Précédences généralisées et ordonnançabilité des tâches de suivi temps réel d'un laminoir

Pascal Richard[*] — **Francis Cottet**[*] — **Claude Kaiser**[**]

[*] *Laboratoire d'Informatique Scientifique et Industrielle*
ENSMA, BP 40109
F-86960 Futuroscope Cedex

[**] *Cedric/CNAM*
292, rue Saint-Martin
75141, Paris Cedex 3
{richardp,cottet}@ensma.fr, kaiser@cnam.fr

RÉSUMÉ. *Nous étudions dans cet article la validation d'un logiciel de suivi temps réel d'un laminoir d'aluminium. Des tâches périodiques, soumises à des échéances strictes, s'exécutent sur un processeur dans le but de garantir la qualité des feuilles d'aluminium fabriquées. La particularité de cette application industrielle réside dans le fait que les tâches respectent des contraintes de précédence généralisée, i.e. les périodes des tâches en précédence ne sont pas nécessairement égales. Pour valider une application temps réel avec de telles contraintes, nous proposons une approche en deux étapes. La configuration de tâches est tout d'abord transformée en une autre composée uniquement de contraintes de précédence simple, puis en une troisième ne comportant que des tâches indépendantes. Ce dernier ensemble de tâches est utilisé pour valider l'application avec les techniques connues d'analyse d'ordonnançabilité.*

ABSTRACT. *In this paper we study the validation of a real-time software for the following up of a rolling mill. Periodic tasks with deadlines are run on a single processor in order to ensure the quality of the products. The particularity of this application is that tasks are subjected to generalized precedence constraints, i.e. dependent tasks do not work necessarily at the same rate. In order to validate an application with such constraints, we present a two steps approach. The tasks set is first transformed in another one that contains only simple precedence constraints, and subsequently in a third one that contains only independent tasks. The latter task set is used to validate the application with well-known schedulability analysis techniques.*

MOTS-CLÉS : *ordonnancement temps réel, contraintes de précédence généralisée, validation temporelle.*

KEYWORDS: *Real-Time Scheduling, Generalized Precedence Constraints, temporal validation*

APII - JESA. Volume 35 - n° 9/2001, pages 1055 à 1071

1. Introduction

Nous étudions dans cet article la validation du logiciel de suivi temps réel d'un laminoir industriel. Le laminoir considéré fabrique des bobines d'aluminium destinées aux marchés de l'emballage (boîtes de boisson et de conserve). Le rôle de ce laminoir à froid est de réduire l'épaisseur de bande en entrée, à l'aide de quatre cylindres, avec une tolérance minimale exigée sur l'épaisseur de sortie de l'ordre de quelques microns. Le schéma de principe du laminoir est présenté figure 1. Trois calculateurs différents réalisent le contrôle du laminoir et transmettent des comptes rendus au calculateur de suivi temps réel de la production. Le premier calculateur pilote les séquences de positionnement des cylindres du laminoir, régule la pression des vérins de serrage et l'épaisseur de sortie de la bande et enfin compense les variations d'épaisseur en entrée du laminoir. Le second réalise la régulation de la planéité de la bande en sortie du laminoir. Le dernier compense les variations d'épaisseur causées par un défaut de concentricité des cylindres du laminoir.

Le système informatique du suivi en temps réel consiste en l'acquisition et l'analyse en temps réel des calculateurs de contrôle du laminoir [KAI 01]. Il permet, entre autre, de trouver rapidement l'origine des défaillances et de fournir une traçabilité du système aux opérateurs de conduite. Ce système informatique repose sur un calculateur monoprocesseur exécutant des tâches à activation périodique. Précisément chaque tâche τ_i est définie par sa durée d'exécution dans le pire cas C_i, sa date d'arrivée dans le système r_i, son délai critique D_i, et sa période T_i. Ainsi la $(k+1)^{ième}$ exécution de τ_i intervient dans la fenêtre temporelle : $[r_i + kT_i, r_i + D_i + kT_i]$. Trois tâches du logiciel effectuent l'acquisition des signaux en provenance du laminoir. Ces acquisitions doivent se faire au même rythme que les acquisitions des calculateurs de contrôle. La tâche τ_1 fait l'acquisition des signaux de régulation rapide de l'épaisseur de la feuille d'aluminium, τ_7 ceux de la régulation lente de l'épaisseur et τ_8 reçoit les signaux de régulation de la planéité de la feuille d'aluminium en cours de laminage. τ_2 effectue l'interprétation dynamique des données acquises par les trois tâches précédentes. Précisément elle détermine en fonction d'équations logiques les valeurs à enregistrer ou le déclenchement d'un enregistrement automatique lorsque le produit sort de ses tolérances. La tâche τ_3 exploite les signaux résultats en effectuant des calculs statistiques et a en charge le calcul de la FFT (Fast Fourrier Transform). Ces résultats sont destinés à l'affichage et le stockage dans la base de données du suivi. La tâche τ_9 génère un compte rendu périodique de fonctionnement ; τ_{10} gère l'affichage temps réel ; τ_4,τ_5 et τ_6 enregistrent dans la base de données les signaux aux rythmes de leur acquisition. Les tâches d'initialisation et de terminaison ne sont pas décrites, car elles n'interviendront pas dans l'étude qui suit. Le tableau 1 présente les caractéristiques des tâches (légèrement adaptées par rapport au cas réel pour mettre en évidence la méthode présentée ci-après) et la figure 2 décrit le graphe de précédence généralisée avec les rapports de périodicité entre les tâches. Les étiquettes n/m est le rapport de proportionnalité des périodes des tâches en précédence. Une définition précise des contraintes de précédence généralisée sera donnée dans le paragraphe 2.

Les relations de précédence entre tâches avec des fréquences différentes d'exécu-

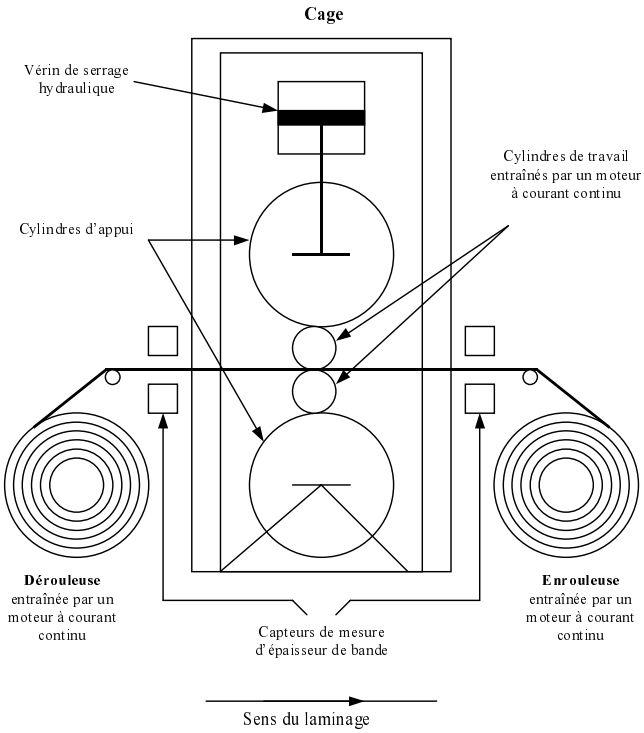

Figure 1. *Schéma de principe d'un laminoir industriel*

tion ont été étudiées principalement dans deux types d'applications :

– l'ordonnancement cyclique de tâches : les problèmes d'ordonnancement cyclique permettent d'étudier l'exécution de tâches à itérations dépendantes pouvant intervenir dans les systèmes informatiques [MUN 96] ou dans les systèmes de production [MUN 93, NAK 99]. L'objectif est de calculer la périodicité des activations des tâches, alors que dans notre cas ces périodes sont des données du problème.

– l'ordonnancement de flot de données (dataflow) : les tâches communiquent par l'intermédiaire de mémoires tampons. Ce type d'application se rencontre notamment en traitement du signal [JEF 99]. Mais les auteurs considèrent généralement un nombre minimum de données devant être présentes dans la mémoire tampon pour déclencher la tâche suivante dans le graphe de flot de données. Vérifier l'ordonnançabilité des tâches est alors $\mathcal{NP}$-Difficile au sens fort [BAR 97].

La validation du logiciel de suivi revient à contrôler que l'ensemble des tâches respecteront leurs échéances, c'est-à-dire que le logiciel pourra suivre en temps réel le fonctionnement du procédé. Ce problème est fondamental dans la conception d'un

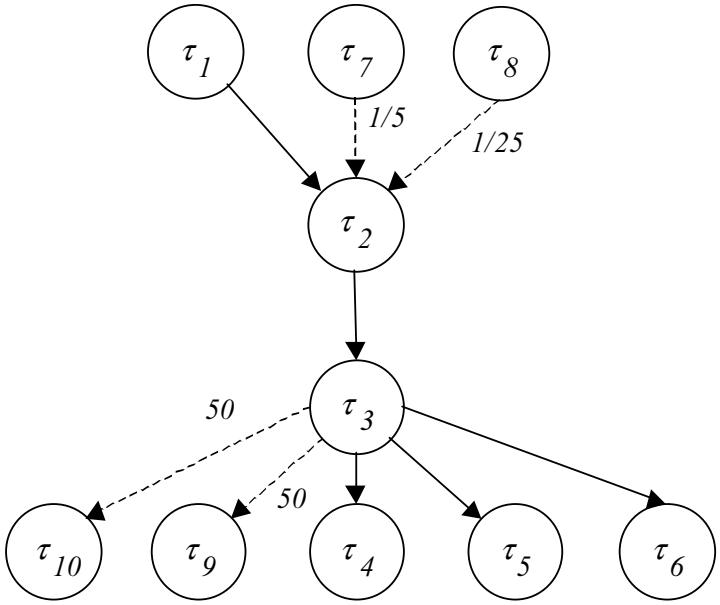

Figure 2. *Relations de précédence entre les tâches*

tâches	$T_i/250$ (ms)	$C_i/250$ (ms)	$D_i/250$ (ms)
τ_1	16	1	4
τ_2	16	2	16
τ_3	16	1	16
τ_4	16	1	16
τ_5	16	1	16
τ_6	16	2	16
τ_7	80	3	40
τ_8	400	5	200
τ_9	800	4	800
τ_{10}	800	4	800

Tableau 1. *Caractéristiques des tâches*

tel système [COT 00, STA 88]. De nombreuses techniques existent pour valider des tâches indépendantes [AUD 99, BUT 97, COT 00, STA 98]. La particularité de l'application considérée est que les relations de précédence sont complexes puisque les tâches reliées ne travaillent pas à la même cadence. A notre connaissance, aucune méthode de validation de système temps réel n'a été proposée dans la littérature pour ce type de contrainte de précédence.

La méthode de validation que nous proposons va dans un premier temps transformer les contraintes de précédence complexe en contraintes de précédence simple. Nous montrerons sous quelles conditions cette opération est réalisable. La configuration de tâches est ensuite transformée en un ensemble de tâches indépendantes. Cette dernière configuration est enfin utilisée pour valider l'application. Le paragraphe 2 formalise les relations de précédence généralisée. Le paragraphe 3 présente une méthode de transformation fondée sur le dépliage du graphe de précédence. Le paragraphe 4 présente la transformation des paramètres des tâches pour obtenir une configuration de tâches indépendantes, utilisée pour valider le logiciel de suivi temps réel.

2. Précédences généralisées

Deux tâches périodiques τ_i et τ_j sont soumises à une contrainte de précédence simple si toute exécution de τ_j est précédée par une exécution de τ_i. Les relations entre les tâches définissent un ordre partiel sur l'ensemble des tâches. Cet ordre partiel est souvent représenté par un graphe de précédence, où les sommets représentent les tâches et les arcs représentent les contraintes de précédence. Le graphe de précédence est nécessairement sans circuit puisque sinon le début d'exécution d'une tâche dépend de sa propre terminaison. Ceci revient à dire que les tâches appartenant à un circuit voient leurs dates de début d'exécution différées indéfiniment.

Comme l'illustre la figure 2, les tâches effectuant le suivi temps réel du laminoir sont soumises à des relations de précédence bien que certaines tâches ne s'exécutent pas en suivant la même période. Cette relation de précédence sera alors dite généralisée dans la suite. Ces relations de précédence sont simples à mettre en oeuvre en utilisant des sémaphores. Il suffit de considérer une tâche périodique qui se synchronise avec une autre toutes les n activations. Ceci peut être simplement mis en œuvre en plaçant une primitive de synchronisation dans une branche *if then else* de l'algorithme de la tâche. Nous formalisons maintenant les contraintes de précédence généralisée. Notons que cette formalisation est indépendante des dates de réveils des tâches et des périodes des tâches.

Définition 1 *Soit $B_t(i)$ (resp. $E_t(i)$) le nombre de débuts (resp. de fins) d'exécution de τ_i à la date t. Une contrainte de précédence généralisée entre les tâches τ_i et τ_j, de périodes respectives T_i et T_j , est définie par :*

$$\forall t \in \mathbb{N} \qquad E_t(i) \times T_i \geq B_t(j) \times T_j \qquad [1]$$

Nous noterons dans la suite que τ_i précède τ_j par : $\tau_i \prec \tau_j$.

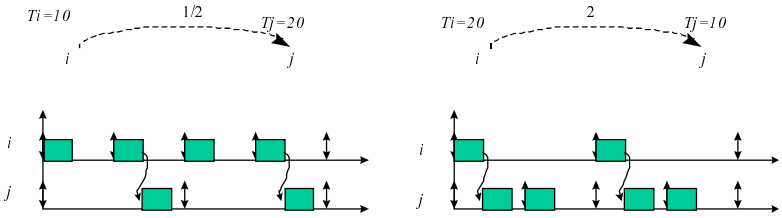

Figure 3. *Diagrammes de Gantt de tâches soumises à la précédence généralisée :* $\tau_i \prec \tau_j$

Le poids d'une précédence généralisée est le rapport des périodes des tâches en relation de précédence. Chaque arc d'un graphe de précédence généralisé sera étiqueté par son poids (les poids unitaires seront omis).

Définition 2 *Soit un arc d'un graphe de précédence,* $p = (\tau_i,\tau_j)$ *le poids de l'arc* p*, noté* $H(p)$*, est :* $H(p) = \frac{T_i}{T_j}$*. Par extension, le poids d'un chemin* ρ *dans un graphe de précédence généralisée* G *est défini par le produit des poids des arcs qui le compose :*

$$H(\rho) = \prod_{p \in \rho} H(p) \qquad [2]$$

La relation de précédence classique $(T_i = T_j)$ simplifie la relation de la définition 1 qui s'écrit alors $E_t(i) \geq B_t(j)$ (i.e. précédence généralisée de poids 1). En conséquence chaque requête de la tâche τ_j est précédée par une requête de la tâche τ_i. La figure 3 donne des exemples de tâches soumises à des contraintes de précédence généralisée. Une double flèche symbolise l'échéance d'une exécution et l'arrivée d'une nouvelle instance au même instant (tâche à échéance sur requête).

3. Dépliage du graphe de précédence

Nous proposons une méthode de validation permettant depuis une configuration de tâches soumises à des contraintes de précédence, de construire une configuration équivalente de tâches indépendantes, en deux étapes. La première étape consiste en le dépliage du graphe de précédence généralisée en un graphe de précédence simple.

Précisément, les contraintes de précédence généralisée peuvent être modélisées par des familles de contraintes de précédence simple. Pour cela une tâche τ_i est modélisée par n_i duplicata. Le $k^{ième}$ duplicata de la tâche τ_i sera noté τ_i^k, $k \in \{1 \ldots n_i\}$. La $n^{ième}$ requête du duplicata τ_i^k représente la $((n-1)n_i + k)^{ième}$ requête de la tâche originelle τ_i.

Les méthodes de dépliage sont utilisées dans de nombreux domaines [MUN 93, MUN 96, NAK 99]. Le travail qui suit s'inspire de ces méthodes, mais le graphe consi-

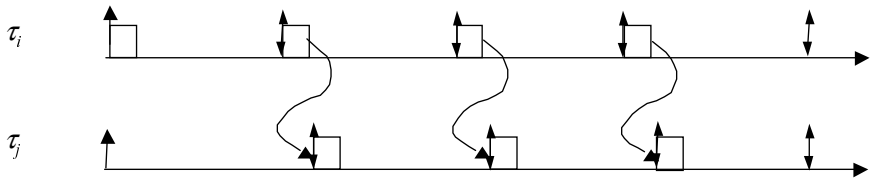

Figure 4. *Diagrammes de Gantt de tâches soumises à la précédence généralisée : $\tau_i \prec \tau_j$. Cet ordonnancement est répété à l'infini*

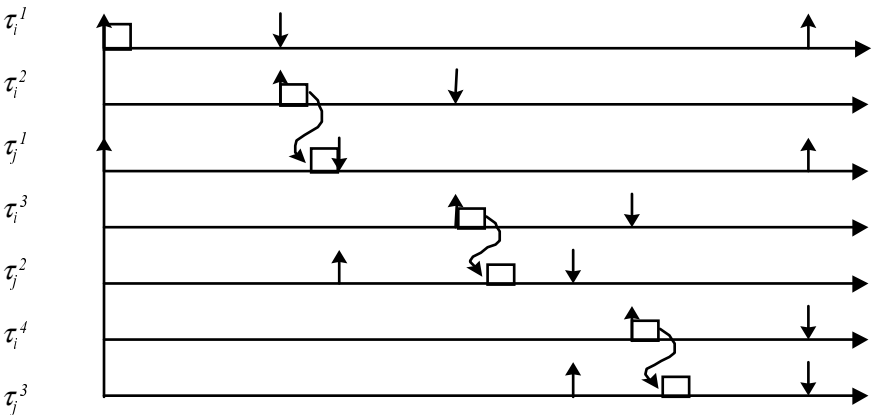

Figure 5. *Configuration après duplication et dépliage du graphe de précédence généralisée. Cet ordonnancement est répété à l'infini. (flèche vers le bas : échéance d'une tâche ; flèche vers le haut : réveil d'une tâche)*

déré a une structure très différente de ceux considérés dans la littérature (acyclique versus fortement connexe). Cette différence dans la structure du graphe engendre des résultats analytiques différents.

Nous illustrons tout d'abord le principe sur un exemple simple. Considérons une contrainte de précédence généralisée entre une tâche τ_i de période 30 ms, et une tâche τ_j d'une période de 40 ms. Le diagramme de Gantt de l'exécution de ces deux tâches est donné figure 4.

Trois exécutions de τ_i sur quatre sont en précédence avec la tâche τ_j. Il est possible de modéliser cette contrainte de précédence généralisée en dupliquant quatre fois τ_i et trois fois τ_j, et en créant des précédences simples entre les duplicata des tâches. Il est clair que l'exécution donnée figure 5 est alors équivalente à celle présentée figure 4. Dans la figure 5, tous les duplicata ont une période de 120 unités de temps. Cet ordonnancement partiel peut donc être répété indéfiniment.

La première étape consiste à calculer le nombre nécessaire de duplicata (théorème 1). Nous ne considérons à ce stade qu'une contrainte de précédence généralisée. Le cas d'un graphe quelconque est ensuite présenté.

Théorème 1 *Soit n_i et n_j le nombre de duplicata de τ_i et τ_j, la contrainte de précédence généralisée $\tau_i \prec \tau_j$ peut être modélisée par un ensemble fini de contraintes de précédence simple entre les duplicata si, et seulement si :*

$$n_i \times T_i - n_j \times T_j = 0 \qquad n_i \in \mathbb{N}^*, n_j \in \mathbb{N}^* \qquad [3]$$

Preuve : [MUN 93] montre, en considérant deux duplicata τ_i^k et τ_j^l respectivement des tâches τ_i et τ_j, que les contraintes de précédence entre les duplicata sont respectées si, et seulement si :

– la $l^{ième}$ instance de τ_j ne peut pas débuter avant la $k^{ième}$ instance de τ_i:

$$kT_i - lT_j \geq 0$$

– la $(l-1)^{ième}$ instance de τ_j peut débuter après la $k^{ième}$ instance de τ_i:

$$(k-1)T_i - (l-1)T_j \geq 0$$

– la $l^{ième}$ instance de τ_j ne peut pas débuter après la $(k-1)^{ième}$ instance de τ_i:

$$(k-1)T_i - lT_j < 0$$

De plus ces contraintes doivent être respectées pour toutes les activations successives des duplicata. Si l'on considère la $n^{ième}$ exécutions des duplicata τ_i^k et τ_j^l, depuis les inégalités précédentes, il est simple de vérifier que $\tau_i^k \prec \tau_j^l$ si, et seulement si :

$$T_i > ((n-1)n_i + k)T_i - ((n-1)n_j + l)T_j \geq \max(T_i - T_j; 0) \qquad [4]$$

L'expression 4 sera vérifiée pour tout $n \in \mathbb{N}$ si, et seulement si (3) admet une solution.

Théorème 2 *Soit $\tau_i \prec \tau_j$ une contrainte de précédence généralisée, n_i et n_j le nombre de duplicata des tâches τ_i et τ_j. Alors elle sera représentée de façon équivalente par :*

– *n_i contraintes de précédence simple si $T_i > T_j$:*

$$\forall k \in \{1 \ldots n_i\} \qquad \tau_i^k \prec \tau_j^{a_k}, a_k = \left\lfloor \frac{(k-1)T_i}{T_j} \right\rfloor + 1 \qquad [5]$$

– *n_j contraintes de précédence simple sinon :*

$$\forall k \in \{1 \ldots n_j\} \qquad \tau_i^{b_k} \prec \tau_j^k, b_k = \left\lceil \frac{kT_i}{T_j} \right\rceil \qquad [6]$$

Preuve : Soit τ_i^k et τ_j^l deux duplicata des tâches τ_i et τ_j. Nous considérons deux cas:

– Si $T_i > T_j$, nous devons calculer le numéro a_k du duplicata de τ_j (i.e. $\tau_j^{a_k}$) qui est précédé par le $k^{ième}$ duplicata de τ_i. Dans la preuve du théorème 1 il est indiqué que $\tau_i^k \prec \tau_j^{a_k}$ si, et seulement si,

$$T_i > ((n-1)n_i + k)T_i - ((n-1)n_j + l)T_j \geq T_i - T_j$$

Puisque $n_i T_i = n_j T_j$, alors l'expression précédente devient : $T_i > kT_i - a_k T_j \geq 0$. L'unique solution entière est :

$$a_k = \left\lfloor \frac{(k-1)T_i}{T_j} \right\rfloor + 1$$

– Si $T_i \leq T_j$, de la même façon nous calculons le numéro b_k du duplicata de τ_i qui précède le $k^{ième}$ duplicata de τ_j. Par le théorème 1 ceci sera vérifié si, et seulement si :

$$T_i > ((n-1)n_i + k)T_i - ((n-1)n_j + l)T_j \geq 0$$

En utilisant le même raisonnement que dans le cas précédent, nous obtenons que l'unique solution entière possible est :

$$b_k = \left\lceil \frac{kT_i}{T_j} \right\rceil$$

Sur l'exemple de la figure 4 : la solution minimale de l'équation (1) est $n_i = 4$ et $n_j = 3$. On remarque que $T_i \leq T_j$, la seconde partie du théorème 2 est donc utilisée pour déplier la contrainte de précédence généralisée. Puisque $k \in \{1 \ldots n_j\}$, les calculs effectués sont :

$$k = 1, b_1 = \left\lceil \frac{1 \times 40}{30} \right\rceil = 2 \quad \Rightarrow \quad \tau_i^2 \prec \tau_j^1$$

$$k = 2, b_2 = \left\lceil \frac{2 \times 40}{30} \right\rceil = 3 \quad \Rightarrow \quad \tau_i^3 \prec \tau_j^2$$

$$k = 3, b_3 = \left\lceil \frac{3 \times 40}{30} \right\rceil = 4 \quad \Rightarrow \quad \tau_i^4 \prec \tau_j^3$$

Le graphe de précédence déplié correspondant est donné figure 6.

Pour calculer le nombre de duplicata, la relation (3) doit être vérifiée pour tous les arcs du graphe de précédence généralisée. Le système d'équations à résoudre est :

$$\Sigma(G) : \qquad n_i T_i = n_j T_j \qquad \tau_i \prec \tau_j, n_i \in \mathbb{N}^*, n_j \in \mathbb{N}^*$$

Le théorème 3 va montrer que tout graphe sans circuit de précédence généralisée peut être déplié en un graphe de précédence simple. Il fournit de plus le nombre minimal de duplicata pour chaque tâches. Sa preuve est basée sur le fait que le nombre

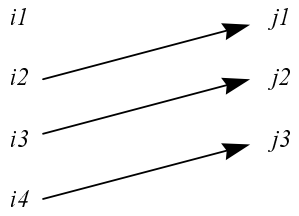

Figure 6. *Graphe déplié d'une contrainte de précédence généralisée*

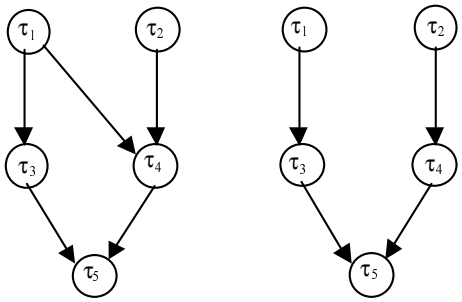

Figure 7. *Graphe G et arbre maximal inclus dans G*

de duplicata peut être calculé en ne considérant qu'un arbre maximal inclus dans le graphe de précédence généralisée G.

Nous illustrons ce fait sur le graphe G de la figure 7. Dans G, il existe deux chemins reliant τ_1 à τ_5. Il est simple de montrer que l'arc (τ_1,τ_4), correspondant à l'équation suivante de $\Sigma(G)$: $n_1T_1 = n_4T_4$, est une combinaison linéaire des autres équations :

$$\begin{aligned} n_1T_1 &= n_3T_3 \\ n_2T_2 &= n_4T_4 \\ n_3T_3 &= n_5T_5 \\ n_4T_4 &= n_5T_5 \end{aligned}$$

Cette équation est donc inutile pour résoudre le système $\Sigma(G)$. En répétant ce principe de suppression des arcs inutiles dans le graphe de précédence G, nous obtenons un arbre maximal inclus dans G. Sur l'exemple de la figure 7 après suppression l'arc (τ_1,τ_4) dans G, le graphe résultant est un arbre maximal. L'ensemble des équations correspondantes permet donc de calculer le nombre minimum de duplicata.

Nous montrons tout d'abord une propriété sur les chemins entre deux sommets dans le graphe de précédence généralisée.

La notion de poids va permettre de caractériser une propriété importante des chemins entre deux sommets d'un graphe de précédence généralisée.

Lemme 1 *Soit τ_i et τ_j deux sommets d'un graphe de précédence généralisée G sans circuit. Tout chemin ρ de τ_i à τ_j vérifie : $H(\rho) = \frac{T_i}{T_j}$ (i.e. sont de même poids).*

Preuve : Soit $\rho = \tau_i e_1 e_2 \dots e_k \tau_j$ un chemin quelconque de τ_i à τ_j, passant par les sommets . On raisonne par récurrence sur la longueur k du chemin. Base : $k = 1$, on vérifie $\rho_1 = \tau_i \tau_j$ et donc $H(\rho_1) = \frac{T_i}{T_j}$
Induction : supposons la relation vérifiée pour un chemin de longueur k : $\rho = \tau_i e_1 e_2 \dots e_{k-1} \tau_j$. Un chemin de longueur $k+1$ peut s'écrire : $\rho_{k+1} = \tau_i e_1 e_2 \dots e_{k-1} e_k \tau_j$. Et par définition du poids d'un chemin :

$$H(\rho_{k+1}) = \frac{T_i}{T_{e_1}} \times \frac{T_{e_1}}{T_{e_2}} \times \dots \times \frac{T_{e_{k-1}}}{T_{e_k}} \times \frac{T_{e_k}}{T_j} = \frac{T_i}{T_j}$$

d'où le résultat.

Théorème 3 *Tout graphe de précédence généralisée sans circuit peut être déplié en un graphe de précédence simple et la solution minimum de $\Sigma(G)$ est donnée par :*

$$n_i = \frac{ppcm(T_1, \dots, T_n)}{T_i} \qquad 1 \leq i \leq n \qquad [7]$$

Preuve : Sans perte de généralité nous considérons un graphe de précédence généralisée G connexe. Si ce n'est pas le cas, chaque composante connexe sera dépliée séparément. Soit n le nombre de sommets du graphe G, et m le nombre d'arcs, puisque G est connexe, on vérifie $m \geq n-1$. Considérons deux chemins d'un sommet τ_i à un autre sommet τ_j. Par le lemme 1 ils sont de même poids. Il existe en conséquence une équation dans $\Sigma(G)$ qui est une combinaison linéaire des autres équations. La suppression de cette équation ne modifie pas le système $\Sigma(G)$. Ceci revient à supprimer l'arc correspondant dans le graphe G. Si l'on répète cette opération pour tous les chemins entre deux sommets quelconques τ_i et τ_j, on supprime exactement $m - n + 1$ arcs. Il restera donc n sommets et $(n-1)$ arcs, et un arbre maximal $\overline{G}$ aura été construit. Les arcs supprimés étant quelconques sur le chemin de τ_i à τ_j, il résulte que $\Sigma(G)$ est équivalent au système $\Sigma(\overline{G})$ associé à un arbre maximal $\overline{G}$ quelconque inclus dans G. Soit $X = (x_i)$ un vecteur d'entiers solutions du système $\Sigma(\overline{G})$, alors λX,$\lambda \in \mathbb{N}$, est aussi un vecteur solution puisque $\Sigma(\overline{G})$ comporte n variables et $(n-1)$ équations. La solution minimale de $\Sigma(\overline{G})$ est donc donnée par :

$$n_i = \frac{ppcm(T_1, \dots, T_n)}{T_i} \qquad 1 \leq i \leq n$$

Ces valeurs seront aussi solutions de $\Sigma(G)$. Enfin, le théorème 2 permet de construire le graphe déplié, ce qui permet de conclure que G est dépliable.

De façon à terminer la transformation, nous devons définir les paramètres temporels des tâches créées par le dépliage (i.e. les duplicata). Clairement les duplicata ont les mêmes durées d'exécution et échéances que la tâche dont ils sont issus. Seules les dates d'activations et les périodes des duplicata sont à calculer de façon à respecter l'équivalence entre la configuration de tâches originelles et leurs duplicata.

Théorème 4 *Soit une configuration de tâches* $S = \{\tau_i(r_i,C_i,D_i,T_i),1 \leq i \leq n\}$, $\prec$ *un ordre partiel sur* S *définissant les précédences généralisées,* n_i *le nombre de duplicata des tâches* τ_i*, alors la configuration de tâches :*

$$S^* = \{\tau_i^k(r_i^k,C_i^k,D_i^k,T_i^k),1 \leq i \leq n,1 \leq k \leq n_i\} \quad [8]$$

est construite telle que :

$$\forall k \in \{1 \ldots n_i\} \begin{cases} r_i^k = r_i + (k-1)T_i \\ T_i^k = n_i T_i \\ D_i^k = D_i \\ C_i^k = C_i \end{cases} \quad [9]$$

alors S *est ordonnançable si, et seulement si,* S^* *est ordonnançable.*

Ce résultat découle directement des résultats précédents. La transformation montre que la configuration construite est équivalente à la configuration initiale. Remarquons que le nombre de duplicata peut être très grand (exponentiel) puisque leur nombre est fonction du ppcm des périodes des tâches originelles.

Dans [RIC 99] nous montrons que la complexité de l'algorithme de dépliage d'un graphe G est :

$$O\left(|Q|\sum_{i\in T} n_i + |P|\sum_{(i,j)\in P}\min(n_i,n_j)\right) \quad [10]$$

où Q est l'ensemble des sommets (i.e. des tâches) et P l'ensemble des arcs (i.e. des précédences généralisées) du graphe de précédence. Ainsi le dépliage conduit à un traitement pseudo-polynomial en supposant que les valeurs n_i sont des entrées du problème, c'est-à-dire qu'elles ont été préalablement calculées. Notons toutefois que la $\mathcal{NP}$-Complétude du problème d'ordonnancement de tâches périodiques soumises à des contraintes de précédence généralisée est un problème ouvert.

4. Validation de la configuration obtenue

Dans le paragraphe précédent nous réduisons la validation d'une configuration de tâches soumises à des précédences généralisées, à une autre ne contenant que

des précédences simples. Le même principe va être utilisé pour traiter les précédences simples. Depuis la configuration de tâches avec des précédences simples nous construisons une configuration équivalente de tâches indépendantes. La transformation consiste à modifier les paramètres r_i et D_i des tâches de telle façon que l'algorithme d'ordonnancement en-ligne respecte les précédences sans avoir recours à un mécanisme de synchronisation. Nous rappelons les résultats connus pour l'algorithme d'ordonnancement Earliest Deadline First (ED).

La transformation s'appuie sur le fait que l'algorithme d'ordonnancement est basé sur l'échéance d_i. Les échéances sont modifiées de telle façon qu'une tâche ait toujours un d_i inférieur à celui de ses successeurs [BLA 76, CHE 90]. Une tâche ne peut débuter son exécution que si tous ses prédécesseurs se sont terminés. Dans la suite, l'astérisque désigne un paramètre modifié. Les deux règles sont les suivantes :

– une tâche ne sera activable que si tous ses prédécesseurs ont terminé leur exécution, c'est-à-dire que la date de début de cette tâche doit être supérieure à toutes les dates de début de ces prédécesseurs immédiats augmentées de leur durée d'exécution :

$$r_i^* = \max(r_i, \max_{\tau_j \in prec(\tau_i)}(r_j^* + C_j)) \qquad [11]$$

– l'échéance d'une tâche doit être inférieure à toutes les échéances de ses successeurs immédiats diminuées de leur temps d'exécution :

$$d_i^* = \min(d_i, \min_{\tau_j \in succ(\tau_i)}(d_j^* - C_j)) \qquad [12]$$

La configuration obtenue ne comporte que des tâches indépendantes. Elle sera ordonnançable si, et seulement si, la configuration de départ (avec des précédences simples) est ordonnançable [CHE 90]. Il est important de noter que l'adaptation des dates d'échéances implique nécessairement l'adaptation des délais critiques :

$$D_i^* = d_i^* - r_i^* \qquad [13]$$

[CHE 90] indique que la complexité algorithmique de la transformation des paramètres est $O(n^2)$, où n est le nombre de tâches. Nous améliorons maintenant ces deux bornes, tout en conservant l'équivalence des deux configurations. L'intérêt de cette amélioration est de rendre plus performant l'algorithme de validation. Le calcul de la nouvelle borne inférieure r_i^* repose sur l'examen plus précis de la charge processeur induite par les tâches en précédant une autre. Précisément, la charge des prédecesseurs de τ_i, devant s'exécuter avant son début, peut être calculée après l'arrivée de chaque prédécesseur. Considérons un prédécesseur quelconque τ_j de τ_i, alors τ_i ne peut débuter son exécution avant la date définie par : la durée de τ_j plus la durée des prédécesseurs de τ_i arrivant après τ_j. On obtient donc :

$$r_i^* \geq r_j^* + \sum_{\substack{\tau_k \in prec(\tau_i) \\ r_k^* \geq r_j^*}} C_k \qquad \forall \tau_j \in prec(\tau_i) \qquad [14]$$

Une borne inférieure de r_i est donc donnée par :

$$r_i^* = max \left(r_i ; \max_{\tau_j \in prec(\tau_i)} \left(r_j^* + \sum_{\substack{\tau_k \in prec(\tau_i) \\ r_k^* \geq r_j^*}} C_k \right) \right) \quad [15]$$

On procède de la même façon pour améliorer la borne supérieure de l'échéance d'une tâche. La charge processeur induite par les successeurs τ_j de τ_i se fait en examinant toutes leurs échéances d_j. Considérons un successeur quelconque τ_j de τ_i, afin que τ_j et tous les successeurs τ_k de τ_i, devant se terminer avant τ_j, puissent s'exécuter il faut que l'échéance d_i soit au minimum :

$$d_i^* \leq d_j^* - \sum_{\substack{\tau_k \in succ(\tau_i) \\ d_k^* \leq d_j^*}} C_k \qquad \forall \tau_j \in succ(\tau_i) \quad [16]$$

Une borne supérieure de l'échéance d_i est donc donnée par :

$$d_i^* = min \left(d_i ; \min_{\tau_j \in succ(\tau_i)} \left(d_j^* - \sum_{\substack{\tau_k \in succ(\tau_i) \\ d_k^* \leq d_j^*}} C_k \right) \right) \quad [17]$$

Il est simple de montrer que le calcul des ces nouvelles bornes peut être fait en $O(n^2)$ en traitant les requêtes dans leur ordre non-décroissant d'arrivée et en considérant l'ordre non-croissant des échéances (i.e. ces tris sont effectués en $O(nlogn)$ avant le traitement de la modification des paramètres).

Après adaptation des paramètres, les résultats connus sur la validation de tâches indépendantes peuvent être appliqués. Nous pouvons citer par exemple :

– la simulation de ED sur l'intervalle $[0,max(r_i) + 2.ppcm(T_1,...,T_n)]$

– l'utilisation de conditions analytiques.

Nous avons implémenté la méthode complète de façon à valider le logiciel de suivi temps réel du laminoir. Pour la validation des tâches indépendantes nous avons utilisé la simulation de ED. Le ppcm des périodes de la configuration de tâches du tableau 1 est 800. Ceci correspond à la borne classique de Leung et Merill [LEU 84]. Notons que cette borne a été améliorée dans [GRO 00]. La périodicité de l'ordonnancement est montrée égale à $[0; tc + ppcm(T_1,...T_n)]$, où tc est la date du dernier temps creux acyclique (i.e. fin du régime transitoire). Par contre la détermination de la date tc n'est

Tâches	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8	τ_9	τ_{10}
Nombre de duplicata	50	50	50	50	50	50	10	2	1	1

Tableau 2. *Nombre de duplicata pour le logiciel de suivi du laminoir*

pas simple et complexifie la mise en œuvre. C'est uniquement pour cette raison que nous utilisons la borne de [LEU 84] qui est simple à calculer. Le nombre de duplicata pour chaque tâche est le résultat de la division du ppcm par la période de la tâche. Le tableau 2 donne le nombre de duplicata pour les tâches assurant le suivi temps réel du laminoir.

Les tests, menés sur une Sun Sparc Station 2, montrent que la configuration est ordonnançable. Pour montrer l'efficacité de la méthode complète nous indiquons les durées opératoires associées à chacune des étapes :

– dépliage et calcul des paramètres des duplicata : 0,04 s
– adaptation des paramètres r_i et d_i : 0,23 s
– simulation de l'ordonnancement par ED : 0,56 s

ceci montre que la méthode du dépliage peut être utilisée pour valider une application temps réel industrielle de taille importante.

5. Conclusion

Nous avons présenté le problème de la validation d'un logiciel de suivi temps réel d'un procédé industriel : un laminoir d'aluminium. La particularité de cette application est la présence de contraintes de précédence complexe entre les tâches périodiques effectuant le contrôle qualité de la production de feuilles d'aluminium. A notre connaissance ce type de contrainte n'a jamais été étudié dans le cadre de l'ordonnancement temps réel.

Pour valider l'application nous avons utilisé une méthode en trois étapes. Tout d'abord, le graphe de précédence généralisée est déplié en un graphe de précédence simple. La configuration de tâches ainsi obtenue est montrée équivalente à la configuration initiale. Les paramètres des tâches en relation de précédence simple sont ensuite adaptés pour rendre les tâches indépendantes. Sur ce point nous avons amélioré les bornes de [BLA 76, CHE 90]. La configuration de tâches résultantes est équivalente à la première. Enfin, la dernière étape consiste en l'utilisation des techniques classiques d'analyse d'ordonnançabilité de tâches indépendantes s'exécutant sur un ordinateur monoprocesseur.

Bien que la méthode soit théoriquement exponentielle, puisque la complexité de l'algorithme de dépliage est fonction du ppcm des périodes des tâches, l'expérimentation menée dans l'application sur le laminoir ont permis de valider le logiciel très efficacement. De plus une mise en œuvre plus performante peut être envisagée en utilisant les résultats sur la cyclicité des ordonnancements présentés dans [GRO 00].

Par ailleurs, dans [RIC 99], nous avons étudié comment éviter, dans certains cas, le dépliage du graphe. Dans le cas particulier où les tâches en précédence ont des périodes proportionnelles, nous avons montré que le principe de la modification des paramètres peut être directement appliqué sans dépliage du graphe de précédence. Mais dans le cas général, aucune condition nécessaire et suffisante ne peut être établie en considérant les algorithmes classiques d'ordonnancement. Le dépliage est alors impérativement nécessaire. Nous avons de plus utilisé la méthode de dépliage de graphe de précédence à l'ordonnancement de tâches communicantes, avec des fréquences différentes d'activation, dans un système distribué [RIC 01]. Seule la seconde étape de la méthode, qui porte sur la vérification de l'ordonnançabilité du système, a été modifiée. Ceci montre l'intérêt de l'approche en deux étapes : premièrement dépliage du graphe de précédence ; deuxièmement vérification de l'ordonnançabilité des duplicata.

6. Bibliographie

[AUD 99] AUDSLEY N. C., BURNS A., DAVIS R., TINDELL K., WELLINGS A., « Fixed priority pre-emptive scheduling: an historical perspective », *Journal of Real-Time Systems*, vol. 8, 199, p. 173-198.

[BAR 97] BARUAH S., GODDARD S., JEFFAY K., « Feasibility concerns in PGM graphs with bounded buffer », *Proc: IEEE Conf. Engineering of Complex Computer Systems*, 1997, p. 130-139.

[BLA 76] BLAZEWICZ J., « Scheduling dependent Task with Different Arrival Times to Meet Deadlines », *Modeling performance Evaluation Computer Systems , E. Gelenbe, H. Belner (ed.),* , 1976, p. 57-65.

[BUT 97] BUTTAZZO G., *Hard Real-Time Computing Systems: Predictable Scheduling, Algorithms and Applications*, Kluwer Academic Publishers, 1997.

[CHE 90] CHETTO H., SILLY M., BOUCHENTOUF T., « Dynamic Scheduling of Real-Time Task under Precedence Constraints », *Journal of Real Time Systems*, vol. 2, 1990, p. 181-194.

[COT 00] COTTET F., DELACROIX J., KAISER C., MAMMERI Z., *Ordonnancement temps-réel*, Hermès Sciences, 2000.

[GRO 00] GROLLEAU E., CHOQUET-GENIET A., « Cyclicité des ordonnancements de systèmes de tâches périodiques différées », *Proc. Conf. Real-Time Systems*, 2000, p. 245-229.

[JEF 99] JEFFAY K., GODDARD S., « A theory of Rate-Based Execution », *Proc: IEEE Real-Time System Symposium*, 1999.

[KAI 01] KAISER C., « Description et critique d'un système temps réel pour le suivi d'un laminoir. Robustesse et potentialité d'évolutivité », *Technique et Science Informatiques*, vol. 20, n° 2, 2001, p. 179-212.

[LEU 84] LEUNG J., MERILL M., « A note on preemptive scheduling of periodic real-time tasks », *Information Processing Letters*, vol. 11, n° 3, 1984, p. 115-118.

[MUN 93] MUNIER A., « Régime asymptotique optimal d'un graphe d'événements temporisés généralisés : application à un problème d'assemblage », *RAIRO APII*, vol. 27, n° 5, 1993, p. 487-513.

[MUN 96] MUNIER A., « The basic cyclic scheduling problem with linear precedence constraints », *Discrete Applied Mathematics*, vol. 64, 1996, p. 219-238.

[NAK 99] NAKAMURA N., SILVA M., « Cycle Time Computation in Deterministically Timed Weighted Marked Graphs », *Proc: IEEE Conf. on Emerging technologies and Factory Automation*, 1999, p. 1037-1046.

[RIC 99] RICHARD P., COTTET F., *Ordonnancement temps réel des contraintes de précédence généralisée*, Rapport de Recherche LISI/ENSMA 9901, 1999.

[RIC 01] RICHARD P., COTTET F., RICHARD M., « On-line scheduling of Real-Time Distributed Computers with complex communication constraints », *Proc: IEEE Conf. on Enginnering of Complex Computer Systems*, 2001, p. 26-34.

[STA 88] STANKOVIC J., « Misconceptions About Real-Time Computing: A Serious Problem For Next Generation Systems », *IEEE Computer Magazine*, vol. 21, n° 10, 1988.

[STA 98] STANKOVIC J. A., SPURI M., RAMARITHAM K., BUTTAZZO G., *Deadline Scheduling For Real-Time Systems: EDF and Related Algorithms*, Kluwer Academic Publishers, 1998.