âge d'une sortie tant qu'elle est utilisée par l'environnement (e.g. la servo-commande qui positionne la gouverne) relativement à l'entrée qui l'a produite. La *fraîcheur* diffère de la *latence* en ce sens qu'elle prend en compte non seulement le temps de propagation de l'entrée vers la sortie, mais également les périodes de rafraichissement de l'entrée, de la sortie et de l'ensemble des flots intermédiaires. Dans les systèmes de contrôle commande, la fraîcheur des sorties est centrale. Par exemple, dans un système de commande de vol, il ne sert à rien de calculer un ordre d'inclinaison de gouverne en $10ms$ si on ne le *rafraîchit*, lui ou l'ordre de pilotage ou les flots intermédiaires, que toutes les minutes. Enfin, la *réactivité* caractérise la granularité temporelle minimale à laquelle le système est sensible. Autrement dit la durée pendant laquelle une entrée doit être présente (e.g. un appui bouton), pour être sûr qu'elle est propagée jusqu'à la sortie du système. Bien évidemment la réactivité

dépend de la période d'échantillonnage de l'entrée considérée (toute modification de cette entrée plus courte que sa période d'échantillonnage pourra ne pas être vue), mais aussi des périodes des différentes étapes de calcul dans le système.

Objectif de l'article. Ces trois propriétés sont généralement étudiées et vérifiées au niveau de l'implantation, c'est-à-dire assez tard dans le cycle de développement. Les techniques utilisées à ce niveau vont du test et de la mesure sur le système réel, à l'analyse de WCET (temps d'exécution pire cas) et de WCRT (temps de réponse pire cas). L'objectif recherché dans cet article, dans la lignée des travaux présentés dans [WBPF12] est de "remonter" ces vérifications au niveau "modèle", c'est-à-dire au niveau des formalismes de haut niveau. Pour ce faire, nous considérons le langage d'architecture synchrone Prelude introduit récemment dans [FBLP08]. Prelude est une extension temps réel formelle de Lustre permettant la modélisation de systèmes multi-périodiques composés de nœuds importés (vu comme des boîtes noires) communiquant par échange de flots de données et s'exécutant à des périodes potentiellement différentes. L'objectif recherché par Prelude est d'offrir un style synchrone enrichi par des horloges temps réel permettant une génération de code multi-tâche pour un exécutif temps réel. La préservation de la sémantique, y compris temporelle, du niveau modèle jusqu'au niveau implantation permet de vérifier des propriétés temporelles au niveau modèle en faisant abstraction de l'implantation finale et de garantir leur préservation à ce niveau implantation. Restant uniquement à vérifier au niveau implantation l'ordonnançabilité du système global vis-à-vis de la sémantique synchrone.

Travaux connexes. Comme indiqué ci-dessus, les latences et les propriétés de performance sont généralement étudiées au niveau de l'implantation. A ce niveau, trois types de latence ont été étudiés : le WCET d'une tâche sur un processeur donné [WEEea08], le WCRT d'une transaction (i.e., un ensemble de tâches formant une chaîne) pour un ordonnancement donné [LPT09,RGR09], et enfin les temps de traversée pire cas (WCTT) de réseaux temps réel (réseaux commutés, bus CAN, etc.) [LBT01,Mar04]. Ces analyses cependant ne sont possibles qu'au niveau implantation, et interviennent donc tard dans le cycle de développement.

[LBEP11] a proposé une approche similaire au niveau modèle, mais pour un système implanté sur une architecture IMA (Integrated Modular Avionic). Les auteurs définissent une notion de latence et de fraîcheur dans un système IMA distribué asynchrone et propose une méthode d'évaluation de ces grandeurs reposant sur la programmation linéaire. Si les notions de latence et de fraîcheur sont proches de celles que nous proposons d'investiguer, la sémantique des modèles (asynchrone pour [LBEP11] et synchrone pour Prelude) conduit à des techniques d'analyse très différentes.

Enfin, à notre connaissance, il n'existe pas de travaux permettant de caractériser et vérifier formellement la propriété de réactivité.

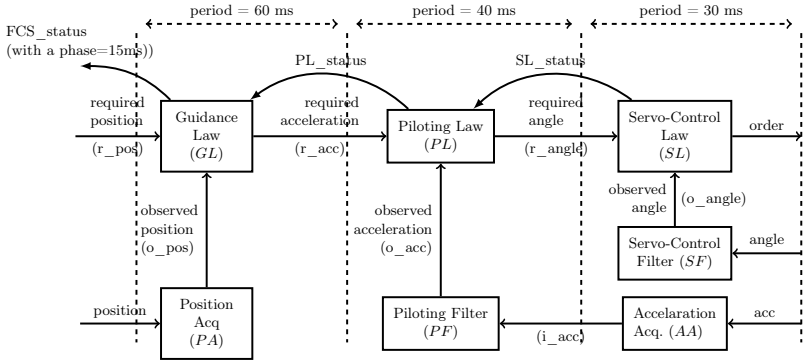


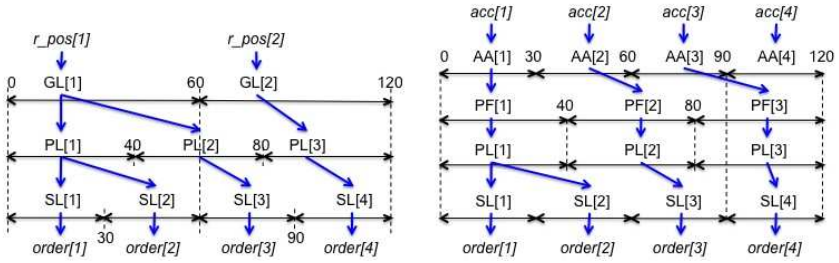
Fig. 1. Une version simplifiée d'un système de contrôle de vol

2 Un exemple avionique

Afin d'illustrer notre propos, considérons un exemple simplifié de logiciel de commande de vol d'un avion (figure 1). Ce logiciel calcule l'ordre d'inclinaison d'une gouverne (*order*) en fonction d'un ensemble d'entrées : l'angle d'inclinaison réel de la gouverne (*angle*), l'accélération verticale de l'avion (*acc*), la position de l'avion (*position*), et la position souhaitée (*required position*) entrée par le pilote ou par le pilote automatique. Ce logiciel est composé de 7 nœuds (vus comme des boîtes noires) et formant trois boucles de commande : une boucle rapide à $30ms$ $SF \rightarrow SL$ qui asservit la gouverne sur un angle requis, une boucle multi-périodique AA (à $30ms$) $\rightarrow PF \rightarrow PL$ (à $40ms$) $\rightarrow SL$ (à $30ms$) qui asservit l'avion sur une valeur d'accélération verticale requise, et enfin une boucle de guidage également multi-péridique $PA \rightarrow GL$ (à $60ms$) $\rightarrow PL$ (à $40ms$) $\rightarrow SL$ (à $30ms$). En plus de ces trois boucles de commande (boucles au sens où le système est bouclé avec l'environnement par l'acquisition des valeurs réelles d'angle de gouverne, d'accélération et de position), le système contient une chaîne de retro-propagation du statut (correct ou erroné) de la commande effectuée : SF (à $30ms$) $\rightarrow SL \rightarrow PL$ (à $40ms$) $\rightarrow GL$ (à $60ms$). Le statut final du système synthétisé par GL est retourné vers le pilote. Notons que ce statut est retourné à la période de $60ms$ avec un décalage de phase de $15ms$. Ce système sera décrit formellement en Prelude dans la section 3.

Ce logiciel est un élément d'un système globalement critique qui doit satisfaire des exigences temporelles.

Exemple d'exigence de latence. Considérons la chaîne $GL \rightarrow PL \rightarrow SL$. Entre le moment où le pilote entre (via un dispositif approprié) une position requise (*required position*) et le moment où la gouverne est dans la position correspondant à cette requête, il doit s'écouler au plus une seconde. Autrement dit l'avion doit réagir en au plus une seconde à un ordre pilote. En considérant que la gouverne a sa propre latence (le temps qu'elle met à bouger vers l'angle requis) égal à 0.8 seconde, la latence induite par le logiciel de *required position* à *order* doit donc



(a) chaîne r_pos à $order$ (b) chaîne acc à $order$

Fig. 2. Comportement temporel du système

être inférieure à $200ms$. En considérant que le système est ordonnançable, c'est-à-dire que chaque nœud est exécuté en respectant sa période et en respectant les précédences dues aux flots de données, la latence de la chaîne $GL \rightarrow PL \rightarrow SL$ dépend des périodes de chaque nœud de la chaîne. Le comportement temporel de cette chaîne sur une hyper-période est illustré figure 2(a). La première requête $r\_pos[1]$ impacte en bout de chaîne $order[1]$, puis $order[2]$ et $order[3]$. La seconde requête $r\_pos[2]$ impacte $order[4]$. En conséquence, en supposant le pire cas où r_pos arrive au début de sa période, et $order$ est produit à la fin de sa période, alors la pire latence est $60ms$, c'est-à-dire celle de la seconde requête (entre l'arrivée de $r\_rpos[2]$ et la sortie de $order[4]$).

Exemple d'exigence de fraîcheur. Considérons la chaîne $AA \rightarrow PF \rightarrow PL \rightarrow SL$. Cette chaîne asservit l'avion sur une accélération requise. Pour des raisons de stabilité de l'asservissement (non détaillées ici), il est exigé que tout ordre appliqué sur la gouverne corresponde à une accélération réelle mesurée au maximum $100ms$ plus tôt. Le comportement de la chaîne considérée est représenté figure 2(b). On voit que la latence de la chaîne est au maximum de $60ms$. En revanche, en supposant un pire scénario où $acc[1]$ est mesuré au début de sa période (à $t = 0$) et $order[3]$ n'est produit qu'à la fin de sa période (à $t' = 90$), alors on voit que $order[2]$ est utilisé par la gouverne jusqu'à t' (non inclus). La fraîcheur dans ce cas de l'ordre appliqué est $t' - t = 90ms$. Ce qui montre que la latence et la fraîcheur sont deux propriétés différentes.

Exemple d'exigence de réactivité. Considérons à nouveau la chaîne $AA \rightarrow PF \rightarrow PL \rightarrow SL$. Cette chaîne doit également satisfaire une exigence de réactivité. Supposons que l'avion subisse une rafale de vent qui augmente brutalement et momentanément l'accélération verticale. Pour le confort des passagers mais aussi pour éviter de fatiguer la structure des ailes, le système doit réagir à toute rafale, et donc toute augmentation brutale de l'accélération, plus longue que $120ms$ en braquant les ailerons de façon appropriée. Il est donc nécessaire de montrer que toute modification de acc qui dure plus de $120ms$ sera prise en compte par la chaîne AA jusqu'à SL. Considérons encore la figure 2(b). On voit que $acc[4]$ n'est pas utilisé. Si on considère que acc est mesurée au début de chaque période, toute

date	1	2	3	4	5	6	7	...	horloge
x	x^1		x^2		x^3		x^4	...	(2,0)
$\tau(x)$	$\tau(x^1)$		$\tau(x^2)$		$\tau(x^3)$		$\tau(x^4)$	...	(2,0)
0 fby x	0		x^1		x^2		x^3	...	(2,0)
x *^ 2	x^1	x^1	x^2	x^2	x^3	x^3	x^4	...	(1,0)
x /^ 4	x^1				x^3				(4,0)
x ~> 1/2		x^1		x^2		x^3			(2,1/2)

Fig. 3. Opérateurs du langage PRELUDE

modification de l'accélération réelle entre $t = 60$ et $t' = 120$ ne sera pas perçue par le système. On dit que le système n'est pas réactif à des variations de *acc* de durée inférieure à $60ms$.

Notons que les raisonnements informels ci-dessus ne prennent pas en compte les choix d'implantation. En d'autres termes, ils sont valables quelle que soit l'implantation finale (pourvu qu'elle respecte la sémantique du langage). L'objectif de l'article est de montrer qu'il est possible de mécaniser le calcul de ces propriétés sur un programme synchrone augmenté par des horloges temps réel.

Notons enfin que l'exemple présenté figure 1 est un système simplifié. Un logiciel de commande de vol réel contient environ 5000 nœuds. Calculer « à la main », comme on vient de le faire, des latences, fraîcheurs, et réactivités sur de tels systèmes devient impossible. Il est nécessaire de mécaniser ces calculs.

3 Processus synchrones multi-périodiques

3.1 PRELUDE : présentation succincte

Nous supposons dans la suite que le cadre considéré pour décrire un logiciel de contrôle commande est le langage PRELUDE [FBLP08]. PRELUDE reprend les hypothèses et la sémantique des langages flots de données synchrones en les étendant avec la notion d'horloge temps réel strictement périodique. Un système PRELUDE est un assemblage de nœuds importés (i.e. dont le contenu est défini dans un autre langage) communicant par consommations et productions synchrones de flots de données. Chaque flot d'entrée de l'assemblage (reçu de l'environnement) est typé par une horloge temps réel *strictement périodique*. C'est ce typage qui introduit la dimension temps réel dans le système : conformément à la sémantique flots de données, chaque nœud de l'assemblage est activé sur l'horloge temps réel de ses flots d'entrée, et produit ses flots de sortie de façon synchrone sur cette même horloge.

Une horloge est strictement périodique si l'intervalle entre deux tops successifs est constant. Ainsi, si $h[i]$ représente la date du i^{ieme} top de l'horloge h, alors pour tout i, $h[i+1] - h[i]$ est la période de l'horloge, et $h[0]$ est la date du premier top (la phase de l'horloge). Une horloge strictement périodique h est donc définissable par un couple (p, q), où $p \in \mathbb{N}^*$ est la valeur de sa période ($h[i+1] - h[i] = p$), et $q \in \mathbb{Q}^+$ est la valeur de sa phase en fraction de la période ($h[0] = pq$). On supposera dans la suite que toutes les horloges sont entières.

L'horloge la plus rapide est $(1, 0)$ car elle est présente à tout instant à partir du premier instant. De même $(3, 2/3)$ est l'horloge activée toutes les 3 unités de temps et qui commence à 2.

Conformément à la sémantique synchrone, un nœud ne peut consommer que des flots de même horloge. Pour la description de systèmes multi-périodiques, il est alors nécessaire d'utiliser des opérateurs de sur ou sous échantillonnage :

1. $x *\hat{} k$ est un opérateur d'accélération de rythme, qui sur-échantillonne x sur une horloge k fois plus rapide ;
2. $x/\hat{} k$ est un opérateur de décélération de rythme, qui sous-échantillonne x sur une horloge k fois plus lente ;
3. $x \sim> q$ est un opérateur de déphasage, qui décale x de q fois sa période ;

où $k \in \mathbb{N}^*$ et $q \in \mathbb{Q}$. Le comportement de ces trois opérations est illustré par le tableau figure 3 (où `fby` est l'opérateur de retard avec initialisation de SCADE, τ un nœud importé, et x^n dénote la $n^{ième}$ occurrence du flot x).

D'un point de vue concret, l'exécution d'un système PRELUDE suit l'hypothèse « synchrone relâchée » [Cur05] : les expressions ne sont pas nécessairement évaluées strictement à chaque top de leur horloge, mais dans l'intervalle défini par leur période avant leur échéance réelle, c'est-à-dire avant leur utilisation par d'autres nœuds du système ; ce qui permet une implantation multi-tâche (un nœud importé par tâche) gérée par un ordonnancer temps réel. Par exemple la chaine $SF \rightarrow SL$ consomme et produit des flots de période $30ms$. Conformément à l'hypothèse « synchrone relâchée », elle devra être ordonnancée et exécutée globalement dans cette période (à la fois SF puis SL).

C'est l'explicitation des périodes et des phases des horloges (contrairement à LUSTRE qui n'autorise que des horloges booléennes) et des opérations de changement de rythme qui permet l'analyse de propriétés temporelles sur un assemblage PRELUDE quels que soient les choix d'implantation respectant l'hypothèse « synchrone relâchée ».

Exemple 1 *Considérons le système figure 1. Ce système est exprimé par le programme figure 4 et par la représentation graphique (équivalente) figure 5. Le système est décrit sous la forme d'un nœud (appelé ici* `FCS`*), admettant des flots d'entrée (ici, il y en a 4, les deux premiers étant sur l'horloge (30,0), et les deux derniers sur l'horloge (60,0)), et des flots de sortie (ici* `ordre` *et* `FCS_status`*). Le système est ensuite défini par un ensemble d'équations. Par exemple* `i_acc = AA(acc);` *définit le flot* `i_acc` *comme le résultat du nœud* `AA` *sur le flot* `acc` *et sur la même horloge que ce flot (i.e.,* `i_acc` *est sur (30,0)). De même les équations* `x1 = i_acc *^3;` *et* `x2 = x1 /^4;` *définissent* `x1` *comme égal à* `i_acc` *sur-échantillonné sur une horloge trois fois plus fréquente, donc sur l'horloge (10,0), et* `x2` *comme égal à* `x1` *sous-échantillonné sur une horloge quatre fois moins fréquente, donc sur (40,0). Le nœud* `PF` *est alors activé sur cette même horloge à chaque réception de* `x2` *et produit en résultat* `o_acc` *sur (40,0). Remarquons que le passage par l'horloge (10,0) n'est que virtuel puisque cela n'entraine pas de synchronisation sur cette horloge. Les nœuds restent synchrones avec l'horloge de leurs entrées uniquement. L'implantation temps réel ne*

```
imported node SF(i:int) returns (o: int);
imported node SL(i1,i2:int) returns (o1:int; o2:bool);
...
node FCS (angle:rate(30,0); acc:rate(30,0); position:rate(60,0); r_pos:rate(60,0))
returns (ordre, FCS_status);
var x1, x2, x4, x5, x6, x7, x8, x9, x10, x11, x12,
    i_acc, o_pos, o_acc, r_acc, r_angle, o_angle, SL_status, PS_status, GL_status;
let
  o_pos = PA(position);
  (r_acc, GL_satus) = GL(r_pos, o_pos, x12);
  x3 = r_acc *^ 3;     x4 = x3 /^ 2;
  i_acc = AA(acc);
  x1 =i_acc *^ 3;      x2 = x1 /^ 4;       o_acc = PF(x2);
  (r_angle, PL_status) = PL(x4, o_acc, SL_status);
  x5 = r_angle *^4;    x6 = x5 /^ 3;    o_angle = SF(angle);
  (order, SL_status) = SL(x6, o_angle);
  x7 = true fby SL_satus ;   x8 = x7 *^3;    x9 = x8 /^ 4;
  (order, SL_status) = SL(x6, o_angle);
  x10 = true fby PL_satus ;  x11 = x10 *^2;  x12 = x11 /^ 3;
  FCS_status = GL_status ~> 1/4;
tel
```

Fig. 4. Description textuelle PRELUDE du système de contrôle de vol

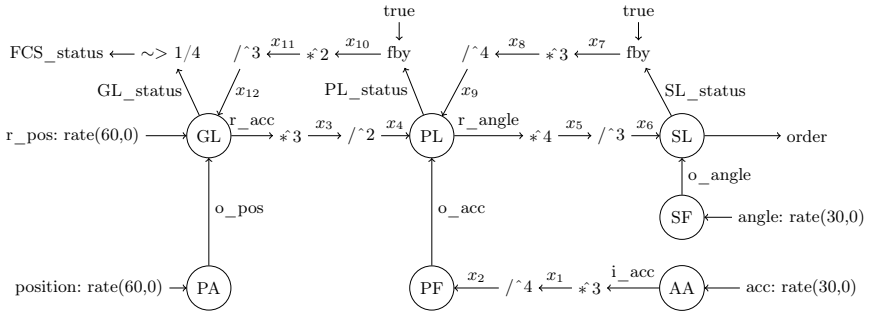


Fig. 5. Description graphique PRELUDE du système de contrôle de vol

contiendra pas d'horloge à $10ms$*. Notons enfin que comme en* LUSTRE*, l'ordre syntaxique des équations n'est pas significatif. Les traitements sont exécutés dans l'ordre induit par les dépendances de données.*

A partir d'une telle description, on peut inférer les horloges des flots du système. Ce calcul d'horloge se fait simplement par propagation des horloges à travers les équations. Pour plus de détails sur PRELUDE, sa sémantique et son calcul d'horloges, le lecteur est invité à se référer à [For09] et aux outils associés[3].

Formalisation succincte. Dans la suite de l'article, nous considérons qu'un système est modélisée par $S =< F^i, F, F^o, Eqs, \pi, \phi >$ où F^i est l'ensemble de flots d'entrée (i.e., les flots définis par aucune équation), F les flots internes, et F^o les flots de sortie, $\pi : F^i \cup F \cup F^o \rightarrow \mathbb{N}^*$ est la fonction qui associe à chaque flot sa période, $\phi : F^i \cup F \cup F^o \rightarrow \mathbb{Q}$ associe à chaque flot sa phase (rappelons

[3] http://www.lifl.fr/forget/prelude.html, et http://sites.onera.fr/schedmcore

que π et ϕ sont calculées par le compilateur du langage), et Eqs est un ensemble d'équations définies par la syntaxe suivante:

$$\begin{aligned} eqs &::= eq; eqs \\ eq &::= (y_1, \ldots, y_n) = N(x_1, \ldots, x_p) \mid y = cte \texttt{ fby } x \mid \\ &\quad y = x *\hat{} k \mid y = x/\hat{} k \mid y = x \sim> q \end{aligned}$$

où $y_i \in F \cup F^o$, $x_i \in F^i \cup F \cup F^o$, N un nœud importé, cte une constante, $k \in \mathbb{N}^*$, et $q \in \mathbb{Q}$. Les équations définissent la structure interne du système. Dans la suite, on ne considère que des systèmes bien formés, c'est-à-dire qui ne contiennent aucun cycle instantané (tous les cycles contiennent au moins un `fby` ou un décalage de phase), et tous les appels de nœuds importés $N(y_1, \ldots, y_p)$ portent sur des flots de même horloge.

Soit x un flot d'horloge, dans la suite on note x^p la p^{ieme} occurrence de x.

3.2 Mots de dépendance de données

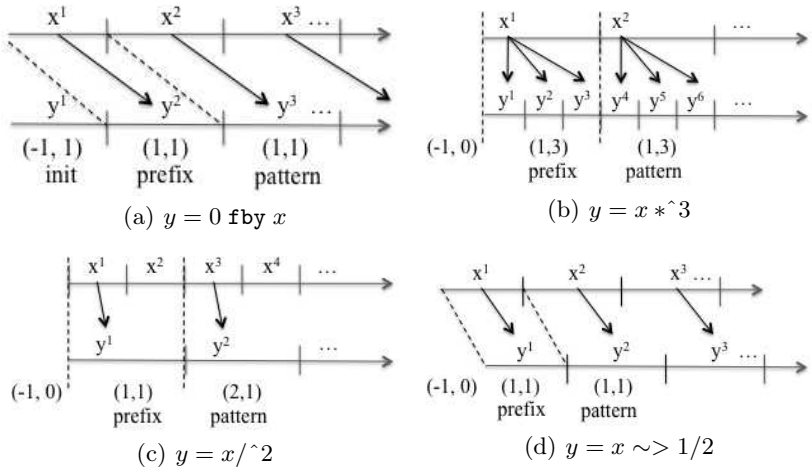
PRELUDE est défini par une sémantique formelle de type sémantique de Kahn. Cette sémantique permet de construire une relation de précédence entre les flots d'un système [For09], qui permet ensuite de transformer un programme PRELUDE en un code multi-tâche temps réel synchronisé par un ensemble de mécanismes (sémaphores ou priorités) implantant ces précédences (et donc la sémantique flots de données synchrone). Notons $y^p \leftarrow x^n$ cette relation de précédence exprimant que x^n est nécessaire pour calculer y^p (par exemple, si y est défini par l'équation `y=N(x)` alors $y^p \leftarrow x^p$ pour tout p; de même si `y=0 fby x` alors $y^{p+1} \leftarrow x^p$ pour tout p, etc.).

Cette relation a été exprimée et étendue dans [PFB+11] et [WBFP13] sous une forme compacte et périodique appelée "mot de dépendance de données". Définie initialement pour servir la génération de code, nous allons utiliser cette formulation pour calculer les propriétés temporelles d'un programme PRELUDE.

Définition 1 (Mot de dépendance de données) *Un mot de dépendance de données entre un flot i et un flot o est une suite finie de couples de la forme $w = (-1, d_0)(k_1, d_1) \ldots (k_n, d_n)$ où $d_0 \in \mathbb{N}$, et $k, d \in \mathbb{N}^\star$, et exprimant les dépendance suivantes: $\forall p \in \mathbb{N}^\star$:*

$$o^p \leftarrow \begin{cases} init & si\ p \in [1, d_0] \\ i^{k_1} & si\ p \in [d_0+1, d_0+d_1] \\ i^{k_1+k_2} & si\ p \in [d_0+d_1+1, d_0+d_1+d_2] \\ \ldots & \\ i^{q\cdot(\Sigma_{j\in[2,n]}\ k_j)+\Sigma_{j\le l}\ k_j} & si\ q \in \mathbb{N}, l \in [1,n], p \in [a_p, b_p]\ avec \\ \quad a_p = q\cdot(\Sigma_{j\in[2,n]}\ d_j) + \Sigma_{j\le l-1}\ d_j + 1 & \\ \quad b_p = q\cdot(\Sigma_{j\in[2,n]}\ d_j) + \Sigma_{j\le l}\ d_j & \end{cases}$$

Les deux premiers couples $(-1, d_0)(k_1, d_1)$ correspondent au préfixe du mot : les d_0 premières occurrences de o sont égales à la valeur $init$ (introduite par un `fby`), tandis que les d_1 occurrences suivantes de o (numérotées de d_0+1 à d_0+d_1)



(a) $y = 0$ `fby` x (b) $y = x *\hat{}\, 3$

(c) $y = x/\hat{}\, 2$ (d) $y = x \sim> 1/2$

Fig. 6. Exemple de mot de dépendance de données pour les opérateurs Prelude

dépendent de l'occurrence k_1 de i. Ensuite, on entre dans la partie périodique du mot. Les d_2 occurrences suivantes de o (numérotées de d_0+d_1+1 à $d_0+d_1+d_2$) dépendent de $i^{k_1+k_2}$, les d_3 occurrences suivantes de o dépendent de $i^{k_1+k_2+k_3}$. Ainsi de suite jusqu'à (d_n, k_n). On recommence ensuite à (k_2, d_2).

Exemple 2 $(-1,0)(1,1)(1,1)$ *est le mot de dépendance de données le plus simple. Il relie un flot* i *et un flot* o *tels que* $o^p \leftarrow i^p$ *pour tout* $p \in \mathbb{N}^\star$. *D'un point de vue fonctionnel ce mot caractérise une équation (ou un enchainement d'équations) du type* $o = \tau(i)$ *où* τ *est un nœud importé. Plus globalement, la figure 6 montre les mots de dépendance de données caractérisant les principales constructions du langage considérées dans cet article.*

L'algorithme construisant le mot de dépendance de données reliant deux flots est présenté dans [PFB+11].

3.3 Chaîne fonctionnelle

Une propriété temporelle est relative à une chaîne fonctionnelle reliant un flot d'entrée à un flot de sortie.

Définition 2 (Chaîne fonctionnelle) *Soit* $S =< F^i, F, F^o, Eqs, \pi, \phi >$. *Une chaîne de* S *est une liste ordonnée* $L = (x_1, \ldots, x_n)$ *de flots telle que chaque* x_i $(i > 1)$ *est défini par une équation de* Eqs *en fonction de* x_{i-1}.

Exemple 3 *Reprenons le programme* Prelude *figure 5. Il contient plusieurs chaînes fonctionnelles, dont:*

- *la boucle d'asservissement de la gouverne* $L_1 = (angle, o\_angle, order)$,

- *la boucle d'asservissement en accélération de l'avion* $L_2 = (acc,\ i\_acc,\ x_1,\ x_2,\ o\_acc,\ r\_angle,\ x_5,\ x_6,\ order)$,
- *la chaîne de guidage* $L_3 = (r\_pos,\ r\_acc,\ x_3,\ x_4,\ r\_angle,\ x_5,\ x_6,\ order)$,
- *la chaîne de propagation du statut* $L_4 = (angle,\ o\_angle,\ SL\_status,\ x_7,\ x_8,\ x_9,\ PL\_status,\ x_{10},\ x_{11},\ x_{12},\ GL\_status,\ FCS\_status)$

L_1 est mono-périodique et est caractérisée par le mot de dépendance de données le plus simple $w_{L_1} = (-1,0)(1,1)(1,1)$. *Les trois autres chaînes sont multi-périodiques, et sont caractérisées par des mots de dépendance de données plus complexes. Par exemple, le mot de dépendance de données reliant la sortie de L_2 à son entrée est (calculé de façon incrémentale) :*

	mots de dépendance de données
$acc \leftarrow i\_acc$	$(-1,0)(1,1)(1,1)$
$acc \leftarrow^* x_1$	$(-1,0)(1,3)(1,3)$
$acc \leftarrow^* x_2$	$(-1,0)(1,1)(1,1)(1,1)(2,1)$
$acc \leftarrow^* o\_acc$	$(-1,0)(1,1)(1,1)(1,1)(2,1)$
$acc \leftarrow^* r\_angle$	$(-1,0)(1,1)(1,1)(1,1)(2,1)$
$acc \leftarrow^* x_5$	$(-1,0)(1,4)(1,4)(1,4)(2,4)$
$acc \leftarrow^* x_6$	$(-1,0)(1,2)(1,1)(1,1)(2,2)$
$acc \leftarrow^* order$	$w_{L_2} = (-1,0)(1,2)(1,1)(1,1)(2,2)$

En d'autres termes, comme montré sur la figure 7, $order[1]$ *et* $order[2]$ *dépendent de* $acc[1]$ *(couple (1,2) du mot) ;* $order[3]$ *dépend de* $acc[2]$ *(premier couple (1,1)) ;* $order[4]$ *dépend de* $acc[3]$ *(deuxième couple (1,1)) ;* $order[5]$ *dépend de* $acc[5]$ *(couple (2,2)), puis on recommence au premier couple (1,1). De la même manière, les chaînes L_3 et L_4 sont caractérisées par* $w_{L_3} = (-1,0)(1,3)(1,1)(1,3)$ *et* $w_{L_4} = (-1,2)(2,1)(2,1)(2,1)$.

Dans les sections suivantes, nous montrons comment définir et calculer des propriétés de latence, de fraîcheur et de réactivité à partir du mot de dépendance de données caractérisant une chaîne fonctionnelle.

4 Latences pire et meilleur cas

La latence d'une chaîne est le temps de propagation de l'entrée jusqu'à sa sortie. Considérons la chaîne $L_1 = (angle, o\_angle, order)$, mono-périodique d'horloge $(30, 0)$. Selon la sémantique synchrone de Prelude, une occurrence de *angle* acquise à la date logique $t = 30k$ est propagée jusqu'à *order* à cette même date t. Donc $\forall p \in \mathbb{N}^*$, $order^p \leftarrow angle^p$. D'un point de vue synchrone, la latence est nulle. En prenant en compte l'implantation, *order* sera produit pendant sa période, c'est-à-dire entre 0 et $30ms$ après t. La latence de L_1 est donc dans $[0, 30[$. Considérons maintenant la chaîne figure 6(c) formée par l'équation `y=x /^2` avec x d'horloge (10,0). En suivant le même raisonnement la latence de x^1 est égale à 0 plus la période de y, soit $20ms$. Alors que la latence de x^2 est d'une période de x ($10ms$) plus la latence de x^3, c'est-à-dire au total $30ms$. Dit autrement, si

une valeur arrive sur x lors du deuxième top de x, et si elle est maintenue, il lui faudra attendre le $3^{ième}$ top de x puis encore une période de y avant de « sortir » de la chaîne, soit $30ms$ en tout. Notons que si la valeur arrivant en x^2 n'est pas maintenue, elle sera perdue. Ce point est capturé par la notion de réactivité (section 6). Considérons enfin la chaîne 6(b) formée par l'équation `y=x *^3` avec x d'horloge (30,0). Alors, bien que y^2 et y^3 dépendent aussi de x^1, la latence de la chaîne est celle de la dépendance $y^1 \leftarrow x^1$, soit 0 plus la période de y ($10ms$). Généralisons ces trois cas par la définition suivante.

Définition 3 (Latences) *Soit une chaîne $L = (i, \ldots, o)$. Soit l'ensemble des triplets d'indices (p, q, q') $q > q' \geqslant 1$ tels que $o^p \leftarrow i^q$ et $o^{p-1} \leftarrow i^{q'}$ (o^p est la première occurrence de o dépendant de i^q et l'occurrence précédente de o dépend d'une précédente occurrence de i). La latence pire cas notée $WCL(L)$, et la latence meilleur cas notée $BCL(L)$ de L sont définies par :*

$$WCL(L) \stackrel{def}{=} \max_{\{(p,q,q')|q>q'\geqslant 1,\ o^p \leftarrow i^q \text{ et } o^{p-1} \leftarrow i^{q'}\}} (date(o^p) - date(i^{q'+1})) + \pi(o)$$

$$BCL(L) \stackrel{def}{=} \min_{\{(p,q)|o^p \leftarrow i^q\}} (date(o^p) - date(i^q))$$

où $date(x^k) = \phi(x) + (k-1)\pi(x)$ est la date logique de x^k.

Le terme $\pi(o)$ dans $WCL(L)$ permet de prendre en compte le pire temps possible induit par l'ordonnancement temps réel. Du point de vue de l'implantation, o peut en effet être produit n'importe quand pendant sa période, donc au plus $\pi(o)$ après la date logique de son occurrence.

Il est possible de calculer ces pire et meilleure latences à partir du mot de dépendance de données caractérisant la chaîne fonctionnelle.

Proposition 1 (Latence pire cas). *Soit une chaîne $L = (i, \ldots, o)$ de mot de dépendance de données $w = (-1, d_0)(k_1, d_1) \ldots (k_m, d_m)$ (caractérisant les dépendances des occurrences de o vis-à-vis des occurrences de i). Alors*

$$WCL(L) = \phi(o) - \phi(i) + \pi(o) + \max_{j=0\ldots m-1} l_j$$

avec

$$for\ j = 0 \ldots m-1,\ l_j = \pi(o) \sum_{l=0}^{j} d_l - \pi(i) \sum_{l=0}^{j} k_l$$

en posant $k_0 = 0$.

Proposition 2 (Latence meilleur cas). *Soit une chaîne $L = (i, \ldots, o)$ de mot de dépendance de données $w = (-1, d_0)(k_1, d_1) \ldots (k_m, d_m)$ (caractérisant les dépendances des occurrences de o vis-à-vis des occurrences de i). Alors*

$$BCL(L) = \phi(o) - \phi(i) + \pi(i) + \min_{j=1\ldots m} l_j$$

avec

$$for\ j = 1 \ldots m,\ l_j = \pi(o) \sum_{l=0}^{j-1} d_l - \pi(i) \sum_{l=1}^{j} k_l$$

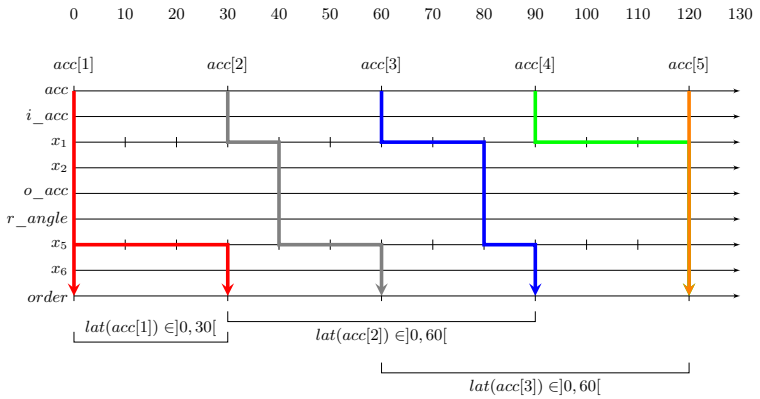


Fig. 7. Dépendance de données et latence de la chaîne L_2

Exemple 4 *Considérons la chaîne* $L_2 = (acc,\ i\_acc,\ x_9,\ x_{10},\ o\_acc,\ r\_angle,\ x_7,\ x_8,\ order)$ *de mot de dépendance de données* $w_{L_2} = (-1,0)(1,2)(1,1)(1,1)(2,2)$. *D'après la proposition 1, on a :*

- $l_0 = \pi(order)d_0 - \pi(acc)k_0 = 0$
- $l_1 = \pi(order)(d_0 + d_1) - \pi(acc)(k_0 + k_1) = 30 \cdot 2 - 30 = 30$
- $l_2 = \pi(order)(d_0 + d_1 + d_2) - \pi(acc)(k_0 + k_1 + k_2) = 30 \cdot 3 - 30 \cdot 2 = 30$
- $l_3 = \pi(order)(d_0+d_1+d_2+d_3) - \pi(acc)(k_0+k_1+k_2+k_3) = 30 \cdot 4 - 30 \cdot 3 = 30$

Alors

$$WCL(L_2) = \phi(order) - \phi(acc) + \pi(order) + \max_{i=0\ldots3} l_i = 30 + 30 = 60$$

En appliquant la proposition 2 on obtient également que $BCL(L_2) = 0$. *La latence de* L_2 *est donc dans l'intervalle* $]0, 60[$, *ce qui est confirmé par la figure 7 qui déroule sur une hyper-période les dépendances internes de la chaîne.*

Exemple 5 *Considérons la chaîne* L_3 *de* r_pos *à* $order$. *Son mot de dépendance de données est* $w_{L_3} = (-1,0)(1,3)(1,1)(1,3)$. *On obtient alors* $WCL(L_3) = 60ms$ *et* $BCL(L_3) = 0$. *Enfin, considérons la chaîne* $L_4 = (angle,\ o\_angle,\ SL\_status,\ x_7,\ x_8,\ x_9,\ PL\_status,\ x_{10},\ x_{11},\ x_{12},\ GL\_status,\ FCS\_status)$ *qui retro-propage une erreur éventuelle. Supposons que l'angle physiquement mesuré sur la gouverne soit incorrect et soit détecté comme tel par* SL. SL_status *est positionné à faux puis envoyé vers le cockpit via* PL *et* GL. *La question est alors : combien de temps après la mesure de l'angle erroné l'alarme est elle retournée dans le cockpit ? L'application des propositions 1 et 2 au mot de dépendance de données* $w_{L_4} = (-1,2)(2,1)(2,1)(2,1)$ *de la chaîne donne* $WCL(L_4) = 165ms$ *et* $BCL(L_4) = 105ms$. *En cas d'erreur sur la mesure de l'angle, on peut donc garantir, sous l'hypothèse que l'implantation temps réel du système satisfait l'hypothèse synchrone relachée, que l'équipage sera averti en au plus* $165ms$.

5 Fraîcheur pire cas

La notion de fraîcheur caractérise une sortie, au moment où est elle consommée, par rapport à l'entrée dont elle dépend ; alors que la latence est une notion qui se rapporte à l'entrée (on parle de la latence d'une entrée et de la fraîcheur d'une sortie). Soit une chaîne $L = (i, \ldots, o)$, et supposons que o est utilisée à t par un dispositif en aval de la chaîne, par exemple un actionneur de gouverne. La fraîcheur de o à t est l'âge de o en prenant comme date de naissance la date d'arrivée de l'occurrence de i dont elle dépend. Si donc o continue à être utilisée par l'actionneur à $t + \delta$ tout en dépendant de la même occurrence de i (par exemple parce qu'aucune nouvelle occurrence de i n'est arrivée), alors la fraîcheur augmentera d'autant (de δ). A titre d'exemple considérons la chaîne $L_1 = (angle, o\_angle, order)$ mono-périodique d'horloge $(30, 0)$. Soit une date $t \in [30k, 30(k+1)[$ pour $k \geqslant 0$, alors deux cas sont possibles pour la fraîcheur de *order* à t :

1. Soit une nouvelle occurrence de *order* a été produite dans $[30k, 30(k+1)[$ avant t. Dans ce cas, c'est cette occurrence qui est utilisée à t par la servo-commande. Elle dépend de *angle* acquis à la date $30k$. Sa fraîcheur à t est donc $t - 30k$.
2. Soit aucune nouvelle occurrence de *order* n'a été encore produite avant t (par exemple pour des raisons d'ordonnancement interne à l'implantation temps-réel). Dans ce cas, c'est l'occurrence précédente qui est utilisée à t et qui dépend de *angle* reçu à la date $30(k-1)$. Sa fraîcheur est $t - 30(k-1)$.

En supposant dans le pire cas que *order* est produit à la fin de sa période (juste avant $30(k+1)$), alors la fraîcheur maximale de *order* est $30(k+1) - 30(k-1) = 60ms$. Alors que la latence maximale de la chaîne (de *angle* vers *order*) est $30ms$. Généralisons ce raisonnement.

Définition 4 (Fraîcheur) *Soit une chaine $L = (i, \ldots, o)$. Soit l'ensemble des couples d'indices (p, q) tels que $o^p \leftarrow i^q$. La fraîcheur pire cas notée $WCF(L)$ est définie par :*

$$WCF(L) \stackrel{def}{=} \max_{\{(p,q)|o^p \leftarrow i^q\}} (date(o^p) - date(i^q)) + 2\pi(o)$$

Le terme $2\pi(o)$ permet de prendre en compte le pire temps possible induit par l'ordonnancement temps réel pour produire l'occurrence suivante de o, c'est-à-dire celle qui « écrasera » l'occurrence utilisée par le dispositif externe.

Il est alors possible de calculer WCF à partir du mot de dépendance de données caractérisant la chaîne fonctionnelle.

Proposition 3 (Fraîcheur pire cas). *Soit $L = (i, \ldots, o)$ une chaîne fonctionnelle de mot de dépendance de données $w = (-1, d_0)(k_1, d_1) \ldots (k_m, d_m)$ (caractérisant les dépendances des occurrences de o vis-à-vis des occurrences de i). Alors la pire fraîcheur de L, notée $WCF(L)$, est :*

$$WCF(L) = \phi(o) - \phi(i) + \pi(o) + \pi(i) + \max_{j=1\ldots m} f_j$$

avec

$$for\ j = 1 \dots m,\ \ f_j = \pi(o) \sum_{l=0}^{j} d_l - \pi(i) \sum_{l=1}^{j} k_l$$

Exemple 6 *Considérons à nouveau la chaîne* $L_2 = (acc,\ i\_acc,\ x_9,\ x_{10},\ o\_acc,\ r\_angle,\ x_7,\ x_8,\ order)$ *de mot de dépendance de données* $w = (-1,0)(1,2)(1,1)(1,1)(2,2)$. *Alors selon la proposition 3 :*

- $f_1 = \pi(order)(d_0 + d_1) - \pi(acc)k_1 = 30 \cdot 2 - 30 \cdot 1 = 30$
- $f_2 = \pi(order)(d_0 + d_1 + d_2) - \pi(acc)(k_1 + k_2) = 30 \cdot 3 - 30 \cdot 2 = 30$
- $f_3 = \pi(order)(d_0 + d_1 + d_2 + d_3) - \pi(acc)(k_1 + k_2 + k_3) = 30 \cdot 4 - 30 \cdot 3 = 30$
- $f_4 = \pi(order)(d_0+d_1+d_2+d_3+d_4)-\pi(acc)(k_1+k_2+k_3+k_4) = 30 \cdot 6 - 30 \cdot 5 = 30$

La pire fraîcheur de order vis-à-vis de acc est donc $WCF(L_2) = \phi(order) - \phi(acc) + \pi(acc) + \pi(order) + \max_{i=1\dots4} f_i = 30 + 30 + 30 = 90$. *Ce qui est confirmé par la figure 7.*

6 Réactivité pire cas

Soit une chaîne $L = (i, \dots, o)$. La réactivité de L est l'intervalle de temps minimal pendant lequel au moins une occurrence de i sera nécessairement prise en compte dans le calcul de o. C'est donc la plus grande distance séparant deux occurrences distinctes de i impactant deux occurrences consécutives de o.

Définition 5 (Réactivité) *Soit une chaîne* $L = (i, \dots, o)$. *Soit l'ensemble des occurrences* i^q *tels qu'il existe une occurrence* o^p *avec* $o^p \leftarrow i^q$ *(*o^p *dépend de* i^q*). La réactivité de L notée* $WCR(L)$ *est le plus grand intervalle de temps séparant deux occurrences* i^q *consécutives dans cet ensemble.* $WCR(L)$ *est définie par :*

$$WCR(L) \stackrel{def}{=} \max_{\{(q,q')|q<q' \ et\ \exists p\ o^p \leftarrow i^q \ et\ o^{p+1} \leftarrow i^{q'}\}} (date(i^{q'}) - date(i^q))$$

Proposition 4 (Réactivité pire cas). *Soit* $L = (i, \dots, o)$ *une chaîne fonctionnelle de mot de dépendance de données* $w = (-1, d_0)(k_1, d_1) \dots (k_m, d_m)$ *(caractérisant les dépendances des occurrences de o vis-à-vis des occurrences de i). Alors la pire réactivité de L, notée* $WCR(L)$, *est :*

$$WCR(L) = (\max_{j=1\dots m} k_j)\pi(i)$$

Exemple 7 *Considérons à nouveau la chaîne* $L_2 = (acc,\ i\_acc,\ x_9,\ x_{10},\ o\_acc,\ r\_angle,\ x_7,\ x_8,\ order)$ *de mot de dépendance de données* $w = (-1,0)(1,2)(1,1)(1,1)(2,2)$. *Alors selon la proposition 4* $WCR(L_2) = 2 \cdot 30 = 60ms$. *La chaîne n'est donc pas réactive à des variations de acc qui durent moins que* $60ms$. *Ce qui est confirmé par la figure 7 qui montre que* $acc[4]$ *est perdu, écrasé par* $acc[5]$. *En conséquence, si une rafale de vent survient et provoque une accélération verticale juste après la mesure de* $acc[3]$ *et cesse juste avant celle de* $acc[5]$, *c'est-à-dire sur une durée juste inférieure à* $60ms$, *alors cette rafale de vent ne sera pas perçue par le système. Notons encore une fois que nous n'avons pas eu besoin de connaître l'implantation du système pour prédire ce résultat.*

7 Conclusion

Cet article présente trois nouvelles propriétés : des propriétés de latence, fraîcheur et réactivité. Ces grandeurs peuvent être calculées de façon exacte à partir du mot de dépendance de données caractérisant la chaîne fonctionnelle considérée. L'objectif est de permettre l'analyse de ces propriétés temporelles tôt dans le cycle de développement avant d'effectuer les choix d'implantation.

Les nœuds importés sont supposés synchrones au sens où ils n'introduisent pas de latence. Dans une approche de spécification hiérarchique, cette hypothèse devra être abrogée, un nœud importé pouvant devenir lui-même un assemblage complexe. Les prochains travaux consisteront à lever cette restriction.

References

[Cur05] A. Curic. *Implementing Lustre programs on distributed platforms with real-time constraints*. PhD thesis, Univ. Joseph Fourier, Grenoble, 2005.

[FBLP08] J. Forget, F. Boniol, D. Lesens, and C. Pagetti. A multi-periodic synchronous data-flow language. In *11th IEEE High Assurance Systems Engineering Symposium (HASE'08)*, Nanjing, China, December 2008.

[For09] J. Forget. *A Synchronous Language for Critical Embedded Systems with Multiple Real-Time Constraints*. PhD thesis, Univ. de Toulouse, 2009.

[HCRP91] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programming language LUSTRE. In *Proceedings of the IEEE*, pages 1305–1320, 1991.

[LBEP11] M. Lauer, F. Boniol, J. Ermont, and C. Pagetti. Latency and freshness analysis on IMA systems. In *Emerging Technologies and Factory Automation (ETFA), Toulouse, France*. IEEE, septembre 2011.

[LBT01] J.Y. Le Boudec and P. Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*. Springer-Verlag, 2001.

[LPT09] K. Lampka, S. Perathoner, and L. Thiele. Analytic real-time analysis and timed automata: a hybrid method for analyzing embedded real-time systems. In *EMSOFT'09*, pages 107–116, 2009.

[Mar04] S. Martin. *Maîtrise de la dimension temporelle de la Qualité de Service dans les Réseaux*. PhD thesis, Univsersité Paris XII, 2004.

[PFB+11] C. Pagetti, J. Forget, F. Boniol, M. Cordovilla, and D. Lesens. Multi-task implementation of multi-periodic synchronous programs. *Discrete Event Dynamic Systems*, 21(3):307–338, 2011.

[RGR09] A. Rahni, E. Grolleau, and M. Richard. An efficient response-time analysis for real-time transactions with fixed priority assignment. *ISSE*, 2009.

[WBFP13] R. Wyss, F. Boniol, J. Forget, and C. Pagetti. End-to-end latency computation in a multi-periodic design. In *SAC*, 2013.

[WBPF12] R. Wyss, F. Boniol, C. Pagetti, and J. Forget. Calcul et vérification de propriétés de latence sur une spécification fonctionnelle synchrone multi-périodique. In *AFADL'12*, pages 10–25, 2012.

[WEEea08] R. Wilhelm, J. Engblom, A. Ermedahl, and et al. The worst-case execution-time problem - overview of methods and survey of tools. *ACM Trans. Embedded Comput. Syst.*, 7(3), 2008.