Réseaux d'automates stochastiques à temps discret

Anne Benoit* — **Brigitte Plateau*** — **William J. Stewart****

** Laboratoire Informatique et Distribution (CNRS - INPG - INRIA - UJF)*
Antenne Ensimag de Montbonnot, ZIRST
51, avenue Jean Kuntzmann
F-38330 Montbonnot Saint Martin
{Anne.Benoit, Brigitte.Plateau}@imag.fr

*** North Carolina State University*
Raleigh, NC 27695-8206,USA
William J. Stewart's research was supported in part by NSF Grant ACI-0203971
billy@csc.ncsu.edu

RÉSUMÉ. *Les chaînes de Markov facilitent l'analyse des performances des systèmes dynamiques dans de nombreux domaines d'application. Elles sont souvent utilisées par le biais d'un formalisme de haut niveau. Parmi les différents formalismes couramment utilisés, on se place dans le cadre des réseaux d'automates stochastiques (SAN). De nombreux travaux dans ce domaine traitent les systèmes à temps continu. Les systèmes à temps discret sont plus délicats à modéliser, car plusieurs événements peuvent avoir lieu pendant une même unité de temps (événements en conflit). Nous définissons un formalisme de SAN pour les modèles à temps discret qui gère les événements en conflit, et nous proposons un algorithme de génération de la chaîne de Markov équivalente.*

ABSTRACT. *Markov Chains facilitate the performance analysis of dynamic systems in many areas of application. They are often used through a high-level formalism. Several of these are currently used, specially for continuous-time systems, and we work on Stochastic Automata Networks (SAN). Discrete-time systems are more difficult to model, because several events can occur during the same time slot (conflicting events). We define a SAN formalism for discrete-time models, and we present an algorithm to generate the equivalent Markov chain.*

MOTS-CLÉS : *évaluation des performances, chaînes de Markov, réseaux d'automates stochastiques, temps discret, événements en conflit.*

KEYWORDS: *performance evaluation, Markov chains, stochastic automata networks, discrete time, conflicting events.*

1. Introduction

Les chaînes de Markov facilitent l'analyse des performances des systèmes dynamiques dans de nombreux domaines d'application [STE 94] grâce à un ensemble de théorèmes qui permettent de mettre en œuvre des calculs matriciels pour l'obtention des indices de performance attendus.

Plusieurs formalismes de haut niveau ont été proposés pour permettre de générer des chaînes de Markov très grandes et très complexes de façon compacte et structurée. Par exemple, les réseaux d'automates stochastiques (SAN) [PLA 84, FER 98], les réseaux de files d'attente [BUC 94], les réseaux de Petri stochastiques généralisés [MAR 95, DON 94], les algèbres de processus [HIL 95] sont largement utilisés dans divers domaines d'application, grâce à leur grand pouvoir de modélisation. Le système est alors représenté sous forme de sous-systèmes qui interagissent. Il est alors possible d'obtenir le générateur infinitésimal de la chaîne de Markov à partir du formalisme de haut niveau, et de calculer les solutions stationnaires et transitoires. Lors de l'utilisation de tels formalismes, un logiciel permet souvent de générer l'espace d'états et de calculer des indices de performances [KEM 96, BUC 00, CIA 99, PLA 91].

Le formalisme des réseaux d'automates stochastiques (SAN) à temps continu a déjà été largement développé depuis une vingtaine d'années [PLA 84, PLA 91, FER 98]. Les sous-systèmes sont des automates traditionnels auxquels on rajoute des comportements stochastiques et des mécanismes de synchronisation. La matrice de transition de la chaîne de Markov peut alors être exprimée sous un format tensoriel.

La plupart des travaux en évaluation de performance utilisent des chaînes de Markov à **temps continu** et notamment les approches précédemment citées. La raison en est double : pour les systèmes que nous voulons modéliser, les hypothèses probabilistes qui sont celles du temps continu (durée ayant une distribution exponentielle, donc non bornée et sans mémoire, aucune simultanéité d'événements) conviennent d'une part, et d'autre part, ces hypothèses probabilistes rendent aisée la génération de la matrice de la chaîne de Markov sous-jacente.

Il n'en est pas moins vrai que l'intérêt d'étudier un système en **temps discret** est réel : les modèles en temps discret permettent de coller à la réalité (donc d'ajuster correctement les probabilités de transition), quand celle-ci est perçue lors d'intervalles de temps discrétisé, ils permettent de modéliser des durées fixes et bornées et enfin la simultanéité d'événements est possible. Si les théorèmes de calcul de la théorie de Markov sont les mêmes en temps discret et continu, la difficulté, pour les chaînes de Markov en temps discret est le calcul effectif de sa formulation matricielle. Cette difficulté vient essentiellement de la combinatoire générée par la possibilité d'avoir des événements simultanés. Cette combinatoire est particulièrement importante dans le cas de modèles à base de composants, comme les SAN, dédiés à l'étude des systèmes parallèles et distribués.

La section suivante présente plus en détail le problème lié aux événements en conflit, à travers différentes techniques de modélisation à temps discret que l'on trouve

dans la littérature, notamment pour le formalisme des réseaux de Petri, des files d'attente, et des réseaux d'automates stochastiques. La principale contribution de cet article consiste en un nouveau formalisme de réseaux d'automates stochastiques (SAN) à temps discret, présenté en section 3. Un algorithme pour générer la chaîne de Markov équivalente à un tel modèle est détaillé en section 4. Nous présentons en section 5 un exemple d'application. Pour conclure, nous évoquons les travaux en cours sur ce formalisme, ainsi que les perspectives ouvertes par ces travaux.

2. Modélisation en temps discret

Parmi les modèles de performance à temps discret, les plus développés sont les réseaux de files d'attente et les réseaux de Petri. Nous détaillons également une approche basée sur le formalisme des réseaux d'automates stochastiques.

2.1. *Réseaux de Petri*

Les réseaux de Petri à temps discret peuvent être classés en deux familles. Tout d'abord, les *réseaux de Petri temporisés* [HOL 87, ZUB 91] permettent une modélisation du temps sur un espace entier, et leur objectif consiste principalement à faire de la vérification de systèmes temporisés temps réel. Leur comportement n'est pas stochastique, mais déterministe. Les techniques d'analyse de ces modèles sont relativement complexes.

L'autre famille est celle des *réseaux de Petri stochastiques*. De nombreux travaux existent sur ces modèles, et proposent souvent différentes sémantiques pour traiter le problème des événements en conflit.

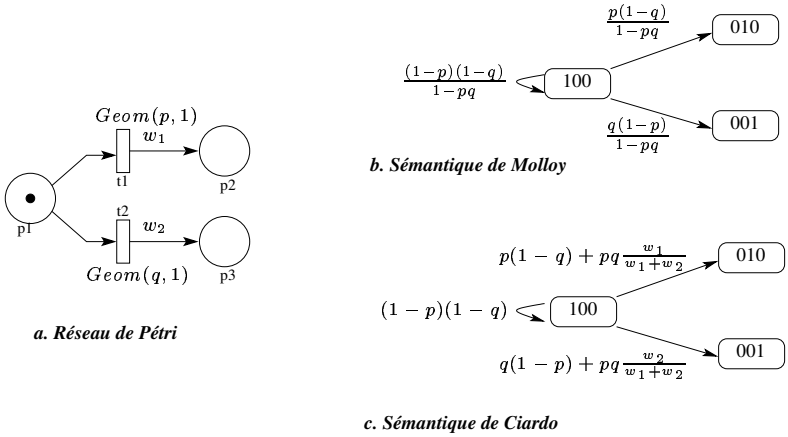


Figure 1. *Différentes sémantiques pour traiter les événements en conflit*

Lorsque nous considérons un modèle en temps discret, il se peut que plusieurs transitions soient candidates au même instant, comme par exemple les transitions $t1$ et $t2$ dans la figure 1a. Les transitions sont dites *en conflit*. Il faut alors choisir quelle transition tirer. Nous associons l'événement e à la transition $t1$ et l'événement f à la transition $t2$. La probabilité d'occurrence de e est p, celle de f est q.

Le problème des conflits ne se pose pas en temps continu. En effet, dans ce cas, l'occurrence des événements est strictement ordonnée avec le temps, et la simultanéité de deux événements ne se produit donc jamais.

Nous trouvons différentes sémantiques pour traiter les conflits dans la littérature des réseaux de Petri. Ces sémantiques sont expliquées à l'aide du graphe des marquages du réseau de Petri. Les sommets de ce graphe correspondent aux marquages possibles. Dans l'exemple de la figure 1, le marquage xyz signifie qu'il y a x jetons dans la place $p1$, y jetons dans la place $p2$ et z jetons dans la place $p3$. Les étiquettes sur les transitions correspondent à la probabilité de passer d'un marquage à un autre dans le réseau de Petri.

Dans les premiers travaux de Molloy [MOL 85] sur les réseaux de Petri stochastiques à temps discret, les deux événements en conflit e et f ne peuvent pas avoir lieu en même temps. Dans le graphe des marquages du réseau de Petri (figure 1b), nous divisons les probabilités par $(1-pq)$ pour avoir une somme des probabilités égale à 1. Ainsi, si aucun événement n'a lieu (probabilité $(1-p)(1-q)$), nous restons dans le même état, sinon le jeton va dans la place $p2$ ou $p3$.

Dans l'approche de Ciardo [CIA 94, CIA 95, ZIJ 96], nous affectons un poids à chaque événement : $w1$ et $w2$. Les probabilités ne sont alors plus divisées comme pour [MOL 85], mais nous considérons que les deux événements peuvent se déclencher simultanément (probabilité pq), et la place du jeton dépend alors des poids (probabilité $w1/(w1+w2)$ que l'événement e ait effectivement lieu). La figure 1c illustre cette sémantique.

L'approche évoquée dans [SCA 98] est similaire à celle de Ciardo, avec des poids fixés dans ce cas à $1/2$. Ces travaux proposent également un formalisme tensoriel pour représenter la chaîne de Markov à temps discret.

2.2. *Réseaux de files d'attente*

Les *réseaux de files d'attente à temps discret* ont été développés dans [GEL 98, WOO 94]. Dans ces modèles, la simultanéité des arrivées et des départs de clients est prise en compte avec différentes hypothèses qui permettent de représenter des remplacements de clients dans une file. Par exemple, si l'on se place dans une hypothèse d'arrivées tardives et que l'on considère une file d'attente pleine, on peut avoir dans une même unité de temps le service d'un client puis l'arrivée d'un nouveau client dans la file.

2.3. *Réseaux d'automates stochastiques*

Un formalisme de *réseaux d'automates stochastiques* à temps discret a été proposé dans [ATI 92, GUS 03]. Dans cette approche, la notion d'*événements compatibles* a été introduite pour identifier les événements pouvant avoir lieu simultanément pendant une même unité de temps. L'ensemble des événements compatibles doit alors être donné dans le modèle SAN, il est supposé connu. De plus, une contrainte globale sur la somme des probabilités (inférieure à 1) doit être vérifiée pour s'assurer que le modèle est correct. Cependant, la sémantique en cas de conflit ne précise pas quel événement va avoir lieu en premier, et ceci peut engendrer plusieurs comportements différents du modèle. Ainsi, si l'on a deux événements compatibles avec des taux fonctionnels, il se peut que l'un d'eux ait lieu et qu'alors le deuxième devienne impossible.

Nous présentons une refonte du formalisme de SAN à temps discret, en recherchant d'une part la simplicité du modèle, et d'autre part en explicitant la sémantique en cas de conflit. Un algorithme permet de générer la chaîne de Markov pour cette sémantique, et les événements en conflit sont identifiés automatiquement lors de la génération.

La différence principale avec les SAN à temps continu est liée à la simultanéité possible des événements dans les modèles en temps discret. Nous devons alors choisir la sémantique à appliquer lorsque cela est possible.

Si nous considérons par exemple le problème classique de l'exclusion mutuelle, nous pouvons envisager qu'il y ait dans un même intervalle de temps une relâche de la ressource suivie d'une prise de ressource par un autre client. Nous choisissons pour notre part de donner des priorités aux événements, et de prendre en compte les événements dans l'ordre de priorité tant qu'ils sont possibles. Cela n'était pas le cas dans l'approche de [ATI 92]. Nous détaillons maintenant ce nouveau formalisme de SAN à temps discret.

3. Formalisme de SAN à temps discret

L'approche modulaire des réseaux d'automates stochastiques nous amène à décrire le système comme un ensemble de sous-systèmes qui interagissent. Chaque sous-système est modélisé par un automate stochastique, et des règles établies entre les états internes de chaque automate permettent de décrire les interactions entre les sous-systèmes. Nous commençons par une rapide présentation informelle des SAN à temps discret avant de définir le formalisme, afin d'en aider la compréhension.

3.1. *Présentation informelle*

Un réseau d'automates stochastiques à temps discret est constitué d'un ensemble d'automates et d'un ensemble d'événements. Les **états internes**, ou **états locaux** d'un

automate correspondent aux différents états possibles du sous-système modélisé par cet automate.

L'occurrence d'un événement peut modifier l'état interne d'un ou de plusieurs automates. Lorsqu'un seul automate est impliqué, l'événement est dit **local**. Sinon, nous parlons d'événement **synchronisant**. Dans tous les cas, nous avons un changement de l'**état global** du système, qui est un tuple des états internes de chaque automate.

Chaque automate est ainsi caractérisé par un ensemble d'états et un ensemble de **transitions**, étiquetées par des événements, qui permettent de changer l'état de l'automate.

Pour représenter graphiquement un réseau d'automates stochastiques, nous pouvons ainsi associer un graphe à chaque automate. Les **nœuds** du graphe correspondent aux états locaux de l'automate, et les **arcs** sont étiquetés par une liste d'événements. Il se peut en effet que plusieurs événements distincts produisent le même changement d'état local dans un automate. Dans ce cas, nous avons formellement une transition par événement, et l'arc du graphe représente toutes ces transitions en même temps.

3.2. *Définition formelle*

Nous considérons la formalisation d'un **réseau d'automates stochastiques** comportant N automates stochastiques $\mathcal{A}^{(i)}$, $i \in \{1, \ldots, N\}$, et un ensemble d'événements $\mathcal{E}$.

Un **automate stochastique** $\mathcal{A}^{(i)}$ est constitué de :

– un ensemble d'états locaux $S^{(i)}$;

– un ensemble de transitions $T^{(i)}$.

Le nombre d'états locaux de l'automate est $n_i = |S^{(i)}|$, et $x^{(i)} \in S^{(i)}$ est un **état local** de l'automate $\mathcal{A}^{(i)}$.

Un **état global** du SAN est un tuple $(x^{(1)}, ..., x^{(N)})$ avec, pour $i \in \{1, \ldots, N\}$, $x^{(i)} \in S^{(i)}$. L'**espace d'états produit**, noté $\hat{S}$, est l'ensemble des états globaux du SAN. Le nombre d'états globaux est $|\hat{S}| = \prod_{i=1}^{N} n_i$.

Définition 1. *Un* **événement** $e \in \mathcal{E}$ *est défini par :*

– une probabilité d'occurrence de l'événement pt_e*, qui est une fonction de* $\hat{S}$ *dans* $[0, 1]$ *;*

– une priorité $prio_e$*, qui détermine comment agir en cas de conflit. C'est un entier strictement positif, 1 est la priorité maximum et* $+\infty$ *la priorité minimum ;*

– un ensemble d'automates impliqués par cet événement O_e *(l'occurrence de* e *entraîne une modification de l'état local de chacun de ces automates).*

L'événement e *est dit* **local** *à l'automate* $\mathcal{A}$ *si* O_e *est réduit à* $\{\mathcal{A}\}$. *Sinon, l'événement est dit* **synchronisant**.

Notons qu'une probabilité d'occurrence constante peut être vue comme une fonction qui a toujours la même évaluation, quels que soient les arguments. A chaque intervalle de temps, tout événement peut soit avoir lieu, soit ne pas avoir lieu : la loi d'occurrence d'un événement est une variable aléatoire indépendante suivant une loi de Bernouilli.

Pour permettre la simultanéité des événements, nous voulons pouvoir rendre un événement possible dans certains états même si cet événement ne peut en fait avoir lieu que si un autre événement a eu lieu avant, au cours de la même unité de temps. Par exemple, dans une file d'attente de capacité finie, un nouveau client peut être admis dans une file pleine lorsqu'un départ a lieu durant cette même unité de temps. Cette possibilité est indiquée dans le modèle par une transition vers un état fictif, appelé **état vide**, dont nous détaillerons le fonctionnement par la suite. Cet état particulier est noté x_v.

Définition 2. *Pour chaque automate* $\mathcal{A}^{(i)}$, *une* **transition** $t^{(i)}$ *est définie par :*

– *un état de départ* $x_{t^{(i)},dep} \in S^{(i)}$ *et un état d'arrivée* $x_{t^{(i)},arr} \in S^{(i)} \cup x_v$;

– *un événement associé à la transition* $evt_{t^{(i)}} = e \in \mathcal{E}$;

– *une probabilité de routage* $\pi_{t^{(i)}}$, *qui est une fonction de* $\hat{S}$ *dans* $[0,1]$. *Si* $x \in \hat{S}$, $\pi_{t^{(i)}}(x)$ *est la fonction évaluée pour l'état* x.

$T^{(i)}$ *est* **l'ensemble** *des transitions.*

Notons qu'un même événement e peut être associé à plusieurs transitions. C'est le cas d'un événement synchronisant, ou d'un événement qui peut avoir lieu dans plusieurs états locaux distincts d'un même automate (donc générant plusieurs transitions dans cet automate).

De plus, pour tout état global $x = (x^{(1)}, ..., x^{(N)})$ et tout événement $e \in \mathcal{E}$, si nous considérons le sous ensemble des transitions $t^{(i)} \in T^{(i)}$ telles que $x_{t^{(i)},dep} = x^{(i)}$, $x_{t^{(i)},arr} \in S^{(i)}$ et $evt_{t^{(i)}} = e$, alors la somme des probabilités de routage correspondant à ces transitions, évaluées pour l'état x, doit être égale à 1.

Il faut également remarquer que, dans certains cas, plusieurs événements peuvent conduire à partir d'un état local de départ donné dans un même état local d'arrivée. Dans ce cas, l'arc du graphe représentant l'automate est étiqueté par une liste d'événements, mais formellement nous définissons une transition de $T^{(i)}$ pour chaque événement de la liste.

Définition 3. *Dans un état global* $x = (x^{(1)}, ..., x^{(N)}) \in \hat{S}$, **l'événement** $e \in \mathcal{E}$ **est possible** *si et seulement si, pour tout automate* $\mathcal{A}^{(i)} \in O_e$, *il existe au moins une transition* $t^{(i)} \in T^{(i)}$ *telle que* $x_{t^{(i)},dep} = x^{(i)}$ *et* $evt_{t^{(i)}} = e$.

L'ensemble des événements possibles *dans l'état x est noté $EP(x)$*[1].

L'événement e **est réalisable** *depuis x si et seulement si $e \in EP(x)$, et si chaque transition $t^{(i)}$ définie ci-dessus vérifie $x_{t^{(i)},arr} \in S^{(i)}$ et $pt_e(x) \neq 0$.*

L'événement e réalisable depuis x qui a la priorité maximale dans l'ensemble des événements réalisables depuis x **a lieu** *avec la probabilité $pt_e(x)$ (probabilité d'occurrence fonctionnelle évaluée pour l'état de départ), et le nouvel état du SAN $x' = (x'^{(1)}, ..., x'^{(N)})$ est défini par :*

– pour $\mathcal{A}^{(i)} \in O_e$, pour chaque transition $t^{(i)}$ telle que $x_{t^{(i)},dep} = x^{(i)}$ et $evt_{t^{(i)}} = e$, nous avons $x'^{(i)} = x_{t^{(i)},arr}$ avec la probabilité de routage $\pi_{t^{(i)}}(x)$;

– pour les autres automates $\mathcal{A}^{(i)}$, nous avons $x'^{(i)} = x^{(i)}$.

x' est un **état successeur** *de x.*

Du fait des taux fonctionnels et des événements synchronisant, certains états de $\hat{S}$ ne sont pas accessibles depuis un état initial fixé (la probabilité que le SAN arrive dans cet état à tout instant est nulle). L'état initial, qui est par définition accessible, est choisi arbitrairement (c'est la modélisation du système qui détermine souvent ce choix). A partir de cet état, nous pouvons calculer les états successeurs de x. Ces états sont accessibles par définition, et les états successeurs de chaque état accessible sont également accessibles. Nous pouvons donc obtenir par itération l'ensemble des **états accessibles** du SAN, noté S, et $S \subseteq \hat{S}$.

Afin de comprendre l'intérêt de ces définitions, considérons le scénario suivant qui sera formalisé dans la section 4. Dans le même intervalle de temps, tous les événements de $EP(x)$ sont susceptibles d'avoir lieu, du plus prioritaire au moins prioritaire. Après avoir atteint l'état successeur x', nous considérons que $EP(x)$ est diminué de l'événement e qui a eu lieu. Nous recommençons alors cette procédure tant qu'il y a des événements dans $EP(x)$, pour l'événement le plus prioritaire, à partir de l'état x' (recherche des successeurs de x'). A chaque étape, l'état de départ que nous examinons est appelé l'*état courant*.

Une transition t vers l'état vide permet de rendre un événement possible dans un état donné, sans qu'il soit réalisable. Soit $e = evt_t$ et $x = x_{t,dep}$. L'état d'arrivée est $x_{t,arr} = x_v$. Dans ce cas, l'événement e ne peut avoir lieu depuis x seulement si un événement e' plus prioritaire a lieu avant, et si l'événement e est réalisable depuis l'état atteint par l'occurrence de e'.

Ainsi, une transition ne peut mener à l'état vide que s'il n'existe pas d'autres transitions définies par le même état de départ et le même événement, dont l'état d'arrivée est différent de l'état vide. En effet, dans ce dernier cas, l'événement est possible depuis l'état de départ donc il est inutile de rajouter une transition menant à l'état vide, cette dernière ne servant qu'à rendre possible l'événement depuis l'état de départ. Par

1. Dans [ATI 92], $EP(x)$ correspond à l'ensemble des événements compatibles à partir de x.

définition, aucune transition ne part de l'état vide vu que ce n'est qu'un état fictif qui sert à rendre possible certains événements.

Considérons par exemple une file d'attente dans laquelle les départs sont plus prioritaires que les arrivées. Lorsque la file est pleine, nous pouvons autoriser une arrivée (transition vers l'état vide qui rend l'événement *arrivée* possible), mais une arrivée ne pourra effectivement avoir lieu que si un départ a lieu avant. Sinon, la file étant pleine, il ne peut pas y avoir d'arrivée.

Nous détaillons maintenant comment construire la chaîne de Markov à temps discret qui représente le système. Cette chaîne ne contient que les états accessibles du modèle. La sémantique d'un réseau d'automates stochastiques est définie par une chaîne de Markov, et nous appelons cette chaîne de Markov la chaîne de Markov **équivalente** au modèle.

4. Construction de la chaîne de Markov équivalente

Les chaînes de Markov permettent de décrire l'évolution temporelle d'un système dynamique, en définissant un espace d'état dans lequel se promène aléatoirement le système. Une chaîne de Markov a la propriété suivante : son évolution ne dépend que de l'état courant, mais pas de son passé.

Nous présentons tout d'abord un algorithme qui permet de construire la chaîne de Markov qui représente le système que l'on modélise (chaîne de Markov équivalente), puis nous illustrons le déroulement de l'algorithme à l'aide d'un exemple. Enfin, nous expliquons comment la chaîne de Markov équivalente est définie.

4.1. *Algorithme de construction de la chaîne de Markov équivalente*

Les états de la chaîne de Markov sont les états accessibles du SAN (ensemble S). Pour construire les transitions de la chaîne de Markov, nous étudions les états de départ les uns après les autres, et nous calculons les probabilités d'aller dans un autre état.

Nous présentons tout d'abord le squelette de l'algorithme qui permet de générer tous les états successeurs d'un état accessible donné x, et les probabilités de transition associées. Nous obtenons ainsi toutes les transitions partant de l'état x dans la chaîne de Markov.

Au démarrage de l'algorithme, l'état courant $y = x$, et nous examinons alors tous les événements $e \in EP(x)$, du plus au moins prioritaire.

– si e est réalisable depuis y, il peut avoir lieu ou non, avec les probabilités respectives $pt_e(y)$ et $1 - pt_e(y)$. S'il n'a pas lieu, l'état courant ne change pas. Sinon, nous obtenons un ensemble d'états successeurs de y qui peuvent être obtenus par différents routages ;

– si e n'est pas réalisable, la probabilité qu'il n'ait pas lieu est 1. On reste donc dans le même état.

A partir de tous les états successeurs obtenus (nouveaux états courants), nous traitons les événements de $EP(x)$ qui suivent par ordre de priorité.

Remarquons que nous décidons ainsi de la sémantique lorsque plusieurs événements arrivent dans le même intervalle de temps. Après avoir déclenché l'événement le plus prioritaire, on atteint un état successeur, et nous regardons si l'événement de $EP(x)$ suivant (par ordre de priorité) est réalisable depuis cet état.

Lorsque l'ensemble des événements est épuisé, les états atteints sont des états de la chaîne de Markov, obtenus éventuellement par l'occurrence simultanée de plusieurs événements (les événements de $EP(x)$ qui ont eu lieu). Ces états sont appelés les *états accessibles depuis x*.

L'algorithme 1 calcule l'ensemble des états accessibles depuis un état x donné, ainsi que la probabilité d'accéder à chacun de ces états. Cet ensemble est représenté par un ensemble de couples (état, probabilité). Toutes les transitions de la chaîne de Markov partant de l'état x sont alors obtenues, ainsi que les probabilités correspondantes. Les nouveaux états sont des états de la chaîne de Markov, et nous appelons itérativement l'algorithme 1 avec ces nouveaux états en entrée. Ces appels permettent d'explorer l'espace des états accessibles du SAN S, et nous obtenons au final toutes les probabilités de transition d'un état à un autre.

4.2. *Exemple d'illustration du déroulement de l'algorithme*

Nous illustrons maintenant le déroulement de l'algorithme sur un petit exemple d'étude. Le système considéré est celui de l'exclusion mutuelle. Deux clients désirent utiliser une unique ressource. Chaque client peut être soit dans l'état *repos* (il n'utilise pas la ressource), soit dans l'état *activité* (il utilise la ressource). Nous modélisons donc chaque client par un automate à deux états : $\mathcal{A}^{(1)}$ et $\mathcal{A}^{(2)}$ sont les automates décrivant respectivement les clients 1 et 2. Les états internes de chacun de ces automates sont *repos* et *activité*.

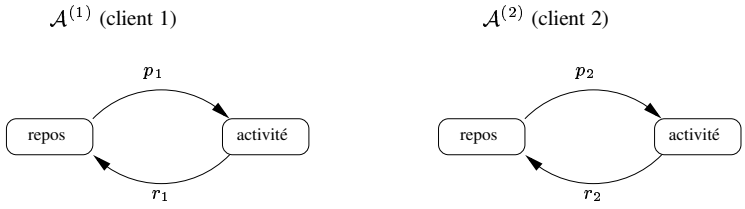


Figure 2. *Modèle d'exclusion mutuelle*

Entrée : un réseau d'automates stochastiques, et un état $x = (x^{(1)}, ..., x^{(N)})$.

Sortie : un ensemble de couples (état, probabilité) correspondant aux états accessibles depuis x dans le SAN, et la probabilité d'y accéder.

1. Initialiser l'ensemble des états à traiter $E = \{(x, 1)\}$ (au démarrage de l'algorithme, la probabilité d'être dans l'état x est 1).
Soit L l'ensemble des événements possibles depuis x, $L = EP(x)$.

2. Tant que L n'est pas vide,

– Choisir un événement $e \in L$ parmi ceux de plus haute priorité, et supprimer e de l'ensemble L.

– Initialiser le nouvel ensemble d'états accessibles $E' = \{\ \}$ (c'est l'ensemble des états que l'on peut atteindre suivant l'occurrence ou non de e depuis x).

– Pour tout $(y, p) \in E$,

- Si e n'est pas réalisable depuis y, $E' = (y, p) \cup E'$

(il n'y a pas de nouvel état accessible, on reste dans l'état y).

- Sinon (e est réalisable),

a. e n'a pas lieu avec la probabilité $1 - pt_e(y)$, d'où
$E' = (y, p(1 - pt_e(y))) \cup E'$.

b. Il faut envisager tous les cas où e a lieu, on obtient un ensemble d'états successeurs y' ainsi que la probabilité que l'on soit dans cet état ($p' = p.pt_e(y) \times$ *un produit de probabilités de routage*), et rajouter tous les couples (y', p') à E'. Le produit de probabilités de routage est obtenu en regardant chaque état local possible dans l'état y' pour les automates impliqués par l'événement e, d'après la définition 3.

– Lorsqu'on a terminé le traitement d'un événement de L, le nouvel ensemble des états à traiter est $E = E'$. On réitère alors l'étape 2 s'il reste des événements à traiter.

3. Lorsque L est vide, E contient l'ensemble des états accessibles depuis l'état x. La probabilité associée à un état donné correspond à la probabilité de transition entre x et cet état.

Algorithme 1. *Génération des états accessibles et des probabilités de transition associées*

NOTE. — Les ajouts de couples (y, p) dans l'ensemble E' qui ont lieu lors de l'étape 2 de l'algorithme 1 sont effectués de manière à éviter les redondances d'états dans E'. En effet, si l'état y apparaît déjà dans E' (élément (y, p')), il est inutile de créer une nouvelle entrée dans l'ensemble, il suffit de rajouter la priorité p à la priorité déjà existante p'. L'ajout correspond alors à modifier l'élément (y, p') en $(y, p + p')$.

Ce système est modélisé uniquement à l'aide d'événements locaux possédant des probabilités d'occurrence fonctionnelles. Le modèle est représenté par la figure 2.

Les probabilités de prise et de relâche de ressource sont respectivement λf et μ. La fonction f a pour rôle de garantir l'unicité de la ressource. Elle vaut 1 si et seulement si les deux clients sont dans l'état repos, et 0 sinon.

Nous devons également préciser les priorités de chaque événement. Dans notre cas, nous considérons que la relâche de ressource est plus prioritaire que la prise de ressource. Ainsi, il se peut que dans une même unité de temps, un client relâche la ressource, et l'autre client la prenne. Ceci ne peut pas se produire en temps continu. Nous donnons également des priorités différentes aux deux événements de prise de ressource pour pouvoir décider quel client prend la ressource lorsqu'ils la demandent tous les deux en même temps. Pour notre exemple, le premier client est plus prioritaire.

Nous avons alors les événements suivants :

– la prise de la ressource par le premier client, p_1,
avec $pt_{p_1} = \lambda f$, $prio_{p_1} = 2$, $O_{p_1} = \{\mathcal{A}^{(1)}\}$;

– la prise de la ressource par le deuxième client, p_2,
avec $pt_{p_2} = \lambda f$, $prio_{p_2} = 3$, $O_{p_2} = \{\mathcal{A}^{(2)}\}$;

– la relâche de la ressource par le premier client, r_1,
avec $pt_{r_1} = \mu$, $prio_{r_1} = 1$, $O_{r_1} = \{\mathcal{A}^{(1)}\}$;

– la relâche de la ressource par le deuxième client, r_2,
avec $pt_{r_2} = \mu$, $prio_{r_2} = 1$, $O_{r_2} = \{\mathcal{A}^{(2)}\}$.

Pour construire la chaîne de Markov équivalente à ce modèle, nous appliquons l'algorithme en partant de l'état $x = (repos, repos)$.

Le déroulement de l'algorithme peut être représenté de façon arborescente, comme illustré dans la figure 3. La racine de l'arbre correspond à l'initialisation de l'algorithme 1 (étape 1), puis nous descendons d'un niveau dans l'arbre lors du traitement d'un événement.

A un niveau donné, nous disposons donc d'un ensemble d'états, ainsi que la probabilité d'y accéder depuis l'état x. Nous regardons pour chaque état si l'événement est réalisable ou non depuis cet état, et nous générons les états successeurs éventuels, tenant compte de la probabilité d'occurrence de l'événement et des probabilités de routage éventuelles. A chaque niveau, les ensembles L et E sont mis à jour (boucle de l'algorithme 1 dans l'étape 2).

Les feuilles de l'arbre correspondent aux éléments de l'ensemble des couples (état, probabilité) correspondant aux états accessibles depuis x, ainsi que la probabilité d'y accéder depuis x (étape 3 de l'algorithme 1).

Pour notre exemple, nous codons l'état *repos* par 1 et l'état *activité* par 2, un état global est alors noté Sij, où $i = 1, 2$ est l'état du premier client, et $j = 1, 2$ est l'état du deuxième client (S fait référence au mot anglais *state* pour *état*). L'état initial nous

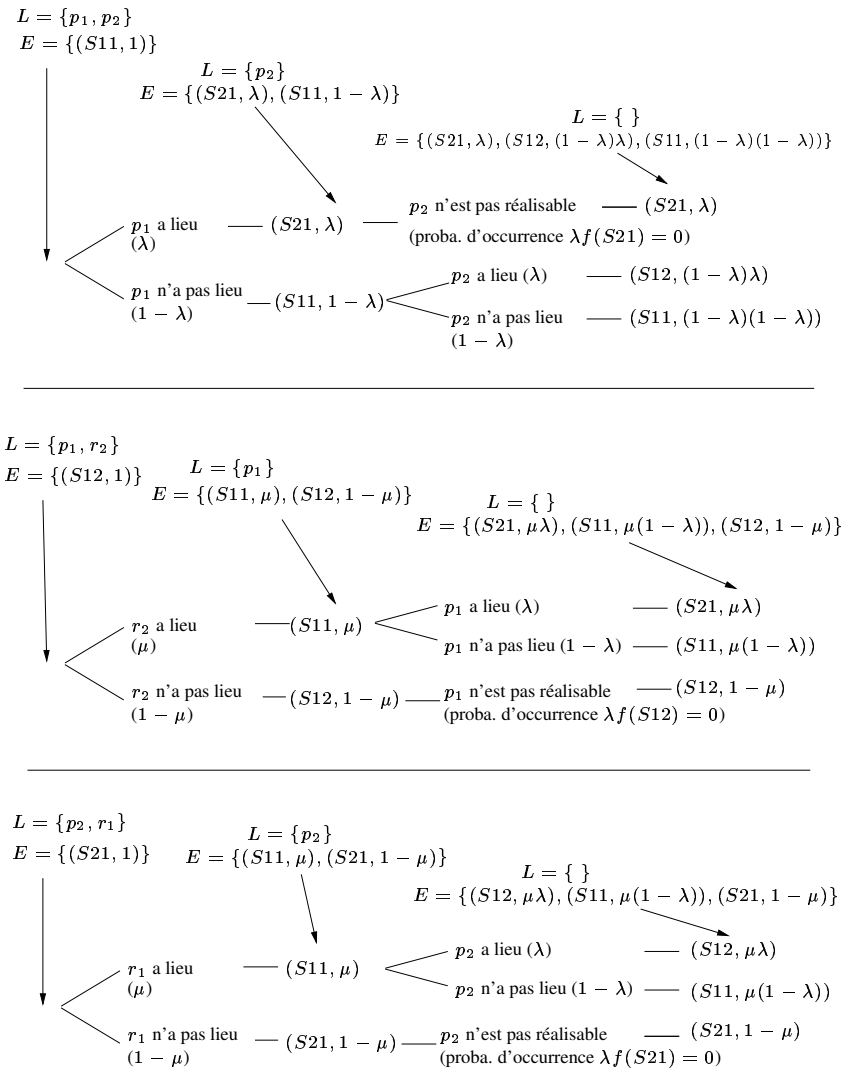


Figure 3. *Déroulement de l'algorithme 1 pour l'exemple de l'exclusion mutuelle*

donne un certain nombre d'états accessibles, nous relançons alors l'algorithme 1 à partir de ces états pour obtenir tous les états accessibles du modèle et les probabilités de transition d'un état à un autre. Nous obtenons alors les arbres de la figure 3 (un arbre par appel à l'algorithme).

4.3. *Définition de la chaîne de Markov*

Les états de la chaîne de Markov équivalente (qui définit la sémantique du réseau d'automates stochastiques) sont les états accessibles. L'ensemble de ces états a été obtenu par l'utilisation itérative de l'algorithme 1, ainsi que les probabilités de transition d'un état à un autre.

Les probabilités de transition ne dépendent que de l'état de départ, ce qui nous assure que cette chaîne est un processus sans mémoire. Pour s'assurer que nous obtenons bien une chaîne de Markov, il reste à vérifier que pour chaque état de la chaîne, la somme des probabilités partant de cet état est égale à 1 (c'est la somme des probabilités sur une ligne de la matrice de transition).

En partant de l'algorithme 1, nous montrons qu'à chaque itération, la somme des probabilités des états dans E est égale à 1. Notons $\sum E$ cette somme. La preuve est faite par récurrence.

Au départ, $E = \{(x, 1)\}$, donc $\sum E = 1$.

Supposons qu'à une étape donnée (une étape correspond au traitement d'un événement de L), nous avons $\sum E = 1$. Nous générons à partir de tout (y, p) de E un ensemble

1) $E1 = \{(y, p)\}$ si l'événement n'est pas réalisable, ou bien

2) $E2 = \{(y, p(1 - pt_e(y))), (y'_1, p.pt_e(y).\pi_1(y)), ..., (z_r, p.pt_e(y).\pi_r(y))\}$
si l'événement est réalisable.

Dans le deuxième cas, $\pi_i(y)$ ($i \in \{1, \ldots, r\}$) représente la probabilité de routage pour aller dans l'état y'_i (dans le cas d'événements synchronisant, c'est le produit des probabilités de routage dans chaque automate).

Nous avons alors

$$\sum E1 = p$$

$$\sum E2 = p(1 - pt_e(y)) + p.pt_e(y).\pi_1(y) + ... + p.pt_e(y).\pi_z(y)$$

$$\sum E2 = p(1 - pt_e(y)) + p.pt_e(y).(\pi_1(y) + ... + \pi_r(y))$$

Par définition du routage, $\pi_1(y) + ... + \pi_r(y) = 1$ donc

$$\sum E2 = p(1 - pt_e(y)) + p.pt_e(y)$$

$$\sum E2 = p$$

Dans les deux cas, la somme des probabilités de l'ensemble généré par l'élément (y, p) de E est égale à p. Le nouvel ensemble E' est l'union des ensembles générés pour chaque élément de E, donc

$$\sum E' = \sum E = 1$$

La chaîne obtenue est bien une chaîne de Markov, ce qui prouve la cohérence de la sémantique utilisée.

Reprenons l'exemple de l'exclusion mutuelle présenté précédemment. A partir des différents appels à l'algorithme, détaillés dans la figure 3, nous pouvons en déduire la chaîne de Markov équivalente (figure 4) et sa matrice de transition (états dans l'ordre $S11$, $S12$, $S21$) :

$$\begin{pmatrix} (1-\lambda)(1-\lambda) & \lambda(1-\lambda) & \lambda \\ \mu(1-\lambda) & (1-\mu) & \lambda\mu \\ \mu(1-\lambda) & \lambda\mu & (1-\mu) \end{pmatrix}$$

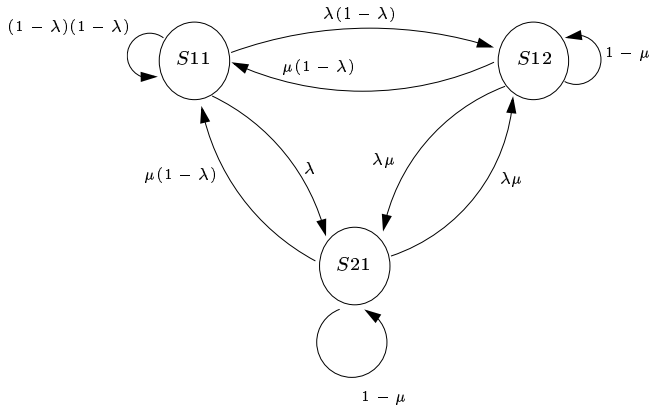


Figure 4. *Graphe de la chaîne de Markov équivalente : exemple de l'exclusion mutuelle*

5. Exemple de modélisation

Nous proposons dans cette section la modélisation d'une unique file d'attente de capacité finie C à *temps discret*, avec différentes politiques de service [GEL 98].

La file est modélisée par un unique automate $\mathcal{A}$ comportant $C+1$ états. L'état i ($i \in \{0, \ldots, C\}$) signifie qu'il y a i clients dans la file. L'état vide x_v, qui est un état fictif permettant de rendre possibles certains événements, est également utilisé dans cet automate.

Les arrivées de clients sont modélisées par l'événement local a, et les départs par l'événement local d. En temps discret, il peut y avoir durant une même unité de temps une arrivée et un départ. On n'autorise en revanche pas plusieurs départs ou arrivées dans la même unité de temps dans cet exemple par souci de simplification.

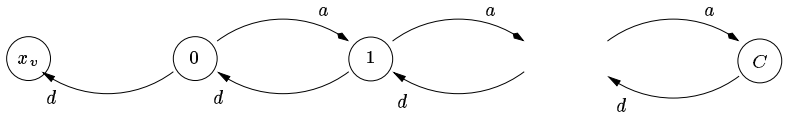
Il faut décider l'ordre dans lequel ont lieu les arrivées et le service des clients [GEL 98] :

– **early arrival system** (départs tardifs) – dans ce système, les départs ont lieu après les arrivées. Ainsi, nous pouvons avoir dans une même unité de temps une arrivée, puis un départ ;

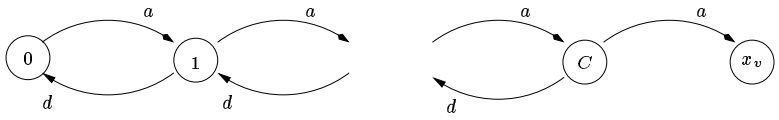
– **late arrival system** (arrivées tardives) – dans ce cas, les départs ont lieu au début de l'unité de temps, et les arrivées à la fin.

Dans le cas de départs tardifs, l'événement a (arrivée) doit donc être plus prioritaire que l'événement d (départ). En revanche, lorsque les arrivées sont tardives, c'est d qui est plus prioritaire que a.

Ce modèle de file d'attente à temps discret est illustré dans la figure 5 pour chacun des deux cas. Le SAN est réduit à un unique automate pour chaque cas.



a. Early arrival system (départs tardifs)



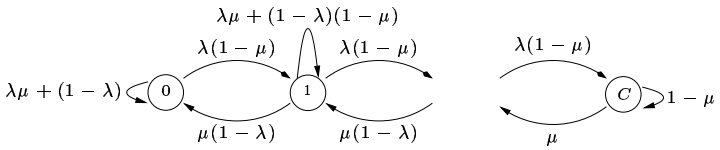
b. Late arrival system (arrivées tardives)

Figure 5. *File d'attente à temps discret – SAN*

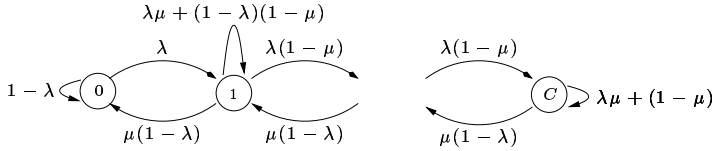
Dans le modèle a (départs tardifs), une transition est ajoutée de l'état 0 vers l'état vide x_v. Cette transition permet de rendre possible le départ d'un client même lorsque la file est vide. L'événement d n'est cependant pas réalisable lorsqu'il n'y a pas de client, mais il est rendu réalisable lorsqu'une arrivée d'un client se produit avant (dans la même unité de temps). On autorise ainsi l'occurrence «simultanée» d'une arrivée et d'un départ lorsque la file est vide dans ce modèle. Pour assurer les départs tardifs, l'événement a est plus prioritaire que l'événement de départ d dans ce modèle. Notez qu'aucune transition ne part de l'état vide x_v vu que ce n'est pas un état réel de l'automate, mais un simple état fictif qui permet d'autoriser cette simultanéité des événements. L'automate n'est jamais dans l'état x_v.

Dans le modèle b (arrivées tardives), c'est le phénomène inverse. On autorise une arrivée dans l'état C correspondant à une file d'attente pleine, car l'arrivée peut avoir

lieu si un client est servi durant la même unité de temps. Dans ce cas, une place est libérée et un nouveau client peut arriver. Pour ce modèle, ce sont les départs qui sont plus prioritaires que les arrivées, les arrivées sont donc tardives.



a. Early arrival system (départs tardifs)



b. Late arrival system (arrivées tardives)

Figure 6. *File d'attente à temps discret – Graphe de la chaîne de Markov*

La figure 6 représente le graphe des chaînes de Markov associées à ces deux modèles. Les valeurs λ et μ correspondent respectivement aux probabilités d'occurrence des événements a et d. Les probabilités de transition pour passer d'un état à un autre sont calculé à l'aide de l'algorithme 1 présenté dans la section précédente.

6. Conclusions et perspectives

Nous avons présenté un formalisme de SAN à temps discret, et donné un algorithme de génération de la chaîne de Markov équivalente à partir du SAN. Cet algorithme procède en générant l'ensemble des états accessibles depuis un état de départ donné, ainsi que les probabilités de transition associées à chacun de ces états. C'est un algorithme de génération de l'espace des états accessibles pour des modèles à temps discret, qui détecte et traite au passage les événements du modèle en conflit. La sémantique du modèle en cas de conflit a été détaillée. Pour cela, une priorité doit être associée à chaque événement, et le comportement du modèle est déterminé à l'aide de ces priorités lorsqu'il y a un conflit (dans ce cas, la spécification stochastique du modèle est insuffisante).

Un problème ouvert reste l'attribution des priorités, qui doit être effectuée par l'utilisateur pour donner un sens au modèle. Une attribution automatique pourrait être envisagée, consistant par exemple à fixer un ordre de priorité sur les automates et sur certains événements, puis à générer automatiquement la priorité globale de chaque événement à partir de ces informations.

L'étude de ce nouveau formalisme reste cependant un domaine de recherche pleinement ouvert.

Une première piste de recherche est basée sur l'expression de la matrice générateur de la chaîne de Markov. En temps continu, le **format tensoriel** est largement utilisé pour représenter la matrice sous forme compacte et structurée. Des travaux similaires ont été effectués pour des réseaux de Petri tels que l'évolution du processus de marquage est représentée par une chaîne de Markov à temps discret [SCA 98]. Nous envisageons également de représenter la matrice générateur d'un SAN à temps discret à l'aide d'une expression tensorielle. Ceci nécessite la définition de nouveaux opérateurs tensoriels tenant compte des priorités, ce qui n'était pas le cas dans [SCA 98] (aucune notion de priorité n'est introduite). Chaque élément du générateur ainsi obtenu doit alors correspondre à une probabilité calculée par l'algorithme de génération de la chaîne de Markov.

Enfin, un travail important reste à faire sur les **méthodes de résolution** sur ce formalisme. La représentation du générateur sous forme d'expression tensorielle devrait permettre d'adapter les méthodes classiques, et notamment d'utiliser l'algorithme du shuffle [FER 98]. Cependant, la présence des priorités et la modification probable des opérateurs tensoriels risque d'induire des changements par rapport aux méthodes utilisées sur les réseaux d'automates stochastiques à temps continu. Ceci constitue donc un domaine de recherche totalement ouvert, mais les travaux sur le sujet ne pourront commencer que lorsque la définition d'un générateur en format tensoriel aura été obtenue.

7. Bibliographie

[ATI 92] ATIF K., *Modélisation du parallélisme et de la synchronisation*, Institut National Polytechnique de Grenoble, France, 1992.

[BUC 94] BUCHHOLZ P., « A Class of Hierarchical Queueing Networks and Their Analysis », *Queueing Systems*, vol. 15, 1994, p. 59-80.

[BUC 00] BUCHHOLZ P., KEMPER P., « Efficient computation and representation of large reachability sets for composed automata », *Proc. 5th workshop on Discrete Event Systems*, Ghent, Belgium, August 2000.

[CIA 94] CIARDO G., GERMAN R., LINDEMANN C., « A Characterization of the Stochastic Process Underlying a Stochastic Petri Net », *IEEE Transactions on Software Engineering*, vol. 20, n° 7, 1994, p. 506-515.

[CIA 95] CIARDO G., « Discrete-time Markovian stochastic Petri nets », STEWART W. J., Ed., *Computation with Markov Chains*, p. 339-358, Raleigh, NC, USA, Jan 1995.

[CIA 99] CIARDO G., MINER A., « Efficient reachability set generation and storage using decision diagram », *In proc. 20th int. Conf. Application and Theory of Petri Nets, LNCS 1639, Springer*, 1999.

[DON 94] DONATELLI S., « Superposed Generalized Stochastic Petri nets : definition and efficient solution », VALETTE, R., Ed., *Lecture Notes in Computer Science ; Application and Theory of Petri Nets 1994, Proceedings 15th International Conference, Zaragoza, Spain*,

vol. 815, Springer-Verlag, 1994, p. 258-277.

[FER 98] FERNANDES P., *Méthodes Numériques pour la solution de systèmes Markoviens à grand espace d'états*, Institut National Polytechnique de Grenoble, France, 1998.

[GEL 98] GELENBE E., PUJOLLE G., *Introduction to Queueing Networks, second edition*, John Wiley & Sons, 1998.

[GUS 03] GUSAK O., DAYAR T., FOURNEAU J.-M., « Discrete-time stochastic automata networks and their efficient analysis », *Performance Evaluation*, vol. 53, n° 1, 2003, p. 43–69.

[HIL 95] HILLSTON J., « Compositional Markovian Modelling Using a Process Algebra », STEWART W., Ed., *Numerical Solution of Markov Chains.*, Kluwer, 1995.

[HOL 87] HOLLIDAY M., VERNON M., « A Generalized Timed Petri Net Model for Performance Analysis », *IEEE Transactions on Software Engineering*, vol. 13, n° 12, 1987, p. 1297-1310.

[KEM 96] KEMPER P., « Reachability analysis based on structured representations », J. BILLINGTON W. R., Ed., *Proc. 17th int. conf. Application and Theory of Petri Nets, Springer LNCS 1091*, 1996, p. 269-288.

[MAR 95] MARSAN M., BALBO G., CONTE G., DONATELLI S., FRANCESCHINIS G., *Modelling with generalized stochastic Petri nets*, John Wiley & Sons, 1995.

[MOL 85] MOLLOY M., « Discrete Time Stochastic Petri Nets », *IEEE Transactions on Software Engineering*, vol. 11, n° 4, 1985, p. 417-423.

[PLA 84] PLATEAU B., *De l'Evaluation du parallélisme et de la synchronisation*, Université de Paris XII, Orsay (France), 1984.

[PLA 91] PLATEAU B., ATIF K., « Stochastic automata networks for modelling parallel systems. », *IEEE Transactions on Software Engineering*, vol. 17, n° 10, 1991, p. 1093-1108.

[SCA 98] SCARPA M., BOBBIO A., « Kronecker Representation of Stochastic Petri Nets with Discrete PH Distributions », *International Computer Performance and Dependability Symposium IPDS98, Duke University, Durham, NC*, IEEE Computer Society Press, September 1998.

[STE 94] STEWART W. J., *An introduction to numerical solution of Markov chains*, Princeton University Press, New Jersey, 1994.

[WOO 94] WOODWARD M. E., *Communication and Computer Networks : Modelling with Discrete-time Queues*, IEEE Computer Society Press, Los Alamitos, CA, 1994.

[ZIJ 96] ZIJAL R., CIARDO G., « Discrete Deterministic and Stochastic Petri Nets », rapport, 1996, ICASE Technical Report 96-72, Institute for Computer Applications in Science and Engineering, NASA/Langley Research Center, Hampton, VA.

[ZUB 91] ZUBEREK W., « Timed Petri Nets, Definitions, Properties, and Applications. », *Microelectronics and Reliability*, vol. 31, n° 4, 1991, p. 627-644.

Article reçu le 27 octobre 2003
Version révisée le 24 avril 2004
Rédacteur responsable : Patrice Moreaux

Anne Benoit *est ingénieur ENSIMAG (Ecole Nationale Supérieure d'Informatique et de Mathématiques Appliquées de Grenoble). Elle a obtenu sa thèse en 2003 à l'Institut National Polytechnique de Grenoble (INPG), effectuée au laboratoire Informatique et Distribution (I.D.). Elle est maintenant assistante de recherche à l'université d'Edimbourg. Ses travaux portent principalement sur l'évaluation des performances des systèmes à grand espace d'états, en utilisant des modèles de réseaux d'automates stochastiques et des algèbres de processus. Elle s'intéresse également à la programmation parallèle haut niveau, et notamment à l'utilisation de squelettes algorithmiques, ainsi qu'aux applications pour grilles d'ordinateurs.*

Brigitte Plateau *est professeur à l'ENSIMAG (Institut National Polytechnique de Grenoble) depuis 1988, responsable du projet (CNRS-INPG-INRIA-UJF) Apache (Algorithmique, Programmation Parallèle et pArtage de CHargE) et directeur du laboratoire I.D. (Informatique et Distribution) UMR 5132 (CNRS-INPG-INRIA-UJF). Son domaine de recherche est centré sur l'étude des performances des systèmes informatiques, et plus particulièrement des performances des systèmes répartis et parallèles. Elle s'intéresse aux techniques de modélisation par réseaux d'automates, à la théorie de files d'attente, à l'algorithmique distribuée et aux calculateurs parallèles ainsi qu'à leur programmation et leur observation.*

William J. Stewart *est professeur d'informatique à la «North Carolina State University». Ses centres de recherche incluent la théorie et la solution numérique des chaînes de Markov, l'évaluation des performances des ordinateurs et des systèmes de communication, et l'algèbre linéaire numérique. Dr. Stewart a initié une série de conférences internationales sur la solution numérique des chaînes de Markov, et il a publié de nombreux ouvrages et papiers sur ce sujet.*