HAL
open science

Synthèse optimale de contrôleurs discrets et systèmes répartis tolérants aux fautes

Emil Dumitrescu, Alain Girault, Hervé Marchand, Eric Rutten

To cite this version:

Emil Dumitrescu, Alain Girault, Hervé Marchand, Eric Rutten. Synthèse optimale de contrôleurs discrets et systèmes répartis tolérants aux fautes. 6ème Colloque Francophone sur la Modélisation des Systèmes Réactifs, Oct 2007, Lyon, France. pp.71-86. inria-00424279

HAL Id: inria-00424279

https://inria.hal.science/inria-00424279v1

Submitted on 14 Oct 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Synthèse optimale de contrôleurs discrets et systèmes répartis tolérants aux fautes

Emil Dumitrescu* — **Alain Girault**** — **Hervé Marchand***** — **Éric Rutten****

* *Laboratoire AMPERE, INSA-Lyon,* Emil.Dumitrescu@insa-lyon.fr

** *INRIA, POP ART, Grenoble,* Alain.Girault@inria.fr, Eric.Rutten@inria.fr

*** *INRIA, VERTECS, Rennes,* Herve.Marchand@irisa.fr

RÉSUMÉ. *Les systèmes embarqués requièrent des méthodes de conception sûres fondées sur des méthodes formelles, ainsi qu'une exécution sûre fondée sur des techniques de tolérance aux fautes. Nous proposons une méthode de conception sûre pour des systèmes à l'exécution sûre : elle utilise la synthèse de contrôleurs discrets (SCD) pour générer un système tolérant aux fautes, reconfigurable et correct. Les propriétés assurées concernent l'exécution consistante, l'assurance de la fonctionnalité (quelles que soient les fautes, sous une certaine hypothèse), et plusieurs optimisations, notamment sur le temps des exécutions passant par des points de reprise. Nous proposons un algorithme de SCD optimale sur des chemins bornés. Nous proposons des motifs de modèles pour des tâches, des processeurs à mémoire répartie et des motifs de fautes potentiels. Nous utilisons des modèles synchrones, l'outil de SCD symbolique Sigali et les Automates de Modes.*

ABSTRACT. *Embedded systems require safe design methods based on formal methods, as well as safe execution based on fault-tolerance techniques. We propose a safe design method for safe execution systems: it uses optimal discrete controller synthesis (DCS) to generate a correct reconfiguring fault-tolerant system. The properties enforced concern consistent execution, functionality fulfillment (whatever the faults, under some failure hypothesis), and several optimizations, particularly on the execution time when going through checkpoints. We propose an algorithm for optimal DCS on bounded paths. We propose model patterns for a set of periodic tasks with checkpoints, a set of distributed, heterogeneous and fail-silent processors, and an environment model that expresses the potential fault patterns. We use synchronous models, the Sigali symbolic DCS tool and Mode Automata.*

MOTS-CLÉS : *Systèmes temps-réel, conception sûre, tolérance aux fautes, synthèse optimale de contrôleurs discrets, programmation synchrone.*

KEYWORDS: *Real-time systems, safe design, fault tolerance, optimal discrete control synthesis, synchronous programming.*

1. Motivation

1.1. *Systèmes embarqués à sûreté critique*

Les systèmes embarqués réactifs à sûreté critique comportent entre autres des systèmes de transport public, nucléaires ou liés à la santé. Les deux principales contraintes imposées à ces systèmes viennent de leur sûreté et de leur caractère temps-réel. Nous proposons l'utilisation de méthodes formelles pour la conception de ces systèmes, en garantissant par construction ces propriétés essentielles pour leur mise en œuvre.

Plus spécifiquement, nous voulons concevoir des systèmes réactifs multi-tâches et multi-processeurs. Les processeurs étant sujets à des défaillances, nous allons assurer que nos systèmes tolèrent les fautes (contrainte de sûreté critique). Nous garantirons aussi qu'ils réagissent dans la borne de l'empreinte mémoire possible, et de temps imparti (contrainte de temps-réel).

Notre proposition est d'utiliser la synthèse de contrôleurs discrets (SCD) pour obtenir automatiquement un système contrôlé qui garantisse par construction le respect de ses contraintes de tolérance aux fautes et de temps-réel.

1.2. *Formulation du problème*

Un système *réactif* doit réagir continûment à son environnement, à un rythme imposé par ce dernier. Beaucoup de systèmes embarqués sont réactifs. Par exemple, c'est le cas du système de contrôle de vol d'AIRBUS. Cette propriété impose une contrainte de temps-réel : l'environnement envoie des stimuli au système, auxquels celui-ci doit réagir avant les stimuli suivants. Quand le système est à sûreté critique, cette contrainte est *stricte*, et elle doit donc être garantie sur la mise en œuvre.

Notre approche adopte un niveau d'abstraction où un système est représenté par son comportement en termes de démarrage et reconfiguration de calculs de plus bas niveau [ALT 03]. Chacun de ces calculs est représenté par une *tâche périodique*, dont la période correspond au temps de réaction imposé par l'environnement. Chaque tâche est elle-même décomposée en plusieurs *phases* successives, séparées par des points de reprise. Une *reconfiguration* du système consiste à migrer une ou plusieurs tâches sur un autre processeur. Ceci doit être fait, en particulier, chaque fois qu'une défaillance de processeur a lieu. La tâche redémarre alors depuis son dernier point de reprise (*roll-back*).

Cette technique de tolérance aux fautes est *dynamique* puisque les migrations sont décidées à l'exécution. L'inconvénient est qu'en général il est très difficile de prouver que le système tolère les fautes requises tout en satisfaisant ses contraintes temps-réel, ceci quelles que soient les fautes. Par contraste, les techniques *statiques* consistent à ajouter de la redondance spatiale au système, en exécutant pour cela plusieurs copies de chaque tâche. L'inconvénient est naturellement un très grand sur-coût.

L'avantage majeur de l'utilisation de la SCD pour la tolérance aux fautes est que, si la SCD réussit, alors le système contrôlé résultant se reconfigurera *dynamiquement* sur l'occurrence de fautes (donc avec un faible sur-coût), tout en étant garanti *par*

construction satisfaire ses contraintes temps-réel quelles que soient les fautes. Toutefois, la procédure de SCD peut échouer ; en effet, comme elle est fondée sur une exploration exacte de l'espace d'état, cela signifie qu'il n'existe aucun contrôleur qui satisfasse à la fois les contraintes de temps-réel et de tolérance aux fautes. Cela signifie que le concepteur doit relâcher une contrainte (par ex. le nombre de fautes à tolérer ou le nombre de processeurs disponibles).

1.3. *Contributions*

Cet article se fonde sur des résultats précédents où la SCD était appliquée avec des objectifs d'invariance, de bornage de coûts et de SCD optimale *à un pas* [GIR 04]. Ici, nous avons une modèle de tâche *plus riche* avec des phases et des points de reprise, et nous appliquons la SCD sur des *chemins finis*. Pour ceci, nous proposons une variante de l'algorithme de SCD optimale classique basée sur la programmation dynamique (c.f. [BEL 57] et [MAL 04] pour plus de détails) de façon à gérer des chemins ayant des boucles infinies, comme il s'en trouve dans les systèmes réactifs.

Dans la suite, nous décrivons d'abord notre technique de SCD pour la tolérance aux fautes, et particulièrement la SCD optimale sur des chemins traitant des boucles d'attente rencontrées dans les systèmes réactifs. Puis nous décrivons notre modèle de systèmes répartis avec fautes, sous forme de motifs de modèles pour des tâches avec points de reprise, des processeurs hétérogènes à mémoire répartie et silencieux sur défaillance, ainsi qu'un modèle de l'environnement qui exprime les motifs de fautes potentiels. Ensuite, nous définissons la tolérance aux fautes en termes de propriétés sur notre modèle, qui sont utilisées comme objectifs de SCD et mènent à la dérivation automatique d'un contrôleur pour la tolérance aux fautes.

2. Modèles et algorithmes de synthèse

Cette section donne une brève description des modèles et des techniques de synthèse de contrôleurs que nous utilisons, et qui sont détaillés ailleurs [MAR 00a].

2.1. *Systèmes de transitions étiquetés (STE)*

Formellement, un STE est donné par un tuple $\langle \mathcal{Q}, q_0, Q_f, \mathcal{I}, \mathcal{O}, \delta \rangle$, où $\mathcal{Q}$ est un ensemble fini d'états, $q_0 \in \mathcal{Q}$ est l'état initial, $Q_f \subseteq \mathcal{Q}$ est l'ensemble des états finaux, $\mathcal{I}$ est un ensemble fini de signaux d'entrées (produits par l'environnement), $\mathcal{O}$ est un ensemble fini de signaux de sortie (envoyés par le système vers l'environnement) et δ est une relation de transition, i.e., une fonction de $\mathcal{Q} \times \mathcal{B}ool(\mathcal{I}) \times \mathcal{O}^*$ dans $\mathcal{Q}$. $\mathcal{B}ool(\mathcal{I})$ est l'ensemble des expressions booléennes construites avec les variables de $\mathcal{I}$. Chaque transition est dotée d'une étiquette de la forme g/a, où $g \in \mathcal{B}ool(\mathcal{I})$ doit être vraie pour que la transition puisse être tirée (g est la garde de la transition), tandis que $a \in \mathcal{O}^*$ est une conjonction de signaux de sortie qui sont émis quand la transition est tirée (a est l'action de la transition). Une transition (s, g, a, s') sera graphiquement notée $s \overset{g/a}{\rightarrow} s'$. Par la suite, nous utilisons ce niveau de description, avec une vue graphique, pour présenter nos modèles et nos exemples.

Un *chemin* est une séquence de transitions (potentiellement infinie) initialisée dans l'état initial. Une *trace* est une séquence d'étiquettes (potentiellement infinie) associée à un chemin. Le langage d'un STE est l'ensemble des traces qu'il peut générer. Dans cet article, nous nous focalisons sur des STEs qui sont *déterministes* (le système réagit toujours de la même manière à une séquence d'entrée identique) et *réactifs* (le système réagit toujours à la stimulation d'un événement d'entrée fourni par l'environnement). Deux STEs $\langle \mathcal{Q}_1, q_{01}, \mathcal{I}_1, \mathcal{O}_1, \delta_1 \rangle$ et $\langle \mathcal{Q}_2, q_{02}, \mathcal{I}_2, \mathcal{O}_2, \delta_2 \rangle$ sont *compatibles* s'ils sont tels que $\mathcal{O}_1 \cap \mathcal{O}_2 = \emptyset$. Le *produit synchrone* entre deux tels STEs compatibles est le STE $\langle \mathcal{Q}_1 \times \mathcal{Q}_2, (q_{01}, q_{02}), \mathcal{I}_1 \cup \mathcal{I}_2, \mathcal{O}_1 \cup \mathcal{O}_2, \delta_1 \times \delta_2 \rangle$.

2.2. *Synthèse de contrôleurs*

Apparue dans les années 80 [RAM 87], la théorie du contrôle consiste à restreindre le comportement d'un système $\mathcal{P}$ par le biais d'un superviseur $\mathcal{C}$ de manière à ce que le système ainsi contrôlé soit correct vis-à-vis d'un ensemble de propriétés $\mathcal{D}$ (ou d'objectifs de contrôle/spécifications) que le système initial ne vérifiait pas : $\mathcal{P} \cap \mathcal{C} \subseteq \mathcal{D}$. La théorie du contrôle s'apparente donc à un problème d'inversion dans la mesure où le contrôleur $\mathcal{C}$ doit être calculé à partir de $\mathcal{D}$ et $\mathcal{P}$. Originellement, le système, l'objectif ainsi que le contrôleur étaient modélisés par des langages (réguliers) ; depuis, de nombreuses extensions et implémentations ont été proposées en se basant sur des STEs, des réseaux de Petri, des automates temporisés ou encore des automates symboliques, etc.

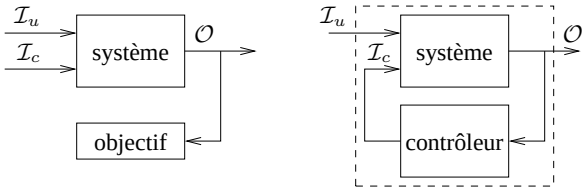


Figure 1. *Du système incontrôlé (à gauche) au contrôle en boucle fermée (à droite).*

Dans notre approche, $\mathcal{P}$ est modélisé par un STE et $\mathcal{D}$ est un objectif que le système contrôlé doit satisfaire (typiquement assurer une propriété de sûreté, rendre invariant un sous-ensemble d'états ou le rendre toujours atteignable). Le contrôleur $\mathcal{C}$ calculé de manière à assurer cet (ces) objectif(s) est obtenu en restreignant les transitions de $\mathcal{P}$ (ou d'un dépliage de $\mathcal{P}$), en inhibant les transitions qui violent l'objectif. Le point important est que l'ensemble des signaux d'entrée du système est partitionné en deux sous-ensembles $\mathcal{I}_c$ et $\mathcal{I}_u$, représentant respectivement, les entrées contrôlables et les entrées incontrôlables. Le contrôleur $\mathcal{C}$ peut seulement interdire les transitions de $\mathcal{P}$ pour lesquelles la satisfaction de la garde peut être invalidée en choisissant une valeur particulière pour les signaux d'entrées contrôlables. La figure 1 permet d'illuster comment l'objectif est exprimé en fonction des signaux de sortie. Le contrôleur obtenu en fonction de $\mathcal{P}$ et $\mathcal{D}$ (tous les deux spécifiés par l'utilisateur) est calculé automatiquement par des algorithmes *ad hoc* (qui dépendent de l'objectif) [MAR 00a].

2.3. *Synthèse optimale*

2.3.1. *Synthèse de contrôleurs et optimisation*

La théorie du contrôle telle que vue précédemment permet de calculer des contrôleurs lorsque l'objectif à assurer s'exprime de manière logique (invariance, atteignabilité, etc). À ce niveau, une deuxième question peut se poser : quelle est la meilleure façon d'y arriver, la plus rapide, la moins coûteuse, etc. La théorie de la synthèse optimale peut fournir des réponses à ces questions. Le champ d'action de cette théorie couvre le contrôle de systèmes où une performance optimale est souhaitée. Il est ainsi possible d'associer des poids aux états et/ou aux signaux d'entrée/sortie du système $\mathcal{P}$ dans le but de spécifier des bornes que le système ne doit jamais atteindre/dépasser (on peut alors se ramener dans ce cas à un problème classique de contrôle par invariance). D'autres objectifs de contrôle qualitatifs peuvent également être considérés, comme l'optimisation à un pas. Le principe est d'associer aux états du système un coût ; le contrôle revient alors à choisir pour un état donné, les signaux d'entrée contrôlables faisant évoluer le système vers l'état de plus faible (resp. fort) coût [MAR 00b]. Cette notion a par la suite été étendue sur une trajectoire bornée du système. Le problème qui se pose consiste alors en la synthèse d'un contrôleur capable de choisir une séquence d'événements contrôlables faisant évoluer le système d'un ensemble d'états initiaux vers un ensemble d'états finaux tout en minimisant une fonction de coût sur les événements [KUM 95, TRO 96, SEN 98] ou les états [MAR 98] (on peut également se référer à [MAL 04] pour une vue d'ensemble sur le sujet).

Dans le cadre de cet article, étant donné un système P, le critère d'optimalité s'exprime par l'intermédiaire d'une fonction de coût totale C portant sur les états. Le problème est donc de trouver un contrôleur de telle manière que le chemin d'exécution emprunté entre l'état initial q_o et un des états finaux $q_f \in Q_f$ ait le plus petit coût d'exécution possible. Toutefois, même si un ou plusieurs chemins ayant un coût minimal existent, il n'est pas possible de garantir que ces chemins vont effectivement être empruntés dans la mesure où l'occurrence d'événements incontrôlables peut amener le système dans des états où le minimum global ne peut plus être atteint. D'un point de vue théorique, pour un système modélisé par un STE $\mathcal{P}$, on introduit une fonction de coût $C(q)$ qui à chaque état q de l'espace d'états associe un coût entier strictement positif $C : \mathcal{Q} \rightarrow \mathbb{N}$. Le *coût d'exécution* EC d'un chemin de longueur k initialisé en l'état q_1, $E(q_1) = (q_1, \ldots, q_k)$, est simplement obtenu en sommant les coûts statiques des états traversés par E : $EC(E) = \sum_{i=1}^{k} C(q_i)$. Le but de la synthèse optimale est alors de calculer un contrôleur qui force le système à emprunter les trajectoires de plus faible coût allant de l'état initial vers un des états finaux.

2.3.2. *Calcul du contrôleur*

Nous proposons une variante de l'algorithme [BEL 57] qui prend en compte cet aspect d'optimalité sur des séquences finies (voir également [MAL 04] pour une vision plus théorie des jeux et des variantes sur l'algorithmique). Il se compose de deux étapes.

La première étape permet d'affecter à chaque état du système un coût correspondant au coût minimal nécessaire pour aller de cet état vers un des états finaux. Cette étape affecte donc à chaque état q de $\mathcal{P}$ le meilleur coût d'exécution réalisable pour atteindre Q_f, tout en sachant que l'événement incontrôlable de pire coût peut se produire. Le principe d'optimalité consiste donc à choisir la meilleure trajectoire parmi les pires que l'on ne peut éviter. Bien évidemment, il est alors possible d'affecter des coûts infinis à certains états, par exemple dès lors qu'il n'existe pas de chemin fini (par contrôle) permettant d'atteindre les états finaux ; on peut penser par exemple à des boucles dans le sytème que l'on ne peut éviter.

Par la suite nous noterons $W : Q \rightarrow \mathbb{N}$ la fonction qui affecte à chaque état de $\mathcal{P}$ le coût minimal inévitable des trajectoires issues de cet état permettant d'atteindre les états finaux. W est défini comme le plus grand point fixe défini par :

$$W^0(q) = \begin{cases} 0 & \text{ssi } q \in Q_f \\ +\infty & \text{sinon} \end{cases}$$

$$W^i(q) = \min \begin{cases} W^{i-1}(q) \\ \max_{\mathcal{I}_u} \min_{\mathcal{I}_c} C(q) + W^{i-1}(\delta(q, i_u, i_c)) \end{cases}$$

La deuxième étape consiste à partir de W à construire le contrôleur qui va piloter le système de manière à générer les meilleures trajectoires possibles atteignant Q_f. Ainsi, à chaque état q, la politique de contrôle consiste en le calcul du (ou des) successeur(s) immédiat(s) q' ayant le moindre coût inévitable par contrôle, c'est-à-dire qu'on cherche à faire transiter le système dans un des états q' tels que :

$$W(q') = \min\{W(\delta(q, i_u, i_c)\} \quad \text{et} \quad \forall i_u \exists i_c : q' = \delta(q, i_u, i_c)$$

Il est possible de montrer (c.f. [MAL 04] et [SEI 96]) que sur des systèmes finis avec des coûts positifs sur les états, cet algorithme est certain de terminer avec une complexité polymiale en la taille du système. La solution du point fixe correspond à la solution optimale au problème de la synthèse optimale. Plus de détails sur la théorie et l'implémentation sont données dans [MAR 98].

2.3.3. *Chemins optimaux dans les systèmes réactifs*

Une caractéristique de la solution obtenue est que l'ensemble des chemins qui sont solutions du problème de synthèse optimale sont des chemins acycliques. Cela implique en particulier que tous les cycles présents initialement dans le système sont coupés durant le calcul de la solution :

– Si le cycle est contrôlable, il est alors possible de casser ce cycle en contrôlant une des transitions de celui-ci.

– Si il est incontrôlable (il n'existe pas de moyen d'empêcher le système de traverser le cycle), la solution consiste donc à empêcher par contrôle le système d'entrer dans le cycle.

Comme mentionné précédemment, l'algorithme que nous avons présenté marque tous les états appartenant à des cycles incontrôlables avec des coûts infinis. Toutefois, il est a noter que, pour les systèmes que nous considérons, le cas particulier des boucles triviales (i.e. une transition qui reboucle sur le même état) requiert un traitement spécial. En effet, les boucles triviales incontrôlables s'avèrent être un artefact permettant de modéliser l'attente d'un événement dans les systèmes réactifs. Il nous semble donc normal que cette attente ne pénalise pas le meilleur coût d'exécution d'une séquence.

De manière à prendre en compte cet aspect, nous proposons une variante à l'agorithme précédent, telle que le coût engendré par les boucles triviales ne soit pris en compte qu'une seule fois. Ceci est fait en considérant dans le calcul du point fixe que les états sources et cibles doivent être différents (i.e., $q \neq q'$ dans l'équation ci-dessous) :

$$W^0(q) = \begin{cases} 0 & \text{ssi } q \in Q_f \\ +\infty & \text{sinon} \end{cases}$$

$$W^i(q) = \min \begin{cases} W^{i-1}(q) \\ \max_{\mathcal{I}_u} \min_{\mathcal{I}_c} C(q) + W^{i-1}(\delta(q, i_u, i_c)) \\ \qquad \forall q' = \delta(q, i_u, i_c) \text{ t.q. } q \neq q' \end{cases}$$

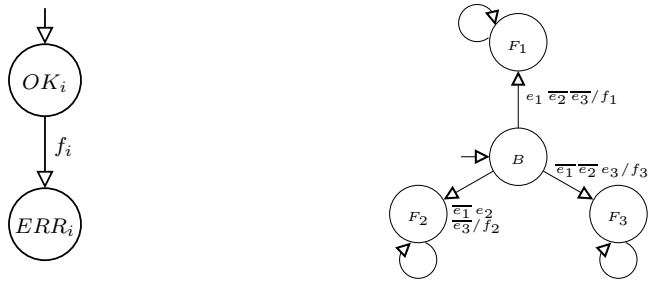
Des exemples illustrant cet algorithme et les différences entre les deux algorithmes sont donnés dans les section suivantes.

3. Modèle du système réparti

3.1. *Modèle de l'architecture*

Modèle local de processeur. Il est donné par le STE de la Figure 2(a), où OK_i signifie que le processeur i fonctionne bien, alors que ERR_i indique qu'il est défaillant. Nous supposons que seuls les processeurs peuvent défaillir, selon un mode *fail-silent*. Des études récentes ont démontré que ce fonctionnement est réalisable à un coût raisonnable [BAL 03]. Les fautes sont permanentes, donc un processeur ne peut retourner de l'état ERR à l'état OK. Modéliser des fautes intermittentes ou des modes dégradés (par ex. à une vitesse plus lente, en surcharge) est possible aussi [DUM 07]. Les processeurs peuvent être utilisés en partage de temps, de sorte que plusieurs tâches peuvent être actives sur le même processeur en même temps.

Modèle d'architecture hétérogène. Les processeurs sont reliés par un réseau complètement connecté de liens de communication point à point. Nous notons $\mathcal{S}$ l'ensemble de tous ces processeurs. Nous supposons que les liens de communication ne peuvent pas faillir. Un processeur spécifique P_0 est dédié à l'exécution du contrôleur, et seuls les autres processeurs sont disponibles pour l'exécution des tâches du système. Le modèle consiste en la composition de tous les STE comme ci-dessus. Dans notre exemple courant (voir Figure 4), nous en considérons trois, un pour chacun des processeurs P_1, P_2 et P_3, pour lesquels les bornes de capacité b_i vis-à-vis de l'empreinte mémoire possible sont, respectivement, 5, 3, et 6.



(a) Modèle de processeur : faute permanente.

(b) Modèle de faute : une seule faute.

Figure 2. *Modèle d'architecture.*

Modèles d'environnement ou de faute. Nous devons modéliser quelles fautes peuvent avoir lieu dans le système. Par exemple, combien de fautes peuvent arriver ? Ou plus précisément combien de fautes voulons-nous que le système tolère ? Ces fautes peuvent-elles arriver dans n'importe quel ordre ou y a-t-il des séquences ou motifs connus ? Peuvent-elles avoir lieu simultanément ? En termes de notre modèle de la figure 2, la question est comment les événements f_i peuvent-ils avoir lieu ? Il paraît naturel que les événements f_i soient incontrôlables (i.e., dans $\mathcal{I}_u$), puisqu'une faute est un événement intrinsèquement incontrôlable. Toutefois, ceci signifierait qu'il n'y ait aucune contrainte sur eux. En particulier, *tous* les événements f_i pourraient alors se produire, induisant une perte de tous les processeurs. Il en résulterait donc un dysfonctionnement total du système, sans aucune possibilité d'assurer sa tolérance aux fautes. On ne peut attendre d'un système de tolérer des fautes de tous les processeurs dont il est constitué. C'est pourquoi nous devons spécifier les motifs de fautes que nous considérons.

Pour modéliser ceci, nous choisissons d'avoir un STE modélisant l'environnement. Son but est d'émettre les signaux f_i à partir des signaux e_i reçus de l'environnement. Ces signaux e_i sont incontrôlables (i.e., dans $\mathcal{I}_u$), ce qui reflète le fait qu'une faute peut avoir lieu à tout moment. Les signaux f_i, eux, seront locaux, c'est-à-dire ni dans $\mathcal{I}_u$ ni dans $\mathcal{I}_c$, et seront utilisés pour calculer le produit synchrone de tous les STE. Le modèle d'environnement simple de la figure 2(b) permet une seule faute dans le système. Selon la connaissance disponible sur le système à concevoir, on peut directement spécifier le *motif de faute* en donnant le STE qui produit les signaux locaux f_i à partir des signaux d'entrée e_i [DUM 07]. Donner un tel modèle d'environnement fait partie du travail de conception. De plus, nous introduisons un événement de faute qui signale l'occurrence d'au moins une faute : $f \stackrel{def}{=} \bigvee_i f_i$.

Quand le STE du modèle de faute est acyclique, alors par composition avec les autres parties du modèle, il en résultera un STE global acyclique. Ceci correspond au fait intuitivement satifaisant que nous voulons concevoir des systèmes tolérant des motifs de fautes bornés. La technique de SCD optimale sur des chemins est alors

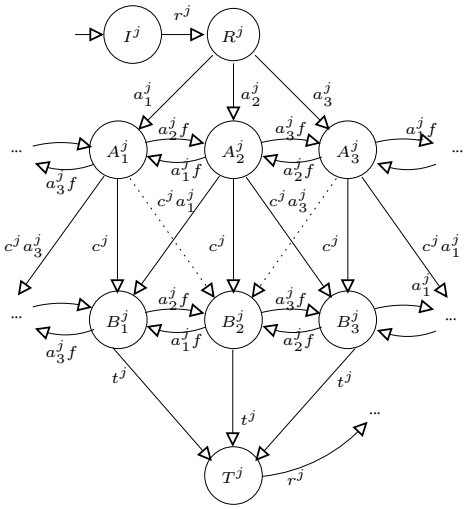


Figure 3. *Modèle de contrôle de tâche, avec points de reprise.*

applicable, et peut garantir des propriétés sur les performances, en particulier en temps de réponse, du système tolérant aux fautes résultant. On peut noter toutefois que nous n'excluons pas de considérer des applications où des motifs de fautes non bornés doivent être pris en compte. Dans ces cas-là, le modèle de faute peut décrire quelles sont les séquences infinies à tolérer, mais aucune tolérance aux fautes garantissant une optimalité ou une borne de coût sur ces chemins ne peut être obtenue.

3.2. *Modèle de tâche*

Motif de base de structure de contrôle. Chaque tâche j est formellement modélisée par un STE qui décrit comment le contrôle de l'activité de la tâche est réalisé en réaction à des événements. Par exemple, nous supposons que la tâche peut être exécutée sur trois processeurs, comme dans la figure 3 (certaines transitions ont été omises ou pointillées pour plus de lisibilité ; les transitions allant vers "..." vont vers l'état mentionné de l'autre côté de la figure). Le STE comporte un état initial *inactif* I^j, un état *prêt* R^j après réception du signal de *requête* r^j, un état *terminal* T^j, et plusieurs états *actifs* A_i^j représentant des *configurations* de la tâche, un pour chaque placement sur un processeur du système. Par convention, les indices supérieurs et inférieurs se réfèrent respectivement aux tâches et processeurs. Dans l'état A_i^j, la tâche j est exécutée sur le processeur i, jusqu'à l'occurrence de l'événement de contrôle c^j. De telles tâches peuvent être modélisées à l'aide des Automates de Modes [MAR 03].

Une transition de l'état A_i^j à l'état A_k^j représente la *reconfiguration* du système, avec l'*arrêt* de la tâche j sur le processeur i et sa *migration* et son *redémarrage* sur le processeur k. On pourrait avoir des cycles de telles reconfigurations, en particulier vis-à-vis d'un équilibrage de charge ; toutefois de tels cycles dans le STE global

modélisant le système empêcheraient l'application des techniques de SDC optimale sur des chemins. C'est pourquoi nous considérons ces reconfigurations uniquement sur l'occurrence d'événement de contrôle (e.g., $c^j a_3^j$) ou de faute (e.g., $a_2^j f$), et nous conditionnons toutes les transitions de reconfiguration par l'événement f introduit ci-dessus dans le modèle de faute : $a_i^j \wedge f$. Grâce à cela, la SCD optimale sur des chemins est applicable.

Nous introduisons une notion de phase et de point de reprise. Le passage en séquence d'une phase A_2^j à une phase B_2^j est associé à un événement de point de reprise incontrôlable c^j. Le dernier point de reprise est en fait la terminaison. Quand une tâche est migrée, elle est redémarrée depuis le début de la *phase courante* et non depuis le début de la tâche complète. En ce sens, chaque transition de phase est un point de contrôle de reprise, et quand une tâche est migrée elle fait un retour arrière au dernier point de reprise. En termes de SCD, les signaux r^j, c^j et f seront incontrôlables (i.e., dans $\mathcal{I}_u$), alors que les signaux a_i^j seront contrôlables (i.e., dans $\mathcal{I}_c$).

Caractéristiques quantitatives. Des caractéristiques intéressantes peuvent être modélisées par des poids associés aux états. Nous considérons ici le temps d'exécution, sous forme de simples fonctions des états vers les entiers. Le *temps d'exécution* est mesuré pour chaque tâche par une analyse de temps d'exécution au pire cas. Quand une tâche migre, elle recommence au dernier point de reprise, donc son nouveau processeur doit accepter la charge de la phase redémarrée de la tâche. On considère aussi l'empreinte mémoire des tâches, qui est donnée relativement à chaque processeur.

En perspective, on pourrait aussi considérer la *consommation d'énergie*, ou encore la *qualité* d'une tâche [MAR 02] donnée relativement à chaque processeur. Cette dernière peut rendre compte par exemple de la précision de résultats produits par un calcul numérique selon la disponibilité d'un co-processeur spécialisé, de différentes versions d'un algorithme de recherche heuristique à différentes profondeurs.

3.3. *Modèle d'application*

Serveur de tâches. Si le système est composé de n tâches, il y aura n STE correspondants en parallèle. Leur composition synchrone, comme dans les Automates de Modes [MAR 03], représente tous les comportements, c'est-à-dire toutes les configurations possibles, en réponse à toutes les séquences possibles de requêtes et d'événements de terminaison. La composition de caractéristiques quantitatives est considérée, dans cet article, comme étant additive. C'est intuitif pour l'empreinte mémoire pour chaque processeur P_i, où nous avons pour les tâches j : $C_i = \sum_j C_i^j$.

Comme nous le notions plus haut, dans un tel système, il y a un cycle sur le démarrage de la tâche, et donc les chemins sont infinis, et la SCD optimale sur les chemins n'est pas applicable ; toutefois les autres objectifs de synthèse proposés restent significatifs.

Ordonnanceur ou programme. Le programme a la charge d'émettre les requêtes de tâches selon un séquencement donné. Son but est d'ordonnancer les tâches selon un graphe de dépendances donné par l'utilisateur. Il doit émettre les signaux r^j dans

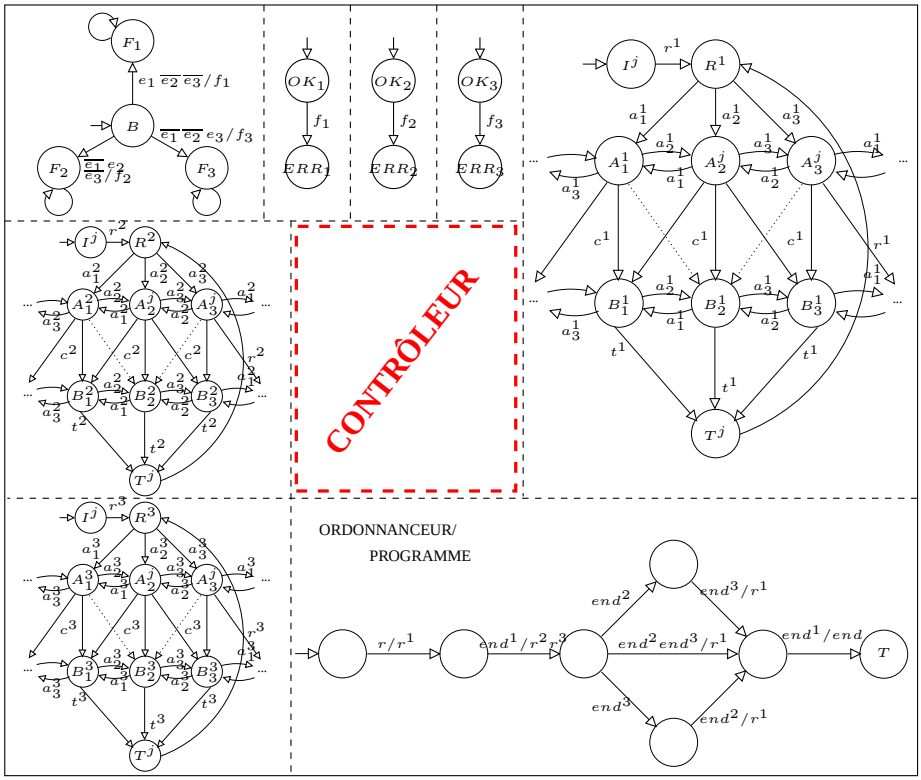


Figure 4. *Modèle de système complet : 3 processeurs et 3 tâches et un contrôleur.*

le bon ordre, de façon à ce que les tâches deviennent prêtes (dans l'état R^j) en satisfaisant les contraintes de précédence. Il est représenté par un STE, qui comporte un état terminal T. On peut noter que de tels ordonnanceurs ou programmes peuvent être obtenus à partir de langages de plus haut niveau spécifiques à des domaines d'applications [DEL 07].

3.4. *Modèle du système*

Finalement, le modèle du système multi-tâches et multi-processeurs est construit en composant les différents modèles locaux introduits précédemment, comme illustré en Figure 4 : un pour le modèle d'environnement, un pour chaque processeur, un pour chaque tâche, un pour le programme. Ceci définit un STE global, de toutes les configurations possibles du système considéré. L'application de la SCD nous fournira un contrôleur qui restreint ses comportements à ceux qui respectent les propriétés requises pour un système réparti tolérant aux fautes, tout en garantissant l'optimalité du coût en temps.

4. Propriétés de tolérance aux fautes

La manière de tolérer les fautes est spécifiée par un ensemble de propriétés qualitatives ainsi que d'objectifs quantitatifs. Ces propriétés sont spécifiques pour la tolérance aux fautes : d'une part, elles sont exprimées sur l'intégralité du modèle, tel que décrit précédemment, où tous les scénarios de faute et de recouvrement sont représentés ; d'autre part, elles caractérisent des ensembles d'états dits *de défaillance* (par exemple, une contrainte de placement cohérent caractérise l'ensemble des états où le système ne peut fonctionner), ainsi que la tolérance aux fautes en elle-même – la capacité du système à délivrer ses fonctionnalités quelles que soient les fautes.

4.1. *Propriétés d'invariance*

Rappelons brièvement les propriétés de base utilisées en tant qu'objectifs pour la synthèse de contrôleurs [GIR 04, DUM 07].

Garantir une exécution cohérente. Cette propriété exprime le fait qu'*aucune tâche ne doit être active sur un processeur défaillant* : $\neg \bigvee_j \bigvee_i (A_i^j \wedge Err_i)$. Elle est infirmée dès qu'une tâche T^j est active sur un processeur P_i (i.e., dans l'état A_i^j) alors que P_i est dans l'état Err_i. L'objectif de synthèse est de rendre cette propriété *toujours vraie*.

Aussi il est requis que *les tâches actives ne doivent pas dépasser la capacité du processeur* en mémoire disponible : $\forall i, C_i \leq b_i$. Cette propriété est infirmée dès que le coût cumulé de toutes les tâches actives sur un processeur donné dépasse la borne exprimant la capacité de ce processeur. L'objectif de synthèse est de rendre cette propriété *toujours vraie*.

Garantir le non-blocage. Ce n'est pas seulement une propriété liée aux états du système : elle concerne également des traces. Concrètement, l'objectif est d'éviter que la synthèse de contrôleurs interdise indéfiniment le démarrage d'une tâche : chaque tâche doit être activée uniquement lorsqu'on a la certitude qu'elle pourra atteindre son état terminal, même en cas d'occurrence de fautes. Cette propriété est satisfaite si et seulement si quel que soit l'état atteignable q du système, l'état terminal T du programme est atteignable à partir de q.

Dans le cas d'un serveur de tâches, en absence d'un ordonnanceur ou d'un programme, l'état terminal qui doit rester toujours atteignable est $(T^1, \ldots, T^n)$. Cette propriété est essentielle dans la caractérisation de la tolérance aux fautes, car elle se borne à exclure les comportements selon lesquels toute activité du système serait bloquée dans les états d'attente. Un tel scénario satisferait de façon triviale les propriétés précédentes, mais ne serait pas acceptable.

4.2. *Optimisations des coûts*

L'optimisation à un pas est une technique permettant de décider de la *prochaine* configuration du système qui maximise le critère global de qualité. Dans notre cas, la qualité d'exécution de chaque tâche peut varier selon le processeur sur lequel elle est exécutées. D'autre part, cette même technique permet de minimiser la consommation globale, définie comme la somme des coûts des tâches sur les processeurs.

L'optimisation des coûts sur des chemins comportant des phases d'exécution permet de choisir, pour une tâche donnée, le chemin tel que le coût total d'exécution entre l'état R et l'état T soit minimal. Ce choix doit tenir compte à tout moment des éventuelles occurrences de faute des processeurs et des migrations qui s'en suivraient.

Dans l'exemple présenté à la figure 3, l'ensemble des chemins d'exécution possibles entre les états R et T peuvent avoir des coûts d'exécution différents. Aussi, le choix des migrations doit tenir compte des autres contraintes de l'application : les propriétés exprimant des invariants doivent rester toujours vraies, et l'ordonnancement de l'ensemble des tâches doit avoir le meilleur coût d'exécution possible, malgré les pires scénarios provoqués par l'environnement incontrôlable, à travers les entrées incontrôlables du système. L'algorithme de synthèse optimale effectue une optimisation des coûts d'exécution, tenant compte des phases d'exécution de chaque tâche. Il calcule W_{Q_f}, la fonction qui renvoie le meilleur coût, qui associe à chacun des états de $\mathcal{P}$ le meilleur coût d'exécution pouvant être garanti pour atteindre la cible $Q_f = \{(T^1, \ldots, T^n)\}$. Une solution optimale existe si et seulement si $W_{Q_f}(q_0) < +\infty$. Alors, l'objectif de synthèse optimale est exprimé de la manière suivante :

$$\begin{array}{l} \forall q \in Q \text{ choisir } i_c^{min} \text{ t.q. :} \\ \quad \forall i_u, \forall i_c \neq i_c^{min} : W_{Q_f}(\delta(q, i_c^{min}, i_u)) \leq W_{Q_f}(\delta(q, i_c, i_u)) \end{array}$$

Si une solution optimale existe, alors l'atteignabilité de l'état cible du système est garantie et donc la propriété de non-blocage est satisfaite.

5. Exemple

Le contrôleur synthétisé est obtenu au bout de huit itérations de point fixe. Ce nombre d'itérations est borné par la profondeur du graphe d'états du système global. Pour des tâches avec phases d'exécution s'exécutant en parallèle et sans contraintes spécifiques sur le moment de leur démarrage, le graphe d'états global décrit l'ensemble des scénarios d'exécution possibles. Le chemin le plus long dans ce graphe, démarrant à l'état initial et allant à l'état terminal correspond à une exécution séquentielle des tâches T_1 et T_2 (par ex. T_2 ne démarre qu'après la fin de l'exécution de T_1). C'est pourquoi une estimation du nombre total d'itérations de point fixe est donnée par la somme du nombre de phases de chaque tâche du modèle.

Le contrôleur exprime une relation entre les états q du système, les événements incontrôlables i_u et les événements contrôlables i_c acceptables du point de vue du contrôle. Cette relation est décrite par l'équation $CTR(q, i_u, i_c) = 0$. Chaque pas de simulation du système contrôlé nécessite une résolution de cette équation : pour un état courant q et un événement incontrôlable i_u donnés, chercher le ou les événements contrôlables i_c satisfaisant l'équation. En cas de solutions multiples, c'est à l'utilisateur de faire le choix de l'événement contrôlable qu'il souhaite tirer. L'expression de CTR, étant relativement volumineuse, n'est pas reproduite ici pour des raisons de lisibilité.

La figure 5 illustre l'exécution du système contrôlé par le contrôleur synthétisé. Le scénario particulier mis en œuvre est celui de deux tâches qui peuvent s'exécuter sur trois processeurs. Chaque tâche possède deux phases. Dans cet exemple, les deux tâches démarrent au même moment (les événements r_1 et r_2 arrivent simultanément). On suppose par ailleurs que les deux tâches ne s'exécutent qu'une seule fois (r_1 et r_2 arrivent une seule fois). Le modèle de faute utilisé est celui présenté dans la Figure 2(b). Les coûts statiques sont représentés sous forme de nombres entiers associés à chacun des états du système. Dans cet exemple, le meilleur coût d'exécution pour la tâche T_1 serait $1 + 1 + (2 + 1) + 1 = 6$, ce qui correspond à une exécution de sa première phase sur le processeur P_3 suivie d'une exécution de la seconde phase sur le processeur P_1. Le meilleur coût d'exécution de la tâche T_2 est $1+1+(1+1)+1 = 5$, ce qui correspond à une première phase sur P_2 et une seconde phase sur P_1.

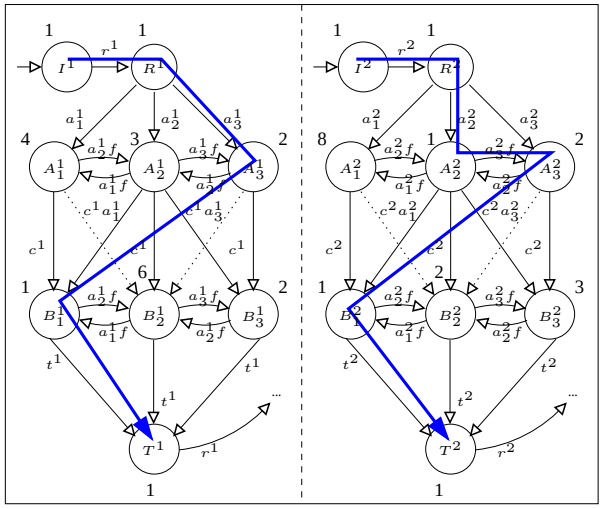


Figure 5. *Résultats de simulation pour deux tâches s'exécutant sur trois processeurs.*

L'exécution se déroule de la manière suivante. Au début de la simulation, T_1 est affectée sur P_3 et T_2 est affectée sur P_2. C'est à ce moment-là que le processeur P_2 défaille. Suite à cette défaillance, T_2 doit migrer immédiatement, car elle ne peut continuer à rester sur un processeur défaillant. La meilleure solution de migration, du point de vue du coût d'exécution, est offerte par le processeur P_3. La tâche T_1 continue quant à elle de s'exécuter sur le processeur P_3. Les deux tâches peuvent effectuer leur changement de phase indépendamment l'une de l'autre, lorsqu'elles reçoivent l'événement incontrôlable c_1 ou c_2. Suite à un changement de phase, des migrations de processeurs peuvent se produire pour des raisons d'optimisation. Ainsi, T_1 et T_2 migrent chacune vers le processeur P_1 afin de réduire au mieux leur coût d'exécution respectif. L'exécution de chacune des tâches se termine dès réception des événements t^1 et t^2. Ainsi, T_1 se termine avec un coût total d'exécution égal à 6, qui se trouve être le meilleur coût d'exécution possible. En revanche, T_2 se termine avec un coût égal à 7 au lieu du coût minimal 5, à cause de la défaillance du processeur P_2 suivie d'une

migration depuis P_2 vers P_3, qui entraîne le paiement intégral du coût de la phase A, précédemment amorcée sur P_2, mais qui n'a pas pu se terminer suite à la défaillance.

La gestion des boucles incontrôlables est fondamentale. Ces cycles se retrouvent en grand nombre dans notre exemple : à chaque fois qu'il y a une attente indéfinie sur l'occurrence d'un événement incontrôlable. Les événements de départ r^1 et r^2, de changement de phase d'exécution c^1 et c_2 et de terminaison t^1 et t^2 peuvent être attendus indéfiniment. L'algorithme classique de Bellman pour la synthèse optimale calcule systématiquement le pire coût associé à ces attentes indéfinies, qui est égal à $+\infty$. Ainsi, il n'y aurait aucune solution optimale et tous les chemins d'exécution seraient équivalents du point de vue du coût d'exécution, car leur coût serait $+\infty$. Cependant, comme précédemment mentionné, les boucles d'attente ne devraient pas pénaliser le coût d'exécution. L'algorithme proposé dans cet article calcule les meilleurs coûts d'exécution pouvant être garantis, tout en comptant une et une seule fois les coûts des états à l'origine d'une attente indéfinie. Le contrôleur obtenu permet ainsi une attente indéfinie pour tous les événements incontrôlables, tout en pilotant l'exécution des tâches lorsqu'une défaillance se produit ou pour améliorer le coût d'exécution globale.

6. Conclusion

Nous avons montré comment modéliser un système réparti temps-réel, son architecture hétérogène et son environnement, de façon à produire automatiquement un contrôleur qui assure la tolérance aux fautes. Il réagit à l'occurrence de défaillances par la migration des tâches suivant la politique de tolérance. Nous avons appliqué des techniques de synthèse de contrôleurs discrets (SCD) à des modèles sous forme de système de transitions étiquetées (STE) du système complet, avec des objectifs d'exécution consistante, de fonctionnalité et des optimisations. Pour cela, nous avons proposé une variante de l'algorithme de Bellman de SCD optimale sur les chemins finis atteignant une configuration but, de façon à traiter correctement les transitions cycliques d'attente, ce qui est pertinent dans les systèmes réactifs. Du point de vue de la tolérance aux fautes, l'intérêt de notre approche est que, quand la SCD produit une solution, nous obtenons un système qui tout à la fois assure une reconfiguration *dynamique* pour traiter les fautes et une garantie *statique* que toutes les fautes spécifiées seront tolérées, avec un borne connue sur le temps d'exécution global.

Les perspectives s'envisagent par exemple dans la direction de variantes de modèles de tâche, avec des modes différents (e.g. de consommation ou dégradés) ; de propriétés logiques d'exclusion ou de séquencement entre tâches, utilisant des observateurs, ou plus quantitatives sur l'utilisation de mémoire, le coût des migrations, etc.

7. Bibliographie

[ALT 03] ALTISEN K., CLODIC A., MARANINCHI F., RUTTEN E., « Using Controller-Synthesis Techniques to Build Property-Enforcing Layers », *Proceedings of the European Symposium on Programming, ESOP'03*, Warsaw, Poland, avril 2003.

[BAL 03] BALEANI M., FERRARI A., MANGERUCA L., PERI M., PEZZINI S., SANGIOVANNI-VINCENTELLI A., « Fault-Tolerant Platforms for Automotive Safety-Critical Applications », *International Conference on Compilers, Architectures and Synthesis for Embedded Systems, CASES'03*, San Jose, USA, novembre 2003, ACM.

[BEL 57] BELLMAN R., *Dynamic programming*, Princeton University Press, 1957.

[DEL 07] DELAVAL G., RUTTEN E., « A Domain-Specific Language for Multi-Task Systems, Applying Discrete Controller Synthesis », *Journal on Embedded Systems* (special issue on Synchronous Paradigm in Embedded Systems), vol. 2007, n° 84192, 2007, `http ://www.hindawi.com/journals/es/raa.84192.html` .

[DUM 07] DUMITRESCU E., GIRAULT A., MARCHAND H., RUTTEN E., « Optimal discrete controller synthesis for the modeling of fault-tolerant distributed systems », Rapport de Recherche n° 6137, mars 2007, INRIA, hal.inria.fr.

[GIR 04] GIRAULT A., RUTTEN E., « Discrete Controller Synthesis for Fault-Tolerant Distributed Systems », *Proc. of the Ninth Int. Workshop on Formal Methods for Industrial Critical Systems, FMICS 04*, Linz, Austria, September 20-21, 2004, ENTCS, Vol. 133, `http ://dx.doi.org/10.1016/j.entcs.2004.08.059`.

[KUM 95] KUMAR R., GARG V., « Optimal Supervisory Control of Discrete Event Dynamic Systems », *SIAM J. Control Optim.*, vol. 33, n° 2, 1995, p. 419–439.

[MAL 04] MALER O., « On Optimal and Sub-Optimal Control in the Presence of Adversaries », *Workshop on Discrete Event Systems, WODES'04*, Reims, France, September 2004.

[MAR 98] MARCHAND H., LE BORGNE M., « On the Optimal Control of Polynomial Dynamical Systems over Z/pZ », *4th IEE International Workshop on Discrete Event Systems*, Cagliari, Italie, August 1998, p. 385–390.

[MAR 00a] MARCHAND H., BOURNAI P., BORGNE M. L., GUERNIC P. L., « Synthesis of Discrete-Event Controllers based on the Signal Environment », *Discrete Event Dynamic System : Theory and Applications*, vol. 10, n° 4, 2000, p. 325–346.

[MAR 00b] MARCHAND H., SAMAAN M., « Incremental Design of a Power Transformer Station Controller using Controller Synthesis Methodology », *IEEE Trans. on Software Engineering*, vol. 26, n° 8, 2000, p. 729–741.

[MAR 02] MARCHAND H., RUTTEN E., « Managing Multi-Mode Tasks with Time Cost and Quality Levels Using Optimal Discrete Controller Synthesis », *Euromicro Conference on Real-Time Systems, ECRTS'02*, Vienna, Austria, juin 2002.

[MAR 03] MARANINCHI F., RÉMOND Y., « Mode-Automata : a new Domain-Specific Construct for the Development of Safe Critical Systems », *Science of Computer Programming*, vol. 46, n° 3, 2003, p. 219–254, Elsevier.

[RAM 87] RAMADGE P., WONHAM W., « Supervisory Control of a Class of Discrete Event Processes », *SIAM Journal on Control and Optimization*, vol. 25, n° 1, 1987, p. 206–230.

[SEI 96] SEIDL H., « Least and Greatest Solutions of Equations over N », *Nordic Journal of Computing*, vol. 3, 1996, p. 41-62.

[SEN 98] SENGUPTA R., LAFORTUNE S., « An Optimal Control Theory for Discrete Event Systems », *SIAM J. Control Optim.*, vol. 36, n° 2, 1998, p. 488–541.

[TRO 96] TRONCI E., « Optimal Finite State Supervisory Control », *IEEE Conference on Decision and Control, CDC'96*, Kobe, Japan, décembre 1996, IEEE.