Approches formelles pour l'analyse de la performabilité des systèmes communicants mobiles : Applications aux réseaux de capteurs sans fil

Robert Abo

le cnam

ECOLE DOCTORALE EDITE

Centre d'Etudes et de Recherche en Informatique et Communications

THÈSE présentée par :

Robert ABO

soutenue le : 06 décembre 2011

pour obtenir le grade de : **Docteur du Conservatoire National des Arts et Métiers**

Discipline / Spécialité : **Informatique**

Approches Formelles pour l'Analyse de la Performabilité des Systèmes Communicants Mobiles

Applications aux réseaux de capteurs sans fil

THÈSE dirigée par :

M.	BARKAOUI Kamel	Professeur au CNAM

RAPPORTEURS :

M.	DJOUANI Karim	Professeur à l'Université Paris Est-Créteil
Mme	PALAMIDESSI Catuscia	Directrice de Recherche au LIX - INRIA

JURY :

M.	BARKAOUI Kamel	Professeur au CNAM
Mme	BOUMERDASSI Selma	Maître de conférences au CNAM
M.	DJOUANI Karim	Professeur à l'Université Paris Est-Créteil
M.	MONSUEZ Bruno	Professeur à ENSTA ParisTech
Mme	PALAMIDESSI Catuscia	Directrice de Recherche au LIX - INRIA
M.	SEKHRI Larbi	Professeur à l'Université d'ORAN Sénia

Approches Formelles pour l'Analyse de la Performabilité des Systèmes Communicants Mobiles

Applications aux réseaux de capteurs sans fil

Ad augusta. Per angusta.
Victor Hugo, Hernani, acte IV, scène 3.

Remerciements

Je souhaite remercier toutes les personnes qui m'ont aidé à réaliser mon projet de thèse de doctorat.

Au membres du jury qui ont consacré de leur temps à la lecture de mon manuscrit, à la rédaction des rapports, et qui m'ont auditionné le mardi 6 décembre 2011 : Mme Catuscia Palamidessi du LIX-INRIA, Mme Selma Boumerdassi du CNAM, M. Bruno Monsuez de l'ENSTA ParisTech, M. Karim Djouani de l'Université Paris Est-Créteil, M. Larbi Sekhri de l'Université d'Oran Sénia.

Mes remerciements les plus chaleureux sont adressés à M. Kamel Barkaoui avec qui je travaille depuis de nombreuses années. Il m'a permis de réaliser mon projet de thèse de doctorat en acceptant de devenir mon directeur de thèse et en me permettant de le financer en étant ATER pendant deux ans.

Résumé

La complexité croissante des systèmes demande une vérification rigoureuse de leurs paramètres de performabilité, c'est-à-dire de sûreté de fonctionnement et de performance, d'une part pour s'assurer que ces systèmes sont sûrs, et ne présentent pas de danger pour leurs utilisateurs, et d'autre part pour vérifier que leur performance sont en adéquation avec la spécification de leurs exigences. Parmi les techniques d'analyse moderne, les techniques de model checking présentent de nombreux avantages sur d'autres techniques comme l'expérimentation ou les techniques analytiques, dont leur exhaustivité et leur automaticité. Ces techniques formelles consistent à modéliser le système à analyser, à spécifier ses exigences sous la forme de propriétés écrites en logique temporelle, avant de vérifier que ce modèle vérifie ces propriétés. Cette technique est relativement classique, mais lorsqu'on s'intéresse aux systèmes mobiles, on est confronté au problème de devoir modéliser la mobilité des systèmes à l'aide d'un formalisme de bas niveau à états-transitions support des techniques de model checking.

Nous nous intéressons à l'analyse des exigences de performabilité des systèmes communicants mobiles par model checking. Nous modélisons ces systèmes à l'aide d'un formalisme de haut niveau issu du π-calcul, permettant de considérer des comportements stochastiques, temporels, déterministes, ou indéterministes. Cependant, dans le π-calcul, la primitive de communication de base des systèmes est la communication en point-à-point synchrone. Or, les systèmes mobiles, qui utilisent des réseaux sans fil, communiquent essentiellement par diffusion locale. C'est pourquoi, dans un premier temps, nous définissons la communication par diffusion dans le π-calcul, afin de mieux modéliser les systèmes que nous étudions. Nous proposons d'utiliser des versions probabilistes et stochastiques de l'algèbre que nous avons défini, pour permettre des études de performance. Nous en définissons une version temporelle permettant de considérer le temps dans les modèles. Mais l'absence d'outils d'analyse des propriétés sur des modèles spécifiés en une algèbre issue du π-calcul est un obstacle majeur à notre travail. La définition de règles de traduction en langage Prism, nous permet de traduire nos modèles, en modèles de bas niveau supports du model checking, à savoir des chaînes de Markov à temps discret, à temps continu, des automates temporisés, ou des automates temporisés probabilistes. Nous avons choisi l'outil Prism car, à notre connaissance, dans sa dernière version, il est le seul outil à supporter les formalismes de bas niveau que nous venons de citer, et ainsi de réaliser des études de performabilité complètes. Cette façon de procéder nous permet de pallier à l'absence d'outils d'analyse pour nos modèles. Par la suite, nous appliquons ces concepts théoriques aux réseaux de capteurs sans fil mobiles.

Abstract

The increasing complexity of systems requires a rigorous audit of their performability parameters, that is to say, dependability and performance, firstly to ensure that these systems are safe and present no danger to their users, and secondly to ensure that their performance is in line with the specification of their requirements. Among the techniques of modern analysis, model checking techniques have many advantages of other techniques, such as experimental or analytical techniques including completeness and automaticity. These formal techniques are to model the system to analyze, specify its requirements in the form of properties written in temporal logic, before checking that the model satisfies these properties. This technique is relatively conventional, but when looking at mobile systems, we are confronted with the problem of modeling mobility using a low-level formalism based on labelled transition systems to support model checking techniques.

We are interested in analyzing the performability requirements of mobile communication systems by using model checking techniques. We model these systems using a high-level formalism derived from the π-calculus, for considering stochastic, timed, deterministic or indeterministic behaviors. However, in the π-calculus, the basic communication primitive of systems is the synchronous point-to-point communication. But mobile systems that use wireless networks mostly communicate by local broadcast. Therefore, we first define the broadcast communication in the π-calculus to better model the systems we study. We propose to use probabilistic and stochastic versions of the calculus we have defined to allow performance studies. We define a temporal version to consider time in the models. The lack of tools for analyzing properties of models specified with π-calculus is a major obstacle to our work and its objectives. The definition of translation rules into the Prism language allows us to translate our models into low-level models which can support model checking, namely discrete-time, or continuous-time Markov chains, timed automata, or probabilistic timed automata. We chose the Prism model checker because, in our best knowledge, in its latest version, it is the only tool that supports the low-level formalisms that we have previously cited, and thus, makes it possible our performability studies. This approach allows us to overcome the lack of model checkers for our models. Subsequently, we apply these theoretical concepts for mobile wireless sensor networks.

Table des matières

Liste des tableaux

Table des figures

Avant-propos

Cette thèse porte sur les *approches formelles d'analyse de la performabilité des systèmes communicants mobiles appliquées aux réseaux de capteurs sans fil* (sous entendu *mobiles*). L'analyse de la performabilité associe la vérification des critères de sûreté de fonctionnement des systèmes, avec l'évaluation de leurs performances. Parmi toutes les techniques existantes, nous nous intéressons plus précisément à l'analyse des exigences de performabilité par les techniques formelles du *model checking*. Ces techniques consistent à modéliser le comportement du système à analyser, à spécifier ses exigences de performabilité sous la forme de propriétés écrites en une logique temporelle adaptée, avant de vérifier automatiquement à l'aide d'un outil de vérification, appelé un *model checker*, que *ce* modèle respecte *ces* exigences. Dans le cas contraire, un contre-exemple est fourni qui montre l'origine du problème. Plusieurs techniques de model checking se distinguent selon que l'on considère que le système à analyser a un comportement aléatoire ou non, a un comportement déterministe ou non, ou encore que le temps a une influence sur son comportement ou non, etc. Il existe plusieurs types de logiques temporelles permettant de spécifier les différents types d'exigences à analyser. Les techniques de model checking ont les avantages de l'exhaustivité de la vérification d'une exigence par rapport à un modèle, couplée à son automaticité à l'aide d'un outil et de la désignation d'éventuels contre-exemples. Elles ont l'inconvénient d'utiliser des algorithmes dont la complexité en temps et en espace mémoire des machines peut s'avérer être un handicap à leur utilisation. Différentes techniques existent ou sont étudiées, car l'activité de la recherche dans ce domaine est intense, pour réduire ces problèmes que l'on résume par l'*explosion combinatoire des états*. Par ailleurs, la puissance des machines modernes fait que le model checking, qui est apparu au début des années 1980, est aujourd'hui utilisé dans l'industrie informatique tant matérielle que logicielle. Nous consacrons une annexe à la présentation des techniques de model checking. Par ailleurs, les autres méthodes d'analyse des exigences de performabilité sont présentées succinctement au chapitre 1.

La première étape de l'analyse des exigences de performabilité d'un système par model checking est la modélisation du comportement de ce système par un formalisme suffisamment expressif pour pouvoir considérer tous les paramètres qui influencent ce comportement. Une caractéristique comportementale majeure des systèmes auxquels nous nous intéressons dans cette thèse, celle qui est au cœur de notre travail, est leur *mobilité*. Les systèmes qui nous intéressent sont mobiles, c'est-à-dire que leurs moyens de communication se reconfigurent en fonction de leurs interactions avec leur environnement, ou bien qu'ils se déplacent dans l'espace. Dans cette thèse, nous nous focalisons sur l'aspect reconfiguration des liens de communication des systèmes pour définir leur mobilité. Nous expliquons au chapitre 2 que cet aspect de la mobilité n'est pas complètement déconnectée de l'aspect qui définit la mobilité comme des déplacements dans l'espace. Parmi les autres caractéristiques des systèmes, nous nous intéressons à leurs comportements stochastiques et temporels. L'aspect stochastique caractérisent les modèles aléatoires, utiles lorsque l'on veut modéliser des interactions entre un système et son environnement dont l'origine n'est pas connue, ou utiles lorsque l'on veut abstraire un comportement réel difficilement modélisable. La modélisation stochastique per-

met des études de performance. L'aspect temporel, quant à lui, considère le temps dans les modèles. Le temps n'est pas être un paramètre anodin, voire inutile : de nombreux protocoles de communication utilisent des *timeout* pour éviter les situations de blocage permanent. Il faut les considérer pour analyser la performabilité de ces systèmes rigoureusement. Pour modéliser les systèmes qui nous intéressent, nous avons choisi comme formalisme de base le *π-calcul* [MPW92] qui, à l'origine est un modèle de calculabilité, mais peut être utilisée comme une algèbre de processus orientée mobilité des systèmes. La mobilité des systèmes s'y exprime par la reconfiguration de leurs liens de communication (on parle de *topologie de communication*) au gré de leurs interactions avec leur environnement. C'est justement l'aspect de la mobilité des systèmes qui nous intéresse. Mais le π-calcul de base a des inconvénients pour modéliser les systèmes : il ne permet, ni de modéliser différents types de communication puisqu'il ne considère que les communications en point à point, ni les aspects stochastiques et temporels. Pour ce faire, il doit être étendu pour modéliser les diffusions (broadcast), ou encore pour permettre de modéliser des comportements stochastiques, ou des comportements influencés par le temps. Ces extensions font l'objet de la première partie théorique de cette thèse. La deuxième étape consiste à traduire les modèles en un formalisme exploitable par l'outil de vérification Prism que nous avons choisi pour analyser les exigences de performabilité des systèmes. Enfin, nous appliquons ces concepts théoriques aux réseaux de capteurs sans fil. Cependant les résultats théoriques de cette thèse ne dépendent nullement de ce champ d'application et nous aurions aussi pu l'appliquer à l'étude de la performabilité des *réseaux WiFi* [IEE07], des *réseaux mobiles ad hoc* (MANETs), des *réseaux véhiculaires* (VANETs) [Col10] qui constituent des *systèmes de transport intelligents*, etc. D'une façon plus générale, les résultats théoriques de cette thèse sont applicables à tous les systèmes qui communiquent par des protocoles de communication sans fil supports de leur mobilité.

Il est clair que la mobilité des systèmes communicants est un thème majeur de cette thèse. La première raison qui nous a poussé à nous intéresser à ce sujet est, qu'aujourd'hui, les systèmes communicants mobiles sont omniprésents et qu'ils appartiennent à notre quotidien. Notre société est fondée sur une connexion permanente de l'homme à une machine, qui lui permet d'accéder à des services à distance, d'effectuer des achats, de programmer des opérations pour sa santé, de contrôler à distance des appareils de gestion de l'énergie, de compenser des situations de handicap, de travailler à distance, ou encore de développer sa vie sociale. Parmi ces systèmes, beaucoup sont critiques. Il n'est donc pas inintéressant de vouloir analyser leur exigences de performabilité dans le but d'améliorer leur qualité de service. Nous rappelons au paragraphe suivant l'émergence de la mobilité des systèmes dans l'informatique contemporaine.

Émergence de la mobilité des systèmes en informatique

L'histoire de l'informatique nous apprend que ses débuts se caractérisent par l'utilisation de *systèmes centralisés*, constitués par un seul calculateur grâce auquel tous les utilisateurs effectuent des traitements par lots sur des données. Cette vision centralisée de l'informatique a évolué grâce aux progrès technologiques, tant matériels que logiciels, pour offrir plus de souplesse d'utilisation, plus de disponibilité, mais aussi un meilleur rendement à un moindre coût.

Plus particulièrement, l'essor des ordinateurs personnels et des réseaux de communication a permis l'émergence de l'*informatique distribuée*. Un système distribué est une collection de machines autonomes connectées à l'aide d'un réseau de communication. Chaque machine exécute des composants logiciels et utilise un intergiciel qui coordonne l'activité des composants de telle sorte qu'un utilisateur perçoive le système distribué global comme un système unique

intégré. La distribution du système est généralement transparente pour ses utilisateurs. Un système distribué est ouvert et extensible, hétérogène car ses composants logiciels peuvent être écrits en différents langages. Il doit permettre l'accès et le partage des ressources matérielles et logicielles, ainsi que des données. Il doit aussi être tolérant aux pannes. Plusieurs standards ont été définis pour normaliser le développement d'applications logicielles pour les systèmes distribués.

Au début des années 1990, l'apparition des ordinateurs portables et des réseaux locaux sans fil ont conduit les chercheurs à devoir résoudre les problèmes posés par le développement d'un système distribué composés de clients mobiles. Comment assurer la continuïté des services lorsqu'un client peut se déplacer dans l'espace ? Un système distribué devient mobile lorsqu'il permet une transition continue de ses services entre ses espaces d'utilisation [Kat95]. Ces de la résolution de ces problèmes qu'est née l'*informatique mobile*. Bien que de nombreux principes de base de la conception des systèmes distribués continuaient à s'appliquer, quatre contraintes majeures liées à la mobilité des ordinateurs ont contraint les ingénieurs à développer des techniques spécifiques : la variation imprévisible de la qualité du réseau, la confiance et la robustesse faibles des éléments mobiles, les limitations des ressources locales imposées par des contraintes de poids et de taille et l'économie d'énergie en quantité finie sembarquée dans une pile.

A partir du début des années 2000, de distribuée et mobile, l'informatique évolue vers l'omniprésence. L'*informatique ubiquitaire*[1] est une vision de l'informatique introduite par Mark Weiser (1952-1999) au début des années 1990 alors qu'il était le directeur de la recherche de Xerox à Palo Alto [Wei91, Wei93]. L'informatique ubiquitaire représente la troisième vague technologique de l'informatique, après la vague des systèmes centraux, puis celle des ordinateurs personnels. Alors que ces deux dernières se caractérisent respectivement par l'utilisation d'une seule machine par plusieurs utilisateurs et par l'utilisation d'une machine par un seul utilisateur, l'informatique ubiquitaire se caractérise par un grand nombre, voire un très grand nombre d'entités informatiques, souvent de très petite taille, embarquées dans des objets d'usage quotidien voire à l'intérieur de notre corps, agissant au service d'un seul utilisateur sans que celui-ci en soit nécessairement conscient. L'informatique ubiquitaire « informe sans demander, ni qu'on se focalise sur elle, ni l'attention des utilisateurs » [WB96]. Une machine ubiquitaire est entièrement dédiée au service des utilisateurs. Elle doit pouvoir être utilisée sans attention, ni effort particulier. Par extension, un système ubiquitaire est un ensemble homogène d'entités ubiquitaires, qui interagissent et qui communiquent à l'aide de liaisons sans fil. La raison d'être d'un tel système est d'offrir un accès transparent aux ressources qu'il abrite depuis n'importe où, à n'importe quel moment et de n'importe quelle façon. L'agrégation de l'ensemble des systèmes ubiquitaires forme l'*ordinateur ubiquitaire global* qui est la vision futuriste des systèmes communicants mobiles.

L'*intelligence ambiante* est le prolongement technologique naturel de l'informatique ubiquitaire, puisque dans cette vision de l'informatique, une multitude d'objets courants ont des capacités ubiquitaires. Ces entités sont connectées à l'aide de réseaux sans fil, elles communiquent entre-elles et elles peuvent coordonner leurs actions pour servir leurs utilisateurs. La technologie se miniaturise, leur intégration dans les objets s'accroît, ils se fondent dans notre environnement. La *réalité augmentée* prolonge et complètent la capacité d'ubiquité

[1] Les auteurs anglais parlent d'*ubiquitous computing*. L'adjectif *ubiquitous* qui signifie « omniprésent » en français, est généralement traduit par « ubiquitaire ». L'appellation « informatique pervasive » est souvent employée comme synonyme d'« informatique ubiquitaire ». De l'avis de certains auteurs, elle fait plus référence aux aspects techniques, notamment aux supports matériels et logiciels, qui sont nécessaires à la mise en œuvre de l'omniprésence des services rendus par les machines.

des machines modernes. Dans le cadre de la réalité augmentée, l'utilisateur pourra évoluer dans un monde où apparaissent des éléments réels complétés par des éléments virtuels. Un autre aspect de l'intelligence ambiante est l'*internet des choses*. L'idée fondamentale est de connecter tous les objets divers et variés qui nous entoure à Internet, ceux que nos sens nous permettent de percevoir et les autres, par exemple des objets si petits que l'on parle de « poussières électroniques ». Ces entités de très petites tailles possèdent tout les équipements matériels et logiciels nécessaires à leur connexion en réseau. Une nouvelle dimension de l'informatique ubiquitaire se profile, où l'homme agit dans un espace à mi-chemin entre le réel et le virtuel.

Positionnement du sujet et contributions

Nous voulons analyser la performabilité des systèmes mobiles par les techniques formelles du model checking. Ces systèmes ont deux caractéristiques : ils communiquent en s'échangeant des informations et ils sont mobiles dans le sens où leur capacité de communication n'est pas statique, mais au contraire, évolue en fonction de leurs interactions avec leur environnement. Leur mobilité influence leur capacité à communiquer. C'est une évidence : un exemple simple est celui d'un utilisateur qui peut se connecter, ou non, à internet avec son ordinateur portable en fonction de la distance qui le sépare d'une borne WiFi. Un autre exemple est celui d'un réseau de capteurs sans fil mobile qui n'est plus opérant, dès lors que les nœuds ne peuvent plus communiquer parce qu'ils sont trop éloignés les uns des autres.

Parmi les techniques d'analyse des propriétés d'un système, les techniques du model checking sont largement utilisées. De nombreux outils d'analyse sont librement disponibles : SPIN [Spi] qui permet d'analyser des modèles spécifiés en langage PROMELA (*Protocol Meta Language*) [PROb], PRISM [PRIb,HKNP06,KNP11], UPPAAL [UPPa,GBL06,LPY97], UPPAAL-PRO [UPPb], etc. Ces outils modélisent les systèmes par des formalismes à états-transitions étiquetés, difficiles à utiliser pour modéliser des systèmes mobiles et de trop bas niveau pour être facilement exploitables par un utilisateur. Pourtant, l'analyse par model checking des exigences de performabilité des systèmes mobiles utilisent nécessairement ces formalismes. Par ailleurs, le π-calcul est une algèbre de processus permettant de modéliser des systèmes mobiles. Les algèbres de processus sont des langages textuels, concis, précis, formels, modélisant un système par la composition parallèle des modèles de ses composants. Mais, à notre connaissance, au moment de la rédaction de cette thèse, il n'existe pas d'outils de vérification permettant des analyses de performabilité directement sur des modèles spécifiés en π-calcul. Il faut cependant citer deux outils de vérification : *The Mobility Workbench* (MWB) [VM94] et *Mobility Model Checker* (MMC) [YRS04]. Ces outils supportent des modèles de systèmes en π-calcul. Mais ils ne permettent pas des analyses de performance et par suite, de performabilité. Nous proposons d'apporter les contributions suivantes à cet axe de recherche :

- Le π-calcul n'est pas un langage de modélisation des systèmes communicants. C'est un formalisme destiné à l'origine à formaliser la notion de calculabilité. Il ne modélise qu'un seul type de communication entre sous-systèmes : les communications en point-à-point synchrones. La diffusion de messages n'y est pas définie. Pourtant c'est un mécanisme de communication majeur des systèmes mobiles qui utilisent les communications sans fil. Utiliser le π-calcul pour modéliser les communications des systèmes mobiles revient à devoir faire des restrictions de spécification, engendrant des *sous-spécifications*. Nous proposons une version du π-calcul où les communications peuvent être par diffusion et non plus seulement en point à point.
- Nous voulons prendre en compte le temps dans les modèles des systèmes communicants

mobiles. Pour ce faire nous définissons une extension temporisée du π-calcul dont la sémantique s'exprime à l'aide d'*automates temporisés*. De la même façon, pour modéliser les systèmes temporels présentant un comportement probabiliste et indéterministe, nous proposons une extension probabiliste et temporisée du π-calcul.

- Nous nous intéressons aux *systèmes à événements finis*, *clos* et *finis*. La première caractéristique signifie que le comportement de ces systèmes est modélisable par un formalisme à états-transitions étiquetés : chaque état représente une configuration particulière du système et un comportement particulier de ce système est représenté par un ensemble de transitions entre états appelé un chemin. Un système *clos* est un système qui ne communique pas avec son environnement. Enfin, un système *fini* est modélisable par un formalisme à états-transitions fini. Deux types de formalismes sont utilisés : un formalisme de haut niveau issu du π-calcul pour décrire le système mais qui ne permet pas le model checking et un formalisme de bas niveau permettant l'analyse de la performabilité du système, mais trop abstrait pour permettre directement la modélisation des systèmes, en particulier leur mobilité. Les formalismes de bas niveau que nous utilisons sont les chaînes de Markov à temps discret, ou à temps continu, les automates temporisés et les automates temporisés probabilistes. La première étape consiste à établir un lien entre le formalisme de haut niveau issu du π-calcul et le formalisme de bas niveau à états-transitions étiquetées. Se faisant, il est possible d'établir une correspondance bijective, ou *isomorphisme*, entre un modèle de haut niveau et un modèle de bas niveau exprimés avec ces deux formalismes différents qui modélisent pourtant le même système. Les liens bijectifs entre les formalismes de haut et de bas niveau sont établis par des *règles de traduction*. Plus précisément, devant l'absence d'outils de vérification des modèles spécifiés en π-calcul, nous proposons d'utiliser l'outil PRISM pour ce faire. Les modèles de bas niveau sont exprimés en langage PRISM et des règles de traduction permettent de traduire un modèle de haut niveau spécifié en π-calcul étendu, en un formalisme de bas niveau exprimé en langage PRISM et modélisant des chaînes de Markov, ou des automates temporisés probabilistes . Nous avons choisi l'outil PRISM, parce que depuis sa dernière version, parue en juillet 2011, cet outil permet de modéliser des automates temporisés probabilistes et d'analyser les propriétés temporelles des systèmes. Ce qui permet de prendre en compte désormais le facteur temps dans les comportements des systèmes où il a de l'influence, voire d'étudier le comportement temps réel des systèmes communicants mobiles. L'outil UPPAAL-PRO aurait pu être un autre candidat.
- Les aspects communication et mobilité des systèmes que nous étudions sont considérés dans deux modèles différents : le *modèle des communications* et le *modèle de mobilité*, dont la réunion forme le *modèle global* du système. Ces deux modèles sont spécifiés en π-calcul étendu. Pour pouvoir spécifier au mieux ces deux modèles, nous considérons des extensions probabiliste, stochastique ou temporisée du π-calcul. En ce qui concerne la mobilité des systèmes, une autre approche aurait pu être de considérer, pour chaque composant mobile du système analysé, un *invariant de mobilité*, qui définit les mouvements acceptables de ce composant. Nous n'avons pas choisi cette dernière approche dans cette thèse.
- Les contributions précédentes sont les principales de cette thèse. Elles visent à définir un cadre formel à l'analyse de la performabilité des systèmes communicants mobiles basé sur des extensions du π-calcul et sur l'outil PRISM. Une analyse de performabilité consiste à considérer plusieurs modèles de mobilité pour un même modèle des communications. Pour faciliter cette analyse, nous proposons d'utiliser des masques de commandes dans les modèles de bas niveau en langage PRISM permettant de rendre

indépendant le modèle des communications et un modèle de mobilité d'un système et ainsi ne pas à devoir tout réécrire le modèle global lorsque l'on veut analyser les exigences de performabilité d'un système avec différents modèles de mobilité.

Plan de cette thèse

Cette thèse s'articule autour de quatre chapitres.

Le chapitre 1 introduit *l'analyse de la performabilité* des systèmes. Nous y définissons les notions de système et de performabilité des systèmes. Nous expliquons pourquoi il est important d'analyser la performabilité des systèmes avant de passer en revue différentes méthodes d'analyse existantes, en mettant l'accent sur les méthodes de modélisation.

Le chapitre 2 est consacré à la *mobilité* des systèmes et présente les formalismes permettant de formaliser cette notion. Ce sont essentiellement le *π-calcul*, dont nous présentons sa version de base [MPW92] et quelques unes de ses extensions importantes. Ce sont aussi le *calcul des ambients* [CG98] et les *bigraphes* qui ont été proposés pour l'analyse des systèmes ubiquitaires [Mil01].

Le chapitre 3 est consacré à notre contribution. Le π-calcul est un formalisme permettant de décrire la mobilité des systèmes. Cependant il n'est pas adapté à la modélisation des systèmes communicants pour les raisons que nous avons expliquées ci-dessus. Les communications d'un système peuvent être asynchrones et peuvent être des diffusions. Certes il existe une version asynchrone du π-calcul [Bou92,Hon92,HT91b,HT91a], mais les communications restent de type point à point. Une première solution, simple, a été utilisée par l'auteur de cette thèse dans [AB11] pour modéliser des diffusions entre deux nœuds : elle consiste à modéliser une diffusion par autant de communications point à point séquentielles, qu'il y a de récepteurs potentiels d'une transmission de données. Mais cette façon de faire complexifie le modèle des communications d'un système. Le π-calcul ne permet pas non plus de modéliser les aspects temporels des systèmes communicants. Nous avons donc besoin d'un formalisme étendu. Nous proposons au chapitre 3 des extensions du π-calcul dont la sémantique s'exprime par des automates temporisés et des automates temporisés probabilistes lorsqu'il s'agit de modéliser des systèmes mobiles stochastiques et temporisés. Nous proposons une méthodologie de vérification des propriétés de performabilité de ses systèmes basée sur la dernière version de l'outil PRISM.

Nous appliquons ces résultats au chapitre 4 sur un modèle de réseaux de capteurs sans fil implémenté en milieu hospitalier. Les résultats que nous obtenons par l'étude de notre modèle sont cohérents et conformes à ceux réalisés par expérimentation sur le système réel et publiés dans [KHH11].

Finalement, nous concluons cette thèse en présentant quelques unes des directions possibles de ce travail vers lesquelles nous souhaiterions poursuivre nos travaux de recherche.

Des annexes complètent les chapitres. Nous y exposons les techniques de model checking, l'outil PRISM, les *réseaux de capteurs sans fil* et le modèle complet du réseau présenté au chapitre 4.

Chapitre 1

Introduction à l'analyse de la performabilité des systèmes

Ce chapitre introduit l'analyse de la performabilité des systèmes[1]*. Il nous a paru indispensable de consacrer un chapitre à la définition des notions et des termes sur lesquels notre travail est basé. Ainsi, nous définissons ce qu'est un système, avant de nous intéresser plus particulièrement dans la suite de cette thèse aux systèmes à événements discrets. Nous ne faisons qu'évoquer les systèmes hybrides et les systèmes continus. Nous définissons ce qu'est la performabilité des systèmes, et nous présentons les méthodes permettant de l'analyser. Nous présentons également dans ce chapitre une méthodologie d'analyse de la performabilité des systèmes basée sur les techniques de model checking. Enfin nous présentons les réseaux de capteurs sans fil que nous avons choisi comme champ d'application.*

1 Qu'est-ce qu'un système ?

Un *système* (du grec σύστημα, *systéma*, qui signifie « ensemble organisé ») est un objet complexe ayant une fonction particulière[2]. Il est formé d'un ensemble cohérent de *sous-systèmes* interdépendants et communicants, chacun d'eux se distinguant des autres par sa « macro-fonction » opérationnelle particulière. Chaque sous-système peut lui-même être décomposé en sous-systèmes, et ainsi de suite, jusqu'à obtenir des *composants* atomiques, chacun d'eux étant doté d'une « micro-fonction » qui ne peut plus être subdivisée, ou qu'il n'est pas possible de subdiviser sans complexifier inutilement le système global. Un système possède un degré de complexité plus important que ses sous-systèmes, ce qui signifie qu'il possède des propriétés irréductibles aux leurs. Les relations qui lient les sous-systèmes entre-eux expliquent cette irréductibilité.

Un système a un *environnement intérieur* formé par l'ensemble de ses sous-systèmes, et un *environnement extérieur*, encore appelé son « milieu », ou son « contexte », et une *frontière* qui séparent et isole ces deux environnements définie par la fonction du système. Un système communique et on distingue habituellement deux types d'échanges : les échanges

[1] En toute rigueur, on s'intéresse à l'analyse des *exigences de performabilité* des systèmes. Une exigence est, dans une spécification, l'expression claire et précise d'une qualité du système à construire, ou se rapportant à sa fonctionnalité. Une spécification d'un système n'est rien d'autre qu'un ensemble d'exigences plus ou moins raffinées selon le niveau de référence du cycle de développement du système. Nous n'en disons pas plus dans cette thèse qui n'est pas consacrée à l'ingénierie des exigences. Pour plus d'information, le lecteur peut consulter [HJD10].

[2] La norme française NFE 90.001 définit un système comme *une association de sous-systèmes constituant un tout organique complexe destiné à remplir une fonction générale.*

internes entre les sous-systèmes, et les échanges externes entre les sous-systèmes et l'environnement au travers de la frontière. La fonction du système se traduit par la transformation de matière d'œuvre brute issue de l'environnement, en produit à valeur ajoutée et en déchets. La production de valeur ajoutée et des déchets dépend d'éléments de l'environnement dit *de contrôle* correspondant à l'énergie nécessaire, les commandes programmées, les commandes manuelles, etc. Nous avons illustré cette définition d'un système communicant par la figure 1.1.

Plusieurs caractéristiques majeures nous intéresse dans cette thèse que nous souhaitons présenter :

- Du point de vue *structurel* :
  - Les systèmes sont *distribués* car ils sont formés de multiples sous-systèmes et composants ayant des fonctions bien précises qui contribuent tous ensembles à la fonctionnalité du système global.
  - Les systèmes sont *concurrents* car les composants se partagent des ressources communes nécessaires à la réalisation de leur propres fonctions. Ce peut être des ressources physiques (processeurs, mémoires, périphériques d'ordinateurs, ou voies de communication, etc.), ou des ressources logiques (variables, bibliothèques d'utilitaires, pilotes de périphérique, etc.).
  - Les systèmes sont *coordonnés* parce que les communications entre sous-systèmes et composants s'effectuent généralement au moyen de techniques permettant de les synchroniser : transmission de messages, partage de variables, sémaphores, etc.
  - Les systèmes sont *mobiles* parce qu'ils se déplacent, soit dans un espace physique (par exemple un ordinateur portable appartenant à un réseau local IEEE 802.11 [IEE07], ou encore un capteur dans un réseau de capteur Zigbee [IEE03], etc.), soit dans un espace virtuel (par exemple, une applette Java téléchargée dans une page Internet), ou encore parce que leur structure évolue en fonction de leurs interactions avec l'environnement. La notion de mobilité des systèmes est définie au chapitre 2.
  - Les systèmes *clos*, encore appelés *fermés*, sont des systèmes dont la frontière est imperméable aux échanges avec leur environnement. Au contraire, les systèmes *ouverts* interagissent avec leur environnement. Un système ouvert reçoit des informations de son environnement conditionnant ses comportements, et il peut agir sur celui-ci. Lorsqu'on cherche à analyser la performabilité de ces systèmes, il est donc nécessaire de considérer toutes les actions possibles de l'environnement. Pour simplifier les analyses, nous considérerons toujours des systèmes clos, quitte à considérer l'environnement comme une partie du système, si la prise en compte des échanges est indispensable pour l'analyse.
- Du point de vue *temporel* : les sous-systèmes peuvent, soit cadencer leur fonctionnement de manière *synchrone* à partir d'une horloge commune, soit évoluer indépendamment les uns des autres de manière *asynchrone*. Dans tous les cas, chacun réagit de façon *réactive* à la réception d'occurrences d'événements issues de l'environnement. Leur réactivité est conditionnée aux instants effectifs de réception de ces occurrences et ces instants dépendent de la technologie employée, à la fois pour la transmission des événements, et pour le principe (synchrone ou asynchrone de leur observation) .
- Les systèmes à *événements discrets* forment une classe particulière de systèmes, ceux dont le comportement peut être décrit par des variables d'état à valeurs discrètes. Les changements d'état ne sont donc pas continus mais discrets, et ils se produisent

sous l'effet d'événements qui, s'ils peuvent avoir lieu de façon continue dans le temps, induisent des modifications quantifiées des variables d'états. Un exemple typique est un système composé d'un guichet, d'une file d'attente de clients, et d'un guichetier qui les sert. Parce qu'il ne peut y avoir qu'un nombre entier de clients dans la file, et qu'un nombre entier de clients ne peut être servi à la fois, en l'occurrence un, ce système est à événements discrets. Cette classe de système s'oppose aux *systèmes continus* dont les variables d'états peuvent prendre des valeurs continues. Un système qui réagit à la température ambiante est un système continu puisque celle-ci varie continûment. Les *systèmes hybrides*, quant-à-eux, comportent des évolutions continues et des phénomènes discrets qui leur sont liés.

2 Qu'est-ce que la performabilité des systèmes ?

2.1 Rappels de sûreté de fonctionnement

La *sûreté de fonctionnement* d'un système caractérise son aptitude à réaliser sa fonction, ou à fournir son (ou ses) service(s), dans des conditions données, sans engendrer de risques majeurs en terme de vies humaines, de matériels coûteux, de flux financiers, d'environnement, etc. Elle traduit la confiance que l'on peut accorder à un système, la sûreté de fonctionnement étant, selon une définition proposée par Jean-Claude Laprie dans [ABC+96], la *propriété qui permet aux utilisateurs de ce système de placer une confiance justifiée dans le service qu'il leurs délivre*. La sûreté de fonctionnement d'un système s'évalue grâce à quatre paramètres qui sont : sa *fiabilité*, sa *maintenabilité*, sa *disponibilité*, sa *sécurité*, auxquels on peut ajouter sa *testabilité*.

- La *fiabilité* d'un système est son *aptitude à accomplir une fonction requise dans des conditions données et pendant une période de temps donnée* (norme Afnor NF X 06-501). C'est une fonction du temps qui estime, de manière probabiliste, l'aptitude du système à accomplir sa fonction dans des conditions données, et pour un intervalle de temps donné. Un exemple de mesure de fiabilité est le taux de défaillance qui est l'inverse du MTTF (*Mean Time To Failure*) qui correspond lui-même au temps moyen jusqu'à la première défaillance du système.
- La *maintenabilité* d'un système est son *l'aptitude à être rétablie dans l'état de fonctionnement spécifié, selon les procédures prescrites, dans un laps de temps fixé* (norme Afnor). Un exemple de mesure de maintenabilité est le MTTR (*Mean Time To Recover*) qui correspond au temps moyen de réparation ou de restauration de l'état de bon fonctionnement du système.
- La *disponibilité* d'un système est son *aptitude qu'un système soit en état d'accomplir une fonction requise, dans des conditions fixées, et à un instant donné* (norme Afnor).
- La *sécurité* présente deux aspects différents : l'aspect *innocuité*, et l'aspect *confidentialité*. La sécurité-innocuité vise à se protéger des défaillances catastrophiques d'un système, c'est-à-dire celles dont les conséquences sont inacceptables. La sécurité-confidentialité correspond à la prévention d'accès ou de manipulations non autorisées du système, et concerne la lutte contre les fautes intentionnelles telles que les virus, les bombes logiques, les chevaux de Troie, etc. Elle vise également à garantir l'intégrité des informations fournies aux utilisateurs.
- La *testabilité* (testability) mesure la facilité de tester le système durant tout son cycle de vie, y compris en exploitation.

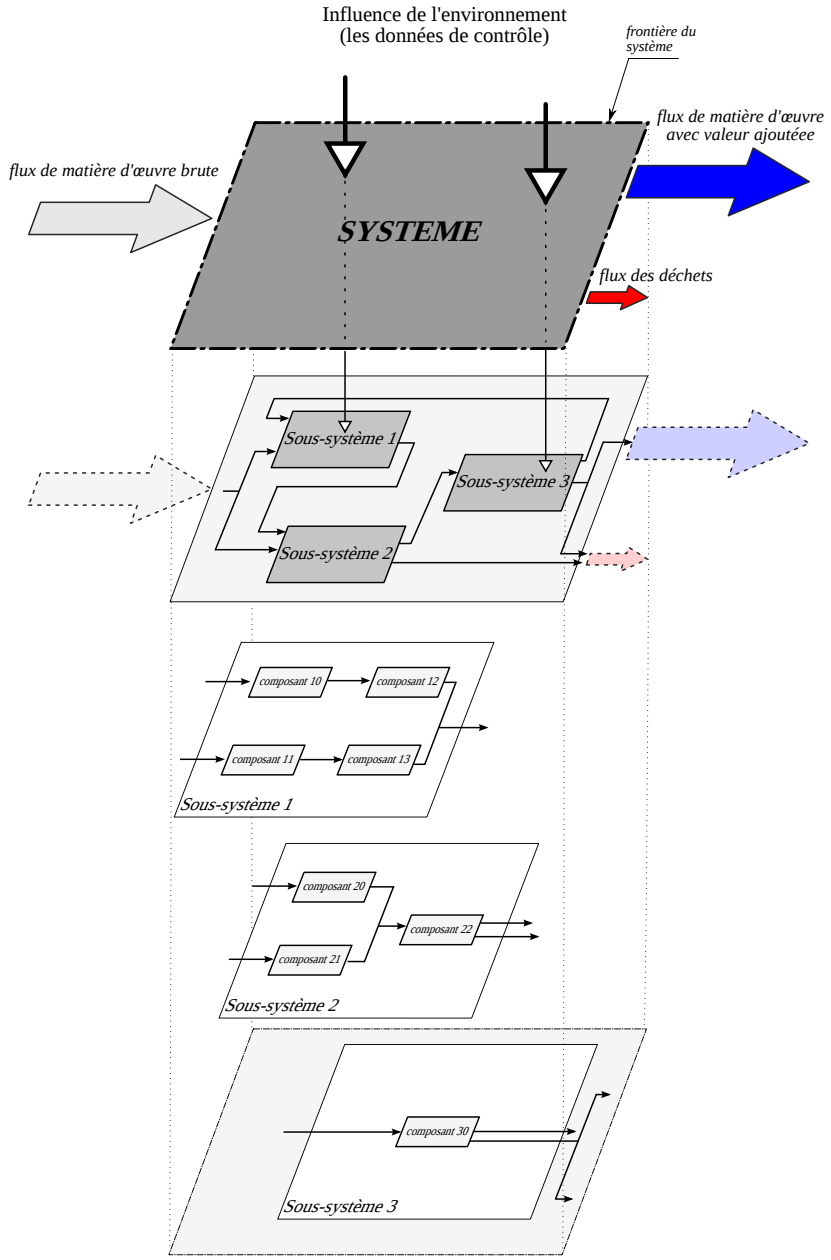


FIG. 1.1 – Représentation d'un exemple de système décomposé en sous-systèmes et en composants ayant respectivement une macro-fonctionnalité et une micro-fonctionnalité particulière. On parle de *systèmes communicants* parce qu'ils communiquent en interne ou avec leur environnement. On parle de *systèmes mobiles* lorsqu'ils sont capables de se déplacer dans l'espace, ou parce que leurs liens de communication évoluent au gré de leurs interactions. (cf. chapitre 2). Nous modéliserons ses systèmes selon la *composition parallèle* de leurs composants. Les analyses de performabilité par model checking nous amènerons à considérer des *systèmes clos*, qui ne communiquent pas avec leur environnement, et des *systèmes finis* dont le nombre de composants est fini.

2.2 Performance

La performance est une caractéristique importante des systèmes. Elle qualifie et (ou) quantifie son engagement à délivrer le service pour lequel le système a été conçu. Comme la sûreté de fonctionnement, la performance d'un système est évaluée selon plusieurs paramètres. Dans les systèmes communicants, un paramètre important est le *temps de réponse*, c'est-à-dire le délai d'acheminement des communications de bout en bout. Il mesure le temps qui sépare l'émission d'un message, de sa réception par le destinataire. Un autre paramètre de performance important des systèmes communicants est le nombre de messages échangés par unité de temps, c'est-à-dire le *débit* des communications. Pour d'autres systèmes, ce peut être le nombre de produits finis fabriqués par le système. On peut également s'intéresser à la charge d'un système, que l'on appelle son *taux d'utilisation*, qui mesure la proportion de temps pendant lequel ce système est occupé. Un système ayant un taux d'utilisation proche de 100% peut constituer un goulet d'étranglement pour un autre système, et expliquer son mauvais fonctionnement. On peut également s'intéresser au *temps d'attente* avant de pouvoir utiliser le système, ou au *nombre de clients* en attente de son utilisation. D'autres paramètres de performance peuvent être définis.

2.3 Notion de système sûr

Un système *sûr de fonctionnement*, ou plus simplement un système *sûr*, est un système capable de délivrer, tout au long de son activité opérationnelle, le service qui a été prévu par sa spécification, et ce, dans tous ses états admissibles, et pour tous les états prévus de son environnement. La sûreté de fonctionnement d'un système ne signifie pas que le système reste pleinement opérationnel, avec des performances optimales tout au long de son fonctionnement. Cela signifie que, dans toutes les situations possibles qui ont été envisagées pendant la conception de ce système, il a été prévu les modalités qui lui permettent, soit de se maintenir dans son état opérationnel nominal, soit de commuter de mode pour délivrer un service de qualité dégradée mais acceptable. Le qualificatif *sûr* signifie donc que, dans toutes les circonstances prévues, le système continue à délivrer un service qui a été spécifié, et que ce service est bien celui attendu dans les circonstances présentes de l'état du système. Au contraire, un système *non sûr* est un système qui peut délivrer un service inattendu, ou un service incohérent, c'est-à-dire un service qui ne répond à aucune spécification [ABC+96].

2.4 Définition de la performabilité des systèmes

La *performabilité* des systèmes est une notion composite qui intègre les paramètres de sûreté de fonctionnement présentées au paragraphe 2.1, avec la notion de *performance*. La notion de performabilité est essentielle pour les *systèmes dégradables*, qui sont des systèmes sûrs fonctionnant en mode opérationnel avec une performance optimisée lorsqu'il n'y a pas de pannes, mais pouvant montrer une dégradation de leur performance en présence de dysfonctionnements ou de pannes de leurs composants, tout en continuant à fonctionner dans un mode dégradé.

La performance d'un système est habituellement considérée comme la qualité de service (QoS) que ce système fournit en mode opérationnel. Analyser la performance d'un système demande de représenter la nature de ce système, mais aussi la nature de l'environnement dans lequel il évolue, et qui est à l'origine des ses stimuli. Analyser les performances d'un système en mode opérationnel demande de pouvoir prédire la capacité de celui-ci à réaliser ses services, sous l'hypothèse que sa structure demeure intacte, c'est-à-dire qu'il n'est pas soumis à des dysfonctionnement ou des pannes de ses composants, et par suite, à leur réparation. Par ailleurs, analyser la sûreté de fonctionnement d'un système nécessite de représenter les

changements qui peuvent affecter sa structure, et comment de telles modifications affectent le système global.

La performance et la sûreté de fonctionnement des systèmes sont donc des notions distinctes. Doit-on pour autant les distinguer dans l'analyse des systèmes ? Dans ce cas, intuitivement, si les aspects de sûreté de fonctionnement d'un système sont satisfaits, alors ses performances sont optimisées. Mais agir de la sorte revient à considérer des systèmes qui présentent des performances optimales lorsqu'ils sont pleinement opérationnels, avec des baisses brutales de leurs performances quand une panne survient. Ce qui revient à considérer de façon binaire, soit que le système est pleinement opérationnel et fonctionne correctement, soit que le système est en panne. Cette considération binaire ne prend pas en compte les systèmes dégradables que nous avons évoqués au début de ce paragraphe. Les performances de ces systèmes ne peut pas être évaluées finement sans prendre en compte les conséquences des changements de leur structure causées par les pannes et les réparations qui s'en suivent. Analyser ces systèmes seulement du point de vue de leurs performances est optimiste puisque cela néglige les possibles changements de leurs structures. A l'inverse, une analyse uniquement du point de vue de la sûreté de fonctionnement est conservative, puisque les considérations sur leurs performances ne sont pas prises en compte. D'où l'intérêt de combiner ces différents aspects à travers des analyses de performabilité qui, finalement, considère les effets des modifications de la structure d'un système dégradable sur ses performances globales.

3 Analyser la performabilité

L'analyse de la performabilité d'un système revient à vérifier ses paramètres de sûreté de fonctionnement, et à évaluer sa performance. Nous pouvons distinguer deux catégories de méthodes d'analyse de la performabilité des systèmes :

1. L'**expérimentation** du système réel, ou à défaut, d'un système équivalent, d'une maquette, ou encore d'un prototype du système. Dans ce cas le système réel existe, mais on souhaite le modifier ou le tester en dehors de son point de fonctionnement normal. Il s'agit, par exemple, de concevoir un système différent répondant à de nouveaux objectifs. Dans ce sens, l'expérimentation rejoint la conception. Il s'agira de tester le fonctionnement du système après avoir remplacé un ou plusieurs de ses sous-systèmes dont les paramètres de performabilité doivent être améliorés, avant d'analyser les résultats, et éventuellement de recommencer si les résultats obtenus ne sont pas ceux attendus. Il est aussi intéressant de tester un système dans des conditions anormales de fonctionnement, telles que des pannes de sous-systèmes, ou des surcharges de travail. On peut vouloir auditer un système existant en vue de la conception d'un système futur. On peut vouloir mettre au point, ou calibrer un système pour améliorer ses performances. On peut encore vouloir comparer les performances de deux systèmes en exploitation : c'est le *benchmarking*. L'expérimentation a l'avantage d'être la méthode d'analyse la plus précise, puisqu'elle utilise directement le système réel ou un système équivalent. Mais il s'agit d'une méthode coûteuse, qui n'est pas toujours possible à mettre en œuvre lorsque le système est utilisé, ou du moins, difficile à le faire. Par ailleurs, l'expérimentation souffre d'un manque d'exhaustivité. Pour résumer ce problème, Edsger W. Dijkstra[3]

[3]Edsger W. Dijkstra (1930-2002) est un des informaticiens les plus influents du XXe siècle. Parmi ses nombreux domaines de recherche figure l'algorithmique, puisqu'il a proposé l'*algorithme du plus court chemin* utilisé par les protocoles de routage OSPF et IS-IS, ou encore l'*algorithme du banquier* pour l'allocation de ressources entre processus avec évitement de blocages. Ses travaux portent également sur les langages de programmation, la conception des systèmes d'exploitation des ordinateurs, l'informatique distribuée, la conception formelle de programmes sûrs. Il reçut le prix Turing en 1972.

disait : *testing shows the presence, not the absence of bugs* [EB70].

2. La **modélisation** qui consiste à spécifier un modèle d'un système, dont l'analyse s'effectue généralement à l'aide d'un ordinateur, ou si modèle n'est pas trop complexe, « manuellement» à l'aide d'un crayon, d'une gomme et de papier. Parmi ces techniques, on distingue la simulation à événements discrets, les techniques de modélisation analytique, et les techniques de modélisation formelle, en particulier le *model checking* auquel cette thèse est consacrée.

3.1 La modélisation d'un système

Une autre manière d'analyser le performabilité d'un système, est d'en spécifier un *modèle* qui reproduit son comportement, et de pratiquer l'analyse sur ce modèle plutôt que sur le système réel qui, dans ce cas, peut ne pas exister.

3.1.1 Notions de modèle et de sous-modèle

Un modèle est une abstraction plus ou moins précise d'un système réel. En fonction du degré d'abstraction, on obtient un modèle plus ou moins fidèle. Si l'on considère tous les paramètres qui influencent le système réel, on obtient un modèle très fidèle mais inexploitable parce que trop compliqué pour être analyser. A l'inverse, si l'on considère un modèle trop abstrait du système réel, celui-ci pourra être analysé avec facilité, mais il sera trop éloigné du modèle réel pour que son analyse soit pertinente. La modélisation consiste donc à trouver un compromis entre l'adéquation d'un modèle et du système réel, et la facilité d'analyser le modèle spécifié. Différents types de formalismes ont été proposés pour modéliser les systèmes. Parmi eux, citons les *automates* [Sak03], *les files d'attente* [Bay00] et les *réseaux de files d'attente*, les *réseaux de Kahn* [Kah74], les *réseaux de Petri* [Rei85], les *chaînes de Markov* et *les processus de décision markoviens* [KNP10], les *algèbres de processus* [Bae05], etc. Ces formalismes permettent de modéliser des systèmes à événements discrets. Les *systèmes d'équations différentielles* [HW99] sont plus adaptés à la modélisation des comportements des systèmes continus.

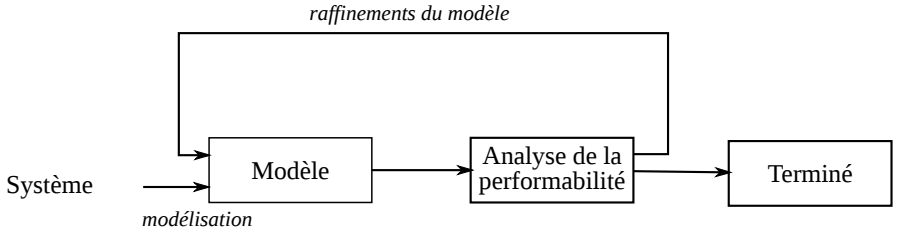


FIG. 1.2 – Principe de l'analyse de la performabilité d'un système à partir d'un modèle.

L'analyse de la performabilité d'un système à partir de son modèle peut se schématiser par la figure 1.2. Elle se décompose en deux étapes : une étape de modélisation qui permet d'abstraire le système en un modèle, et d'une étape d'analyse de ce modèle. Ce sont donc les paramètres de performabilité du modèle, et non ceux du système réel, qui sont analysés. D'où l'importance de l'étape de modélisation pour la justesse de l'analyse. Une éventuelle étape supplémentaire de rebouclage consiste à raffiner le modèle à chaque étape d'analyse. Dans un premier temps, on commence l'analyse sur un modèle grossier très simple, dans lequel seuls les paramètres importants ont été pris en compte. En compliquant progressivement le modèle, et en effectuant après chaque modification une analyse des résultats, on peut affiner

le modèle pour s'approcher du système réel. A chaque étape de raffinement, il est nécessaire de s'assurer de la préservation des propriétés déjà établies. Le raffinement s'arrêtera lorsque le modèle sera suffisamment proche de la réalité, ou lorsque les techniques d'analyse deviendront plus complexes à mettre en œuvre.

Enfin, parce qu'un système est une structure hiérarchisée de sous-systèmes plus simples, il est utile de modéliser un système à partir de la composition de modèles déjà définis et sauvegardés. L'analyse de la performabilité des modèles composés se ramène aux techniques développées pour l'analyse des sous-modèles.

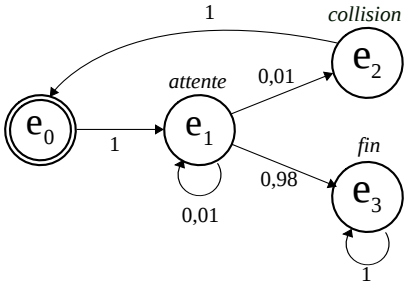
3.1.2 Les différentes types de modèles

Les modèles peuvent être *déterministes* ou *indéterministes* selon que leurs comportements qu'ils décrivent peuvent être décrit précisément, sans ambiguïté, ou non ; *stochastiques* ou *non stochastiques* selon qu'ils prennent en compte, ou non, des comportements aléatoires ; *statiques* ou *dynamiques*, selon qu'ils considèrent des comportements invariables, ou variables, dans le temps. Les notions d'indéterminisme et de stochasticité sont différentes : un système peut être déterministe, et présenter des comportements aléatoires exprimés dans son modèle par des distributions de probabilités ou des taux d'occurrence. Dans ce cas, ces derniers sont parfaitement connues. En particulier, la somme des probabilités des comportements possibles à la suite d'un comportement donné est égale à 1. C'est typiquement le cas des *chaînes de Markov à temps discret* (DTMCs), et des *chaînes de Markov à temps continu* (CTMCs) [KNP10]. A l'inverse, *les processus de décision markoviens* modélisent des systèmes dont les comportements sont probabilistes, donc aléatoires, et indéterministes car leurs comportements successifs ne peuvent pas être connus à leurs probabilités d'occurrence près [KNP10]. Ces trois formalismes sont présentés à l'annexe A.

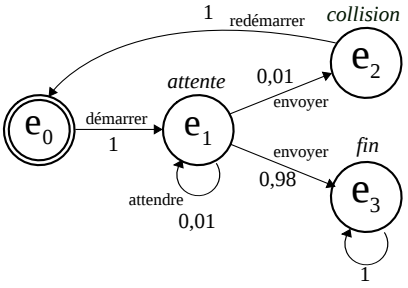
Modèles déterministes ou indéterministes. L'indéterminisme est la théorie épistémologique opposée au déterminisme. Elle nie le fait que tout évènement soit prévisible en vertu d'une quelconque loi. Dans les systèmes, et donc leurs modèles déterministes, les conditions initiales déterminent l'évolution comportementale future. Les mêmes conditions initiales produisent toujours les mêmes effets comportementaux. Ces conditions ne sont plus respectées dans les systèmes indéterministes et leurs modèles.

Modèles certains ou stochastiques. Nous appelons les modèles « non stochastiques », des modèles *certains*. Cette dichotomie concerne la prise en compte des aléas du comportement d'un système dans son modèle. Dans les modèles *certains*, ils ne sont pas pris en compte car leur influence est considérée comme négligeable. Par contre, ils sont pris en compte dans les *modèles stochastiques* car on juge qu'ils ont un rôle significatif. La modélisation par abstraction stochastique et (ou) indéterministe permet de s'abstraire des comportements réels qui sont trop difficiles à modéliser, ou qui rendent le modèle trop difficile à exploiter pour pouvoir être considérés. Les différents états de l'environnement d'un système, ou les pannes d'un système qui peuvent éventuellement survenir, sont des exemples classiques d'aléas.

Modèles statiques ou dynamiques. Une autre distinction concerne les *modèles statiques* et les *modèles dynamiques* des systèmes. Dans les premiers, le temps n'intervient pas. Dans les autres, le temps est un facteur essentiel de la modélisation des états et du comportement du système. Il peut être modélisé de manière discrète ou continu. Parmi les modèles dynamiques, on distingue les modèles à *événements discrets*, des modèles *hybrides*, et des modèles *continus* des systèmes de même nom que nous avons déjà évoqués au paragraphe 1. Parmi les modèles dynamiques, on distingue les *systèmes temporisés*, dont les comportements sont modélisés



(a) Modèle déterministe et stochastique d'un protocole de communication.



(b) Modèle indéterministe et stochastique du même protocole de communication. A partir de l'état e_1 la transition *envoyer* entraîne l'état e_2 ou l'état e_3 : les comportements sont indéterministes.

FIG. 1.3 – Exemples de modèles déterministe (ou indéterministe) et stochastiques. Les figures 1.3(a) et 1.3(b) représentent respectivement une chaîne de Markov à temps discret (DTMC) et un processus de décision markovien (MDP). Il s'agit pratiquement du même système, en l'occurrence un protocole de communication décrit au § A.3.2.1, sauf que la DTMC modélise des comportements connus à leur probabilité d'occurrence près, alors que le MDP représente des comportements indéterministes (d'après [Kwi07]).

à l'aide de formalismes mettant en œuvre des horloges, ou de chronomètres, permettant de mesurer le temps qui passe.

3.1.3 Modèles de mobilité

La mobilité des systèmes demande que leurs communications utilisent le sans-fil. Nous définissons la mobilité au chapitre 2. Une caractéristique d'un système mobile est d'avoir une *topologie de communication*[4] dynamique qui charge au gré de ses interactions, ou de ses mouvements. Ces changements de topologies influencent sa performabilité, puisque la position du système influence ses communications avec les systèmes voisins.

En phase de conception, l'analyse de la performabilité du système peut être analysée à l'aide de différents *modèles de mobilité*. Ce genre de modèle consiste à faire des hypothèses sur les mouvements du système. Pour spécifier un tel modèle, on peut, soit utiliser des traces sur le déplacements observés en fonctionnement réel, soit utiliser des modèles synthétiques qui spécifient les règles de déplacements. Les modèles spécifient des paramètres physiques portant sur le système (tels que sa position initiale, sa vitesse, son accélération, la direction du mouvement, la durée d'un mouvement, la fréquence de changement de mouvement, et la durée de pause entre deux changements successifs), et sur leur environnement (les dimensions, la portée de la transmission des signaux, les obstacles sur le chemin). Par exemple, une synthèse complète des modèles de mobilité pour les *réseaux ad hoc* a été proposée dans [CBD02]. Une autre pour les *réseaux sans fil* a été proposée dans [Sch06]. Les modèles proposés sont souvent utilisés dans outils de simulation. A notre connaissance les outils de vérification formelle automatique (les *model checker*) ne les intègrent pas.

[4] C'est-à-dire l'ensemble de ses liens de communications avec les autres systèmes qui forment son environnement.

3.2 Vérification qualitative et analyse quantitative

A partir d'un modèle, l'analyse de la performabilité d'un système portent sur [BHHK10] :

- La *vérification qualitative* de ses propriétés structurelles et comportementales, telles que l'absence de blocage (vivacité), les invariants du système, les comportements finis ou bornés (stabilité), l'équité, l'inéluctabilité, etc.
- L'*analyse quantitative*, c'est-à-dire l'évaluation, de ses paramètres de performance. Elle n'a de sens que si une vérification qualitative a été menée. Il est en effet inutile de vouloir obtenir les performances d'un système qui qui est bloqué. La vérification qualitative et l'analyse quantitative se combinent en incluant un comportement stochastique, avec des aspects temps réels. De telles propriétés quantitatives permettent l'évaluation de la performance d'un système, en plus de sa correction. Par ailleurs, un modèle qui comprend des probabilités et des aspects temporels permet d'inclure des exigences quantitatives dans des propriétés liées à la sûreté de fonctionnement, telle que « le protocole doit assurer que la probabilité d'envoi d'un message avec succès en au plus 5 secondes est supérieure à 95% ».

4 Techniques de modélisation

4.1 La simulation à événements discrets

La *simulation à événements discrets* consiste à reproduire le comportement d'un système en étudiant une réalisation particulière de son modèle. L'avantage de la simulation est d'offrir une approche très générale permettant d'étudier n'importe quel modèle, du moment que l'outil de simulation est adapté au modèle considéré. Par contre, son inconvénient est de demander beaucoup de temps de calcul machine. Par ailleurs, la simulation n'est pas une technique exacte : les résultats fournis sont exacts que si le temps de simulation est infini. Dans la pratique, la simulation du modèle d'un système dure entre quelques minutes et quelques jours, et ses résultats s'accompagnent toujours d'intervalles de confiance permettant d'estimer leur degré d'approximation, et permettant de savoir si la simulation effectuée a fonctionné suffisamment longtemps.

Il existe de nombreux outils de simulation à événements discrets. Parmi eux, citons les simulateurs de réseaux NS-2 [NS2], NS-3 [NS3] (« NS » pour *network simulator* en anglais), OMNeT++ [OMN], qui permet de simuler différents types de réseaux dont les réseaux de files d'attente, et OPNET [OPN] qui est un outil destiné aux études de performances.

4.2 Les méthodes analytiques

Les *méthodes analytiques* proposent d'étudier le comportement stationnaire[5] d'un système en résolvant son modèle mathématique, ou plus précisément, en résolvant les équations mathématiques sous-jacentes à son modèle mathématique. L'intérêt des méthodes analytiques réside principalement dans la résolution des équations généralement peu coûteuse en temps de calcul. Par ailleurs, les méthodes analytiques permettent d'avoir une bonne compréhension du fonctionnement du système, car on est plus en mesure d'analyser certains de ses dysfonctionnement en résolvant son modèle, et donc de proposer des modifications pour les régler. L'inconvénient des méthodes analytiques est qu'il est généralement nécessaire de faire des hypothèses restrictives sur le système réel pour pouvoir obtenir des modèles exploitables. Une

[5] Les comportements stationnaire et à l'équilibre sont décrits pour les processus stochastiques au paragraphe A.3.1.

solution alternative consiste à utiliser des techniques analytiques approximatives, qui en n'introduisant qu'une approximation raisonnable, et bien souvent négligeable par rapport à celles engendrées par l'étape de modélisation, permettent de simplifier considérablement l'analyse. Par ailleurs, les méthodes analytiques ne permettent d'analyser que les valeurs moyennes de la performance du comportement stationnaire d'un système. Enfin, les méthodes analytiques, qui sont des méthodes quantitatives, ne sont pas aptes à analyser la fiabilité d'un système.

4.3 Les méthodes formelles

Les *méthodes de vérification formelle* ou, plus simplement *les méthodes formelles*, offrent un cadre mathématique permettant de modéliser de manière précise et stricte les systèmes à analyser et à vérifier. Les deux principales techniques de vérification formelle sont *la construction assistée de preuves*, et *la vérification de modèles*.

4.3.1 La construction assistée de preuves ou *theorem proving*.

Le comportement du système est modélisé par un ensemble d'axiomes. Un outil, appelé un *assistant d'aide à la preuve de théorèmes*, génère les pas de la preuve de ces axiomes. Cette technique est une *assistance* car elle n'est automatique. Elle nécessite la direction d'un expert. Celui-ci doit penser la preuve, qui est vérifiée par l'outil, ou au moins il doit fournir les axiomes et guider le processus de vérification. Il existe plusieurs outils d'assistanat à la preuve. Parmi eux, citons Coq [COQ], PVS [PVS], HOL [HOL], et Isabelle [ISA]. Cependant, la construction assistée de preuves ne permet pas de faire des analyses quantitatives, telles que des études de performances.

4.3.2 La vérification de modèles ou *model checking*.

Cette technique consiste à construire un modèle M d'un système $\mathcal{S}$, et à spécifier les exigences de performabilité que ce système doit satisfaire sous la forme de propriétés écrites en logique temporelle $\{\varphi_i\}_{i\in I}$, avant de vérifier automatiquement avec l'aide d'un outil appelé un *model checker*, que *ce* modèle respecte *ces* propriétés. Dans le cas contraire, l'outil fournit un contre-exemple. Les modèles et les propriétés sont *qualitatifs* ou, si l'outil de vérification utilisé le supporte, *quantitatifs*, exprimant des probabilités, des taux, des durées, des coûts, ou encore des récompenses (voir annexe A). Les modèles, finis, sont dérivés des *systèmes à états-transitions étiquetées* (LTS pour *labeled transition system*), encore appelés des *structures de Kripke*[6]. La vérification s'effectue par l'exploration exhaustive des états du modèle de manière algorithmique. Les utilisateurs spécifient généralement les modèles à l'aide de formalismes de haut-niveau permettant d'abstraire les LTSs, par exemple des automates [Sak03], des réseaux de Petri [Rei85], ou encore des algèbres de processus [Bae05]. Dans cette thèse, nous avons choisi cette dernière solution, ce choix étant expliqué dans les avant-propos.

Les propriétés à analyser sont spécifiées à l'aide d'une *logique temporelle*[7] compatible avec le modèle du système. L'outil de vérification génère un contre-exemple en cas d'échec de la vérification de la véracité d'une propriété. Parmi les outils de model checking, nous pouvons citer SPIN [Spi] qui permet de vérifier des modèles spécifiés en langage PROMELA (*Protocol Meta Language*) [PROb], PRISM [PRIb, HKNP06, KNP11] qui est présenté à l'annexe

[6] Du nom de leur concepteur, Saul A. Kripke, un logicien et philosophe américain né en 1940.

[7] Les logiques temporelles sont des langages formels permettant de spécifier et de raisonner sur la façon dont le comportement d'un système évolue au cours du temps. Elles étendent la logique propositionnelle avec des opérateurs modaux et temporels. Il en existe plusieurs types : les logiques à temps arborescent (de type CTL) et les logiques à temps linéaire (de type LTL), ou leurs extensions probabilistes ou temporisées.

B, UPPAAL [UPPa, GBL06, LPY97] pour la vérification de modèles temporisés, UPPAAL-PRO [UPPb] pour la vérification de modèles probabilistes temporisés. L'annexe A détaille le model checking. Le lecteur peut également consulter [BBF+01].

5 Notre méthodologie d'analyse de la performabilité

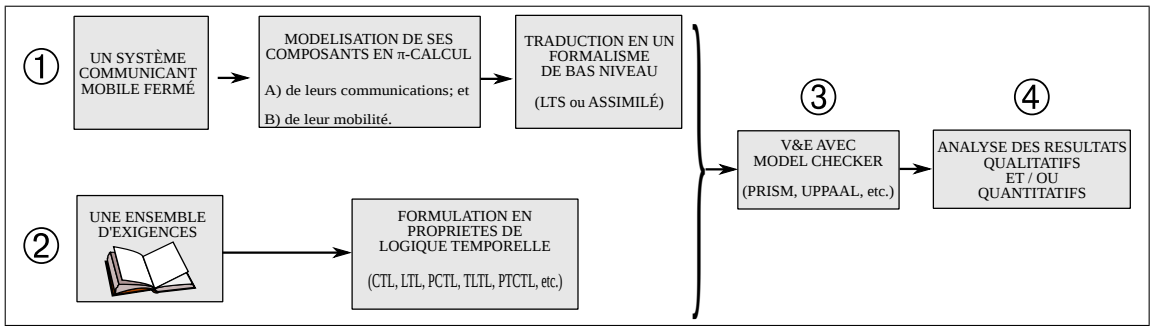


FIG. 1.4 – Méthodologie d'analyse des systèmes communicants mobiles.

La méthodologie d'analyse des systèmes communicants mobiles que nous avons adoptée dans cette thèse, illustrée à la figure 1.4, est classique au model cheking avec, toutefois, une nuance liée à la mobilité des systèmes. A noter que l'on considère des systèmes clos qui ne communiquent pas avec leur environnement.

Cette méthodologie consiste d'abord à modéliser le système : c'est l'étape 1 de la figure précédente. Le comportement du système est décomposé en deux aspects qui sont, d'une part ses communications internes (internes parce que l'on considère que le système est clos), et d'autre part la mobilité de ses sous-systèmes. Les modèles correspondants sont spécifiés à l'aide d'un formalisme de haut niveau, plus facile à manipuler qu'un formalisme à états-transitions de bas niveau utilisé par les outils de model checking. Ce formalisme de haut niveau est une algèbre de processus issue du π-calcul [MPW92] présenté au chapitre 2. Les modèles de haut niveau sont ensuite traduit en un formalisme à états transitions, tels qu'une chaîne de Markov à temps discret ou à temps continu, ou un automate temporisé, ou encore un automate temporisé probabiliste. Les formalismes de bas niveau utilisés dépendent des propriétés à analyser : un modèle à comportement aléatoire sera modélisé par une chaîne de Markov. Un système dont le comportement est temps réel sera modélisé par un automate temporisé, etc. Le passage d'un modèle de haut niveau en un modèle de bas niveau nécessite l'application de *règles de traduction* qui constituent une partie de notre contribution proposée dans cette thèse. Par ailleurs, les exigences de performabilité des systèmes sont, quant-à-elles spécifiées sous la forme de propriétés de logique temporelle adéquate (étape 2). Les logiques temporelles utilisées doivent permettre l'expression et l'analyse de propriétés stochastiques et/ou temporelles. Ces propriétés sont ensuite vérifiées sur le modèle par le model checker choisi (étape 3), puis analysées par l'opérateur (étape 4).

Chapitre 2

Penser, modéliser, et maîtriser la notion de mobilité

1 Quel est le sens du terme « mobilité » ?

En avant-propos de ce mémoire, nous avons voulu mettre en évidence la prépondérance des systèmes communicants mobiles dans les sociétés humaines contemporaines, au travers d'appareils qui assurent la continuité de leurs services, tout en se déplaçant dans l'espace. Cette tendance ne fait que croître pour répondre aux besoins des utilisateurs. C'est pourquoi lorsque l'on cherche à analyser la performabilité d'un système communicant, considérer sa mobilité, ou la mobilité de ses composants, est une nécessité, qui, à notre avis, ne peut pas être négligée. Lorsque l'on veut vérifier les exigences de performabilité d'un système, à moins de savoir que ce système n'est pas mobile, il est nécessaire de vérifier leur véracité sous les contraintes d'immobilité et de mobilité. Et même, à notre avis, il est nécessaire de vérifier les exigences d'un système sous la contrainte d'une échelle de mobilité, l'échelle la plus simple étant de distinguer une mobilité faible, d'une mobilité forte. Ne pas considérer la mobilité d'un système lors de la vérification de ses exigences, restreint cette vérification à son immobilisme, ce qui peut s'avérer être très restreignant si le système est mobile. Autrement dit, lorsque l'on s'intéresse à la vérification formelle c'est-à-dire au *model checking*, on ne peut pas prétendre avoir vérifier qu'un modèle d'un système respecte des exigences de performabilité spécifiées sous la forme de propriétés énoncées en logique temporelle, si ce modèle ne prend pas en compte la possible mobilité du système, puisque dans ce cas, seuls les cas d'immobilisme sont considérés. Il se pose alors le problème de savoir quels types de mobilité, ou plutôt quels modèles de mobilité du système, doivent être pris en considération pour vérifier les propriétés de performabilité auxquelles on s'intéresse. Nous reviendrons sur ce point.

Cependant, avant d'évoquer la vérification formelle de système mobile, il est nécessaire de présenter les travaux qui ont conduit à formaliser la notion de mobilité, à bâtir une théorie de la mobilité, qui doit servir de socle à tous les formalismes utilisés pour la vérification formelle des systèmes communicants mobiles. Mais qu'est-ce que la mobilité ? Quel est le sens de cette notion ? Intuitivement, comme nous l'évoquons au début de ce chapitre à propos des systèmes communicants mobiles, ce terme désigne la qualité d'un système capable de se déplacer dans l'espace tout en assurant un service continu. Cependant, la mobilité évoque aussi la qualité d'un système capable de se reconfigurer. Le dictionnaire *le nouveau petit Robert de la langue française* (2009) [Col09] en donne la définition suivante :

> ***Mobilité*** *: (nom féminin, v. 1200 du latin mobilitas) ① Caractère de ce qui peut se mouvoir ou être mû, changer de place, de position. ② Caractère de ce qui change rapidement d'aspect ou d'expression.*

Cette définition a l'inconvénient d'être ambiguë : elle ne précise pas quelles sont les entités qui se déplacent, dans quel espace réel ou virtuel ces entités sont capables de se déplacer, ou encore quelles entités sont capables de se reconfigurer. Donner une définition non ambigüe de la mobilité doit être la première étape de sa modélisation formelle. Ainsi, dans le cadre des systèmes communicants, on peut considérer que les entités mobiles sont des *processus* informatiques qui bougent dans l'espace formé par des *liens* qui existent entre-eux. La mobilité des processus se traduit alors par la reconfiguration de la topologie de leurs liens. Par exemple, les connections entre des téléphones mobiles et des antennes-relais peuvent apparaître et disparaître en fonction des déplacements des téléphones. On peut aussi considérer que les liens entre les processus se reconfigurent d'eux-mêmes dans l'espace virtuel formé par l'ensemble des processus liés, les processus restant fixes. On peut encore considérer que les processus se déplacent entre des espaces appelées *localités*, sièges des calculs qui permettent à ces processus d'évoluer. Un exemple concret d'un tel système mobile serait un programme informatique transitant de nœuds en nœuds dans un système distribué.

Robin Milner[1] remarque dans *Communicating and mobile systems : the π-calculus* [Mil99] qu'à un niveau suffisant d'abstraction, la localisation d'un processus, dans l'espace virtuel formé par l'ensemble des processus qui sont liés, est déterminée par les liens qu'il partage avec les autres processus. Autrement dit, « les voisins d'un processus sont ceux avec lesquels il peut communiquer », c'est-à-dire ceux avec qui il partage une voie de communication. Dans ce cas, les mouvements d'un processus sont entièrement déterminés par les mouvements de ses liens avec les autres processus. Donc les deux premiers choix parmi les précédents peuvent être considérés comme synonymes. Par contre, le dernier choix fait référence à la notion de localité, qui est absente des deux premiers. Robin Milner remarque qu'une localité peut être modélisée par un processus, et que la localisation d'un processus dans une localité particulière peut alors être modélise par l'existence d'un lien entre deux processus [Mil99]. Dans ce cas, les trois choix précédents, bien qu'il n'exprime pas la même notion de mobilité, peuvent se réduire au même cas. Cette remarque n'est pas anodine, car elle sous-entend qu'un seul formalisme permet d'exprimer tous les aspects de la mobilité, comme le remarque Davide Sangiorgi dans [SW01]. Ce formalisme, sur lequel nous allons revenir tout au long de ce chapitre, s'appelle le *π-calcul*. Cependant, de l'avis même de Robin Milner, la réduction des trois cas de mobilité à un seul est moins évidente que la réduction des deux premiers cas [Mil99]. La mobilité par changements de localité doit rester un cas particulier de mobilité des systèmes.

En effet, si la notion de processus reste la notion abstraite pour représenter un système mobile en boîte noire, ou un sous-système mobile lorsque l'on s'intéresse aux interactions internes d'un système multi-processus, deux aspects fondamentaux de mobilité des systèmes se distinguent dans les publications scientifiques : la mobilité par l'*évolution des liens* qui existent entre les processus, et la mobilité en tant que *déplacements physiques des processus entre différentes localités*. Ces deux aspects sont masqués dans le premier point de la définition donnée le dictionnaire *le nouveau petit Robert* [Col09]. Le deuxième aspect est modélisable en π-calcul, comme le premier d'ailleurs, du moment que l'on ne considère pas la notion de

[1] Robin Milner (1934-2010) est un informaticien britannique majeur. Parmi ces travaux, nous pouvons citer LCF (*Logic for Computable Functions*), le premier système de démonstration automatique de théorèmes, le langage de programmation fonctionnel ML, ainsi qu'une théorie des systèmes concurrents (CCS) et mobiles (le π-calcul). Ses derniers travaux portent sur la théorie des bigraphes, un formalisme pour l'informatique ubiquitaire, présentée au paragraphe 5. Il fut lauréat du prix Turing en 1991.

« place », ou que l'on modélise cette notion à l'aide d'un processus. Dans une étude bibliographique consacrée à la mobilité des systèmes et aux formalismes qui la modélisent [Dal01], Silvano Dal Zilio appelle les processus qui opèrent des changements de liens fréquents, des *processus labiles*, par analogie avec les composants labiles d'une réaction chimique. De la même façon, il nomme les processus qui se déplacent de localités en localités, des *systèmes motiles*, par analogie avec le même terme utilisé en biologie pour désigner des formes de vie qui peuvent se déplacer de manière indépendante. Au travers la classification binaire de S. Dal Zilio, on comprend bien que les auteurs des publications scientifiques consacrées à la mobilité des systèmes distinguent deux classes de formalismes : celle des formalismes qui modélisent la mobilité des systèmes comme la modification des liens qui lient des processus entre-eux, et celle des formalismes qui modélisent la mobilité comme le déplacement de processus entre différentes localités. Lorsque l'on s'intéresse à la mobilité des systèmes, la notion de localité est une notion pivot.

Le *π-calcul* est le formalisme majeur pour modéliser les systèmes labiles. Il permet d'exprimer la mobilité des systèmes à événements discrets[2], par les mouvements des liens qui existent entre les processus modélisant ces systèmes, dans l'espace virtuel formé par l'ensemble de ces processus liés entre-eux [MPW92]. En fait, le *π-calcul* a deux aspects : une théorie des processus labiles et un modèle de calcul qui se base sur la notion d'*interaction*. C'est ce deuxième aspect qui nous permet de modéliser des systèmes labiles, et de raisonner sur leurs comportements. Le π-calcul ne considère que deux types d'entités : les *processus* et les *noms*, c'est-à-dire les liens ou voies de communication entre les processus. Les processus communiquent en s'échangeant des noms, et modifient ainsi leur *topologie de communication*, c'est-à-dire leur capacité à communiquer entre-eux. En recevant un nom, un processus peut acquérir la capacité à échanger d'autres noms avec des processus avec lesquels il n'était pas lié auparavant. Cependant, le *π-calcul* ne considère pas la notion de localité. Certaines de ses extensions le font, telles que le *join-calcul distribué* [FGL+96], ou le *π-calcul distribué* (Dπ) [RH98]. C'est aussi le cas du *calcul des ambiants* [CG98] qui a été influencé par le π-calcul. Les deux notions de mobilité, c'est-à-dire la labilité et la motilité, ont été réunies dans la *théorie des bigraphes*, et leurs extensions les *systèmes réactifs bigraphiques* [Mil09]. Dans la suite de ce chapitre, nous présentons le π-calcul, le calcul des ambiants, et les bigraphes, pour illustrer la formalisation de la notion de mobilité des systèmes communicants mobiles.

2 Le π-calcul : un calcul de processus labiles

Nous caractérisons le π-calcul comme un « calcul de processus labile », parce qu'il modélise la mobilité des systèmes par la reconfiguration de leur topologie de communication respective, et conformément à la notion de labilité expliquée au paragraphe 1. Ce n'est pas contradictoire avec l'appellation donnée par ses concepteurs qui parlent d'un « calcul de processus mobiles » (*a calculus of mobile processes*) [MPW92].

2.1 Ses origines

Le π-calcul est une algèbre de processus communicants, concurrents et labiles. Il fut introduit par Robin Milner, Joachim Parrow et David Walker au début des années 1990 [MPW92]. La volonté de ses concepteurs était de décrire un modèle universel pour les calculs concurrents,

[2] Les *systèmes à événements discrets* sont définis au paragraphe 1.1. Le ϕ-calcul, qui est une extension du π-calcul, permet de décrire les systèmes hybrides mobiles [RS03]. Nous n'en parlons que très peu dans cette thèse.

à la manière du λ-calcul non-typé conçu pour décrire les calculs fonctionnels[3]. A l'époque de la conception du π-calcul, les formalismes majeurs pour exprimer les communications et la concurrence entre processus étaient, d'une part les *réseaux de Petri* [Rei85] et d'autre part, les algèbres de processus, plus précisément *CCS* (*Calculus of Communicating Systems*) [Mil80], *CSP* (*Communicating Sequential Processes*) [Hoa85], et *ACP* (*Algebra of Communicating Processes*) [BK85]. Mais aucun de ces formalismes considère une quelconque notion de mobilité.

De nombreux chercheurs ont proposé des formalismes définissant une notion de mobilité sans en développer une théorie algébrique. C'est le cas en 1970 du *le modèle des acteurs* d'Hewitt [HBS73], où les acteurs peuvent changer de configurations. Plusieurs chercheurs ont travaillé sur la sémantique du modèle, dont Irene Greif [Gre75], Henry G. Baker [HB77], William D. Clinger [Cli81], et Gul Agha [Agh86]. Leurs travaux ont conduit à définir une véritable théorie du modèle des acteurs. En 1984, Egidio Astesiano et Elena Zucca étendent CCS en y incluant la notion de voies paramétrées [AZ84]. Dans certains formalismes, la mobilité est obtenue en permettant aux processus d'être échangés comme des valeurs dans les communications. La différence avec le π-calcul, qui permet les échanges de liens entre processus mais pas les échanges de processus eux-mêmes (dans sa version de base), contraste avec des tentatives pour combiner les idées du λ-calcul et des algèbres de processus, en considérant les processus comme des valeurs. Dans ce cadre, nous devons citer les travaux de Gérard Boudol [Bou89], Flemming Nielson [Nie89], et Bent Thomsen [Tho89].

Mais hormis les travaux de Robin Milner à propos de CCS [Mil80], c'est probablement les travaux de Uffe Engberg et Mogens Nielsen qui ont le plus influencé le π-calcul. Ces deux chercheurs, qui n'ont jamais publié leur rapport technique, ont étendu l'algèbre CCS par la notion de mobilité, tout en préservant ses propriétés algébriques [EN]. Robin Milner écrit dans [MPW92], que le π-calcul est une extension de CCS par les travaux de U. Engberg et M. Nielsen. C'est certainement parce que beaucoup de caractéristiques du π-calcul leurs sont dues : le remplacement de la notion de renommage de CCS par la notion de substitution syntaxique, le traitement sémantique de l'extrusion de portée, l'extension de la notion de bisimulation pour considérer les noms des paramètres, la définition de la notion de bisimilarité forte, ainsi que de nombreuses lois algébriques. Cependant, les auteurs du π-calcul ont simplifié l'approche de U. Engberg et M. Nielsen en gommant les distinctions qui existaient entre les variables, les constantes et les valeurs, en les considérant simplement comme des « noms ». Les autres contributions des auteurs du π-calcul ont été de distinguer plusieurs formes d'équivalence (*ground equivalence* et *non-ground equivalence*), de renforcer les lois algébriques permettant des théories équationnelles complètes, de représenter le calcul de valeurs avec l'aide de l'opérateur MATCH, et de fournir des traductions simples de formalismes fonctionnels dont le λ-calcul et l'algèbre combinatoire.

2.2 Sa syntaxe

Nous décrivons, dans la suite de ce paragraphe, le *π-calcul monadique synchrone*, ou plus simplement le « π-calcul » qui est la version basique de cette algèbre, publiée au début des années 1990 dans *A Calculus of Mobile Processes, Part 1 & 2* [MPW92]. Sa syntaxe

[3]Le λ-calcul [Chu32, Chu41] a été introduit par Alonzo Church dans les années 1930 avec pour objectif de fonder les mathématiques. En 1936, Church isole puis publie la partie du calcul consacrée à la calculabilité, connue maintenant sous le nom de λ-calcul non typé, après que Kleene et Rosser aient démontré l'inconsistance du système original. Celui-ci est universel dans le sens où n'importe quelle fonction calculable peut être exprimée et évaluée en utilisant ce formalisme. Le λ-calcul a été redécouvert dans les années 1960 comme outil théorique pour l'informatique, avec les travaux de McCarthy, Strachey, Landin, et Scott.

s'exprime à l'aide de deux entités : les *noms* et les *processus*, encore appelés des *agents*. Les noms sont dénotés par des lettres minuscules, par exemple x, y, z, etc., et leur ensemble, infini, est dénoté par $\mathcal{N}$. Ce sont des représentations abstraites de structures de données diverses, qui, sont utilisés par les processus comme des voies de communication[4]. Ces derniers sont habituellement dénotés par des lettres majuscules, par exemple P, Q, R, etc, et leur ensemble, lui aussi infini, est $\mathcal{P}$. Ils représentent un état du comportement d'un système qui communique avec d'autres systèmes formant son environnement. Les processus réalisent des *actions*. Celles-ci, au nombre de quatre et décrites ci-après, leurs permettent d'échanger des noms, ou des références sur des noms, via les voies de communication qu'ils partagent, qui ne sont rien d'autres que des noms ! Ce qui signifient que les processus du π-calcul peuvent s'échanger des voies de communications qui modifient leur capacité d'échange avec d'autres processus. Ce sont donc des processus mobiles, plus précisément des processus *labiles* au sens décrit en introduction. Cette algèbre permet de modéliser le comportement des systèmes qui sont capables de se reconfigurer, ou de ce déplacer, sous l'effet de leurs interactions avec leur environnement.

2.2.1 Les processus

Définition 2.1. L'ensemble $\mathcal{P}^\pi$ des expressions des processus est défini par :

$$P ::= 0 \mid \alpha_i.P \mid P_1 + P_2 \mid P_1|P_2 \mid (\nu x)P \mid [x = y]P \mid A(x_1, \ldots, x_n).$$

Dans cette définition :

- Le processus 0 est le *processus inactif* qui ne fait rien.
- $\alpha_i.P$ est le processus qui réalise l'*action* α_i avant de se comporter comme le processus P. Il existe quatre actions qui sont définies ci-après.
- $P_1 + P_2$ est la *sommation* de processus ou encore le *choix* non déterministe entre processus. Ce processus se comporte comme P_1 ou P_2 de façon non déterministe. La sommation vide correspond au processus inactif.
- $P_1|P_2$ est la *composition de processus*. Ce processus correspond à P_1 et P_2 qui évoluent en parallèle. Ces processus peuvent interagir, par exemple par l'émission d'un nom de l'un d'eux synchronisée avec une réception d'un nom par l'autre, via une voie de communication (c'est-à-dire un nom) qu'ils partagent. Cet échange correspond à l'*action silencieuse* τ interne au processus $P_1|P_2$. L'action silencieuse est décrite ci-après.
- La *restriction de nom* $(\nu x)P$ dénote un processus qui se comporte comme P, sauf que les actions mettant en jeu le nom x sont restreintes aux composants de P. Ce sont alors des actions silencieuses de P.
- La *correspondance* $[x = y]P$ dénote un processus qui se comporte comme P si, et seulement si, les noms x et y sont identiques sinon il se comporte comme le processus inerte 0.
- Un processus peut être défini comme un processus existant d'*arité* n : $P \stackrel{def}{=} A(x_1, \ldots, x_n)$.

N.B. Dans [Mil91], Robin Milner inclut la *réplication* d'un processus P, dénotée par $!P$, dans le paragraphe consacré à la version monadique du π-calcul. Cette opération n'apparait pas dans la publication [MPW92] qui nous sert de référence. Comme son nom l'indique, elle dénote la duplication infinie d'un processus qui évolue en parallèle avec ses duplications précédentes : $!P \stackrel{def}{=} P|!P$. La réplication de processus génère des modèles de systèmes à états

[4] La notion de port de communication, que l'on trouve dans CCS, est absente du π-calcul.

infinis. Elle ne peut donc pas être utilisée pour construire des modèles finis de systèmes, qui pourraient être vérifiées automatiquement par la technique du *model checking* fini.

Les actions. Elles sont au nombre de quatre :

- L'action silencieuse τ. L'action $P \xrightarrow{\tau} Q$ signifie que P peut évoluer en Q sans interagir avec d'autres processus de son environnement. Cet évolution est du à des interactions entre composants à l'intérieur de P, inobservables depuis son environnement.
- L'action $\bar{x}y$ représente l'émission du nom y via le nom x. La transition $P \xrightarrow{\bar{x}y} Q$ implique que P peut émettre le nom y sur x, à condition qu'un autre processus puisse recevoir y via x. En effet, dans le π-calcul basique, une action d'émission est synchronisée avec une action de réception. Le *π-calcul asynchrone*, présenté au paragraphe 3.2, casse ce synchronisme en ne définissant pas l'action d'émission, mais, à la place, un processus d'émission. Par ailleurs, les interactions émission-réception sont de type *point-à-point* (*unicast* en anglais) : le π-calcul basique ne définit pas des interactions de type diffusion restreinte, c'est-à-dire un émetteur pour plusieurs récepteurs dans le système (*multicast*), et à fortiori, des interactions de type diffusion générale, c'est-à-dire un émetteur pour tous les récepteurs du système (*broadcast*).
- L'action $x(y)$ représente la réception d'un nom via le nom x. La transition $P \xrightarrow{x(y)} Q$ signifie que P peut recevoir n'importe quel nom w sur x avant d'évoluer en $Q\{w/y\}$. Ici (y) représente une référence de nom, une place où sera placé le nom reçu.
- L'action $\bar{x}(y)$ représente l'émission d'une référence d'un nom, appelée y, via le nom x.

Les noms impliqués dans une action, peuvent être *libres* ou *liés*, selon le type de cette action. Le tableau 2.1 résume les quatre actions possibles et l'attache correspondante des noms impliqués.

α	Nom de l'action	$fn(\alpha)$	$bn(\alpha)$
τ	silencieuse	$\emptyset$	$\emptyset$
$\bar{x}y$	émission	$\{x, y\}$	$\emptyset$
$x(y)$	réception	$\{x\}$	$\{y\}$
$\bar{x}(y)$	émission d'une référence	$\{x\}$	$\{y\}$

TAB. 2.1 – Les quatre actions du π-calcul et l'attache des noms correspondante.

On définit également les noms libres et les noms liés d'un processus selon le tableau 2.2. L'occurrence d'un nom dans un processus est libre, si elle n'est pas liée. L'ensemble des noms libres du processus P est dénoté par $fn(P)$, alors que celui de ses noms liés est dénoté par $bn(P)$. L'ensemble de tous les noms de P est alors $n(P) = fn(P) \cup bn(P)$, avec $fn(P) \cap bn(P) = \emptyset$.

Deux processus P et Q qui ne diffèrent que par leurs noms liés, sont appelés des *processus α-convertibles*, et ils ne sont pas distingués. Dans ce cas, on a l'habitude de noter $P \equiv_\alpha Q$.

Les noms libres d'un processus circonscrivent les capacités de ce processus à communiquer, et donc interagir avec d'autres processus de son environnement. Pour pouvoir émettre ou recevoir un nom, un processus doit posséder des noms libres qui permettent les échanges. Dans la figure 2.1, les processus P, Q, et R peuvent communiquer parce qu'ils partagent les noms libres x, y, z et w. Le nom x est partagé par les trois, alors que y est exclusivement réservé aux processus P et Q : on dit qu'il est restreint, ou interne, ou, plus habituellement,

P	$fn(P)$	$bn(P)$
0	$\emptyset$	$\emptyset$
$\alpha_i.Q$	$fn(\alpha_i) \cup (fn(Q) \setminus bn(\alpha_i))$	$bn(\alpha_i) \cup bn(Q)$
$Q_1 + Q_2$	$fn(Q_1) \cup fn(Q_2)$	$bn(Q_1) \cup bn(Q_2)$
$Q_1 \vert Q_2$	$fn(Q_1) \cup fn(Q_2)$	$bn(Q_1) \cup bn(Q_2)$
$(\nu x)Q$	$fn(Q) \setminus \{x\}$	$bn(Q) \cup \{x\}$
$[x = y]Q$	$\{x, y\} \cup fn(Q)$	$bn(Q) \setminus \{x, y\}$
$!Q$	$fn(Q)$	$bn(Q)$
$A(x_1, \ldots, x_n)$	$fn(P) \subseteq \{x_1, \ldots, x_n\}$	$bn(A(x_1, \ldots, x_n))$
$!P$	$fn(P)$	$bn(P)$

TAB. 2.2 – Les noms libres et les noms liés des processus du π-calcul.

lié au processus $(P|Q)$. Aucun autre processus du système ne peut l'utiliser.

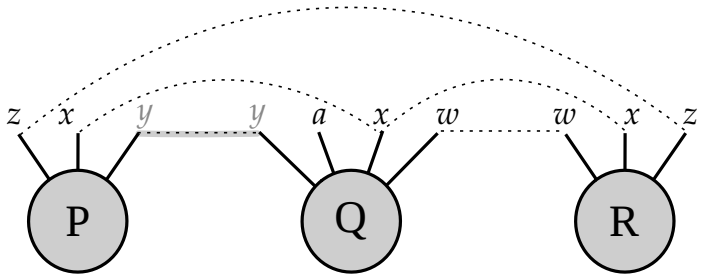
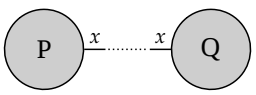
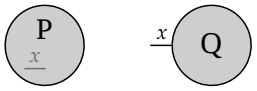


FIG. 2.1 – Communications entre processus en π-calcul. Le nom y est lié au processus $P|Q$.

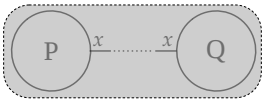
Un processus qui n'a pas de noms libres, ne peut pas réaliser d'actions d'entrées-sorties avec d'autres processus de son environnement : il est dit *clos* ou *fermé*. Par extension, un système clos est un système qui ne partage aucun nom libre avec son environnement, ce qui revient à dire qu'il ne peut pas réaliser d'actions d'entrées/sorties avec lui (cf. chapitre 1.1). Par contre, les interactions entre les composants de ce processus ou de ce système qui, depuis son environnement sont modélisées par des actions silencieuses, restent possibles. La figure 2.2 illustre les possibilités de communication entre deux processus P et Q.



(a) Le système $P \mid Q$. Les processus P et Q partagent le nom *libre* x : ils peuvent communiquer.



(b) Le système $(\nu x)P \mid Q$. Le nom x est *lié* (ou restreint) à P, et utilisé par certains de ses composants non représentés. Les deux processus ne peuvent pas communiquer.



(c) Le système $(\nu x)\,(P \mid Q)$. Le nom x est lié au processus $P \mid Q$. Les processus P et Q peuvent communiquer par cette voie qui leurs est exclusivement réservée.

FIG. 2.2 – Illustration de la capacité d'échange d'un processus en π-calcul.

Finalement, l'analyse de l'attache des occurrences des noms d'un processus permet de connaître, dans une certaine mesure seulement, les comportements de celui-ci. Dans $x(z).P$, les occurrences libres de z dans P indiquent les places où le nom libre reçu via x sera substitué après une interaction. C'est grâce à de tels échanges de noms que la topologie de

communication d'un processus peut évoluer. Si $fn(P) = \emptyset$, alors P est clos, et il ne peut pas communiquer avec son environnement. Donc, si son comportement est dynamique, alors ce dynamisme provient nécessairement de l'interaction entre ses composants. D'où l'intérêt de la vérification formelle des systèmes clos : on isole le système de son environnement, avant d'étudier son évolution comportementale.

2.2.2 Substitutions de noms

Une *substitution*, sous-entendue de noms, est une fonction σ définie de $\mathcal{N}$ dans $\mathcal{N}$ par l'identité, l'image d'un nom étant lui même, sauf pour un sous-ensemble fini de noms. La substitution σ appliquée au nom x est notée $x\sigma$. L'image du sous-ensemble de noms $\{x_1, \ldots, x_n\}$ par σ, notée $\{y_1/x_1, \ldots, y_n/x_n\}$, est telle que quel que soit $i \in [1..n]$ $x_i\sigma = y_i$ et $x\sigma = x$ pour $x \notin \{x_1, \ldots, x_n\}$. On note $P\{y_1/x_1, \ldots, y_n/x_n\}$, ou $P\{y_i/x_i\}_{1 \leqslant i \leqslant n}$, le processus obtenu en substituant simultanément chaque occurrence libre de x_i par y_i dans P avec, si nécessaire, le changement des noms liés portant la même appellation. On retrouve ce changement nécessaire de nom dans l'intrusion dans la portée d'un nom privé présentée au paragraphe 2.3 suivant.

Une substitution σ peut également être appliquée à une action α, ce que l'on dénote par $\alpha\sigma$. On a alors : $\tau\sigma = \tau$, $(x(y))\sigma = \sigma(x)(y)$ si $y \notin n(\sigma)$, $(\bar{x}y)\sigma = \sigma(\bar{x})\sigma(y)$, et $(\bar{x}(y))\sigma = \sigma(\bar{x})(y)$ si $y \notin n(\sigma)$. Dans les cas de la réception et de l'émission d'une référence, la substitution n'est définie que lorsque la substitution ne change pas les noms liés.

2.3 Exemples d'interactions entre processus

Les exemples de ce paragraphe, tirés de [MPW92], illustrent quelques interactions élémentaires entre processus du π-calcul.

Le passage d'un lien. Considérons le système composé des processus P, Q et R. P partage le nom x vers Q, qu'il souhaite envoyer à R via le lien y. R souhaite le recevoir. La transition, illustrée par la figure 2.3, est donc :

$$\bar{y}x.P' \mid y(z).R' \mid Q \xrightarrow{\tau} P' \mid R'\{x/z\} \mid Q$$

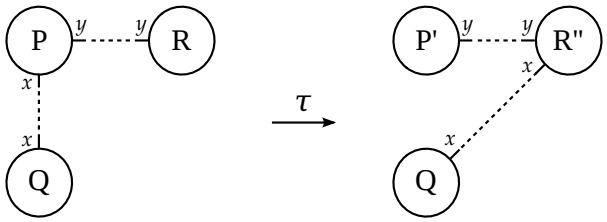


FIG. 2.3 – Illustration du passage d'un lien entre deux processus en π-calcul.

Ce schéma illustre le cas où $x \notin fn(R)$ et $x \notin fn(P')$. Ces deux conditions n'affectent pas la transition.

L'intrusion dans une portée d'un nom privé. Considérons le système composé par les processus P, Q, R et S. P partage le lien x avec Q, et le lien y avec R. Par ailleurs, R un accès exclusif au processus S, via le lien privé x qu'ils partagent. Le système global comprend donc deux liens x : l'un est public, alors que l'autre est privé au sous-système $R \mid S$. On suppose que P veut envoyer son lien x à R, qui souhaite le recevoir. Il est alors nécessaire de

renommer le lien privé x de ce dernier, pour éviter une confusion. Ce cas de figure se résume en disant que P présente une *intrusion dans la portée*, ou s'immisce dans la portée, du lien privé x entre R et S. Les interactions entre les processus de ce système sont représentées par la figure 2.4, et se modélisent par :

$$\bar{y}x.P' \mid Q \mid (\nu x)(y(z).R' \mid S) \xrightarrow{\tau} P' \mid Q \mid (\nu x')(R'\{x'/x\}\{x/z\} \mid S\{x'/x\})$$

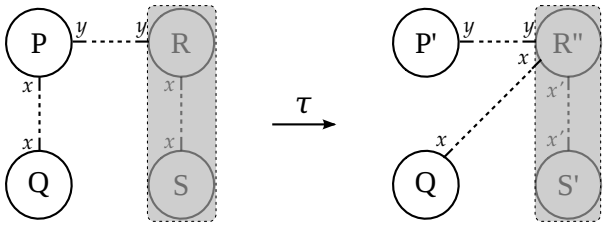


FIG. 2.4 – Illustration de la notion d'intrusion dans la portée d'un nom privé en π-calcul.

L'extrusion et la migration de portée. Considérons maintenant le système composé par les processus P, Q et R. P a un accès exclusif à Q via x, un lien privé au processus $P \mid Q$. Par ailleurs, P et R partagent le lien public y. On suppose que P veut envoyer x à R qui veut le recevoir. Les interactions entre les processus de ce système sont représentées par la figure 2.5, et se modélisent par :

$$(\nu x)(\bar{y}x.P' \mid Q) \mid y(z).R' \xrightarrow{\tau} (\nu x)(P' \mid Q \mid R'\{x/z\})$$

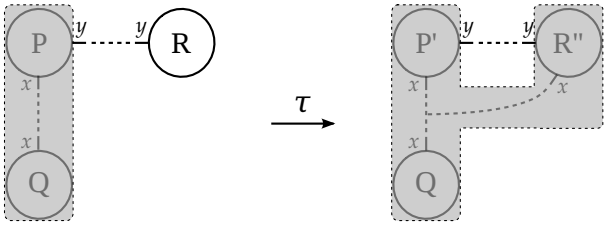


FIG. 2.5 – Illustration de la notion d'extrusion de portée d'un nom privé en π-calcul.

La portée du caractère privée de x à $P \mid Q$ est étendue au processus R. Deux cas peuvent se présenter :

- R possède déjà un lien public x, c'est-à-dire $x \in fn(R)$. Dans ce cas, le lien privé x doit être renommé avant l'interaction pour le distinguer du lien public existant.
- x n'est pas libre dans P' ($x \notin fn(P')$). Dans ce cas, le système global évolue en :
  $$P' \mid (\nu x)(Q \mid R'\{x/z\}),$$
  par application de la règle STRUCT EX énoncée au tableau 2.3. Cela revient à considérer que P' ne peut plus communiquer via x puisqu'il ne le possède plus. Ce cas de figure correspond à la *migration de portée* de x, qui est un cas particulier d'extrusion.

2.4 Sa sémantique

La sémantique du π-calcul est définie par les relations binaires de *réductions*, et de *transitions étiquetées* sur l'ensemble $\mathcal{P}$ des processus de l'algèbre.

Les réductions entre processus, dénotée par $\rightarrow$, spécifie l'évolution des processus lorsqu'ils réalisent un calcul atomique. On dit que P évolue en Q, ou, plus rarement, que Q réduit P

en une étape si $P \rightarrow Q$. Dans le π-calcul et, dans beaucoup d'autres algèbres de processus, de telles étapes atomiques concernent généralement l'échange d'un message. La sémantique des réductions du π-calcul utilise la notion de *congruence structurelle* entre deux expressions de processus qui autorise leur manipulation, de façon à placer des composants susceptibles d'interagir dans des positions contiguës. Cette approche a été proposée par Gérard Berry et Gérard Boudol dans *The Chemical Abstract Machine* [BB92], et introduite pour le π-calcul par Robin Milner dans [Mil92].

Les transitions étiquetées entre processus décrivent leurs évolutions sous l'effet de l'action qui est décrite par l'étiquette. Elles sont de la forme $P \xrightarrow{\alpha} Q$, et sont interprétées par le fait que P devient Q sous l'effet de α.

2.4.1 Congruence structurelle

Deux expressions de processus P et Q sont structurellement congruentes, et l'on note $P \equiv Q$, si l'une des deux se transforme en l'autre par l'utilisation des règles définies au tableau 2.3.

$P \equiv Q$ *si* $P \equiv_\alpha Q$	(STRUCT ALPHA)
$P + Q \equiv Q + P$	(STRUCT CHOICE COMMUT)
$(P + Q) + R \equiv P + (Q + R)$	(STRUCT CHOICE ASSOC)
$P \mid 0 \equiv P$	(STRUCT PAR ID)
$P \mid Q \equiv Q \mid P$	(STRUCT PAR COMMUT)
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$	(STRUCT PAR ASSOC)
$\nu x\, 0 \equiv 0$	(STRUCT ZERO)
$\nu x\, \nu y\, P \equiv \nu y\, \nu x\, P$	(STRUCT SWAP)
$\nu x\, (P \mid Q) \equiv P \mid \nu x\, Q$ *si* $x \notin fn(P)$	(STRUCT EX)
$!P \equiv P \mid !P$	(STRUCT REPL)

TAB. 2.3 – Règles définissant la congruence structurelle entre processus du π-calcul.

2.4.2 Règles de réduction entre les processus

Une relation de *réduction* entre deux processus P et P' se modélise par $P \rightarrow P'$, et signifie que P peut évoluer en P' sous l'effet d'une action interne. La sémantique de la relation de réduction de processus est définie par les règles présentées au tableau 2.4.

RED COMM : $(x(y).P + M) \mid (\bar{x}z.Q + N) \longrightarrow \{z/y\}P \mid Q$

RED PAR : $\dfrac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q}$ RED RES : $\dfrac{P \longrightarrow P'}{\nu x P \longrightarrow \nu x P'}$

RED STRUCT : $\dfrac{P \longrightarrow P'}{Q \longrightarrow Q'}$ si $P \equiv Q$ et $P' \equiv Q'$

TAB. 2.4 – Règles d'inférence définissant la relation de réduction entre processus en π-calcul.

2.4.3 Règles de transitions entre les processus

Une relation de *transition* entre deux processus P et P' se modélise par $P \xrightarrow{\alpha} P'$, et signifie que P peut évoluer en P' en réalisant l'action α. La sémantique des transitions entre processus du π-calcul est définie par les règles d'inférence présentées au tableau 2.5.

$$\text{TAU-ACT} : \frac{-}{\tau.P \xrightarrow{\tau} P} \qquad \text{OUTPUT-ACT} : \frac{-}{\bar{x}y.P \xrightarrow{\bar{x}y} P}$$

$$\text{INPUT-ACT} \ \frac{-}{x(z).P \xrightarrow{x(w)} P\{w/z\}} \ w \notin fn((z)P)$$

$$\text{CHOICE} : \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad \text{MATCH} : \frac{P \xrightarrow{\alpha} P'}{[x = x]P \xrightarrow{\alpha} P'}$$

$$\text{PAR} : \frac{P \xrightarrow{\alpha} P'}{P \,|\, Q \xrightarrow{\alpha} P' \,|\, Q} \ bn(\alpha) \cap fn(Q) = \emptyset$$

$$\text{COM} : \frac{P \xrightarrow{\bar{x}y} P' \ Q \xrightarrow{x(z)} Q'}{P \,|Q \xrightarrow{\tau} P' \,|\, Q'\{y/z\}} \qquad \text{CLOSE} : \frac{P \xrightarrow{\bar{x}(w)} P' \ Q \xrightarrow{x(w)} Q'}{P \,|Q \xrightarrow{\tau} (\nu w)(P' \,|\, Q')}$$

$$\text{RES} : \frac{P \xrightarrow{\alpha} P'}{(\nu y)P \xrightarrow{\alpha} (\nu y)P'} \ y \notin n(\alpha) \qquad \text{OPEN} : \frac{P \xrightarrow{\bar{x}y} P'}{(\nu y)P \xrightarrow{\bar{x}(w)} P'\{w/y\}} \ y \neq x, \ w \notin fn((y)P')$$

TAB. 2.5 – Règles d'inférence définissant la relation de transition entre processus en π-calcul.

3 Extensions remarquables du π-calcul

Nous présentons dans ce paragraphe quelques extensions du π-calcul conçues pour permettre de modéliser des propriétés des systèmes qui ne peuvent pas l'être dans la version de base de cet algèbre. Nous ne cherchons pas à être exhaustif en présentant toutes ses extensions. Nous cherchons seulement à montrer que le π-calcul peut être étendu de plusieurs manières pour répondre à des besoins spécifiques d'analyse. Les extensions suivantes pouvant être combinées pour définir d'autres extensions.

3.1 Quelques mots sur le π-calcul polyadique

En 1991, Robin Milner propose une version polyadique du π-calcul, dans lequel les processus peuvent s'échanger des vecteurs de noms, et non pas des noms uniques, comme c'est la cas dans sa version monadique exposée ci-dessus [Mil91]. Un processus « émetteur » sera de la forme $\bar{x}\langle y_1, \ldots, y_n\rangle.P$ où tous les y_i sont distincts, tandis qu'un processus récepteur s'écrira $x(z_1, \ldots, z_n).Q$; dans les deux cas n étant un entier naturel strictement positif. Le π-calcul polyadique peut être codé à partir du π-calcul monadique, en considérant que :

$$\bar{x}\langle y_1, \ldots, y_n\rangle.P = (\nu u)\ \bar{x}\langle u\rangle.\bar{u}\langle y_1\rangle. \ \ldots \ .\bar{u}\langle y_n\rangle.P$$
$$x(z_1, \ldots, z_n).Q = x(v).v(z_1). \ \ldots \ .v(z_n).Q$$

Cette façon de coder en utilisant un nom privé u, permet de rendre atomique les échanges des vecteurs de noms entre les processus P et Q. La règle de transition IDE doit être ajoutée aux règles du tableau 2.5 :

$$\frac{P\{\tilde{y}/\tilde{x}\} \xrightarrow{\alpha} P'}{A(\tilde{y}) \xrightarrow{\alpha} P'} \; A(\tilde{x}) \stackrel{def}{=} P \quad \text{(IDE)}$$

Par ailleurs pour permettre la transmission de vecteurs de noms, la règle de réduction de processus RED COMM est étendue de la façon suivante :

$$(x(\vec{y}).P + M)|(\bar{x}\langle\vec{z}\rangle.Q + N) \longrightarrow \{\vec{z}/\vec{y}\}P|Q$$

où les vecteurs $\vec{x}$ et $\vec{y}$ doivent avoir la même longueur.

3.2 Le π-calcul asynchrone

Le π-calcul asynchrone a été introduit par Gérard Boudol [Bou92], et de manière indépendante, par Kohei Honda et Mario Tokoro [Hon92, HT91b, HT91a]. Il diffère du π-calcul par l'absence du préfixe d'émission, ce qui signifie qu'un processus se termine immédiatement après une émission. Par ailleurs, ces versions ne définissent pas l'opérateur + du choix indéterministe entre processus. La syntaxe de cette algèbre est la suivante :

$$P ::= 0 \mid \bar{x}\langle\vec{y}\rangle \mid x(\vec{y}).P \mid !x(\vec{y}).P \mid P|Q \mid (\nu x)P$$

Le processus $x(\vec{y}).P$ attend une interaction sur x avec la sortie correspondante, en bloquant son évolution en P tant que cette interaction ne s'est pas réalisée. Le processus $\bar{x}\langle\vec{y}\rangle$ représente une émission de noms libres sur x. A la différence du π-calcul, il ne s'agit pas d'un préfixe, mais d'un processus : ce dernier n'a pas de suite et on ne peut pas écrire $\bar{x}\langle\vec{y}\rangle.P$ en π-calcul asynchrone, comme on ne peut pas écrire le processus $P.Q$ en π-calcul. Alternativement, nous pouvons dire que $P = 0$ est la seule continuation possible pour $\bar{x}\langle\vec{y}\rangle.P$. Finalement, une émission est interprétée comme *jeter un message dans la soupe* pour les laisser migrer tout seuls vers leurs récepteurs. C'est ainsi que l'asynchronisme des communications est décrit dans cette version de l'algèbre.

La sémantique du π-calcul asynchrone est la même que celle du π-calcul en éliminant la règle du choix SUM, et en remplaçant la règle de l'émission OUTPUT-ACT par la règle suivante :

$$\frac{-}{\bar{x}y \xrightarrow{\bar{x}y} 0}$$

3.3 Le π-calcul probabiliste

Plusieurs travaux proposent des extensions probabilistes du π-calcul. Dans [HP00], Oltea M. Herescu et Catuscia Palamidessi étendent le π-calcul asynchrone par l'ajout du choix aléatoire. La version proposée dans [CP05], a été conçue pour analyser les protocoles de sécurité aléatoires. Dans [BP05], le π-calcul probabiliste a été utilisé pour formaliser les définitions de l'anonymat. Nous décrivons dans ce paragraphe la version dénommée π_{prob} décrite dans [NPPW09]. Cette version, qui est pratiquement la même que celle de [CP05], étend le π-calcul par l'ajout d'un opérateur de choix probabiliste entre deux processus. Cet opérateur associe des probabilités de réalisation aux actions silencieuses τ, et à elles seules.

3.3.1 Sa syntaxe

L'ensemble $\mathcal{P}^{\pi_{prob}}$ des expressions des processus du π-calcul probabiliste est défini par :

$$P ::= 0 \mid \alpha.P \mid P_1 + P_2 \mid p_1\tau.P_1 \oplus p_2\tau.P_2 \mid P_1|P_2 \mid (\nu x)P \mid [x = y]P \mid A(y_1, \ldots, y_n).$$

où :

- Les actions α sont les mêmes qu'en π-calcul : τ, $x(y)$, $\bar{x}y$, et $\bar{x}(y)$.
- Le processus $p_1\tau.P_1 \oplus p_2\tau.P_2$ représente le choix probabiliste entre les processus $\tau.P_1$ et $\tau.P_2$, tel que la probabilité de choisir le premier est égale à p_1, alors que la probabilité de choisir le deuxième est égale à p_2. Ce choix effectué, le processus réalise une action silencieuse τ, avant de se comporter comme $P_{i,i\in\{1,2\}}$. Les probabilités p_1 et p_2 sont telles que : $p_1, p_2 \in]0, 1]$ et $p_1 + p_2 = 1$. Le choix probabiliste est distinct du choix indéterministe $P_1 + P_2$. Il y a donc deux opérateurs de choix en π_{prob}.
- Les autres expressions de processus sont les mêmes qu'en π-calcul définies au paragraphe 2.2.1.

3.3.2 Présentation de sa sémantique symbolique

La sémantique opérationnelle des extensions probabilistes du π-calcul, dont π_{prob}, s'exprime à l'aide de *processus de décisions de Markov* (MDPs) présentés à l'annexe A ou, de façon équivalente, à l'aide d'*automates probabilistes* [SL95]. Les sémantiques existantes sont *concrètes*, dans le sens où les règles de sémantique définissent directement le MDP qui correspond à un terme du processus. On peut également utiliser une représentation *symbolique* de la sémantique opérationnelle [HL95]. Les particularités de la sémantique symbolique qui permettent d'obtenir des modèles compacts sont :

- Comme dans la sémantique en retard du π-calcul, la variable d'entrée des réceptions est conservée comme une inconnue, à la différence de la sémantique en avance, où une transition différente est générée pour chaque instance de cette variable.
- De façon analogue à la règle du match, dans la règle de communication, la correspondance entre la voie de réception et la voie d'émission est représentée par une condition.

L'intérêt de la sémantique symbolique est la représentation des effets des communications lorsqu'elles se produisent.

Considérons, par exemple, le processus $P \stackrel{def}{=} a(x).\bar{x}b.0$ qui reçoit un nom sur la voie a, utilise le nom reçu pour envoyer le nom b, puis se termine. Une approche concrète de sa sémantique établit que ce processus peut accepter une réception sur la voie a, mais son comportement qui suit cette réception ne peut être appréhendé qu'une fois que le nom effectivement reçu est connu, puisque c'est en partie de lui que dépendent les interactions avec les autres processus de l'environnement. Par contre, l'approche symbolique permet de considérer des « inconnues » utilisées dans des actions, tel que x utilisée par $\bar{x}b$ dans notre exemple. Cette façon de procéder permet d'adopter une approche compositionnelle : la sémantique d'une composition parallèle de plusieurs processus peut être établie à partir de la sémantique de chacun d'eux établie séparément.

La sémantique symbolique des expressions de processus en π_{prob} s'exprime en terme de *graphes de transitions symboliques probabilistes* (PSTGs, *probabilistic symbolic transition graphs*). Ce sont des extensions probabilistes des graphes de transitions symboliques présentés dans [HL95], et utilisés pour les processus du π-calcul [**?**]. Les PSTGs peuvent être considérés

comme des extensions symboliques des MDPs [NPPW09]. Ils se définissent de la manière suivante :

Définition 2.2. Le graphe des transitions symboliques probabilistes représentant la sémantique d'un processus P exprimé en π_{prob} est un n-uplet $(S, s_{init}, \tau_{prob})$ où :

- S est l'ensemble des états symboliques, chacun d'eux étant un terme de π_{prob}.
- $s_{init} \in S$, l'état initial, est le terme P.
- $\tau_{prob} \subseteq S \times Cond \times Act \times Dist(S)$ est la relation des transitions symboliques probabiliste définie à l'article [NPPW09]. $Cond$ est l'ensemble de toutes les conditions (c'est-à-dire des conjonctions finies de correspondances) sur $\mathcal{N}$. Act est l'ensemble d'actions : τ, $x(y)$, $\bar{x}y$ et $x(\bar{y})$, où $x, y \in \mathcal{N}$. $Dist(S)$ est l'ensemble des distributions de probabilités sur S.

La transition symbolique probabiliste $(Q, M, \alpha, \mu) \in \tau_{prob}$ telle que $\mu(R) = \sum_{Q_i=R} p_i$ pour n'importe quel terme R du π_{prob} est notée $Q \xrightarrow{M,\alpha} \{|p_i : Q_i|\}_i$. Par ailleurs, la transition $Q \xrightarrow{M,\alpha} \{|1 : Q'|\}$ est notée plus simplement $Q \xrightarrow{M,\alpha} Q'$.

Un état symbolique Q code un ensemble de termes π_{prob}, obtenus à partir de Q par des substitutions de ses noms liés (ses « inconnues »). Une substitution σ est appliquée à un processus Q, ce que l'on dénote par $Q\sigma$, en remplaçant chaque action α dans Q par $\alpha\sigma$. Par exemple, à partir du processus $Q = a(x).\bar{x}b.0$, on a $Q \xrightarrow{a(x)} Q'$ avec $Q' = \bar{x}b.0$. L'état symbolique Q' représente tous les termes $Q'\{z/x\}$ pour n'importe quel nom libre z. La transition symbolique $Q \xrightarrow{M,\alpha} \{|p_i : Q_i|\}_i$ représente le fait que, sous n'importe quelle substitution σ satisfaisant M, le terme de processus $Q\sigma$ peut réaliser l'action $\alpha\sigma$, avant d'évoluer en $Q_i\sigma$ avec la probabilité p_i.

3.4 Le π-calcul stochastique

Une première extension stochastique du π-calcul a été proposée par C. Priami, dans laquelle les actions préfixées sont enrichies par des taux [Pri95]. Depuis, plusieurs variantes ont été proposées, se distinguant sur la façon dont les taux sont ajoutées au préfixe. Nous présentons la version stochastique du π-calcul proposée dans [NPPW09]. Cette version annote une action silencieuse par le taux r auquel elle se produit, soit τ_r , et associe à chaque voie x le taux noté $taux(x)$.

3.4.1 Sa syntaxe

L'ensemble $\mathcal{P}^{\pi_{stoc}}$ des expressions des processus du π-calcul stochastique est défini par :

$$P ::= 0 \mid \alpha.P \mid P_1 + P_2 \mid P_1|P_2 \mid (\nu x)P \mid [x = y]P \mid A(y_1, \ldots, y_n).$$

où :

- Les actions α sont τ_r, $x(y)$, $\bar{x}y$, et $\bar{x}(y)$, où r est un nombre réel strictement positif.
- Les expressions de processus sont les mêmes qu'en π-calcul définies au paragraphe 2.2.1. A noter que les actions d'émission et de réception sur une voie x, se réalisent au taux de cette voie soit $taux(x)$. Par suite, les processus impliqués dans ces actions se synchronisent au même taux. Ce qui est différent de la sémantique de PEPA, où les taux sont associés à des actions, et où deux processus qui réalisent ensembles les actions d'un ensemble $\mathcal{L}$, ce que l'on note par $P \underset{\mathcal{L}}{\bowtie} Q$ sont synchronisés sur le plus lent des deux [Hil05a, Hil05b].

3.4.2 Présentation de sa sémantique symbolique

La sémantique des processus du π_{stoc} s'exprime à l'aide de *chaîne de Markov à temps continu* (CTMCs) présentées à l'annexe A. Chaque transition est étiquetée avec un taux qui représente le paramètre d'une distribution exponentielle, caractérisant le délais nécessaire au franchissement de la transition. Pour un taux r, la probabilité que la transition soit franchie en t unités de temps est donnée par $1 - e^{-r.t}$. Habituellement, une sémantique concrète est présentée qui associe chaque terme d'un processus directement avec la CTMC qu'il représente. Pour adopter une approche par composition, on adopte une sémantique symbolique basée sur une extension des graphes de transition symbolique [HL95].

Définition 2.3. Le graphe des transitions symboliques stochastique (SSTG, *stochastic symbolic transition graph*) représentant la sémantique d'un processus P du π_{stoc}, est un n-uplet $(S, s_{init}, \tau_{stoc})$ où :

- S est l'ensemble des états symboliques, chacun d'eux étant un terme de π_{stoc}.
- $s_{init} \in S$, l'état initial, est le terme P.
- $\tau_{stoc} \subseteq S \times Cond \times Act \times S$ est la multi-relation stochastique des transitions symboliques définie dans l'article [NPPW09]. $Cond$ dénote l'ensemble de toutes les conditions (conjonctions finis de match) sur $\mathcal{N}$. Act est l'ensemble d'actions : τ_r, $x(y)$, $\bar{x}y$ et $x(\bar{y})$, où $r \in \mathbb{R}_{>0}$, et $x, y \in \mathcal{N}$.

De façon analogue au PSTGs, une transition symbolique stochastique $Q \xrightarrow{M,\alpha} Q'$ d'une SSTG représente le fait que, sous n'importe quelle substitution σ qui satisfait M, le terme $Q\sigma$ peut réaliser l'action $\alpha\sigma$, avant d'évoluer en $Q'\sigma$.

3.5 Le π-calcul temporisé

Le π-calcul ne permet pas de modéliser les applications mobiles temps réels. Pour ce faire, plusieurs extensions temporisées de cette algèbre de processus ont été proposées. Ainsi, Jeremy Y. Lee & John Zic ont proposé le *πRT-calcul* qui étend le π-calcul en permettant d'inhiber les actions d'un processus avant une échéance temporelle [LZ02]. Dans sa thèse de doctorat, Martin Berger propose le π-calcul temporisé asynchrone, ou *π_t-calcul*, définissant des processus-chronomètres (*timers*) pour modéliser les échéances temporelles [Ber04]. Ces deux extensions temporelles du π-calcul se ressemblent sur ce point. Par contre, M. Berger étend son calcul par la définition de localités, pour avoir une algèbre de processus destinée à analyser les systèmes distribuées. Christian Prisacariu, alors un jeune chercheur supervisé par Gabriel Ciobanu, propose le π-calcul distribué temporisé, ou $tD\pi$, qui définit des chronomètres pour les voies de communications (les noms), ainsi que des chronomètres pour des types de voies [PC05]. Il existe d'autres versions temporisées du π-calcul : notre intention n'étant pas d'en faire la liste exhaustive, mais plutôt de montrer que l'on peut considérer le temps dans les interactions du π-calcul, nous arrêtons cette description à ces trois exemples.

Par contre, nous avons choisi de présenter plus en détail le πRT-calcul de J. Y. Lee & J. Zic [LZ02], parce qu'il allie la simplicité avec la puissance d'expression. Ce calcul définit un opérateur d'échéances temporelles, que nous appelons dans la suite l'*opérateur de timeout*, qui fait référence à un paramètre temporel t pour inhiber les actions d'un processus pendant une durée égale à t. Comme les voies de communications, c'est-à-dire les noms, ces paramètres temporels peuvent être échangés par les processus. Il est alors possible de modéliser des comportements temporels dynamiques, et des configurations dynamiques des systèmes. La sémantique du πRT-calcul est décrite par des *systèmes de transitions temporisées* qui sont aussi utilisés pour décrire la sémantique des automates temporisés [Bé08] (voir chapitre 3).

Ainsi, intuitivement, un système décrit en πRT-calcul peut également être décrit par un automate temporisé.

3.5.1 Sa syntaxe

La πRT-calcul se caractérise sur les particularités suivantes :

- Il n'y a qu'une seule horloge dans le modèle.
- Le domaine du temps $\mathcal{D}$ est l'ensemble $\mathbb{N}$ des entiers naturels : le temps est discret et il est strictement croissant. La portée de $\mathcal{D}$ est parcourue par les noms t et u. Ces noms ne peuvent pas être utilisés comme des voies de communications. Le passage du temps représente un événement noté $\varepsilon(t)$, où ε n'est pas un nom, et le nom t représente la durée de cet événement exprimée en tops d'horloge. L'ensemble de tous les événements temporels est $\Delta = \{\varepsilon(t) \mid t \in \mathcal{D}\}$.
- Les actions des processus sont distinguées des événements temporels. Le temps peut s'écouler sans que des actions soient réalisées par les processus.
- Le temps progresse de façon synchrone, c'est-à-dire que les événements temporels sont observés par tous les composants du système, et ils peuvent se produire seulement si tous les composants sont d'accord pour y participer.
- Le temps peut être échangé par les processus à la manière des noms qui leurs permettent de communiquer. Les échanges de noms décrivent le dynamisme des configurations des processus, et l'échange du temps décrit leur comportement temporel dynamique.

L'ensemble $\mathcal{P}^{\pi RT}$ des expressions des processus du πRT-calcul est défini par :

$$P ::= 0 \mid \alpha_i.P \mid P_1 + P_2 \mid P_1|P_2 \mid P \overset{t}{\triangleright} Q \mid (\nu x)P \mid [x = y]P \mid A(y_1, \ldots, y_n).$$

où :

- Les actions α_i sont : τ_r, $x(y)$, et $\bar{x}y$; l'émission d'une référence de nom n'est pas définie.
- Le processus $P \overset{t}{\triangleright} Q$ se comporte comme le processus P avant t unités de temps, et comme le processus Q à partir, et après, t unités de temps. La première action de P est autorisée, et peut être exécutée, sur l'intervalle temporel $[0, t[$ durant lequel Q est ignoré. Mais, si cette action n'est pas exécutée pendant cet intervalle, alors, à partir de t unités de temps et après cette date, la première action de Q est autorisée, et peut donc être exécutée, alors que P est ignoré. Si une action de P est exécutée alors Q est ignorée. Les processus P et Q sont appelés respectivement le *corps* et l'*exception* de l'opérateur de *timeout*. Par exemple $0 \overset{t}{\triangleright} P$ spécifie que P est retardé d'une durée égale à t unités de temps. Le paramètre t est considéré comme un nom libre : $fn(P \overset{t}{\triangleright} Q) = fn(P) \cup fn(Q) \cup \{t\}$. Il peut être substitué par un nom transmis durant l'exécution. Ce qui contribue au comportement dynamique de l'opérateur de *timeout*.
- Les autres expressions de processus sont les mêmes qu'en π-calcul définies au paragraphe 2.2.1.

3.5.2 Sa sémantique

Les règles de congruence structurelles se rapportant à l'opérateur de *timeout* énoncées au tableau 2.6, complètent celles du π-calcul énoncées par le tableau 2.4 qui, bien évidemment,

$P \overset{t}{\triangleright} Q \overset{u}{\triangleright} R \equiv P \overset{t}{\triangleright} (Q \overset{u}{\triangleright} R)$	(TIMEOUT ASSOC RIGHT)
$P \overset{0}{\triangleright} Q \equiv Q$	(TIMEOUT ZERO)
$(\nu x)(P \overset{t}{\triangleright} Q) \equiv (\nu x)P \overset{t}{\triangleright} Q$ si $x \notin fn(Q)$	(TIMEOUT EX 1)
$(\nu x)(P \overset{t}{\triangleright} Q) \equiv P \overset{t}{\triangleright} (\nu x)Q$ si $x \notin fn(P)$	(TIMEOUT EX 2)
$P \overset{t}{\triangleright} Q + R \overset{t}{\triangleright} S \equiv (P + R) \overset{t}{\triangleright} (Q + S)$	(TIMEOUT CHOICE)

TAB. 2.6 – Règles additionnelles de congruence structurelle entre processus du πRT-calcul.

restent vraies pour le πRT-calcul.

La sémantique opérationnelle (c'est-à-dire les règles de transitions entre processus) du πRT-calcul est exprimée au tableau 2.7. Elles complètent celles du π-calcul, énoncées au tableau 2.5, qui restent vraies.

$$\text{INACTION-DEL} : \frac{-}{0 \xrightarrow{\varepsilon(t)} 0}$$

$$\text{OUTPUT-DEL} : \frac{-}{\bar{x}y.P \xrightarrow{\varepsilon(t)} \bar{x}y.P} \qquad \text{INPUT-DEL} \frac{-}{x(z).P \xrightarrow{\varepsilon(t)} x(z).P}$$

$$\text{TIMEOUT-ACT} : \frac{P \xrightarrow{\alpha} P'}{P \overset{t}{\triangleright} Q \xrightarrow{\alpha} P'} \qquad \text{TIMEOUT-DEL} : \frac{P \xrightarrow{\varepsilon(u)} P'}{P \overset{t}{\triangleright} Q \xrightarrow{\varepsilon(u)} P' \overset{t-u}{\triangleright} Q} \; 0 < u \leqslant t$$

$$\text{SUM-DEL} : \frac{P \xrightarrow{\varepsilon(t)} P' \quad Q \xrightarrow{\varepsilon(t)} Q'}{P + Q \xrightarrow{\varepsilon(t)} P' + Q'} \qquad \text{PAR-DEL} : \frac{P \xrightarrow{\varepsilon(t)} P' \quad Q \xrightarrow{\varepsilon(t)} Q' \quad \neg(P|Q \xrightarrow{\tau})}{P|Q \xrightarrow{\varepsilon(t)} P'|Q'}$$

$$\text{MATCH RT} : \frac{P \xrightarrow{\gamma} P'}{[x = x]P \xrightarrow{\gamma} P'} \qquad \text{RES RT} : \frac{P \xrightarrow{\gamma} P'}{(\nu x)P \xrightarrow{\gamma} (\nu x)P'} \; x \notin n(\gamma)$$

TAB. 2.7 – Règles d'inférence additionnelles pour les transitions entre processus en πRT-calcul. Le domaine du temps étant $\mathbb{N}$, le paramètre t est strictement positif.

- Les règles INACTION-DEL, OUTPUT-DEL, et INPUT-DEL permettent au temps de progresser pour le processus inactif et les processus comportant des préfixes. La règle PAR-DEL évite les récursions non gardées, la source et la cible de ces transitions étant les mêmes.

- L'action silencieuse τ ne permet pas l'écoulement du temps.
- La règle TIMEOUT-ACT stipule que si une action permise du corps P est exécutée pendant la période $t - 1$, alors celle-ci continue son exécution par la suite, et donc l'exception Q est ignorée. L'exécution d'une action interne à P rend également caduque l'exception Q.

$P, Q ::=$	Processus :
$(\nu n)\ P$	restriction
0	inactivité
$P \mid Q$	composition
$!P$	réplication
$M[P]$	ambiant
$M.P$	action liée à une capacité
$(x).P$	input
$\langle M \rangle$	output asynchrone
$M ::=$	Messages :
x	variable
n	nom
$in\, M$	capacité à entrer dans l'ambiant M
$out\, M$	capacité à sortir de l'ambiant M
$open\, M$	capacité à ouvrir (à « dissoudre »la frontière de) l'ambiant M
ε	chemin vide (null)
$M.M'$	chemin composite

TAB. 2.8 – Syntaxe du calcul des ambiants (primitives de communication et de mobilité).

- La règle TIMEOUT-DEL permet au temps d'avancer, et réduit la période de *timeout* d'autant. Le corps P avance de la même durée.
- La règle SUM-DEL stipule que le choix entre processus avance d'une certaine durée, si ces processus avancent de la même durée.
- La règle PAR-DEL stipule que la composition parallèle de processus avance d'une certaine durée, si ces processus avancent de la même durée, et qu'aucune réaction ne se produit entre-eux pendant cette durée.
- Les règles MATCH RT et RES RT utilisent la méta-variable γ, ce qui signifie que ces règles peuvent également s'appliquer à des transitions temporelles, contrairement aux règles RES et OPEN énoncées au tableau 2.5 qui s'appliquent seulement aux actions portant sur les noms autres que des temps.

3.6 Conclusion

Nous avons voulu présenter, au travers de ce paragraphe, un panorama des formalismes issus du π-calcul caractérisant la labilité des systèmes. Pour compléter la vision que l'on doit avoir de la mobilité des systèmes, nous présentons dans la suite de ce chapitre, le calcul des ambients [CG98], puis les bigraphes [Mil01].

4 Le calcul des ambiants : un calcul de processus motiles

4.1 Présentation

Le *calcul des Ambiants* est une algèbre de processus conçue par Luca Cardelli et Andrew D. Gordon en 1998 [CG98]. La notion de localisation des processus, absente du π-calcul, y est très importante puisqu'elle c'est sur elle que repose la mobilité des processus. L'ambition de ses concepteurs était de traiter la mobilité des calculs effectués par des objets qui peuvent se déplacer, et les calculs effectués par du code mobile qui se déplace entre différents matériels informatiques (applettes, agents mobiles, etc.). Ces différents aspects de la mobilité

sont décrit dans un cadre formel unique qui comprend les agents mobiles, les ambiants qui ne sont rien d'autres que les localisations où ces agents interagissent entre-eux, et la mobilité des ambiants elle-même.

Un ambiant est une zone limitée par une frontière marquant son intérieur et son extérieur, et dans laquelle des processus peuvent interagir (des calculs peuvent se produire). La notion de zone délimitée est une notion clé pour représenter la mobilité puisqu'une frontière délimite une zone pouvant contenir quelque chose, ou ne pas le contenir, et qui peut être déplacé globalement. Un ambiant est caractérisée par son nom qui va permettre de contrôler les accès à cet ambiant, et son contenu. Un ambiant peut être inclus dans d'autres ambiants (qui représenteront par exemple des domaines administratifs). D'une certaine manière les processus contenu dans un ambiant permettent de le contrôler, par exemple ils peuvent commander son déplacement.

L'existence de localisations séparées est représentée par une topologies des frontières. Cette topologie induit une notion abstraite de distance entre des localisations. La mobilité des processus s'exprime par la traversée de frontières, et non pas par la communication de processus ou de noms de processus par des voie de communications. La sécurité est représentée par la possibilité ou non de traverser des frontières, et non pas par l'existence de primitives cryptographiques ou des listes de contrôle d'accès. L'interaction entre processus se fait par des localisations partagées à l'intérieur de frontières communes.

Parmi les exemples d'ambiants, nous trouvons une page web (limitée à un fichier), un espace d'adressage virtuel (limité à une fenêtre d'adressage), un système de fichiers Unix (limité à l'intérieur d'un volume physique), une donnée-objet (limité à elle-même) ou encore un ordinateur portable (limité à ses ports).

4.2 Syntaxe et sémantique du calcul

Le calcul des ambiants manipule des noms, des processus et des capacités qui sont des primitives agissant sur les frontières d'un ambiant. Le tableau 2.8 représente la syntaxe du calcul des ambiants. Une relation de réduction $P \longrightarrow Q$ décrit l'évolution d'un processus P en processus Q.

Processus. La notion de processus du calcul des ambiants est similaire à celle du π-calcul. Les notions de processus inactif (0), de composition parallèle de processus $(P|Q)$, et de réplication d'un processus $(!P)$ sont également similaires. Il en est de même de la restriction de nom $(\nu n)\ P$, avec, toutefois, une différence importante, car si en π-calcul n est un nom, sous-entendu de voie de communication, il s'agit du nom d'un ambiant dans l'algèbre de processus de Luca Cardelli et Andrew D. Gordon.

Noms. La différence entre la notion de nom du π-calcul et celle du calcul des ambiants vient d'être évoquée au paragraphe précédent. La substitution de nom est également présente dans le calcul des ambiants : la substitution de chaque occurrence libre du nom n dans le processus P par le nom m, est notée $P\{n \leftarrow m\}$. Il en est de même dans $M\{n \leftarrow m\}$ pour l'ambiant M. Le tableau 2.9 donne la définition des noms libres fn de chaque expression.

Variables. Cette notion est absente du π-calcul. Les variables du calcul des ambiants peuvent être instanciées par des noms ou des capacités. La substitution de chaque occurrence libre de la variable x dans le processus P par la capacité M, est notée $P\{x \leftarrow M\}$. Il

$fn((\nu n)P) \triangleq fn(P) - \{n\}$	$fn(M[P]) \triangleq fn(M) \cup fn(P)$
$fn(0) \triangleq \emptyset$	$fn((x).P) \triangleq fn(P)$
$fn(P\|Q) \triangleq fn(P) \cup fn(Q)$	$fn(\langle M \rangle) \triangleq fn(M)$
$fn(!P) \triangleq fn(P)$	$fn(x) \triangleq \emptyset$
$fn(n[P]) \triangleq \{n\} \cup fn(P)$	$fn(n) \triangleq \{n\}$
$fn(M.P) \triangleq fn(M) \cup fn(P)$	$fn(\varepsilon) \triangleq \emptyset$
$fn(in\, n) \triangleq \{n\}$	$fn(M.M') \triangleq fn(M) \cup fn(M')$
$fn(out\, n) \triangleq \{n\}$	
$fn(open\, n) \triangleq \{n\}$	

TAB. 2.9 – Noms libres du calcul des ambiants.

en est de même de $M\{x \leftarrow M'\}$ pour l'ambiant M. Les variables libres sont définies par le tableau 2.10.

$fv((\nu n)P) \triangleq fv(P)$	$fv(x) \triangleq \{x\}$
$fv(0) \triangleq \emptyset$	$fv(n) \triangleq \emptyset$
$fv(P\|Q) \triangleq fv(P) \cup fv(Q)$	$fv(in\ M) \triangleq fv(M)$
$fv(!P) \triangleq fv(P)$	$fv(out\ M) \triangleq fv(M)$
$fv(M.P) \triangleq fv(M) \cup fv(P)$	$fv(open\ M) \triangleq fv(M)$
$fv(M.P) \triangleq fv(M) \cup fv(P)$	$fv(\varepsilon) \triangleq \emptyset$
$fv((x).P) \triangleq fv(P) \setminus \{x\}$	$fv(M.M') \triangleq fv(M) \cup fv(M')$
$fv(\langle M \rangle) \triangleq fv(M)$	

TAB. 2.10 – Variables libres du calcul des ambiants.

Ambiant. Un ambiant est noté $n[P]$, où n est le nom de l'ambiant, et P est un processus qui s'exécute à l'intérieur de n. Ce dernier peut être la composition parallèle de plusieurs processus, et il s'exécute même si n se déplace. Son exécution s'exprime par la règle suivante, qui exprime le fait que n'importe quelle réduction de P devient une réduction de $n[P]$:

$$P \rightarrow Q \Rightarrow n[P] \rightarrow n[Q]$$

Un ambiant exhibe une structure arborescente induite par l'imbrication des ambiants contenus. Chaque feuille de cet arbre peut contenir un ensemble de processus qui s'exécutent en parallèle, ou des sous-ambiants. C'est pourquoi un ambiant est généralement de la forme :

$$n[P_1 \mid P_2 \mid \ldots \mid P_n \mid m_1[\ldots] \mid m_2[\ldots] \mid \ldots \mid m_k[\ldots]\,] \ (\forall i, P_i \neq n_i[\ldots])$$

Plusieurs ambiants peuvent avoir le même nom, qu'il soient au même niveau, ou qu'ils soient imbriqués. Par exemple $!n[P]$ génère plusieurs ambiants qui portent le même nom.

Actions liées à une capacité. La structure hiérarchique des ambiants peut être modifiée par des actions spéciales appelées *capacités*. Le processus $M.P$ exécute une action contrôlée par la capacité M, avant de se comporter comme le processus P. Trois sortes de capacités M se distinguent par leur action respective : *in* m qui provoque l'entrée à l'intérieur de l'ambiant m, *out* m qui permet d'en sortir, et *open* m qui supprime cet ambiant.

- *in* $m.P$ provoque l'entrée de l'ambiant qui englobe *in* $m.P$ dans l'ambiant m. Si ce dernier n'existe pas, alors l'opération est bloquante tant qu'il n'apparaît pas. Si plusieurs ambiants appelés m existe, alors l'un d'entre-eux est choisi arbitrairement. L'opération de réduction est la suivante :

$$n[in\ m.P|Q] \mid m[R] \rightarrow m[n[P|Q] \mid R]$$

– *out* $m.P$ provoque la sortie de l'ambiant englobant *out* $m.P$ de son ambiant parent m. Si l'ambiant parent ne s'appelle pas m, l'opération est bloquante tant qu'il n'apparait pas. La règle de réduction est :

$$m[n[out\ m.P|Q] \mid R] \rightarrow n[P|Q] \mid m[R]$$

– *open* $m.P$ supprime l'ambiant m ou, plus précisément, provoque la dissolution de la frontière de l'ambiant m. Cette capacité est bloquante tant que l'ambiant m n'existe pas. Si plusieurs ambiants de nom m existent, alors l'un d'eux est choisi arbitrairement. La règle de réduction correspondante est :

$$open\ m.P \mid m[Q] \rightarrow P \mid Q$$

Communications. Le mécanisme de communication le plus simple est la communication locale asynchrone à l'intérieur d'un ambiant :

$$(x).P \text{ (input)}$$
$$\langle M \rangle \text{ (output asynchrone)}$$

Une action de sortie (*output*) produit une capacité (éventuellement un nom) dans l'environnement local de l'ambiant englobant. Une action d'entrée (*input*) capture une capacité de l'environnement local, et la lie à une variable à l'intérieur d'un périmètre donné.

4.3 Sémantique du calcul des ambients

La sémantique du calcul des ambients est définie à partir de la congruence structurelle entre les processus notée $\equiv$, et la relation de réduction $\longrightarrow$.

4.3.1 Congruence structurelle entre les processus

4.3.2 Règles de réduction

Les règles de réduction du calcul des ambients sont définies par le tableau 2.12.

4.4 Exemple

Un message *msg* se déplace de la localité a à la localité b, où il est traité, sous les actions séquentielles des capacités *out* a (qui lui permet de sortir de a), *in* b (qui lui permet d'entrer dans b), et *open msg* (qui permet de l'ouvrir).

	$a[msg[\langle M \rangle \mid out\ a.in\ b]]$	$\mid$	$b[open\ msg.(n).P]$	
$\longrightarrow$	$a[0] \mid msg[\langle M \rangle \mid in\ b]$	$\mid$	$b[open\ msg.(n).P]$	(avec Red Out)
$\longrightarrow$	$a[0]$	$\mid$	$b[msg[\langle M \rangle] \mid open\ msg.(n).P]$	(avec Red In)
$\longrightarrow$	$a[0]$	$\mid$	$b[\langle M \rangle \mid (n).P]$	(avec Red Open)
$\longrightarrow$	$a[0]$	$\mid$	$b[P\{n \leftarrow M\}]$	(avec Red Comm)

5 Introduction aux bigraphes

La notion de *bigraphe* a été proposée au début des années 2000 par Robin Milner, autour duquel ce sont rassemblés depuis de nombreux auteurs, pour modéliser graphiquement les systèmes communicants mobiles [Mil01]. Un bigraphe est la composition de deux graphes indépendants : un *graphe des liens*, ou monographe, qui représente la connectivité des composants du système modélisé, et un *graphe des places*, ou topographe, qui représente leur localité respective. Ainsi, l'idée majeure des bigraphes est de faire la synthèse entre les notions de

$P \equiv P$	(Struct Refl)
$P \equiv Q \Rightarrow Q \equiv P$	(Struct Symm)
$P \equiv Q,\ Q \equiv R \Rightarrow P \equiv R$	(Struct Trans)
$P \equiv Q \Rightarrow (\nu n)P \equiv (\nu n)Q$	(Struct Res)
$P \equiv Q \Rightarrow P\|R \equiv Q\|R$	(Struct Par)
$P \equiv Q \Rightarrow !P \equiv !Q$	(Struct Repl)
$P \equiv Q \Rightarrow n[P] \equiv n[Q]$	(Struct Amb)
$P \equiv Q \Rightarrow M.P \equiv M.Q$	(Struct Action)
$P\|Q \equiv Q\|P$	(Struct Par Comm)
$(P\|Q)\|R \equiv P\|(Q\|R)$	(Struct Par Assoc)
$!P \equiv P\|!P$	(Struct Repl Par)
$(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P$	(Struct Res Res)
$(\nu n)(P\|Q) \equiv P\|(\nu n)Q$ *si* $n \notin fn(P)$	(Struct Res Par)
$(\nu n)(m[P]) \equiv m[(\nu n)P]$ *si* $n \neq m$	(Struct Res Amb)
$P\|0 \equiv P$	(Struct Zero Par)
$(\nu n)0 \equiv 0$	(Struct Zero Res)
$!0 \equiv 0$	(Struct Zero Repl)
$P \equiv Q \Rightarrow M[P] \equiv M[Q]$	(Struct Amb)
$P \equiv Q \Rightarrow (x).P \equiv (x).Q$	(Struct Input)
$\varepsilon P \equiv P$	(Struct ε)
$(M.M').P \equiv M.M'.P$	(Struct .)

Tab. 2.11 – Congruence structurelle entre processus du calcul des ambiants.

$n[in\ m.P\|Q]\|m[R] \rightarrow m[n[P\|Q]\|R]$	(Red In)
$m[n[out\ m.P\|Q]\|R] \rightarrow n[P\|Q]\|m[R]$	(Red Out)
$open\ n.P\|n[Q] \rightarrow P\|Q$	(Red Open)
$P \rightarrow Q \Longrightarrow (\nu n)P \rightarrow (\nu n)Q$	(Red Res)
$P \rightarrow Q \Longrightarrow n[P] \rightarrow n[Q]$	(Red Amb)
$P \rightarrow Q \Longrightarrow P\|R \rightarrow Q\|R$	(Red Par)
$P' \equiv P,\ P \rightarrow Q,\ Q \equiv Q' \Longrightarrow P' \rightarrow Q'$	(Red $\equiv$)
$(x).P\|\langle M\rangle \rightarrow P\{x \leftarrow M\}$	(Red Comm)

Tab. 2.12 – Sémantique des règles de réduction du calcul des ambiants.

connectivité et de *localité* qui, comme nous l'avons évoqué, sont sous-jacentes à la notion de mobilité des systèmes. Par ailleurs, la structure statique des bigraphes peut évoluer sous le contrôle de *règles de réactions*, qui ne sont rien d'autres que des règles de réécriture des bigraphes. Cette dynamique est modélisée par les *systèmes bigraphiques réactifs* (*bigraphical reactive systems*), dont une des propriétés est d'inclure les règles de versions du π-calcul et du calcul des ambiants [JM04, Mil09].

5.1 Définitions et exemple

Ainsi un bigraphe, ou plus précisément un *bigraphe nu*, est composée du *graphe des places* et le *graphe des liens*, qui se partagent les nœuds représentant les composants du système modélisé. Le graphe des places est une forêt qui représente l'imbrication des nœuds. Le graphe des liens, quant-à-lui, est un hypergraphe, c'est-à-dire un graphe dont les arcs peuvent connecter plusieurs éléments, qui représente la connectivité des nœuds.

Interfaces. Un bigraphe G est un bigraphe nu doté de deux *interfaces* (ou *faces*) interne I_i et externe I_e, qui définissent sa possibilité d'être utilisé, en tant que bloc de construction, pour élaborer des bigraphes plus grands. Le bigraphe G est noté $G : I_i \longrightarrow I_e$. Il est possible d'insérer un bigraphe A dans un bigraphe B pour obtenir le bigraphe noté $B \circ A$, uniquement si l'interface interne de B est identique à l'interface externe de A. Ce qui revient à écrire :

$$B \circ A : I \longrightarrow J \Longleftrightarrow A : I \longrightarrow I' \text{ et } B : I' \longrightarrow J.$$

La notion d'interfaces d'un bigraphe est décomposée en notions d'interfaces de chacun de ces deux graphes. Les éléments de l'interface interne du graphe des places sont appelés des *sites*, alors que ceux de son interface externe sont appelés des *racines*. Intuitivement, les sites d'un bigraphes sont des « prises »(Robin Milner parle de « trous ») dans lesquels les racines d'un autre bigraphe peuvent être branchées. Le graphe des places est habituellement représenté de la façon suivante :

$$\textit{nombre de sites} \longrightarrow \textit{nombre de racines}$$

Les arcs sortant du graphe des liens sont reliés aux *noms externes* et aux *noms internes* du bigraphe. Le graphe des liens est représenté par leur ensemble respectif, de la façon suivante :

$$\{\textit{noms internes}\} \longrightarrow \{\textit{noms externes}\}$$

Finalement, l'interface externe d'un bigraphe $B = \langle B^P, B^L \rangle$, est la paire $\langle n, Y \rangle$, où n et Y sont respectivement les interfaces externes de B^P et de B^L. Son interface interne $\langle m, X \rangle$ est définie de la même façon. Parmi les interfaces possibles, on distingue l'interface triviale $\langle 0, \emptyset \rangle$, notée ϵ, qui est appelée l'*origine*. Un bigraphe G est donc représenté par :

$$\langle \textit{nombre de sites}, \{\textit{noms internes}\} \rangle \longrightarrow \langle \textit{nombre de racines}, \{\textit{noms externes}\} \rangle$$

Signature. On définit une *signature* (K, ar) où K est un ensemble dont les éléments sont appelés des *contrôles*, et $ar : K \to \mathbb{N}$ est une fonction qui associe un entier naturel, appelé *arité*, à chaque contrôle. Un bigraphe sur K assigne un contrôle à chacun de ses nœuds, dont l'arité détermine le nombre de ses ports.

Exemple. La figure 2.6 représente le bigraphe $G = \langle G^P, G^L \rangle$, avec $G^P : 2 \longrightarrow 2$ et $G^L : \{x_0, x_1\} \longrightarrow \{y_0, y_1, y_2\}$. Soit $G : \langle 2, \{x_0, x_1\} \rangle \longrightarrow \langle 2, \{y_0, y_1, y_2\} \rangle$. La signature de G est $\mathbb{K} = \{K : 0, L : 2, M : 4\}$.

5.2 Les systèmes bigraphiques réactifs

5.2.1 La dynamique dans les bigraphes

Un bigraphe peut se reconfigurer à l'aide de règles de réaction (ou réécriture). Chacune de ces règles consiste en un *redex* (le bigraphe avant sa modification) et un *reactum* (le bigraphe modifié). Les modifications concernent le placement (l'imbrication) et les liens des nœuds.

La figure 2.7 représente la règle de communication suivante du π-calcul :

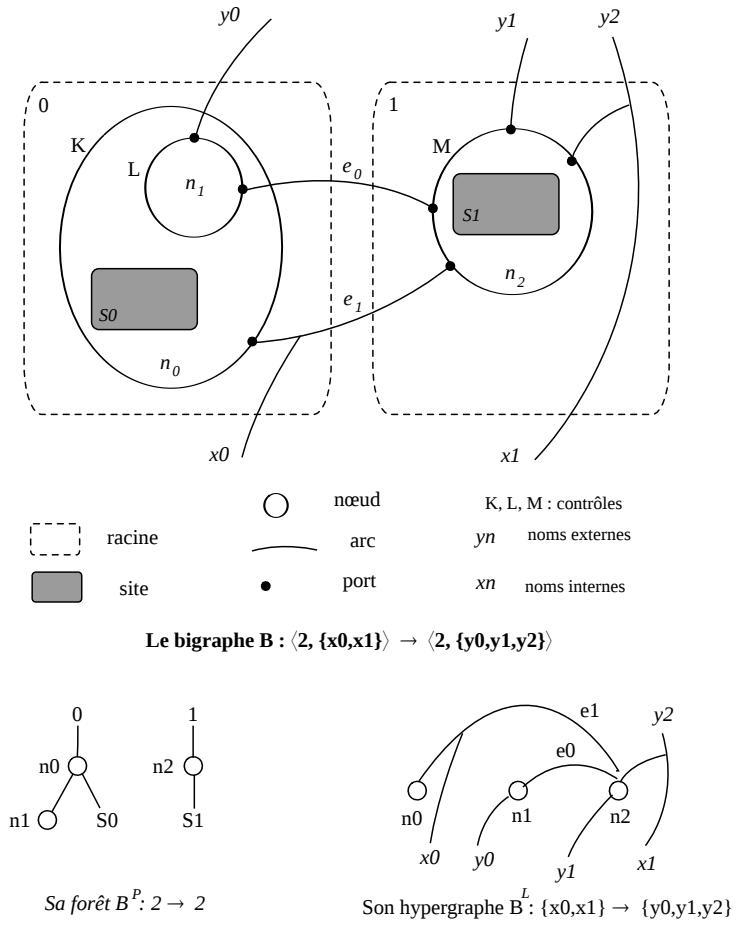


FIG. 2.6 – Exemple de bigraphe décomposé en sa forêt et son hypergraphe.

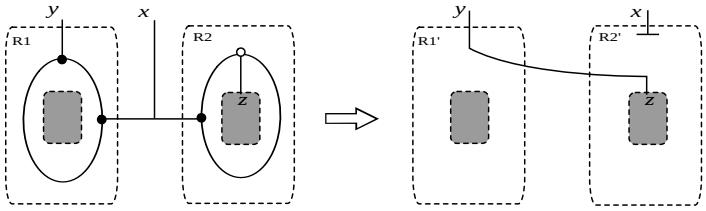


FIG. 2.7 – Exemple de système bigraphique réactif.

$$\bar{x}y.P|x(z).Q \longrightarrow P|Q\{y/z\}$$

Cette règle énonce l'envoi, par le processus $\bar{x}y.P$, du nom y, au processus $x(z).Q$ via le nom x qu'ils partagent.

Dans la figure 2.7, $R1\|R2$ forment le redex, alors que $R1'\|R2'$ forment le réactum. Le nœud $n1$ a pour contrôle *Send*, et a deux ports auxquels sont liés y et x. Le nœud $n2$ a pour contrôle *Get*, et a également deux ports ; l'un est lié à x, alors que l'autre peut est un port liant lié à d'autre nœuds ou sites (*binding port*). Après la réaction, les nœuds $n1$ et $n2$ disparaissent, et x ne forme plus une voie entre-eux. L'expression $\{y/z\}$ est représentée par l'arc courbe. Ainsi le processus $\bar{x}y.P$ est représenté par $R1$ dans le redex, alors que $x(z).Q$ est représenté par $R2$. Après la communication, le processus P est représenté par $R1'$, alors que le processus $Q\{y/z\}$ est représenté par $R2'$.

Chapitre 3

Extensions du π-calcul pour modéliser les systèmes communicants mobiles

Nous avons présenté au chapitre 2, le π-calcul [MPW92] et son intérêt pour modéliser des systèmes mobiles. Cependant, ces systèmes ont une particularité : ils communiquent essentiellement par diffusion locale. Or, le π-calcul ne permet pas de modéliser ces communications puisque la seule primitive de communication qui y est définie est la communication en point-à-point synchrone (dans la version originale définie dans [MPW92]). Donc si l'on veut utiliser cet algèbre pour modéliser les systèmes communicants mobiles, il est nécessaire de l'étendre, ne serait-ce que pour y inclure les diffusions. Par ailleurs, on peut vouloir l'étendre pour y inclure des paramètres permettant de prendre en compte des paramètres stochastiques et temporels des systèmes. De nombreux travaux ont déjà été réalisés. Nous proposons une démarche pour construire des extensions du π-calcul dont la sémantique est définie par des structures à états-transitions permettant des analyses de performabilité par model checking. Ces structures sont les automates temporisés et les automates temporisés probabilistes, qui sont supportés par la dernière version de l'outil PRISM [PRIb, PRIa]. La définition de règles de traduction entre une algèbre issue du π-calcul et le langage PRISM, nous permet de traduire les modèles de haut niveau des systèmes communicants mobiles en automates temporisés, ou en automates temporisés probabilistes permettant d'analyser la performabilité des systèmes.

1 Introduction : une démarche pour étendre le π-calcul

Les techniques de model checking consistent à vérifier la véracité d'une formule de logique temporelle exprimant une exigence de performabilité, par rapport à un modèle de bas niveau d'un système qui est une structure de Kripke, ou une extension de ce formalisme. Les *chaînes de Markov à temps discret* et les *processus de Markov* sont les supports du model checking probabiliste [KNP10]. Les *chaînes de Markov à temps continu* sont le support du model checking stochastique [KNP10]. Le model checking temporisé s'appuie sur les *automates temporisés* [Bé08], alors que les *automates temporisés probabilistes* permettent de vérifier des systèmes présentant des comportements probabilistes et temps réel [KNP10]. Nous pouvons ajouter à cette liste, les *automates hybrides* qui supportent le model checking des systèmes exhibant un comportement hybride [ACHH92, HHWT97].

Cependant, à notre avis, ces formalismes sont de trop bas niveau pour être directement utilisés pour modéliser des systèmes mobiles qui présentent des caractères probabilistes, stochastiques, temps réel ou hybrides. Pour ce faire, il nous paraît plus simple d'utiliser une algèbre de processus qui étend le π-calcul. La version de base de cette algèbre permet uniquement d'exprimer la mobilité des systèmes, pas leurs caractères probabiliste, stochastique,

temps réel, ou hybride. Pour ce faire, il est nécessaire d'étendre la version de base du π-calcul en y incluant le caractère désiré, et en donnant à cette algèbre une sémantique exprimée par le formalisme à états-transitions correspondant, support à des analyses par model checking. La figure 3.1 résume l'association entre un formalisme support au model checking et l'algèbre de processus issue du π-calcul correspondante.

Les extensions probabilistes et stochastiques du π-calcul ont été proposées par Gethin Norman, Catuscia Palamidessi, David Parker, et Peng Wu dans [NPPW09]. Dans ce chapitre, nous en proposons une extension temporisée non probabiliste, ainsi qu'une extension temporisée et probabiliste permettant de spécifier des modèles des systèmes exhibant des comportements probabiliste et temps réel. Ces extensions sont complétées par les règles de traduction de cette algèbre en langage PRISM. Pourquoi avoir choisi cet outil de vérification ? Pour deux raisons : d'abord parce que la dernière version de cet outil supporte les automates temporisés probabilistes ; ensuite parce qu'ainsi nos travaux sont une extension des travaux présentés dans [NPPW09].

Algèbre de processus *(formalisme de haut niveau)*	**Expression de sa sémantique** *(formalisme de bas niveau)*
π-calcul	Structure de Kripke
π-calcul probabiliste	DTMC, MDP
π-calcul stochastique	CTMC
π-calcul temporisé	Automate temporisé
π-calcul temporisé probabiliste	Automate temporisé probabiliste

Tab. 3.1 – Expression de la sémantique des extensions du π-calcul [MPW92] en un formalisme à états-transitions. Nous pouvons ajouter à cette liste une version hybride du π-calcul, appelée le Φ-calcul [RS03], dont la sémantique s'exprime par des automates hybrides.

2 Les communications dans le π-calcul

Le π-calcul définit des communications synchrones en point-à-point entre deux processus. La notion de diffusion en est absente. Pourtant, c'est le mode de diffusion par défaut de nombreux systèmes mobiles, en particulier les réseaux de capteurs sans fil. Nous recensons dans ce paragraphe les caractéristiques des systèmes communicants, en mettant l'accent sur les particularités des communications sans fil.

2.1 Caractéristiques des communications sans fil

– La communication de base entre les nœuds du système est la *diffusion locale* de messages (*broadcast*). La diffusion dans les réseaux sans fil est différente de la diffusion dans les réseaux filaires de type Ethernet. En effet, dans ces derniers, la diffusion est globale au système, puisque les messages diffusés sont reçus par tous les nœuds. A l'inverse, dans les systèmes sans fil, la diffusion est locale car elle ne concerne que la zone de couverture de l'émetteur, c'est-à-dire la portion d'environnement où les transmissions de l'émetteur sont susceptibles d'être reçues, et que nous appellerons la *cellule* de l'émetteur. Cette zone définit la notion de *nœud voisin* d'un émetteur, qui sont les nœuds situés dans la cellule d'un émetteur, et qui donc peuvent recevoir ses transmissions en un seul saut (c'est-à-dire sans routage). Par suite, le *voisinage* d'un émetteur est l'ensemble de ses nœuds voisins. Une autre différence entre la diffusion dans les réseaux filaires

et la diffusion dans les réseaux sans fil, est que cette dernière utilise des fréquences de communication c'est-à-dire des voies de communication explicites.

- Les voies de communication d'un réseau filaire sont bidirectionnelles (*full-duplex*), ce qui signifie qu'un nœud peut transmettre et recevoir des messages en même temps. Grâce à la diffusion globale et aux voies de communication bidirectionnelles, les interférences entre deux transmissions simultanées dans un réseau filaire à contention sont immédiatement détectées par les nœuds émetteurs. Ces derniers sont donc immédiatement avertis d'interférences entre plusieurs transmissions simultanées et qu'il est nécessaire de retransmettre les messages. Par contre, dans les réseaux sans fil, les voies de communications sont bidirectionnelles à l'alternat (*half-duplex*). Cela signifie qu'un nœud peut transmettre *ou* recevoir, mais il ne peut pas faire les deux à la fois. Donc, une interférence entre deux transmissions n'est détectée que par les récepteurs qui se trouvent dans les zones de couverture des émetteurs, mais pas par les émetteurs eux-mêmes, comme c'est le cas dans les réseaux filaires de type Ethernet. L'absence d'un accusé de réception indiquera à ces derniers que la transmission a échoué, et qu'ils doivent réessayer leur émission après un temps d'attente choisi aléatoirement. Pour les réseaux sans fil, les interférences entre transmissions sont un problème plus délicat à traiter, et sont réglées par des protocoles utilisant, par exemple, la stratégie CSMA/CA (*carrier sense multiple access with collision avoidance*).
- Les communications entre les nœuds d'un système communicant sont *synchrones* et *asynchrones*. L'asynchronisme fait référence au canal de communication qui existe entre un émetteur et un récepteur. Si le médium se comporte comme une mémoire tampon de taille illimitée, ce qui est une abstraction réaliste dans Internet, le temps qui s'écoule entre l'émission d'un message et sa réception est limité mais il être peut arbitrairement grand. Dans un soucis de recherche d'une qualité de service optimale, il est donc raisonnable de ne pas bloquer les émetteurs tant que leurs messages n'ont pas été reçus, ou que leur réception n'a pas été confirmée par un accusé de réception. D'où l'asynchronisme des communications entre les nœuds distants d'un système communiquant. Cependant la synchronisation peut être une nécessité. Par exemple, certains nœuds des réseaux sans fil, tels que les réseaux de capteurs sans fil, sont ordonnancés en périodes d'activité suivies par des périodes de sommeil pendant lesquels leur activité est inexistante pour réduire leur consommation d'énergie embarquée. Il est clair qu'un émetteur ne peut communiquer *localement* qu'avec un récepteur éveillé, et par suite ces deux nœuds doivent être synchronisés. Pour ce faire, ils sont dotés d'horloges matérielles qui, généralement, dérivent de façon différente par rapport à la fréquence nominale. Afin de réduire les effets de ses dérives à long terme, chaque nœud resynchronise sa propre horloge à partir des transmissions avec les autres nœuds. Mais, pour les nœuds distants, la distance qui les sépare, et les cellules des transmetteurs, qui ont des tailles limitées, font que les horloges peuvent ne pas coïncider. Cet forme d'asynchronisme peut affecter la façon dont les collisions entre transmissions apparaissent et sont détectées. Finalement, si l'influence du médium de communication est négligeable, on considérera que les nœuds sont synchrones. C'est notamment le cas pour les nœuds d'un même voisinage. Au contraire, si cette influence n'est plus négligeable, alors les nœuds seront considérés comme asynchrones ; ce qui est la cas des nœuds distants d'un système.
- Avant de transmettre, un nœud d'un réseau sans fil vérifie que le canal de transmission n'est pas occupé *localement*.
- Du fait des liaisons sans fil, les nœuds peuvent être mobiles ce qui modifie le voisinage de chacun d'eux. Cette mobilité peut être prévisible, si on peut la restreindre pour chaque nœud, ou si elle est retreinte par les contraintes du terrain où sont déployés les

nœuds ; mais elle peut être imprévisible, voire aléatoire.

– Les voies de communications entre les nœuds ne peuvent pas toujours être considérées comme bidirectionnelles. A cause des contraintes du terrain, un nœud A peut communiquer avec un nœud voisin B sans que la réciproque soit vraie.

2.2 Travaux existants

Les communications par diffusion ont été analysées par K. V. S. Prasad dans CBS [Pra95]. Plus récemment, plusieurs algèbres de processus ont été développés destinées à modéliser des réseaux ad hoc mobiles : CBS# [NH06], CWS [MS06], CMN [Mer09], CMAN [God07], MBS [Pra06], ou encore l'ω-calcul [SRS10]. Ces algèbres définissent la communication par diffusion locale, et distinguent le comportement des nœuds composant un système, des informations concernant leur voisinage destinées à définir les diffusions.

CBS (*The calculus of broadcasting systems*) [Pra95] s'inspire de CCS de Robin Milner [Mil80]. L'objectif principal était de fournir un modèle formel pour les diffusion de paquets dans les communications de type Ethernet. CBS# de Sebastian Nanz et Chris Hankin se base sur CBS. L'objectif des auteurs était d'utiliser leur calcul pour spécifier et analyser les paramètres de sécurité des protocoles de communication des réseaux mobiles ad hoc. La connectivité des nœuds est définie indépendamment de la spécification d'un système à l'aide de graphes de connectivité des nœuds. Les effets de la mobilité des nœuds sont définis en choisissant, de façon indéterministe, un graphe de connectivité des nœuds parmi les graphes possibles. Les récepteurs d'une diffusion sont déterminés par le graphe des connectivités.

Dans CMN [Mer09], un nœud d'un réseau ad hoc mobile est un processus séquentiel local qui peut diffuser ses messages à l'intérieur d'une zone ayant un rayon spécifique. La localisation et le rayon de transmission sont des valeurs définies dans un système de coordonnées physique. Les nœuds sont désignés comme fixe ou mobile, et ces derniers peuvent se déplacer vers une localisation arbitraire résultant en une transition τ.

CMAN [God07] se base sur le passage de nom pour les réseaux ad hoc. La relation de voisinage est définie en terme de localisations logiques, et la topologie être une partie explicite de la modélisation du réseau. Cet algèbre ne considère pas les collisions entre transmissions, contrairement à CWS [MS06]. Chaque nœud a une localisation donnée. De plus, chaque nœud n est annoté par un ensemble de connexion qui est l'ensemble des localisations des nœuds avec lesquels n est connecté. Les ensembles de connections déterminent la topologie du réseau. La diffusion locale synchrone est l'unique primitive de communication entre les nœuds. L'ensemble des connections d'un nœud identifie explicitement ses voisins. Ainsi, quand un nœud bouge, ses voisins participent activement en supprimant, ou en ajoutant, de leur ensemble des connections respectif, la localisation du nœud mobile.

CWS (*Calculus of Wireless Systems*), de Nicola Mezzetti et Davide Sangiorgi, est une algèbre qui définit des nœuds fixes, représentant les éléments d'un réseau et des processus [MS06]. La mobilité est absente de CWS. Les processus modélisent les comportements des nœuds. Les processus des nœuds communiquent. Les nœuds peuvent être composés en parallèle. 0 est un nœud particulier. Chaque nœud n, noté $n[P]^c_{l,r}$, peut exécuter un processus P (éventuellement inactif, 0), est localisé en l, et a un rayon de communication égal à r. Le processus P est synchronisé sur la voie c. Les nœuds ne peuvent pas être créés ou détruits. L'identifiant n représente une localisation physique (les adresses du nœud dans le réseau). La définition des valeurs pour l et r nécessite l'existence d'un repère de coordonnées. Ces valeurs

permettent de dériver des informations sur les connectivités des nœuds dans le réseau. Enfin la collion entre plusieurs transmissions simultanées est modélisée $\perp$.

L'ω-calcul [SRS10] offre un langage simple et intégré pour spécifier le contrôle du comportement et les informations de connectivité, et permet de raisonner sur les changements de la connectivité des nœuds. Il s'agit d'une extension du π-calcul conçue pour modéliser les protocoles des réseaux ad hoc mobiles. La particularité majeure de cette algèbre est qu'elle sépare le comportement des nœuds (leur comportement et leurs calculs), de la description de leurs transmissions en particulier leur portée physique. Celle-ci est modélisée en annotant les processus de l'ensemble des groupes de noms auquel le nœud appartient, et avec qui il peut communiquer par diffusion. L'ω-calcul a été utilisée pour modéliser le protocole AODV. Au contraire, dans l'ω-calcul, les informations de voisinage sont mises à jour de manière implicite par la notion de groupe de nœuds. Un outil particulier est lié à cet algèbre, mais il ne permet pas de faire des analyses quantitatives.

Enfin, citons le $b\pi$-calcul [EM01], et PRISMA [BL08]. Le premier est une version du π-calcul où la primitive de communication est la diffusion, et non pas la communication par point à point. Utiliser le $b\pi$-calcul pour modéliser un système communicant induit des problèmes similaires à l'utilisation du π-calcul, sauf que dans ce cas, ce sont les communications en point à point que l'on ne peut pas modéliser. PRISMA est une algèbre paramétrique qui peut être instanciée avec différentes politiques d'interaction, et fournit un cadre uniforme pour exprimer différents modèles de synchronisation tel que les communications en point-à-point et par diffusion. La mobilité dans PRISMA est définie par l'échange de nom comme dans le π-calcul.

3 Construction du π_b-calcul

Nous organisons l'étude en trois étapes. Nous commençons d'abord par présenter le π_b-calcul de façon informelle, à partir d'un exemple de système communicant mobile. Puis, nous définissons la syntaxe et la sémantique de sa sous-algèbre consacrée aux diffusions locales, que nous avons appelée le $(\pi_b)_0$-calcul. Enfin, nous ajoutons à cette sous-algèbre, la syntaxe et les règles de sémantique permettant d'obtenir l'algèbre complète.

3.1 Une présentation informelle

Considérons le système $\mathcal{S}$ illustré par la figure 3.1, composé de six nœuds mobiles, nommés A, B, C, D, E, et F. Chaque nœud est représenté par un disque noir portant son nom, situé au centre d'un cercle en pointillés qui représente sa *cellule*, c'est-à-dire la zone géographique de couverture maximale de ses transmissions. Deux nœuds peuvent échanger des messages, uniquement s'ils sont situés dans la zone formée par l'intersection de leur cellule respective. Dans cet exemple, A peut communiquer avec B, avec C, ainsi qu'avec D, les réciproques sont également vraies. Par ailleurs, D peut communiquer avec E et réciproquement. Par contre, F n'est dans aucune cellule, hormis la sienne : il ne peut communiquer avec aucun autre nœud du système.

Système parfait. D'après cette représentation du système, le rayon des cellules de tous les nœuds sont les mêmes. Ce qui implique que si un nœud N_i peut communiquer avec un nœud N_j, alors N_j peut communiquer avec N_i. Cette convention de représentation ne reflète pas toujours la réalité. Par exemple, si l'énergie embarquée de N_j est insuffisante pour lui permettre d'envoyer des messages, il pourra en recevoir de N_i, sans avoir la possibilité de lui répondre. Dans ce cas leur cellule respective ont des rayons différents, celui de la cellule

FIG. 3.1 – Le système communicant $\mathcal{S}$ composé de six nœuds mobiles.

de N_j étant plus petit que celui de N_i. De plus, les cellules ne sont pas nécessairement des cercles : un obstacle opaque aux transmissions, par exemple un mur suffisamment épais d'une maison, déforme la zone de couverture des transmissions d'un nœud. Ainsi, notre convention de représentation correspond à un *système parfait*, où les cellules des nœuds sont des disques dans un espace à deux dimensions, ou des sphères pleines dans un espace à trois dimensions, de même rayon. Pour modéliser un système imparfait, c'est-à-dire un système qui ne répond pas aux critères précédents portant sur les cellules de ses nœuds, nous pouvons toujours appliquer les principes présentés dans ce chapitre, en faisant l'hypothèse que certaines voies de communication, voire toutes les voies, sont unidirectionnelles. Il faut alors deux voies unidirectionnelles pour modéliser les communications entre deux nœuds, ou bien une seule voie si l'un des nœuds peut recevoir des données d'un nœud, mais ne peut pas lui en envoyer. Les principes de modélisation des systèmes imparfaits restent identiques à ceux présentés dans la suite de ce paragraphe, même si les liens de communications entre les nœuds sont plus difficile à établir.

Localités et groupes de diffusion local. Lorsqu'un nœud diffuse localement un message, tous les nœuds « voisins » qui sont situés dans sa cellule reçoivent ce message. Nous dirons que les nœuds voisins forment une *localité* du système auxquels ils appartiennent, et que les voies de communication entre les nœuds d'une même localité forment *un groupe de diffusion local* de ce même système. De façon informelle, un groupe de diffusion d'un système n'est autre qu'un sous-ensemble de voies de communication de ce système, qui sont partagées deux-à-deux par un sous-ensemble de nœuds de ce même système, de cardinalité maximale, que nous appelons une localité. Dans l'exemple précédent, A, B et C partagent deux-à-deux une voie de communication différente, ce qui est aussi le cas de A et D, mais pas de B et D, ou de C et D. Ces trois nœuds forment donc une localité, que nous appelons l_1, et les voies de communications entre ces trois nœuds forment un groupe de diffusion local, que nous appelons g_1. Dans le système $\mathcal{S}$, nous pouvons mettre en évidence trois localités, qui constituent l'ensemble $\mathcal{L}_S$:

$$\mathcal{L}_S = \{l_1, l_2, l_3\} \text{ avec } l_1 = \{A, B, C\},\ l_2 = \{A, D\}, \text{ et } l_3 = \{D, E\},$$

F n'appartient à aucune localité parce qu'il n'a pas de nœud voisin. A noter que, si $\mathcal{S}$ eût été imparfait, l'ensemble $\mathcal{L}_S$ aurait été différent. Cet ensemble est représenté à la figure 3.2.

Nous pouvons déterminer l'ensemble $\mathcal{L}_{N_i}$ des localités auxquelles appartient le nœud N_i :

$$\mathcal{L}_A = \{l_1, l_2\},\ \mathcal{L}_B = \{l_1\},\ \mathcal{L}_C = \{l_1\},\ \mathcal{L}_D = \{l_2, l_3\},\ \mathcal{L}_E = \{l_3\}, \text{ et } \mathcal{L}_F = \emptyset$$

Il existe plusieurs possibilités de représenter les voies de communication entre n nœuds d'un système. La première, peut-être la plus intuitive, est de représenter chaque voie par une

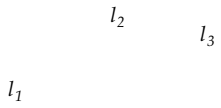


FIG. 3.2 – Modélisation de l'ensemble des localités $\mathcal{L}_S$ du système $\mathcal{S}$.

liaison entre deux nœuds. Par exemple, dans notre exemple précédent, on peut considérer que le nœud A est lié au nœud B par le voie v_{AB}, qu'il est lié au nœud C par une autre voie nommée v_{AC}, alors que les nœuds B et C sont liés par une troisième voie v_{BC}. Le système $\mathcal{S}$ étant parfait, la voie de communication v_{N_i,N_j} entre les nœuds N_i et N_j est bidirectionnelle, et elle est la même que la voie de communication v_{N_j,N_i}. Dans ce cas, nous pouvons mettre en évidence trois groupes de diffusion dans le système $\mathcal{S}$ qui forment l'ensemble $\mathcal{G}_S$:

$$\mathcal{G}_S = \{g_1, g_2, g_3\} \text{ avec } g_1 = \{v_{AB}, v_{AC}, v_{BC}\},\ g_2 = \{v_{AD}\}, \text{ et } g_3 = \{v_{DE}\},$$

où $v_{N_iN_j}$ représente la voie de communication entre les nœuds N_i et N_j. A noter que, parce que $\mathcal{S}$ est un système parfait, la voie de communication v_{N_i,N_j} est la même que la voie de communication v_{N_j,N_i}. Si ce n'avait pas été le cas, l'ensemble $\mathcal{G}_S$ serait différent. Cette approche a été suivie par A. Singh, C.R. Ramakrisnhan et S.A. Smolka dans [SRS10]. Ils considèrent que les communications se fondent sur la connectivité entre les nœuds, définie comme une relation binaire sur l'ensemble des nœuds. Ils définissent alors un port de diffusion local comme étant une clique maximale dans le graphe des connectivité des nœuds du système. Cette interprétation correspond à la figure 3.3(a). Cela revient à considérer que les nœuds A et B communiquent à l'aide d'une voie inconnue de C, que A et C communiquent à l'aide d'une voie inconnue de B, et finalement que B et C communiquent à l'aide d'une voie inconnue de A.

Mais, on peut aussi considérer que les trois nœuds partagent la même voie. Dans ce cas :

$$\mathcal{G}_S = \{g_4, g_2, g_3\} \text{ avec } g_4 = \{v_{ABC}\},\ g_2 = \{v_{AD}\}, \text{ et } g_3 = \{v_{DE}\},$$

Cette interprétation correspond à la figure 3.3(b). Dans ce cas, la connectivité des nœuds n'est plus une relation binaire, on ne peut plus parler du graphe de connectivité des nœuds, comme le font les auteurs de [SRS10], mais plutôt de relation n-aire de connectivité des nœuds d'un système et, par suite, de l'hypergraphe des connectivités des nœuds, qui évoque le même type de relation dans les bigraphes présentés au paragraphe 2.5.

Ces définitions ont plusieurs inconvénients. D'abord, comme nous venons de le montrer, elles dépendent de l'interprétation d'une voie de communication que l'on peut avoir. De plus, elles nécessitent la connaissance du graphe, ou de l'hypergraphe, de connectivité des nœuds du système. Or cette donnée n'est pas statique, mais dynamique, puisque les nœuds sont mobiles. Sa mise à jour permanente, voire en temps réel si le système est réactif, peut s'avérer fastidieuse. Pour éviter ce problème, nous préférons définir un groupe de diffusion en π_b-calcul plus loin dans ce paragraphe, comme un partage de noms libres entre des processus modélisant les nœuds.

(a) Interprétation 1 (b) Interprétation 2

FIG. 3.3 – Interprétations de la connectivité (communications) des nœuds dans le système $\mathcal{S}$ précédent.

Nous pouvons déterminer l'ensemble $\mathcal{G}_{N_i}$ des groupes de diffusion locaux détenues par le nœud N_i[1] :

$$\mathcal{G}_A = \{g_1, g_2\}, \mathcal{G}_B = \{g_1\}, \mathcal{G}_C = \{g_1\}, \mathcal{G}_D = \{g_2, g_3\}, \mathcal{G}_E = \{g_3\}, \text{ et } \mathcal{G}_F = \emptyset$$

Ces A détient les groupes g_1 et g_2. Si A diffuse un message, tous les nœuds qui partage au moins un groupe avec A recevront ce message, en l'occurrence les nœuds B et C parce qu'ils partagent g_1 avec A, et D parce qu'il partage g_2 avec A. Si D diffuse un message, les nœuds détenant les groupes g_2 et g_3, le recevront, en l'occurrence A et E. Il en est de même pour tous les nœuds d'un système : un groupe de diffusion local représente un *port* de diffusion local dans un système communicant. Le nœud F ne peut pas communiquer parce qu'il ne partage aucune voie de communication avec un autre nœud du système. Mais nous dirons qu'il ne peut diffuser localement un message, parce qu'il ne possède aucun groupe.

Interface d'un processus. Chaque nœud est caractérisé par une *interface*, que nous définissons comme l'ensemble des localités auxquelles appartient ce nœud. Un nœud N_i dont l'interface est $\{l_i\}_{i\in I}$ est représenté par $[N_i]_I$ ou, $N_{i,I}$ sans les crochets, s'il n'y a pas d'ambiguïté. Dans cette définition, I est un sous-ensemble fini de $\mathbb{N}$, puisque nous étudions les systèmes finis, donc composés d'un nombres finis de sous-systèmes dont des nœuds. Rappelons que nous avons défini au chapitre 1 un nœud, encore appelé un « composant », comme un sous-système atomique d'un système qui le contient. En fait, la notion d'interface n'est pas propre au π_b-calcul, et nous avons vu au paragraphe 2.2, qu'elle apparait dans plusieurs algèbres de processus récentes développées pour modéliser les réseaux sans fil mobiles. En particulier dans CWS de N. Mezzetti et D. Sangiorgi [MS06], où elle est fait implicitement référence à un repère de coordonnées cartésiennes, ou polaires, qui permet de donner des valeurs numériques aux localisations et aux rayons des cellules des nœuds. Un autre exemple de différence avec notre définition d'interface, celle d'un nœud de l'ω-calcul de de A. Singh, C.R. Ramakrishnan et S.A. Smolka [SRS10] est l'ensemble des cliques maximales dans le graphe des connectivité des nœuds, alors que nous avons vu que nous pouvons considérer les connectivités des nœuds comme un hypergraphe avec plusieurs interprétations possibles pour leurs voies de communication, et donc leur connectivité. La définition d'interface que nous donnons est, plus simple, parce qu'elle est plus abstraite ! Par ailleurs, la notion de nœud a l'inconvénient d'être intuitive, et elle est nécessite une formalisation pour être utilisée dans des calculs formels utilisant des algèbres de processus. Nous considérons donc qu'un nœud d'un système communicant n'est rien d'autre qu'un processus de même nom et de même interface. Ainsi, un processus du π_b-calcul est doté d'une interface qui représente l'ensemble des

[1] Un nœud *a*, ou *détient* ou encore *possède* un groupe de diffusion local, mais il n'*appartient* pas à un groupe de diffusion local. En effet, la notion de nœud fait intuitivement référence à un processus calculatoire, alors qu'un groupe de diffusion local est un ensemble de voies de communication. Ces deux notions sont sémantiquement disjointes.

localités auxquelles il appartient. Le système précédent est représenté comme la composition de processus suivante :

$$S \triangleq A_{\{l_1,l_2\}} \mid B_{\{l_1\}} \mid C_{\{l_1\}} \mid D_{\{l_2,l_3\}} \mid E_{\{l_3\}} \mid F_{\emptyset}$$

L'interface de F est vide, non pas parce qu'il ne peut pas communiquer avec un autre nœud (ce qui, somme toute, est vrai), mais parce qu'il ne peut pas diffuser ses messages à d'autres nœuds. Par ailleurs, ce n'est pas parce ce que son interface est vide qu'il ne pourra jamais communiquer avec d'autres nœuds : il peut recevoir des noms libres d'un processus qui n'a pas été représenté dans les schémas précédents, par exemple un processus spécifiant un modèle de mobilité des nœuds du système, qui lui permettront de communiquer. C'est le principe même de la mobilité des processus du π-calcul, conservée en π_b-calcul.

Diffusion locale. L'action du π_b-calcul qui permet à un processus de diffuser localement un nom libre x est notée $\bar{\beta}x$, alors que $r(y)$ est l'action permettant de recevoir un nom. Les processus récepteurs d'une diffusion locale sont ceux qui partagent une localité avec le processus émetteur, autrement dit, les processus dont l'interface n'est pas disjointe de celle de l'émetteur. Dans l'exemple précédent, si le processus A diffuse le nom libre x et que les processus des localités de son interface veulent le recevoir, le système :

$$S \triangleq [\bar{b}x.A']_{\{l_1,l_2\}} \mid [r(y).B']_{\{l_1\}} \mid [r(z).C']_{\{l_1\}} \mid [r(u).D']_{\{l_2,l_3\}} \mid [r(v).E]_{\{l_3\}} \mid [r(w).F]_{\emptyset}$$

évolue en S' :

$$S' \triangleq [A']_{\{l_1,l_2\}} \mid [B'\{x/y\}]_{\{l_1\}} \mid [C'\{x/z\}]_{\{l_1\}} \mid [D'\{x/u\}]_{\{l_2,l_3\}} \mid [r(v).E]_{\{l_3\}} \mid [r(w).F]_{\emptyset}$$

On considère qu'il s'agit d'une action silencieuse, invisible de l'extérieur du système : $S \xrightarrow{\tau} S'$. Les nœuds E et F, dont les interfaces sont disjointes de celles de A, ne reçoivent pas les noms qu'il peut diffuser.

Mobilité des processus. Comme en π-calcul, la mobilité des processus est représentée par l'échange de noms libres (*name passing*) entre les processus. Ces échanges font évoluer la topologie de communication des processus, et donc les localités auxquels ils appartiennent. Intuitivement, il existe un lien entre les noms et les localités dans le π_b-calcul. Dans l'exemple précédent, supposons que le nœud F se rapproche du nœud E, de sorte qu'ils se situent tous les deux dans l'intersection de leur cellule respective ; ils peuvent donc communiquer. Dans ce cas, ces deux nœuds partagent une nouvelle localité, distincte des localités existantes, puisque l'on suppose que F n'est pas entrer dans la cellule des autres nœuds du système. En nommant l_4 cette nouvelle localité, le système devient :

$$S'' \triangleq [A]_{\{l_1,l_2\}} \mid [B]_{\{l_1\}} \mid [C]_{\{l_1\}} \mid [D]_{\{l_2,l_3\}} \mid [E]_{\{l_3,l_4\}} \mid [F]_{\{l_4\}}$$

3.2 Syntaxe et sémantique du $(\pi_b)_0$-calcul

3.2.1 Syntaxe

La syntaxe du $(\pi_b)_0$-calcul comprend des noms, des localités, et des processus dotés d'interface. Une localité d'un système S est un ensemble fini $\mathcal{L}_S$ de noms libres (correspondants à des voies privées ou publiques) partagé deux-à-deux par un sous-ensemble $\mathcal{E}_S$, de cardinalité maximale, de processus de ce système noté $\mathcal{P}_S$; ce système étant supposé être clos.

On note $\mathcal{L}$ l'ensemble des localités du π_b-calcul ; cet ensemble est un sous-ensemble des noms $\mathcal{N}$ du π-calcul. Le système S est considéré comme clos, pour éviter qu'un système extérieur à S utilise une localité comme voie de communication, et ainsi interférer dans les

échanges de S.

N.B. L'interface du processus inactif est l'ensemble vide. C'est aussi le cas d'un processus qui ne partage aucun nom libre avec les autres processus du système auquel il appartient. La différence entre les deux est que l'interface du processus inactif est figée et ne peut pas changer, ce qui n'est pas le cas du deuxième processus dont la mobilité influence l'état de son interface.

Définition 3.1. La syntaxe du $(\pi_b)_0$-calcul est définie de la façon suivante :

$$P ::= 0 \mid \sum_{i\in I} \pi_i.[P_i]_{I_i} \mid [P_1]_{I_1}|[P_2]_{I_2} \mid [x = y][P]_I \mid [A(\vec{x})]_I$$

où I est un sous-ensemble fini de $\mathbb{N}$ et les π_i sont les actions préfixées suivantes :

$$\pi_i ::= \bar{\beta}x \mid r(x) \mid \tau$$

L'action $\bar{\beta}x$ représente la diffusion locale du nom x, alors que $r(x)$ représente la réception du nom x qui a été diffusé localement. Les actions internes sont dénotées par τ.

Noms libres et noms liés. Pour un processus $[P]_I$ d'interface I, l'ensemble de ses noms libres et l'ensemble de ses noms liés, dénotés respectivement par $fn(P)$ et $bn(P)$ sont définis de la façon suivante :

$fn(0)$	$=$	$\emptyset$		$bn(0)$	$=$	$\emptyset$
$fn(\bar{\beta}x.P)$	$=$	$fn(P) \cup \{x\}$		$bn(\bar{\beta}x.P)$	$=$	$bn(P)$
$fn(r(x).P)$	$=$	$fn(P) \setminus \{x\}$		$bn(r(x).P)$	$=$	$bn(P) \cup \{x\}$
$fn(\tau.P)$	$=$	$fn(P)$		$bn(\tau.P)$	$=$	$bn(P)$
$fn(P + Q)$	$=$	$fn(P) \cup fn(Q)$		$bn(P + Q)$	$=$	$bn(P) \cup bn(Q)$
$fn([x = y]P)$	$=$	$fn(P) \cup \{x, y\}$		$bn([x = y]P)$	$=$	$bn(P)$
$fn(A(x_1, \ldots, x_n))$	$=$	$\{x_1, \ldots, x_n\}$		$bn(A(x_1, \ldots, x_n))$	$=$	$\emptyset$

TAB. 3.2 – Noms libres et noms liés du $(\pi_b)_0$-calcul.

L'ensemble de tous les noms libres et liés de P est dénoté par $n(P)$ avec $n(P) = fn(P) \cup bn(P)$. Rappelons qu'un système qui n'a pas de noms libres est appelé un système *clos* (ou encore un système *fermé*), par opposition à un système *ouvert* qui possède au moins un nom libre, lui permettant de communiquer avec son environnement.

3.2.2 Sémantique

$[P]_I \equiv [Q]_I$ *si* $P \equiv_\alpha Q$	(Struct Alpha)
$[P]_{I_1} + [Q]_{I_2} \equiv [Q]_{I_2} + [P]_{I_1}$	(Struct Choice Commut)
$([P]_{I_1} + [Q]_{I_2}) + [R]_{I_3} \equiv [P]_{I_1} + ([Q]_{I_2} + [R]_{I_3})$	(Struct Choice Assoc)
$[P]_I \mid 0 \equiv [P]_I$	(Struct Par Id)
$[P]_{I_1} \mid [Q]_{I_2} \equiv [Q]_{I_2} \mid [P]_{I_1}$	(Struct Par Commut)
$([P]_{I_1} \mid [Q]_{I_2}) \mid [R]_{I_3} \equiv [P]_{I_1} \mid ([Q]_{I_2} \mid [R]_{I_3})$	(Struct Par Assoc)

TAB. 3.3 – Règles définissant la congruence structurelle entre processus du $(\pi_b)_0$-calcul.

La sémantique du $(\pi_b)_0$-calcul est définie par une relation de congruence structurelle $\equiv$, et par une relation de transition $\rightarrow$.

Les règles de transitions entre les processus du $(\pi_b)_0$-calcul sont définies par le tableau 3.4. Les règles MCAST et RECV définissent, avec la règle COM, une communication par diffusion locale. La règle RECV stipule qu'un processus P d'interface I peut recevoir une diffusion locale sur n'importe quelle de ses localités appartenant à I. Cette règle avec la règle COM signifie qu'un émetteur qui diffuse localement un nom, peut se synchroniser avec n'importe quel récepteur local avec qui il partage une localité. Un processus dont l'interface est vide ne peut ni envoyer, ni recevoir de nom. La règle PAR indique le sémantique d'entrelacement pour les actions des processus en parallèle. Sa condition est utilisée pour éviter la capture de noms. La sémantique de la diffusion locale du $(\pi_b)_0$-calcul permet à un récepteur d'ignorer une diffusion de nom, même si le nœud correspondant se trouve dans la cellule de l'émetteur. Ce qui permet de capturer les transmissions perdus dans les systèmes sans fil.

$$\text{MCAST} : \frac{-}{[\bar{b}y.P]_I \xrightarrow{\bar{b}y} [P]_{I'}} \quad I \neq \emptyset$$

$$\text{RECV} : \frac{-}{[r(x).P]_I \xrightarrow{r(w)} [P\{w/x\}]_{I'}} \quad w \notin fn((x)P),\ I \neq \emptyset$$

$$\text{PAR} : \frac{[P]_I \xrightarrow{\alpha} [P']_{I'}}{[P]_I \mid [Q]_J \xrightarrow{\alpha} [P']_{I'} \mid [Q]_J} \quad bn(\alpha) \cap fn(Q) = \emptyset$$

$$\text{COM} : \frac{[P]_I \xrightarrow{\bar{b}y} [P']_{I'} \quad [Q]_J \xrightarrow{r(x)} [Q']_{J'}}{[P]_I \mid [Q]_J \xrightarrow{\bar{b}y} [P']_I \mid [Q'\{y/x\}]_{I'}} \quad I \cap I' \neq \emptyset$$

$$\text{CHOICE} : \frac{[P]_I \xrightarrow{\alpha} [P']_{I'}}{[P]_I + [Q]_J \xrightarrow{\alpha} [P']_{I'}}$$

$$\text{MATCH} : \frac{[P]_I \xrightarrow{\alpha} [P']_I}{[x = x][P]_{I'} \xrightarrow{\alpha} [P']_{I'}}$$

$$\text{DEF} : \frac{[P\{\vec{y}/\vec{x}\}]_I \xrightarrow{\alpha} [P']_{I'}}{[A(\vec{y})]_I \xrightarrow{\alpha} [P']_{I'}} \quad [A(\vec{y})]_I \stackrel{def}{=} [P]_I$$

TAB. 3.4 – Règles d'inférence définissant les transitions entre processus du $(\pi_b)_0$-calcul.

3.3 L'algèbre complète : le π_b-calcul

Le π_b-calcul se distingue du $(\pi_b)_0$-calcul par l'ajout de la communication point-à-point entre les composants du système. C'est la primitive de communication de base du π-calcul. Nous utilisons la notation habituelle : $\bar{x}y$ dénote l'envoi du nom libre y via le nom libre x, alors que $x(z)$ représente la réception d'un nom libre via le nom libre x, z étant lié. Dans le cas du π_b-calcul, nous posons l'action r qui permet aux processus de recevoir un nom diffusé dans une localité, est équivalente aux réceptions de ce nom sur chacune des voies définissant cette localité :

$$[r(x).P]_I \equiv a_1(x) \mid a_2(x) \mid \ \ldots \ \mid a_n(x) \text{ avec } I = \{a_1, a_2, \ldots, a_n\}$$

L'opérateur de réplication ! du π-calcul n'a pas été conservé. La raison est que nous nous intéressons au model checking fini, et que nous voulons modéliser des systèmes finis pour analyser leur performabilité par les techniques de model checking. Or l'opérateur de réplication peut induire des systèmes infinis.

Définition 3.2. La syntaxe du π_b-calcul est définie par :

$$P ::= 0 \mid \sum_{i\in I} \pi_i.[P_i]_{I_i} \mid [P_1]_{I_1}|[P_2]_{I_2} \mid (\nu x)[P]_I \mid [x = y][P]_I \mid [A(\vec{x})]_I$$

où I est un sous-ensemble fini de $\mathbb{N}$ et les π_i sont les actions préfixées suivantes :

$$\pi_i ::= \bar{x}y \mid x(y) \mid \bar{\beta}x \mid r(x) \mid \tau$$

Nous ajoutons aux règles de congruence structurelles du tableau 3.3, les règles définies par la tableau 3.5.

$(\nu x)[P]_I \equiv [P]_I,\ si\ x \notin fn(P)$	(Struct Ex 1)
$(\nu x)(\nu y)[P]_I \equiv (\nu y)(\nu x)[P]_I$	(Struct Swap)
$[P]_I \mid [Q]_J \equiv [Q]_J \mid [P]_I$	(Struct Par Commut)
$(\nu x)[P]_I \mid [Q]_J \equiv (\nu x)([P]_I \mid [Q]_J),\ si\ x \notin fn(Q)$	(Struct Ex 2)

Tab. 3.5 – Règles additionnelles congruence additionnelles pour le π_b-calcul.

Les règles additionnelles de sémantique opérationnelle propres au π_b-calcul sont représentées par le tableau 3.6. Les règles définies au tableau 3.4 sont toujours applicables.

$$\text{PROC-PAR} : \frac{[P]_I \xrightarrow{\alpha} [P']_{I'}}{([P]_I|[Q]_J \xrightarrow{\alpha} [P']_{I'}|[Q]_J}\ bn(\alpha) \cap fn(Q) = \emptyset$$

$$\text{PROC-COM} : \frac{[P]_I \xrightarrow{z:Ix} [P']_I \quad [Q]_I \xrightarrow{z:I(y)} [Q']_I}{([P]_I|[Q]_I \xrightarrow{\tau} [P']_I|[Q'\{x/y\}]_I}$$

$$\text{PROC-CLOSE} : \frac{[P]_I \xrightarrow{(\nu x)z:Ix} [P']_I \quad [Q]_I \xrightarrow{z:I(x)} [Q']_I}{([P]_I|[Q]_I \xrightarrow{\tau} ((\nu x)([P']_I|[Q']_I))}$$

Tab. 3.6 – Règles additionnelles de sémantique opérationnelle pour le π_b-calcul

3.4 Extension probabiliste du π_b-calcul

Nous voulons étendre le π_b-calcul de façon probabiliste à la manière de Catuscia Palamidessi et al. qui, dans [NPPW09], étendent le π-calcul en considérant la sommation probabiliste de deux processus : le processus $P = p_1.\tau.Q_1 \oplus p_2.\tau.Q_2$ se comporte comme Q_1 avec la probabilité p_1, et comme Q_2 avec la probabilité Q_2. Cet algèbre est appelée le π-calcul probabiliste (π_{prob}). Par suite, ils proposent des règles de traduction des modèles π_{prob} en processus de décision markoviens spécifiés en langage PRISM pour les analyser par model checking. Notre

approche est la même. Elle nous permettra de modéliser des diffusions locales et de traduire les modèles correspondants en processus de décision markoviens exprimés en langage PRISM.

Sémantique. Nous ajoutons les deux règles suivantes à la sémantique symbolique[2] du π_{prob} définie dans [NPPW09] :

$$\text{OUTPUT} : \frac{-}{\bar{\beta}x.P \xrightarrow{true,\ \bar{\beta}x} \{|1 : P|\}}$$

$$\text{INPUT} : \frac{-}{r(x).P \xrightarrow{true,\ r(w)} \{|1 : P\{w/x\}|\}}, w \notin fn((\nu z)P)$$

TAB. 3.7 – Règles de sémantique symbolique additionnelles pour le π_b-calcul probabiliste.

La règles OUTPUT exprime le fait que le processus $\bar{\beta}x.P$ diffuse le nom x localement, avant de se comporter comme P avec la probabilité 1. La règle INPUT exprime le fait que le processus $r(x).P$ reçoit un nom diffusé localement, avant de se comporter comme le processus P avec la probabilité 1, où toutes les occurrences libres de x auront été remplacées par le nom libre effectivement reçu. La sémantique du π_b-calcul probabiliste est établie par les règles précédentes, et celle de la figure 1 de [NPPW09].

Traduction en langage PRISM. Nous voulons traduire les modèles π_b en processus de décision markoviens spécifiés en langage PRISM. Pour cela, on considère qu'une diffusion locale correspondant à un ensemble de communications en point-à-point sur chacune des voies définissant la localité. De la même façon, la réception des noms diffusés localement correspond à la réception de ce nom via chaque voie constituant la localité. Cette manière de faire permet d'assurer la synchronisation entre diffusion locale et réception. Les règles de traduction des communications en point-à-point sont les mêmes que celles exposées dans [NPPW09].

3.5 Extension stochastique du π_b-calcul

Nous pouvons également étendre le π_b en considérant des taux associés aux actions silencieuses et aux actions de diffusion ou d'émission en point à point. Nous adoptons l'approche de Catuscia Palamidessi et al. qui, dans [NPPW09], ont proposé de traduire les modèles en π-calcul stochastique en chaîne de Markov à temps continu exprimées en langages PRSM.

Sémantique. Nous ajoutons les deux règles suivantes à la sémantique symbolique du π_{stoc} définie dans [NPPW09] :

Ces règles expriment la diffusion locale et la réception d'un nom diffusé localement. A chaque diffusion est associé un *taux* qui correspond au taux de communication en point-à-point *le plus faible* de l'ensemble des voies définissant la localité. Le taux de réception est, quant-à-lui de 1 [NPPW09]. La sémantique du π_b-calcul stochastique est établie par les règles précédentes, et celle de la figure 2 de [NPPW09].

[2]La sémantique symbolique d'une algèbre de processus se distingue de sa sémantique opérationnelle concrète dont nous avaons parlé jusqu'ici, par son abstraction : elle considère l'environnement dans lequel est exécutée une action d'un processus, ou des opérations de processus (cf. paragraphe 4.3.2.

$$\text{OUTPUT} : \frac{-}{\bar{\beta}x.P \xrightarrow{true,\ \bar{\beta}x} P}$$

$$\text{INPUT} : \frac{-}{r(x).P \xrightarrow{true,\ r(w)} P\{w/x\}}, w \notin fn((\nu z)P)$$

TAB. 3.8 – Règles de sémantique symbolique additionnelles pour le π_b-calcul stochastique.

Traduction en langage PRISM. Nous traduisons les modèles en π_b-calcul stochastique en chaînes de Markov à temps continu. Pour cela, nous considérons les mêmes hypothèses que précédemment, qui nous permettent de décomposer une diffusion locale ou une réception d'une diffusion locale, en autant de communications point-à-point qu'il y a de voies de communications dans la localité. Par ailleurs nous supposons que le taux de diffusion local est le plus faible taux de communication sur l'ensemble de ces voies. Là encore ces hypothèses nous permettent de conserver le synchronisme local. Les règles de traduction des communications en point-à-point sont les mêmes que celles exposées dans [NPPW09].

4 Une extension temporisée du π-calcul :le π_t-*calcul*

4.1 Préliminaire : rappels sur la théorie des automates temporisés

Ce paragraphe préliminaire est inspiré du chapitre *Model checking temporisé* rédigé par Béatrice Bérard, de l'ouvrage collectif *Approches formelles des systèmes embarqués communicants* [Bé08].

Les *automates temporisés* ont été introduits par Rajeev Alur et David L. Dill dans les années 1990 pour modéliser les systèmes temps-réels [AD90, AD94]. De façon informelle, un automate temporisé est un automate fini utilisant un ensemble d'*horloges* qui sont des variables définies sur le *domaine du temps*, noté $\mathbb{T}$, évoluant de manière synchrone et continue dans le temps absolu. Le domaine du temps peut être l'ensemble des entiers naturels $\mathbb{N}$, ou l'ensemble des nombres rationnels positifs $\mathbb{Q}_+$, ou, plus habituellement, l'ensemble des nombres réels positifs $\mathbb{R}_+$. Dans chaque état, le temps peut s'écouler et la valeur d'une horloge est le temps écoulé depuis la dernière initialisation à 0 de cette horloge. Chaque transition de l'automate est instantanée, mais elle comporte une contrainte sur les horloges de l'automate, appelée une *garde*, qui indique quand celle-ci peut être exécutée. Les transitions peuvent réinitialiser une ou plusieurs d'horloges lorsqu'elles sont exécutées. De la même façon, chaque état de l'automate comporte une contrainte sur les horloges, appelée un *invariant*[3], qui restreint la durée de stationnement dans cet état, et donc peut forcer l'exécution d'une transition.

4.1.1 Notations

Soit X un ensemble fini de variables, appelées horloges, prenant leur valeurs dans $\mathbb{T}$. Une valuation d'horloge v sur X est une fonction $v : X \to \mathbb{T}$ qui associe à chaque horloge x de X, sa valeur $v(x)$ dans $\mathbb{T}$. L'ensemble des valuations d'horloge sur X est noté $\mathbb{T}^X$. Soit $d \in \mathbb{T}$, $v + d$ est la valuation d'horloge qui associe à l'horloge x la valeur $v(x) + d$. Si r est un sous-ensemble de X, $v[r \leftarrow 0]$ représente la valuation v' telle que $v'(x) = 0$ si $x \in r$, et

[3]La notion d'*invariant* a été introduite par Thomas A. Henzinger et al. dans [HNSY94].

$v'(x) = v(x)$ sinon.

$\mathcal{C}(X)$ dénote l'ensemble des contraintes d'horloge sur X, c'est-à-dire l'ensemble des combinaisons booléennes des contraintes atomiques de la forme $x \bowtie c$, où x est une horloge de X, c est un entier naturel, et $\bowtie$ est un opérateur de comparaison appartenant à l'ensemble $\{<, \leqslant, =, \geqslant, >\}$. On note $\mathcal{C}_{\{<,\leqslant\}}(X)$, la restriction de $\mathcal{C}(X)$ aux combinaisons booléennes faisant intervenir uniquement les opérateurs $<$ et $\leqslant$. Les contraintes d'horloges sont interprétées de la façon suivante : une valuation v satisfait la contrainte atomique $x \bowtie c$, chaque fois que $v(x) \bowtie c$. On note $v \models g$ pour signifier que la valuation v satisfait la contrainte g.

4.1.2 Définition et exemple

Définition 3.3. Un automate temporisé $\mathcal{A}$, sur un alphabet A et sur un ensemble de propositions atomiques PA, est un sextuple $\mathcal{A} = (L, l_0, X, Inv, T, e)$, où L est un ensemble d'états (également appelés des « modes de contrôle » ou des « localités »), $l_0 \in L$ est l'état initial, X est un ensemble fini d'horloges, $Inv : L \to C_{\{<,\leqslant\}}(X)$ associe une contrainte d'horloges appelée un *invariant* à chaque état, $T \subseteq L \times C(X) \times A \times 2^X \times L$ est un ensemble fini de transitions, et $e : L \to 2^{PA}$ est une fonction d'étiquetage des états par des propositions atomiques issues de PA.

Une transition $t = (l, g, a, r, l') \in T$, notée $l \xrightarrow{g,a,r} l'$, représente le passage de l'état l à l'état l' par l'action a, si la garde g est vérifiée, r étant l'ensemble des horloges qui sont réinitialisées à 0 lors de ce passage. On fait les hypothèses fondamentales suivantes :

1. Les automates temporisés sont *non Zénon*[4], c'est-à-dire il n'est pas possible d'effectuer un nombre infini de transitions entre états d'un automate, en un temps borné qui converge vers une limite finie.

2. Le temps peut toujours progresser. Ce qui signifie que, pour chaque état, soit l'invariant est toujours vrai, soit il existe au moins une transition vers un autre état dont la garde lui permet d'être exécutée avant que l'invariant de l'état courant devienne faux. Dans le cas contraire, on parle de « blocages temporels » ou *timelocks*. L'outil PRISM présenté à l'annexe B, détecte les blocages temporels, mais sa version courante ne permet pas de détecter des automates temporisés Zénon.

Exemple. La figure 3.4 modélise un système composé d'un brûleur à gaz qui fuit. Le système peut être dans l'état initial l_0 où le gaz fuit, et dans l'état l_1 où il ne fuit pas. Les fuites de gaz sont détectées et stoppées en moins d'une seconde. Cependant, une fois une fuite stoppée, le brûleur peut recommencer à fuir au bout 30 secondes. L'horloge x mesure les délais en secondes. Elle est réinitialisée à chaque transition entre les états. Le système peut rester dans l'état l_1 indéfiniment, s'il n'y a pas de fuite. Il n'y a donc pas de contrainte de durée maximale de stationnement du système dans l'état l_1 : son invariant est toujours vrai, noté *true* dans la figure.

[4] D'après le philosophe grec de l'Antiquité, *Zénon d'Élée* (vers 480 av. J.-C - vers 420 av. J.-C), qui imagina un ensemble de huit paradoxes pour soutenir, par l'absurde, l'hypothèse de la continuité du mouvement, et s'ériger contre l'École de Pythagore qui affirmait sa divisibilité. Par exemple, le *paradoxe de la dichotomie* prétend que le caillou lancé par Zénon ne pourra jamais atteindre l'arbre situé à quelques mètres devant lui, parce que la distance qui reste à parcourir par le caillou sur son trajet peut être divisée de moitié une infinité de fois, les segments ainsi obtenus étant parcourus en un temps non nul.

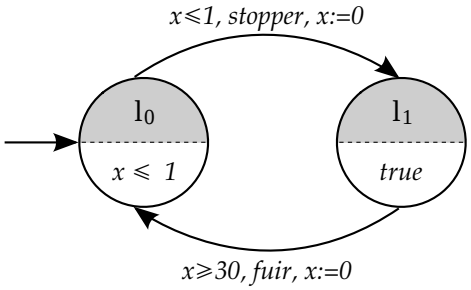


FIG. 3.4 – Automate temporisé représentant un brûleur de gaz qui fuit (d'après un exemple de [Bé08], lui-même tiré de [ACHH92]).

4.1.3 Sémantique d'un automate temporisé.

La sémantique d'un automate temporisé est donnée par un *système de transitions temporisé* qui comporte deux types de transitions : des transitions d'actions, et des transitions de temps.

Définition 3.4. Un système de transitions temporisé, sur un alphabet A et sur un ensemble PA de propositions atomiques, est un quadruplet $\mathcal{T} = (S, s_0, R, E)$ où : S est un ensemble d'états appelés des *configurations*, s_0 est la configuration initiale, $R \subseteq S \times (Act \cup \mathbb{T}) \times S$ est la relation de transition, et $E : S \to 2^{PA}$ est la fonction d'étiquetage des configurations par des éléments de PA.

Une transition (s, l, s') de R est notée $s \xrightarrow{l} s'$. Les transitions $\xrightarrow{a}$ avec $a \in A$, correspondent à des actions considérées comme instantanées. Les transitions $\xrightarrow{d}$ avec $d \in \mathbb{T}$ expriment l'écoulement d'une durée d et doivent vérifier les conditions suivantes :

- délai nul : $s \xrightarrow{0} s'$ si et seulement si $s = s'$;
- additivité : si $s \xrightarrow{d} s'$ et $s' \xrightarrow{d'} s''$ alors $s \xrightarrow{d+d'} s''$;
- déterministe par rapport au temps : si $s \xrightarrow{d} s'$ et $s \xrightarrow{d} s''$ alors $s' = s''$;
- continuité : $s \xrightarrow{d} s'$ implique que, pour tout d' et d'' tels que $d = d' + d''$, il existe s'' tel que $s \xrightarrow{d'} s''$ et $s'' \xrightarrow{d''} s'$.

Une exécution de $\mathcal{T}$ est un chemin infini $\rho = s_0 \xrightarrow{d1} s'_0 \xrightarrow{a1} s_1 \xrightarrow{d2} s'_1 \xrightarrow{a2} s_2 \ldots$ partant de la configuration initiale et où les durées alternent strictement avec les actions. Un début d'exécution possible du brûleur de gaz illustré par la figure 3.4 pourrait être :

$$(e_0, 0) \xrightarrow{0,6} (e_0, 0, 6) \xrightarrow{stopper} (e_1, 0) \xrightarrow{33} (e_1, 33) \xrightarrow{fuir} (e_0, 0) \ldots$$

La sémantique d'un automate temporisé $\mathcal{A} = (L, l_0, X, Inv, T, e)$ défini sur A et PA, est donnée par le système de transitions temporisé $\mathcal{S}_\mathcal{A} = (S, s_0, R, E)$ sur $A \cup \mathbb{T}$ et PA avec :

- $S = \{(l, v) \in Q \times \mathbb{T}^X \mid v \models Inv(l)\}$. Les configurations sont des couples (l, v) où l est un état de L, et v est une valuation d'horloges satisfaisant l'invariant de l'état l.
- La configuration initiale est $s_0 = (l_0, 0)$.
- Les transitions de E sont :
  - soit $(l, v) \xrightarrow{d} (l, v + d)$: une transition de durée $d \in \mathcal{T}$, possible si $v + d \models Inv(l)$
  - soit $(l, v) \xrightarrow{a} (l', v')$: une transition d'étiquette $a \in A$, avec $v' = v[r \leftarrow 0]$, possible s'il existe $l \xrightarrow{g,a,r} l'$ dans T telle que la valuation v satisfasse la garde g.

– La fonction d'étiquetage des configurations L est définie pour toute les configurations $(l, v) \in S$ par $E(l, v) = e(l)$.

Les transitions de durée correspondent au temps passé dans un état. Le système part de la configuration initiale correspondant à l'état initial l_0 et toutes les horloges initialisées à zéro, puis effectue deux types de transitions : les transitions de temps qui augmentent toutes les horloges d'une même durée (les horloges sont synchrones) en respectant l'invariant associé à l'état courant, et les transitions d'action si la valeur courante des horloges le permet (ce type de transition s'effectue de manière instantanée et certaines horloges peuvent être remises à zéro).

4.1.4 Composition parallèle

Il est possible de modéliser un système à partir des modèles de ses sous-systèmes. Chacun de ces sous-systèmes est alors modélisé par un automate temporisé $\mathcal{A}_i$, et le modèle du système global correspond à la composition parallèle *synchrone* de ces différents automates, après leur éventuel renommage pour éviter des confusions.

4.2 Syntaxe

Nous cherchons à définir une algèbre de processus permettant de décrire les comportements temps réel des systèmes mobiles, soit le π_t-calcul. Sa sémantique est décrite en terme d'automates temporisés. A chaque processus est associé un ensemble d'horloges définies sur le domaine temporel $\mathbb{T}$. Les ensembles d'horloges de deux processus P et Q tels que $P \neq \alpha.Q$ sont deux à deux disjoints. Toutes les horloges sont synchrones. Les horloges d'un processus sont des noms privés à ce processus. Chaque action réalisée par un processus comporte une contrainte sur les horloges de ce processus. Par ailleurs, chaque action a un état « avant » et un état « après », correspondant respectivement aux états du processus avant et après la réalisation de cette action. Aux états avant et après peuvent être associés des invariants. Les *invariants avants* forcent la réalisation d'une action s'ils sont différents de *true*. Enfin, la réalisation d'une action peut réinitialiser un sous-ensemble de l'ensemble des horloges du processus.

Soient l'ensemble des noms $\mathcal{N}$, l'ensemble des horloges $\mathcal{C}$, dont les éléments sont des noms ne pouvant pas être utilisés comme voie de communication, et l'ensemble des processus $\mathcal{P}$. Les horloges sont des noms pouvant prendre des valeurs numériques définies sur le domaine du temps $\mathbb{T}$. Cet ensemble peut être l'ensemble des entiers naturels $\mathbb{N}$, ou l'ensemble des nombres rationnels positifs $\mathbb{Q}_+$, ou encore l'ensemble des nombres réels positifs $\mathbb{R}_+$.

Définition 3.5. L'ensemble $\mathcal{P}^{\pi_t}$ des expressions des processus du π-calcul temporisé (π_t-calcul) est défini par :

$$P ::= 0 \mid \alpha_{c,r}.P \mid P_1 + P_2 \mid P_1|P_2 \mid (\nu x)P \mid [x = y]P \mid A(y_1, \ldots, y_n).$$

où :

– Les actions $\alpha_{c,R}$ spécifient des contraintes temporelles c portant sur les horloges du processus qui réalise cette action, et peuvent réinitialiser un sous-ensemble d'horloges r de ce processus. Les contraintes temporelles d'une action sont sa garde, et ses invariants avant et après. La garde spécifie à quelle date, ou à partir de quelle date, ou encore, après quelle date cette action peut avoir lieu. Par exemple, en considérant une horloge t du processus qui réalise une action, celle-ci peut se passer à une date $d \in \mathbb{T}$ $(t = d)$, au

plus tard à cette date ($t \leqslant d$), avant cette date ($t < d$), au plus tôt à cette date ($t \geqslant d$), ou après cette date ($t > d$). Par ailleurs, avant et après la réalisation d'une action, un processus est dans un certain état dans lequel il doit respecter un invariant, c'est-à-dire une contrainte temporelle portant sur ses horloges. L'action ne peut se produire que si les invariants sont respectés, en particulier l'invariant avant peut forcer l'exécution d'une action avant de devenir faux. Dans la suite, l'invariant avant la réalisation d'une action est noté i, alors que l'invariant après sa réalisation est noté i'. Les actions du π_t-calcul sont donc : $\tau_{\{i,g,r,i'\}}$, $x(y)_{\{i,g,r,i'\}}$, $\bar{x}y_{\{i,g,r,i'\}}$, et $\bar{x}(y)_{\{i,g,r,i'\}}$.

- La garde g, et les invariants avant, i et après, i', d'une action, sont des booléens dont la conjonction ne doit pas être fausse après la réinitialisation des horloges de r ($i \wedge g \wedge v[r \leftarrow 0] \wedge i' \neq false$). Le cas contraire correspond à une situation où un processus doit réaliser une action imposée par l'invariant avant, mais celle-ci ne peut pas être réalisée à cause de sa garde, ou ne peut pas être réalisée parce que l'invariant après serait faux. C'est une situation de « blocage temporel » (*timelock*).
- On ne considère que des processus finis ne réalisant qu'une suite finie d'actions. En particulier, la réplication d'un processus P, notée $!P$, présentée au paragraphe 2.2.2.1, n'est pas définie. La suite d'actions réalisée par un processus, qui n'est pas infinie mais qui peut être très grande, ne doit pas générer une convergence temporelle. En quelque sorte, les processus du π_t-calcul ne sont pas des processus Zénon (voir paragraphe 3.4.1).

4.3 Sémantique opérationnelle du π_t-calcul

4.3.1 Sémantique concrète

La sémantique concrète du π_t-calcul est définie par les règles de transitions définies au tableau 3.9, où $I = \{i, g, r, i'\}$. Ces règles de transitions sont *concrètes*, dans le sens où elles peuvent être exécutées quel que soit le contexte dans lequel sont placés les termes.

4.3.2 Sémantique symbolique

La sémantique *symbolique* est composée de règles de transitions plus abstraites que les règles précédentes, dans le sens où elles considèrent le contexte d'exécutions des termes. Une transition symbolique est de la forme $P \xrightarrow{M,\alpha} Q$ où M est une conjonction de conditions booléennes de correspondances entre noms, et α est une action. M représente les environnements dans lesquels P peut exécuter α pour évoluer en Q. Par ailleurs, l'approche symbolique permet de définir une sémantique de systèmes à l'aide de variables non instanciées x, qui peuvent être utilisées pour définir des actions telles que, par exemple, $\bar{x}b$. Cette particularité permet de définir la sémantique d'un système à partir de la composition parallèle des sémantiques de ses sous-systèmes considérées séparément[5].

La sémantique symbolique du π_t-calcul est définie par le tableau 3.10. Elle s'exprime en termes de *graphes de transitions symboliques temporisés* (*TSTGs* pour *timed symbolic transition graphs*). Ce sont des extensions temporisées des *graphes de transitions symboliques* définis par M. Hennessy et H. Lin [HL95], et utilisés pour le π-calcul dans [BN96,IL01,Lin94,Lin03]. Les TSTGs peuvent être vus comme des extensions symboliques des automates temporisés.

[5]Nous utilisons la même approche que Catuscia Palamidessi et al., qui, dans [NPPW09], définissent les sémantiques symboliques d'une extension probabiliste (π_{prob}) et d'une extension stochastique (π_{stoc}) du π-calcul. Ces extensions ont été présentées respectivement aux paragraphes 2.3.3 et 2.3.4.

$$\text{TAU} : \frac{-}{\tau_{\{i,g,r,i'\}}.P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{\tau_{\{i,g,r,i'\}}} P}$$

$$\text{OUTPUT} : \frac{-}{\bar{x}y_{\{i,g,r,i'\}}.P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{\bar{x}y_{\{i,g,r,i'\}}} P}$$

$$\text{INPUT} : \frac{-}{x(z)_{\{i,g,r,i'\}}.P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{x(w)_{\{i,g,r,i'\}}} P\{w/z\}}, w \notin fn((\nu z)P)$$

$$\text{SUM} : \frac{P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{\alpha_{\{i,g,r,i'\}}} P'}{P + Q \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{\alpha_{\{i,g,r,i'\}}} P'} \qquad \text{MATCH} : \frac{P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{\alpha_{\{i,g,r,i'\}}} P'}{[x = x]P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{\alpha_{\{i,g,r,i'\}}} P'}$$

$$\text{IDE} : \frac{P\{\vec{y}/\vec{x}\} \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{\alpha_{\{i,g,r,i'\}}} P'}{A(\vec{y}) \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{\alpha_{\{i,g,r,i'\}}} P'}, A(\vec{x}) \stackrel{def}{=} P$$

$$\text{PAR} : \frac{P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{\alpha_{\{i,g,r,i'\}}} P'}{P|Q \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{\alpha_{\{i,g,r,i'\}}} P'|Q}, bn(\alpha) \cap fn(Q) = \emptyset$$

$$\text{COM} : \frac{P \xrightarrow[i_1 \wedge g_1 \wedge v[r_1 \leftarrow 0] \wedge i'_1]{x(z)_{\{i_1,g_1,r_1,i'_1\}}} P' \quad Q \xrightarrow[i_2 \wedge g_2 \wedge v[r_2 \leftarrow 0] \wedge i'_2]{\bar{x}v_{\{i_2,g_2,r_2,i'_2\}}} Q'}{P|Q \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{\tau_{\{i,g,r,i'\}}} P'\{v/z\}|Q'},$$
avec $i = i_1 \wedge i_2$, $g = g_1 \wedge g_2$, $r = r_1 \cup r_2$, $i' = i'_1 \wedge i'_2$

$$\text{CLOSE} : \frac{P \xrightarrow[i_1 \wedge g_1 \wedge v[r_1 \leftarrow 0] \wedge i'_1]{x(z)_{\{i_1,g_1,r_1,i'_1\}}} P' \quad Q \xrightarrow[i_2 \wedge g_2 \wedge v[r_2 \leftarrow 0] \wedge i'_2]{\bar{x}(v)_{\{i_2,g_2,r_2,i'_2\}}} Q'}{P|Q \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{\tau_{\{i,g,r,i'\}}} (\nu v)(P'\{v/z\}|Q')},$$
avec $i = i_1 \wedge i_2$, $g = g_1 \wedge g_2$, $r = r_1 \cup r_2$, $i' = i'_1 \wedge i'_2$

$$\text{RES} : \frac{P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{\alpha_{\{i,g,r,i'\}}} P'}{(\nu x)P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{\alpha_{\{i,g,r,i'\}}} (\nu x)P'}, \quad x \notin n(\alpha)$$

$$\text{OPEN} : \frac{P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{\bar{x}y_{\{i,g,r,i'\}}} P'}{(\nu y)P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{\bar{x}(w)_{\{i,g,r,i'\}}} P'\{w/y\}}, \quad x \neq y,\ w \notin fn((\nu y)P')$$

TAB. 3.9 – Sémantique concrète du π-calcul temporisé (π_t-calcul).

Définition 3.6. Soit P un processus du π_t-calcul. Le *graphe de transitions symbolique temporisé* (TSTG), qui définit la sémantique de P, est un triplet $(S, s_0, \mathcal{T}_t)$, où S est un ensemble d'états symboliques, chacun d'eux est un terme du π_t-calcul ; $s_0 \in S$, l'état initial, est le terme P ; et $\mathcal{T} \subseteq S \times Cond \times Act \times S$ est la plus petite relation de transition symbolique temporisée définie par les règles du tableau 3.10. Dans cette définition, $Cond$ est l'ensemble de toutes les conditions (conjonctions finies de correspondances) sur $\mathcal{N}$, Act est l'ensemble des quatre actions du π_t-calcul (τ, $x(y)$, $\bar{x}y$, et $\bar{x}(y)$ où $x, y \in \mathcal{N}$).

La transition symbolique temporisée $(Q, M, \alpha, Q') \in \mathcal{T}_t$ est notée $Q \xrightarrow{M,\alpha} Q'$. Une transition symbolique $Q \xrightarrow{M,\alpha} Q'$ d'un TSTG représente le fait que, quelle que soit la substitution σ satisfaisant M, le processus $Q\sigma$ peut réaliser l'action $\alpha\sigma$, puis évoluer en $Q'\sigma$.

4.4 Traduction des TSTGs en PRISM

4.4.1 Principes

Pour les systèmes clos et fini, composés de sous-processus parallèles, nous adoptons une traduction composée en utilisant le langage de l'outil PRISM. Plus précisément, nous considérons les systèmes de la forme $P = (\nu x_1) \dots (\nu x_n)(P_1 | \dots | P_n)$ où chaque processus P_i ne contient aucune instance de l'opérateur ν, c'est-à-dire qu'il ne comporte aucun nom privé. L'idée de base est de construire le graphe des transitions symboliques pour chacun des sous-processus P_i, de traduire chacun de ces graphes en un module PRISM particulier, et de considérer que le système global P est la composition parallèle de l'ensemble des modules. Cette approche par composition modulaire dépend de la sémantique symbolique du π_t-calcul. Sans la définition de la sémantique symbolique, nous ne pourrions pas générer la sémantique de chaque P_i pris isolément.

La structure globale du système, à savoir une composition parallèle d'un ensemble de sous-systèmes modélisés par des processus donc les voies de communications sont privées, est typique. Dans la plupart des cas, un processus peut être réécrit en un processus structurellement congruent, qui est de la forme correcte c'est-à-dire en plaçant les occurrences de l'opérateur de restriction ν à l'extérieur. Par exemple, les processus $P_1|(\nu x)P_2$ et $(\nu x)(P_1|P_2)$ sont structurellement congruent sous l'hypothèse que x n'est pas libre de P_1 : c'est la règle STRUCT EX du tableau 2.3. Les seuls processus qui ne peuvent pas être réécrits ainsi sont ceux qui incluent l'opérateur ν dans des définitions récursives. Dans ce cas, le processus peut en principe générer un nombre infini de nouveaux noms. Mais nous considérons que les systèmes finis.

Deux problèmes majeurs doivent être résolus pour la traductions des TSTGs en langage PRISM : faire correspondre un nom en π_t-calcul à une variable typée du langage PRISM ; et faire correspondre les échanges de nom entre processus avec la synchronisation sans passage de valeurs de PRISM. Le premier problème est résolu en énumérant l'ensemble de tous les noms libres, et en assignant à chacun d'eux, nommé de la même façon, une constante de type entier pour le représenter. Le deuxième problème est résolu en introduisant une étiquette d'action pour chaque paire « processus émetteur - processus récepteur », voie de communication et nom échangé. L'échange de nom entre processus est réglée en incluant dans chaque processus récepteur avec une variable entrée liée x, une variable locale appelée de la même façon qui sera utilisée pour enregistrer le nom libre assigné à x.

$$\text{TAU} : \frac{-}{\tau_{\{i,g,r,i'\}}.P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{true,\ \tau_{\{i,g,r,i'\}}} P} \qquad \text{OUTPUT} : \frac{-}{\bar{x}y_{\{i,g,r,i'\}}.P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{true,\ \bar{x}y_{\{i,g,r,i'\}}} P}$$

$$\text{INPUT} : \frac{-}{x(z)_{\{i,g,r,i'\}}.P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{true,\ x(w)_{\{i,g,r,i'\}}} P\{w/z\}}, w \notin fn((\nu z)P)$$

$$\text{MATCH} : \frac{P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{M,\ \alpha_{\{i,g,r,i'\}}} P'}{[x = y]P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{[x=y] \wedge M,\ \alpha_{\{i,g,r,i'\}}} P'}, \{x, y\} \cap bn(\alpha) = \emptyset$$

$$\text{SUM} : \frac{P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{M,\ \alpha_{\{i,g,r,i'\}}} P'}{P + Q \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{M,\ \alpha_{\{i,g,r,i'\}}} P'} \qquad \text{IDE} : \frac{P\{\vec{y}/\vec{x}\} \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{M,\ \alpha_{\{i,g,r,i'\}}} P'}{A(\vec{y}) \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{M,\ \alpha_{\{i,g,r,i'\}}} P'}, A(\vec{x}) \stackrel{def}{=} P$$

$$\text{PAR} : \frac{P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{M,\ \alpha_{\{i,g,r,i'\}}} P'}{P|Q \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{M,\ \alpha_{\{i,g,r,i'\}}} P'|Q}, bn(\alpha) \cap fn(Q) = \emptyset$$

$$\text{COM} : \frac{P \xrightarrow[i_1 \wedge g_1 \wedge v[r_1 \leftarrow 0] \wedge i'_1]{M,\ y(z)_{\{i_1,g_1,r_1,i'_1\}}} P' \quad Q \xrightarrow[i_2 \wedge g_2 \wedge v[r_2 \leftarrow 0] \wedge i'_2]{N,\ \bar{x}v_{\{i_2,g_2,r_2,i'_2\}}} Q'}{P|Q \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{[x=y] \wedge M \wedge N\ \tau_{\{i,g,r,i'\}}} P'\{v/z\}|Q'},$$
avec $i = i_1 \wedge i_2$, $g = g_1 \wedge g_2$, $r = r_1 \cup r_2$, $i' = i'_1 \wedge i'_2$

$$\text{CLOSE} : \frac{P \xrightarrow[i_1 \wedge g_1 \wedge v[r_1 \leftarrow 0] \wedge i'_1]{M,\ y(z)_{\{i_1,g_1,r_1,i'_1\}}} P' \quad Q \xrightarrow[i_2 \wedge g_2 \wedge v[r_2 \leftarrow 0] \wedge i'_2]{N,\ \bar{x}(v)_{\{i_2,g_2,r_2,i'_2\}}} Q'}{P|Q \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{[x=y] \wedge M \wedge N,\ \tau_{\{i,g,r,i'\}}} (\nu v)(P'\{v/z\}|Q')},$$
avec $i = i_1 \wedge i_2$, $g = g_1 \wedge g_2$, $r = r_1 \cup r_2$, $i' = i'_1 \wedge i'_2$

$$\text{RES} : \frac{P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{M,\ \alpha_{\{i,g,r,i'\}}} P'}{(\nu x)P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{M,\ \alpha_{\{i,g,r,i'\}}} (\nu x)P'}, \quad x \notin n(M, \alpha)$$

$$\text{OPEN} : \frac{P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{M,\ \bar{y}x_{\{i,g,r,i'\}}} P'}{(\nu x)P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{M,\ \bar{y}(x)_{\{i,g,r,i'\}}} P'}, \quad x \notin n(M, x)$$

TAB. 3.10 – Sémantique symbolique du π-calcul temporisé (π_t-calcul).

4.4.2 Règles de traduction

On suppose que l'ensemble des noms du système est $\mathcal{N}$ et que celui-ci est partitionné en sous-ensembles disjoints : $\mathcal{N}^{fn}$, l'ensemble de tous les noms libres qui apparaissent dans les processus $P_1, \ldots, P_n$, et $\mathcal{N}_1^{bn}, \ldots, \mathcal{N}_n^{bn}$, les ensembles des noms liés (par une action de réception) pour les processus $\{P_i\}_{i\in[1..n]}$

Lorsque c'est possible, on conserve les mêmes appellations entre les termes du π-calcul (processus et noms) et leur description en langage PRISM. Ainsi, chacun des n sous-processus, ou graphe de transitions symbolique, P_i devient un module PRISM P_i, et l'ensemble fini des termes $S_i = \{Q_1^i, \ldots, Q_{k_i}^i\}$ qui constituent les états du graphe de transitions symbolique de P_i devient un ensemble d'indices entiers $Q_1^i, \ldots, Q_{k_i}^i$ qui représente chacun d'eux de façon unique.

Le module P_i a $|\mathcal{N}_i^{bn}| + 1$ variables locales : son état local (c'est-à-dire l'état du graphe de transitions symbolique correspondant) est représenté par la variable s_i dont les valeurs appartiennent à $[Q_1^i, \ldots, Q_{k_i}^i]$, et chaque nom lié $x_j^i \in \mathcal{N}_i^{bn}$ a une variable correspondante x_j^i appartenant à $[0, \ldots, |\mathcal{N}^{fn}|$.

Le modèle inclue également $|\mathcal{N}^{fn}|$ constantes entières, une pour chaque nom libre, auxquelles sont assignées, dans un ordre arbitraire, des valeurs distinctes, non nulles et consécutives.Si la valeur de la variable x_j^i est égale à une de ces constantes, alors le nom lié correspondant a été assigné du nom libre approprié par une action de réception. Si $x_j^i = 0$, alors cela signifie qu'aucune réception pour le nom lié n'a encore eu lieu.

Ainsi, les conditions qui étiquettent les transitions du graphe de transitions symbolique peuvent être traduites directement en langage PRISM. Par exemple, si la condition M est $[x = a] \wedge [y = b]$ où x, y sont des noms libres, alors la traduction de M en PRISM est la même : $(x = a)\&(y = b)$, où x, y sont des variables de type entier, et a, b sont des constantes entières.

Les horloges associées à chaque processus sont traduites en des variables de même nom, de type `clock` dans le module PRISM correspondant au processus. Les invariants avant et après des transitions portent sur les états des processus, donc sur les états du TSTG, et sur les horloges des processus. Seuls les invariants avant sont traduits en langage PRISM en utilisant la construction `invariant ... endinvariant`. En effet, un invariant après de l'action courante se confond avec l'invariant avant de l'action suivante dans la séquence des actions réalisées par le processus.

Pour chaque transition du graphe de transitions symbolique de P_i, on inclue un ensemble de commandes PRISM correspondantes dans le module P_i. Dans la suite, on considère chaque type de transition séparément.

Transitions internes. Pour une transition :

$$Q_i \xrightarrow{M,\tau_{\{i,g,r,i'\}}} R_i$$

On ajoute, pour chaque horloge t de r, la commande :

$$[]\ (s_i = Q_i)\ \&\ M\ \&\ g\ \rightarrow (s_i' = R_i)\ \&\ (t' = 0);$$

Émission sur nom libre. Pour une transition :

$$Q_i \xrightarrow{M,\bar{x}y_{\{i,g,r,i'\}}} R_i \text{ où } x \in \mathcal{N}^{fn}$$

On ajoute, pour chaque $j \in \{1,\ldots,n\}\backslash\{i\}$, et pour horloge t de r, la commande :

$$[x\_P_i\_P_j\_y]\ (s_i = Q_i)\ \&\ M\ \&\ g\ \rightarrow (s_i' = R_i)\ \&\ (t' = 0);$$

La voie x, l'émetteur P_i, le récepteur P_j, et le nom transmis y sont encodés dans l'étiquette d'action.

Émission sur nom lié. Pour une transition :

$$Q_i \xrightarrow{M,\bar{x}y_{\{i,g,r,i'\}}} R_i, \text{ où } x \in \mathcal{N}_i^{bn}$$

On ajoute, pour chaque $a \in \mathcal{N}^{fn}$, pour chaque $j \in \{1,\ldots,n\}\backslash\{i\}$, et pour chaque horloge t de R, la commande :

$$[a\_P_i\_P_j\_y]\ (s_i = Q_i)\ \&\ M\ \&\ g\&(x = a)\ \rightarrow (s_i' = R_i)\ \&\ (t' = 0);$$

Ce cas est similaire au cas précédent, sauf que l'on inclut une commande pour chaque valeur possible a de x.

Réception sur nom libre. Pour une transition :

$$Q_i \xrightarrow{M,x(z)_{\{i,g,r,i'\}}} R_i \text{ où } x \in \mathcal{N}^{fn}$$

On ajoute, pour chaque $y \in \mathcal{N}\backslash\mathcal{N}_i^{bn}$, pour chaque $j \in \{1,\ldots,n\}\backslash\{i\}$, et pour chaque horloge t de r, la commande :

$$[x\_P_j\_P_i\_y]\ (s_i = Q_i)\ \&\ M\ \&\ g \rightarrow (s_i' = R_i)\ \&\ (z' = y)\ \&\ (t' = 0);$$

Pour les actions d'émission, on ajoute une ligne pour chaque nom y pouvant être reçu. L'affectation $(z' = y)$ modélise la mise à jour du nom lié z en y.

Réception sur nom lié. Pour une transition :

$$Q_i \xrightarrow{M,x(z)_{\{i,g,r,i'\}}} R_i \text{ où } x \in \mathcal{N}_i^{bn}$$

On ajoute, pour chaque $a \in \mathcal{N}^{fn}$, $y \in \mathcal{N}\backslash\mathcal{N}_i^{bn}$, pour chaque $j \in \{1,\ldots,n\}\backslash\{i\}$, et pour chaque horloge t de r, la commande :

$$[a\_P_j\_P_i\_y]\ (s_i = Q_i)\ \&\ M\ \&\ g\ \&\ (x = a) \rightarrow (s_i' = R_i)\ \&\ (z' = y)\ \&\ (t' = 0);$$

Ainsi, on ajoute une commande pour chaque paire possible de voie a que x peut représenter, et de nom y qu'il peut recevoir.

Enfin, il est nécessaire de supprimer les fausses commandes ajoutées dans les cas de la réception sur nom libre et de la réception sur nom lié, puisqu'elles correspondent à des émissions qui ne se produiront jamais. Plus précisément, pour chaque module P_j, on identifie les étiquettes $x\_P_i\_P_j\_y$ qui apparaissent dans une commande de P_j mais qui n'apparaissent dans aucune des commandes du module P_i. Les commandes avec de telles étiquettes sont supprimées de P_j.

N.B. La correction de la traduction suit exactement le même principe que celle établit dans [NPPW09] pour traduire le π-calcul probabiliste en processus de décision markoviens (MDP), et celle établit dans le même article pour traduire le π-calcul stochastique en chaîne de Markov à temps continu (CTMCs). Nous ne l'avons pas repris ici.

4.5 Exemple

On considère la composition parallèle de deux processus exprimé Q_1 et Q_2 en π_t-calcul. Ces processus sont dotés d'horloges, respectivement t_1 et t_2.

- $Q \triangleq (\nu a)(Q_1 \mid Q_2)$
- $Q_1 \triangleq (\nu c)(\nu d)(\bar{a}c_{I_1}.c(v)_{I_2}.0 + \bar{a}d_{I_3}.d(w)_{I_4}.0)$ avec :
  - $I_1 = \{t_1 \leqslant 2; t_1 \geqslant 1; t_1 \leqslant 5; t_1\}$,
  - $I_2 = \{t_1 \leqslant 5; t_1 \geqslant 3; true; t_1\}$,
  - $I_3 = \{t_1 \leqslant 2; t_1 \geqslant 1; true; t_1\}$,
  - $I_4 = \{true; t_1 \geqslant 1; true; t_1\}$;
- $Q_2 \triangleq (\nu b)(a(x)_{I_5}.\bar{b}x_{I_6}.0 \mid b(y)_{I_7}.\bar{y}e_{I_8}.0)$ avec :
  - $I_5 = \{t_2 \leqslant 2; t_2 \geqslant 1; t_2 \leqslant 5; t_2\}$,
  - $I_6 = \{t_2 \leqslant 5; t_2 \geqslant 3; true; t_2\}$,
  - $I_7 = \{true; t_2 \geqslant 1; true; t_2\}$,
  - $I_8 = \{true; t_2 \geqslant 1; true; t_2\}$.

Le processus Q est réécrit en P. Ces deux processus sont congruents. Chaque processus P_i est doté d'une horloge t_i. L'horloge t_2 du processus Q_2 est scindée en deux horloges synchrones t_3 et t_4, associées respectivement au processus P_2 et P_3. La réinitialisation de t_2 entraîne la réinitialisation de t_3 et t_4. Par ailleurs les contraintes temporelles sur les actions ne changent pas.

- $P \triangleq (\nu a)(\nu b)(\nu c)(\nu d)(P_1|P_2|P_3)$
- $P_1 \triangleq \bar{a}c_{I_1}.c(v)_{I_2}.0 + \bar{a}d_{I_3}.d(w)_{I_4}.0$ avec :
  - $I_1 = \{t_1 \leqslant 2; t_1 \geqslant 1; t_1 \leqslant 5; t_1\}$,
  - $I_2 = \{t_1 \leqslant 5; t_1 \geqslant 3; true; t_1\}$,
  - $I_3 = \{t_1 \leqslant 2; t_1 \geqslant 1; true; t_1\}$,
  - $I_4 = \{true; t_1 \geqslant 1; true; t_1\}$;
- $P_2 \triangleq a(x)_{I_5}.\bar{b}x_{I_6}.0$ avec :
  - $I_5 = \{t_3 \leqslant 2; t_3 \geqslant 1; t_3 \leqslant 5; t_3\}$,
  - $I_6 = \{t_3 \leqslant 5; t_3 \geqslant 3; true; t_3\}$,
- $P_3 \triangleq b(y)_{I_7}.\bar{y}e_{I_8}.0$ avec :
  - $I_7 = \{true; t_4 \geqslant 1; true; t_4\}$,
  - $I_8 = \{true; t_4 \geqslant 1; true; t_4\}$.

Les TSTGs correspondants sont :

- $P_1 : Q_1^1 \xrightarrow{\bar{a}c_{I_1}} Q_2^1 \xrightarrow{c(v)_{I_2}} Q_3^1$ ou $P_1 : Q_1^1 \xrightarrow{\bar{a}d_{I_3}} Q_4^1 \xrightarrow{d(w)_{I_4}} Q_5^1$
- $P_2 : Q_1^2 \xrightarrow{a(x)_{I_5}} Q_2^2 \xrightarrow{\bar{b}x_{I_6}} Q_3^2$
- $P_3 : Q_1^3 \xrightarrow{b(y)_{I_7}} Q_2^3 \xrightarrow{\bar{y}e_{I_8}} Q_3^3$

La traduction en langage PRISM par application des règles du paragraphe 4.4.2 donne le code PRISM représenté à la figure 3.11.

5 Une extension temporisée et probabiliste du π-calcul : le π_{pt}-calcul

Nous souhaitons maintenant prendre en compte les caractères temps réel et probabiliste des systèmes mobiles dont on veut analyser les propriétés de performabilité. Ces systèmes

```
1   pta
2
3   const int a = 1;
4   const int b = 2;
5   const int c = 3;
6   const int d = 4;
7   const int e = 5;
8
9   module P1
10    s1 : [1..5] init 1;
11    v : [0..5] init 0;
12    w : [0..5] init 0;
13    t1 : clock;
14    invariant
15    (s1=1=>t1<=2) & (s1=2=>t1<=5) & (s1=3=>t1<=2) &
16    (s1=4=>true) & (s1=5=>true)
      //i(s5)=i'(s4)
20    endinvariant
21
22    [a_P1_P2_c] (s1 = 1) & (t1>=1) -> (s1' = 2) & (t1'=0);
23    [a_P1_P2_d] (s1 = 1) & (t1>=1) -> (s1' = 4) & (t1'=0);
24    [c_P3_P1_e] (s1 = 2) & (t1>=3) -> (s1' = 3) & (v' = e) & (t1'=0);
25    [d_P3_P1_e] (s1 = 4) & (t1>=1) -> (s1' = 5) & (w' = e) & (t1'=0);
26  endmodule
27
28  module P2
29    s2 : [1..3] init 1;
30    x : [0..5] init 0;
31    t2 : clock;
32    invariant
33    (s2=1=>t2<=2) & (s2=2=>t2<=5) & (s2=3=>true)
36    endinvariant
37
38    [a_P1_P2_c] (s2 = 1) & (t2>=1) -> (s2' = 2) & (x' = c) & (t2'=0);
39    [a_P1_P2_d] (s2 = 1) & (t2>=1) -> (s2' = 2) & (x' = d) & (t2'=0);
40    [b_P2_P3_x] (s2 = 2) & (t2>=3) -> (s2' = 3) & (t2'=0);
41  endmodule
42
43  module P3
44    s3 : [1..3] init 1;
45    y : [0..5] init 0;
46    t3 : clock;
47    invariant
48    (s3=1=>true) & (s3=2=>true) & (s3=3=>true)
51    endinvariant
52
53    [b_P2_P3_x] (s3 = 1) & (t3>=1) -> (s3' = 2) & (y' = x) & (t3'=0);
54    [c_P3_P1_e] (s3 = 2) & (y = c) & (t3>=1) -> (s3' = 3) & (t3'=0);
55    [d_P3_P1_e] (s3 = 2) & (y = d) & (t3>=1) -> (s3' = 3) & (t3'=0);
56  endmodule
```

TAB. 3.11 – Traduction en langage PRISM d'un système spécifié en π_t-calcul.

peuvent être modélisés par des automates temporisés probabilistes. Nous proposons d'utiliser un formalisme de haut niveau, le π_{pt}-calcul, pour modéliser cette classe d'automates. Cette *algèbre de processus mobiles probabiliste et temps réel* fait l'objet de ce chapitre.

5.1 Préliminaire : les automates temporisés probabilistes

Ce paragraphe préliminaire est inspiré de l'article *Automatic Verification of Real-time Systems with Discrete Probability Distributions* de Marta Kwiatkowska et al. [KNSS02].

Les *automates probabilistes temporisés* (*PTAs* pour *probabilistic timed automata*) modélisent des systèmes dont les comportements sont probabilistes, non-déterministes, et temps réel. De façon informelle, les PTAs sont des processus de décision de Markov munis d'horloges (cf. annexe A). On peut également les considérer comme des automates temporisés exhibant des choix probabilistes.

5.1.1 Définition et propriétés

Définition 3.7. Un *automate temporisé probabiliste*, sur un ensemble de propositions atomiques PA, est un n-tuplet $(Loc, l_{init}, Act, X, inv, prob, L)$, où Loc est un ensemble fini de localisations, $l_{init} \in Loc$ est la localisation initiale, Act est un ensemble fini d'actions, X est un ensemble fini d'horloges, $inv : Loc \to Zones(X)$ est l'invariant, $prob \subseteq Loc \times Zones(X) \times Act \times Dist(Loc \times 2^X)$ est la relation probabiliste des transitions, et $L : Loc \to PA$ est une fonction d'étiquetage des localisations à l'aide de propositions atomiques de PA.

La loi de distribution des probabilités est une loi discrète. Un arc probabiliste est un élément $(l, g, a, p) \in prob$ où l est la *localisation source*, g est la *garde*, a est l'*action*, p est la *probabilité cible*. Un arc est un sextuplet (l, g, a, p, l', Y) : à partir de l'arc probabiliste (l, g, a, p) où $p(l', Y) > 0$, l' est la localisation cible, Y est l'ensemble des horloges devant être réinitialisées.

Exemple. Soit un protocole de communication probabiliste simple. La localisation de départ est d_i. Entre 1 et 2 unités de temps, le protocole essaie d'envoyer un message. Il y réussit avec la probabilité 0,9 et passe alors dans la localisation s_r. Avec la probabilité 0,1, le message est perdue et il passe dans la localisation s_i. Dans la localisation s_i, après 2 ou 3 unités temporelles, le protocole réessaie d'envoyer le message. Avec la probabilité 0,95, la donnée est correctement envoyée et avec la probabilité 0,05, le donnée est perdue.

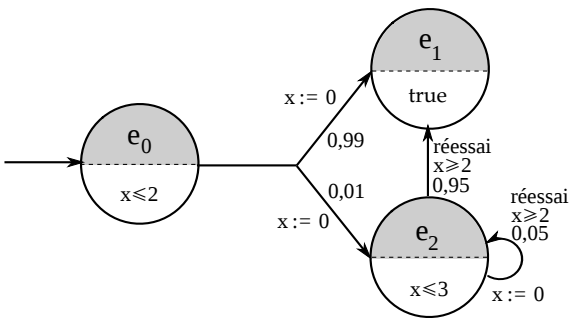


FIG. 3.5 – Exemple d'automate temporisé probabiliste.

5.1.2 Comportement d'un automate temporisé probabiliste

Un état d'un APT est une paire $(l, v) \in Loc \times \mathbb{R}^X_{\geqslant 0}$ telle que $v \triangleright inv(l)$. Un APT commence dans la localisation initiale avec toutes les horloges mises à zéro, ce qui est dénoté par $\underline{0}$. Pour tout état (l, v), il y a un choix non-déterminisme entre réaliser une transition discrète et laisser le temps passer :

- la transition discrète (l, g, a, p) est franchie si $v \triangleright g$ et si la probabilité de se déplacer dans la localisation l' et la réinitialisation des horloges Y est égal à $p(l', Y)$;
- la transition temporelle est disponible seulement si l'invariant $inv(l)$ est continuellement satisfait pendant que le temps s'écoule.

5.1.3 Sémantique d'un automate temporisé probabiliste

La sémantique d'un APT se définit à l'aide d'un processus de décision de Markov à états infinis $M_P = (S_P, s_{init}, Steps, PA, L_P)$ avec :

- L'espace des états : $S_P = \{(l, v) \in Loc \times \mathbb{R}^X_{\geqslant 0}$ tel que $v \triangleright inv(l)\}$;
- L'état initial : $s_{init} = (l_{init}, \underline{0})$;
- La fonction des probabilités des transitions : $Steps : S_P \to 2^{(Act \cup \mathbb{R}_{\geqslant 0}) \times Dist(S)}$ tel que $(\alpha, \mu) \in Steps(l, v)$ si et seulement si les actions du MDP M_P sont les actions de l'APT P ou des délais temps réels) :
  - (transition temporelle) $\alpha = t \in \mathbb{R}, \mu(l, v + t) = 1$ et $v + t' \triangleright inv(l) \forall t' \leqslant t$;
  - (transition discrète) $\alpha = a \in Act$ et $\exists (l, g, a, p) \in prob$ tel que $v \triangleright g$ et, $\forall (l'v') \in S_P \mu(l', v') = \sum_{Y \subseteq X \wedge v[Y:=0]=v'} p(l', Y)$: (plusieurs réinitialisations peuvent donner la même valuation d'horloge).
- La fonction d'étiquetage : $L_P(l, v) = L(l)$.

N.B. A la manière des automates temporisés, on restreint notre étude aux systèmes présentant un comportement à temps divergent, correspondant à des systèmes non Zénon.

5.1.4 Syntaxe

L'ensemble $\mathcal{P}^{\pi_{pt}}$ des expressions des processus du π-calcul probabiliste est défini par :

$$P ::= 0 \mid \alpha_{c,r}.P \mid P_1 + P_2 \mid p_1\tau_{c1,r1}.P_1 \oplus p_2\tau_{c2,r2}.P_2 \mid P_1|P_2 \mid (\nu x)P \mid [x = y]P \mid A(y_1, \ldots, y_n).$$

où :

- Les actions $\alpha_{c,r}$ sont les mêmes qu'en π_t-calcul présentées au paragraphe 4.2.
- Le processus $p_1\tau.P_1 \oplus p_2\tau.P_2$ représente le choix probabiliste entre les processus $\tau_{c1,r1}.P_1$ et $\tau_{c2,r2}.P_2$, tel que la probabilité de se comporter comme le premier est égale à p_1, alors que la probabilité de se comporter comme le deuxième est égale à p_2. Ce choix effectué, le processus réalise une action silencieuse τ_{c_i,r_i}, avant de se comporter comme $P_{i,i\in\{1,2\}}$. Les probabilités p_1 et p_2 sont telles que : $p_1, p_2 \in]0, 1]$ et $p_1 + p_2 = 1$. Le choix probabiliste est distinct du choix non-déterministe $P_1 + P_2$. Comme en π_{prob}, il y a deux opérateurs de choix en π_{pt}.
- Les autres expressions de processus sont les mêmes qu'en π-calcul définies au paragraphe 2.2.1.

6 Sémantique opérationnelle

Nous utilisons la même approche que celle utilisée pour établir la sémantique du π_t-calcul.

6.1 Sémantique concrète

La sémantique concrète du π_{pt}-calcul est définie par les règles de transitions du tableau 3.12 où I = i, g, r, i .

6.2 Sémantique symbolique

La sémantique symbolique des processus du π_{pt}-calcul s'exprime à l'aide de *graphes de transitions symboliques temporisés probabilistes* (*PTSTGs* pour *probabilistic timed symbolic transition graphs*). Ce sont des extensions temporisées et probabilistes des *graphes de transitions symboliques* définis par M. Hennessy et H. Lin dans [HL95].

Définition 3.8. Soit P un processus du π_{pt}-calcul. Le *graphe de transitions symbolique temporisé probabiliste* (PTSTG), qui définit la sémantique de P, est un triplet $(S, s_0, \mathcal{T}_{pt})$, où S est un ensemble d'états symboliques, chacun d'eux est un terme du π_{pt}-calcul ; l'état initial $s_0 \in S$ représente le terme P ; et $\mathcal{T} \subseteq S \times Cond \times Act \times Dist(S)$ est la plus petite relation de transition symbolique temporisée probabiliste définie par les règles du tableau 3.13. Dans cette définition, $Cond$ est l'ensemble de toutes les conditions (conjonctions finies de correspondances) sur $\mathcal{N}$, Act est l'ensemble des quatre actions du π_{pt}-calcul (τ, $x(y)$, $\bar{x}y$, et $\bar{x}(y)$ où $x, y \in \mathcal{N}$) ; et $Dist(S)$ est l'ensemble des distributions de probabilités sur S.

La transition symbolique temporisée $(Q, M, \alpha, \mu, Q') \in \mathcal{T}_{pt}$ est notée $Q \xrightarrow{M,\alpha} \{|p_i : Q'|\}$, où $\mu(R) = \sum_{Q_i=R} p_i$ pour n'importe quel terme R du π_{pt}. La transition $Q \xrightarrow{M,\alpha} \{|1 : Q'|\}$ est simplifiée en $Q \xrightarrow{M,\alpha} Q'$. Les transitions d'un processus $Q = p_1\tau_{c1,r1}.Q_1 \oplus p_2\tau_{c2,r2}.Q_2$, défini par une sommation probabiliste entre de deux processus Q_1 et Q_2, sont notées $Q \rightarrow \{|p_1 : \tau_{c1,r1}.Q_1, p_2 : \tau_{c2,r2}.Q_2|\}$. Une transition symbolique $Q \xrightarrow{M,\alpha} \{|p_i : Q'|\}$ représente le fait que, quelle que soit la substitution σ satisfaisant M, le terme de processus $Q\sigma$ peut réaliser l'action $\alpha\sigma$, avant d'évoluer en processus $Q'\sigma$ avec la probabilité p_i.

6.3 Traduction des PTSTGs en PRISM

Les principes de la traductions des PTSTGs en langage PRISM sint les mêmes que ceux présentés au paragraphe 4.4.1. On ne considère que les systèmes clos finis. les hypothèses faites pour traduire des TSTGs sont également conservées. Dans ce cas, les règles de traduction des PTSTGs en langage PRISM sont les mêmes que les règles de traduction précédentes, à l'exception de la traduction des transitions internes.

Transitions internes. Pour une transition :

$$Q_i \xrightarrow{M,\tau_{\{i,g,i',R\}}} \{|p_1 : R_1^i, \ldots, p_m : R_m^i|\}$$

On ajoute, pour toutes les horloges r, la commande :

$$[]\ (s_i = Q_i)\ \&\ M\ \&\ g \rightarrow p_1 : (s_i' = R_1^i)\ \&\ (r' = 0)\ +\ \cdots + p_m : (s_i' = R_m^i)\ \&\ (r' = 0);$$

La correction des règles de traduction s'établit de la même façon que celles établit dans [NPPW09] pour le π-calcul probabiliste et pour le π-calcul stochastique. Nous ne l'avons pas adaptée ici.

$$\text{TAU} : \frac{-}{\tau_{\{i,g,r,i'\}}.P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{\tau_{\{i,g,r,i'\}}} \{|1 : P|\}}$$

$$\text{OUTPUT} : \frac{-}{\bar{x}y_{\{i,g,r,i'\}}.P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{\bar{x}y_{\{i,g,r,i'\}}} \{|1 : P|\}}$$

$$\text{INPUT} : \frac{-}{x(z)_{\{i,g,r,i'\}}.P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{x(w)_{\{i,g,r,i'\}}} \{|1 : P\{w/z\}|\}}, w \notin fn((\nu z)P)$$

$$\text{SUM} : \frac{P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{\alpha_{\{i,g,r,i'\}}} \{|p_i : P'|\}}{P + Q \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{\alpha_{\{i,g,r,i'\}}} \{|p_i : P'|\}} \qquad \text{MATCH} : \frac{P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{\alpha_{\{i,g,r,i'\}}} \{|p_i : P'|\}}{[x = x]P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{\alpha_{\{i,g,r,i'\}}} \{|p_i : P'|\}}$$

$$\text{IDE} : \frac{P\{\vec{y}/\vec{x}\} \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{\alpha_{\{i,g,r,i'\}}} \{|p_i : P'|\}}{A(\vec{y}) \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{\alpha_{\{i,g,r,i'\}}} \{|p_i : P'|\}}, A(\vec{x}) \stackrel{def}{=} P$$

$$\text{PAR} : \frac{P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{\alpha_{\{i,g,r,i'\}}} \{|p_i : P'|\}_i}{P|Q \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{\alpha_{\{i,g,r,i'\}}} \{|p_i : (P'|Q)|\}_i}, bn(\alpha) \cap fn(Q) = \emptyset$$

$$\text{COM} : \frac{P \xrightarrow[i_1 \wedge g_1 \wedge v[r_1 \leftarrow 0] \wedge i'_1]{x(z)_{\{i_1,g_1,r_1,i'_1\}}} \{|1 : P'|\} \quad Q \xrightarrow[i_2 \wedge g_2 \wedge v[r_2 \leftarrow 0] \wedge i'_2]{\bar{x}v_{\{i_2,g_2,r_2,i'_2\}}} \{|1 : Q'|\}}{P|Q \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{\tau_{\{}i,g,r,i'\}} \{|1 : P'\{v/z\}|Q'|\}},$$
avec $i = i_1 \wedge i_2$, $g = g_1 \wedge g_2$, $r = r_1 \cup r_2$, $i' = i'_1 \wedge i'_2$

$$\text{CLOSE} : \frac{P \xrightarrow[i_1 \wedge g_1 \wedge v[r_1 \leftarrow 0] \wedge i'_1]{x(z)_{\{i_1,g_1,r_1,i'_1\}}} \{|1 : P'|\} \quad Q \xrightarrow[i_2 \wedge g_2 \wedge v[r_2 \leftarrow 0] \wedge i'_2]{\bar{x}(v)_{\{i_2,g_2,r_2,i'_2\}}} \{|1 : Q'|\}}{P|Q \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{\tau_{\{i,g,r,i'\}}} \{|1 : (\nu v)(P'\{v/z\}|Q')|\}},$$
avec $i = i_1 \wedge i_2$, $g = g_1 \wedge g_2$, $r = r_1 \cup r_2$, $i' = i'_1 \wedge i'_2$

$$\text{RES} : \frac{P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{\alpha_{\{i,g,r,i'\}}} \{|p_i : P'|\}}{(\nu x)P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{\alpha_{\{i,g,r,i'\}}} \{|p_i : (\nu x)P'|\}}, \quad x \notin n(\alpha)$$

$$\text{OPEN} : \frac{P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{\bar{x}y_{\{i,g,r,i'\}}} \{|1 : P'|\}}{(\nu y)P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{\bar{x}(w)_{\{i,g,r,i'\}}} \{|1 : P'\{w/y\}|\}}, \quad x \neq y,\ w \notin fn((\nu y)P')$$

TAB. 3.12 – Sémantique concrète du π-calcul temporisé probabiliste (π_{pt}-calcul).

$$\text{TAU} : \frac{-}{\tau_{\{i,g,r,i'\}}.P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{true,\ \tau_{\{i,g,r,i'\}}} \{|1 : P|\}}$$

$$\text{OUTPUT} : \frac{-}{\bar{x}y_{\{i,g,r,i'\}}.P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{true,\ \bar{x}y_{\{i,g,r,i'\}}} \{|1 : P|\}}$$

$$\text{INPUT} : \frac{-}{x(z)_{\{i,g,r,i'\}}.P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{true,\ x(w)_{\{i,g,r,i'\}}} \{|1 : P\{w/z\}|\}}, w \notin fn((\nu z)P)$$

$$\text{SUM} : \frac{P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{M,\ \alpha_{\{i,g,r,i'\}}} \{|p_i : P'|\}}{P + Q \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{M,\ \alpha_{\{i,g,r,i'\}}} \{|p_i : P'|\}} \qquad \text{MATCH} : \frac{P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{M,\ \alpha_{\{i,g,r,i'\}}} \{|p_i : P'|\}}{[x = y]P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{[x=y] \wedge M,\ \alpha_{\{i,g,r,i'\}}} \{|p_i : P'|\}}$$

$$\text{IDE} : \frac{P\{\vec{y}/\vec{x}\} \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{M, \alpha_{\{i,g,r,i'\}}} \{|p_i : P'|\}}{A(\vec{y}) \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{M,\ \alpha_{\{i,g,r,i'\}}} \{|p_i : P'|\}}, A(\vec{x}) \stackrel{def}{=} P$$

$$\text{PAR} : \frac{P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{M,\ \alpha_{\{i,g,r,i'\}}} \{|p_i : P'|\}_i}{P|Q \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{M,\ \alpha_{\{i,g,r,i'\}}} \{|p_i : (P'|Q)\}_i|}, bn(\alpha) \cap fn(Q) = \emptyset$$

$$\text{COM} : \frac{P \xrightarrow[i_1 \wedge g_1 \wedge v[r_1 \leftarrow 0] \wedge i'_1]{M,\ x(z)_{\{i_1,g_1,r_1,i'_1\}}} \{|1 : P'|\} \quad Q \xrightarrow[i_2 \wedge g_2 \wedge v[r_2 \leftarrow 0] \wedge i'_2]{N,\ \bar{x}v_{\{i_2,g_2,r_2,i'_2\}}} \{|1 : Q'|\}}{P|Q \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{[x=y] \wedge M \wedge N,\ \tau_{\{}i,g,r,i'\}} \{|1 : P'\{v/z\}|Q'|\}},$$
avec $i = i_1 \wedge i_2$, $g = g_1 \wedge g_2$, $r = r_1 \cup r_2$, $i' = i'_1 \wedge i'_2$

$$\text{CLOSE} : \frac{P \xrightarrow[i_1 \wedge g_1 \wedge v[r_1 \leftarrow 0] \wedge i'_1]{M,\ x(z)_{\{i_1,g_1,r_1,i'_1\}}} \{|1 : P'|\} \quad Q \xrightarrow[i_2 \wedge g_2 \wedge v[r_2 \leftarrow 0] \wedge i'_2]{N,\ \bar{x}(v)_{\{i_2,g_2,r_2,i'_2\}}} \{|1 : Q'|\}}{P|Q \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{[x=y] \wedge M \wedge N,\ \tau_{\{i,g,r,i'\}}} \{|1 : (\nu v)(P'\{v/z\}|Q')|\}},$$
avec $i = i_1 \wedge i_2$, $g = g_1 \wedge g_2$, $r = r_1 \cup r_2$, $i' = i'_1 \wedge i'_2$

$$\text{RES} : \frac{P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{M,\ \alpha_{\{i,g,r,i'\}}} \{|p_i : P'|\}}{(\nu x)P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{M,\ \alpha_{\{i,g,r,i'\}}} \{|p_i : (\nu x)P'|\}}, \quad x \notin n(M, \alpha)$$

$$\text{OPEN} : \frac{P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{M,\ \bar{x}y_{\{i,g,r,i'\}}} \{|1 : P'|\}}{(\nu y)P \xrightarrow[i \wedge g \wedge v[r \leftarrow 0] \wedge i']{M,\ \bar{x}(w)_{\{i,g,r,i'\}}} \{|1 : P'\{w/y\}|\}}, \quad x \notin n(M, x)$$

TAB. 3.13 – Sémantique symbolique du π-calcul temporisé probabiliste (π_{pt}-calcul).

6.4 Exemple

On modifie l'exemple précédent en considérant :

- $Q \triangleq (\nu a)(Q_1 \mid Q_2)$
- $Q_1 \triangleq (\nu c)(\nu d)(\frac{1}{2}\tau.\bar{a}c_{I_1}.c(v)_{I_2}.0 \oplus \frac{1}{2}\tau.\bar{a}d_{I_3}.d(w)_{I_4}.0)$ avec :
  - $I_1 = \{t_1 \leqslant 2; t_1 \geqslant 1; t_1 \leqslant 5; t_1\}$,
  - $I_2 = \{t_1 \leqslant 5; t_1 \geqslant 3; true; t_1\}$,
  - $I_3 = \{t_1 \leqslant 2; t_1 \geqslant 1; true; t_1\}$,
  - $I_4 = \{true; t_1 \geqslant 1; true; t_1\}$;
- $Q_2 \triangleq (\nu b)(a(x)_{I_5}.\bar{b}x_{I_6}.0 \mid b(y)_{I_7}.\bar{y}e_{I_8}.0)$ avec :
  - $I_5 = \{t_2 \leqslant 2; t_2 \geqslant 1; t_2 \leqslant 5; t_2\}$,
  - $I_6 = \{t_2 \leqslant 5; t_2 \geqslant 3; true; t_2\}$,
  - $I_7 = \{true; t_2 \geqslant 1; true; t_2\}$,
  - $I_8 = \{true; t_2 \geqslant 1; true; t_2\}$.

Le processus Q est réécrit en P. Les hypothèses de l'exemple précédent sont conservées.

- $P \triangleq (\nu a)(\nu b)(\nu c)(\nu d)(P_1|P_2|P_3)$
- $P_1 \triangleq \frac{1}{2}\tau.\bar{a}c_{I_1}.c(v)_{I_2}.0 \oplus \frac{1}{2}\tau.\bar{a}d_{I_3}.d(w)_{I_4}.0$ avec :
  - $I_1 = \{t_1 \leqslant 2; t_1 \geqslant 1; t_1 \leqslant 5; t_1\}$,
  - $I_2 = \{t_1 \leqslant 5; t_1 \geqslant 3; true; t_1\}$,
  - $I_3 = \{t_1 \leqslant 2; t_1 \geqslant 1; true; t_1\}$,
  - $I_4 = \{true; t_1 \geqslant 1; true; t_1\}$;
- $P_2 \triangleq a(x)_{I_5}.\bar{b}x_{I_6}.0$ avec :
  - $I_5 = \{t_3 \leqslant 2; t_3 \geqslant 1; t_3 \leqslant 5; t_3\}$,
  - $I_6 = \{t_3 \leqslant 5; t_3 \geqslant 3; true; t_3\}$,
- $P_3 \triangleq b(y)_{I_7}.\bar{y}e_{I_8}.0$ avec :
  - $I_7 = \{true; t_4 \geqslant 1; true; t_4\}$,
  - $I_8 = \{true; t_4 \geqslant 1; true; t_4\}$.

Les TSTGs correspondants sont :

- $P_1 : Q_1^1 \xrightarrow{\tau} \{|\frac{1}{2} : Q_2^1, \frac{1}{2} : Q_3^1|\}$ et $Q_2^1 \xrightarrow{\bar{a}c_{I_1}} Q_4^1 \xrightarrow{c(v)_{I_2}} Q_6^1$ et $Q_3^1 \xrightarrow{\bar{a}d_{I_3}} Q_5^1 \xrightarrow{d(w)_{I_4}} Q_6^1$
- $P_2 : Q_1^2 \xrightarrow{a(x)_{I_5}} Q_2^2 \xrightarrow{\bar{b}x_{I_6}} Q_3^2$
- $P_3 : Q_1^3 \xrightarrow{b(y)_{I_7}} Q_2^3 \xrightarrow{\bar{y}e_{I_8}} Q_3^3$

Le code PRISM correspondant est représenté à la figure 3.14.

```
1   pta
2
3   const int a = 1;
4   const int b = 2;
5   const int c = 3;
6   const int d = 4;
7   const int e = 5;
8
9   module P1
10    s1 : [1..6] init 1;
11    v : [0..5] init 0;
12    w : [0..5] init 0;
13    t1 : clock;
14    invariant
15    (s1=1=>t1<=2) & (s1=2=>t1<=5) & (s1=3=>t1<=2) &
16    (s1=4=>true) & (s1=5=>true)
20    endinvariant
21
22    [] (s1 = 1) -> 0.5 : (s1 = 2) + 0.5 : (s1 = 3);
22    [a_P1_P2_c] (s1 = 2) & (t1>=1) -> (s1' = 4) & (t1'=0);
23    [a_P1_P2_d] (s1 = 3) & (t1>=1) -> (s1' = 5) & (t1'=0);
24    [c_P3_P1_e] (s1 = 4) & (t1>=3) -> (s1' = 6) & (v' = e) & (t1'=0);
25    [d_P3_P1_e] (s1 = 5) & (t1>=1) -> (s1' = 6) & (w' = e) & (t1'=0);
26  endmodule
27
28  module P2
29    s2 : [1..3] init 1;
30    x : [0..5] init 0;
31    t2 : clock;
32    invariant
33    (s2=1=>t2<=2) & (s2=2=>t2<=5) & (s2=3=>true)
36    endinvariant
37
38    [a_P1_P2_c] (s2 = 1) & (t2>=1) -> (s2' = 2) & (x' = c) & (t2'=0);
39    [a_P1_P2_d] (s2 = 1) & (t2>=1) -> (s2' = 2) & (x' = d) & (t2'=0);
40    [b_P2_P3_x] (s2 = 2) & (t2>=3) -> (s2' = 3) & (t2'=0);
41  endmodule
42
43  module P3
44    s3 : [1..3] init 1;
45    y : [0..5] init 0;
46    t3 : clock;
47    invariant
48    (s3=1=>true) & (s3=2=>true) & (s3=3=>true)
51    endinvariant
52
53    [b_P2_P3_x] (s3 = 1) & (t3>=1) -> (s3' = 2) & (y' = x) & (t3'=0);
54    [c_P3_P1_e] (s3 = 2) & (y = c) & (t3>=1) -> (s3' = 3) & (t3'=0);
55    [d_P3_P1_e] (s3 = 2) & (y = d) & (t3>=1) -> (s3' = 3) & (t3'=0);
56  endmodule
```

TAB. 3.14 – Traduction en langage PRISM d'un système spécifié en π_{pt}-calcul.

Chapitre 4

Applications

Dans ce chapitre nous mettons en œuvre les concepts précédents pour analyser une application de sécurité hospitalière basée sur un réseau de capteurs sans fil.

1 Un réseau de capteurs sans fil pour la sécurité hospitalière

L'application permet au personnel médical d'envoyer des alarmes dans des situations d'urgence, par l'intermédiaire d'un capteur sans fil que chaque personne porte sur elle en permanence [KHH11]. Cette application permet également des mesures bio-médicales de capteur, et des contrôles d'actionneurs. Comme cela est illustré à la figure 4.1, cette application se base sur un RdCSF connecté à deux serveurs via un réseau local Ethernet (IEEE 802.3 [IEE08]). Le réseau de capteurs est composé de nœuds mobiles (c'est-à-dire, les dispositifs d'alarme portés en permanence par le personnel médical en intervention), de routeur fixes, et de stations de base fixes également fixes. Les communications sans fil se basent sur le standard IEEE 802.11 [IEE07]. Les nœuds mobiles ont une activité cyclique. Chacune période active comprend une fenêtre de transmission, durant laquelle les nœuds peuvent diffuser une alarme aux routeurs voisins et une fenêtre de réception pour recevoir l'accusé de réception correspondant qui indique que l'aide arrive.

Les routeurs sont actifs perpétuellement. Ils envoient les alarmes à une ou plusieurs stations de base. Ils accèdent au canal de transmission de façon aléatoire selon l'*algorithme du backoff*. En cas des collisions, les trames MAC entrées en collision ne sont pas l'objet d'accusés de réception, et donc les routeurs les retransmettent après des temps d'attente aléatoires. Les balises de synchronisation transmises sur un canal de fréquence différente sont échangées par les routeurs pour maintenir les tables de routage à jour.

Les stations de base transmettent les alarmes envoyées par les nœuds mobiles, et relayées par les routeurs, au serveur principal nommé WSN à la figure 4.1. Les stations se comportent comme des passerelles entre le réseau sans fil, et le réseau local Ethernet. Le serveur WSN interprète les alarmes qu'il reçoit, avant de déterminer l'emplacement du dispositif qui a envoyé l'alarme. Il sert également d'interface au serveur d'alarmes dont le rôle est de transmettre les alarmes au personnel de garde concerné via des interfaces, qui peuvent être, par exemple, des ordinateurs individuels ou des téléphones portables.

Nous considérons l'implémentation du réseau illustrée à la figure 4.2, mettant en œuvre deux dispositifs mobiles $N1$ et $N2$, trois routeurs $R1$, $R2$ et $R3$ et deux stations de base $Si1$ et $Si2$. Par exemple, les alarmes modiles sont portées par deux infirmières pendant leur service. Comme cela est précisé dans [KHH11], chaque chambre de patient est équipée d'un

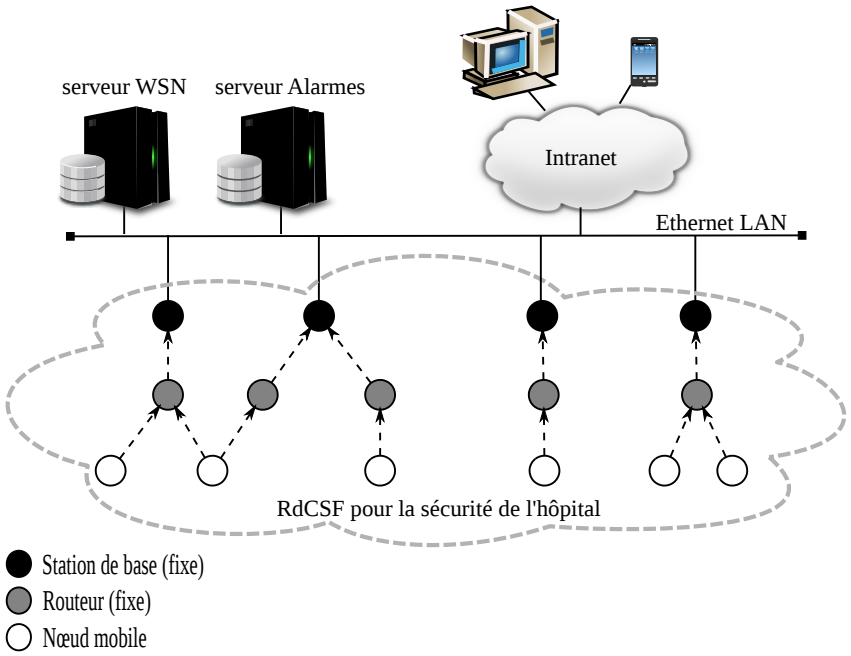


FIG. 4.1 – Architecture d'un réseau de sécurité implémenté à l'hôpital de Tampere (Finlande). Les lignes en pointillés représentent les communications sans fil. Tous les nœuds existants n'ont pas été représentés dans cette figure.

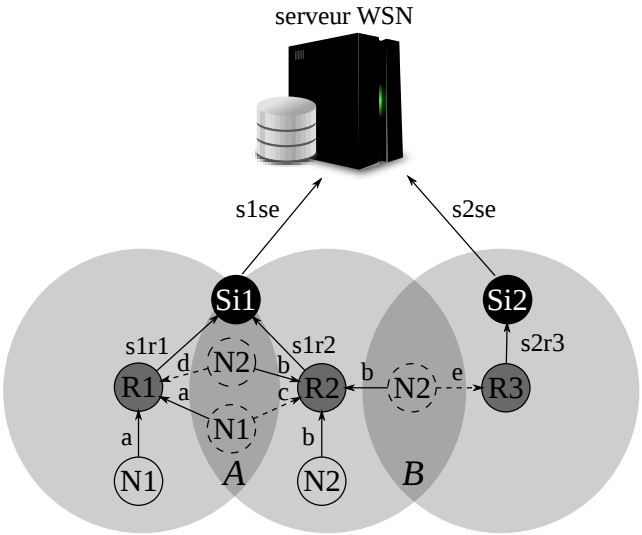


FIG. 4.2 – Une implémentation du réseau sans fil mobile. Les zones circulaires grisées représentent les zones de couverture des routeurs. Dans le cas figuré, on suppose que ces surfaces sont les mêmes.

routeur fixe. Nous supposons que les infirmières peuvent déclencher une alarme en restant immobile, ou en se déplaçant entre deux chambres ou dans le couloir. Les chambres ne sont pas nécessaires proches l'une de l'autre, mais les zones de couverture de leur routeur respectif ne sont pas disjointes. Nous supposons que $N1$ et $N2$ peut toujours communiquer avec des routeurs $R1$ et $R2$, respectivement via les canaux de communication a et b. Au début de sa période active, après une phase de contention modélisée comme une action silencieuse, τ, le nœud N_i envoie une alarme ($alarm_i$) au routeur R_j, qui la retransmet au serveur WSN via une station de base. Dans le cas où l'alarme est transmise au serveur sans être corrompue, le serveur renvoie un accusé de réception pour avertir l'utilisateur que l'alarme qu'il vient de déclencher a été prise en considération et que les secours arrivent.

Nous supposons que lorsque $N1$ est dans la zone « A » , il peut communiquer avec R_2 via le canal c. De la même manière, $N2$ peut communiquer avec $R1$ via le canal d lorsqu'il est dans la zone « A », ou avec $R3$ via le canal e quand il est dans la zone « B ». Les canaux d et e ne peuvent pas exister simultanément, parce que $N2$ ne peut pas être dans le secteur « A » et « B » en même temps. $R3$ transmet les alarmes qu'il reçoit à la station $Si2$ via le canal $s2r3$, qui communique elle même avec le serveur WSN via le canal $s2se$.

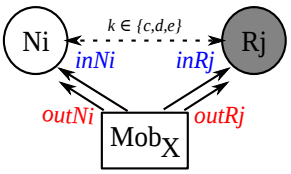


FIG. 4.3 – Apparition et disparition d'une voie de communication k entre un nœud mobile N_i et un routeur R_j, selon le modèle de mobilité Mob_X.

Comme cela est illustré à la figure 4.3, le canal k entre N_i et R_j apparaît, quand Mob_X leur envoie k, respectivement sur les canaux inR_j et inN_i. Ce canal disparaît quand le même modèle de mobilité leurs envoie respectivement les canaux ouN_i et ouR_j. Par exemple, l'apparition du canal c entre $N1$ et $R2$, suivie par sa disparition est modélisé par :

$$\begin{cases} Mob_{C1} \triangleq \overline{in_{R2}}\langle c\rangle.\overline{in_{N1}}\langle c\rangle.Mob_{C2} \\ Mob_{C2} \triangleq \overline{out_{N1}}\langle c\rangle.\overline{out_{R2}}\langle c\rangle.Mob_{C1} \end{cases}$$

Nous considérons les modèles de mobilité suivants, qui correspondent à dix scénarii de mouvements des nœuds mobiles.

- Les nœuds mobiles sont immobiles : $Mob_0 \triangleq 0$

- Mob_C : la voie c apparaît puis disparaît. Le modèle de mobilité correspondant est donné ci-dessus.

- De la même façon, la voie d apparaît puis disparaît (Mob_D), ou la voie e apparaît puis disparaît (Mob_E).

- Mob_F : la voie c apparaît puis disparaît, puis la voie d apparaît et disparaît à son tour :

$$\begin{cases} Mob_{F1} \triangleq \overline{inR2}\langle c\rangle.\overline{inN1}\langle c\rangle.Mob_{F2} \\ Mob_{F2} \triangleq \overline{outN1}\langle c\rangle.\overline{outR2}\langle c\rangle.Mob_{F3} \\ Mob_{F3} \triangleq \overline{inR1}\langle d\rangle.\overline{inN2}\langle d\rangle.Mob_{F4} \\ Mob_{F4} \triangleq \overline{outN2}\langle d\rangle.\overline{outR1}\langle d\rangle.Mob_{F1} \end{cases}$$

$$
\begin{array}{rcl}
N_1 & \triangleq & \tau_{contention1}.\overline{a}\langle alarm1\rangle.a(v1a).([v1a = ack1]0 + [v1a = nack]N_1) + inN1(x1).N_{1a} \\
N_{1a} & \triangleq & \tau_{contention1}.\overline{\beta}_{\{a,x_1\}}\langle alarm1\rangle.a(v1a).([v1a = ack1]x1(v1b).([v1b = ack1]0 + [v1b = nack]0)// \ (1) \\
 & & \quad + ([v1a = nack]x1(v1b).([v1b = ack1]0 + [v1b = nack]N_{1a}) + outN1(x1).N_1 \\
 & & \\
N_2 & \triangleq & \tau_{contention2}.\overline{b}\langle alarm2\rangle.b(v2a).([v2a = ack2]0 + [v2a = nack]N_2) + inN2(x2).N_{2a} \\
N_{2a} & \triangleq & \tau_{contention2}.\overline{\beta}_{\{b,x_2\}}\langle alarm2\rangle.b(v2a).([v2a = ack2]x2(v2b).([v2b = ack2]0 + [v2a = nack]0)// \ (2) \\
 & & \quad + ([v2a = nack]x2(v2b).([v2b = ack2]0 + [v2b = nack]N_{2a}) + outN2(x2).N_2 \\
 & & \\
R_1 & \triangleq & a(u3a).R_{1a} \\
R_{1a} & \triangleq & \overline{s1r1}\langle u3a\rangle.s1r1(v3a).([v3a = ack1]\overline{a}\langle v3a\rangle.R_1 + [v3a = nack]\tau_{contentionR1}R_{1a}) + inR1(x3).R_{1b} \\
R_{1b} & \triangleq & a(u3a).R_{1c} + x3(u3b).R_{1d} + outR1(x3).R_1 \\
R_{1c} & \triangleq & \overline{s1r1}\langle u3a\rangle.s1r1(v3a).([v3a = ack1]\overline{a}\langle v3a\rangle.R_{1b} + [v3a = nack]\tau_{contentionR1}.R_{1c}) \\
 & & \quad + x3(u3b).\overline{a}\langle nack\rangle.\overline{x3}\langle nack\rangle.R_{1b} \\
R_{1d} & \triangleq & \overline{s1r1}\langle u3b\rangle.s1r1(v3b).([v3b = ack2]\overline{x3}\langle v3b\rangle.R_{1b} + [v3b = nack]\tau_{contentionR1}.R_{1d}) \\
 & & \quad + a(u3a).\overline{x3}\langle nack\rangle.\overline{a}\langle nack\rangle.R_{1b} \\
 & & \\
R_2 & \triangleq & b(u4a).R_{2a} \\
R_{2a} & \triangleq & \overline{s1r2}\langle u4a\rangle.s1r2(v4a).([v4a = ack2]\overline{b}\langle v4a\rangle.R_2 + [v4a = nack]\tau_{contentionR2}.R_{2a}) + inR2(x4).R_{2b} \\
R_{2b} & \triangleq & b(u4a).R_{2c} + x4(u4b).R_{2d} + outR2(x4).R_2 \\
R_{2c} & \triangleq & \overline{s1r2}\langle u4a\rangle.s1r2(v4a).([v4a = ack2]\overline{b}\langle v4a\rangle.R_{2b} + [v4a = nack]\tau_{contentionR2}R_{2c}) \\
 & & \quad + x4(u4b).\overline{b}\langle nack\rangle.\overline{x4}\langle nack\rangle.R_{2b} \\
R_{2d} & \triangleq & \overline{s1r2}\langle u4b\rangle.s1r2(v4b).([v4b = ack1]\overline{x4}\langle v4b\rangle.R_{2b} + [v4b = nack]\tau_{contentionR2}R_{2d}) \\
 & & \quad + b(u4a).\overline{x4}\langle nack\rangle.\overline{b}\langle nack\rangle.R_{2b} \\
 & & \\
R_3 & \triangleq & inR3(x5).R_{3a} \\
R_{3a} & \triangleq & x5(u5a).\overline{s2r3}\langle u5a\rangle.s2r3(v5a).\overline{x}\langle v5a\rangle.R_{3a} + outR3(x5).R_3 \\
 & & \\
Si_1 & \triangleq & s1r1(u6a).Si_{1a} + s1r2(u6b).Si_{1b} \\
Si_{1a} & \triangleq & \overline{s1se}\langle u6a\rangle.s1se(v6a).([v6a = ack1]\overline{s1r1}\langle v6a\rangle.Si_1 + [v6a = ack2]\overline{s1r1}\langle v6a\rangle.Si_1 \\
 & & \quad + [v6a = nack]\tau_{contentionSi1}.Si_{1a}) \\
 & & \quad + s1r2(u6b).\overline{s1r1}\langle nack\rangle.\overline{s1r2}\langle nack\rangle.Si_1 \\
Si_{1b} & \triangleq & \overline{s1se}\langle u6b\rangle.s1se(v6b).([v6b = ack1]\overline{s1r2}\langle v6b\rangle.Si_1 + [v6b = ack1]\overline{s1r2}\langle v6b\rangle.Si_1 \\
 & & \quad + [v6b = nack]\tau_{contentionSi1}.Si_{1b}) \\
 & & \quad + s1r1(u6a).\overline{s1r2}\langle nack\rangle.\overline{s1r1}\langle nack\rangle.Si_1 \\
 & & \\
Si_2 & \triangleq & s2r3(u7a).Si_{2a} \\
Si_{2a} & \triangleq & \overline{s2se}\langle u7a\rangle.s2se(v7a).([v7a = ack2]\overline{s2r3}\langle v7a\rangle.Si_2 + [v7a = nack]\tau_{contentionSi2}.Si_{2a}) \\
 & & \\
Se & \triangleq & s1se(u8a).([u8a = alarm1]\overline{s1se}\langle ack1\rangle.Se \\
 & & \quad + [u8a = alarm2]\overline{s1se}\langle ack2\rangle.Se + s2se(u8b).\overline{s1se}\langle nack\rangle.\overline{s2se}\langle nack\rangle.Se) \\
 & & \quad + s2se(u8b).([u8b = alarm2]\overline{s2se}\langle ack2\rangle.Se + s1se(u8a).\overline{s2se}\langle nack\rangle.\overline{s1se}\langle nack\rangle.Se) \\
 & & \\
Mob_X & \triangleq & Mob_0 \ // \ or \ Mob_C \ \ldots \ or \ Mob_K \ (only \ one \ at \ the \ same \ time) \\
 & & \\
System & \triangleq & (\nu\, a)\ (\nu\, b)\ (\nu\, c)\ (\nu\, d)\ (\nu\, e)\ (\nu\, alarm1)\ (\nu\, alarm2)\ (\nu\, ack1)\ (\nu\, ack2)\ (\nu\, nack) \\
 & & (\nu\, s1r1)\ (\nu\, s1r2)\ (\nu\, s2r3)\ (\nu\, inN1)\ (\nu\, outN1)\ (\nu\, inN2)\ (\nu\, outN2) \\
 & & (\nu\, inR1)\ (\nu\, outR1)\ (\nu\, inR2)\ (\nu\, outR2)\ (\nu\, inR3)\ (\nu\, outR3)\ (\nu\, s1se)\ (\nu\, s2se) \\
 & & \quad (N_1 \mid N_2 \mid R_1 \mid R_2 \mid R_3 \mid Si_1 \mid Si_2 \mid Se \mid Mob_X)
\end{array}
$$

FIG. 4.4 – Le modèle du réseau global (le réseau de capteur sans fil et le réseau Ethernet) comprenant les communications et la mobilité des nœuds modélisé en π_{stoc}. Les modèles de mobilité des nœuds Mob_X sont présentés dans ce paragraphe.

– Mob_G : la voie c apparaît puis la voie d apparaît, avant que les deux disparaîssent dans l'ordre où elles sont apparues :

$$\begin{cases} Mob_{G1} \triangleq \overline{inR2}\langle c\rangle.\overline{inN1}\langle c\rangle.Mob_{G2} \\ Mob_{G2} \triangleq \overline{inR1}\langle d\rangle.\overline{inN2}\langle d\rangle.Mob_{G3} \\ Mob_{G3} \triangleq \overline{outN1}\langle c\rangle.\overline{outR2}\langle c\rangle.Mob_{G4} \\ Mob_{G4} \triangleq \overline{outN2}\langle d\rangle.\overline{outR1}\langle d\rangle.Mob_{G1} \end{cases}$$

– Mob_H : la voie d apparaît et disparaît, suivie par l'apparition et la disparition de la voie e (ce modèle est similaire à Mob_F).

– Mob_I : la voie c apparaît puis la voie e apparaît, avant que les deux disparaissent dans le même ordre où elles sont apparues ; ce modèle est similaire à Mob_G).

– Mob_J : la voie c apparaît, la voie d apparaît puis disparaît, la voie e apparaît, puis disparaît :

$$\begin{cases} Mob_{J1} \triangleq \overline{inR2}\langle c\rangle.\overline{inN1}\langle c\rangle.Mob_{J2} \\ Mob_{J2} \triangleq \overline{inR1}\langle d\rangle.\overline{inN2}\langle d\rangle.Mob_{J3} \\ Mob_{J3} \triangleq \overline{outN2}\langle d\rangle.\overline{outR1}\langle d\rangle.Mob_{J4} \\ Mob_{J4} \triangleq \overline{inR3}\langle e\rangle.\overline{inN2}\langle e\rangle.Mob_{J5} \\ Mob_{J5} \triangleq \overline{outN2}\langle e\rangle.\overline{outR3}\langle e\rangle.Mob_{J1} \end{cases}$$

– Mob_K : la voie d apparait, la voie c apparait, la voie d disparaît, la voie e apparaît, la voie c disparaît, la voie e disparaît puis le système n'évolue plus (ce qui est modélise par le processus inactif 0) :

$$\begin{cases} Mob_{K1} \triangleq \overline{inR1}\langle d\rangle.\overline{inN2}\langle d\rangle.Mob_{K2} \\ Mob_{K2} \triangleq \overline{inR2}\langle c\rangle.\overline{inN1}\langle c\rangle.Mob_{K3} \\ Mob_{K3} \triangleq \overline{outN2}\langle d\rangle.\overline{outR1}\langle d\rangle.Mob_{K4} \\ Mob_{K4} \triangleq \overline{inR3}\langle e\rangle.\overline{inN2}\langle e\rangle.Mob_{K5} \\ Mob_{K5} \triangleq \overline{outN1}\langle c\rangle.\overline{outR2}\langle c\rangle.Mob_{K6} \\ Mob_{K6} \triangleq \overline{outN2}\langle e\rangle.\overline{outR3}\langle e\rangle.0 \end{cases}$$

N.B. Ici, nous avons modélisé un broadcast entre N_i et (R_1, R_2), par deux transmissions unicast entre ce nœud et chacun de ces routeurs (ce sont les points 1 et 2 de la figure 4.4). Ceci est du au fait que le π-calcul, que le π_{stoc} étend, est plus conçu comme un modèle de calculabilité, qu'un langage de modélisation des réseaux mobiles [MPW92].

1.1 Ajout de masques aux commandes PRISM

Le π_{stoc} figuré à figure 4.4 est alors traduit en langage PRISM en appliquant les règles présentées au chapitre 3 (sans considérer le temps qui n'intervient pas dans ce cas). Cependant, étant donné que nous pouvons utiliser plusieurs modèles de mobilité de nœuds, pour le même modèle de communication (un seul à la fois), les commandes PRISM qui apparaissent dans le modèle de communication et qui sont synchronisées avec les commandes d'un modèle de mobilité non utilisé, doivent être masquées. Le masquage est réalisé en utilisant un nombre entier pour renforcer la garde de ces commandes, afin de les autoriser ou, au contraire, de les inhiber. Par exemple, dans le code PRISM du processus N_1 donné ci-dessous, la commande 3 est autorisée, quand le modèle de mobilité utilisé initialise le masque à c (qui est un entier) ; dans le cas contraire, la commande est inhibée. De la même manière, la commande 4 est inhibée tant que le masque n'est pas égal à $res - c$, où res est un entier arbitrairement choisi de sorte qu'il est unique dans le modèle. Seul un modèle de mobilité peut changer la valeur du

masque. Les mêmes gardes apparaissent dans le module de $R2$. A noter que la transmission réussi d'une alarme par $N1$ correspond à $s1 = 11$.

```
module N1
  s1 : [1..11] init 1;
  // N1 a 3 trois noms liés : v1a, v1b, x1
  v1a : [0..25] init 0;
  v1b : [0..25] init 0;
  x1 : [0..25] init 0;

  [backoff1]         (s1=1) -> rate_backoff1 : (s1'=2);
  [a_N1_R1_alarm1]   (s1=2) -> rate_a : (s1'=3);
  [a_R1_N1_v3a]      (s1=3) & (v3a=ack1) -> (s1'=11) & (v1a'=v3a);
  [a_R1_N1_v3a]      (s1=3) & (v3a!=ack1) -> (s1'=1) & (v1a'=v3a);
  [a_R1_N1_nack]     (s1=3) -> (s1'=1) & (v1a'=nack);

  [in_N1_Mob_N1c]    (s1=1) & (mask=c) -> (s1'=4) & (x1'=c);    // 3
  [out_N1_Mob_N1c]   (s1=4) & (mask=res-c)-> (s1'=1) & (x1'=c); // 4
  [backoff1]         (s1=4) -> rate_backoff1 : (s1'=5);
  [a_N1_R1_alarm1]   (s1=5) -> rate_a : (s1'=6);
  [a_R1_N1_v3a]      (s1=6) & (v3a=ack1) -> (s1'=7) & (v1a'=v3a);
  [a_R1_N1_v3a]      (s1=6) & (v3a!=ack1) -> (s1'=9) & (v1a'=v3a);
  [c_N1_R2_alarm1]   (s1=7) & (x1=c) -> rate_c : (s1'=8);
  [c_R2_N1_v4b]      (s1=8) & (x1=c) -> (s1'=11) & (v1b'=v4b);
  [c_R2_N1_nack]     (s1=8) & (x1=c) -> (s1'=11) & (v1b'=nack);
  [a_R1_N1_nack]     (s1=6) -> (s1'=9) & (v1a'=nack);
  [c_N1_R2_alarm1]   (s1=9) & (x1=c) -> rate_c : (s1'=10);
  [c_R2_N1_v4b]      (s1=10) & (x1=c) & (v4b=ack1) -> (s1'=11) & (v1b'=v4b);
  [c_R2_N1_v4b]      (s1=10) & (x1=c) & (v4b!=ack1)-> (s1'=4) & (v1b'=v4b);
  [c_R2_N1_nack]     (s1=10) & (x1=c) -> (s1'=4) & (v1b'=nack);

  [end]              (s1=11) -> true; // end
endmodule
```

FIG. 4.5 – Le code PRISM de $N1$

La traduction de Mob_C est donnée à la figure 4.6. Le masque peut prendre des valeurs entre 0 et l'entier le plus haut représentant la disparition d'un canal mobile (ici $res - a$). A la commande 5, le masque, qui était initialement égal à 0, est mis à c pour autoriser la commande 3 du module $N1$. Ceci permet de modéliser l'apparition du canal c entre le routeur $R2$ et le nœud $N1$. A la commande 6, le masque est initialisé à $res - c$ pour autoriser la commande 4, et inhiber la commande 3 dans le module $N1$. A la commande 7, après la disparition du canal c, le masque est réinitialisé. Tous les modèles de mobilité de nœud fonctionnent de la même manière.

Le module correspondant au processus $N2$ est illustré à la figure 4.7. A noter que `s2=18` signifie que $N2$ a envoyé son alarme avec succès.

```
module Mob
  // dans le cas présent : Mob = MobC
  s : [1..4] init 1;
  mask :[0..res-a] init 0;

  [] mask=0 -> (mask'=c);                                    //5
  [in_R2_MobR2c] s=1 -> rate_iR2 :(s'=2);
  [in_N1_MobN1c] s=2 -> rate_iN1 :(s'=3) & (mask'=res-c);   //6
  [out_N1_MobN1c] s=3 -> rate_oN1 :(s'=4);
  [out_R2_MobR2c] s=4 -> rate_oR2 :(s'=1) & (mask'=0);      //7
endmodule
```

FIG. 4.6 – Le code PRISM de Mob_C

1.2 Les paramètres du modèle PRISM

Les actions silencieuses et les canaux de communication utilisées pour des réceptions d'informations sont associés à des taux, que nous devons évaluer. Ainsi, le nombre maximal des tentatives d'une transmission d'alarme est égal à 7 prévu par le protocole IEEE 802.11 (c'est la valeur de `dot11LongRetryLimit`) [IEE07], tandis que cette valeur est égale à 16 pour le protocole IEEE 802.3 (c'est la valeur de `attemptLimit`) [IEE08]. Cependant, pour appeler à l'aide, un utilisateur doit appuyer sur le bouton-poussoir de l'alarme mobile de façon continue pendant au moins 4 secondes [KHH11]. C'est pourquoi nous considérons que le taux moyen d'une transmission d'alarme dans le pire des cas est $\frac{7}{4} = 1.75$ pour les communications sans fil, et $\frac{16}{4} = 4$ pour des communications câblées.

Par ailleurs, nous supposons que le taux moyen du temps de calcul nécessaire au le serveur pour interpréter une alarme qu'il vient de recevoir est égal à $\frac{2}{4} = 0.5$ parce que deux alarmes peuvent être envoyées par deux nœuds au cours d'une période de 4 secondes.

Enfin, nous supposons les taux moyens d'apparitions et de disparitions des canaux de communication dus aux mouvements de nœuds mobiles est égale à 1.4, qui correspond à la vitesse moyenne d'un être humain en mètres par seconde (soit $5km.s^{-1}$). Le tableau 4.1 récapitule tous ces taux.

PARAMÈTRE	TAUX ASSOCIÉ
Voies de communication sans fil (a,b,c,...)	1.75
Voies de communication câblées ($s1se$ et $s2se$)	4
Contention des nœuds mobiles et des routeurs	1.75
Contention des stations de base	4
Temps d'interprétation d'une alarme par le serveur	0.5 (variable)
Mobilité des nœuds	1.4

TAB. 4.1 – Taux associés aux voies pour les réceptions et aux actions silencieuses utilisés dans le modèle PRISM.

1.3 Récompenses dans le modèle

Pour évaluer les durées des transmissions d'une alarme entre deux nœuds voisins, nous devons connaître la longueur des trames utilisées par les protocoles de cette architecture réseau. Elles sont données par la table 4.2, ainsi que les durées correspondantes pour transmettre

```
module N2
 s2 : [1..18] init 1;
 // N2 a trois noms liés : v2a, v2b, x2
 v2a : [0..25] init 0;
 v2b : [0..25] init 0;
 x2 : [0..25] init 0;

 [backoff2]          (s2=1) -> rate_backoff2 : (s2'=2);
 [b_N2_R2_alarm2]    (s2=2) -> rate_b : (s2'=3);
 [b_R2_N2_v4a]       (s2=3) & (v4a=ack2) -> (s2'=18) & (v2a'=v4a);
 [b_R2_N2_v4a]       (s2=3) & (v4a!=ack2) -> (s2'=1) & (v2a'=v4a);
 [b_R2_N2_nack]      (s2=3) -> (s2'=1)&(v2a'=nack);

 [in_N2_Mob_N2_d]    (s2=1) & (mask=d) -> (s2'=4) & (x2'=d);
 [out_N2_Mob_N2_d]   (s2=4) & (mask=res-d) -> (s2'=1) & (x2'=d);
 [backoff2]          (s2=4) -> rate_backoff2 : (s2'=5);
 [b_N2_R2_alarm2]    (s2=5) -> rate_b : (s2'=6);
 [b_R2_N2_v4a]       (s2=6) & (v4a=ack2) -> (s2'=7) & (v2a'=v4a);
 [b_R2_N2_v4a]       (s2=6) & (v4a!=ack2) -> (s2'=9) & (v2a'=v4a);
 [d_N2_R1_alarm2]    (s2=7) & (x2=d) -> rate_d : (s2'=8);
 [d_R1_N2_v3b]       (s2=8) & (x2=d) -> (s2'=18) & (v2b'=v3b);
 [d_R1_N2_nack]      (s2=8) & (x2=d) -> (s2'=18) & (v2b'=nack);
 [b_R2_N2_nack]      (s2=6) -> (s2'=9) & (v2a'=nack);
 [d_N2_R1_alarm2]    (s2=9) & (x2=d) -> rate_d : (s2'=10);
 [d_R1_N2_v3b]       (s2=10) & (x2=d) & (v3b=ack2) -> (s2'=18) & (v2b'=v3b);
 [d_R1_N2_v3b]       (s2=10) & (x2=d) & (v3b!=ack2)-> (s2'=4) & (v2b'=v3b);
 [d_R1_N2_nack]      (s2=10) & (x2=d) -> (s2'=4) & (v2b'=nack);

 [in_N2_Mob_N2e]     (s2=1) & (mask=e) -> (s2'=11) & (x2'=e);
 [out_N2_Mob_N2e]    (s2=10) & (mask=res-e) -> (s2'=1) & (x2'=e);
 [backoff2]          (s2=11) -> rate_backoff : (s2'=12);
 [b_N2_R2_alarm2]    (s2=12) -> rate_b : (s2'=13);
 [b_R2_N2_v4a]       (s2=13) & (v4a=ack2) -> (s2'=14) & (v2a'=v4a);
 [b_R2_N2_v4a]       (s2=13) & (v4a!=ack2)-> (s2'=16) & (v2a'=v4a);
 [e_N2_R3_alarm2]    (s2=14) & (x2=e) -> rate_e : (s2'=15);
 [e_R3_N2_v5a]       (s2=15) & (x2=e) -> (s2'=18) & (v2b'=v5a);
 [b_R2_N2_nack]      (s2=13) -> (s2'=16) & (v2a'=nack);
 [e_N2_R3_alarm2]    (s2=16) & (x2=e) -> rate_e : (s2'=17);
 [e_R3_N2_v5a]       (s2=17) & (x2=e) & (v5a=ack2) -> (s2'=18) & (v2b'=v5a);
 [e_R3_N2_v5a]       (s2=17) & (x2=e) & (v5a!=ack2)-> (s2'=11) & (v2b'=v5a);

 [end]               (s2=18) -> true; // end
endmodule
```

FIG. 4.7 – Le code PRISM de $N2$

ces trames entre deux nœuds voisins, en considérant que le débit des données est égal à $\frac{1}{2}$ Mbps comme cela est précisé dans [KHH11]. Ces durées sont utilisées comme des valeurs des récompenses associées aux transitions du modèle PRISM, afin évaluer le temps nécessaire à la réception de l'accusé de réception correspondant quand une alarme est envoyée. Nous évaluons que le temps nécessaire par le serveur WSN pour interpréter une alarme est égal à t secondes, où $t \in \{0.5; 1; 1.5; 2\}$.

Protocole	Type	Longueur (en octets)	Temps prévu (en sec.)
IEEE 802.11	Alarme	[34 - 2346]	[68μs - 4.692ms]
	ACK	14	28μs
Ethernet	Les deux	[64 - 1518]	[128μs - 3.036ms]

Tab. 4.2 – Longueur des trames MAC et durée nécessaire à leur transmission entre deux nœuds voisins.

Nous distinguons deux cas, appelés Min et Max, qui représentent respectivement les durées minimales (c'est-à-dire, 68 μs, 28 μs et 128 μs) et les durées maximales pour faire la même chose (c'est-à-dire, 4.992 ms, 28 μs, et 3.036 ms). Dans ce paragraphe, on ne donne que les récompenses correspondantes à Min. Pour calculer le temps maximal attendu pour transmettre une alarme au serveur, l'interpréter et recevoir la reconnaissance correspondante, les valeurs Min doivent être remplacées par les valeurs Max.

Les temps de contention sont égaux aux durées DIFS, plus des durées backoff aléatoires. Nous supposons que le DISF est égal à 50 μs. Mais, parce que PRISM ne permet pas de générer des nombres aléatoires qui permettraient de modéliser le backoff, nous supposons qu'il est égal à 0 dans les calculs dont les résultats sont donnés à la figure 4.3. En changeant cette valeur, les résultats n'en sont pas affectés car nous arrondissons les résultats à 3 chiffres après la virgule.

Les récompenses associées au module $N1$ sont données à la figure 4.8, où la fonction PRISM $pow(x, y)$ calcule x à la puissance y. La récompense de transition 9 représente la durée maximale d'attente de réception d'un accusé de réception sans renvoyer l'alarme correspondante (*timeout*). Nous supposons que ce temps est égal à $(28+1)\mu s$ où $28\mu s$ est la durée nécessaire pour recevoir un accusé de réception. Les récompenses de transition 10 et 11 sont en rapport avec l'apparition et la disparition du canal de communication c pour $N1$. Elles sont égales à 0 parce que nous supposons que les actions correspondantes sont instantanées.

```
[backoff1]        true : 0*pow(10,-6) ;   // (8)
[a_N1_R1_alarm1]  true : 68*pow(10,-6) ;
[a_R1_N1_v3a]     true : 28*pow(10,-6) ;  // ACK
[a_R1_N1_nack]    true : 29*pow(10,-6) ;  // (9) (NACK)
[in_N1_Mob_N1c]   true : 0 ;              // (10)
[out_N1_Mob_N1c]  true : 0 ;              // (11)
[c_N1_R2_alarm1]  true : 68*pow(10,-6) ;
[c_R2_N1_v4b]     true : 28*pow(10,-6) ;  // (ACK)
[c_R2_N1_nack]    true : 29*pow(10,-6) ;  // (9) (NACK)
```

Fig. 4.8 – Les récompenses de transitions pour le module $N1$.

La figure 4.9 représente les récompenses de transition des modules $Si1$, $Si2$ et Se qui sont liées aux transmissions de trames Ethernet. La récompense 12 correspond à la non-réception d'un accusé de réception pour le protocole d'Ethernet. Comme dans le cas de IEEE 802.11,

(29μs), nous supposons que la fin d'un timeout est égal à la durée nécessaire pour recevoir un accusé de réception plus une micro-seconde c'est-à-dire à $(128+1)\mu s = 129\mu s$. Les autres récompenses ne sont pas figurées dans ce paragraphe.

```
// Sink Si1
[s1se_Si1_Se_6a]    true : 128*pow(10,-6) ;
[s1se_Se_Si1_ack1]  true : 128*pow(10,-6) ;
[s1se_Se_Si1_ack2]  true : 128*pow(10,-6) ;
[s1se_Se_Si1_nack]  true : 129*pow(10,-6) ;  // (12) (NACK)
[s1se_Si1_Se_u6b]   true : 128*pow(10,-6) ;

// Sink Si2
[s2se_Si2_Se_u7a]   true : 128*pow(10,-6) ;
[s2se_Se_Si2_ack2]  true : 128*pow(10,-6) ;
[s2se_Se_Si2_nack]  true : 129*pow(10,-6) ;  // (12) (NACK)

// Server Se
[processing]        true : t ;               // t ∈ {0.5 ;1 ;1.5 ;2}
```

FIG. 4.9 – Les récompenses de transitions des modules $Si1$, $Si2$, et Se

1.4 Conclusion

La formule en logique temporelle CSL (*Continuous Stochastic Logic*) [ASSB00,BHHK03] :

$$R_{=?}\ [F((s1 = 11)\ \&\ (s2 = 18))]$$

permet de calculer la durée entre l'envoi d'une alarme et la réception de son accusé de réception. Les expériences ont été effectuées en utilisant un ordinateur équipé d'un processeur Véritable-T2130 Intel avec une vitesse de 1.86 GHz, 2 Go de RAM et Linux Fedora 13 pour système d'exploitation. Le tableau 4.3 donne les résultats arrondis à trois chiffres significatifs, en fonction du paramètre t (le temps d'interprétation des alarmes), et des différents cas de mobilité des nœuds mobiles.

Ces résultats montrent que les durées attendues pour des transmissions d'alarme réussies sont proches de $1s$, $2s$, $3s$, et $4s$ lorsque le temps nécessaire pour interpréter une alarme est respectivement de $0.5s$, $1s$, $1.5s$, et $2s$. Dans tous les cas, les différents cas de mobilité des alarmes mobiles ont peu d'impacts sur eux. Ainsi, les durées d'interprétation des alarmes sont prédominantes par rapport à la mobilité de nœuds qui reste négligeable pour la performance du réseau. Les expériences faites en situation réelle à l'hôpital de Tampere en déclenchant 44 alarmes dans les chambres et les couloirs, sans considérer la mobilité des alarmes mobiles, montrent que 95 % des alarmes sont transmises en $2.3s$, et 97 % le sont en $3.2s$ [KHH11]. C'est pourquoi, nous pouvons considérer que nos résultats, en particulier en cas de Mob_0 (aucune mobilité des alarmes mobiles), sont conformes à l'expérimentation pratiquée sur le réseau réel.

Cas de	Modèle global		Temps de transmission (*en secondes*)							
			t=0.5s		t=1s		t=1.5s		t=2s	
mobilité	états	transitions	min	max	min	max	min	max	min	max
Mob_0	125	321	1.001	1.033	2.001	2.033	3.001	3.033	4.001	4.033
Mob_C	1054	2033	1.014	1.046	2.026	2.058	3.039	3.071	4.051	4.083
Mob_D	1054	2033	1.014	1.046	2.026	2.058	3.039	3.071	4.051	4.083
Mob_E	699	1399	1.037	1.071	2.073	2.107	3.109	3.143	4.145	4.179
Mob_F	2162	3927	1.014	1.046	2.026	2.058	3.039	3.071	4.051	4.084
Mob_G	2363	4298	1.025	1.058	2.050	2.082	3.074	3.107	4.098	4.131
Mob_H	1253	2346	1.014	1.046	2.027	2.059	3.040	3.072	4.053	4.085
Mob_I	1926	3735	1.026	1.059	2.051	2.084	3.075	3.108	4.100	4.133
Mob_J	2229	4023	1.025	1.058	2.049	2.081	3.073	3.105	4.096	4.129
Mob_K	2877	5361	1.025	1.058	2.050	2.083	3.074	3.107	4.098	4.131

TAB. 4.3 – Durées attendues des transmissions des alarmes selon différent cas de mobilité des nœuds, et différentes durées nécessaires au serveur WSN pour interpréter une alarme (le paramètre t).

Conclusion

Cette thèse est consacrée aux approches formelles pour l'analyse de la performabilité des systèmes communicants mobiles, appliquées aux réseaux de capteurs sans fil. Plus précisément nous nous sommes intéressés aux techniques formelles de model checking. Ces techniques consistent à modéliser un système en prenant en prenant en compte ses caractéristiques de performance, temporelles, de mobilité, etc., et à spécifier les propriétés que l'on veut vérifier ou évaluer, dans le cas de propriétés quantitatives, en utilisant une logique temporelle. Un outil automatique permet d'évaluer ces propriétés par rapport à ce modèle, et fournit un contre-exemple si une propriété n'est pas vérifiée.

Pour modéliser les systèmes mobiles, nous utilisons un formalisme de haut niveau, le π-calcul [MPW92], qui permet de modéliser la reconfiguration des liens de communication des systèmes en fonction de leurs interactions. C'est cet aspect de la mobilité des systèmes qui nous intéresse, et c'est pourquoi nous avons choisi le π-calcul pour modéliser les systèmes mobiles. Plus précisément, nous nous intéressons à la famille du π-calcul : la sémantique de ce dernier s'exprime par des systèmes à états-transitions (ou structure de Kripke), et ne permet de considérer que l'aspect mobilité, sans pouvoir considérer l'aspect performance, temps réel, et plus généralement les aspects quantitatifs. Il existe des *model checkers* pour analyser les modèles en π-calcul. Par exemple, MWB (*The Mobility Workbench*) [VM94], MMC (*Mobility Model Checker*) [YRS04]. ProVerif [PROa] permet la vérification de modèles cryptographiques en π-calcul appliqué, une variante de l'algèbre de base. Deux autres approches de vérification de modèles spécifiés en π-calcul sont PIPER [CRR02] qui vérifie des modèles en π-calcul typé, et une traduction d'un sous-ensemble du π-calcul en langage PROMELA pour que les modèles correspondants puissent être vérifiés à l'aide de l'outil SPIN [Wu03, SC]. Mais tous ces outils ne considèrent que le π-calcul de base, sans considération pour des éléments quantitatifs.

La première étape de ce travail a été de proposer des extensions du π-calcul permettant de modéliser les éléments quantitatifs des systèmes, et de traduire les modèles correspondants en langage PRISM, puisque nous avons choisi cet outil comme model checker. Catuscia Palamidessi et al. ont proposé des traductions du π-calcul probabiliste et du π-calcul stochastique en PRISM. Nous étendons ces traductions en proposant des versions probabilistes et stochastiques du π-calcul, où les communications peuvent se faire par diffusion, contrairement à la version de base où les communications sont de type point-à-point. La communication par diffusion locale est la communication de base des systèmes sans fil. Par ailleurs, nous proposons des versions temporisées du π-calcul permettant de modélisant les paramètres temporels des systèmes mobiles. Cet aspect n'est pas anodin : beaucoup de systèmes exhibent des comportements temporisés, par exemple les protocoles de communication qui utilisent des *timeout* pour ne pas rester dans une situation de blocage perpétuelle. Les modèles correspondants sont traduits en automates temporisés probabilistes, et exprimés à l'aide du langage PRISM, dont la dernière version supporte ce formalisme.

Ce travail peut être poursuivi de différentes manières. D'abord, bien que la dernière version de PRISM permettre des analyses temporelles, il pourrait être intéressant de prévoir des règles de traduction des modèles mobiles, dans des langages supportés par d'autres model checkers tels que UPPAAL-PRO ou MODEST. Il serait également intéressant de prévoir, lorsque cela est possible, la traduction automatique des modèles exprimés en un formalisme issu du π-calcul, à l'aide d'un langage de haut niveau par exemple Java.

Mais la piste la plus intéressante de poursuite de ces travaux est, sans doute, la traduction de la mobilité spatiale des systèmes, telle que décrite par le calcul des ambients [CG98], en des langages supportés par les model checkers, en particulier PRISM. L'idée consisterait à définir la notion de localité est de définir le mouvement d'un système entre différentes localités. Ce qui permettrait d'analyser d'autres types de propriété de performabilité, par exemple des propriétés de sécurité se rapportant à une autorisation d'accéder à une localité. A cette piste de recherche peut s'ajouter l'utilisabilité de la logique spatiale [CC01, CC04], et non plus seulement des logiques temporelles, pour spécifier des propriétés de performabilité, et les analyser avec des model checkers modernes.

Annexes

Annexe A

Techniques de model checking

La complexité des systèmes explique l'intérêt porté aux méthodes de vérification de leurs exigences de performabilité qui permettent d'accroître la confiance que l'on a de ces systèmes lorsqu'ils sont exploités. Au début des années 1980, Joseph Sifakis et Jean-Pierre Queille en France [QS82] et, de manière indépendante, Edmund M. Clarke et E. Allen Emerson aux Etats-Unis [CE81] proposent le model checking, une méthode de vérification automatique des systèmes concurrents[1]*. Cette méthode consiste à vérifier qu'un modèle fini d'un système satisfait une exigence exprimée sous la forme d'une formule de logique temporelle. De façon schématique, un algorithme de model checking prend en entrée un modèle à états-transitions d'un système, et une formule de logique temporelle, et répond si ce modèle vérifie, ou ne vérifie pas, cette formule. Si la propriété n'est pas vérifiée, un contre-exemple est fourni. Le model checking a plusieurs avantages : son exhaustivité, car tous les états du modèle sont explorés lors de la vérification de la propriété, et son automaticité, car cette vérification est effectuée à l'aide d'un outil. Paradoxalement, l'exhaustivité du model checking est aussi à l'origine de son inconvénient majeur, connu sous le nom d'explosion combinatoire des états, puisque le nombre d'états du modèle à vérifier peut être très grand, équivalent, selon Joseph Sifakis, aux dimensions de l'Univers. Cette inconvénient constitue la limite de l'applicabilité de cette méthode formelle de vérification. Néanmoins, les améliorations des algorithmes de vérification, et les progrès des capacités de calculs et de traitements des machines, ont permis que le model checking s'impose comme une méthode de vérification dans de nombreux domaines variés : pour vérifier des circuits VLSI, des protocoles de communication, des pilotes de périphériques, des systèmes embarqués temps réel, ou encore des protocoles de sécurité. Divers types de modèles peuvent être utilisés tels que des modèles stochastiques, des modèles temporisés, des modèles hybrides, etc., qui conduisent à différentes techniques de vérification. C'est pourquoi nous parlons « des techniques » de model checking au pluriel. Cette annexe les présente. Pour plus d'informations sur le model checking, outre les articles mentionnés, le lecteur peut consulter [BBF+01].*

1 Introduction

La vérification des exigences de performabilité d'un système par model checking se déroule en trois phases. Tout d'abord, il s'agit de modéliser le système à étudier (M), et à traduire ces exigences en formule d'une logique temporelle adéquate ($\{\Phi_i\}_{i\in I}$). Il existe plusieurs logiques temporelles, que nous présentons dans cette annexe, permettant de spécifier plusieurs types d'exigences. La deuxième étape consiste à vérifier à l'aide d'un outil que le modèle M

[1] Joseph Sifakis, Edmund M. Clarke, et E. Allen Emerson ont reçu le prix Turing en 2007 pour *leur rôle dans le développement du model checking en une technologie de vérification très efficace largement adoptée par les industries du matériel et du logiciel* (citation officielle) [CES09].

satisfait, ou non, les formules $\{\Phi_i\}_{i\in I} : \forall i \in I,\ M \models \Phi_i$. Dans le cas où une de ces formules n'est pas vérifiée, un contre-exemple qui illustre l'origine du problème est automatiquement fourni. La dernière étape consiste à analyser les résultats obtenus : si toutes les formules sont vérifiées, alors le modèle M du système respecte les exigences de performabilité, en matière de sûreté de fonctionnement et de performance. Dans ce cas, la vérification est terminée so l'on est assuré que le modèle est une abstraction correcte du système réel. Par contre, si au moins une des formules n'est pas vérifiée, alors il faut rejouer le contre-exemple sur le système réel (si c'est possible !). Deux cas se présentent : soit il s'agit d'un véritable défaut du système réel qui doit être corrigé, soit le défaut n'apparaît pas sur le système réel, et dans ce cas il s'agit d'un problème de modélisation. Dans ce dernier cas, il faut améliorer le modèle, par exemple en le raffinant, avant de recommencer la procédure de vérification. C'est la procédure que nous avons déjà schématisée au paragraphe 1.3.1.1.

Les exigences de performabilité d'un système que l'on cherche à vérifier concernent habituellement :

- L'*accessibilité* d'une certaine configuration du système, c'est-à-dire que l'état correspondant dans son modèle peut être atteint.
- L'*invariance* de propriété, c'est-à-dire que tous les états du modèle étudié satisfont cette propriété.
- La *sûreté*, c'est-à-dire que quelque chose de mauvais pour le système ne peut pas se produire.
- La *vivacité*, c'est-à-dire que quelque chose de bon pour le système finira par arriver.
- L'*équité*, c'est-à-dire que quelque chose se répétera infiniment souvent. On distingue l'*équité forte* (si un processus demande continuellement son exécution, il finira par l'avoir), de l'*équité forte* (si un processus demande infiniment souvent son exécution, il finira par l'avoir).
- L'*absence de blocage*, c'est-à-dire que le système ne peut pas se trouver dans une configuration à partir de laquelle il ne peut plus évoluer.
- L'*équivalence comportementale* entre au moins deux systèmes qui consiste à vérifier qu'ils se comportent de la même manière ou non.

2 Le model checking classique

2.1 Structure de Kripke

Le système à analyser est modélisé sous la forme d'une machine à états-transitions étiquetés, finis et non-déterministe, encore appelée *une structure de Kripke*[2]. Chaque état de la machine représente une configuration du système réel, alors qu'un chemin dans la machine représente un comportement de ce dernier.

De façon formelle, une structure de Kripke $\mathcal{K} = (Q, q_0, \rightarrow, P, \mathcal{E})$ est un n-uplet où Q est un ensemble d'états (encore appelés des configurations) ; $q_0 \in Q$ est l'état initial ; $\rightarrow \subseteq Q \times Q$ est la relation de transition entre états ; P est un ensemble de propositions atomiques ; et $\mathcal{E} : Q \rightarrow 2^P$ est une fonction d'étiquetage des états avec des propositions atomiques qui sont des éléments de l'ensemble P.

[2]D'après Saul Aaron Kripke, un logicien et philosophe américain né en 1940 qui a proposé ces structures.

Nous avons dit que les exigences à vérifier sont traduites en formules de logique temporelle. Ces formules sont interprétées sur la structure de Kripke qui modélise le système auquel on s'intéresse. La notation $K, q_0 \models f$ signifie que « dans le modèle K, à l'état q_0, la formule de logique temporelle f est vraie ». Lorsque q_0 est compris, on écrira $K \models f$. Par exemple, on écrira $K, q_0 \models AFp$ si pour tous les chemins $x = q_0, q_1, q_2, \ldots$ de K, $\exists i \geqslant 0, p \in \mathcal{E}(q_i)$.

2.2 Les logiques temporelles

Les *logiques temporelles* sont des formalismes permettant de décrire l'évolution comportementale d'un système au cours du temps. Ces logiques sont plus expressives que le langage naturel, trop ambigu, et plus concises que des formalismes plus généralistes, par exemple la logique du premier ordre.

Logique LTL. L'idée d'utiliser une logique temporelle pour raisonner sur l'évolution comportementale des programmes concurrents est due à Amir Pnueli[3] dans *The temporal logic of programs* [Pnu77]. Il utilisa une logique temporelle comprenant les opérateurs temporels de base **F** (*un jour*) et **G** (*toujours*). Enrichie avec **X** (*la prochaine fois*), et **U** (*jusqu'à*), elle est maintenant connue sous le nom de LTL (*Linear Time Logic*). Les formules de LTL sont :

$$\Psi ::= true \mid p \mid \neg\Psi \mid \Psi \wedge \Psi \mid \Psi \vee \Psi \mid X\Psi \mid F\Psi \mid G\Psi \mid \Psi \; U \; \Psi$$

où p est une proposition atomique. Le domaine d'interprétation est un ensemble de couples M, σ associant un modèle M du système (c'est-à-dire une structure de Kripke), et un chemin σ dans cette structure. On appelle un chemin dans une structure de Kripke, une séquence infinie $\sigma = q_0 q_1 q_2 \ldots$ d'états $q \in Q$. On note σ^i le chemin extrait de σ à partir du ième état. Par exemple $\sigma^2 = q_2 q_3 \ldots$. On note $\sigma(k)$ le k-ième état de σ. La relation de satisfaction $M, \sigma \models \Phi$ est définie par :

- $M, \sigma \models true$ (toujours) ;
- $M, \sigma \models p \Leftrightarrow p \in l(\sigma(0))$ (p est vraie dans l'état courant) ;
- $M, \sigma \models \neg\Psi \Leftrightarrow M, \sigma \models \Psi$ est faux ;
- $M, \sigma \models \Psi_1 \vee \Psi_2 \Leftrightarrow M, \sigma \models \Psi_1$ ou $M, \sigma \models \Psi_2$;
- $M, \sigma \models \Psi_1 \wedge \Psi_2 \Leftrightarrow M, \sigma \models \Psi_1$ et $M, \sigma \models \Psi_2$;
- $M, \sigma \models X\Psi \Leftrightarrow M, \sigma^1 \models \Psi$ (cf. figure A.1(a)) ;
- $M, \sigma \models F\Psi \Leftrightarrow \exists\, k \geqslant 0 \; M, \sigma^k \models \Psi$ (cf. figure A.1(b)) ;
- $M, \sigma \models G\Psi \Leftrightarrow \forall\, k \geqslant 0 \; M, \sigma^k \models \Psi$ (cf. figure A.1(c)) ;
- $M, \sigma \models \Psi_1 U \Psi_2 \Leftrightarrow \exists k \geqslant 0 \; M, \sigma^k \models \Psi_2$ et $\forall j, 0 \leqslant j < k, M, \sigma^j \models \Psi_1$ (cf. figure A.1(d)).

Logique CTL*. On distingue dans CTL* des formules d'états (Ψ_s), interprétées sur les états d'une structure de Kripke, et des formules de chemins (Ψ_p), interprétées sur ses chemins. Les formules CTL* sont alors définies par la grammaire suivante :

$$\Psi_s ::= true \mid p \mid \neg\Psi_s \mid \Psi_s \vee \Psi_s \mid \Psi_s \wedge \Psi_s \mid A\Psi_p \mid E\Psi_p$$
$$\Psi_p ::= \Psi_s \mid \neg\Psi_p \mid \Psi_p \vee \Psi_p \mid \Psi_p \wedge \Psi_p \mid F\Psi_p \mid G\Psi_p \mid \Psi_p U \Psi_p$$

[3] Amir Pnueli (1941-2009) est un informaticien israélien, lauréat du prix Turing en 1996 pour ses travaux sur l'introduction de la logique temporelle en informatique, et pour sa contribution « exceptionnelle » à la vérification des systèmes.

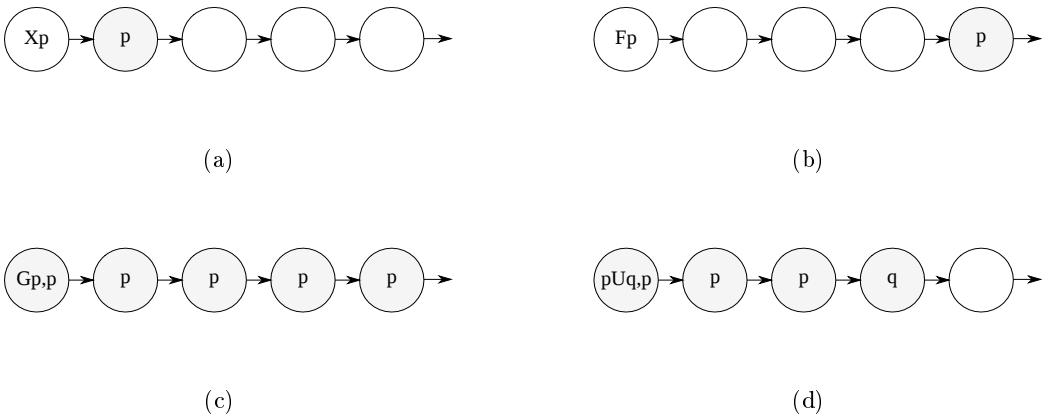


FIG. A.1 – Illustration des opérateurs de LTL.

où $p \in PA$ est une proposition atomique. Le domaine d'interprétation est un couple (M, s) associant une structure de Kripke et un état (pour les formules d'états), et un couple (M, σ) associant une structure de Kripke et un chemin (pour les formules de chemins). On définit $M, s \models \Psi$ par :

1. Les formules d'états (relation $\models_s$) :
   - $M, s \models_s p \Leftrightarrow p \in l(s)$ (p est vraie dans l'état courant) ;
   - $M, s \models_s \neg\Psi \Leftrightarrow M, s \models_s \Psi$ est faux ;
   - $M, s \models_s \Psi_1 \vee \Psi_2 \Leftrightarrow M, s \models_s \Psi_1$ ou $M, s \models_s \Psi_2$;
   - $M, s \models_s \Psi_1 \wedge \Psi_2 \Leftrightarrow M, s \models_s \Psi_1$ et $M, s \models_s \Psi_2$;
   - $M, s \models_s Af \Leftrightarrow$ tous les chemins σ partant de s vérifient $M, \sigma \models_p f$;
   - $M, s \models_s Ef \Leftrightarrow$ il existe un chemin σ partant de s qui vérifie $M, \sigma \models_p f$.
2. Les formules de chemins (relation $\models_p$) :
   - $M, \sigma \models_p f \Leftrightarrow M, \sigma(0) \models_s f$ (f étant une formule d'états) ;
   - $M, \sigma \models_p \neg\Psi \Leftrightarrow M, \sigma \models_p \Psi$;
   - $M, \sigma \models_p \Psi_1 \wedge \Psi_2 \Leftrightarrow M, \sigma \models_p \Psi_1$ ou $M, \sigma \models_p \Psi_2$;
   - $M, \sigma \models_p \Psi_1 \vee \Psi_2 \Leftrightarrow M, \sigma \models_p \Psi_1$ et $M, \sigma \models_p \Psi_2$,
   - $M, \sigma \models_p X\Psi \Leftrightarrow M, \sigma^1 \models_p \Psi$;
   - $M, \sigma \models_p F\Psi \Leftrightarrow \exists k \geqslant 0,\ M, \sigma^k \models_p \Psi$;
   - $M, \sigma \models_p G\Psi \Leftrightarrow \forall k \geqslant 0,\ M, \sigma^k \models_p \Psi$;
   - $M, \sigma \models_p \Psi_1 U \Psi_2 \Leftrightarrow \exists k \geqslant 0\ M, \sigma k \models_p \Psi_2\ \forall j, 0 \leqslant j < k,\ M, \sigma^j \models_p \Psi_1$.

Logique CTL. La logique CTL (*Computation Tree Logic*) est une restriction de CTL* dans laquelle les connecteurs temporels X, F, G et U doivent être directement précédés d'un quantificateur de chemin A ou E. Une formule CTL s'obtient à partir de $\neg$,$\vee$,$\wedge$ et des huit opérateurs AX, EX, AF, EF, AG, EG, AU et EU. Par exemple :

- EFp : il existe un chemin partant de s qui atteint p (figure A.2(c)).

- AFp : tous les chemins partant de s finissent par atteindre p (figure A.2(a)).
- EGp : il existe un chemin partant de s tel que p est vrai tout au long du chemin, y compris en s (figure A.2(d)).
- AGp : p est toujours vrai en partant de s, y compris en s (figure A.2(b)).

L'influence des logiques LTL, CTL, CTL* a été très importante, et elles ont engendré de nombreuses extensions utilisées dans l'industrie. C'est le cas par exemple d'IBM Sugar basé sur CTL, Intel ForSpec basé sur LTL, et PSL (norme IEEE-1850) qui comprend des caractéristiques de CTL*.

2.3 L'explosion combinatoire des états et ses solutions

L'explosion combinatoire des états est le problème principal du model checking. Le nombre d'états décrivant les comportement d'un système composés d'un grand nombre de composants concurrents peut être immensément grand. Par exemple, un système composé de n processus asynchrone, chacun d'eux ayant des m états, peut avoir m^n états. Le système à états-transitions modélisant un compteur à n bits a 2^n états. Ce problème est inévitable. Cependant les techniques suivantes ont été mises au point pour lutter contre lui.

- Le *model checking symbolique*, proposé par Ken McMillan [McM93], utilise une représentation symbolique des structures de Kripke à base de *diagrammes ordonnés de décision binaire* (*OBDDs* pour *ordered binary decision diagrams*). Aujourd'hui, cette technique permet de vérifier des systèmes ayant plus de 10^{120} états [CES09].
- La *réduction d'ordre partiel* exploite l'indépendance entre des composants asynchrones. Les composants A et B sont indépendants, si l'exécution de A avant B conduit le système global dans le même état que l'exécution de B avant A. Cette technique permet de réduire le nombre d'entrelacements possibles entre deux composants A et B indépendants, en évitant d'explorer certains chemins dans la structure de Kripke modélisant un système. Le model checking avec réduction d'ordre partiel est adaptée à la vérification de systèmes logiciels.
- Le *model checking borné* consiste à vérifier s'il existe dans la structure de Kripke, un contre-exemple violant une propriété à vérifier de longueur k ($k \geqslant 1$). Ce qui signifie qu'un algorithme de model checking borné vérifie s'il existe un chemin de longueur k se terminant par un cycle dans lequel tous les états sont étiquetés par $\neg p$, p étant la propriété à vérifier. Le processus peut être répété avec des valeurs plus grandes de k, jusqu'à ce que toutes les violations possibles de p aient été écartées.
- Le *raffinement d'abstractions* vérifie des propriétés sur un modèle en commençant par le simplifier. Le modèle simplifié est obtenu à partir du modèle original par raffinement. Au mieux, le modèle raffiné doit vérifier les propriétés déjà vérifiées sur le modèle plus abstrait. Un exemple d'abstraction d'un programme consiste à ignorer certaines valeurs des variables, pour ne considérer que les autres, avec les flux de contrôle du programme. Une telle abstraction peut être suffisante pour vérifier une propriété d'exclusion mutuelle. La technique CEGAR (*Counterexample Guided Abstraction Refinement*) [CGJ+03] commence par vérifier une propriété sur un modèle grossier, puis raffine celui-ci de façon itérative. Quand un contre-exemple est détecté, sa faisabilité est analysée sur le système réel, ou à défaut, est simulée, pour vérifier qu'il s'agit d'une véritable erreur de conception, ou bien d'un défaut de modélisation. S'il s'agit d'une erreur de conception, alors elle doit être corrigée. Sinon le contre-exemple est utilisé pour raffiner le modèle abstrait, avant de recommencer le processus de vérification en utilisant le modèle raffiné.

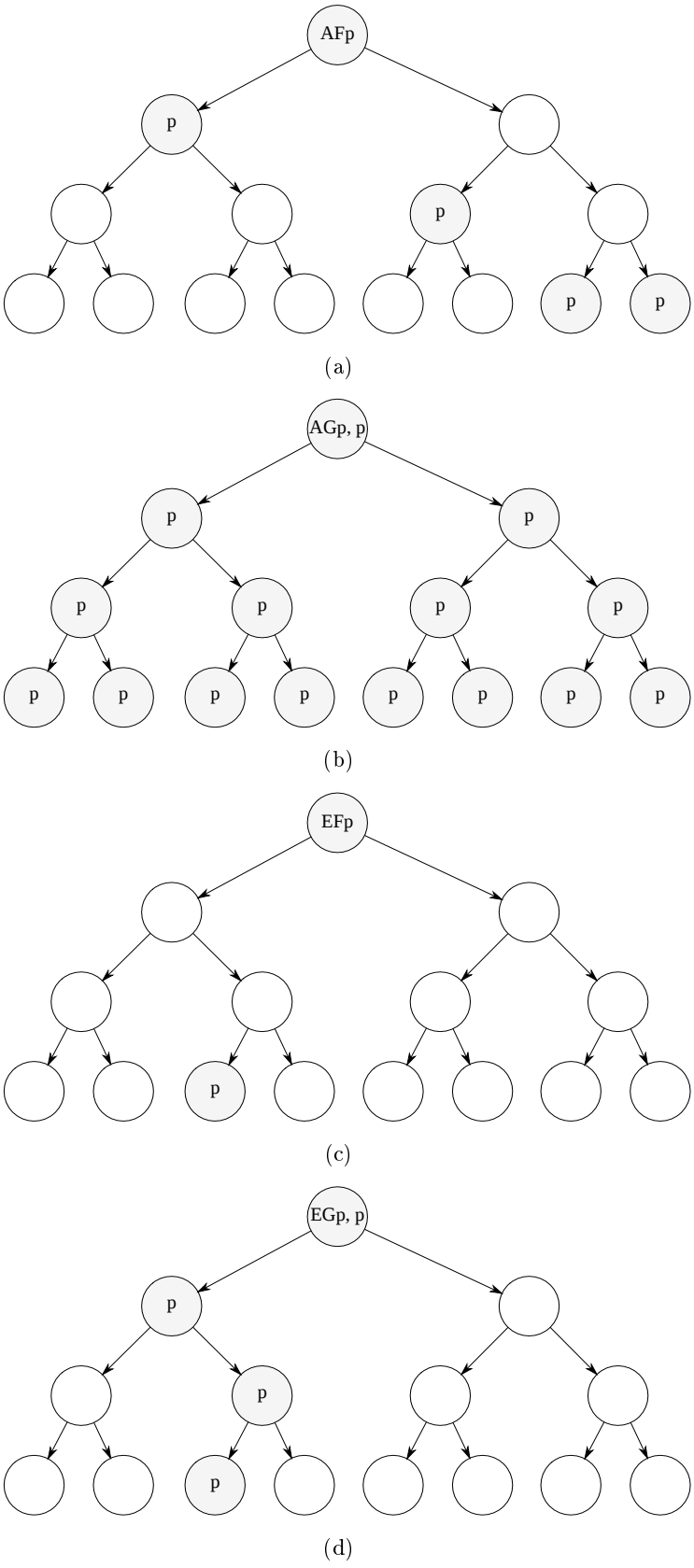


FIG. A.2 – Illustration de quelques formules de CTL.

3 Le model checking stochastique

Si le système à analyser présente un caractère aléatoire que l'on souhaite prendre en compte, ce système sera modélisé sous la forme d'un *processus stochastique*. Les processus stochastiques dont les comportements sont discrets sont souvent représentés par des *chaînes de Markov à temps discrets* (*DTMCs* pour *discrete time Markov chains*), ou des processus de décision markoviens (*MDPs* pour *Markov decision processes*), si l'on souhaite les analyser selon un temps discret, ou encore des chaînes de Markov à temps continu (*CTMCs* pour *continuous time Markov chains*) si l'on souhaite les analyser selon un temps continu. Il est alors possible d'évaluer la probabilité de satisfaction d'une exigence. Dans ce cas, une logique temporelle permettant d'exprimer des probabilités est nécessaire. Cette annexe présente la vérification d'un système stochastique modélisé par une chaîne de Markov, à temps discret ou à temps continu, ou par un processus de décision markovien. Ces formalismes sont décrits dans ce paragraphe comme des extensions probabilistes ou stochastiques des structures de Kripke.

3.1 Notion de processus stochastique

Un *processus stochastique* $(X_t)_{t \in T}$ est une famille de variables aléatoires, indexée sur le temps T. Si le temps est discret ($T \subset \mathbb{N}$), il s'agit d'un *processus à temps discret*, alors que lorsque le temps est continu ($T \subset \mathbb{R}$), on parle de *processus à temps continu*. Pour t fixé, X_t est une variable aléatoire prenant ces valeurs dans un certain espace d'états $\mathcal{E}$ suivant des distributions qui peuvent dépendre de t. L'ensemble des états possibles d'un processus stochastique peut, lui-même, être discret ou continu.

Si l'on tire aléatoirement une valeur y_t de Y_t pour chaque valeur de t, la séquence de valeurs obtenues est appelée une *trajectoire* du processus. Cela représente une succession d'états que le système peut prendre au cours du temps. On peut s'intéresser au *comportement transitoire* d'un processus (*transient behaviour*), c'est-à-dire à l'évolution des fonctions de répartition $F_{Y_t}(x)$ au cours du temps t. On peut également s'intéresser au *comportement stationnaire* d'un processus (c'est-à-dire à l'équilibre, *steady-state behavior* en anglais), s'il existe, qui est obtenu si $F_{Y_t}(x)$ converge vers une distribution unique $F_Y(x)$ indépendante du temps lorsque t devient infini.

Parmi les processus stochastiques, certains respectent la *propriété sans mémoire* qui énonce que la distribution des probabilités de l'état futur du processus, étant donné son état courant et ses états passés, ne dépend que de l'état courant et non pas des états passés. Soit :

$$Pr[X_{t+1} = s_{k+1} | X_t = s_k, ..., X_0 = s_0] = Pr[X_{t+1} = s_{k+1} | X_t = s_k]$$

Ces processus sont appelés des *processus de Markov*[4]. Les processus de Markov à espace d'états discrets sont appelés des *chaînes de Markov*, et font l'objet du paragraphe suivant. Pour d'autres exemples de processus stochastiques, le lecteur peut se référer à [].

3.2 Chaînes de Markov à temps discret

Les *chaînes de Markov à temps discret* (DTMCs) modélisent des systèmes à événements discrets, dont les comportements sont déterministes et probabilistes, et sont observés à des

[4] D'après le mathématicien russe Andreï Andreïevitch Markov (1856 - 1922), qui étudia en 1913 la succession des lettres dans le roman *Eugène Onéguine* d'Alexandre Pouchkine. Markov nota que les lettres utilisées, qui se répartissent selon les statistiques spécifiques de l'alphabet russe, suivent des contraintes très précises puisque chaque lettre dépend étroitement de la précédente.

intervalles temporels discrets. De manière informelle, une DTMC est une structure de Kripke étendue, où les transitions entre états sont pondérées par leur probabilité d'occurrence respective. Chaque état représente une configuration possible du système modélisé, et les transitions entre les états représentent les évolutions possibles de la configuration de ce système, c'est-à-dire son comportement. Sachant que chaque transition est pondérée par une probabilité, il est possible de calculer la probabilité d'occurrence d'un comportement particulier. Parmi les systèmes modélisables par des DTMCs, nous trouvons les algorithmes stochastiques dont les comportements sont déterministes, la transmission d'un message sur une voie de communication non fiable dont la probabilité de panne est connue, etc.

3.2.1 Définition

Définition A.1. Une DTMC est un quintuplet $D = (E, e_{init}, P, Prop, L)$ où E est un ensemble fini d'états (appelé l'« espace des états »), $e_{init} \in E$ est l'état initial, $P : E \times E \to [0, 1]$ est la matrice des probabilités des transitions telle que, pour tout état e de E, $\sum_{e' \in E} P(e, e') = 1$[5], $Prop$ est un ensemble de propositions atomiques, et $L : E \to 2^{Prop}$ est la fonction partielle[6] d'étiquetage des états qui associe à chaque état e de E un ensemble $L(e)$ de propositions atomiques appartenant à l'ensemble $Prop$.

Soit une DTMC $D = (E, e_{init}, P, Prop, L)$. La probabilité de transiter de l'état e à l'état e' en une unité de temps est donné par $P(e, e')$. Un état e est *absorbant* si et seulement si $P(e, e) = 1$ et $\forall e' \neq e, P(e, e') = 0$. La matrice P étant stochastique, chaque état a au moins une transition de sortie et par suite, il ne peut pas y avoir d'états bloquants dans une DTMC. Un état absorbant n'est pas bloquant. Enfin, une DTMC est *à temps homogène* si les probabilités de ses transitions sont indépendantes du temps.

Exemple (d'après [Kwi07]). Soit le protocole de communication à contention dont la spécification est la suivante. Un nœud émetteur qui veut envoyer un message commence par attendre pendant une unité de temps avant d'essayer de transmettre ses données. La probabilité que le canal de transmission soit occupé est égale à 0,01. Dans ce cas, l'émetteur attend un temps supplémentaire avant de réessayer. Si le canal n'est pas occupé, il envoie son message. La probabilité que cet envoi soit un succès est égale à 0,98. Dans ce cas, le nœud émetteur a terminé sa transmission. La probabilité d'une collision de transmission est égale à 0,01. Dans ce cas, l'émetteur doit recommencer. La DTMC qui modélise ce système est représentée par la figure A.4. L'état e_3 est absorbant ; la transition entre e_3 et lui-même est nécessaire car il n'y a pas d'état bloquant dans une DTMC.

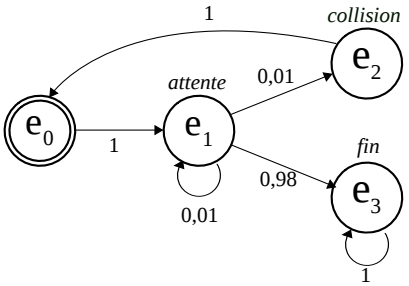


FIG. A.3 – Modélisation d'un protocole de communication déterministe par une DTMC.

[5] Du fait de cette propriété, on dit que P est une matrice *stochastique*.

[6] Une *fonction partielle* est une fonction dont certains éléments de l'ensemble de départ n'ont pas d'image. Dans le cas présent, cela signifie que certains états d'une DTMC ne sont pas étiqueté.

3.2.2 Espace probabilisé sur les chemins d'une DTMC.

Un *chemin* dans une DTMC $D = (E, e_{init}, P, Prop, L)$ est une séquence non vide d'états successifs, finie ou infinie, $\omega = e_0 e_1 e_2 e_3 \ldots$ telle que $\forall i \in \mathbb{N}, P(e_i, e_{i+1}) > 0$[7]. Un chemin dans une DTMC représente un comportement particulier du système modélisé.

Le ième état du chemin ω est noté $\omega(i)$. Le dernier état d'un chemin fini ω_{fin} est noté par $dernier(\omega_{fin})$ et sa longueur par $|\omega_{fin}|$. L'ensemble de tous les chemins infinis et l'ensemble de tous les chemins finis de la DTMC D qui commencent dans l'état e, sont respectivement notés $Chemins_D(e)$ et $Chemins_D^{fin}(e)$. Dans la suite de ce paragraphe, si l'on ne précise pas qu'un chemin est fini alors il sera considéré comme infini.

Un chemin fini ω_{fin} de longueur n est un *préfixe* du chemin ω si $\forall i \in [0, \ldots, n], \omega_{fin}(i) = \omega(i)$. Le *cylindre* (encore appelé le *cône*) $C(\omega_{fin}) \subseteq Chemins_D(e)$ est l'ensemble des chemins infinis de D qui ont le même préfixe ω_{fin} : $C(\omega_{fin}) = \{\omega \in Chemins_D(e) \mid \omega_{fin}$ est un préfixe de $\omega\}$.

Pour raisonner quantitativement sur un système modélisé par une DTMC D, il est nécessaire de déterminer la probabilité d'occurrence des comportements qui intéressent l'analyse, c'est-à-dire la probabilité des chemins correspondants de D. Il est donc nécessaire de définir un espace probabilisé sur l'ensemble des chemins d'une DTMC. De façon intuitive, pour une DTMC donnée, l'*univers* Ω est l'ensemble de tous les chemins infinis qui commencent dans un état e particulier de cette DTMC. Un *événement* est un ensemble de chemins infinis qui commencent en e. Un événement élémentaire est un cylindre commençant en e. On définit alors une *mesure de probabilité Prob* sur l'ensemble des événements qui commence en e ($Prob : \Sigma_{Chemins_D(e)} \rightarrow [0, 1]$), et par extension la probabilité de chaque chemin fini ω_{fin} dont le premier état est e, qui est mesurable grâce à la matrice des probabilités des transitions de la DTMC.

Définition A.2. L'espace probabilisé $(\Omega, \Sigma_{Chemins(e)}, \mathcal{P})$ est défini de la façon suivante :
- $\Omega = Chemins_D(e)$;
- $\Sigma_{Chemins(e)} = C(\omega_{fin})$;
- $Pr_e : \Sigma_{Chemins(e)} \rightarrow [0, 1]$ définit par extension la probabilité $\mathcal{P}_e(\omega)$ pour chaque chemin fini $\omega = ee_1 \ldots e_n$ de la façon suivante :

$$\mathcal{P}_e(\omega_{fin}) = \begin{cases} 1 & si\ |\omega_{fin}| = 1\ (soit\ \omega_{fin} = e) \\ P(e, e_1) \times P(e_1, e_2) \times \ldots P(e_{n-1}, e_n) & sinon. \end{cases}$$

Exemple (suite). Dans la DTMC précédente, on cherche à évaluer la probabilité d'un envoi de message réussi, quel que soit le temps d'attente dû à l'occupation du canal de transmission. Les chemins qui nous intéressent sont les chemins qui commencent en e_0, et qui se terminent en e_3 : $C(e_0e_1e_3) \cup C(e_0e_1e_1e_3) \cup C(e_0e_1e_1e_1e_3) \cup \ldots$. La probabilité correspondante est :
$Pr_{e_0}(C(e_0e_1e_3) \cup C(e_0e_1e_1e_3) \cup C(e_0e_1e_1e_1e_3) \cup \ldots)$
$= P_{e_0}(e_0e_1e_3) + P_{e_0}(e_0e_1e_1e_3) + P_{e_0}(e_0e_1e_1e_1e_3) + \ldots$
$= 1 \times 0,98 + 1 \times 0,01 \times 0,98 + 1 \times 0,01 \times 0,01 \times 0,98 + \ldots$
$\simeq 0,989899$
$= 98/99$

[7] L'inégalité est stricte car la transition entre e_i et e_{i+1} doit exister, et donc la probabilité correspondante doit être non nulle

3.2.3 Extensions des DTMCs par des récompenses ou des coûts.

Il est possible d'étendre les DTMCs par des informations numériques associées à leurs états et/ou à leurs transitions. Ces informations se rapportent à des ressources utilisées par le système modélisé, telles que, par exemple, une quantité d'énergie, la taille d'une file de messages, le nombre de messages reçus avec succès, le nombre de messages perdus, des délais, etc. Ce sont des *récompenses* ou des *coûts*. Du point de vue mathématique, on ne distingue pas ces deux notions : leur seule différence est qu'un utilisateur cherche généralement à minimiser les coûts, et à maximiser les récompenses. Dans la suite de ce paragraphe, les définitions données s'appliquent à des récompenses, mais elles valent aussi pour des coûts.

De façon formelle, pour une DTMC $D = (E, e_{init}, P, Prop, L)$, une *récompense* est une couple de fonctions (ρ, ι) où :

- $\rho : E \to \mathbb{R}_{\geq 0}$ est la fonction de récompenses des états qui associe à chaque état de E un nombre réel positif ou nul ; et
- $\iota : S \times S \to \mathbb{R}_{\geq 0}$ est la fonction de récompense des transitions qui associe à chaque transition entre états, un nombre réel positif ou nul.

La récompense d'état $\rho(e)$ est la récompense acquise par le système lorsqu'il est dans l'état e. La récompense de transition $\iota(e, e')$ est acquise à chaque fois qu'une transition entre les états e et e' est réalisée.

3.2.4 Les limites des DTMCs pour modéliser les systèmes réels.

Les DTMCs modélisent des systèmes dont les comportements sont déterministes, et les probabilités de transitions entre les différents états sont connues[8]. Par ailleurs, la somme des probabilités de toutes les transitions issues d'un état est égale à 1. En fait, les systèmes réels ne sont pas toujours aussi bien connus, et ils peuvent exhiber des comportements non déterministes. Par exemple, deux composants concurrents qui opèrent en parallèle, un système qui dépend de ses interactions avec son environnement dont le comportement est inconnu, l'absence de connaissance de certains paramètres d'un système étudié, ou encore l'abstraction d'un système complexe par un système plus simple sont des exemples de systèmes qui ne sont donc pas modélisables à l'aide des DTMCs. L'impossibilité de décrire parfaitement les comportements des systèmes stochastiques à temps discret demande l'introduction d'un modèle adapté : celui des *processus de décisions de Markov*. Nous décrivons ces modèles dans le paragraphe suivant.

3.3 Processus de décision de Markov

Les *processus de décision de Markov* (MDPs), similaires aux automates probabilistes indéterministes, modélisent des systèmes stochastiques dont les comportements non déterministes sont observés à des échéances temporelles discrètes. A partir de chaque état d'un MDP, un choix non-déterministe se présente entre plusieurs distributions de probabilités discrètes sur des états successeurs. La somme des probabilités des transitions issues d'un état n'est pas forcément égale à 1, comme c'est le cas dans une DTMC. De la même façon que les DTMCs, les MDPs peuvent être étendus par des récompenses ou des coûts.

[8] Dans [KNP10], Marta Kwiatkowska, Gethin Norman et David Parker décrivent les DTMCs comme des modèles « complètement probabilistes » (*fully probabilistic*).

3.3.1 Définition

Définition A.3. Un MDP est un sextuplet $M = (E, e_{init}, Act, Steps, Prop, L)$, où E est un ensemble fini d'états (l'« espace des états »), $e_{init} \in E$ est l'état initial, Act est un ensemble d'étiquettes d'action, $Steps : E \to 2^{Act \times Dist(E)}$ est la fonction des probabilités des transitions où $Dist(E)$ dénote l'ensemble des distributions probabilistes discrètes sur E, $Prop$ est un ensemble de propositions atomiques, et $L : E \to 2^{Prop}$ est une fonction d'étiquetage des états qui associe à chaque état e de E, un ensemble $L(e)$ de propositions atomiques appartenant à $Prop$.

Exemple. La spécification du protocole de communication de l'exemple 3.2.1 est modifiée de la façon suivante. Après une étape temporelle, le nœud émetteur commence le processus d'envoi d'un message (action *start*). Un choix non-déterministe se présente alors : soit le canal de transmission n'est pas prêt et alors l'émetteur attend un temps (action *attendre*) ; soit le canal est libre et l'émetteur envoie le message (action *envoyer*). Dans ce dernier cas, la probabilité que le message soit envoyé avec succès est égale à 0,99, alors que la probabilité qu'il se produise une collision est égale à 0,01. Si le message est envoyé, l'émetteur s'arrête, alors qu'en cas de collision, il retente son émission (action *réessayer*).

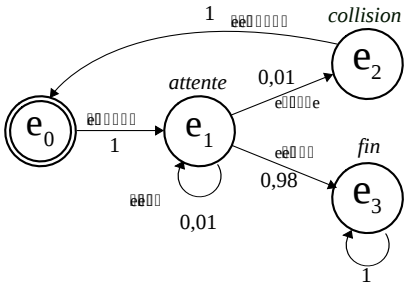


FIG. A.4 – Modélisation d'un protocole de communication indéterministe par un MDP.

A partir de chaque état e d'un MDP, l'état successeur est choisi en deux temps : d'abord une action disponible $a \in Act$ (c'est-à-dire une action pour laquelle $Steps(e, a)$ est défini) est choisie de façon non-déterministe, puis, un état successeur est choisi aléatoirement selon la distribution de probabilité $Steps(e, a)$. Par ailleurs, quelque soit l'état e, $Steps(e)$ est toujours non-vide ce qui signifie qu'il ne peut pas y avoir d'état bloquant dans un MDP.

Pour pourvoir raisonner formellement sur le comportement des MDPs, on utilise la notion d'*adversaire* qui permet de résoudre tous les choix non-déterministes qui se présentent dans un modèle. Par exemple, dans le cas où le non-déterminisme est utilisé pour modéliser la concurrence entre composants d'un système, un adversaire représente un ordonnancement possible des différents composants au cours du fonctionnement du système global. En considérant chaque adversaire, le comportement d'un MDP est déterministe et probabiliste et, comme pour les DTMCs, il est possible de définir un espace probabilisé sur l'ensemble des chemins dans le modèle. Par suite, il est possible de raisonner sur le meilleur ou le pire comportement du système modélisé, en quantifiant sur tous les adversaires possibles. Par exemple, il est possible de calculer la probabilité minimum ou maximum qu'un événement se produise.

3.3.2 Limite de la modélisation à temps discret

Comme les DTMCs, les MDPs sont des modèles markoviens à temps discret. Pour modéliser de nombreux systèmes, il est nécessaire d'utiliser un modèle continu du temps, où les délais des transitions entre états sont des nombres réels positifs arbitraires. C'est notamment le cas des *chaînes de Markov à temps continu* (CTMCs), où les délais des transitions sont modélisés par des distributions exponentielles.

3.4 Chaînes de Markov à temps continu

De façon informelle, une CTMC est une structure de Kripke dont les transitions sont pondérées par des nombres réels non nuls appelés des *taux*. Les CTMCs sont des modèles très populaires pour la modélisation et l'évaluation des performances des systèmes dans de nombreux domaines variés, par exemple les réseaux informatiques, les réseaux de communications, ou encore en bioinformatique pour modéliser les échanges entre cellules biologiques. En particulier, les CTMCs sont les formalismes sous-jacents aux modèles PEPA (*Performance Evaluation Process Algebra*) une algèbre de processus dédiée à l'analyse des performances des systèmes non mobiles [Hil05a, Hil05b]. A l'instar des DTMCs et des MDPs, les CTMCs peuvent être étendues par des récompenses ou des coûts.

3.4.1 Notion de distribution exponentielle

Une fonction de distribution cumulée est définie par :

$$F(t) = Pr(X \leqslant t) = \int_{-\infty}^{t} f(x)dx$$

où f est la *fonction de densité de probabilité*. On a : $\forall t, Pr(X = t) = 0$.

Une variable aléatoire continue X est dite *exponentielle de taux* $\lambda > 0$ si la fontion de densité est donnée par :

$$f(t) = \begin{cases} \lambda.e^{-\lambda t} & si\ t > 0 \\ 0 & sinon \end{cases}$$

Les distributions exponentielles sont adaptées à la modélisation de nombreux phénomènes réels tels que l'occurrence des pannes (par exemple le temps avant que les machines tombent en panne), les délais inter-arrivée (par exemple le temps avant qu'un autre appel arrive à un centre d'appel), ou encore les systèmes biologiques (par exemple les temps de réaction entre des protéines).

3.4.2 Définition

Définition A.4. Une CTMC est un quintuplet (E, e_{init}, R, PA, L) où E est un ensemble fini d'états (l'« espace des états »), $e_{init} \in E$ est l'état initial, $R : E \times E \to \mathbb{R}_0$ est la matrice des taux des transitions, PA est un ensemble de propositions atomiques, et $L : S \to 2^{PA}$ est une fonction d'étiquetage des états, qui associe à chaque état e de E, un ensemble $L(e)$ de propositions atomiques issues de l'ensemble PA.

La matrice R assigne un taux λ, qui est le paramètre d'une distribution exponentielle, à chaque paire d'états e et e' de la CTMC. De la même façon que pour les DTMCs et les MDPs, une transition existe entre e et e', si et seulement si $R(e, e') > 0$. La probabilité que la transition entre e et e' soit déclenchée avant t unités de temps est égale à $1 - e^{-R(e,e').t}$.

Exemple. Un exemple classique de CTMC est le modèle simple d'une file d'attente de travaux. Initialement, cette file d'attente est vide. Les travaux arrivent dans la file au taux de $\frac{3}{2}$ (c'est-à-dire que le temps inter-arrivée moyen des travaux dans la file est égal à $\frac{2}{3}$). Les travaux sont réalisés et sortent de la file d'attente au taux de 3 (c'est-à-dire que le temps moyen de réalisation d'un travail est égal à $\frac{1}{3}$). La file d'attente ne peut contenir qu'au plus 3 travaux. Cette file d'attente peut être modélisée par la CTMC suivante :

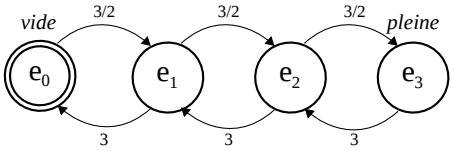


FIG. A.5 – Modélisation d'une file d'attente de travaux à l'aide d'une CTMC.

La matrice R attribue un taux à chaque paire d'états dans la CTMC. Une transition ne peut se produire entre les états s et s' que si $R(s,s') > 0$ et, dans ce cas, le délai avant que la transition puisse se produire est modélisée par une distribution exponentielle de taux $R(s,s')$, c'est à dire la probabilité de cette transition est déclenchée en t unités de temps est égale à $1 - e^{-R(s,s').t}$. Habituellement, à partir d'un état s, il y a plus d'un état s' pour lequel $R(s,s') > 0$, ce qui est considéré comme une condition de course. La première transition à être réalisée détermine le prochain état de la CTMC.

Enfin, comme dans le cas des DTMCs, un espace probabilisé sur les chemins d'une CTMC peut être défini [BHHK03], et il est alors possible de raisonner sur la probabilité d'occurrence de certains comportements des systèmes modélisés.

3.5 Logiques temporelles stochastiques

3.5.1 PCTL : une logique temporelle pour les DTMCs.

La logique temporelle PCTL (*Probabilistic Computation Tree Logic*) permet de spécifier des propriétés des chaînes de Markov à temps discrets. Elle a été définie en 1994 par Hans Hansson and Bengt Jonsson dans *A Logic for Reasoning about Time and Reliability* [HJ94]. Elle étend la logique temporelle CTL par l'ajout de l'opérateur probabiliste P, et par les extensions quantitatives des opérateurs A et E.

Définition A.5. La syntaxe de PCTL est définie par :
- des formules d'états : $\Phi ::= true \mid a \mid \Phi \wedge \Phi \mid \neg\Phi \mid P_{\sim p}[\Psi]$; et
- des formules de chemins : $\Psi ::= X\,\Phi \mid \Phi\,U^{\leqslant k}\,\Phi \mid \Phi\,U\,\Phi$

où :
- a est une proposition atomique utilisée pour identifier les états qui intéressent l'analyse ;
- $\sim \in \{<, >, \leqslant, \geqslant\}$ et $p \in [0,1]$ est une probabilité ;
- $k \in \mathbb{N}$;
- $X\Phi$ signifie « Φ est vraie à la prochaine étape » (opérateur « next ») ;
- $\Phi_1\ U^{\leq k}\ \Phi_2$ signifie « Φ_2 est vraie en k étapes et Φ_1 est vraie jusqu'à ce point » (opérateur « bounded until ») ;
- $\Phi_1\ U\ \Phi_2$ signifie « Φ_2 est finalement vraie et Φ_1 est vraie jusque là » (opérateur « until »).

Une propriété s'exprime toujours à l'aide d'une formule d'état. Les formules de chemins ne peuvent apparaître qu'à l'intérieur de l'opérateur probabiliste P. La formule d'état $P_{\sim p}[\Psi]$ signifie que « la formule de chemin Ψ est vraie avec la probabilité $\sim p$ ». Le résultat de l'analyse de cette propriété qualitative est vrai ou faux. Par ailleurs, il est possible d'évaluer la

probabilité que la formule de chemin Ψ soit vraie. La propriété quantitative correspondante est $P_{=?}[\Psi]$.

Les formules PCTL sont analysées sur l'ensemble des états d'une DTMC. Un état s d'une DTMC satisfait une formule PCTL Φ, et on note $s \vDash \Phi$, si et seulement si Φ est vraie pour s. La sémantique des formules d'états et des formules de chemins peut alors être définie.

Définition A.6. Pour un état s d'une DTMC D :
- $s \vDash true$ (toujours) ;
- $s \vDash a \Leftrightarrow a \in L(s)$;
- $s \vDash \Phi_1 \wedge \Phi_2 \Leftrightarrow s \vDash \Phi_1 \wedge s \vDash \Phi_2$;
- $s \vDash \neg\Phi \Leftrightarrow s \vDash \Phi$ est faux ;
- $s \vDash P_{\sim p}[\Psi] \Leftrightarrow \mathcal{P}_s(\Psi) \sim p$ où $\mathcal{P}_s(\Psi) = Pr_s\{\omega \in Chemins_D(e) \mid \omega \vDash \Psi\}$ (Les ensembles des chemins qui satisfont Ψ sont toujours mesurables). De façon informelle, $s \vDash P_{\sim p}[\Psi]$ signifie que *la probabilité, qu'à partir de l'état s Ψ soit vraie pour un chemin sortant, satisfait $\sim p$.*

De manière similaire, $\omega \vDash \Psi$ dénote que « le chemin ω satisfait Ψ » ou bien que « Ψ est vraie le long de ω ». Pour un chemin $\omega = e_0 e_1 e_2 e_3 \ldots$ dans la DTMC (S, e_{init}, P, AP, L), il vient :
- $\omega \vDash X\Phi \Leftrightarrow e_1 \vDash \Phi$;
- $\omega \vDash \Phi_1 U^{\leqslant k} \Phi_2 \Leftrightarrow \exists i \leqslant k$, tel que $e_i \vDash \Phi_2$ et $\forall j < i, e_j \vDash \Phi_1$;
- $\omega \vDash \Phi_1 U \Phi_2 \Leftrightarrow \exists k \geqslant 0$ tel que $\omega \vDash \Phi_1 U^{\leqslant k} \Phi_2$.

3.5.2 Extension pour les récompenses

Pour pouvoir exprimer des propriétés spécifiant des récompenses, la logique PCTL est étendue par l'ajout de l'opérateur R.

Définition A.7. La syntaxe des formules d'états de PCTL étendue est :

$$\Phi ::= \ldots \mid R_{\sim r}[I = k] \mid R_{\sim r}[C \leq k] \mid R_{\sim r}[F\Phi]$$

où Φ est une formule PCTL d'état, $r \in \mathbb{R}_{\geq 0}$, $\sim \in \{<, >, \leq, \geq\}$ et $k \in \mathbb{N}$. Les points de suspension représentent les formules d'états précédentes.

Dans la définition précédente, $R_{\sim r}[I = k]$ dénote une *propriété instantanée* et signifie que « la valeur prévue de la récompense d'état au temps k est $\sim r$ ». La formule $R_{\sim r}[C \leqslant k]$ dénote une *propriété cumulée* et signifie que « la récompense cumulée jusqu'au temps k, prévue est $\sim r$ ». Enfin, la formule $R_{\sim r}[F\Phi]$ dénote une *propriété d'atteignabilité* et signifie « la récompense cumulée avant d'atteindre un état satisfaisant Φ prévue est $\sim r$ ».

La sémantique des trois opérateurs de récompense se base sur des variables aléatoires sur des chemins infinis. La sémantique des opérateurs R est alors, pour un état e d'une DTMC :
- $s \vDash R_{\sim r}[l = k] \Leftrightarrow Exp(e, X_{l=k}) \sim r$;
- $s \vDash R_{\sim r}[C \leqslant k] \Leftrightarrow Exp(e, X_{C \leqslant k}) \sim r$;
- $s \vDash R_{\sim r}[F\Phi] \Leftrightarrow Exp(e, X_{F\Phi}) \sim r$

où $Exp(e, X)$ est l'espérance de la variable aléatoire $X : Chemins(s) \rightarrow \mathbb{R}_{\geqslant 0}$ en considérant la mesure de probabilité $\mathcal{P}$.

L'établissement des propriétés cumulatives sont une variante de la méthode de calcul des probabilités pour la formule $U^{\leqslant k}$ et sont les solutions d'équations récursives. Les propriétés d'atteignabilité sont similaires au calcul des probabilités des formules U et leur établissement nécessite une phase de pré-calcul (identification d'une infinité d'états récompensant) avant la résolution d'un système d'équation linéaire.

3.5.3 Limite de l'expressivité de la logique PCTL

La limite de l'expressivité de PCTL s'explique essentiellement par le fait qu'on établit la probabilité d'atteindre des états dans X en passant seulement par des états de Y (et en k étapes temporelles). Des logiques temporelles plus expressives peuvent être utilisées, par exemple la logique non probabiliste à temps linéaire LTL [Pnu77], ou PCTL* [ASB+95, BdA95] qui englobe à la fois les logiques PCTL et LTL. Ces deux logiques permettent de combiner des opérateurs de chemins, alors qu'en PCTL l'opérateur $P_{\sim p}[\Psi]$ contient toujours un opérateur simple.

3.5.4 La logique PCTL pour les MDPs

La logique temporelle PCTL permet également de spécifier les propriétés des MDPs. La syntaxe des formules d'état et des formules de chemins ne sont pas modifiées, mais avec une quantification implicite sur les adversaires. L'opérateur $P_{=?}$ utilisé pour les DTMCs est remplacé par deux variants qui sont $P_{min=?}$ et $P_{max=?}$, c'est-à-dire « quelle est la probabilité maximale/minimale (en considérant tous les adversaires) que la formule de chemin Ψ soit vraie ? ». Par exemple, pour un protocole, on cherchera à connaître les probabilités minimale et maximale qu'un message soit envoyé avant l'échéance. Cela correspond à une analyse du meilleur et du pire comportement du système. Le model checking n'est pas plus difficile puisque, de toutes façons, on calcule les valeurs de $p_{min}(s, \Psi)$ ou $p_{max}(s, \Psi)$.

La vérification de $P_{\sim p}[\Psi]$ se ramène aux calculs des probabilités minimale et maximale que Ψ soit vraie, en considérant tous les adversaires possibles. On montre qu'il y a toujours des adversaires sans mémoire A_{min} et A_{max} pour lesquels, $Prob^{A_{min}}(s, \Psi) = p_{min}(s, \Psi)$ et $Prob^{A_{max}}(s, \Psi) = p_{max}(s, \Psi)$.

3.6 CSL : une logique temporelle pour les CTMCs.

La logique temporelle CSL (*Continuous Stochastic Logic*) [ASSB00,BHHK03] est destinée à spécifier des propriétés des CTMCs. La syntaxe de CSL distingue les formules d'état, des formules de chemin ; une formule CSL étant toujours une formule d'état. CSL permet d'exprimer des propriétés quantitatives : $P_{=?}[\Psi]$ et $S_{=?}[\Phi]$.

Définition A.8. La syntaxe de CSL est définie par :
- formules d'état : $\Phi ::= true \mid a \mid \Phi \wedge \Phi \mid \neg\ \Phi \mid P_{\sim p}[\Psi] \mid S_{\sim}[\Phi]$
- formules de chemin : $\Psi ::= X\Phi \mid \Phi U^{l} \Phi$

où :
- a est une proposition atomique ;
- $p \in [0, 1]$ est une probabilité ;
- $\sim \in \{<, >, \leqslant, \geqslant\}$;
- $P_{\sim p}[\Psi]$ signifie que la formule de chemin Ψ est vraie avec la probabilité $\sim p$;
- $S_{\sim p}[\Phi]$ signifie qu'à long terme Φ est vrai avec la probabilité $\sim p$;
- X : « next » ; U^{l} : « jusqu'à en temps limité ».

Les formules d'états CSL sont interprétées sur les états d'une CTMC. $s \vDash \Phi$ dénote que « Φ est vraie dans l'état s » ou que « Φ est satisfaite dans l'état s ».

Définition A.9. Pour un état s de la CTMC (S, s_{init}, R, PA, L) :
- $s \vDash a \Leftrightarrow a \in L(s)$;
- $s \vDash \Phi_1 \wedge \Phi_2 \Leftrightarrow s \vDash_1$ et $s \vDash \Phi_2$;
- $s \vDash \neg\Phi \Leftrightarrow s \vDash \Phi\ est\ false$;

- $s \vDash P_p[\Psi] \Leftrightarrow Prob(s, \Psi) \sim p$; (probabilité, en partant de l 'état s, de satisfaire la formule de chemin Ψ) ;
- $s \vDash S_{\sim p}[\Phi] \Leftrightarrow \sum s' \vDash \Pi_s(s') \sim ps \vDash S_{\sim p}[\Phi] \Leftrightarrow \sum s' \vDash \Pi_s(s') \sim p$ (probabilité, en partant de l'état s, d'être dans l'état s' à long terme).

La sémantique des formules de chemins est définie de la façon suivante. Pour un chemin ω d'une CTMC :

- $\omega \vDash X\Phi \Leftrightarrow \omega(1)$ est défini et $\omega(1) \vDash \Phi$ (si $\omega(0)$ est absorbant alors $\omega(1)$ n'est pas défini) ;
- $\omega \vDash \Phi_1 U^l \Phi_2 \Leftrightarrow \exists t \in l, (\omega_t \vDash \Phi_2 \wedge \forall t' < t, \omega_{t'} \vDash \Phi_1)$ (il existe un instant dans l'intervalle l où Φ_2 est vraie et Φ_1 est vraie à tous les instants précédents).

4 Le model checking temporisé

Le model checking temporisé considère les aspects liés au temps dans le comportement des systèmes. Ce peut être un paramètre important. Par exemple, beaucoup de protocoles de communication utilisent des *timeout* pour permettre aux nœuds qui communiquent de ne pas rester bloqués en attente d'une transmission qui ne vient pas. D'autres systèmes dits *temps réels* imposent des délais maximaux pour les tâches qu'ils accomplissent avec, éventuellement, leur suspension. Il est alors utile d'analyser les exigences de performabilité en considérant le temps, en particulier les temps de réponse des tâches temps réel. Le model checking temporisé s'appuie sur la théorie des automates temporisés présentée au paragraphe 3.4.1.

4.1 Logiques temporelles temporisées

Il existe de nombreuses extensions de logiques temporelles permettant d'exprimer des contraintes quantitatives sur les délais des actions d'un système à analyser. La première consiste à compléter l'opérateur temporel U (*until*) avec une contrainte de la forme $\sim k$ avec $\sim \in \{=, <, \leqslant, \geqslant, >\}$ et $k \in \mathbb{N}$. La formule $\varphi \, U_{<k} \, \psi$ est vraie pour un chemin ρ, s'il existe un état l situé à moins de k unités de temps de l'état initial vérifiant ψ, et tel que tous les états le précédant vérifient φ. Les opérateurs F et G sont étendus de la même façon. L'exigence « l'alarme doit se déclencher en au plus 3 secondes après la détection d'une fuite de gaz » s'écrit en logique TCTL (*Timed CTL*) [AH91, ACD93, HNSY94] :

$$AG(pb \Rightarrow AF_{\leqslant 3} \; alarme)$$

ou en TLTL (*Timed LTL*)

$$G(pb \Rightarrow F_{\leqslant 3} \; alarme)$$

Une autre méthode consiste à ajouter des horloges aux logiques temporelles (on parle horloges de formules), des opérateurs de remises à zéro, et des contraintes du même types que celles utilisées dans les automates. La formule précédente s'écrit alors de la manière suivante :

$$AG(pb \Rightarrow xin(AF(x < 3 \wedge alarme)))$$

L'opérateur *in* réinitialise l'horloge x à zéro lorsqu'on rencontre un état vérifiant pb, et il suffit donc de vérifier que $x < 3$ lorsqu'on rencontre un état vérifiant *alarme* pour s'assurer que le délai séparant les deux positions est bien inférieur à 3. Cette méthode permet d'exprimer des propriétés plus fines mais est moins intuitive que la précédente. Il est également possible d'étendre les logiques modales avec des horloges de formules et d'utiliser des point fixes pour exprimer des propriétés sur le comportement des systèmes.

4.1.1 Extension temporisée de CTL

La logique TCTL est interprétée sur des systèmes de transitions temporisées présentés au paragraphe 3.4.1. L'opérateur U est assorti d'une contrainte $\sim c$ où $\sim$ représente un opérateur de comparaison parmi $\{<, >=, =, >=, >\}$, et c est une constante entière ($c \in \mathbb{N}$). Ce qui permet de contraindre les dates d'occurrence de la propriété ψ.

Définition A.10. La syntaxe de TCTL est définie par :

$$\phi,\ \psi ::= P \mid \neg\ \phi \mid \phi\ \wedge\ \psi \mid \phi\ U_{\bowtie c}\ \psi \mid A\ \phi U_{\bowtie c}\psi$$

où P est une proposition atomique.

4.1.2 Extension temporisée de LTL

La logique MTL [Koy90] est une extension temporisée de LTL.

Définition A.11. La syntaxe de MTL est définie par :

$$\phi, \psi ::= P \mid \neg\ \phi \mid \phi\ \wedge\ \psi \mid \phi\ U_I\ \psi$$

où P est une proposition atomique et I est un intervalle.

4.1.3 Outils

L'outil UPPAAL est certainement le plus connu des outils de vérification temporisés [UPPa, GBL06, LPY97]. L'outil Kronos [KRO, BDM+98] permet de vérifier des propriétés TCTL. Il utilise un algorithme d'étiquetage classique et permet aussi la vérification de propriétés d'accessibilité à la volée. L'outil HCMC [HCM, LLW95] permet de vérifier des propriétés écrites en logique modale temporisée. Il utilise un algorithme compositionnel : étant donné un problème $(A_1|\ldots|A_n) \models \varphi$, on construit une formule φ/A_n telle que $(A_1|\ldots|A_n) \models \varphi$ si et seulement si $(A_1|\ldots|A_{n-1}) \models \varphi/A_n$. Des règles de simplification permettent généralement d'éviter que la taille de formule n'explose au cours de processus.

5 Model checking probabiliste et temporisé

Les comportements des systèmes probabilistes et temps réel se modélisent avec des automates temporisés probabilistes présentés au paragraphe 3.5.1. Les propriétés de ces systèmes peuvent se spécifier avec la logique temporelle PTCTL (*probabilistic timed CTL*) qui est dérivée des logiques PCTL [BdA95] et de TCTL [AD94].

Définition A.12. La syntaxe de PTCTL est :

$$\Phi ::= true \mid a \mid \zeta \mid z.\Phi \mid \Phi \wedge \Phi \mid \neg\Phi \mid P_{\sim p}[\Phi U \Phi]$$

où :

- a est une proposition atomique ;
- $p \in [0, 1]$;
- $\sim \in \{<, >, \leqslant, \geqslant\}$;
- $z.\Phi$ est le « quantificateur de gel » (freeze quantifier).
- Z est un ensemble de formules d'horloges
- ζ est la zone sur $X \cup Z$; $\zeta \in Zones(X \cup Z), z \in Z$.

6 Conclusion

La technique du model checking a eu des difficultés a se faire accepter. L'idée que cette façon de procédé pouvait être intéressante pour la vérification de modèles à états finis n'a pas été reconnue à l'origine. Les premières réactions au model checking ont surtout été de la confusion et du désintéressement. Beaucoup de spécialistes du sdomaine de la validation des systèmes estimaient que le model checking ne pourrait pas bien fonctionner en pratique. Lentement cette technique a commencé à susciter des commentaires moins amers : le model checking est « une béquille acceptable » disait Edsger W. Dijkstra, et « une première étape vers l'ingénierisation du domaine » disait A. Pnueli [Pnu97].

Pourtant le model checking est aujourd'hui une technique reconnue. Quels facteurs ont contribué à cette reconnaissance ? E. Allen Emerson en note trois [CES09]. D'abord, son cadre théorique réalisable et compréhensible qui s'appuie sur les logiques temporelles et sur l'algorithmique. Ensuite son automaticité : il permet de détecter la présence d'erreurs de conception des systèmes de façon automatique. Par ailleurs, en ce qui concerne la vérification des programmes informatiques, il sépare nettement leur conception, de leur débogage, contrairement à la technique de vérification déductive de Floyd-Hoare, en vigueur au début du model checking, qui consistait à inclure des assertions dans les programmes lors de leur conception. De plus, cette méthode de vérification n'est pas automatique. Enfin, le troisième facteur qui a contribué à la reconnaissance du model checking est certainement le gain en puissance des ordinateurs depuis le début des années 1980. Des machines plus puissantes ont permis de supporter des outils de model checking demandant plus de ressources pour fonctionner.

A ces trois facteurs, on pourrait ajouter le facteur recherche et développement, qui est largement orienté vers la résolution du problème majeur du model checking, à savoir l'explosion combinatoire des états. Cette recherche a permis de repousser les limites de l'applicabilité du model checking, et a aussi contribué, avec les trois facteurs précédents, à son succès. Aujourd'hui, la vérification par model checking se fait systématiquement à une grande échelle sur beaucoup de systèmes, tant matériel (en particulier pour vérifier des puces électroniques) que logiciels, y compris des systèmes industriels. Les grandes organisations, depuis les fournisseurs de matériel, jusqu'aux organismes gouvernementaux adoptent la technique du model checking. E. Allen Emerson remarque que, contrairement à il y a quelques décennies, on ne fait plus seulement que parler du model checking, mais on l'utilise.

Annexe B

L'outil PRISM

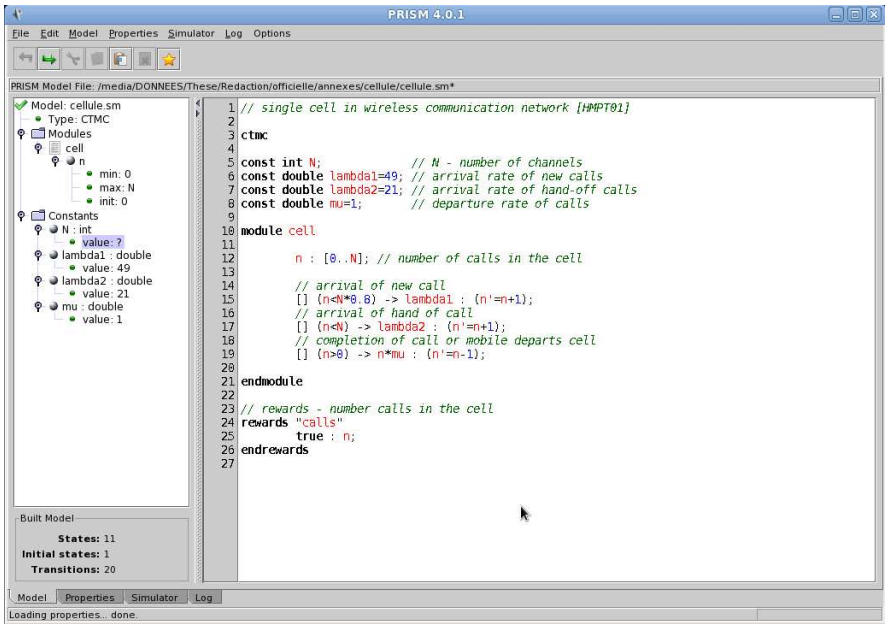
Cet annexe décrit l'outil PRISM, y compris sa dernière version parue en juillet 2011 qui supporte les automates temporisés probabilistes. Les principes de la modélisation et de la vérification des propriétés des modèles sont présentés, ainsi que les possibilités de l'outil, mais cette annexe n'est pas un tutoriel. Si le lecteur souhaite avoir de plus amples renseignements sur cet outil, il peut consulter le site internet de PRISM [PRIb], où il trouvera son manuel d'utilisation [PRIa], un article général sur l'outil [HKNP06], une présentation de sa dernière version [KNP11], ainsi que de très nombreux exemples commentés.

1 Introduction

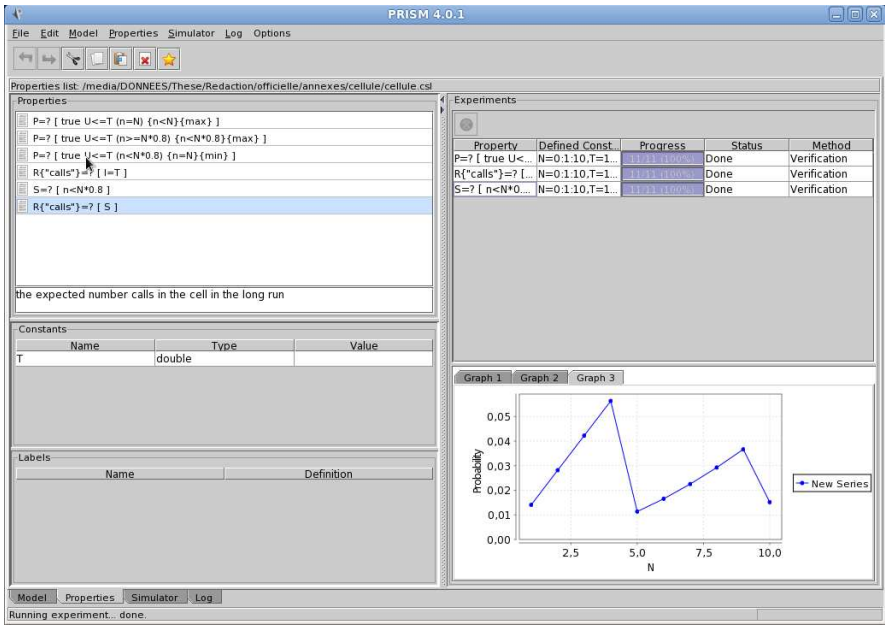
PRISM est un outil de vérification automatique des propriétés des modèles stochastiques (*probabilistic model checker*), permettant d'analyser des systèmes dont les comportements sont aléatoires. Les techniques de *model checking*, exposées à l'annexe précédente, sont des techniques de vérification formelle basées sur la construction d'un modèle mathématique du système à analyser. Les exigences de ce système sont exprimées sous la forme de propriétés qualitatives et qualitatives de logique temporelle. PRISM supporte plusieurs types de modèles stochastiques : les *chaînes de Markov à temps discret* (DTMCs), les *chaînes de Markov à temps continu* (CTMCs), les *processus de décisions de Markov* (MDPs), et depuis la version 4.0.1, les *automates temporisés probabilistes* (PTAs). Tous ces modèles peuvent être enrichis par des récompenses associées aux états et/ou aux transitions. Les propriétés, quant-à-elles, sont spécifiées dans le langage de spécification des propriétés qui incorpore les opérateurs des logiques temporelles PCTL (*probabilistic computation tree logic*), CSL (*continuous stochastic logic*), LTL (*linear time logic*), et PCTL*. Les modèles et les propriétés sont spécifiés à l'aide d'une interface graphique conviviale, illustrée à la figure B.1. L'analyse des propriétés se base sur une recherche exhaustive et une solution numérique. L'outil contient également un moteur de simulation à événements discrets permettant une vérification approchée des propriétés, selon la technique de la vérification statistique de modèle (*statistical model checking*). L'outil supporte les modèles de CMTCs décrit avec l'algèbre de processus stochastique PEPA [PEP, Hil05a,Hil05b,TDG09]. Il est également possible d'importer des modèles décrits dans d'autres langages que celui de PRISM, par exemple SBML (*systems biology markup language*).

2 Principes de la modélisation

Les modèles sont spécifiés à l'aide du langage de description de PRISM, hormis les modèles PEPA que nous ne décrivons pas dans cette annexe. Pour indiquer son type, le modèle peut incorporer un mot réservé parmi : `dtmc`, `ctmc`, `mdp`, ou bien `pta`. Ils sont enregistrés dans des fichiers dont l'extension est : `.pm` pour les DTMCs, `.sm` pour les CTMCs, et `.nm` pour les



(a) L'onglet destiné à la modélisation d'un système.



(b) L'onglet destiné à la vérification des propriétés.

FIG. B.1 – L'interface homme-machine de PRISM.

MDPs et les PTAs.

Un modèle d'un système est composé par un ensemble de plusieurs modules spécifiés par :

`module` *`nom_module`* `...` `endmodule`

où *`nom_module`* est le nom obligatoire du module. Les modules peuvent interagir entre-eux et être synchronisés. Chacun d'eux comporte un ensemble de variables locales pouvant prendre un nombre fini de valeurs. L'état d'un module à un instant donnée est défini par les valeurs de ses variables locales. L'ensemble de toutes les variables locales de tous les modules est dénoté $\mathcal{V}$. Les valeurs de ces variables à un instant donné définissent l'état du modèle global.

Le comportement de chaque module est décrit par un ensemble de commandes gardées. Pour les DTMCs et des MDPs, la forme de ces commandes est :

`[action] garde` $\rightarrow$ `p_1 : u_1 + ... + p_m : u_m ;`

où `[action]` est une étiquette d'action optionnelle qui permet de synchroniser les modules entre-eux, `garde` est un prédicat sur l'ensemble des variables du modèle global, y compris celles des autres modules, `p_i` $\in [0,1]$ est une probabilité, et `u_i` sont des mises à jour des variables locales du module `x_j` (`j` $\in \{$`1..n`$\}$), de la forme :

`(x_1' =` $u_{i,1}$`) & ... & (x_n' =` $u_{i,n}$`)`

où $u_{i,j}$ est une fonction sur $\mathcal{V}$. Une commande est permise si l'état global courant s du modèle satisfait `garde`. Dans ce cas, le module mettra à jour, avec la probabilité `p_i`, ses variables locales `x_j` (`j` $\in \{$`1..n`$\}$), en réinitialisant leur valeur respective en $u_{i,j}(s)$ en fonction de `u_i`. Pour les CTMCs, les commandes sont de la forme :

`[action] garde` $\rightarrow$ `taux : u ;`

où `action`, `garde`, et `u` ont la même signification et la même forme que ci-dessus, et `taux` $\in \mathbb{R}_{>0}$. Lorsque la garde est satisfaite, une transition de taux `taux` met à jour les variables locales en fonction de `u`. Si plusieurs commandes avec la même mise à jour peuvent être exécutées, les transitions correspondantes sont combinées en une seule transition dont le taux est la somme de tous les taux individuels. Pour les transitions qui résultent de la synchronisation entre plusieurs modules, la probabilité ou le taux associé est obtenu en multipliant ceux de chaque transition composante.

PRISM permet également d'établir des mesures quantitatives sur les modèles, telles que le nombre de messages perdus pendant la transmission, le temps mis par un accusé de réception pour parvenir à un émetteur après l'envoi d'un message, ou encore l'énergie consommée prévisionnelle. Pour cela, les modèles sont enrichies de coûts ou de récompenses, qui sont des valeurs réelles associés à certains états ou certaines transitions du modèle. En fait, PRISM ne supporte que les récompenses, mais ces deux notions ont pratiquement similaires, leur seule différence est qu'un coût est perçu comme quelque chose de mauvais, alors que c'est le contraire pour une récompense. Ainsi on utilise que la construction pour les récompenses qui est

`rewards "nom" ... endrewards`

où `nom` est le nom optionnel de la récompense. Les récompenses apparaissent toujours à l'extérieur des modules. Elles sont composées d'éléments qui spécifient les récompenses associées à certains états, ou certaines transitions du modèle. Chaque élément de récompense d'état est de la forme

```
guarde : recompense ;
```

où `garde` est un prédicat sur toutes les variables du modèle, et `recompense` est une expression numérique exprimant la valeur de la récompense assignée aux états qui satisfont `garde`. Par ailleurs, les éléments des récompenses de transitions sont de la forme

```
[action] garde : recompense ;
```

permettant d'assigner `recompense` aux transitions qui satisfont `garde`, et qui sont étiquetées par l'action `action`.

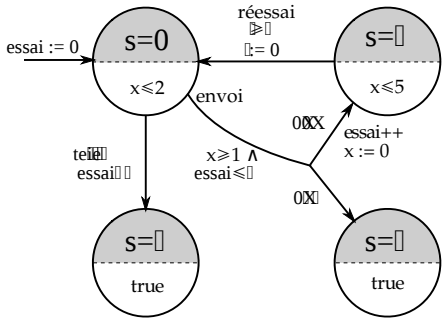
Les modèles ne doivent pas exhiber d'états bloquants, qui sont des états qu'il est possible d'atteindre, mais pour lesquels il n'existe aucune transition de sortie. Ces états constituent des erreurs de modélisation, détectées par PRISM et affichés dans l'onglet « Log » de l'interface, qu'il est possible de corriger en ajouter des transitions en boucle sur ces états.

3 Modélisation des automates temporisés probabilistes

Depuis la version 4.0.1, PRISM permet de modéliser des *automates temporisés probabilistes* (*probabilistic timed automata*, PTAs). Ce sont des automates finis temporisés munis d'horloges à valeurs réelles positives, et de choix probabilistes discrets à la manière des processus de décision de Markov (MDPs). Les horloges sont incrémentées simultanément dans le temps absolu. Des gardes et des invariants sont assignés, respectivement aux transitions et aux états de l'automate, imposant ainsi des restrictions sur le moment où ces transitions peuvent se produire, et combien de temps il est possible de stationner dans un état. Il est également possible de spécifier des variables à valeurs finies. Les transitions entre états peuvent réinitialiser les horloges et mettre à jour les variables. Les PTAs sont des automates probabilistes : l'état cible, les réinitialisations des horloges, ainsi que les mises à jour des variables sont spécifiés par une *distribution probabiliste discrète*. Par ailleurs, les choix entre plusieurs transitions et l'écoulement du temps sont indéterministes. Comme les autres modèles probabilistes supportés par PRISM, les PTAs peuvent être enrichis par des coûts ou des récompenses. PRISM ne supporte que les *PTAs enrichis linéaires* dont l'accumulation des coûts/récompenses est proportionnelle au temps, selon un taux dépendant de l'état courant.

Modélisation des PTAs. Le mot réservé `clock` permet de spécifier des horloges.Chaque horloge spécifiée doit être locale à un module particulier, et elle ne peut pas être globale à l'ensemble du modèle. Les horloges peuvent apparaître dans les gardes situées dans le côté gauche d'une commande, et peuvent être réinitialisée, comme toutes les variables, selon une mise à jour apparaissant dans le côté droit d'une commande. Le mot clé `invariant` permet de spécifier des invariants d'états pour chaque module. Pour faciliter la vérification, les expressions qui utilisent des horloges doivent être sous la forme de conjonctions de contraintes simples, de la forme $x \sim c$ ou $x \sim y$, où x et y sont des horloges, c est un entier naturel ou une expression entière, et $\sim$ un opérateur appartenant à l'ensemble $\{<, \leqslant, =, \geqslant, >\}$. Enfin, les modèles de PTAs supportés par PRISM doivent être bien formés selon les principes présentés précédemment, ils ne doivent pas présenter de blocages temporels (*timelocks*), et ne pas être Zénon. Ces deux dernier termes sont expliqués au paragraphe 3.4.1. PRISM recherche les blocages temporels et signale une erreur de modélisation s'il en trouve un. Par contre, dans sa version courante, PRISM ne vérifie pas automatiquement que la condition non Zénon est respectée, ce qui reste donc à la charge du modélisateur.

Exemple (d'après [KNP11]). La figure B.2(a) modélise un protocole de communication entre un émetteur qui essaie d'envoyer un message à un récepteur, via une voie de transmission non fiable. L'émetteur émet un message en au plus 2 unités de temps. Il y parvient avec la probabilité 0,99, et il échoue avec la probabilité 0,01. Dans ce cas, il recommence la procédure en au moins 3 unités de temps, et en au plus 5 unités de temps. Le nombre maximal de tentatives d'émission d'un message est N ($N \in \mathbb{N}$), après quoi l'émission est définitivement abandonnée. Le nom des différentes états du modèle correspond à la valeur de la variable s définie pour modéliser ces états dans la modèle PRISM. La figure B.2(b) représente le code en langage PRISM de cet automate temporisé probabiliste.



(a)

```
pta

const int N;

module emetteur
  // variables locales
  s : [0..3] init 0;
  e : [0..N+1] init 0; // essais
  x : clock;
  invariant
    (s=0 => x<=2) & (s=1 => x<=5)
  endinvariant

  // commandes gardées
  [envoi]     s=0 & x>=1 & e<=N ->
                0.99: (s'=3) +
                0.01: (s'=1) & (e'=e+1) & (x'=0);
  [reessai]  s=1 & x>=3 -> (s'=0) & (x'=0);
  [terminer] s=0 & e>N -> (s'=2);
endmodule
```

(b)

FIG. B.2 – Exemple d'automate temporisé probabiliste (a) et sa spécification en PRISM (b).

Vérification des PTAs. PRISM analyse deux classes de propriétés sur les PTAs : les probabilités minimale et maximale d'accessibilité d'un état particulier éventuellement en un temps limité ; et les récompenses minimale et maximale accumulées avant d'atteindre un état particulier. Pour vérifier les propriétés des PTAs, PRISM utilise deux moteurs de vérification :

- Le *raffinement d'abstractions quantitatives* construit et analyse une suite d'abstractions probabilistes, qui sont raffinées automatiquement à chaque étape de l'analyse pour produire des résultats plus précis. En utilisant des abstractions de jeu aléatoire, définies en terme de zones, cela conduit à un procédé efficace de vérification exacte des propriétés probabilistes d'accessibilité pour les PTAs. C'est le moteur de vérification par défaut de PRISM. Pour utiliser ce moteur, les modèles doivent respecter plusieurs contraintes : les modules ne peuvent pas lire les variables locales des autres modules, et les variables globales ne sont pas autorisées. Par ailleurs, le modèle doit avoir un état initial unique.

- Les *horloges digitales* traduisent un modèle en un modèle fini à temps discret équivalent (avec des horloges entières), et utilisent ensuite les techniques de vérification existantes pour les MDPs. Pour utiliser ce moteur de vérification, les modèles ne doivent pas utiliser les opérateurs de comparaison stricts $>$ et $<$. De plus, les contraintes mettant en œuvre deux horloges, par exemple $x < y$ où x et y sont des horloges, ne sont pas autorisées.

4 Autres outils de vérification probabiliste et temps réel

UPPAAL est un autre outil de vérification d'automates temporisés majeur [UPPa,LPY97, GBL06]. Une extension récente, dénommée *UPPAAL-PRO*, supporte les automates temporisés probabilistes mais, pour le moment, elle ne permet que l'analyse de propriétés d'accessibilité probabilistes maximales [UPPb]. *Fortuna* supporte le même type de modèles, mais cet outil permet également d'inclure des récompenses limites pour les PTAs enrichis [FOR,BJV10]. L'outil *mcpta*, un autre outil de vérification de systèmes temps réel probabilistes, traduit un sous-ensemble du langage de modélisation Modest en PRISM, en utilisant les horloges digitales [mcp, HH09]. Enfin *MRMC*, qui est un outil de vérification des chaînes de Markov, supporte depuis peu les processus de décisions de Markov à temps continu, un modèle combinant l'indéterminisme avec des aspects probabilistes et temps réel des systèmes [KZH+11]. Le site internet de PRISM [PRIb] recense plus abondamment les outils de vérification de modèles probabilistes.

Annexe C

Les réseaux sans fil

1 Généralités

Un *capteur* est un appareil électronique qui transforme une mesure d'un phénomène analogique issu d'un environnement (température, luminosité, humidité, pression, accélération, son, image, vidéo, etc.), en une mesure électrique qui, après sa numérisation, peut être mémorisée, traitée, et exploitée.

Plusieurs capteurs, voire un grand nombre de capteurs, peuvent coordonner leurs actions pour surveiller un phénomène physique dans la zone géographique où ils sont disséminés ou à proximité de celle-ci, en communiquant à l'aide de liaisons sans fil. Ils forment alors un réseau autonome appelé un *réseau de capteurs sans fil* (RCSF). Des nœuds particuliers appelés des *puits* (ou « stations de base ») complètent ce dispositif. Ils sont chargés de collecter les informations transmises par les capteurs avant de les transmettre à un dispositif tiers (serveur, processus, utilisateur, etc.) par l'intermédiaire d'un support qui peut être un réseau filaire, un autre réseau sans fil, un satellite, internet, etc. Les puits peuvent également être chargés de transmettre des requêtes issues des dispositifs tiers aux capteurs. Ils forment de véritables passerelles de communication entre un RCSF et son environnement. Généralement, les capteurs et les puits sont de technologies différentes, bien que certains capteurs peuvent jouer le rôle de puits comme nous l'expliquons au paragraphe 1.1. Les capteurs et (ou) les puits d'un RCSF peuvent être mobiles pour suivre des phénomènes mobiles ou liés à des objets, des animaux, ou encore des êtres humains qui se déplacent.

Leurs particularités technologiques font des RCSFs des réseaux à part. Leur première particularité est que la capacité en énergie électrique embarquée des capteurs est limitée, et leur batterie ne peut généralement pas être rechargée. Par ailleurs, il n'est pas toujours possible de remplacer des nœuds morts, ceux dont l'énergie est épuisée, par des nœuds opérationnels dont les batteries sont chargées. Il est donc important de concevoir des capteurs qui consomment peu d'énergie pour prolonger leur vie, et par suite celle du réseau global, aussi longtemps que possible, de l'ordre de quelques années. Pour les RCSFs, la contrainte d'économie d'énergie est plus importante que d'autres critères de performance, tels que le débit ou la latence, qui eux sont considérés primordiaux pour des réseaux de communications plus classiques. La figure C.1 représente un exemple d'organisation de RCSF.

1.1 Quelques exemples de plateformes

Chaque capteur intègre de nombreux composants matériels et logiciels qui lui permettent d'assurer ses fonctions : on parle plutôt de *plateforme* (*mote* en anglais) que de capteur, ce dernier terme désignant l'unité fonctionnelle chargée de capter les informations issues de

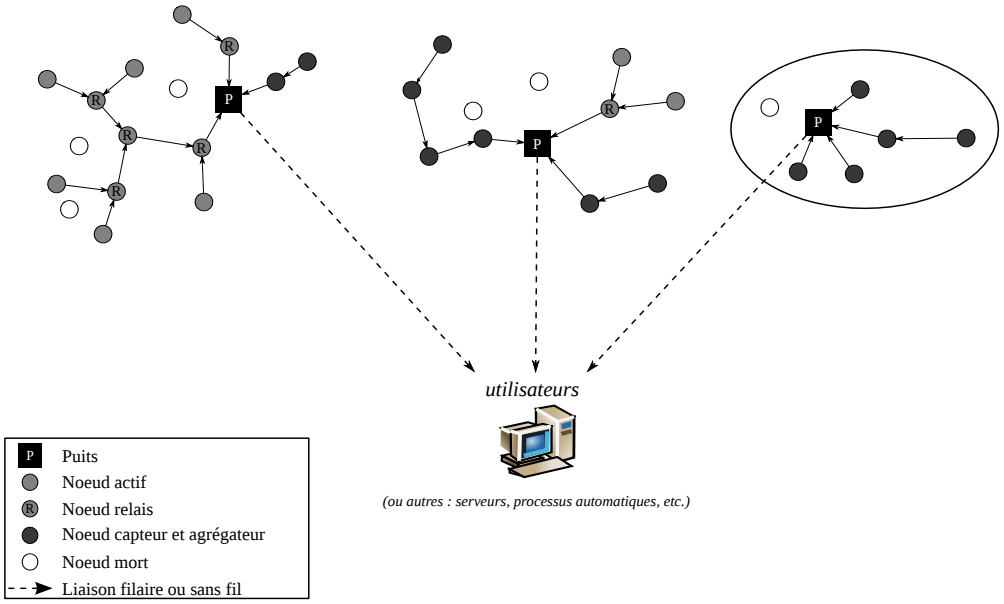


FIG. C.1 – Exemple de réseau de capteurs sans fil.

l'environnement. Nous présentons dans ce paragraphe quatre types différents de plateformes : *BTnode (rev3)*, *MicaZ CM5000*, et *Sun SPOTS*.

BTnode (rev3). Commercialisée par la société *Art of Technology*, cette plateforme est compatible avec Bluetooth [IEE02]. Elle a été utilisée comme prototype pour des MANETs et des RCSFs. Elle utilise un micro-contrôleur Atmel ATmega128L cadencé à 8 MHz. Sa mémoire est composée de 64+180 ko de SRAM, 128 ko de flash, 4 ko de EEPROM. Elle dispose d'interfaces UART, SPI, I2C, GPIO. L'énergie est fournie par 2 piles de type AA.

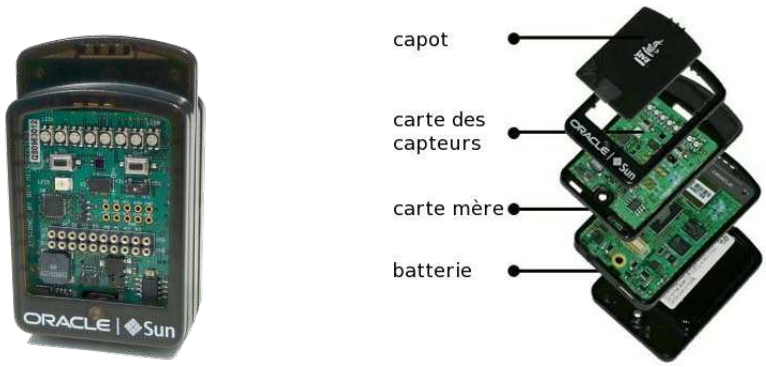
MICAz. Cette plateforme illustrée à la figure C.2(b), est commercialisée par la société Crossbow. Elle est équipée d'un micro-contrôleur 8-bits ATmega128L de Atmel cadencé à 8 MHz. Sa puce radio Texas Instruments CC2420 est compatible avec Zigbee (la norme IEEE 802.15.4) [IEE03]. MicaZ est doté d'une mémoire flash programmable de 128 ko, et une mémoire EEPROM de 4 ko. Sa consommation énergétique est d'environ 28 mA en réception, 25 mA maximum en émission, et moins de 16 μA en mode sommeil. Elle est programmable par l'intermédiaire de sa connexion série UART et elle offre des interfaces digitales telles que I^2C et SPI. Son convertisseur analogique-digital peut mesurer des valeurs comprises entre 0 et 3 V. MicaZ est utilisé pour la sécurité et la surveillance de l'intérieur des bâtiments, pour le captage de données à grande vitesse (données acoustiques, vidéos, vibratoires, etc.) et pour des RCSFs étendus (plus de 1000 nœuds). Une plateforme MICAz peut servir de puits pour envoyer des données vers un ordinateur de type PC ou équivalent.

IRIS. Cette plateforme est compatible avec Zigbee [IEE03]. Elle est doté du micro-contrôleur Atmel ATmega1281, d'une mémoire flash de 512 Ko, d'une mémoire RAM de 8 Ko, d'une mémoire EEPROM de 4 Ko, d'un convertisseur analogique-numérique 10 bits et de divers interfaces I^2C SPI et UART. Sa consommation énergétique est de 24 mA en réception, 25 mA en émission, et moins de 8 μA en mode sommeil. La plateforme Iris est utilisée pour la sécurité, pour la surveillance de systèmes industriels, énergétiques, etc., et dans des RCSFs étendus. Une plateforme Iris peut servir de puits pour collecter des données à envoyer vers un ordinateur de type PC ou équivalent.

(a) BTnode (Art of Technology) (b) MICAz (MEMSIC) (c) IRIS (MEMSIC)

FIG. C.2 – Photographies de quelques plateformes avec le nom de la société qui les commercialise.

Sun SPOT *(Small Programmable Object Technology).* Développé par Sun Microsystems, il s'agit d'un minuscule ordinateur portable mobile compatible Java, équipé d'une interface réseau sans fil. Il est doté d'un microprocesseur RISC 32 bits ARM920T cadencé à 180 MHz. Deux mémoires caches de 16 ko sont utilisées pour les données et pour les instructions. Il n'a pas de système d'exploitation, mais une version appelée Squawk VM de la machine virtuelle Java. Il est équipé d'une mémoire flash de 4 Mo et de 512 ko de RAM pseudo-statique. Les périodes d'activité et de sommeil sont gérées par un micro-contrôleur 8 bits Atmel ATmega88. Ce dernier s'occupe aussi de gérer la capacité de la batterie. Les communications utilisent un CC2420 compatible avec IEEE 802.15.4 et opère dans la bande des 2.4 GHz sans licence. Cette plateforme est dotée de plusieurs ports, tels que USB, Ethernet, des ports séries (SPI), USART (*universal synchronous/asynchronous serial interface*) etc. Les capteurs installés sur une carte dédiée mesurent l'accélération, le température, la luminosité.



(a) photographie (b) éclaté

FIG. C.3 – Photographie et éclaté de la plateforme Sun SPOTS de Sun Microsystems.

1.2 Économiser l'énergie

Les nœuds d'un RCSFs ont une quantité d'énergie embarquée limitée qui, ne pouvant pas être renouvelée, doit être économisée drastiquement pour prolonger leur vie et celle du réseau

global. Dans [YHE02], quatre sources de gaspillage d'énergie électrique sont identifiées :

1. Les *collisions* lorsque plusieurs nœuds communiquent leurs trames en même temps. Les trames transmises sont inutilisables et il faut les retransmettre. D'où la dépense d'énergie causée, d'une part par la transmission de trames qui sont inexploitables parce qu'elles sont reçues corrompues, et d'autre part par leur retransmission nécessaire.

2. L'*écoute passive* (*overhearing*) lorsque les nœuds écoutent des trames qui ne leur sont pas destinées. L'écoute passive est néanmoins nécessaire pour déterminer le moment où le canal de transmission se libère et devient donc utilisable.

3. L'utilisation excessive de *trames de contrôle* qui ne véhiculent pas d'information utiles (des données), mais qui doivent être utilisés au préalable de la transmission de données. Regrouper et transmettre plusieurs trames de données à la fois permet de réduire le nombre de trames de contrôle à utiliser.

4. L'*oisiveté* d'un nœud alors qu'il ne transmet, ni ne reçoit des trames de données. Il laisse son émetteur-transmetteur en marche pour recevoir d'éventuelles trames qui lui sont destinées. La consommation d'énergie est la même qu'en réception. Ainsi, le gaspillage d'énergie est important si le trafic du réseau est faible, ce qui est généralement le cas pour les RCSFs. L'oisiveté est la source de gaspillage est la plus importante.

Les consommations énergétiques causées par l'émission et la réception de trames varient selon le type de nœud utilisé, et il est difficile d'en faire une étude comparée exhaustive. Mais, dans tous les cas, elles sont supérieures à la consommation d'énergie induite par mode sommeil d'un facteur 1000 (mA contre μA), cette dernière n'étant pas nulle. Il appartient à un protocole de la couche MAC (Medium Access Control) de gérer les accès au canal de communication et, par suite, d'ordonnancer l'émetteur-récepteur d'un nœud afin de gérer au mieux ses dépenses en énergie. Pour ce faire, on distingue trois grandes familles de protocoles qui sont les *protocoles à contention* basés sur la stratégie CSMA/CA (*carrier sense multiple access with collisions avoidance*) [IEE07], les *protocoles multiplexés*, et les *protocoles hybrides* qui combinent les techniques des deux familles précédentes pour en tirer des avantages.

1.3 Protocoles MAC à contention pour la réduction d'énergie

En 1998, PAMAS [SWR98] est proposé ; il est considéré comme le premier protocole à contention orienté réduction d'énergie pour les réseaux autonomes. Il s'inspire de MACA [Kar90] et utilise deux canaux de transmission : un pour les trames de contrôle et la tonalité occupée (busy tone), et l'autre pour les trames de données. Les nœuds qui désirent communiquer commencent par échanger les trames de contrôle. Puis ils s'échangent les trames de données alors que la tonalité occupée est transmise sur le premier canal. Les nœuds voisins qui reçoivent cette tonalité diffèrent leurs propres transmissions. La stratégie d'accès au canal de PAMAS a plusieurs avantages. D'abord elle réduit les collisions de MACA. Par ailleurs, les nœuds qui reçoivent la tonalité d'occupation du canal peuvent éteindre leur émetteur-récepteur pour économiser leur énergie. Cependant PAMAS ne propose pas de stratégie pour réduire l'oisiveté des nœuds. Par ailleurs ce protocole se base sur l'utilisation de deux canaux de transmission, alors que la plupart des capteurs utilisent un seul canal pour des raisons de conception et de prix de revient. Après PAMAS, plusieurs travaux ont été menés, avec pour objectif de réduire la consommation d'énergie par la réduction du temps d'oisiveté des capteurs.

Ces travaux ont conduit à proposer plusieurs protocoles à contention permettant des économies d'énergie. Pour réduire l'oisiveté des nœuds, les protocoles à activité cyclique ont été

proposés (*duty-cycled protocols*). Avec ces protocoles, le temps est découpé en cycles composés de périodes d'activité, pendant lesquels un nœud est actif c'est-à-dire qu'il peut émettre ou recevoir des trames de données, suivi par des périodes de sommeil pendant lesquelles son émetteur-transmetteur est coupé. Spécifier des périodes d'activité courtes réduit l'oisiveté des capteurs. Cependant l'activité des nœuds doit être organisée de façon à permettre les communications : un nœud actif ne peut pas communiquer avec un nœud endormi. On distingue donc deux sous-classes de protocoles à contention et à activité cyclique : les *protocoles synchronisés* et les protocoles *à préambules d'échantillonnages*.

1.3.1 Protocoles synchronisés

S-MAC. Parmi les protocoles à contention cyclique, le protocole Sensor-MAC, ou S-MAC, fait office de référence dans le domaine des RCSFs [YHE02]. Il a été proposé en 2002 par Wei Ye, John S. Heidemann et Deborah Estrin, pour réduire la consommation d'énergie causé le WiFi (norme IEEE 802.11 [IEE07]) aux dépends de la latence et de l'équité. S-MAC est basée sur IEEE 802.11, mais il apporte plusieurs innovations : l'activité cyclique des nœuds, la transmission des trames de données à la volée utilisant une seule séquence de trames de contrôle et l'auto-synchronisation des nœuds. Celle-ci est réalisée par l'échange régulier de trames SYNC en début de période active. Lorsqu'un nœud arrive dans un réseau, il cherche un cycle de synchronisation existant. Il s'y adapte s'il en trouve un. Sinon, il crée son propre cycle et il diffuse une trame SYNC à tous ces voisins. La synchronisation par l'échange de trames SYNC crée des groupes de nœuds synchronisés appelés des *clusters*. Les nœuds qui appartient à plusieurs groupes doivent être synchronisés avec les membres de chaque groupe. Les auteurs de S-MAC ont proposé en 2004, une version de SMAC avec écoute adaptative pour réduire la latence [YHE04].

T-MAC. L'objectif de Timeout-MAC ou T-MAC est de remédier aux inconvénients de S-MAC. T-MAC adapte l'intervalle de temps actif des nœuds à différents niveaux de trafic du réseau. Si le trafic est important, les nœuds restent actifs plus longtemps. Si au contraire le trafic est faible, la période active est brève pour économiser l'énergie. Les nœuds se mettent en veille lorsqu'ils détecte aucune activité pendant une durée appelée TA. Ainsi, si le trafic est en hausse, la période d'activité des nœuds augmente, alors que si le trafic est en baisse, la période d'activité baisse. Pour réduire la latence de la transmission multi-sauts, T-MAC propose d'utiliser une trame de contrôle FRTS (Future-RTS). Cette trame a pour but de signaler à un nœud qu'il est récepteur d'une transmission, alors que le transmetteur est dans la portée d'une transmission en cours et ne peut pas transmettre. Lorsqu'un nœud reçoit une trame FRTS, il devient actif à la fin de la transmission en cours pour pouvoir recevoir les trames de données. Ainsi le transmetteur précise le nœud récepteur qui doit rester actif après la transmission.

DMAC. L'intérêt majeur de ce protocole est d'offrir une latence très faible par rapport à d'autres protocoles MAC de la même famille [LKR04]. Pour ce faire, DMAC se base sur une topologie en arbre unidirectionnel. Une période active se divise en deux sous-période : une pour la réception et l'autre pour l'émission des trames. La période de réception d'un nœud correspond à la période de transmission de son fils, et la période de transmission d'un nœud correspond à la période de réception de son père. Ainsi, lorsqu'un nœud a reçu les données de son fils, il peut les transmettre immédiatement à son père. D-MAC requiert une forte synchronisation entre les nœuds. Par ailleurs, D-MAC souffre de quelques inconvénients. Lorsque la topologie change, les cycles d'activité des capteurs doit changer car leur position change dans l'arbre. De plus, les capteurs d'un même niveau dans l'arbre accèdent au canal en même temps ce qui peut accroître le taux de collisions.

1.3.2 Protocoles à préambule d'échantillonage

Une autre méthode pour permettre les communications entre nœuds qui sont soumis à des cycles d'activité est la méthode du *préambule d'échantillonnage*. Les nœuds s'endorment pour économiser l'énergie et ils se réveillent périodiquement pour écouter les éventuelles transmissions. S'ils ne détectent pas d'activité sur le réseau, ils se rendorment. S'ils détectent une activité, ils restent à l'écoute pour recevoir d'éventuelles trames de données. Tous les nœuds utilisent la même période d'échantillonnage qui est le temps entre deux écoutes consécutives. Mais, puisque ces nœuds ne sont pas synchronisés, l'émetteur doit connaitre l'instant ou le récepteur se réveille pour pouvoir recevoir les trames qu'il lui envoie. Des trames envoyées à un récepteur endormi sont perdues : c'est le problème de l'*envoi sourd*. Pour éviter ce problème, l'émetteur envoie avant chaque transmission un préambule contenant une suite non significative de bits 0 et 1. La longueur du préambule est supérieure à la période d'échantillonnage commune à tous les nœuds, pour que le récepteur perçoive l'activité sur le canal lorsqu'il se réveille. Ainsi il est alerté de l'imminence d'une transmission de données et il reste actif.

NP-CSMA-PS. La méthode à préambule d'échantillonnage semble adaptée aux réseaux ou le trafic est faible, en particulier les RCSFs où les nœuds sont chargés de détecter des événements rares. Le premier protocole MAC pour les RCSFs utilisant cette méthode est ALOHA-PS, proposé par A. El-Hoiydi [ElH02]. Par suite, cet auteur a proposé NP-CSMA-PS qui est une version écoute de porteuse CSMA non persistant avec préambule d'échantillonnage [EH02]. La combinaison des méthodes d'accès au médium ALOHA ou CSMA avec la méthode à préambule d'échantillonnage réduisent la consommation d'énergie. S'il le trafic est faible, les récepteurs ne consomment pas d'énergie entre les périodes d'échantillonnage. Plus la période d'échantillonnage est grande, plus les récepteurs économisent leur énergie. Mais la dépense énergétique est accrue pour les émetteurs car ils doivent transmettre un préambule plus long. D'une façon générale la technique à préambule d'échantillonnage permet d'économiser l'énergie des récepteurs mais reporte cette consommation sur les émetteurs. Elle est adaptée à un réseau à faible trafic.

WiseMAC. Ce protocole améliore NP-CSMA-PS en réduisant la longueur du préambule. Chaque nœud maintient une table contenant les périodes d'échantillonnage de tous ses voisins. Ces tables sont mises à jour par l'échange d'informations complémentaires véhiculées dans les trames d'acquittement échangées lors des communications (*piggybacking*), et non pas par l'échange de trames SYNC dédiées comme c'est le cas dans le cas des protocoles synchronisées. Un émetteur commence par transmettre en utilisant un préambule de longueur supérieure à la période d'échantillonnage. Lorsque la transmission est terminée, le récepteur acquitte les trames de données. L'acquittent contient une information supplémentaire qui est sa période d'échantillonnage. Après avoir reçu cet acquittement, l'émetteur met à jour l'entrée de sa table d'échantillonnages correspondant au nœud avec lequel il vient d'échanger. Ainsi, pour la prochaine transmission avec ce nœud, l'émetteur utilisera un préambule plus court qu'il transmettra juste avant le réveil du récepteur. L'utilisation d'un préambule même réduit reste nécessaire pour pallier au problèmes du décalage des horloges des nœuds. Ce décalage fait que les informations sur les temps d'échantillonnage contenues dans les tables ne correspondent plus nécessairement aux temps d'échantillonnages réels des nœuds. Wise-MAC peut donner de bonnes performances dans certains cas. Mais il a deux inconvénients majeurs : il est difficile à un nœud de diffuser une information (broadcast) car les nœuds ne sont pas synchronisés (c'est un problème général à la classe des protocoles à laquelle il appartient). De plus, WiseMAC n'utilise pas de trames de contrôle, et il ne peut pas éviter le *problème de la station cachée* des réseaux sans fil.

SCP-MAC *Synchronized Channel Polling MAC* (SCP-MAC) est un protocole hybride mêlant la technique de synchronisation entre les nœuds avec des périodes d'échantillonnage [YSH06]. Les nœuds se synchronisent de manière passive en ajoutant des informations dans les trames de données et d'une manière active en échangeant des trames SYNC entre voisins. SCP-MAC peut s'adapter à différents trafics. Lorsque le trafic est important, ce protocole augmente le temps d'accès des nœuds au canal de transmission pour augmenter le débit et réduire la latence de transmission multi-sauts.

B-MAC. B-MAC [PHC04] propose l'opération d'*évaluation de canal libre* (CCA, clear channel assessment) permettant de considérer qu'un canal bruité est libre. Si la puissance du signal reçu est supérieure au seuil du canal bruité, le canal est considéré occupé. En revanche, si du bruit est perçu sur le canal mais que sa puissance est inférieure au seuil du canal bruité, alors le canal est considéré libre. Cette technique aide les nœuds à économiser leur énergie en prenant en compte les conditions réelles d'utilisation du canal de communication. Ils ne se réveillent pas si le canal est occupé. Cependant le débit de communication est inversement proportionnel au seuil de canal bruité défini par le CCA. Si le seuil est élevé, le débit de communication est réduit et vice-versa. Ce seuil peut être modifié automatiquement en fonction des conditions d'utilisation du canal. Par ailleurs B-MAC utilise la méthode d'accès ALOHA avec préambule d'échantillonnage : c'est l'*écoute à faible énergie* (LPL pour low power listening). Les nœuds se réveillent périodiquement pour attendre une éventuelle transmission. Ils évaluent l'activité du canal grâce à CCA. Si le canal est occupé, ils restent à l'écoute pour recevoir les trames de données, sinon ils se rendorment.

1.4 Protocoles basés sur le multiplexage

Protocoles à multiplexage temporel. Ces protocoles évitent les collisions, et donc ils apportent un gain énergétique important. Mais ils nécessitent une phase de synchronisation plus délicate que les protocoles à contention, et par suite ils limitent la flexibilité du réseau. En effet, ces protocoles doivent implémenter un algorithme d'allocation des slots sur les différents nœuds du réseau ce qui impose une limitation sur le nombre de nœuds qui partagent le même canal étant donné que le nombre de *slots* dans une frame est fixe et ce qui augmente la complexité de gestion de la topologie. L'accès périodique des nœuds au canal dégrade les performances du réseau en latence et en débit. Parmi les protocoles implémentés par les RCSFs qui utilisent cette technique, citons Bluetooth [IEE02]. Mais une des raisons de gaspillage d'énergies des protocoles TDMA est l'attribution fixe des tranches de temps, et un nœuds peut se voir attribuer des tranches de temps même s'il n'en a pas besoin. Le protocole NAMA pour les réseaux autonomes [BGLA01], propose une attribution dynamique dans un rayon de deux sauts afin d'éviter le problème de la station cachée. Le protocole TRAMA [ROGLA03] réutilise l'idée d'attribution dynamique des tranches de temps proposée par NAMA, tout en réduisant le gaspillage d'énergie par l'écoute passive. Pour ce faire, le temps d'accès au canal est divisé en deux périodes : une période basée sur la contention et une période sans contention. Pendant la première période, les nœuds échangent leur adresse pour connaître la topologie du réseau dans un rayon de deux sauts. Pendant le deuxième période, l'accès se fait par multiplexage temporel et l'attribution des tranches de temps aux différents nœuds est faite par priorité. Les nœuds annoncent les intervalles de temps qu'ils possèdent et les nœuds récepteurs dans un paquet d'ordonnancement. L'écoute passive est évitée par le fait que les nœuds qui ne sont pas récepteurs s'endorment pendant les transmissions courantes. Cependant la proportion entre les deux périodes est fixe, ce qui implique un cycle de travail assez important chez les capteurs. Par ailleurs la priorité est établie par une fonction de hachage calculée par tous les capteurs ce qui peut être préjudiciable par rapport à la capacité de calcul des capteurs.

Protocoles à multiplexage fréquentiel. Ce qui explique que les protocoles FDMA ne sont pas de bons candidats pour les RCSFs, qu'il y ait peu de travaux dans ce domaine. Néanmoins E. Shih et al. ont proposé un protocole hybride entre TDMA et FDMA [SCI+01]. Les auteurs ont montré que des protocoles purement TDMA ou FDMA consomment plus d'énergie que des protocoles hybrides basés sur ces deux approches.

Protocoles à multiplexage de codes. L'inconvénient majeur de l'application de cette technique aux RCSFs est qu'il s'agit d'une technique centralisée, puisque le puits est le seul nœud qui distribue les codes aux autres nœuds du réseau. Néanmoins, nous pouvons citer le protocole CSMAC (*CDMA Sensor MAC*) qui combine les techniques FDMA et CDMA. Ce protocole est robuste par rapport au problème d'interférences. Il offre une transmission fiable avec un taux de réception des paquets élevé et une latence faible. Cependant, les nœuds doivent être dotés d'un émetteur-récepteur complexe avec plusieurs bandes de fréquence, le réseau a besoin d'un temps d'initialisation important et il ne peut pas être mobile.

1.5 Protocoles hybrides

Les protocoles hybrides combinent les techniques à contention et multiplexée (que ce soit à multiplexage temporel ou à multiplexage fréquentiel, ou les deux). Ces protocoles améliorent les performances du réseau et apportent des gains en énergie. Z-MAC, PMAC, EcoMAC sont des protocoles hybrides.

1.6 Routage dynamique dans les réseaux de capteurs sans fil

Le *routage dynamique* consiste à trouver des routes entre deux nœuds du réseau et à être capable de modifier cette route pour répondre à la dynamique du réseau de manière à ce que l'information soit toujours correctement aiguillée pour arriver à bon port.

Partant des protocoles de routage de l'IP existant, les travaux du groupe MANET (*Mobile Ad-hoc NETwork*)[1] ont proposé des extensions qui tiennent compte de la mobilité des nœuds. Ils se sont fondés sur principe fondamentaux du routage que sont routage par inondation, routage à vecteur de distance, routage à la source et état du lien. Parallèlement deux grands familles de protocoles se sont formés à partir de la normalisation MANET : les protocoles réactifs et les protocoles proactifs.

Les *protocoles réactifs* établissent une route entre une source et une destination à la demande. La route est déterminée par inondation de paquets. La source envoie une requête initiale auprès de tous ces voisins, qui à leur tour répète la manœuvre auprès de leurs propres voisins. Cette manière de procéder fait que le destinataire recueille le paquet en plusieurs exemplaires de la part des nœuds limitrophes. Le paquet consomme donc une plus grande quantité de ressources pour découvrir une simple route entre deux points du réseau. Parmi les protocoles réactifs, nous trouvons DSR, AODV.

Les protocoles de routage *proactifs* n'établissent pas les routes à la demande mais à l'avance. Ils ne sont basés sur la technique de l'inondation de paquets, mais sur l'établissement de tables de routage par les nœuds par l'échange périodique de messages de contrôles entre voisins (appelés des balises). Ces tables (dynamiques) sont donc établies avant qu'une demande soit effective. Elles permettent de déterminer la route optimale qu'un paquet en transit entre sa source et sa destination doit emprunter. Un protocole proactif identifie la

[1] MANET est un groupe de travail issu de l'IETF (Internet Engineering Task Force) qui œuvre depuis les années 1990 à la standardisation des protocoles ad hoc fonctionnant sous IP.

topologie du réseau. Parmi cette famille, nous trouvons les protocoles DSDV, OLSR.

Les protocoles de routage *hybrides* combinent les techniques des protocoles réactifs et celles des protocoles réactifs pour essayer d'hériter de leurs avantages. Les protocoles ZRP, ZHLS et LANMAR sont des exemples de protocoles hybrides.

On distingue encore les protocoles de routage *hiérarchiques* se basent sur la structure en grappe (clusters) des réseaux. Les nœuds d'une même grappe envoient leurs paquets à un nœud particulier qui est en charge du routage des paquets vers leur destination. Dans cette famille, nous trouvons les protocoles CGSR, FSR, HSR, CBRP, etc. Les protocoles de routage *géographiques* utilisent la localisation des nœuds tel que le GPS (*Global Positioning System*). Une source désirant émettre un paquet a la connaissance de sa propre localisation, de celles de ses voisins, et de celle du destinataire. Parmi ces protocoles, nous trouvons GeoCast, LAR, DREAM, GPSR, etc. Les protocoles de routage basés sur la *consommation d'énergie* basent l'établissement des routes sur les réserves énergétiques des nœuds. Les protocoles ISAIAH, PARO, EADSR, PAMAS, sont des éléments de cette famille de protocoles.

DSDV. Le protocole DSDV (*Destination-Sequenced Distance-Vector*) est l'un des premiers à être mis au point par le groupe MANET. Largement inspiré par le protocole RIP (Routing Information Protocol) des réseaux filaires, il est aujourd'hui abandonné à causes de ses défauts. DSDV repose sur un vecteur de distance. Chaque nœud possède une table de routage où chaque entrée doit identifier une destination possible, le nombre de sauts pour y parvenir ainsi que le nœud voisin à traverser. Le principal défaut de DSDV (qui est hérité de RIP) réside dans la convergence des tables de routage. Dans RIP, chaque routeur communique sa table de routage à ses voisins. En recevant une table, le routeur compare les entrées de sa table avec celle qu'on lui propose. Grâce à ces calculs, il choisit la route la plus courte pour chaque destination. Cette route de converge pas toujours vers un nombre de saut déterminé, surtout si panne chez un routeur voisin. Les routes deviennent vite égales à l'infini : un seuil est fixé à 16 sauts dans RIP, signifiant que le nœud n'est pas atteignable ou qu'il est en panne. Dans un réseau autonome mobile, le problèmes est d'autant plus important que la mobilité des nœuds ralentit cette convergence. Pour pallier à ce défaut majeur, DSDV insère des numéros de séquence destinés à rafraichir les tables de routage, solution qui interdit toute mise à jour antérieure, le numéro de séquence risquant d'être inférieur au dernier numéro enregistré. Le surplus de signalisation est une autre imperfection de DSDV. En présence de nœuds peu mobiles, les messages de rafraichissement sont moins intéressants. C'est pour pallier à cette insuffisance que les auteurs de DSDV ont créé un nouveau protocole, AODV (*Ad hoc On demand Distance Vector*) qui supprime tous les paquets de contrôle.

DSR. *Dynamic Source Routing* (DSR) est un protocole réactif qui se base sur le principe de routage par la source dans lequel les nœuds n'ont pas besoin de table de routage. Lorsqu'une source souhaite initier un flux vers une destination, elle procède à l'inondation d'une requête spéciale appelée *Route Request*. La reception de la requête est particulière parce que le nœud inscrit son identificateur dans l'en-tête avant de la diffuser à son tour. La Route Request atteint le destinataire en plusieurs exemplaires via différentes routes. La destination sélectionne la route la plus appropriée, généralement la plus courte, pour que le reste du flux arrive à bon port. Une fois qu'elle a établi la trajectoire finale elle l'indique à la source en lui adressant une requête appelée Route Reply. L'ensemble de cette procédure permet à la source d'enregistrer le cheminement de la requête pour le conserver à des fins ultérieures. En possession de cette information, la route la retranscrit dans tous les paquets, puis la boucle se referme. Les nœuds intermédiaires n'ont plus qu'à accueillir les paquets et à les renvoyer

en fonction des instructions inscrites dans l'en-tête. Les liens asymétriques constituent pourtant la faiblesse de ce protocole, car la Route Reply ne peut pas prendre en sens inverse le sillage laissé par la Route Request. Pour revenir à la source, il faut de nouveau procéder par inondation.

AODV. Le protocole AODV (*Ad hoc On-demand Distance Vector*) est un protocole de routage réactif pour les réseaux autonomes. Ce protocole fonctionne sans qu'il soit besoin d'échanger de la signalisation entre voisins pour mettre à jour les tables de routage. Mais, à la demande d'un routage, une table peut être implantée dans chaque nœud. Pour installer cette table, la source diffuse par inondation une requête de route (RREQ, *Route Request*) de la même manière qu'elle l'a fait pour DSR. Tous les nœuds, y compris la destination recevant la RREQ, ménagent une entrée dans leur table locale pour l'orientation du flux. La destination répond à la source en transmettant une réponse de route (RREP, *Route Reply*). Ainsi tous les paquets sont aiguillés d'après les tables locales construites dans chaque nœud, originellement à la demande de la source. A chaque passage dans un nœud relais, ce dernier enregistre la route que la requête a empruntée depuis la source. La topologie peut se modifier provoquant une coupure d'un lien radio. Le nœud victime de cette rupture avertit la source en lui délivrant un message d'erreur. Cette situation permet à la source d'effectuer une reprise, ce qui l'amène à découvrir une nouvelle route.

OLSR. OLSR (*Optimized Link State Routing*) est un protocole proactif. Il applique dans un contexte ad hoc les règles de routage, cette fois, centrées sur l'état du lien. L'état du lien permet à chaque nœud de connaitre parfaitement la position des autres dans le réseau. Ainsi choisir le chemin le plus court ou le plus rapide est simple. Chaque nœud est à même d'appréhender dans sa globalité la topologie du réseau et possède des tables de routage lui permettant de déterminer la route à l'aide de l'algorithme de Dijkstra. L'inondation n'a pas lieu d'être, puisqu'elle génèrerait de la redondance. Les paquets doivent parvenir aux nœuds d'une seule traite. Pour cela le nœud élit parmi ses proches un représentant qui endosse le rôle de relais multi-points ou MPR (Multi-Point Relay) ayant pour mission d'acheminer l'information. La sélection des MPR se fait à partir des messages "Hello" que les nœuds s'échangent mutuellement pour déduire la nature des liens, symétrique ou asymétrique, qui les relient. La condition pour devenir MPR est de pouvoir atteindre, avec un lien symétrique, tous les nœuds à une distance de deux sauts. Les MRP sont ensuite communiqués à tout le réseau par des messages TC (Topology Control) périodiques. A réception des TC, les nœuds mettent à jour leur table de routage.

1.7 Conclusion sur les RCSFs

L'ordonnancement des capteurs selon des courtes périodes d'activité suivies par de longues périodes de sommeil, entraîne la nécessité de synchroniser les capteurs entre eux. En effet, un capteur endormi est déconnecté du réseau global dans le sens où il ne peut pas relayer une information entre un nœud émetteur et son destinataire. Il est donc nécessaire d'éviter qu'un groupe de capteurs endormis se retrouve isolé du reste du réseau. C'est la raison pour laquelle les capteurs sont régulièrement synchronisés afin que les périodes d'activité et de sommeil d'un même groupe de capteurs (on parle de *cluster*) soient les mêmes. La synchronisation impose des contraintes fortes sur les capteurs situés à la frontière de plusieurs clusters.

Certains capteurs captent la grandeur physique et la gardent afin d'agréger l'information mesurée qui sera envoyée plus tard pour réduire la consommation d'énergie. L'agrégation, la synchronisation et la manière de relayer l'information doivent permettre d'optimiser la durée de vie du réseau qui est intimement liée au nombre de nœuds considérés comme finis

car leur batterie est épuisée. Les puits, qui possèdent des capacités supérieures en terme de traitement, de capacité de mémoire et d'autonomie d'énergie.

Le déploiement des capteurs qui peut être aléatoire dans la zone de captage, leur disponibilité (pannes diverses, périodes active ou sommeil), la mobilité des nœuds du réseau mais aussi les particularités géographiques changeantes de la zone de déploiement, font que la topologie d'un RCSF est une topologie dynamique généralement basée sur le maillage complet au moins entre les nœuds d'un même groupe.

Annexe D

Le code PRISM du cas hospitalier

1 Les constantes utilisées

```
ctmc // le modèle mobile est traduit en chaîne de Markov à temps continu

const int a      = 1;
const int b      = 2;
const int c      = 3;
const int d      = 4;
const int e      = 5;
const int alarm1 = 6;
const int alarm2 = 7;
const int ack1   = 8;
const int ack2   = 9;
const int nack   = 10;
const int s1r1   = 11;
const int s1r2   = 12;
const int s2r3   = 13;
const int inN1   = 14;
const int outN1  = 15;
const int inN2   = 16;
const int outN2  = 17;
const int inR1   = 18;
const int outR1  = 19;
const int inR2   = 20;
const int outR2  = 21;
const int inR3   = 22;
const int outR3  = 23;
const int s1se   = 24;
const int s2se   = 25;

const int res = 1000; // valeur choisie arbitrairement

const double rate_a             = 1.75; // 1 alarme toutes les 4 secondes en moyenne
const double rate_b             = 1.75;
const double rate_c             = 1.75;
const double rate_d             = 1.75;
```

```
const double rate_e             = 1.75;
const double rate_s1r1          = 1.75;
const double rate_s1r2          = 1.75;
const double rate_s2r3          = 1.75;
const double rate_contention_1  = 1.75;
const double rate_contention_2  = 1.75;
const double rate_backoff_R1    = 1.75;
const double rate_backoff_R2    = 1.75;
const double rate_s1se          = 4; // nombre d'essai maximal est 16 (IEEE 802.3)
const double rate_s2se          = 4;
const double rate_backoff_Si1   = 4;
const double rate_backoff_Si2   = 4;

// taux de traitement au niveau du serveur
const double rate_processing    = 0.5;

// taux de mouvements des nœuds mobiles
const double rate_inN1   = 1.4;
const double rate_outN1  = 1.4;
const double rate_inN2   = 1.4;
const double rate_outN2  = 1.4;
const double rate_inR1   = 1.4;
const double rate_outR1  = 1.4;
const double rate_inR2   = 1.4;
const double rate_outR2  = 1.4;
const double rate_inR3   = 1.4;
const double rate_outR3  = 1.4;
```

2 Le nœud mobile N_1

```
module N1
s1: [1..11] init 1;
   // N1 a 3 noms liés : va, v1b, et x1
   v1a : [0..25] init 0;
   v1b : [0..25] init 0;
   x1  : [0..25] init 0;

   [contention1]    (s1=1)                        -> rate_contention_1: (s1'=2);
   [a_N1_R1_alarm1] (s1=2)                        -> rate_a: (s1'=3);
   [a_R1_N1_v3a]    (s1=3) & (v3a=ack1)           -> (s1'=11) & (v1a'=v3a);
   [a_R1_N1_v3a]    (s1=3) & (v3a!=ack1)          -> (s1'=1) & (v1a'=v3a);
   [a_R1_N1_nack]   (s1=3)                        -> (s1'=1) & (v1a'=nack);

   [inN1_Mob_N1_c]  (s1=1) & (mask=c)             -> (s1'=4) & (x1'=c);
   [outN1_Mob_N1_c] (s1=4) & (mask=res-c)         -> (s1'=1) & (x1'=c);

   [contention1]    (s1=4)                        -> rate_contention_1: (s1'=5);
   [a_N1_R1_alarm1] (s1=5)                        -> rate_a: (s1'=6);
   [a_R1_N1_v3a]    (s1=6) & (v3a=ack1)           -> (s1'=7) & (v1a'=v3a);
   [a_R1_N1_v3a]    (s1=6) & (v3a!=ack1)          -> (s1'=9) & (v1a'=v3a);
```

```
   [c_N1_R2_alarm1] (s1=7) & (x1=c)                -> rate_c: (s1'=8);
   [c_R2_N1_v4b]    (s1=8) & (x1=c)                -> (s1'=11) & (v1b'=v4b);
   [c_R2_N1_nack]   (s1=8) & (x1=c)                -> (s1'=11) & (v1b'=nack);

   [a_R1_N1_nack]   (s1=6)                         -> (s1'=9) & (v1a'=nack);
   [c_N1_R2_alarm1] (s1=9)  & (x1=c)               -> rate_c: (s1'=10);
   [c_R2_N1_v4b]    (s1=10) & (x1=c) & (v4b=ack1)  -> (s1'=11) & (v1b'=v4b);
   [c_R2_N1_v4b]    (s1=10) & (x1=c) & (v4b!=ack1) -> (s1'=4) & (v1b'=v4b);
   [c_R2_N1_nack]   (s1=10) & (x1=c)               -> (s1'=4) & (v1b'=nack);

   [end] (s1=11) -> true; // l'alarme a été envoyée et son accusé reçu
endmodule
```

3 Le nœud mobile N_2

```
module N2
  s2: [1..18] init 1;
  // N2 has 3 bound names : v2a, v2b, et x2
  v2a : [0..25] init 0;
  v2b : [0..25] init 0;
  x2  : [0..25] init 0;

  [contention2]    (s2=1)                         -> rate_contention_2: (s2'=2);
  [b_N2_R2_alarm2] (s2=2)                         -> rate_b: (s2'=3);
  [b_R2_N2_v4a]    (s2=3)  & (v4a=ack2)           -> (s2'=18) & (v2a'=v4a);
  [b_R2_N2_v4a]    (s2=3)  & (v4a!=ack2)          -> (s2'=1) & (v2a'=v4a);
  [b_R2_N2_nack]   (s2=3)                         -> (s2'=1) & (v2a'=nack);

  [inN2_Mob_N2_d]  (s2=1)  & (mask=d)             -> (s2'=4) & (x2'=d);
  [outN2_Mob_N2_d] (s2=4)  & (mask=res-d)         -> (s2'=1) & (x2'=d);
  [contention2]    (s2=4)                         -> rate_contention_2: (s2'=5);

  [b_N2_R2_alarm2] (s2=5)                         -> rate_b: (s2'=6);
  [b_R2_N2_v4a]    (s2=6)  & (v4a=ack2)           -> (s2'=7) & (v2a'=v4a);
  [b_R2_N2_v4a]    (s2=6)  & (v4a!=ack2)          -> (s2'=9) & (v2a'=v4a);
  [d_N2_R1_alarm2] (s2=7)  & (x2=d)               -> rate_d: (s2'=8);
  [d_R1_N2_v3b]    (s2=8)  & (x2=d)               -> (s2'=18) & (v2b'=v3b);
  [d_R1_N2_nack]   (s2=8)  & (x2=d)               -> (s2'=18) & (v2b'=nack);

  [b_R2_N2_nack]   (s2=6)                         -> (s2'=9) & (v2a'=nack);
  [d_N2_R1_alarm2] (s2=9)  & (x2=d)               -> rate_d: (s2'=10);
  [d_R1_N2_v3b]    (s2=10) & (x2=d) & (v3b=ack2)  -> (s2'=18) & (v2b'=v3b);
  [d_R1_N2_v3b]    (s2=10) & (x2=d) & (v3b!=ack2) -> (s2'=4) & (v2b'=v3b);
  [d_R1_N2_nack]   (s2=10) & (x2=d)               -> (s2'=4) & (v2b'=nack);

  [inN2_Mob_N2_e]  (s2=1)  & (mask=e)             -> (s2'=11) & (x2'=e);
  [outN2_Mob_N2_e] (s2=10) & (mask=res-e)         -> (s2'=1)  & (x2'=e);
  [contention2]    (s2=11)                        -> rate_contention_2: (s2'=12);
  [b_N2_R2_alarm2] (s2=12)                        -> rate_b: (s2'=13);
  [b_R2_N2_v4a]    (s2=13) & (v4a=ack2)           -> (s2'=14) & (v2a'=v4a);
```

```
  [b_R2_N2_v4a]    (s2=13) & (v4a!=ack2)          -> (s2'=16) & (v2a'=v4a);
  [e_N2_R3_alarm2] (s2=14) & (x2=e)               -> rate_e: (s2'=15);
  [e_R3_N2_v5a]    (s2=15) & (x2=e)               -> (s2'=18) & (v2b'=v5a);
  [b_R2_N2_nack]   (s2=13)                        -> (s2'=16) & (v2a'=nack);
  [e_N2_R3_alarm2] (s2=16) & (x2=e)               -> rate_e: (s2'=17);
  [e_R3_N2_v5a]    (s2=17) & (x2=e) & (v5a=ack2)  -> (s2'=18) & (v2b'=v5a);
  [e_R3_N2_v5a]    (s2=17) & (x2=e) & (v5a!=ack2) -> (s2'=11) & (v2b'=v5a);

  [end] (s2=18) -> true; // l'alarme a été envoyée et son accusé reçu
endmodule
```

4 Le routeur R_1

```
module R1
  s3: [1..27] init 1;
  // R1 a 5 noms liés : u3a, u3b, x3, v3a, et v3b
  u3a : [0..25] init 0;
  u3b : [0..25] init 0;
  x3  : [0..25] init 0;
  v3a : [0..25] init 0;
  v3b : [0..25] init 0;

  [a_N1_R1_alarm1]   (s3=1)                -> (s3'=2) & (u3a'=alarm1);
  [s1r1_R1_Si1_u3a]  (s3=2)                -> rate_s1r1: (s3'=3);
  [s1r1_Si1_R1_v6a]  (s3=3)                -> (s3'=4) & (v3a'=v6a);
  [s1r1_Si1_R1_nack] (s3=3)                -> (s3'=22) & (v3a'=nack);
  [backoffR1]        (s3=22)               -> rate_backoff_R1: (s3'=2);
  [a_R1_N1_v3a]      (s3=4)                -> rate_a: (s3'=1);

  [inR1_Mob_R1_d]    (s3=1) & (mask=d)     -> (s3'=5) & (x3'=d);
  [outR1_Mob_R1_d]   (s3=5) & (mask=res-d) -> (s3'=1) & (x3'=d);

  [a_N1_R1_alarm1]   (s3=5)                -> (s3'=6) & (u3a'=alarm1);
  [s1r1_R1_Si1_u3a]  (s3=6)                -> rate_s1r1: (s3'=7);
  [s1r1_Si1_R1_v6a]  (s3=7)                -> (s3'=8) & (v3a'=v6a);
  [s1r1_Si1_R1_nack] (s3=7)                -> (s3'=26) & (v3a'=nack);
  [backoffR1]        (s3=26)               -> rate_backoff_R1: (s3'=6);
  [a_R1_N1_v3a]      (s3=8)                -> rate_a: (s3'=5);
  [d_N2_R1_alarm2]   (s3=6) & (x3=d)       -> (s3'=9) & (u3b'=alarm2);
  [a_R1_N1_nack]     (s3=9)                -> rate_a: (s3'=10);
  [d_R1_N2_nack]     (s3=10) & (x3=d)      -> rate_d: (s3'=5);

  [d_N2_R1_alarm2]   (s3=5) & (x3=d)       -> (s3'=11) & (u3b'=alarm2);
  [s1r1_R1_Si1_u3b]  (s3=11)               -> rate_s1r1: (s3'=12);
  [s1r1_Si1_R1_v6a]  (s3=12)               -> (s3'=13) & (v3b'=v6a);
  [s1r1_Si1_R1_nack] (s3=12)               -> (s3'=27) & (v3b'=nack);
  [backoffR1]        (s3=27)               -> rate_backoff_R1: (s3'=11);
  [d_R1_N2_v3b]      (s3=13) & (x3=d)      -> rate_a: (s3'=5);
  [a_N1_R1_alarm1]   (s3=11)               -> (s3'=14) & (u3a'=alarm1);
  [d_R1_N2_nack]     (s3=14) & (x3=d)      -> rate_d: (s3'=15);
```

```
  [a_R1_N1_nack]     (s3=15)                -> rate_a: (s3'=5);
endmodule
```

5 Le routeur R_2

```
module R2
  s4: [1..27] init 1;
  // R2 has 5 bound names: u4a, u4b, x4, v4a, et v4b
  u4a : [0..25] init 0;
  u4b : [0..25] init 0;
  x4  : [0..25] init 0;
  v4a : [0..25] init 0;
  v4b : [0..25] init 0;

  [b_N2_R2_alarm2]   (s4=1)                 -> (s4'=2) & (u4a'=alarm2);
  [s1r2_R2_Si1_u4a]  (s4=2)                 -> rate_s1r2: (s4'=3);
  [s1r2_Si1_R2_v6b]  (s4=3)                 -> (s4'=4) & (v4a'=v6b);
  [s1r2_Si1_R2_nack] (s4=3)                 -> (s4'=22) & (v4a'=nack);
  [backoffR2]        (s4=22)                -> rate_backoff_R2: (s4'=2);
  [b_R2_N2_v4a]      (s4=4)                 -> rate_b: (s4'=1);

  [inR2_Mob_R2_c]    (s4=1) & (mask=c)      -> (s4'=5) & (x4'=c);
  [outR2_Mob_R2_c]   (s4=5) & (mask=res-c)  -> (s4'=1) & (x4'=c);

  [b_N2_R2_alarm2]   (s4=5)                 -> (s4'=6) & (u4a'=alarm2);
  [s1r2_R2_Si1_u4a]  (s4=6)                 -> rate_s1r2: (s4'=7);
  [s1r2_Si1_R2_v6b]  (s4=7)                 -> (s4'=8) & (v4a'=v6b);
  [s1r2_Si1_R2_nack] (s4=7)                 -> (s4'=26) & (v4a'=nack);
  [backoffR2]        (s4=26)                -> rate_backoff_R2: (s4'=6);
  [b_R2_N2_v4a]      (s4=8)                 -> rate_b: (s4'=5);
  [c_N1_R2_alarm1]   (s4=6) & (x4=c)        -> (s4'=9) & (u4b'=alarm1);
  [b_R2_N2_nack]     (s4=9)                 -> rate_b: (s4'=10);
  [c_R2_N1_nack]     (s4=10) & (x4=c)       -> rate_c: (s4'=5);

  [c_N1_R2_alarm1]   (s4=5) & (x4=c)        -> (s4'=11) & (u4b'=alarm1);
  [s1r2_R2_Si1_u4b]  (s4=11)                -> rate_s1r2: (s4'=12);
  [s1r2_Si1_R2_v6b]  (s4=12)                -> (s4'=13) & (v4b'=v6b);
  [s1r2_Si1_R2_nack] (s4=12)                -> (s4'=27) & (v4b'=nack);
  [backoffR2]        (s4=27)                -> rate_backoff_R2: (s4'=11);
  [c_R2_N1_v4b]      (s4=13) & (x4=c)       -> rate_c: (s4'=5);
  [b_N2_R2_alarm2]   (s4=11)                -> (s4'=14) & (u4a'=alarm2);
  [c_R2_N1_nack]     (s4=14) & (x4=c)       -> rate_c: (s4'=15);
  [b_R2_N2_nack]     (s4=15)                -> rate_b: (s4'=5);
endmodule
```

6 Le routeur R_3

```
module R3
  s5: [1..5] init 1;
  // R3 a 3 noms liés : u5a, v5a, x5
```

```
  u5a: [0..25] init 0;
  v5a: [0..25] init 0;
  x5 : [0..25] init 0;

  [inR3_Mob_R3_e]   (s5=1) & (mask=e)       -> (s5'=2) & (x5'=e);
  [outR3_Mob_R3_e]  (s5=2) & (mask=res-e)   -> (s5'=1) & (x5'=e);
  [e_N2_R3_alarm2]  (s5=2) & (x5=e)         -> (s5'=3) & (u5a'=alarm2);
  [s2r3_R3_Si2_u5a] (s5=3)                  -> rate_s2r3: (s5'=4);
  [s2r3_Si2_R3_v7a] (s5=4)                  -> (s5'=5) & (v5a'=v7a);
  [e_R3_N2_v5a]     (s5=5) & (x5=e)         -> rate_e: (s5'=2);
endmodule
```

7 La station de base Si_1

```
module Si1
  s6 : [1..17] init 1;
  // Si1 a 4 noms liés : u6a, u6b, v6a et v6b
  u6a : [0..25] init 0;
  u6b : [0..25] init 0;
  v6a : [0..25] init 0;
  v6b : [0..25] init 0;

  [s1r1_R1_Si1_u3a]  (s6=1)   -> (s6'=2) & (u6a'=u3a);
  [s1r1_R1_Si1_u3b]  (s6=1)   -> (s6'=2) & (u6a'=u3b);
  [s1se_Si1_Se_u6a]  (s6=2)   -> rate_s1se: (s6'=3);
  [s1se_Se_Si1_ack1] (s6=3)   -> (s6'=4) & (v6a'=ack1);
  [s1se_Se_Si1_ack2] (s6=3)   -> (s6'=4) & (v6a'=ack2);
  [s1se_Se_Si1_nack] (s6=3)   -> (s6'=12) & (v6a'=nack);
  [backoffSi1]       (s6=12)  -> rate_backoff_Si1: (s6'=2);
  [s1r1_Si1_R1_v6a]  (s6=4)   -> rate_s1r1: (s6'=1);
  [s1r2_R2_Si1_u4a]  (s6=2)   -> (s6'=5) & (u6b'=u4a);
  [s1r2_R2_Si1_u4b]  (s6=2)   -> (s6'=5) & (u6b'=u4b);
  [s1r1_Si1_R1_nack] (s6=5)   -> rate_s1r1: (s6'=6);
  [s1r2_Si1_R2_nack] (s6=6)   -> rate_s1r2: (s6'=1);

  [s1r2_R2_Si1_u4a]  (s6=1)   -> (s6'=7) & (u6b'=u4a);
  [s1r2_R2_Si1_u4b]  (s6=1)   -> (s6'=7) & (u6b'=u4b);
  [s1se_Si1_Se_u6b]  (s6=7)   -> rate_s1se: (s6'=8);
  [s1se_Se_Si1_ack1] (s6=8)   -> (s6'=9) & (v6b'=ack1);
  [s1se_Se_Si1_ack2] (s6=8)   -> (s6'=9) & (v6b'=ack2);
  [s1se_Se_Si1_nack] (s6=8)   -> (s6'=17) & (v6b'=nack);
  [backoffSi1]       (s6=17)  -> rate_backoff_Si1: (s6'=7);
  [s1r2_Si1_R2_v6b]  (s6=9)   -> rate_s1r2: (s6'=1);
  [s1r1_R1_Si1_u3a]  (s6=7)   -> (s6'=10) & (u6a'=u3a);
  [s1r1_R1_Si1_u3b]  (s6=7)   -> (s6'=10) & (u6a'=u3b);
  [s1r2_Si1_R2_nack] (s6=10)  -> rate_s1r2: (s6'=11);
  [s1r1_Si1_R1_nack] (s6=11)  -> rate_s1r2: (s6'=1);
endmodule
```

8 La station de base Si_2

```
module Si2
  s7 : [1..5] init 1;
  // Si2 a 2 noms liés : u7a et v7a
  u7a : [0..25] init 0;
  v7a : [0..25] init 0;

  [s2r3_R3_Si2_u5a]    (s7=1) -> (s7'=2) & (u7a'=u5a);
  [s2se_Si2_Se_u7a]    (s7=2) -> rate_s2se: (s7'=3);
  [s2se_Se_Si2_ack2]   (s7=3) -> (s7'=5) & (v7a'=ack2);
  [s2se_Se_Si2_nack]   (s7=3) -> (s7'=4) & (v7a'=nack);
  [backoffSi2]         (s7=4) -> rate_backoff_Si2: (s7'=2);
  [s2r3_Si2_R3_v7a]    (s7=5) -> rate_s2r3: (s7'=1);
endmodule
```

9 Le serveur Se

```
module Se
  s8 : [1..17] init 1;
  // Se a 2 noms liés : u8a et u8b
  u8a : [0..25] init 0;
  u8b : [0..25] init 0;

  [s1se_Si1_Se_u6a]     (s8=1)                  -> (s8'=2) & (u8a'=u6a);
  [processing]          (s8=2) & (u8a=alarm1)   -> rate_processing: (s8'=3);
  [s1se_Se_Si1_ack1]    (s8=3)                  -> rate_s1se: (s8'=1);
  [processing]          (s8=2) & (u8a=alarm2)   -> rate_processing: (s8'=4);
  [s1se_Se_Si1_ack2]    (s8=4)                  -> rate_s1se: (s8'=1);
  [s2se_Si2_Se_u7a]     (s8=2)                  -> (s8'=5) & (u8b'=u7a);
  [s1se_Se_Si1_nack]    (s8=5)                  -> rate_s1se: (s8'=6);
  [s2se_Se_Si2_nack]    (s8=6)                  -> rate_s2se: (s8'=1);

  [s1se_Si1_Se_u6b]     (s8=1)                  -> (s8'=7) & (u8a'=u6b);
  [processing]          (s8=7) & (u8a=alarm1)   -> rate_processing: (s8'=8);
  [s1se_Se_Si1_ack1]    (s8=8)                  -> rate_s1se: (s8'=1);
  [processing]          (s8=7) & (u8a=alarm2)   -> rate_processing: (s8'=9);
  [s1se_Se_Si1_ack2]    (s8=9)                  -> rate_s1se: (s8'=1);
  [s2se_Si2_Se_u7a]     (s8=7)                  -> (s8'=10) & (u8b'=u7a);
  [s1se_Se_Si1_nack]    (s8=10)                 -> rate_s1se: (s8'=11);
  [s2se_Se_Si2_nack]    (s8=11)                 -> rate_s2se: (s8'=1);

  [s2se_Si2_Se_u7a]     (s8=1)                  -> (s8'=12) & (u8b'=u7a);
  [processing]          (s8=12) & (u7a=alarm2)  -> rate_processing: (s8'=13);
  [s2se_Se_Si2_ack2]    (s8=13)                 -> rate_s2se: (s8'=1);
  [s1se_Si1_Se_u6a]     (s8=12)                 -> (s8'=14) & (u8a'=u6a);
  [s2se_Se_Si2_nack]    (s8=14)                 -> rate_s2se: (s8'=15);
  [s1se_Se_Si1_nack]    (s8=15)                 -> rate_s1se: (s8'=1);
  [s1se_Si1_Se_u6b]     (s8=12)                 -> (s8'=16) & (u8a'=u6b);
  [s2se_Se_Si2_nack]    (s8=16)                 -> rate_s2se: (s8'=17);
```

```
  [s1se_Se_Si1_nack]    (s8=17)                  -> rate_s1se: (s8'=1);
endmodule
```

10 Les modèles de mobilité

N.B. Un seul des modules ci-dessous est inclus dans le modèle PRISM global. L'usage de masques de commandes PRISM permet de remplacer un modèle de mobilité par un autre sans avoir à modifier le modèle global qui est constitué par le modèle des communications des nœuds, et un modèle ci-dessous de leurs mobilités.

```
module Mob
  // Mob = Mob0
  smob: [0..1] init 1;
  mask: [0..res-a] init 0;

  [] (smob=1) -> (smob'=1);
endmodule

module Mob
  // Mob = MobC
  smob: [1..4] init 1;
  mask:[0..res-a] init 0;

  [] (mask=0) -> (mask'=c);
  [inR2_Mob_R2_c]  smob=1 -> rate_inR2:  (smob'=2);
  [inN1_Mob_N1_c]  smob=2 -> rate_inN1:  (smob'=3) & (mask'=res-c);
  [outN1_Mob_N1_c] smob=3 -> rate_outN1: (smob'=4);
  [outR2_Mob_R2_c] smob=4 -> rate_outR2: (smob'=1) & (mask'=0);
endmodule

module Mob
  // Mob = MobD
  smob: [1..4] init 1;
  mask:[0..res-a] init 0;

  [] (mask=0) -> (mask'=d);
  [inR1_Mob_R1_d]  smob=1 -> rate_inR1: (smob'=2);
  [inN2_Mob_N2_d]  smob=2 -> rate_inN2: (smob'=3) & (mask'=res-d);
  [outN2_Mob_N2_d] smob=3 -> rate_outN2: (smob'=4);
  [outR1_Mob_R1_d] smob=4 -> rate_outR1: (smob'=1) & (mask'=0);
endmodule

module Mob
  // Mob = MobE
  smob: [1..4] init 1;
  mask: [0..res-a] init 0;

  [] (mask=0) -> (mask'=e);
  [inR3_Mob_R3_e]  smob=1 -> rate_inR3: (smob'=2);
  [inN2_Mob_N2_e]  smob=2 -> rate_inN2: (smob'=3) & (mask'=res-e);
```

```
  [outN2_Mob_N2_e] smob=3 -> rate_outN2: (smob'=4);
  [outR3_Mob_R3_e] smob=4 -> rate_outR3: (smob'=1) & (mask'=0);
endmodule

module Mob
  // Mob = MobF
  smob: [1..8] init 1;
  mask: [0..res-a] init 0;

  [] (mask=0) -> (mask'=c);
  [inR2_Mob_R2_c]  smob=1 -> rate_inR2: (smob'=2);
  [inN1_Mob_N1_c]  smob=2 -> rate_inN1: (smob'=3) & (mask'=res-c);
  [outN1_Mob_N1_c] smob=3 -> rate_outN1: (smob'=4);
  [outR2_Mob_R2_c] smob=4 -> rate_outR2: (smob'=5) & (mask'=d);

  [inR1_Mob_R1_d]  smob=5 -> rate_inR1: (smob'=6);
  [inN2_Mob_N2_d]  smob=6 -> rate_inN2: (smob'=7) & (mask'=res-d);
  [outN2_Mob_N2_d] smob=7 -> rate_outN2: (smob'=8);
  [outR1_Mob_R1_d] smob=8 -> rate_outR1: (smob'=1) & (mask'=0);
endmodule

module Mob
  // Mob = MobG
  smob: [1..8] init 1;
  mask: [0..res-a] init 0;

  [] (mask=0) -> (mask'=c);
  [inR2_Mob_R2_c] smob=1 -> rate_inR2: (smob'=2);
  [inN1_Mob_N1_c] smob=2 -> rate_inN1: (smob'=3) & (mask'=d) ;
  [inR1_Mob_R1_d] smob=3 -> rate_inR1: (smob'=4);
  [inN2_Mob_N2_d] smob=4 -> rate_inN2: (smob'=5) & (mask'=res-c);

  [outN1_Mob_N1_c] smob=5 -> rate_outN1: (smob'=6);
  [outR2_Mob_R2_c] smob=6 -> rate_outR2: (smob'=7) & (mask'=res-d);
  [outN2_Mob_N2_d] smob=7 -> rate_outN2: (smob'=8);
  [outR1_Mob_R1_d] smob=8 -> rate_outR1: (smob'=1) & (mask'=0);
endmodule

module Mob
  // Mob = MobH
  smob: [1..8] init 1;
  mask: [0..res-a] init 0;

  [] (mask=0) -> (mask'=d);
  [inR1_Mob_R1_d]  smob=1 -> rate_inR1: (smob'=2);
  [inN2_Mob_N2_d]  smob=2 -> rate_inN2: (smob'=3) & (mask'=res-d);
  [outN2_Mob_N2_d] smob=3 -> rate_outN2: (smob'=4);
  [outR1_Mob_R1_d] smob=4 -> rate_outR1: (smob'=5) & (mask'=e);

  [inR3_Mob_R3_e]  smob=5 -> rate_inR3: (smob'=6);
  [inN2_Mob_N2_e]  smob=6 -> rate_inN2: (smob'=7) & (mask'=res-e);
```

```
  [outN2_Mob_N2_e] smob=7 -> rate_outN2: (smob'=8);
  [outR3_Mob_R3_e] smob=8 -> rate_outR3: (smob'=1) & (mask'=0);
endmodule

module Mob
  // Mob = MobI
  smob: [1..8] init 1;
  mask: [0..res-a] init 0;

  [] (mask=0) -> (mask'=c);
  [inR2_Mob_R2_c]  smob=1 -> rate_inR2: (smob'=2);
  [inN1_Mob_N1_c]  smob=2 -> rate_inN1: (smob'=3) & (mask'=e);
  [inR3_Mob_R3_e]  smob=3 -> rate_inR3: (smob'=4);
  [inN2_Mob_N2_e]  smob=4 -> rate_inN2: (smob'=5) & (mask'=res-c);

  [outN1_Mob_N1_c] smob=5 -> rate_outN1: (smob'=6);
  [outR2_Mob_R2_c] smob=6 -> rate_outR2: (smob'=7) & (mask'=res-e);
  [outN2_Mob_N2_e] smob=7 -> rate_outN2: (smob'=8);
  [outR3_Mob_R3_e] smob=8 -> rate_outR3: (smob'=1) & (mask'=0);
endmodule

module Mob
  // Mob = MobJ
  smob: [1..10] init 1;
  mask: [0..res-a] init 0;

  [] (mask=0) -> (mask'=c);
  [inR2_Mob_R2_c]  smob=1 -> rate_inR2: (smob'=2);
  [inN1_Mob_N1_c]  smob=2 -> rate_inN1: (smob'=3) & (mask'=d);
  [inR1_Mob_R1_d]  smob=3 -> rate_inR1: (smob'=4);
  [inN2_Mob_N2_d]  smob=4 -> rate_inN2: (smob'=5) & (mask'=res-d);
  [outN2_Mob_N2_d] smob=5 -> rate_outN2: (smob'=6);
  [outR1_Mob_R1_d] smob=6 -> rate_outR1: (smob'=7) & (mask'=e);

  [inR3_Mob_R3_e]  smob=7 -> rate_inR3: (smob'=8);
  [inN2_Mob_N2_e]  smob=8 -> rate_inN2: (smob'=9) & (mask'=res-e);
  [outN2_Mob_N2_e] smob=9 -> rate_outN2: (smob'=10);
  [outR3_Mob_R3_e] smob=10 -> rate_outR3: (smob'=1) & (mask'=0);
endmodule

module Mob
  // Mob = MobK
  smob: [1..13] init 1;
  mask: [0..res-a] init 0;

  [] (mask=0) -> (mask'=d);
  [inR1_Mob_R1_d]  smob=1 -> rate_inR1: (smob'=2);
  [inN2_Mob_N2_d]  smob=2 -> rate_inN2: (smob'=3) & (mask'=c);
  [inR2_Mob_R2_c]  smob=3 -> rate_inR2: (smob'=4);
  [inN1_Mob_N1_c]  smob=4 -> rate_inN1: (smob'=5) & (mask'=res-d);
  [outN2_Mob_N2_d] smob=5 -> rate_outN2: (smob'=6);
```

```
  [outR1_Mob_R1_d] smob=6 -> rate_outR1: (smob'=7) & (mask'=e);

  [inR3_Mob_R3_e]  smob=7 -> rate_inR3: (smob'=8);
  [inN2_Mob_N2_e]  smob=8 -> rate_inN2: (smob'=9) & (mask'=res-c);
  [outN1_Mob_N1_c] smob=9 -> rate_outN1: (smob'=10);
  [outR2_Mob_R2_c] smob=10 -> rate_outR2: (smob'=11) & (mask'=res-e);
  [outN2_Mob_N2_e] smob=11 -> rate_outN2: (smob'=12);
  [outR3_Mob_R3_e] smob=12 -> rate_outR3: (smob'=13);
  []               smob=13 -> true;
endmodule
```

11 Les récompenses

```
rewards "total_time"

  // Nœud N1
  //[backoff1]       true : 0*pow(10,-6); // backoff = 0 (cf. explications)
  [contention1]      true : 50*pow(10,-6); // DIFS = $50 \mu s$
  [a_N1_R1_alarm1]   true : 4692*pow(10,-6);
  [a_R1_N1_v3a]      true : 28*pow(10,-6);
  [a_R1_N1_nack]     true : 29*pow(10,-6);
  [inN1_Mob_N1_c]    true : 0;
  [outN1_Mob_N1_c]   true : 0;
  [c_N1_R2_alarm1]   true : 4692*pow(10,-6);
  [c_R2_N1_v4b]      true : 28*pow(10,-6);
  [c_R2_N1_nack]     true : 29*pow(10,-6);

  // Nœud N2
  [contention2]      true : 50*pow(10,-6);
  [b_N2_R2_alarm2]   true : 4692*pow(10,-6);
  [b_R2_N2_v4a]      true : 28*pow(10,-6);
  [b_R2_N2_nack]     true : 29*pow(10,-6);
  [inN2_Mob_N2_d]    true : 0;
  [outN2_Mob_N2_d]   true : 0;
  [d_N2_R1_alarm2]   true : 4692*pow(10,-6);
  [d_R1_N2_v3b]      true : 28*pow(10,-6);
  [d_R1_N2_nack]     true : 29*pow(10,-6);
  [inN2_Mob_N2_e]    true : 0;
  [outN2_Mob_N2_e]   true : 0;
  [e_N2_R3_alarm2]   true : 4692*pow(10,-6);
  [e_R3_N2_v5a]      true : 28*pow(10,-6);

  // Routeur R1
  [s1r1_R1_Si1_u3a]  true : 4692*pow(10,-6);
  [s1r1_Si1_R1_v6a]  true : 28*pow(10,-6);
  [s1r1_Si1_R1_nack] true : 29*pow(10,-6);
  [inR1_Mob_R1_d]    true : 0;
  [outR1_Mob_R1_d]   true : 0;
  [s1r1_R1_Si1_u3b]  true : 4692*pow(10,-6);
  [backoffR1]        true : 0*pow(10,-6);
```

```
  // Routeur R2
  [s1r2_R2_Si1_u4a]  true : 4692*pow(10,-6);
  [s1r2_Si1_R2_v6b]  true : 28*pow(10,-6);
  [s1r2_Si1_R2_nack] true : 29*pow(10,-6);
  [b_R2_N2_v4a]      true : 28*pow(10,-6);
  [inR2_Mob_R2_c]    true : 0;
  [outR2_Mob_R2_c]   true : 0;
  [b_R2_N2_v4a]      true : 28*pow(10,-6);
  [s1r2_R2_Si1_u4b]  true : 4692*pow(10,-6);
  [backoffR2]      true : 0*pow(10,-6);

  // Routeur R3
  [inR3_Mob_R3_e]    true : 0;
  [outR3_Mob_R3_e]   true : 0;
  [s2r3_R3_Si2_u5a]  true : 4692*pow(10,-6);
  [s2r3_Si2_R3_v7a]  true : 28*pow(10,-6);

  // Station Si1 et serveur Se (Ethernet)
  [s1se_Si1_Se_u6a]  true : 3036*pow(10,-6);
  [s1se_Se_Si1_ack1] true : 3036*pow(10,-6);
  [s1se_Se_Si1_ack2] true : 3036*pow(10,-6);
  [s1se_Se_Si1_nack] true : 3037*pow(10,-6);
  [s1se_Si1_Se_u6b]  true : 3036*pow(10,-6);
  [backoffSi1]      true : 0*pow(10,-6);

  // Station Si2 et serveur Se
  [s2se_Si2_Se_u7a]  true : 3036*pow(10,-6);
  [s2se_Se_Si2_ack2] true : 3036*pow(10,-6);
  [s2se_Se_Si2_nack] true : 3037*pow(10,-6);
  [backoffSi2]      true : 0*pow(10,-6);

  // Serveur Se
  [processing]       true : 0.5; // puis 1; 1,5; 2 (cf. explications)

endrewards
```

12 Étiquettes

On utilise l'étiquette suivante :

```
label "transmission_completed" = (s1=11) & (s2=18);
```

Bibliographie

[AB11] Robert Abo et Kamel Barkaoui : Performability evaluation with probabilistic model checking for mobile wireless sensor networks. *In Proceedings of The 11th IEEE International Workshop on Wireless Local Networks (WLN'11), October 4-7, Bonn, Germany*, 2011.

[ABC+96] Jean Arlat, Jean-Paul Blanquart, Alain Costes, Yves Crouzet, Yves Deswarte, Jean-Charles Fabre, Hubert Guillermain, Mohamed Kaâniche, Karama Kanoun, Jean-Claude Laprie, Corinne Mazet, David Powell, Christophe Rabéjac et Pascale Thévenod : *Guide de la sûreté de fonctionnement*. Cépaduès-Editions, 1996.

[ACD93] Rajeev Alur, Costas Courcoubetis et David L. Dill : Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.

[ACHH92] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger et Pei-Hsin Ho : Hybrid automata : An algorithmic approach to the specification and verification of hybrid systems. *In* Robert L. Grossman, Anil Nerode, Anders P. Ravn et Hans Rischel, éditeurs : *Hybrid Systems*, volume 736 de *Lecture Notes in Computer Science*, pages 209–229. Springer, 1992.

[AD90] Rajeev Alur et D. L. Dill : Automata for modeling real-time systems. *In Proceedings of the seventeenth international colloquium on Automata, languages and programming*, volume 443 de *Lecture Notes in Computer Science*, pages 322–335, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[AD94] Rajeev Alur et David L. Dill : A theory of timed automata. *Journal of Theoretical Computer Science*, 126:183–235, April 1994.

[Agh86] Gul Agha : *Actors : A Model of Concurrent Computation in Distributed Systems*. Thèse de doctorat, MIT, 1986.

[AH91] Rajeev Alur et Thomas A. Henzinger : Logics and models of real time : A survey. *In* J. W. de Bakker, Cornelis Huizing, Willem P. de Roever et Grzegorz Rozenberg, éditeurs : *REX Workshop*, volume 600 de *Lecture Notes in Computer Science*, pages 74–106. Springer, 1991.

[ASB+95] Adnan Aziz, Vigyan Singhal, Felice Balarin, Robert K. Brayton et Alberto L. Sangiovanni-Vincentelli : It usually works : The temporal logic of stochastic systems. *In* Pierre Wolper, éditeur : *CAV*, volume 939 de *Lecture Notes in Computer Science*, pages 155–165. Springer, 1995.

[ASSB00] Adnan Aziz, Kumud Sanwal, Vigyan Singhal et Robert Brayton : Model-checking continuous-time markov chains. *ACM Transactions on Computational Logic*, 1:162–170, july 2000.

[AZ84] Egidio Astesiano et Elena Zucca : Parametric channels via label expressions in CCS. *Journal of Theoretical Computer Science*, 33:45–64, 1984.

[Bae05] Jos C. M. Baeten : A brief history of process algebra. *Journal of Theoretical Computer Science*, 335(2-3):131–146, 2005.

[Bay00] Bruno Baynat : *Théorie des files d'attente - Des chaînes de Markov aux réseaux à forme produit*. Hermes Science Publications, 2000.

[BB92] Gérard Berry et Gérard Boudol : The chemical abstract machine. *Theor. Comput. Sci.*, 96:217–248, April 1992.

[BBF+01] Béatrice Bérard, Michel Bidoit, Alain Finkel, Francois Laroussinie, Antoine Petit, Laure Petrucci et Philippe Schnoebelen : *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer, 2001.

[BdA95] Andrea Bianco et Luca de Alfaro : Model checking of probabalistic and nondeterministic systems. *In* P. S. Thiagarajan, éditeur : *FSTTCS*, volume 1026 de *Lecture Notes in Computer Science*, pages 499–513. Springer, 1995.

[BDM+98] Marius Bozga, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis et Sergio Yovine : Kronos : A model-checking tool for real-time systems (tool-presentation for ftrtft '98). *In* Anders P. Ravn et Hans Rischel, éditeurs : *FTRTFT*, volume 1486 de *Lecture Notes in Computer Science*, pages 298–302. Springer, 1998.

[Ber04] Martin Berger : *Towards Abstractions For Distributed Systems*. Thèse de doctorat, Imperial College of Science, Technology and Medicine - University of London - United Kingdom, november 2004.

[BGLA01] Lichun Bao et J. J. Garcia-Luna-Aceves : A new approach to channel access scheduling for ad hoc networks. *In MOBICOM*, pages 210–221, 2001.

[BHHK03] Christel Baier, Boudewijn R. Haverkort, Holger Hermanns et Joost-Pieter Katoen : Model-checking algorithms for continuous-time markov chains. *IEEE Transactions on Software Engineering (TSE)*, 29(6):524–541, 2003.

[BHHK10] Christel Baier, Boudewijn R. Haverkort, Holger Hermanns et Joost-Pieter Katoen : Performance evaluation and model checking join forces. *Communications of the ACM*, 53(9):76–85, 2010.

[BJV10] Jasper Berendsen, David N. Jansen et Frits W. Vaandrager : Fortuna : Model checking priced probabilistic timed automata. *In QEST*, pages 273–281. IEEE Computer Society, 2010.

[BK85] Jan A. Bergstra et Jan Willem Klop : Algebra of communicating processes with abstraction. *Journal of Theoretical Computer Science*, 37:77–121, 1985.

[BL08] Roberto Bruni et Ivan Lanese : Parametric synchronizations in mobile nominal calculi. *Theoretical Computer Science*, 402(2-3):102–119, 2008.

[BN96] Michele Boreale et Rocco De Nicola : A symbolic semantics for the π-calculus. *Information and Computation*, 126(1):34–52, 1996.

[Bou89] Gérard Boudol : Towards a lambda-calculus for concurrent and communicating systems. *In Proceedings of the International Joint Conference on Theory and Practice of Software Development, Volume 1 : Advanced Seminar on Foundations of Innovative Software Development I and Colloquium on Trees in Algebra and Programming*, TAPSOFT '89/CAAP '89, pages 149–161, London, UK, 1989. Springer-Verlag.

[Bou92] Gérard Boudol : Asynchrony and the π-calculus. Research Report RR-1702, INRIA, 1992.

[BP05] Mohit Bhargava et Catuscia Palamidessi : Probabilistic anonymity. *In* Martín Abadi et Luca de Alfaro, éditeurs : *CONCUR*, volume 3653 de *Lecture Notes in Computer Science*, pages 171–185. Springer, 2005.

[Bé08] Béatrice Bérard : Model checking temporisé. *In Approches formelles des systèmes embarqués communicants*, pages 75–104, 2008.

[CBD02] Tracy Camp, Jeff Boleng et Vanessa Davies : A survey of mobility models for ad hoc network research. *Wireless Communications and Mobile Computing*, 2(5):483–502, 2002.

[CC01] Luís Caires et Luca Cardelli : A spatial logic for concurrency. *In* Naoki Kobayashi et Benjamin C. Pierce, éditeurs : *TACS*, volume 2215 de *Lecture Notes in Computer Science*, pages 1–37. Springer, 2001.

[CC04] Luís Caires et Luca Cardelli : A spatial logic for concurrency - ii. *Theor. Comput. Sci.*, 322(3):517–565, 2004.

[CE81] Edmund M. Clarke et E. Allen Emerson : Design and synthesis of synchronization skeletons using branching-time temporal logic. *In* Dexter Kozen, éditeur : *Logic of Programs*, volume 131 de *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981.

[CES09] Edmund M. Clarke, E. Allen Emerson et Joseph Sifakis : Model checking : algorithmic verification and debugging. *Communications of the ACM*, 52(11): 74–84, 2009.

[CG98] Luca Cardelli et Andrew D. Gordon : Mobile ambients. *In* Maurice Nivat, éditeur : *FoSSaCS*, volume 1378 de *Lecture Notes in Computer Science*, pages 140–155. Springer, 1998.

[CGJ+03] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu et Helmut Veith : Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM*, 50(5):752–794, 2003.

[Chu32] Alonzo Church : A Set of Postulates for the Foundation of Logic. *Annals of Mathematics*, 33(2):346–366, 1932.

[Chu41] Alonzo Church : *The Calculi of Lambda-Conversion*. Numéro 6 de Annals of Mathematical Studies. Princeton University Press, 1941.

[Cli81] William D. Clinger : Foundations of Actor Semantics. Rapport technique, Cambridge, MA, USA, 1981.

[Col09] Collectif : *Le Nouveau Petit Robert de la langue française 2009 Grand format*. Éditions Le Robert, Paris, 2009. Nt2 Référence.

[Col10] Collectif : *VANET - Vehicular Applications and Inter-Networking Technologies*. John Wiley & Sons, 2010.

[COQ] Site internet de COQ : `http://coq.inria.fr/` (2011).

[CP05] Konstantinos Chatzikokolakis et Catuscia Palamidessi : A framework for analyzing probabilistic protocols and its application to the partial secrets exchange. *In* Rocco De Nicola et Davide Sangiorgi, éditeurs : *TGC*, volume 3705 de *Lecture Notes in Computer Science*, pages 146–162. Springer, 2005.

[CRR02] Sagar Chaki, Sriram K. Rajamani et Jakob Rehof : Types as models : model checking message-passing programs. *In POPL*, pages 45–57, 2002.

[Dal01] Silvano Dal Zilio : Mobile Processes : a Commented Bibliography. *In MOVEP'2K – 4th Summer school on Modelling and Verification of Parallel processes*, volume 2067 de *Lecture Notes in Computer Science*, pages 206–222. Springer-Verlag, june 2001.

[EB70] P. Ercoli et F. L. Bauer : Software engineering techniques. Report on a conference sponsored by the NATO science committee. Rome, Italy, 27th to 31st october 1969. Rapport technique, NATO, 1970.

[EH02] Amre El-Hoiydi : Spatial tdma and csma with preamble sampling for low power ad hoc wireless sensor networks. *In ISCC*, pages 685–692. IEEE Computer Society, 2002.

[ElH02] *Aloha with preamble sampling for sporadic traffic in ad hoc wireless sensor networks*, volume 5, 2002.

[EM01] Cristian Ene et Traian Muntean : A broadcast-based calculus for communicating systems. *In IPDPS*, page 149. IEEE Computer Society, 2001.

[EN] Uffe Engberg et Mogens Nielsen : A calculus of communicating systems with label-passing. Rapport technique DAIMI PB-208.

[FGL+96] Cédric Fournet, Georges Gonthier, Jean-Jacques Lévy, Luc Maranget et Didier Rémy : A calculus of mobile agents. *In* Ugo Montanari et Vladimiro Sassone, éditeurs : *CONCUR*, volume 1119 de *Lecture Notes in Computer Science*, pages 406–421. Springer, 1996.

[FOR] Site internet de FORTUNA : `http://www.cs.ru.nl/J.Berendsen/fortuna/` (2011).

[GBL06] Alexandre David Gerd Behrmann et Kim G. Larsen : A tutorial on uppaal 4.0, 2006. Disponible à `http://uppaal.org/`.

[God07] Jens Chr. Godskesen : A calculus for mobile ad hoc networks. *In* Amy L. Murphy et Jan Vitek, éditeurs : *COORDINATION*, volume 4467 de *Lecture Notes in Computer Science*, pages 132–150. Springer, 2007.

[Gre75] Irene Greif : *Semantics of Communicating Parallel Processes.* Thèse de doctorat, MIT, 1975.

[HB77] Carl Hewitt et Henry G. Baker : Laws for communicating parallel processes. *In IFIP Congress*, pages 987–992, 1977.

[HBS73] Carl Hewitt, Peter Bishop et Richard Steiger : A universal modular actor formalism for artificial intelligence. *In IJCAI*, pages 235–245, 1973.

[HCM] Site internet de HCMC : `http://www.lsv.ens-cachan.fr/~fl/cmcweb.html` (2011).

[HH09] Arnd Hartmanns et Holger Hermanns : A modest approach to checking probabilistic timed automata. *In QEST*, pages 187–196. IEEE Computer Society, 2009.

[HHWT97] Thomas A. Henzinger, Pei-Hsin Ho et Howard Wong-Toi : Hytech : A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1(1-2):110–122, 1997.

[Hil05a] Jane Hillston : Process algebras for quantitative analysis. *In LICS*, pages 239–248. IEEE Computer Society, 2005.

[Hil05b] Jane Hillston : Tuning systems : From composition to performance (the Needham lecture). *The Computer Journal*, 48(4):385–400, 2005.

[HJ94] H. Hansson et B. Jonsson : A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[HJD10] Elizabeth Hull, Ken Jackson et Jeremy Dick : *Requirements Engineering (Third Edition).* Springer, 2010.

[HKNP06] A. Hinton, M. Kwiatkowska, G. Norman et D. Parker : PRISM : A tool for automatic verification of probabilistic systems. *In* H. Hermanns et J. Palsberg, éditeurs : *12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, volume 3920 de *LNCS*, pages 441–444. Springer, 2006.

[HL95] Matthew Hennessy et Huimin Lin : Symbolic bisimulations. *Theoretical Computer Science*, 138(2):353–389, 1995.

[HNSY94] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis et Sergio Yovine : Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[Hoa85] C. A. R. Hoare : *Communicating Sequential Processes (Prentice Hall International Series in Computing Science)*. 1985.

[HOL] Site internet de HOL : `http://hol.sourceforge.net/` (2011).

[Hon92] Kohei Honda : Two bisimilarities in ν-calculus. Rapport technique 92-002, Keio University, Department of Computer Science, 1992.

[HP00] Oltea Mihaela Herescu et Catuscia Palamidessi : Probabilistic asynchronous π-calculus. *In* Jerzy Tiuryn, éditeur : *FoSSaCS*, volume 1784 de *Lecture Notes in Computer Science*, pages 146–160. Springer, 2000.

[HT91a] Kohei Honda et Mario Tokoro : On asynchronous communication semantics. *In* Mario Tokoro, Oscar Nierstrasz et Peter Wegner, éditeurs : *Object-Based Concurrent Computing*, volume 612 de *Lecture Notes in Computer Science*, pages 21–51. Springer, 1991.

[HT91b] Kohei Honda et Mario Tokoro : A small calculus for concurrent objects. *SIGPLAN OOPS Messenger*, 2:50–54, April 1991.

[HW99] John R. Hubbard et Beverly West : *Equations différentielles et systèmes dynamiques*. Cassini, 1999.

[IEE02] IEEE Computer Society : *IEEE Standard for information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 15.1 : Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personnal Area Networks*, june 2002.

[IEE03] IEEE Computer Society : *IEEE Standard for information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 15.4 : Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)*, october 2003.

[IEE07] IEEE Computer Society : *IEEE Standard for information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 11 : Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, june 2007.

[IEE08] IEEE Computer Society : *IEEE Standard for information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 3 : Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*, december 2008.

[IL01] Anna Ingólfsdóttir et Huimin Lin : A symbolic approach to value-passing processes. *In Handbook of Process Algebra, chapter 7*. Elsevier, 2001.

[ISA] Site internet de ISABELLE : `http://isabelle.in.tum.de/` (2011).

[JM04] Ole Høgh Jensen et Robin Milner : Bigraphs and mobile processes (revised). Rapport technique UCAM-CL-TR-580, february 2004.

[Kah74] Gilles Kahn : The semantics of simple language for parallel programming. *In IFIP Congress*, pages 471–475, 1974.

[Kar90] Phil KARN : MACA – a new channel access method for packet radio. 1990.

[Kat95] Randy H. KATZ : Adaptation and mobility in wireless hformation systems. *IEEE Communications Magazine*, 1995.

[KHH11] Ville KASEVA, Timo D. HÄMÄLÄINEN et Marko HÄNNIKÄINEN : A wireless sensor network for hospital security : from user requirements to pilot deployment. *EURASIP Journal on Wireless Communication and Networking*, 2011(article ID 920141), 2011.

[KNP10] M. KWIATKOWSKA, G. NORMAN et D. PARKER : Advances and challenges of probabilistic model checking. *In Proceedings of the 48th Annual Allerton Conference on Communication, Control and Computing*. IEEE Press, 2010.

[KNP11] M. KWIATKOWSKA, G. NORMAN et D. PARKER : PRISM 4.0 : Verification of probabilistic real-time systems. *In* G. GOPALAKRISHNAN et S. QADEER, éditeurs : *23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 de *LNCS*, pages 585–591. Springer, 2011.

[KNSS02] M. KWIATKOWSKA, G. NORMAN, R. SEGALA et J. SPROSTON : Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 282:101–150, June 2002.

[Koy90] Ron KOYMANS : Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[KRO] Site internet de KRONOS : `http://www-verimag.imag.fr/DIST-TOOLS/TEMPO/kronos/` (2011).

[Kwi07] M. KWIATKOWSKA : Quantitative verification : Models, techniques and tools. *In 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 449–458. ACM Press, september 2007.

[KZH+11] Joost-Pieter KATOEN, Ivan S. ZAPREEV, Ernst Moritz HAHN, Holger HERMANNS et David N. JANSEN : The ins and outs of the probabilistic model checker mrmc. *Performance Evaluation*, 68(2):90–104, 2011.

[Lin94] Huimin LIN : Symbolic bisimulations and proof systems for the π-calculus. Rapport technique, School of Cognitive and Computer Science, University of Sussex, 1994.

[Lin03] Huimin LIN : Complete inference systems for weak bisimulation equivalences in the π-calculus. *Information and Computation*, 180(1):1–29, 2003.

[LKR04] Gang LU, Bhaskar KRISHNAMACHARI et Cauligi S. RAGHAVENDRA : An adaptive energy-efficient and low-latency mac for data gathering in wireless sensor networks. *In IPDPS*. IEEE Computer Society, 2004.

[LLW95] François LAROUSSINIE, Kim Guldstrand LARSEN et Carsten WEISE : From timed automata to logic - and back. *In* Jirí WIEDERMANN et Petr HÁJEK, éditeurs : *MFCS*, volume 969 de *Lecture Notes in Computer Science*, pages 529–539. Springer, 1995.

[LPY97] Kim Guldstrand LARSEN, Paul PETTERSSON et Wang YI : Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.

[LZ02] J. Y. LEE et John J. ZIC : On modelling real-time mobile processes. *In* Michael J. OUDSHOORN, éditeur : *ACSC*, volume 4 de *CRPIT*, pages 139–147. Australian Computer Society, 2002.

[McM93] Kenneth L McMillan : *Symbolic model checking : an approach to the state explosion problem*. Kluwer Academic Publishers, 1993.

[mcp] Site internet de mcpta : `http://www.modestchecker.net/` (2011).

[Mer09] Massimo Merro : An observational theory for mobile ad hoc networks (full version). *Information and Computation*, 207(2):194–208, 2009.

[Mil80] Robin Milner : *A Calculus of Communicating Systems*, volume 92 de *Lecture Notes in Computer Science*. Springer-Verlag, 1980.

[Mil91] Robin Milner : The polyadic π-calculus : a tutorial. Rapport technique ECS-LFCS-91-18, 1991. Également publié dans Logic and Algebra of Specification, ed. F. L. Hamer, W. Brauer and H. Schwichtenberg, Springer-Verlag, 1993.

[Mil92] Robin Milner : Functions as processes. *Mathematical Structures in Computer Science*, 2(2):119–141, 1992.

[Mil99] Robin Milner : *Communicating and Mobile Systems : the π-Calculus*. Cambridge University Press, june 1999.

[Mil01] Robin Milner : Bigraphical reactive systems. *In* Kim Guldstrand Larsen et Mogens Nielsen, éditeurs : *12th International Conference on Concurrency Theory (CONCUR'01)*, volume 2154 de *Lecture Notes in Computer Science*, pages 16–35. Springer-Verlag, 2001.

[Mil09] Robin Milner : *The Space and Motion of Communicating Agents*. Cambridge University Press, 2009.

[MPW92] R. Milner, J. Parrow et D. Walker : A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–40 and 41–77, september 1992.

[MS06] Nicola Mezzetti et Davide Sangiorgi : Towards a calculus for wireless systems. *Electronic Notes in Theoretical Computer Science*, 158:331–353, May 2006.

[NH06] Sebastian Nanz et Chris Hankin : A framework for security analysis of mobile wireless networks. *Theoretical Computer Science*, 367(1-2):203–227, 2006.

[Nie89] Flemming Nielson : The typed lambda-calculus with first-class processes. *In* Eddy Odijk, Martin Rem et Jean-Claude Syre, éditeurs : *PARLE (2)*, volume 366 de *Lecture Notes in Computer Science*, pages 357–373. Springer, 1989.

[NPPW09] G. Norman, C. Palamidessi, D. Parker et P. Wu : Model checking probabilistic and stochastic extensions of the π-calculus. *IEEE Transactions on Software Engineering*, 35(2):209–223, 2009.

[NS2] Site internet de ns-2 : `http://www.isi.edu/nsnam/ns/` (2011).

[NS3] Site internet de ns-3 : `http://www.nsnam.org/` (2011).

[OMN] Site internet de OMNeT++ : `http://www.omnetpp.org/` (2011).

[OPN] Site internet de OPNET : `http://www.opnet.com/` (2011).

[PC05] Cristian Prisacariu et Gabriel Ciobanu : Timed distributed π-calculus. Research Report FML-05-01, Institute of Computer Science, Romanian Academy, 2005.

[PEP] Site internet de PEPA : `http://www.dcs.ed.ac.uk/pepa/` (2011).

[PHC04] Joseph Polastre, Jason L. Hill et David E. Culler : Versatile low power media access for wireless sensor networks. *In* John A. Stankovic, Anish Arora et Ramesh Govindan, éditeurs : *SenSys*, pages 95–107. ACM, 2004.

[Pnu77] Amir Pnueli : The temporal logic of programs. *In Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS'77)*, pages 46–57, 1977.

[Pnu97] Amir Pnueli : Verification engineering : A future profession (a. m. turing award lecture). *In PODC*, page 7, 1997.

[Pra95] K. V. S. Prasad : A calculus of broadcasting systems. *Science of Computer Programming*, 25(2-3):285–327, 1995.

[Pra06] K. V. S. Prasad : A prospectus for mobile broadcasting systems. *Electronic Notes in Theoretical Computer Science*, 162:295–300, 2006.

[PRIa] Prism manual version 4.0.1.

[PRIb] Site internet de PRISM : `http://www.prismmodelchecker.org` (2011).

[Pri95] Corrado Priami : Stochastic π-calculus. *The Computer Journal*, 38(7):578–589, 1995. Proceedings of PAPM '95.

[PROa] Site internet de ProVerif : `http://www.proverif.ens.fr/` (2011).

[PROb] Site internet du langage PROMELA : `http://www.dai-arc.polito.it/dai-arc/manual/tools/jcat/main/node168.html` (2011).

[PVS] Site internet de PVS : `http://pvs.csl.sri.com/` (2011).

[QS82] Jean-Pierre Queille et Joseph Sifakis : Specification and verification of concurrent systems in cesar. *In* Mariangiola Dezani-Ciancaglini et Ugo Montanari, éditeurs : *Symposium on Programming*, volume 137 de *Lecture Notes in Computer Science*, pages 337–351. Springer, 1982.

[Rei85] Wolfgang Reisig : *Petri Nets : An Introduction*, volume 4 de *Monographs in Theoretical Computer Science. An EATCS Series*. Springer, 1985.

[RH98] James Riely et Matthew Hennessy : A typed language for distributed mobile processes (extended abstract). *In 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'98)*, pages 378–390, 1998.

[ROGLA03] Venkatesh Rajendran, Katia Obraczka et J. J. Garcia-Luna-Aceves : Energy-efficient collision-free medium access control for wireless sensor networks. *In* Ian F. Akyildiz, Deborah Estrin, David E. Culler et Mani B. Srivastava, éditeurs : *SenSys*, pages 181–192. ACM, 2003.

[RS03] William C. Rounds et Hosung Song : The ϕ-calculus : A language for distributed control of reconfigurable embedded systems. *In* Oded Maler et Amir Pnueli, éditeurs : *HSCC*, volume 2623 de *Lecture Notes in Computer Science*, pages 435–449. Springer, 2003.

[Sak03] Jacques Sakarovitch : *Éléments de théorie des automates*. Vuibert, 2003.

[SC] H. Song et K. Compton : Verifying π-calculus processes by promela translation. Rapport technique CSE-TR-472-03.

[Sch06] Christian Schindelhauer : Mobility in wireless networks. *In* Jirí Wiedermann, Gerard Tel, Jaroslav Pokorný, Mária Bieliková et Julius Stuller, éditeurs : *SOFSEM*, volume 3831 de *Lecture Notes in Computer Science*, pages 100–116. Springer, 2006.

[SCI+01] Eugene Shih, Seong-Hwan Cho, Nathan Ickes, Rex Min, Amit Sinha, Alice Wang et Anantha Chandrakasan : Physical layer driven protocol and algorithm design for energy-efficient wireless sensor networks. *In MOBICOM*, pages 272–287, 2001.

[SL95] Roberto Segala et Nancy A. Lynch : Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.

[Spi] Site internet de Spin : `http://spinroot.com/spin/whatispin.html` (2011).

[SRS10] Anu Singh, C. R. Ramakrishnan et Scott A. Smolka : A process calculus for mobile ad hoc networks. *Science of Computer Programming*, 75(6):440–469, 2010.

[SW01] Davide Sangiorgi et David Walker : *The π-calculus : A Theory of Mobile Processes*. Cambridge University Press, New York, NY, USA, 2001.

[SWR98] Suresh Singh, Mike Woo et C. S. Raghavendra : Power-aware routing in mobile ad hoc networks. *In Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking (MobiCom 1998)*, pages 181–190, New York, NY, USA, 1998. ACM.

[TDG09] Mirco Tribastone, Adam Duguid et Stephen Gilmore : The pepa eclipse plugin. *Sigmetrics Perform. Eval. Rev.*, 36:28–33, march 2009.

[Tho89] B. Thomsen : A calculus of higher order communicating systems. *In Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '89, pages 143–154, New York, NY, USA, 1989. ACM.

[UPPa] Site internet de UPPAAL : `http://uppaal.org/` (2011).

[UPPb] Site internet de UPPAAL-PRO : `http://people.cs.aau.dk/~arild/uppaal-probabilistic/` (2011).

[VM94] Björn Victor et Faron Moller : The Mobility Workbench — a tool for the π-calculus. *In* David Dill, éditeur : *CAV'94 : Computer Aided Verification*, volume 818 de *Lecture Notes in Computer Science*, pages 428–440. Springer-Verlag, 1994.

[WB96] Mark Weiser et John Seely Brown : Designing calm technology. *PowerGrid Journal*, 1, 1996.

[Wei91] Mark Weiser : The computer for the twenty-first century. *Scientific American*, pages 94–104, september 1991.

[Wei93] Mark Weiser : Hot topics : Ubiquitous computing. *IEEE Computer*, october 1993.

[Wu03] Peng Wu : Interpreting pi-calculus with spin/promela. *Computer Science*, 8:7–9 (supplement), 2003.

[YHE02] Wei Ye, John S. Heidemann et Deborah Estrin : An energy-efficient mac protocol for wireless sensor networks. *In INFOCOM*, 2002.

[YHE04] Wei Ye, John S. Heidemann et Deborah Estrin : Medium access control with coordinated adaptive sleeping for wireless sensor networks. *IEEE/ACM Transactions on Networking*, 12(3):493–506, 2004.

[YRS04] Ping Yang, C. R. Ramakrishnan et Scott A. Smolka : A logical encoding of the pi-calculus : model checking mobile processes using tabled resolution. *International Journal on Software Tools for Technology Transfer*, 6(1):38–66, 2004.

[YSH06] Wei Ye, Fabio Silva et John S. Heidemann : Ultra-low duty cycle mac with scheduled channel polling. *In* Andrew T. Campbell, Philippe Bonnet et John S. Heidemann, éditeurs : *SenSys*, pages 321–334. ACM, 2006.

Index

R

S

T

Robert ABO

Approches Formelles pour l'Analyse de la Performabilité des Systèmes Communicants Mobiles

Applications aux réseaux de capteurs sans fil

Résumé

Nous nous intéressons à l'analyse des exigences de performabilité des systèmes communicants mobiles par model checking. Nous modélisons ces systèmes à l'aide d'un formalisme de haut niveau issu du π-calcul, permettant de considérer des comportements stochastiques, temporels, déterministes, ou indéterministes. Cependant, dans le π-calcul, la primitive de communication de base des systèmes est la communication en point-à-point synchrone. Or, les systèmes mobiles, qui utilisent des réseaux sans fil, communiquent essentiellement par diffusion locale. C'est pourquoi, dans un premier temps, nous définissons la communication par diffusion dans le π-calcul, afin de mieux modéliser les systèmes que nous étudions. Nous proposons d'utiliser des versions probabilistes et stochastiques de l'algèbre que nous avons défini, pour permettre des études de performance. Nous en définissons une version temporelle permettant de considérer le temps dans les modèles. Mais l'absence d'outils d'analyse des propriétés sur des modèles spécifiés en une algèbre issue du π-calcul est un obstacle majeur à notre travail. La définition de règles de traduction en langage Prism, nous permet de traduire nos modèles, en modèles de bas niveau supports du model checking, à savoir des chaînes de Markov à temps discret, à temps continu, des automates temporisés, ou des automates temporisés probabilistes. Nous avons choisi l'outil Prism car, à notre connaissance, dans sa dernière version, il est le seul outil à supporter les formalismes de bas niveau que nous venons de citer, et ainsi de réaliser des études de performabilité complètes. Cette façon de procéder nous permet de pallier à l'absence d'outils d'analyse pour nos modèles. Par la suite, nous appliquons ces concepts théoriques aux réseaux de capteurs sans fil mobiles.

Mot-clefs : mobilité, performabilité, π-calcul, model checking, outil Prism.

Abstract

We are interested in analyzing the performability requirements of mobile communication systems by using model checking techniques. We model these systems using a high-level formalism derived from the π-calculus, for considering stochastic, timed, deterministic or indeterministic behaviors. However, in the π-calculus, the basic communication primitive of systems is the synchronous point-to-point communication. But mobile systems that use wireless networks mostly communicate by local broadcast. Therefore, we first define the broadcast communication in the π-calculus to better model the systems we study. We propose to use probabilistic and stochastic versions of the calculus we have defined to allow performance studies. We define a temporal version to consider time in the models. The lack of tools for analyzing properties of models specified with π-calculus is a major obstacle to our work and its objectives. The definition of translation rules into the Prism language allows us to translate our models into low-level models which can support model checking, namely discrete-time, or continuous-time Markov chains, timed automata, or probabilistic timed automata. We chose the Prism model checker because, in our best knowledge, in its latest version, it is the only tool that supports the low-level formalisms that we have previously cited, and thus, makes it possible our performability studies. This approach allows us to overcome the lack of model checkers for our models. Subsequently, we apply these theoretical concepts for mobile wireless sensor networks.

Keywords: mobility, performability, π-calculus, model checking, Prism model checker.