Vérification d'objets temps réel à l'aide des Réseaux de Petri et de la Logique Linéaire

François Taïani[*], Mario Paludetto[*], Jérôme Delatour[**], Thierry Cros[1]

[*] Laboratoire d'Analyse et d'Architecture des Systèmes,
LAAS-CNRS, 7 av. du Colonel Roche, F-31077 Toulouse CEDEX 4, France
Tél.: + 33 (0) 5 61 33 62 60, Fax : +33 (0) 5 61 33 69 36
{francois.taiani, mario.paludetto}@laas.fr

[**] Ecole Supérieure d'Electronique de l'Ouest, 4 rue Merlet de la Boulaye, BP 926,
49009 Angers cedex 01. Tél.: +33 (0) 2 41 86 67 90, Fax : +33 (0) 2 41 87 99 27
jerome.delatour@eseo.fr

Résumé: Cet article aborde la validation de systèmes temps réels dans un cadre de conception orientée objet. Plus spécifiquement nous étudions comment certaines contraintes temporelles imposées par un objet sur son environnement (par exemple délai de réponse borné) peuvent être vérifiées. L'étude considère des objets issus de la méthode de conception temps réel UML / PNO qui allie UML (Unified Modeling Language) aux réseaux de Petri. L'idée principale consiste à appliquer une technique de réécriture basée sur la Logique Linéaire de Girard pour calculer sous forme symbolique les durées de scénario induites par la requête d'un objet. Sur des exemples simples, nous montrons en quoi consiste cette approche et discutons de ses points forts et de ses limitations.

Mots Clefs: Conception Orientée Objet, Systèmes Temps Réel embarqués, Réseaux de Petri, Logique Linéaire de Girard, Intégration Logicielle, Vérification, Contraintes Temporelles.

Abstract: This paper deals with the temporal verification of real-time systems from an object-oriented point of view. More specifically, we are interested in explicit temporal constraints that may be set on service invocations from a client object (for example a bounded response time). The objects of the study stem from the real-time development method UML / PNO (Unified Modeling Language / Petri Net Objects), which associates the UML notation with the Petri Net formalism. The main idea is to use a rewriting technique stemming from the Linear Logic theory of J.Y. Girard to calculate the duration of an object's invocation. Using small examples we show how symbolic duration formulae can be computed, and discuss the advantages and limitations of such an approach.

Key Words: Object Oriented Design, Real-Time Embedded Systems, Petri Nets, Linear Logic of Girard, Software Integration, Verification, Temporal Constraints.

Introduction

Des circuits intégrés aux micronoyaux temps réel, l'industrie de l'embarqué a vu apparaître ces dernières années des composants sur étagère (*Components Off The Shelf*, COTS) d'une complexité toujours croissante. Contraints de profiter au mieux des acquis du passé et de réduire les coûts et les délais, les acteurs industriels intègrent des composants tiers dans des systèmes toujours plus critiques économiquement. Ce contexte, caractérisé par la multiplicité des intervenants et des systèmes, pose des problèmes d'intégration de plus en plus difficiles. Ceci à tel point que ces problèmes constituent, aujourd'hui, la pierre d'achoppement de nombreux projets d'envergure.

[1] A la date de mise sous presse, l'auteur n'a pas d'adresse fixe (changement de société). Il est possible d'avoir ses coordonnées en s'adressant aux deux premiers auteurs.

Les réflexions menées sur cette problématique font maintenant le succès industriel des concepts orientés objet. L'encapsulation du comportement, la séparation des concepts d'interface et d'implémentation, ont permis l'émergence de standards d'interopérabilité logicielle, essentiels au succès et à la diffusion de composants sur étagère. Cependant, les techniques à objets usuelles prennent en général mal en compte les contraintes temporelles, aussi bien causales [BriGue96] que quantitatives. Or, il est bien évident que celles-ci apparaissent comme des besoins essentiels dans le processus de développement des systèmes temps réel.

Dans cet article, nous considérons les contraintes temporelles relatives aux interfaces des objets [Lee00]. Nous nous intéressons en particulier à la vérification de la compatibilité, en termes temporels, des interfaces mises en relation dans des communications inter-objets. Plus précisément, nous considérons les contraintes temporelles quantitatives (durées explicites) auxquelles peuvent être soumises les requêtes émises par un objet. Notre réflexion est menée dans le contexte de la méthode de développement des systèmes temps réel proposée par Paludetto et Delatour, UML / PNO [PaDe99, DePa99, DePa98]. Nous proposons d'utiliser une technique de calcul de durées basée sur la Logique Linéaire, elle-même issue des travaux de R. Valette et B. Pradin-Chézalviel [Prad+99]. Cette approche nous permet de proposer une première solution au problème de vérification mentionné ci-dessus.

Cet article est composé de deux sections principales. Dans la première section nous introduisons la Logique Linéaire et nous présentons comment un fragment de cette logique peut être utilisé pour calculer des durées de scénarii dans les réseaux de Petri. La seconde section est consacrée à une application permettant d'illustrer la vérification du respect de contraintes temporelles sur les interfaces d'objets, ceci dans le cadre de la méthode UML / PNO. Enfin, nous concluons en résumant les apports les plus intéressants, selon nous, de cette technique pour l'intégration de composants logiciels, puis en soulignant les difficultés que nous avons identifiées et les orientations que nous pensons donner à la poursuite de ce travail.

La Logique Linéaire dans le cadre des réseaux de Petri

La Logique Linéaire a été définie par J.Y. Girard à la fin des années 80 [Girard87]. Il s'agit d'une forme très particulière de logique qui résulte de l'appauvrissement des règles de calcul de la logique booléenne classique. En particulier, la règle d'idempotence n'existe plus: une proposition *A* donnée n'est notamment plus équivalente à *A et A* ou encore *A ou A*. Si ce n'est par son nom, cette logique n'a aucun rapport avec la Logique Temporelle Linéaire (LTL) [Lam80], utilisée couramment en *Model Checking* [Merz00]. Dans cette partie, nous nous limitons à une présentation informelle du fragment de Logique Linéaire qui nous sera nécessaire pour la suite de l'article. Le lecteur intéressé par de plus amples informations pourra se reporter à l'exposé très didactique proposé dans [Girault97] ainsi qu'aux nombreux articles de Girard [Girard90, Girard95].

Pour un exposé plus aisé, nous présentons conjointement l'interprétation en logique linéaire du calcul des réseaux de Petri proposé par F. Girault [Girault97] et son extension temporelle élaborée par B. Pradin-Chézalviel, R. Valette et L.A. Künzle [Künz97, Prad+99]. Ces résultats constituent la base théorique de notre réflexion pour une vérification des contraintes temporelles inhérentes à la composition d'objets temps réel que nous présentons à la section suivante.

La Logique Linéaire est une logique de *ressources qui s'usent quand on s'en sert* [Girard90], c'est-à-dire de propositions qui peuvent être consommées et produites. Le lien entre Logique Linéaire et réseaux de Petri s'opère en associant à chaque place d'un réseau de Petri (*a, b* et *c* sur la figure 1) un type d'atome de la logique (notés *A, B, C*…). Un jeton du réseau est alors représenté par un atome du type correspondant à la place du jeton, et un marquage par une accumulation d'atomes. Cette notion d'accumulation est traduite par l'opérateur linéaire *fois*, noté ⊗. Dans l'exemple de la figure 1, le marquage initial (*a:3, b:1, c:0*) se traduit ainsi par la proposition $A⊗A⊗A⊗B$. Chaque atome/jeton est annoté par une date qui représente l'instant auquel il a été produit. Nous notons ainsi $A(d_A)$ un atome/jeton de type *A* produit à l'instant d_A dans la place *a*. Si nous considérons le marquage de la figure 1 comme initial, nous le représenterons avec cette convention par la proposition $A(0)⊗A(0)⊗A(0)⊗B(0)$.

Les transitions d'un réseau de Petri sont exprimées par le biais de l'implication linéaire, notée ⊸. L'implication linéaire exprime la possibilité de produire des ressources à partir de la consommation d'autres ressources. Ainsi la transition *t* de la figure 1 sera représentée par $[A⊗A⊗B⊸ C](\delta)$. L'annotation (δ) indique que la production d'un atome de type *C* (c.à.d. d'un jeton dans la place *c*) à partir de deux atomes de type *A* (deux jetons dans la place *a*) et d'un atome de type *B* (un jeton dans la place *b*) dure un temps δ [2,3]. Une implication linéaire comme $[A⊗A⊗B⊸ C](\delta)$ est elle-même une ressource, c'est-à-dire que la possibilité de production disparaît dès que l'implication est utilisée. Comme toute ressource, une implication linéaire peut s'accumuler, chaque exemplaire représentant une possibilité de plus d'utiliser la règle de consommation / production.

Dans le cas de la figure 1, nous souhaitons étudier les évolutions possibles du réseau dans lesquelles la transition *t* est utilisée une fois, ce que nous transposons en Logique Linéaire par la représentation du marquage $A(0)⊗A(0)⊗A(0)⊗B(0)$ et par une instance de l'implication $[A⊗A⊗B⊸ C](\delta)$ représentant *t*.

En logique linéaire, les calculs se définissent dans le cadre, plus large, du calcul des séquents [Girault97]. Dans le cadre de notre travail, et par souci de simplicité, nous utilisons en lieu et place de ce calcul un ensemble de règles de réécriture des

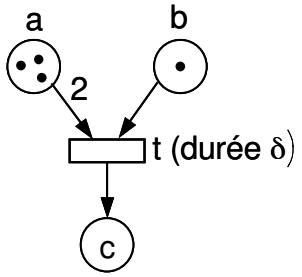
---

[2] Dans la suite de l'article, nous nous limitons à des réseaux de Petri à jetons banalisés et à transitions temporisées.

[3] Les dates de production des ressources et les durées de production peuvent, aux choix, être modélisées par des valeurs ponctuelles ou des intervalles. L'utilisation d'intervalles permet de modéliser une incertitude sur ces grandeurs, si nécessaire.

propositions de logique linéaire[4]. Ces règles traduisent, de fait, le fonctionnement d'un réseau de Petri. Par exemple le franchissement de la transition *t* du réseau de la figure 1 s'exprimera par la réécriture suivante:

$$A(0)\otimes A(0)\otimes A(0)\otimes B(0)\ ;\ [A\otimes A\otimes B \multimap C](\delta) \rightsquigarrow A(0)\ ;\ C(\delta)$$



$A(0)\otimes A(0)\otimes A(0)\otimes B(0)\ ;\ [A\otimes A\otimes B \multimap C](\delta)$

Figure 1: Réseau de Petri et sa traduction en logique linéaire

Dans le cas d'un réseau plus fourni tel que celui de la figure 2, le calcul de la durée d'un scénario prend la forme d'une chaîne de réécriture (figure 3). Cette chaîne nous permet de calculer la durée de passage du marquage $A\otimes A\otimes B\otimes C$ au marquage *G*. Dans ce cas bien précis, ce passage ne s'obtient que par un seul scénario: deux tirs successifs de la transition *r*, tirs indépendants des transitions *t* et *s*, puis tir de la transition *u*. Dans la réécriture de la figure 3, *s* est tirée en premier, mais ceci n'a aucune influence sur le résultat final [Prad+99]. Plus formellement les règles de réécriture s'appliquent sur des propositions de la forme:

<une liste de marquages[5]> ; <une liste de transitions[5]>

La liste de transitions d'une telle proposition est une sorte de *réserve à transitions* à partir de laquelle le scénario se construit. La liste des marquages décrit ce que nous nommons *l'étape courante* du calcul [Prad+99]. Sur la figure 3, $E(2\rho+\sigma),\ D(2\rho),\ D(\rho)$ est, par exemple, une étape courante du calcul présenté (étape *(iv)*). Il est à noter que la présence de l'étape courante $E(2\rho+\sigma),\ D(2\rho),\ D(\rho)$ n'implique pas nécessairement l'observation du marquage $E(2\rho+\sigma)\otimes D(2\rho)\otimes D(\rho)$ au cours du scénario étudié. Une étape courante traduit l'observation séparée de marquages partiels, sans conjecture sur leur simultanéité [Prad+99]. Cette notion est très similaire à celle de *cut* présentée par exemple dans [BeDe87].

---

[4] Le lecteur intéressé pourra se reporter à [Prad+99] pour une présentation directement basée sur le calcul des séquents. Expliqué brièvement, les règles de réécriture que nous utilisons correspondent à l'utilisation *inversée* (i.e. de bas en haut) des règles du calcul des séquents qui ne modifient que le membre gauche Δ d'un séquent $\Delta\vdash\Gamma$. La réécriture répétée d'une proposition revient à construire un arbre de preuve du séquent $\Delta_1\vdash\Delta_2$, où Δ_1 est la proposition de départ de la chaîne de réécriture et Δ_2 la proposition d'arrivée.

[5] Pour ne pas alourdir notre exposé, dans tout ce qui suit, nous appelons *marquage* aussi bien le marquage effectif d'un réseau que la proposition de logique linéaire qui le représente. Nous faisons de même pour les implications linéaires que nous appelons *transitions*.

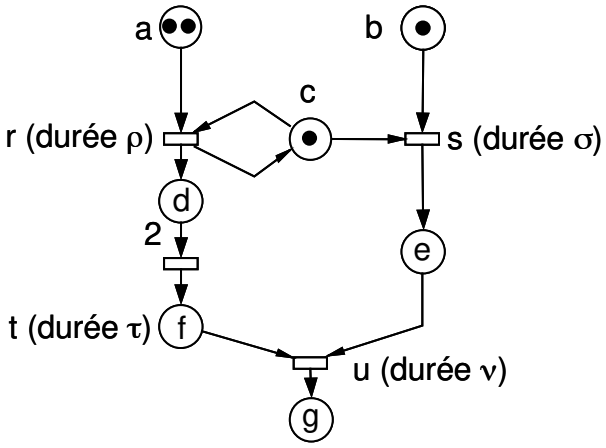


Figure 2: Réseau de Petri avec boucle et ressource partagée

Les règles de réécriture et la génération des estampilles temporelles traduisent les règles de tir du réseau de Petri *sans attente*. C'est-à-dire qu'un réseau n'attend pas dans une situation d'inactivité *instable* (i.e. dans laquelle au moins une transition peut être tirée, sans qu'aucune autre transition ne soit en cours de tir). Cette sémantique garantit que, dans le cas d'un scénario sans conflits, le calcul de réécriture fournit bien la durée d'exécution effective du scénario (notion de scénario complètement spécifiés, [Prad+99]).

(i) $\mathbf{A}(0)\otimes\mathbf{A}(0)\otimes B(0)\otimes\mathbf{C}(0)$; $2\times [\,\mathbf{A}\otimes\mathbf{C} \multimap \mathbf{D}\otimes\mathbf{C}\,]_{\mathbf{r}}(\rho)$, $[\,B\otimes C \multimap E\,]_s(\sigma)$, $[\,D\otimes D \multimap F\,]_t(\tau)$, $[\,F\otimes E \multimap G\,]_u(\upsilon)$

$\rightsquigarrow$

(ii) $\mathbf{D}(\rho)\otimes\mathbf{C}(\rho)$, $\mathbf{A}(0)\otimes B(0)$; $[\,\mathbf{A}\otimes\mathbf{C} \multimap \mathbf{D}\otimes\mathbf{C}\,]_{\mathbf{r}}(\rho)$, $[\,B\otimes C \multimap E\,]_s(\sigma)$, $[\,D\otimes D \multimap F\,]_t(\tau)$, $[\,F\otimes E \multimap G\,]_u(\upsilon)$

$\rightsquigarrow$

(iii) $\mathbf{D}(2\rho)\otimes\mathbf{C}(2\rho)$, $D(\rho)$, $\mathbf{B}(0)$; $[\,\mathbf{B}\otimes\mathbf{C} \multimap \mathbf{E}\,]_{\mathbf{s}}(\sigma)$, $[\,D\otimes D \multimap F\,]_t(\tau)$, $[\,F\otimes E \multimap G\,]_u(\upsilon)$

$\rightsquigarrow$

(iv) $\mathbf{E}(2\rho+\sigma)$, $\mathbf{D}(2\rho)$, $\mathbf{D}(\rho)$; $[\,\mathbf{D}\otimes\mathbf{D} \multimap \mathbf{F}\,]_{\mathbf{t}}(\tau)$, $[\,F\otimes E \multimap G\,]_u(\upsilon)$

$\rightsquigarrow$

(v) $\mathbf{E}(2\rho+\sigma)$, $\mathbf{F}(2\rho+\tau)$; $[\,\mathbf{F}\otimes\mathbf{E} \multimap \mathbf{G}\,]_{\mathbf{u}}(\upsilon)$

$\rightsquigarrow$

(vi) $\mathbf{G}(\max(2\rho+\sigma, 2\rho+\tau)+\upsilon)$;

Les transitions en gras sont les transitions qui sont tirées à la réécriture suivante.
Les jetons en gras sont les jetons produits ou consommés.
Les flèches soulignent les relations de consommation / production entre jetons.

Figure 3: Calcul de la durée du scénario ***r; r; (d ∥ s); u*** par réécriture

D'autres techniques que celle proposée par Valette, Pradin-Chézalviel et al. permettent d'évaluer la durée d'un scénario discret (simulation, analyse structurelle, graphe d'évènements). Le grand intérêt et l'originalité d'une démarche basée sur un

formalisme de ressources comme la Logique Linéaire peut se résumer par les points suivants:

a) Le marquage initial du réseau étudié est pris en compte dans le parallélisme des évènements [Girault97]. Par exemple sur la figure 4, les transitions *t* et *s* forment structurellement une séquence. Cependant, du fait même du marquage que nous considérons (un jeton dans *a*, un jeton dans *b*), *s* et *t* peuvent en pratique être tirées en parallèle. La Logique Linéaire, parce qu'elle intègre une représentation des marquages, décèle ce niveau de parallélisme, ce que ne permettrait pas une simple analyse structurelle du réseau.

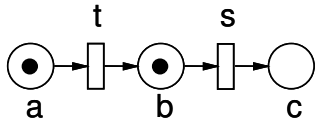


Figure 4: *t* et *s* sont structurellement en séquence, mais peuvent être tirées en parallèle du fait du double marquage [Girault97]

b) L'approche est adaptée aux réseaux non saufs, ainsi qu'aux scénarii avec boucles.

c) La Logique Linéaire offre une représentation compacte du comportement d'un réseau. Cette compacité est due, d'une part, à la non différenciation entre instances d'une même ressource produites à la même date (contrairement aux approches basées sur les réseaux d'occurrence et le *dépliage* d'un réseau de Petri). D'autre part, le concept d'étape courante permet de mener le calcul du réseau *sur plusieurs fronts* en éliminant les entrelacements (ce qui rejoint les travaux sur les ordres partiels). Dit autrement, notre technique traite simultanément plusieurs *portions* de marquage, plusieurs marquages partiels, sans que ces marquages se côtoient nécessairement dans le scénario final.

d) Nous obtenons une formule symbolique (avec des variables) qui couvre toutes les durées possibles des transitions, contrairement à une valeur numérique unique que livrerait une simulation ou un *model checker* temporisé [Yovine97] [PeLa00].

e) Les règles de réécriture, parce qu'elles manipulent directement des chaînes de symboles, sont très faciles à implémenter.

Contraintes Temporelles, Composition logicielle et Logique Linéaire

Dans cette partie, nous montrons comment le calcul de durées de scénarii présenté à la section précédente peut être utilisé pour vérifier la compatibilité des contraintes temporelles associées aux interfaces d'objets communicants. Notre présentation se fonde sur la méthode de conception orientée objet UML / PNO proposée par Paludetto et Delatour [PaDe99, DePa99, DePa98]. Nous rappelons ici brièvement les grandes

lignes de cette méthode. Le lecteur intéressé pourra se reporter aux articles mentionnés pour une discussion plus détaillée de ses principes et de leurs motivations.

UML / PNO s'appuie sur les concepts orientés objet d'UML (Unified Modeling Language, [*UML1.3]) et le formalisme des objets à réseaux de Petri (Petri Net Objects, PNO, [Palu91]) pour proposer une démarche de conception des systèmes temps réel. L'idée maîtresse de l'approche consiste à décrire les différents aspects comportementaux d'un objet (spécification, implémentation, contraintes) à l'aide de réseaux de Petri.

Selon les objectifs de la modélisation et la phase de développement, la méthode UML/PNO préconise l'utilisation de la classe de réseaux de Petri adaptée à la complexité de l'objet, réseaux de Petri colorés ou non, avec ou sans caractère temporel. Dans le cas présent, nous nous intéressons à la vérification d'interfaces contraintes par le temps et, comme dans la section précédente, nous utilisons des réseaux non colorés à transitions temporisées.

Les communications entre objets peuvent alors être représentées selon le type de messages échangés (synchrones, asynchrones, ... etc.) et la granularité du modèle par des places ou des transitions partagées. Dans le cadre de cet exposé nous nous sommes restreints à des appels synchrones que nous modélisons de manière éclatée par deux places partagées: une place d'invocation, et une place de retour d'appel (voir figure 5).

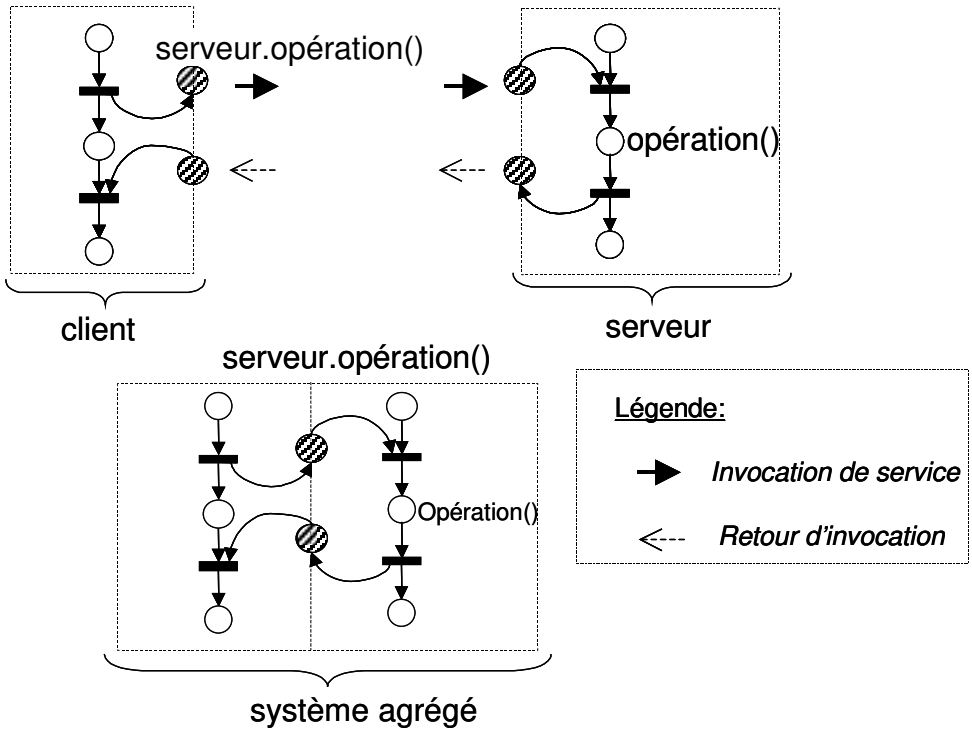


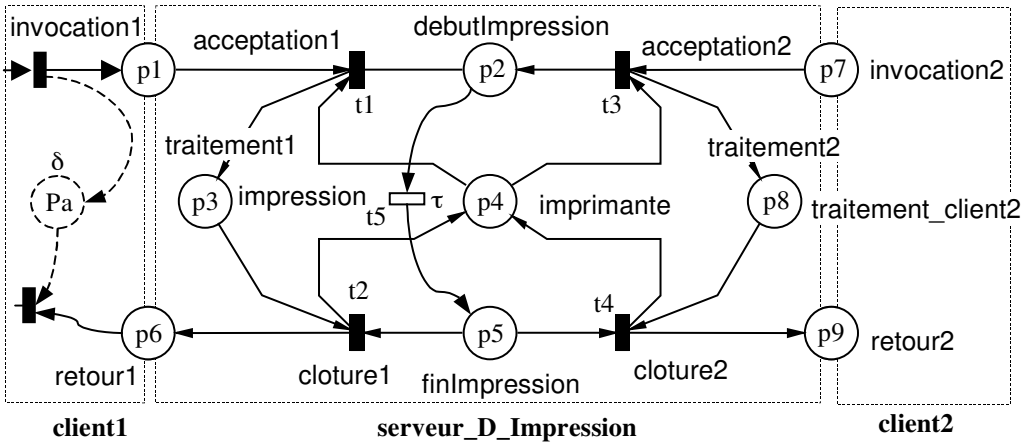
Figure 5: Appel et réception de messages synchrones en UML / PNO

L'ensemble des places qu'un objet partage avec son environnement constitue son interface de communication. Il s'agit de fait d'une forme restreinte de l'interface usuelle d'un objet, puisque le formalisme des réseaux que nous avons choisi n'englobe pas

d'informations de type ni de signatures de méthodes. Cependant, cette interface relativement restreinte permet d'exprimer un certain nombre de contraintes de précédence ainsi que des contraintes temporelles comme le montre l'exemple de la figure 6.

Sur cet exemple, un objet *serveur_D_Impression* offre ses services à deux clients en exclusion mutuelle. Le *client1*, dont le comportement interne n'est pas représenté, est soumis à une contrainte temporelle sur son interface: la réalisation du service d'impression de doit pas durer plus de δ unités de temps. Nous appelons de telles contraintes *contraintes temporelles attendues* pour souligner le caractère de spécification qu'elles recouvrent: l'achèvement de l'impression en un temps inférieur à δ, tout en n'étant pas garanti *a priori*, constitue une pré-condition au bon fonctionnement de l'objet *client1*, et donc du système.

La réalisation d'une telle contrainte ne dépend pas uniquement du serveur d'impression, mais des requêtes tiers (étrangères à la collaboration *serveur_D_Impression / client1*) auquel le serveur peut être soumis. Dans l'exemple de la figure 6 ces interactions *parasites* (du point de vue du *client1*) sont représentées par l'objet *client2*[6].



Contrainte Temporelle Attendue sur client1: ***Durée (invocation1 → retour1) ≤ δ***
Toutes les transitions sont instantanées sauf t5 dont la durée est τ.

Figure 6: Serveur d'impression à deux clients

La vérification de la contrainte de temps à l'interface de l'objet *client1* nécessite donc la prise en compte des objets inhérents aux scénarios mis en jeu. Le couplage de plusieurs objets résulte du caractère critique de la ressource d'impression (représentée par la place *imprimante p4*). La place *Pa* a pour rôle de modéliser la *contrainte temporelle attendue* dans le cas d'une automatisation de la vérification de cette contrainte. Le processus de vérification devra calculer les durées des scénarios suscités

[6] Par souci de cohérence, nous nous somme limités ici à des réseaux à jetons banalisés, d'où la relative complexité du réseau proposé. L'utilisation de réseaux de plus haut niveau (réseaux colorés) permettrait, dans la pratique, une représentation plus compacte et plus élégante du même comportement.

par le marquage de *p1* (invocation de la ressource) puis il vérifiera que l'estampille temporelle du jeton qui arrivera en *p6* sera inférieure à l'estampille δ du jeton *vérificateur* en attente en *Pa*.

Pour vérifier l'intégrité de telles contraintes d'interface, notre proposition consiste à calculer les durées de scénarii suivant la technique basée sur la logique linéaire présentée à la section précédente. En effet, parce qu'elle prend en compte l'influence du marquage initial sur le parallélisme du système, cette technique est relativement bien adaptée aux systèmes distribués et donc, aux *états distribués* auxquels nous nous intéressons. De plus, par sa compacité et sa focalisation sur un scénario particulier, le calcul de durées par la logique linéaire évite d'affronter directement le problème de la complexité combinatoire du système bien connu en vérification. La plupart des techniques qui traitent ce problème sont basées sur l'exploitation du graphe d'états et l'entrelacement. Notons que des avancées intéressantes ont été faites dans le domaine (voir par exemple [JuaGal96] ou [Couv+99]). La proposition de cet article est simplement une autre réflexion menée dans le contexte de développement des systèmes temps réel embarqués suivant les paradigmes *objets* et *réseaux de Petri*.

Cette réflexion suscite quelques remarques :

a) Plusieurs marquages initiaux peuvent être considérés pour calculer la durée du ou des scénarii affectant une contrainte temporelle. Est-il nécessaire de les envisager tous? Dans l'exemple de la figure 6, l'objet *client2* va marquer la place *p7* (*invocation2*) lorsqu'il fait la demande d'impression. Par exemple, devons-nous également étudier l'évolution du système à partir du marquage de *p4* et *p9* (*retour2*) qui obligerait, du reste, à évaluer des évolutions internes de *client2*?

b) Dans la plupart des cas, un marquage induit plusieurs scenarii. (Scenarii non complètement spécifiés.) Est-il possible d'éliminer certains d'entre eux *a priori* sans perdre de la qualité ou de l'exactitude sur l'évaluation à effectuer? Ainsi sur la figure 6, les deux transitions *t1* (*acceptation1*) et *t3* (*acceptation2*) sont en conflit effectif avec le marquage représenté. En général, les alternatives de ce type conduisent des chaînes de réécritures et des durées de scenarii différentes (sauf pour l'exemple de la figure 6 pour lequel il y a symétrie des modèles au niveau des appels et des réponses).

c) Les systèmes étudiés sont le plus souvent cycliques, donc avec des traces infinies. Comment à partir d'un nombre fini de scenarii, eux mêmes finis, tirer des déductions sur l'évolution en boucle du système ? Quelle *réserve de transitions* choisir ? Plus généralement, à quelle profondeur de scénario se limiter ?

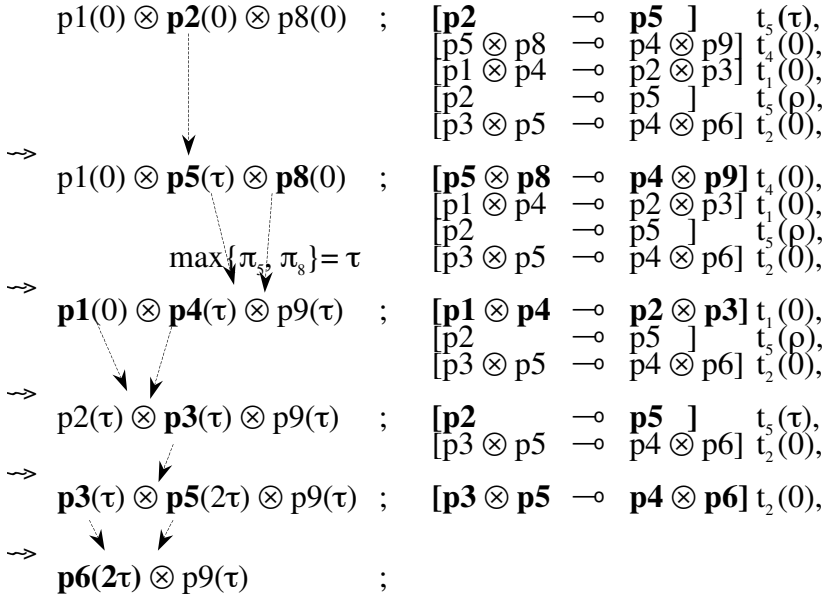
Notons, sans que cela soit une véritable surprise, que ces problèmes réintroduisent l'explosion combinatoire qui avait été évacuée du calcul logique proprement dit. Cependant, dans notre approche, la limitation de l'espace d'états se ramène au choix des scenarii pertinents sur la structure du réseaux et non sur son graphe d'états équivalent, et ceci indépendamment du calcul de durées. Ainsi, le problème des conflits entre

transitions (point b) peut se résoudre (partiellement) en introduisant dans la sémantique de tir une politique d'ordonnancement basée sur les estampilles temporelles[7] (par exemple une politique de type FIFO). Au niveau réseaux de Petri, cela revient à affecter une estampille temporelle π_i aux jetons arrivant dans les places p_i, *i = 1, 2, …, 9*. Dans le cas simple de l'exemple de la figure 6, le marquage initial qui pourrait mettre le modèle en défaut vis-à-vis de la contrainte temporelle à vérifier est :

$$p1(0) \otimes p2(0) \otimes p8(0).$$

En clair, il correspond à la configuration de demande de service du *client1* à l'instant même où le *serveur* accepte la demande du *client2*.

Compte tenu de cette nouvelle sémantique, la vérification de la contrainte de la figure 6 conduit à la séquence de réécriture en logique linéaire suivante :



Ce calcul montre que le scénario a une durée de 2τ. Il correspond au pire cas où le *client1* doit attendre que le service demandé par le *client2* soit exécuté. Ainsi, la contrainte temporelle attendue sera vérifiée si la relation $2 \times \tau \leq \delta$ est elle-même vérifiée. Le calcul de deux scénarios à partir du marquage $p1(0) \otimes p4(0) \otimes p7(0)$ aurait conduit à des durées τ (tir de *t1* avant *t3*) et 2τ (tir de *t3* avant *t1*) pour lesquelles il fallait ensuite prendre la valeur maximum. Malgré son extrême simplicité, cet exemple montre que l'approche que nous avons adoptée demande le calcul d'un nombre de scénarios restreint car elle est basée sur le marquage et la structure du réseau et non sur le graphe d'états équivalent.

---

[7] Dans les systèmes temps réel, l'ordonnancement des tâches fait partie intégrante des missions du serveur. En ajoutant une politique d'ordonnancement à notre modélisation, nous affinons en fait notre représentation du système. Il ne s'agit donc pas ici d'une modification ad-hoc du modèle pour résoudre un problème de calcul. Au contraire, nous éliminons l'indéterminisme qui résultait d'une modélisation trop abstraite du système.

Bien sûr, d'autres solutions plus simples que la Logique Linéaire auraient fourni le même résultat (relativement trivial) à moindre coût. Il s'agissait ici, avant tout, d'illustrer les concepts fondamentaux de notre démarche à l'aide d'un exemple didactique. Un système plus important nous permettrait d'éclairer de manière plus qualitative les avantages de notre approche, mais dépasserait malheureusement le volume restreint de cet article.

En terme de complexité et de performances effectives d'un outil basé sur la réécriture de la logique linéaire, l'algorithme est linéaire (voir travaux référencés en section deux) sauf s'il y a éclatement en sous-ordres partiels (cas du conflit effectif par exemple). Dans de tels cas, il se pose le problème du choix ou de la limitation des scénarios pour améliorer l'efficacité. Les *model-checkers* intégrant des techniques de découpe (*slicing*) pourraient notamment apporter une aide substantive dans la sélection des scénarios. Dans le cadre plus général du calcul symbolique sur des domaines numériques (comme par exemple l'interprétation abstraite sur des domaines tels que les intervalles et les polyèdres convexes), notre approche est plus simple car elle ne manipule que des valeurs *max*. Elle peut paraître réductrice dans un cadre général, mais elle est suffisamment efficace et performante dans le cadre de la vérification des besoins temporels assurés par des objets composant un système temps réel embarqué. La structuration en objets du système n'est probablement pas étrangère à cette efficacité. En effet, le fait que les objets soient vus comme des sous-systèmes à part entière contribue à une diminution de la complexité puisqu'il y a *division* des difficultés. De plus, l'abstraction, la forte cohésion interne et le faible couplage aidant, les états du système non intéressants pour l'objet considéré sont implicitement écartés.

Conclusion et Perspectives

Dans cet article nous avons brièvement décrit comment la logique linéaire pouvait être appliquée au calcul de durées de scenarii dans les objets à réseaux de Petri. Puis, nous avons montré, sur un exemple très simple, comment une telle méthode pouvait être utile à la vérification d'interfaces contraintes par le temps dans le cadre de la conception de systèmes temps réel. A cet effet, nous avons repris le formalisme et les principes fondamentaux de la méthode orientée objet UML / PNO issue de nos travaux.

L'un des intérêts principaux de la Logique Linéaire pour la vérification de systèmes temps réel orientés objet réside, selon nous, dans le découplage qu'elle autorise entre le calcul de durées de scenarii proprement dit et la limitation de l'espace d'états. La Logique Linéaire *ramène* la réduction de l'espace d'états à un choix de scenarii. Ceci permet d'éliminer implicitement certains scenarii *a priori*, ce qui réduit d'autant l'effort de vérification à fournir.

Un autre apport de la logique linéaire consiste en l'obtention d'une durée sous forme symbolique, contrairement à d'autres approches (*Model Checking* temporisé, simulation) qui ne livrent qu'une valeur numérique. Cet aspect nous paraît très prometteur pour le dimensionnement d'architectures, et en particulier pour le paramétrage de composants

sur étagère (COTS). D'autres méthodes fournissent des durées symboliques, mais la Logique Linéaire s'en distingue en ce qu'elle travaille directement sur le réseau de Petri et le marquage considéré, sans requérir de graphe d'évènements déjà partiellement ordonnés.

Nous comptons poursuivre notre étude de l'apport de la Logique Linéaire aux systèmes temps réel en nous basant sur un premier prototype de calcul développé au sein du LAAS [Kabo00]. L'utilisation d'un outil nous permettra de tester plusieurs stratégies de sélection de scenarii et de marquages initiaux, ce que nous avons identifié comme essentiel pour la mise en pratique à grande échelle de notre technique.

Parallèlement, nous poursuivrons notre réflexion sur les possibilités d'extension de la Logique Linéaire à des réseaux de plus haut niveau (t-temporels, colorés), et éventuellement à des formalismes autres que les réseaux de Petri (Machines à Etats d'UML, Action Semantic Language [*OMG00]), afin d'augmenter l'expressivité et de réduire la taille des modèles que nous manipulons dans le cadre du développement des systèmes temps réel embarqués.

Références

[BeDe87] EIKE BEST, RAYMOND DEVILLERS, *Sequential And Concurrent Behaviour In Petri Net Theory*, Theoretical Computer Science, n°55, 1987, 87-136, North-Holland

[BriGue96] JEAN-PIERRE BRIOT, RACHID GUERRAOUI, *Objets pour la programmation parallèle et répartie : intérêts, évolutions et tendances*, Technique et Science Informatiques (TSI), AFCET-Hermès, Paris, France, 15(6):765-800, Juin 1996.

[Couv+99] JEAN-MICHEL COUVREUR, ALAIN GRIFFAULT, DAVID SHERMAN, *Diagrammes de décision pour la vérification de réseaux à files*, Actes du 2ème Congrès sur la Modélisation des Systèmes Réactifs (MSR'99), Cachan (France), Editions Hermès, 1999, pp.61-70

[DePa99] JÉROME DELATOUR, MARIO PALUDETTO, *De HOOD/PNO à UML/PNO : une méthode pour les systèmes temps réels basée UML et objets à réseau de Petri*, Actes du 2ème Congrès sur la Modélisation des Systèmes Réactifs (MSR'99), Cachan (France), Editions Hermès, 1999, pp.223-231

[DePa98] JÉROME DELATOUR, MARIO PALUDETTO, *UML/PNO, A Way to Merge UML and Petri Net Objects for the Analysis of Real-Time Systems*,In Proceedings of the Object-Oriented Technology and Real Time Systems Workshop (ECOOP'98), Bruxelles, Lecture Notes in Computer Science 1543, Springer, 1998, pp.511-514

[Girault97] FRANÇOIS GIRAULT, *Formalisation en logique linéaire du fonctionnement des réseaux de Petri*, Thèse de Doctorat, Université Paul Sabatier, Toulouse, 1997.

[Girard87] J.Y. GIRARD, *Linear Logic*, Theoretical Computer Science, n°50, 1987.

[Girard90] J.Y. GIRARD, *La Logique linéaire*, Pour la Science, n°150, Apr. 1990, pp. 74-85.

[Girard95] J.Y. GIRARD, *Linear Logic: A Survey*,
Cahier du centre de logique, vol. 8, 1995, pp.193-255.

[JuaGal96] GUY JUANOLE , LAURENT GALLON, *Le modèle réseaux de Petri temporisés stochastiques et son adéquation aux systèmes distribués temps critique*, Actes du Congrès sur la Modélisation des Systèmes Réactifs (MSR'96), AFCET, Brest (France), 28 et 29 mars 1996.

[Kabo00] W. ESTHER KABORE, *Réseaux de Petri et Logique Linéaire pour la vérification de systèmes concurrents: algorithmes de preuve*,
Mémoire de Projet de Fin d'Etude, Institut Africain d'Informatique (Libreville, Gabon), LAAS-CNRS (Toulouse, France). Responsables : R. Valette, B. Pradin-Chézalviel. Toulouse, 2000.

[Lee00] E. A. LEE, *What's Ahead for Embedded Software*, Computer, September 2000, Volume 33, Number 9, IEEE Computer Society, pp. 18-26

[Künz97] LUIS ALLAN KÜNZLE, *Raisonnement temporel basé sur les réseaux de Petri pour des systèmes manipulant des ressources*, Thèse de Doctorat, Université Paul Sabatier, Toulouse, 1997.

[Lam80] LESLIE LAMPORT, *Sometimes is sometimes "not never" - on the temporal logic of programs.* In Proceedings of the 7th ACM Symposium on Principles of Programming Languages, page 175-185, January 1980.

[Merz00] STEPHAN MERZ, *Model Checking*, Tutorials of MOVEP'2k, Proceedings of the Summer School on MOdelling and VErification of Parallel processes. Nantes, 19-23 June 2000. Editors: F. Cassez, C. Jard, B. Rozoy, and M. Ryan. pp.52-70

[*OMG00] *Action Semantics for UML RFP*,
http://cgi.omg.org/techprocess/meetings/schedule/Action_Semantics_for_UML_RFP.html, dernière mis à jour: Lundi 10 Juillet 2000.

[PaDe99] MARIO PALUDETTO, JÉROME DELATOUR, *UML et les réseaux de Petri : vers une sémantique des modèles dynamiques et une méthodologie de développement des systèmes temps réel*, Rapport LAAS No99511, Revue L'Objet, Vol.5, N°3-4, pp.443-467, Décembre 1999

[Palu91] MARIO PALUDETTO, *Sur la commande de procédés industriels : une méthodologie basée objets et réseaux de Petri*, Thèse de Doctorat, Université Paul Sabatier, Toulouse, 10 Décembre 1991, N°1071, 270p.

[PeLa00] PAUL PETTERSSON, KIM G. LARSEN, *UPPAAL2k*,
in Bulletin of the European Association for Theoretical Computer Science, volume 70, pages 40-44, 2000

[Prad+99] BRIGITTE PRADIN-CHÉZALVIEL, ROBERT VALETTE, LUIS ALAN KÜNZLE, *Formalisation de scénarios, réseaux de Petri et logique linéaire*, Journées Formalisation des Activités Concurrentes, FAC'99,
CERT-IRIT-LAAS, Toulouse 25-26 février 1999, p.84-95

[*UML1.3] UML REVISION TASK FORCE, *OMG UML v. 1.3*, Object Management Group, http://www.omg.org/cgi-bin/doc?ad/99-06-08.pdf, June 1999.

[Yovine97] SERGIO YOVINE, *KRONOS: A verification tool for real-time systems*, In Springer International Journal of Software Tools for Technology Transfer, Vol. 1, Nber. 1/2, October 1997.