Systèmes multi-agents adaptatifs avec contraintes temps-réel : de la spécification formelle à la vérification et à la génération de code

Guillaume Hutzler — Hanna Klaudel — Dong Yue Wang

LaMI, UMR 8042
Université Evry-Val d'Essonne/CNRS
523, Place des Terrasses
91000 Evry
{nom}@lami.univ-evry.fr

RÉSUMÉ. *La conception de systèmes réactifs répond à des impératifs de correction logique (le système fait ce qu'il doit) et de correction temporelle (le système se conforme à un ensemble de contraintes temporelles). Nous proposons dans cet article une approche globale de conception de systèmes réactifs adaptatifs, c'est-à-dire adaptant dynamiquement leur architecture en fonction du contexte. Cela est rendu possible en autorisant une certaine tolérance dans le respect des contraintes logiques et temporelles (on parlera alors de qualité plutôt que de correction). Pour représenter le comportement des agents composant le système, nous utilisons le formalisme des automates temporisés, ce qui permet d'évaluer a priori les propriétés du système (en termes de qualité logique et temporelle), grâce à des techniques de model-checking et de simulation. Des outils de génération automatique de code, que nous avons développés, nous permettent ensuite de produire rapidement, à partir du modèle, un prototype multi-agent opérationnel qui a le comportement désiré.*

ABSTRACT. *The design of reactive systems must comply with logical correctness (the system does what it supposed to do) and timeliness (the system has to satisfy a set of temporal constraints) criteria. In this paper, we propose a global approach for the design of adaptive reactive systems, i.e. systems that dynamically adapt their architecture depending on the context. This is made possible by relaxing the logical and temporal constraints (we will talk about quality rather than correctness). We use the timed automata formalism for the design of the agents' behaviour. This allows evaluating beforehand the properties of the system (regarding logical and temporal quality), thanks to model-checking and simulation techniques. This model is enhanced with tools that we developed for the automatic generation of code, allowing to produce very quickly a running multi-agent prototype with the desired behaviour.*

MOTS-CLÉS : *génie logiciel orienté multi-agent, modèles formels, programmation orientée multi-agent*

KEYWORDS: *agent oriented software engineering, formal models, agent oriented programming*

1. Introduction

Les systèmes temps-réels réactifs sont définis par leur capacité à réagir constamment aux sollicitations de leur environnement en se conformant à un certain nombre de contraintes temporelles. En un temps limité, le système doit acquérir et traiter les données et les événements caractérisant l'évolution temporelle de cet environnement, prendre les décisions appropriées et les transformer en actions. La fonctionnalité du système provient ainsi de sa capacité à présenter les bonnes sorties (correction logique) au bon moment (correction temporelle). Du fait du caractère souvent critique de ce type d'applications, les architectures logicielles et matérielles correspondantes sont spécifiées, développées, validées avec le plus grand soin et sont ensuite figées de manière à s'assurer que le système aura un comportement déterministe et prédictible. L'apport des systèmes multi-agents dans ce cadre peut alors sembler limité, notamment de par l'autonomie et la proactivité que l'on attribue généralement aux agents. De fait, de nombreux exemples d'utilisation des systèmes multi-agents dans un contexte temps-réel (Attoui, 1997 ; Wolfe et al., 2000) mettent en avant davantage l'aspect distribué que les aspects de décentralisation et d'autonomie. Dit autrement, il s'agit de faire fonctionner ensemble des entités distribuées mais avec une architecture globale figée.

Des études existent, dans le domaine des agents autonomes, qui s'intéressent à l'intégration de problématiques temps-réel. Ces travaux se focalisent cependant en priorité sur la cohabitation, au sein d'un agent unique, de modules de raisonnement (pour la planification par exemple) et de modules de contrôle temps-réel (Musliner et al. 1993 ou Atkins et al. 2001 sur l'architecture CIRCA). L'aspect coopératif entre les agents, reste pour l'instant peu abordé même s'il est cité comme perspective (Musliner et al. 2003 sur MASA-CIRCA). Des travaux apparaissent néanmoins qui vont vers l'intégration des deux problématiques, par le développement de plate-formes d'exécution spécifiques (Soler et al. 2002) ou par le développement de méthodes de planification multi-agent en contexte dynamique (Marc et al. 2003).

Dans le problème auquel nous nous intéressons, ce n'est pas un agent isolé qui doit fonctionner en temps-réel mais un système multi-agent qui doit se comporter globalement comme un système temps-réel. Les contraintes de fonctionnement temps-réel doivent donc être gérées de manière collective et non plus individuelle. Dans ce contexte d'ordonnancement dynamique dans un système réparti, il n'existe pas à l'heure actuelle de solution garantissant le respect des contraintes temporelles. Il s'agira donc d'optimiser le compromis entre respect de la correction logique et respect de la correction temporelle, en relâchant au choix l'une ou l'autre des contraintes lorsque les ressources disponibles ne permettent pas de respecter les deux simultanément. Cela est rendu possible dans notre cas par le fait que les contraintes temporelles ne sont pas critiques (le fait de répondre avec un délai un peu plus long que prévu n'est pas rédhibitoire) ni même strictes (lorsque le délai de réponse normal est dépassé, la valeur du résultat ne devient pas immédiatement

nulle mais diminue plus ou moins rapidement avec le temps). De même, l'organisation des traitements en composants élémentaires pouvant fonctionner à différents niveaux de précision, et pouvant être assemblés de manière dynamique pour construire une chaîne de traitement, permet d'ajuster au besoin la qualité globale du traitement et de fonctionner en mode dégradé. La valeur de l'analyse diminue à mesure que le traitement est dégradé mais ne devient pas immédiatement nulle.

La nature des traitements à effectuer et leur priorité peuvent par ailleurs être variables et imprédictibles, de même que la disponibilité des entités (processeurs) chargées du traitement. En conséquence, il n'est pas réaliste de considérer que la durée d'un traitement est fixe, le traitement pouvant se terminer plus tôt ou plus tard que prévu, sans compter les pannes possibles de machines ou encore les pertes de messages provoquant un ralentissement global du système. De ce fait, un grand nombre de techniques classique d'ordonnancement (optimisation statique, job-shop scheduling, programmation dynamique, Blazewicz et al. 1993) se révèlent inapplicables, soit parce que l'espace de recherche augmente de manière exponentielle, soit parce que l'horizon prédictif est très court (plus on progresse vers le futur et plus l'incertitude augmente quant à la probabilité d'occurrence des événements). On pourrait penser que les méthodes d'ordonnancement sous incertitude seraient des bons candidats dans ce cas, dans la mesure où elles prennent en compte de manière dynamique le temps effectif d'exécution d'une tâche pour replanifier l'exécution des tâches à un horizon borné. Il apparaît cependant que ce travail de planification, exécuté à chaque fois qu'une tâche se termine, engendre beaucoup de calculs en ligne, ce qui limite son applicabilité à des processus « lents » (Maler 2004).

Dans ce cadre, le problème auquel nous nous intéressons plus particulièrement concerne la reconfiguration adaptative de la chaîne de traitement des données (i.e. le système multi-agent chargé du traitement) au cours de l'exécution. Cette reconfiguration peut se produire en fonction des ressources disponibles (capteurs, processeurs, effecteurs), de la qualité logique souhaitée, de la qualité temporelle assurée et des événements se produisant dans l'environnement. Elle ne pourra s'effectuer de manière centralisée et devra donc être prise en charge par les agents eux-mêmes, en plus de leur activité de traitement des données.

La contribution principale de cet article est de montrer l'apport de formalismes comme les automates temporisés dans le processus de modélisation d'un système d'agents, à la fois sous l'angle du traitement de données et sous l'angle de la reconfiguration dynamique de la chaîne de traitement. Nous nous plaçons dans une démarche de prototypage rapide, qui vise à proposer une approche complète de conception de systèmes multi-agents temps-réels adaptatifs. A terme, cette approche devra couvrir l'ensemble des étapes du cycle de vie du logiciel, depuis la spécification de l'architecture logicielle jusqu'à l'implantation et au test, en passant par la vérification formelle de propriétés et la simulation. La méthode se base sur le formalisme des automates temporisés (Alur et Dill, 1994), qui permet la

spécification de systèmes comme un ensemble de processus concurrents dans lesquels on peut exprimer des contraintes temporelles (section 3). Nous illustrons l'utilisation de ce formalisme pour la modélisation d'un système d'agents (section 4) ainsi que l'utilisation du model-checking et de la simulation pour vérifier certaines propriétés du système et analyser son fonctionnement (section 5). Nous abordons enfin le passage semi-automatique de la spécification sous forme d'automates à des agents exécutables (section 6). Nous décrivons auparavant l'application cible et ses spécificités.

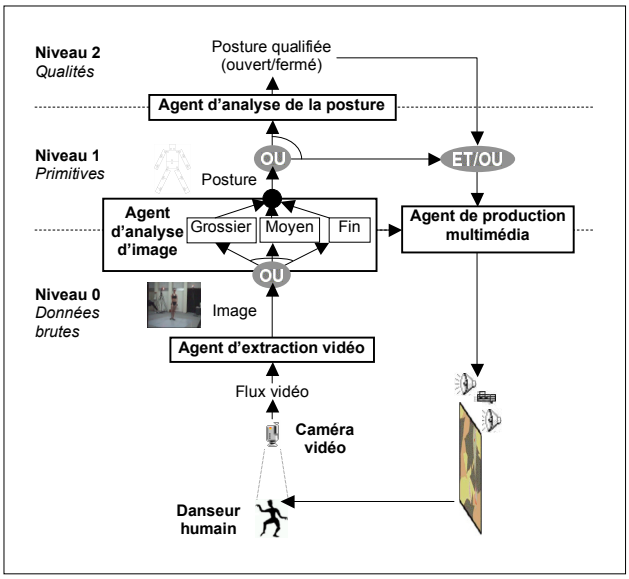


Figure 1. *Architecture globale de la chaîne de traitement pour la modalité visuelle dans le projet « J'ai dansé avec Machine »*

2. Application cible et objectifs

L'application à laquelle nous nous intéressons est le projet *J'ai dansé avec Machine*, dans lequel il s'agit d'établir un dialogue multimodal et multimédia entre un danseur humain sur une scène et un système matériel et logiciel décentralisé (Hutzler et al., 2002). Ce dernier a pour charge de capter par différents moyens la prestation du danseur, de l'analyser en temps-réel et finalement d'y répondre par la production d'animations visuelles projetées sur des écrans autour du danseur, la production de séquences musicales ou encore par la mise en mouvement d'objets physiques (robots ou autres). Nous considérons cette application comme une

transposition métaphorique des interactions que l'on peut anticiper entre des utilisateurs humains et un ensemble d'objets communicants qui s'organiseront dynamiquement autour de lui pour lui fournir un ensemble de services. Cela correspond à ce que nous désignons sous le terme d'Environnements Cognitifs Ambiants (ECA), dans lesquels l'ensemble des moyens de perception, de traitement et d'action présents dans un environnement physique, s'organisent de manière coopérative autour d'un utilisateur pour lui offrir des modalités d'interaction naturelles et des services étendus.

Nous considérons que le système informatique d'interaction avec le danseur est constitué d'un ensemble de processeurs, dotés de moyens de communication et associés ou non à des capteurs (caméras vidéo, capteurs biométriques, capteurs de localisation, etc.) et/ou des effecteurs (écrans, haut-parleurs, moteurs, etc.). Sur chaque processeur peuvent s'exécuter un ou plusieurs agents, chacun spécialisé dans un certain type de traitement. Les données issues des capteurs doivent être traitées par différents agents avant de pouvoir être converties en actions au niveau des effecteurs. L'action des agents consiste à analyser, synthétiser, transformer les données qu'ils reçoivent. Les nouvelles données ainsi produites par un agent sont ensuite transmises à d'autres agents chargés de poursuivre le traitement. Lorsque l'analyse est suffisamment avancée, ou que le temps disponible est insuffisant pour poursuivre cette analyse, les données sont finalement utilisées pour générer des images, des sons ou toute autre action grâce à des effecteurs. La figure 1 montre une vision très simplifiée de ce processus dans laquelle n'apparaît que la modalité de perception correspondant à une caméra vidéo qui filme la scène.

L'utilisation d'agents dans ce contexte se justifie de par la nature distribuée des différents moyens (captation, traitement, action) mis en œuvre dans cette application mais également et surtout pour rendre le système adaptatif dans différents contextes : lorsque des composants sont ajoutés ou supprimés, lorsque l'on souhaite modifier le comportement global du système ou encore lorsque le système ne respecte pas les contraintes temporelles qui lui ont été assignées. La contrainte temporelle essentielle correspond au temps de réponse du système, c'est-à-dire le temps entre le moment où des données sont acquises par les capteurs et le moment où ces données provoquent une réponse au niveau des effecteurs. Ce temps de réponse doit naturellement être maintenu aussi faible que possible pour ne pas provoquer de délai désagréable entre une action du danseur et la réponse correspondante du système. Dans le même temps, l'analyse de la performance du danseur doit rester aussi précise et approfondie que possible. Ces deux contraintes sont potentiellement contradictoires puisqu'une analyse précise et approfondie nécessite évidemment un temps de traitement plus long qu'une analyse grossière et superficielle. La qualité de l'analyse peut être mesurée selon deux dimensions complémentaires, la précision d'analyse (mesure plus ou moins précise d'une caractéristique de la performance) et le niveau d'abstraction (ajout d'une étape supplémentaire de traitement pour obtenir de nouvelles mesures).

On peut formaliser ce compromis en exprimant une mesure de l'efficacité du système (qualité globale) comme une pondération de la qualité de l'analyse (avec quelle précision le système analyse-t-il la performance du danseur ?) et de la qualité temporelle (avec quelle tolérance le système respecte-t-il les contraintes temporelles ?). Dans la mesure où l'on ne place pas dans un contexte de temps-réel dur, ces deux mesures de qualité quantifient par une valeur réelle, le respect relatif des contraintes logiques et temporelles. On peut alors proposer la mesure de qualité globale suivante :

$$qualité\_globale := \alpha * qualité\_analyse + \beta * qualité\_temporelle$$

Il ne s'agit donc pas d'assurer et de prouver le respect de propriétés de correction (de type vrai/faux) mais d'optimiser dynamiquement le critère de qualité globale ci-dessus. L'objectif de la modélisation est alors de pouvoir tester le plus rapidement possible différentes stratégies pour le contrôle de la chaîne de traitement, et de fournir les outils pour automatiser au maximum la phase d'implantation jusqu'à un prototype opérationnel. La modélisation devra aussi pouvoir servir à vérifier a priori le respect par le système de certaines propriétés (temps de réponse global, non-blocage, ordre d'exécution des traitements, etc.). La simulation doit par ailleurs permettre d'obtenir des éléments d'appréciation de la qualité du compromis effectué entre la qualité du traitement effectué et le respect des contraintes temporelles.

3. Introduction aux automates temporisés

Il s'agit pour nous d'étudier l'apport potentiel des techniques de spécification et de vérification de systèmes temps réel pour la conception et la programmation de systèmes à base d'agents dont on souhaite qu'ils aient un comportement global réactif. Parmi l'ensemble des formalismes de spécification de systèmes temps réel, nous avons choisi d'utiliser les automates temporisés (Alur et Dill, 1994) car c'est un formalisme relativement simple à manipuler mais possédant l'expressivité nécessaire pour la modélisation de systèmes concurrents temporisés, et pour lequel existent des outils puissants (Larsen et al., 1998) de model-checking et de simulation.

3.1. *Le modèle standard*

Un automate temporisé est un automate à états finis comportant une représentation du temps continu par l'intermédiaire de variables à valeurs réelles, positives ou nulles, appelées *horloges*, qui permettent d'exprimer des contraintes temporelles. De façon générale, un automate temporisé est représenté par un graphe constitué de places et d'arcs. Une place, associée à l'ensemble des valeurs d'horloge, correspond à un état du système. Un arc définit une transition entre ces états. Les contraintes temporelles s'expriment au travers des *contraintes d'horloges*

et peuvent apparaître sur les places et sur les arcs. Une contrainte d'horloge est une conjonction de contraintes atomiques qui comparent la valeur d'une contrainte *x*, appartenant à l'ensemble fini d'horloges, à une constante rationnelle.

Chaque automate temporisé possède donc un nombre fini de *places* parmi lesquelles on distingue une place dite *initiale*. Dans chaque place, l'écoulement du temps est exprimé par la progression (uniforme) des valeurs d'horloges. Ainsi, dans une place, à tout instant, la valeur d'une horloge *x* correspond au temps écoulé depuis la dernière remise à zéro de *x*. A chaque place est associée une contrainte d'horloge, appelée *invariant*, qui doit être vérifiée pour que le système puisse être dans la place correspondante. Les transitions sont instantanées. Elles sont conditionnées par des contraintes d'horloges, appelées *gardes*, et peuvent remettre certaines horloges à zéro. Elles peuvent aussi porter des *étiquettes* permettant des synchronisations. Un exemple d'automate temporisé et de son exécution possible en fonction du temps est représenté sur la figure 2.

Un système plus complexe, ou un agent, peut être représenté par un produit synchronisé d'automates. L'ensemble des places de cet automate résultant est le produit cartésien des places des automates qui le composent, l'ensemble des horloges est l'union des horloges et similairement pour les étiquettes. Chaque invariant dans l'automate résultant est la conjonction des invariants des places des automates qui le composent et les arcs correspondent à la synchronisation selon les étiquettes des arcs correspondants.

3.2. *Les extensions d'UPPAAL*

Une présentation plus détaillée de l'outil UPPAAL, que nous utilisons pour notre modélisation, peut être trouvée dans (Larsen et al, 1998). Nous rappelons rapidement les principales caractéristiques et extensions par rapport au modèle original.

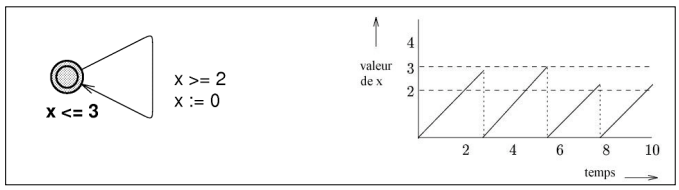


Figure 2. *Exemple d'un automate temporisé où x est une horloge. La garde x>=2 et l'invariant x<=3 impliquent que la transition ne pourra se déclencher qu'après 2 et avant 3 unités de temps passées dans la place.*

Pour UPPAAL, un modèle consiste en un ensemble d'automates temporisés, qui communiquent par une synchronisation binaire, utilisant des étiquettes de transitions et une syntaxe du type émission/réception. Ainsi, par convention, l'étiquette *c !*

indique l'envoi par un émetteur d'un signal sur le canal *c*. Celui-ci est censé être synchronisé avec le signal de réception symbolisé par l'étiquette complémentaire *c ?* du côté du récepteur. L'absence d'étiquette de synchronisation indique une action interne de l'automate. Le formalisme permet également la manipulation de variables discrètes, et leur utilisation dans les gardes et les invariants.

L'exécution du modèle part de la configuration initiale (un état initial de chaque automate avec les valeurs des variables à zéro), et est une suite de configurations accessibles. Le changement de configuration peut se faire de trois façons :

– par écoulement du temps d'une durée *d* dans les états courants des composants, à condition que l'invariant de chacun de ces états reste satisfait. Dans la nouvelle configuration, les valeurs des horloges augmentent de *d* et les valeurs des variables discrètes ne changent pas.

– par synchronisation si deux actions complémentaires de deux composants sont possibles, et que les gardes associées aux transitions sont satisfaites. Les états correspondants sont modifiés dans la nouvelle configuration, et les valeurs des horloges et des variables discrètes sont modifiées selon les indications de remise à zéro et de mise à jour.

– par action interne si une telle action d'un composant est possible, elle peut être exécutée indépendamment des autres composants : l'état et les variables du composant sont modifiés comme dans le cas précédent.

Une autre caractéristique utile pour exprimer le « synchronisme » de mouvements est la notion d'états instantanés ("committed" pour UPPAAL), qui portent l'étiquette C dans les figures. Dans un tel état, aucun délai n'est permis, ce qui impose un mouvement immédiat du composant concerné. Ainsi, deux transitions liées par un état instantané seront exécutées sans délai intermédiaire.

UPPAAL permet de simuler l'exécution du système ainsi spécifié, détecter la présence des blocages (deadlocks) et de vérifier par model-checking des propriétés d'accessibilité. Il peut ainsi répondre à des questions de type « à partir de son état initial, le système peut-il atteindre un état vérifiant une propriété donnée ? », « à partir de l'état initial, une propriété est-elle toujours valide ? » ou encore « à partir de l'état initial, le système peut-il atteindre un état donné dans un délai donné ? ».

4. Modélisation d'un système réactif décentralisé

Le formalisme des automates temporisés permet ainsi la modélisation de systèmes comme un ensemble de processus concurrents. Nous allons détailler incrémentalement la manière dont il peut être appliqué à notre étude.

Le comportement de base de nos agents consiste à traiter des données qui leur sont fournies en entrée pour les transformer et produire de nouvelles données en sortie. Ce traitement a une durée considérée comme fixe, dans des conditions idéales

de fonctionnement, et doit être effectué de manière répétée. En pratique, ce temps sera encadré par des bornes pour tenir compte des différents facteurs susceptibles de modifier cette durée d'exécution. La modélisation correspondante devient ainsi non-déterministe, et sort du cadre classique des méthodes analytiques standards. La modélisation correspondante apparaît sur la figure 3 à gauche.

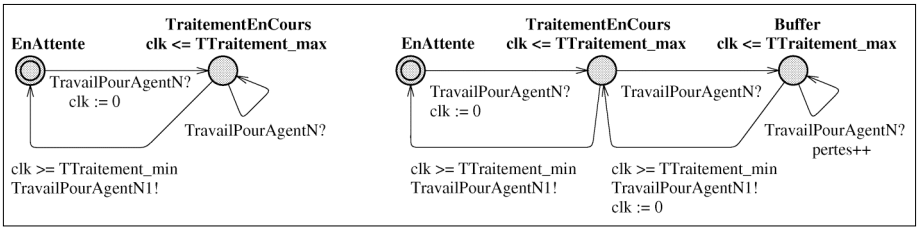


Figure 3. *Modèle d'agent simple (à gauche) et avec buffer (à droite) dédié au traitement de données au sein d'une chaîne de traitements*

Initialement dans l'état `EnAttente`, l'agent démarre son traitement à la réception du signal `TravailPourAgentN`, passant alors dans l'état `TraitementEnCours`. Il revient dans l'état `EnAttente` à la fin de son traitement, soit après un temps compris entre `TTraitement_min` et `TTraitement_max`, informant l'agent suivant qu'il peut démarrer son traitement (synchronisation sur le signal `TravailPourAgentN1`). L'introduction de bornes *min* et *max* sur le temps de traitement correspond ici à une incertitude sur le temps de traitement réel.

L'inconvénient de cette modélisation est que si une demande de traitement parvient à l'agent alors qu'il est en train de traiter la donnée précédente, la donnée correspondante est perdue. Cette boucle sur l'état `TraitementEnCours` est cependant indispensable car son absence donnerait lieu à un blocage lorsque la situation que nous venons de décrire se produit. La solution est d'ajouter un état dans le modèle, équivalent à un buffer (voir figure 3, droite).

A présent, lorsqu'une demande de traitement parvient à l'agent alors qu'il est dans l'état `TraitementEnCours`, ce dernier passe dans l'état `Buffer`, puis il revient dans l'état `TraitementEnCours` à la fin de son traitement, pour en démarrer immédiatement un nouveau. Si une nouvelle demande de traitement parvient à l'agent alors qu'il est déjà dans l'état `Buffer`, la donnée correspondante est perdue, ce que l'on peut comptabiliser en incrémentant à chaque fois une variable `pertes`.

Ce premier type de modélisation correspond cependant à une architecture de traitement des données entièrement figée : un « agent » reçoit des données, les analyse puis les transmet à un autre « agent », toujours le même. La modélisation ne tient donc pas compte de l'adaptation souhaitée en fonction du temps effectif d'exécution des tâches et de la qualité du traitement à obtenir. Pour ce faire, il est

nécessaire d'intégrer un module de contrôle au sein de l'agent pour lui permettre, en fonction des situations rencontrées, d'adapter le traitement qu'il effectue ainsi que le ou les agents auxquels il transmet les données traitées (voir figure 4).

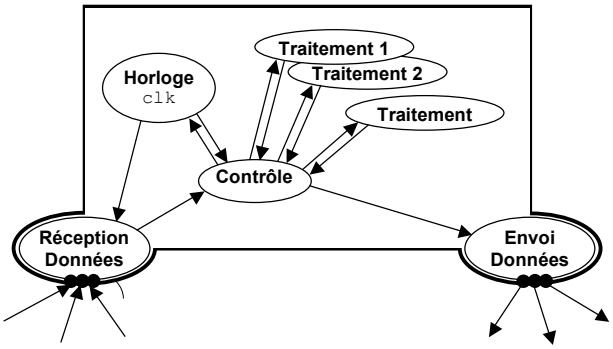


Figure 4. *Modèle d'agent temps-réel adaptatif : les données reçues sont étiquetées temporellement ; en fonction des contraintes à respecter, le contrôleur choisit les traitements à effectuer ; les données analysées sont transmises à un autre agent.*

Dans ce nouveau schéma de fonctionnement, l'agent qui reçoit une donnée lui associe une étiquette temporelle lui permettant de raisonner par rapport à la date d'arrivée des données. En fonction de ces informations temporelles et des contraintes à respecter, le module de contrôle choisit de faire analyser ces données par un ou plusieurs des modules de traitement qu'il a à sa disposition. Une fois analysées, les données peuvent alors être transmises à d'autres agents chargés de poursuivre l'analyse des données ou de les transformer en une réponse pour l'utilisateur.

La figure 5 illustre la modélisation correspondante avec les automates temporisés dans le cas simplifié où l'agent doit choisir entre deux modules de traitement pour chaque donnée à analyser. Lorsqu'une donnée est reçue, l'horloge interne de l'agent est réinitialisée puis l'agent passe dans l'état `Choix`, ce qui revient à donner la main au module de contrôle. En fonction de la valeur de l'expression booléenne `condition_sur_clk`, le module de contrôle choisit de faire effectuer le traitement par l'un ou l'autre des deux modules de traitement puis rend la main. Lorsque le module de traitement a terminé son travail, il informe l'agent grâce au signal `Free`. Ce dernier peut alors transférer les données à l'agent suivant dans la chaîne de traitement.

L'exemple précédent illustre l'utilisation des automates temporisés pour modéliser le comportement adaptatif d'un agent, en rapport avec le traitement qu'il doit effectuer. Le formalisme permet ainsi une décomposition de l'agent en modules dont on modélise le fonctionnement interne et les communications. De manière

similaire, en transposant à l'échelle du système multi-agent, il est possible de modéliser les agents et la circulation des données entre eux, tout en prenant en compte les contraintes de fonctionnement temps-réel. L'adaptation dynamique de l'architecture multi-agent peut ainsi être modélisée ce qui permet de vérifier a priori son fonctionnement et de le simuler.

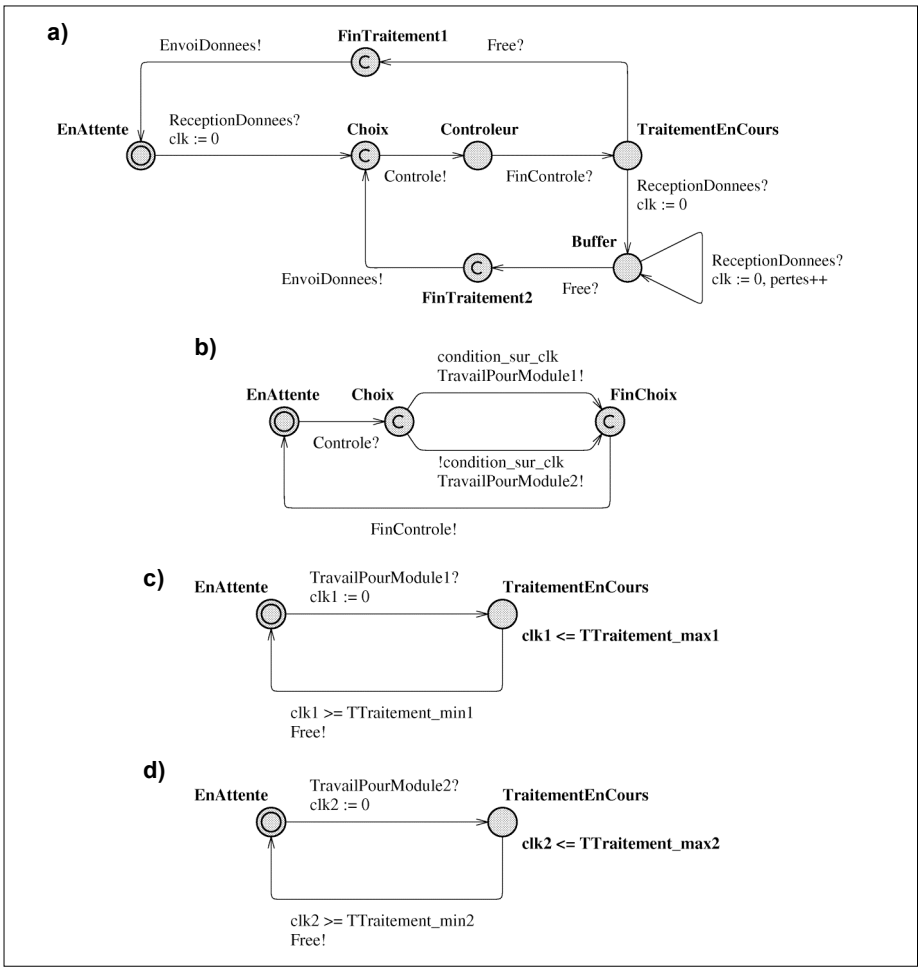


Figure 5. *Modèle d'agent (a) avec un module de contrôle (b) et 2 modules de traitement (c et d).*

5. Vérification et Simulation

Le modèle de contrôleur présenté au paragraphe précédent doit naturellement être instancié en explicitant le ou les critères qui impliquent le choix de l'un ou l'autre des modules de traitement. Des techniques existent pour la synthèse de contrôleurs (Ramadge et Wonham 1978, Tripakis 1998) : à partir d'une spécification du système et d'une propriété attendue de celui-ci, il s'agit de synthétiser un contrôleur qui, placé dans l'environnement et disposant de la spécification, va garantir que les exécutions possibles du système valident cette propriété. L'un des inconvénients de ce type d'approche est que l'on est généralement obligé de se limiter à des spécifications déterministes. Par ailleurs, le type de propriétés que l'on est capable d'exprimer et de vérifier n'est pas approprié dans notre cas. Les propriétés concernent en effet un état ou un sous-ensemble d'états que le système peut atteindre (accessibilité), qu'il doit toujours atteindre (attractivité) ou qu'il doit absolument éviter (invariance). Dans le problème qui nous intéresse, l'objectif est d'optimiser le comportement dynamique du système, ce qui ne s'exprime pas simplement en termes d'états. Comme nous l'avons déjà souligné au paragraphe 2, il ne s'agit pas de propriétés qui sont soit vraies soit fausses, mais d'une mesure de qualité, que l'on cherche à optimiser.

A titre d'exemple, nous présentons dans ce paragraphe sept stratégies possibles et détaillons les vérifications de propriétés et simulations que l'on peut mener avec chacune. Le contexte particulier dans lequel nous nous plaçons pour cette étude est donné sur la figure 6.

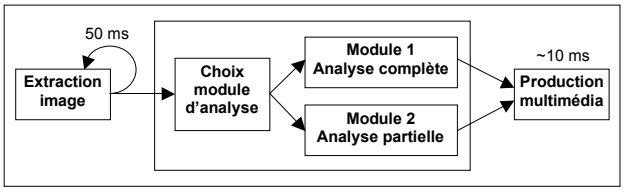


Figure 6. *Schéma de la chaîne de traitement simplifiée*

L'agent d'extraction d'image produit environ toutes les 50ms une image qui doit être traitée par l'agent d'analyse, soit avec un module qui fait une analyse complète, soit avec un module qui ne fait qu'une analyse partielle, mais avec un temps de traitement réduit ($t_{traitement2} < t_{traitement1}$). De même que dans le modèle présenté dans la section 4, ces temps de traitement sont des approximations des temps de traitement réels, et sont encadrés chacun par des bornes *min* et *max*. Le module de choix doit être conçu de manière à concilier deux contraintes potentiellement contradictoires : d'une part analyser toutes les images sans en perdre, c'est-à-dire essayer de respecter au maximum le rythme d'acquisition des données (mesure de qualité

temporelle) ; d'autre part maximiser le nombre d'analyses complètes, c'est-à-dire éviter au maximum de devoir dégrader la précision du traitement (mesure de qualité d'analyse).

5.1. *Stratégies de choix*

Pour commencer, il est nécessaire de définir des stratégies qui pourront être utilisées comme référence. Les autres stratégies pourront alors leur être comparées. On choisit les trois stratégies de référence suivantes : la stratégie 1 utilise toujours le module 1 pour le traitement, de manière à maximiser le nombre d'images pour lesquelles une analyse complète est effectuée ; la stratégie 2 utilise toujours le module 2 pour le traitement de manière à minimiser le nombre d'images perdues ; la stratégie 3 alterne les deux modules, ce qui constitue un premier compromis possible entre analyse fine et respect des contraintes temporelles.

A partir de ces stratégies de référence, une première approche peut être d'essayer de minimiser les pertes d'images. Pour ce faire, l'idée est d'anticiper, au moment du choix (t_{choix}), le moment où l'agent recevra une image à traiter alors qu'il a déjà une image en attente dans son buffer et qu'il n'a pas terminé l'analyse en cours (t_{perte}). Cela est rendu possible par l'arrivée régulière des images à traiter. Dans la stratégie 4, le module 1 sera alors choisi si et seulement si : $t_{traitement1} < t_{perte} - t_{choix}$. La stratégie 5 est encore plus restrictive et n'autorise un traitement par le module 1 que si l'image suivante pourra être traitée sans perte par le module 2 ($t_{traitement1} + t_{traitement2} < t_{perte} - t_{choix}$).

Une autre approche est d'essayer de maximiser le nombre d'analyses complètes. Pour ce faire, on peut relâcher la contrainte précédente et autoriser l'utilisation du module 1 même si cela doit nécessairement conduire à la perte d'une image. Dans la stratégie 6, le module 1 ne sera choisi que si $t_{traitement1} < (t_{perte} - t_{choix}) * coef$, où *coef* détermine la limite de tolérance admise. Par la suite, la stratégie 6 correspondra à une valeur de 1,25 pour *coef*. Finalement, pour que l'analyse des données soit suffisamment bonne, on peut considérer qu'il est nécessaire d'effectuer une analyse complète au moins une fois toutes les n images. Pour cela, le contrôleur de la figure 5 doit être légèrement modifié. Il fonctionnera en deux étapes : il commence par vérifier qu'il y a eu moins de $n-1$ images traitées depuis le dernier appel au module 1 ; si la limite est atteinte, le module 1 est appelé ; si ce n'est pas le cas, le contrôleur choisit le module en utilisant la stratégie 4.

5.2. *Résultats*

L'utilisation de techniques de model-checking permet de prouver automatiquement que certaines propriétés considérées comme importantes pour le système resteront toujours valides quelle que soit l'évolution du système. Ainsi,

pour chaque stratégie, on a pu vérifier formellement au moyen d'outils existants comme UPPAAL (Larsen et al., 1998) que certaines propriétés sont respectées par le système. Les propriétés sont exprimées par des formules de logique temporelle puis soumises au moteur de vérification d'UPPAAL. Dans un premier temps, il faut s'assurer qu'il n'y a pas de deadlocks dans le modèle : `A[] not deadlock`.

L'utilisation de ces techniques permet par ailleurs de vérifier des propriétés exprimant la qualité du système modélisé, par exemple :

– qu'aucune image n'est perdue : `A[] pertes == 0`

– que la proportion d'appels au module 1 est supérieure à un seuil donné :
`A[] (nb1 * 100 / (nb1 + nb2)) > 50`)

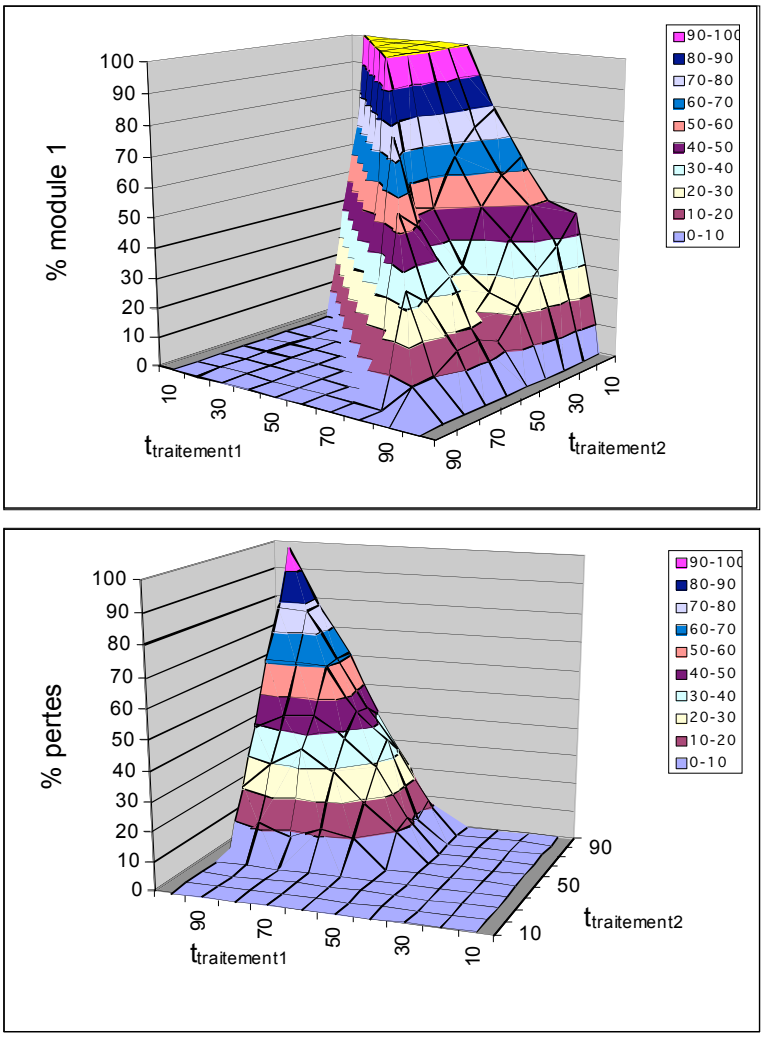
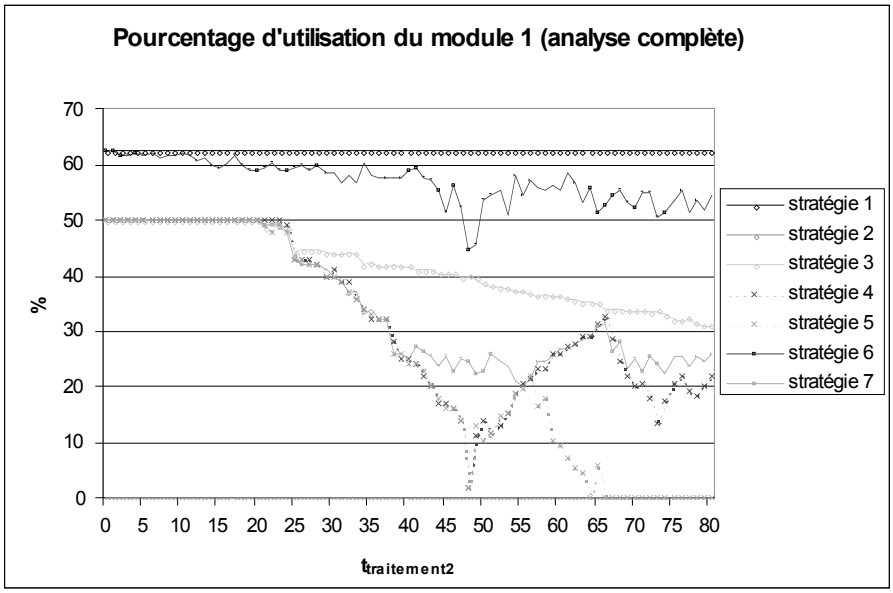


Figure 7. *pourcentage d'images analysées avec le module 1 (en haut) et pourcentage d'images perdues (en bas) obtenus avec la 4ème stratégie pour différentes valeurs de temps de traitement pour les modules 1 et 2*



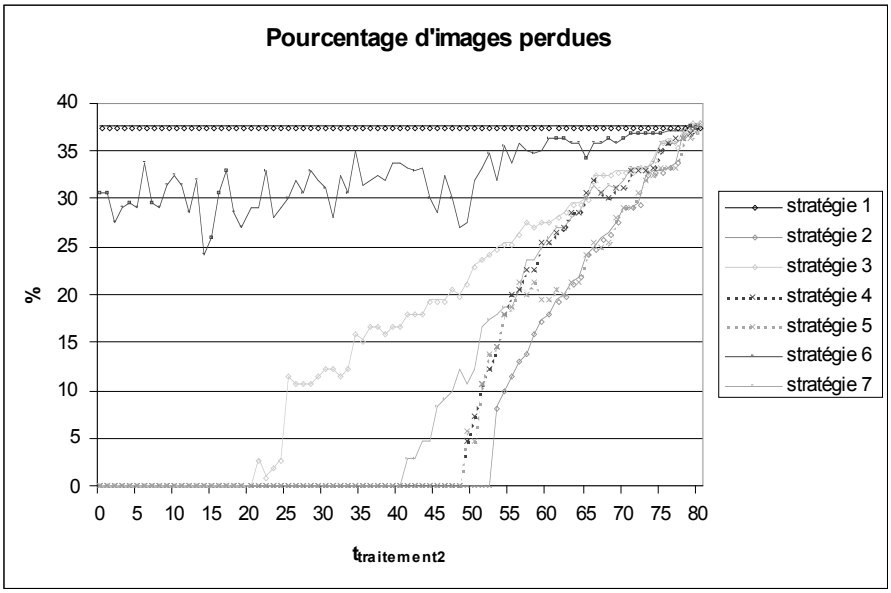


Figure 8. *Comparaison des 7 stratégies pour $t_{traitement1} = 80$ ms, par rapport au pourcentage d'analyses complètes et au pourcentage d'images perdues, en faisant varier $t_{traitement2}$; le caractère irrégulier de certaines courbes reflète le non-déterminisme du système sous-jacent*

De manière complémentaire, l'utilisation de la simulation permet d'obtenir une évaluation empirique des performances du système en termes de qualité logique et temporelle, en fonction des caractéristiques des modules de traitement et du type de stratégie appliquée. En simulant le fonctionnement du système pendant une centaine de cycles, on a pu évaluer expérimentalement les proportions d'images perdues et d'images analysées complètement en fonction des temps de traitement $t_{traitement1}$ *et* $t_{traitement2}$, comme illustré dans les figures 7 et 8. Grâce à ces expérimentations, il est possible d'évaluer l'efficacité de chaque stratégie pour assurer une certaine qualité de traitement, ce qui permet d'envisager un niveau supplémentaire dans le contrôle de l'agent d'analyse des images, correspondant à une « méta-stratégie » qui adapterait dynamiquement la stratégie de choix en fonction des contraintes et des objectifs assignés.

De même, en se plaçant au niveau du système multi-agent dans son ensemble, il est possible de modéliser différentes stratégies pour la réorganisation adaptative du système multi-agent de traitement des données puis de tester ces stratégies par la vérification et la simulation.

5.3. *Retour sur la conception*

Comme nous l'avons vu dans les sections précédentes, les différentes stratégies choisies ne sont pas équivalentes par rapport à la correction logique (avec quelle fréquence l'analyse complète est-elle faite ?) et temporelle (quel pourcentage de données est perdu ?). En fonction des valeurs de $t_{traitement1}$ et $t_{traitement2}$, la « meilleure » stratégie n'a pas toujours été la même, ce qui signifie qu'aucune stratégie n'est universelle. Evidemment, la « meilleure » dépend des critères que nous avons pu retenir pour évaluer son efficacité. Dans ce contexte, ce qui semble intéressant, c'est la possibilité de changer de stratégie lorsque les conditions changent : ou bien parce que les temps de traitement caractéristiques du module ont changé (à cause d'une répartition différente de la charge du processeur) ou bien parce que les contraintes temps réel ont été modifiées. Ceci peut être pris en compte en ajoutant un module supplémentaire, un méta-contrôleur, qui serait chargé de passer dynamiquement d'un contrôleur (associé à une stratégie particulière) à un autre, en fonction des conditions d'exécution. Ceci est illustré dans la figure 9.

A nouveau, différentes stratégies peuvent être proposées pour ce méta-contrôleur. Nous allons expliquer uniquement comment l'une d'elles pourrait être conçue. On peut considérer, par exemple, le critère défini par la formule suivante :

$$efficacité := \alpha * \%module_1 - \beta * \%pertes$$

où α et β correspondent respectivement aux poids affectés à la qualité logique et temporelle.

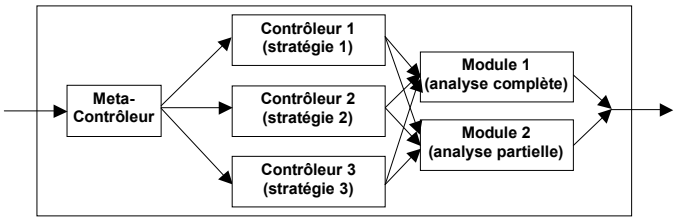


Figure 9. *La stratégie de choix du module de traitement est choisie dynamiquement par un méta-contrôleur, en fonction des paramètres des modules de traitement et des contraintes temps-réel.*

La figure 10 montre une comparaison de différentes stratégies pour ce critère, pour les mêmes conditions que dans la figure 7. On peut y observer, en particulier, qu'en fonction des temps caractéristiques de traitement des modules et en fonction des contraintes temps-réel fixées, les agents doivent adapter leur stratégie de contrôle. Cette façon de faire les autorise à adapter leur comportement dynamiquement lorsque les conditions extérieures changent. Ceci peut être modélisé par les automates de la figure 11.

6. Génération automatique de code

Une fois que le modèle du système multi-agent de traitement de données a été validé à la fois formellement et expérimentalement, il reste encore à le transformer en un modèle exécutable. Pour ce faire, une première idée pourrait conduire à implémenter chaque automate sous forme d'un thread, puisqu'il s'agit de modèles de processus concurrents. Pour un même agent, modélisé par plusieurs automates, cela pourrait cependant conduire à une multiplication des problèmes de synchronisation et se traduire par une baisse sensible de performances, ce qui peut être gênant pour un système réactif. La solution retenue est de ne considérer qu'un seul thread par agent, plutôt qu'un thread par automate. Le système entier est alors implanté sous forme d'une application multi-threadée.

6.1. *Produit synchronisé d'automates*

Une première étape consiste donc à faire le produit synchronisé des automates modélisant un même agent puis à transformer l'automate ainsi obtenu en squelette d'application. Le compilateur que nous avons développé réalise ce produit synchronisé de la manière décrite à la section 3.1. Si l'on fait, par exemple, le produit synchronisé des quatre automates modélisant l'agent de traitement reproduit

à la figure 5, l'automate résultant comporte 84 états et 388 transitions ! Lorsque nous réalisons le produit synchronisé d'automates, nous distinguons cependant deux types de signaux : les signaux « locaux » ou « internes » qui n'apparaissent que dans les automates dont on fait le produit synchronisé, correspondant à une synchronisation entre ces automates ; les signaux « globaux » ou « externes » qui apparaissent également dans d'autres automates, correspondant à une synchronisation avec ces autres automates. Les transitions qui apparaissent dans le produit synchronisé et qui sont étiquetées avec des signaux locaux n'ont aucune chance d'être franchies puisque ces signaux ne sont présents dans aucun autre automate. Les transitions correspondantes peuvent donc être supprimées (voir figure 12.b). Dans l'exemple précédent, on n'a plus alors que 84 états et 120 transitions.

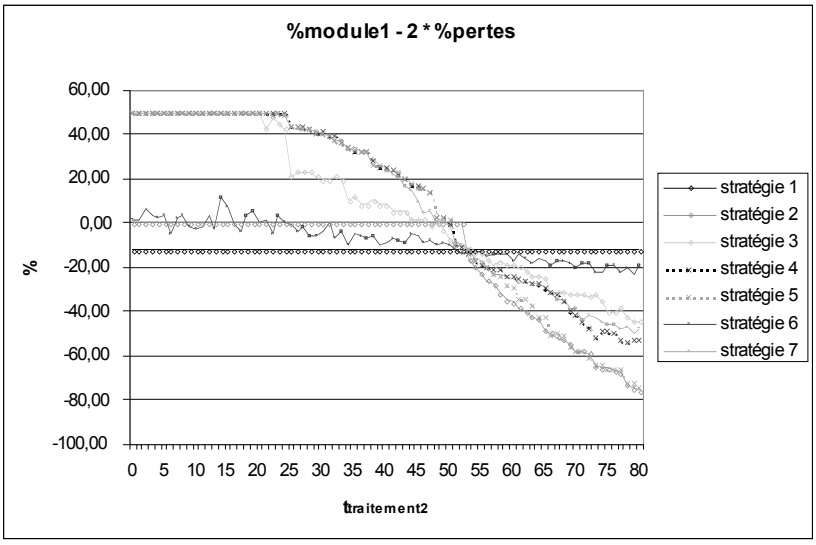


Figure 10. *Comparaison de sept stratégies pour $t_{traitement1}$=80ms, α=1 et β=2 ; pour les valeurs de $t_{traitement2}$ inférieures à 20, les meilleures stratégies sont 3, 4, 5 et 7 ; pour les valeurs de $t_{traitement2}$ entre 20 et 50, les meilleures stratégies sont 4 et 5 ; pour les valeurs autour de 50, toutes les stratégies sont pratiquement équivalentes, mais la meilleure est la 2 (elle appelle toujours le module 2) ; enfin, pour les valeurs au-dessus de 55, la meilleure stratégie est la 1 (elle appelle toujours le module 1).*

Suite à la suppression de ces transitions, certains états deviennent inaccessibles, et peuvent également être supprimés ainsi que les transitions éventuelles qui en partent. D'autres états correspondent à des deadlocks parce qu'ils n'ont que des transitions entrantes ou en boucle. Si l'on s'est assuré au départ que le système n'avait pas de deadlock, on peut également supprimer ces états ainsi que les transitions qui y arrivent. Par extension, tous les états qui ne sont pas atteignables à

partir de l'état initial (problème de couverture dans un graphe orienté) peuvent être supprimés. Toujours pour le même exemple, on n'a plus alors que 11 états et 16 transitions.

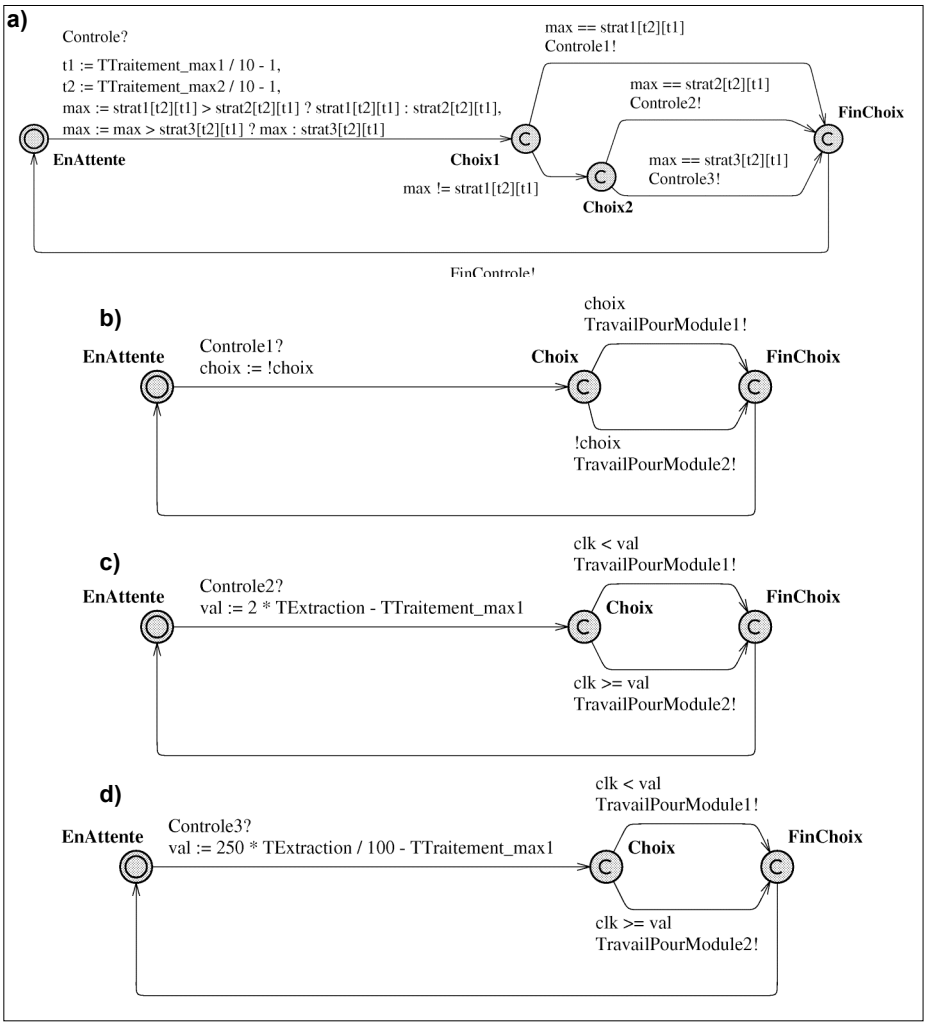


Figure 11. *Le méta-contrôleur modélisé par un automate temporisé (a) ; la meilleure stratégie (max) est calculée en fonction des temps caractéristiques de traitement des modules t1 et t2 ; le module du contrôleur correspondant est alors choisi parmi les différentes stratégies possible : le contrôleur b) correspond à la stratégie 3, le c) à la stratégie 4 et le d) à la stratégie 6.*

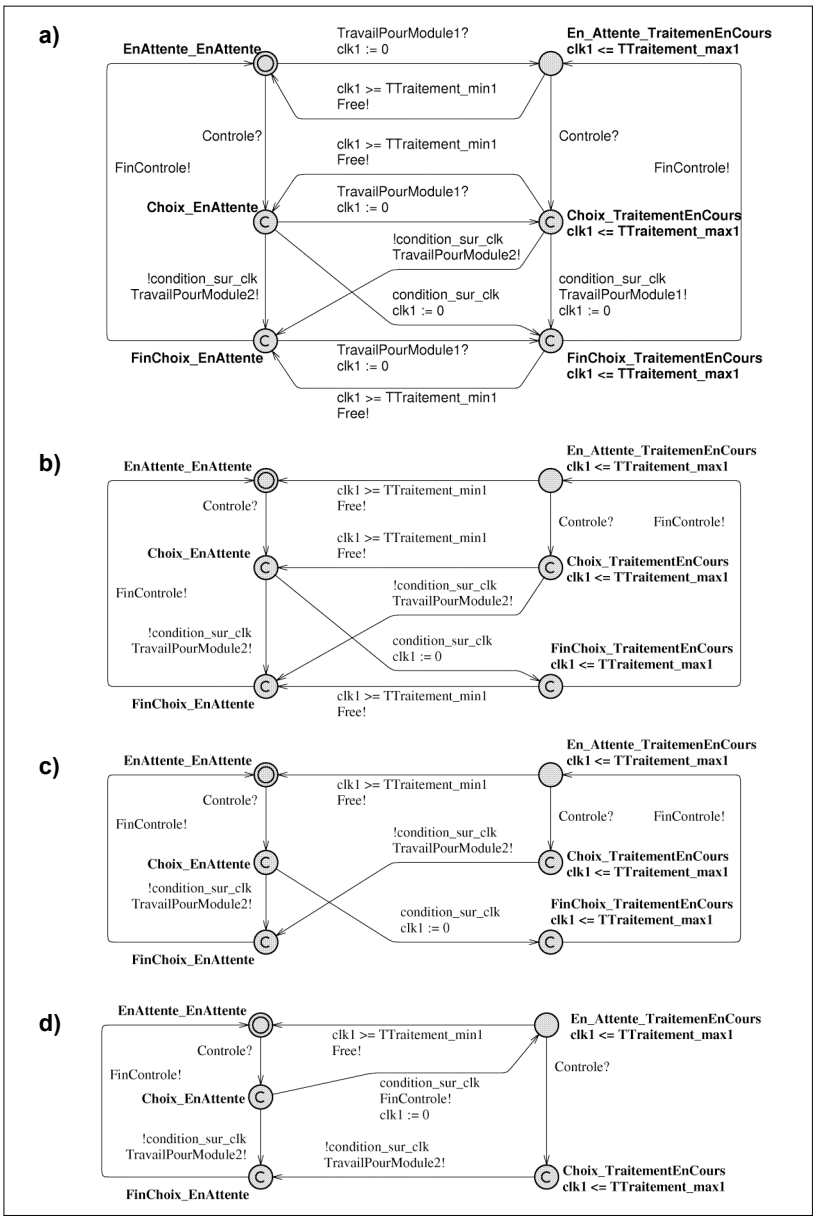


Figure 12. *Automate obtenu en effectuant le produit synchronisé de deux des quatre automates de la figure 5, correspondant au contrôleur et au premier module de traitement ; a) produit synchronisé complet ; b) suppression des signaux locaux ; c) suppression des transitions inutiles sortant d'états instantanés ; d) compression d'états instantanés.*

Le compilateur effectue ensuite une optimisation liée aux propriétés des états instantanés, pour minimiser la taille de l'automate. Cette optimisation se justifie par le fait que, lorsque l'on fusionne deux états dont l'un seulement est un état instantané, seules les transitions sortant de l'état instantané sont autorisées. Il est facile de voir en effet que la fusion de deux états dont l'un au moins est un état instantané, conduit a un état qui est également un état instantané. Par ailleurs, puisqu'il est interdit de laisser le temps s'écouler dans un état instantané, les seules transitions autorisées quand on se trouve dans l'un de ces états sont les transitions permettant de quitter l'état. Ce sont donc celles qui, dans l'automate de départ, permettaient de sortir de l'état instantané. Les autres transitions n'ont aucune chance d'être franchies et peuvent donc être supprimées (voir figure 12.c). De la même manière que précédemment, cela peut conduire, dans certains cas, à des états qui ne sont plus connectés au reste de l'automate et peuvent eux aussi être supprimés.

Un dernier type d'optimisation correspond aux états instantanés qui n'ont qu'une seule transition sortante, ce qui signifie qu'il n'y a qu'une seule possibilité pour poursuivre l'exécution de l'automate. Dans certaines conditions, on peut alors fusionner cet état avec l'état suivant, en fusionnant également la transition sortante avec chacune des transitions entrantes (voir figure 12.d). En particulier, si la transition sortante porte un signal, aucune des transitions entrantes ne doit porter de signal, car une transition ne peut pas porter simultanément deux signaux. La nouvelle garde est alors la conjonction des gardes portées par les transitions et les affectations sont combinées en séquence. Pour le produit synchronisé des quatre automates de la figure 5, on obtient finalement un automate à 8 états et 13 transitions (figure 13).

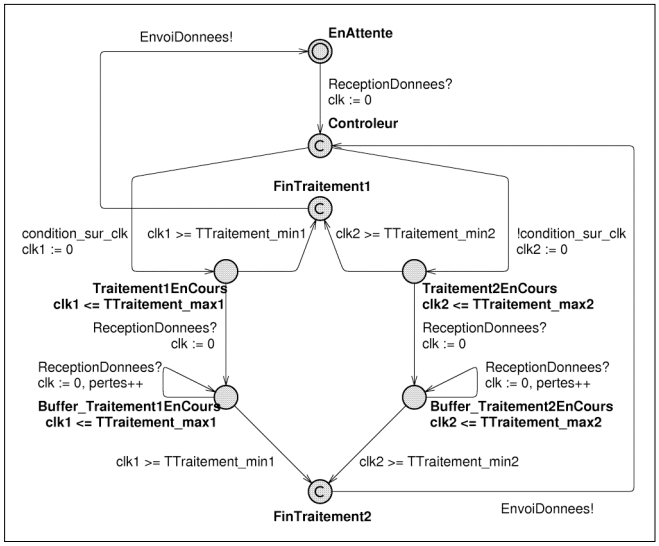


Figure 13. *Automate obtenu en effectuant puis en optimisant le produit synchronisé des quatre automates de la figure 5.*

6.2. *Implantation*

A l'issue de l'étape de l'étape précédente, chaque agent est modélisé par un automate unique que l'on peut alors traduire sous forme exécutable en plusieurs étapes. Nous nous sommes appuyés, pour la génération de code, sur la plate-forme multi-agent JADE (Bellfemine et al. 1999) qui propose un ensemble de bibliothèques et de services pour l'implantation de systèmes multi-agents qui se conforment à la norme FIPA.

Le compilateur que nous avons développé analyse le fichier XML produit par Uppaal et génère automatiquement, à partir de chacun des automates, le squelette d'un agent.

A chaque automate peut être associé un ensemble de variables discrètes. Ces déclarations étant spécifiques à un automate, le compilateur les transforme en autant d'attributs pour l'agent correspondant. De même, des horloges peuvent être associées aux automates, dont l'avancement sera calculé par rapport à l'horloge interne de l'agent. Chaque affectation d'horloge correspond à l'initialisation de l'origine des temps par rapport à cette horloge interne et chaque lecture correspond à la mesure du temps écoulé depuis cette origine.

Dans le même temps, des variables discrètes et horloges globales peuvent être définies. Puisque l'on souhaite que l'application finale soit distribuée (des agents qui fonctionnent sur des machines différentes) et décentralisée (sans contrôle centralisé de la part d'un agent particulier), ces données globales sont potentiellement problématiques. La présence de telles données globales dans le modèle peut se justifier de différentes manières : dans un objectif de mise au point et de simulation, il peut par exemple être utile de compter le nombre de fois où un automate est passé par une place donnée ; par ailleurs, du fait que le formalisme des automates temporisés ne permet pas le passage de données avec les signaux, il peut être utile d'utiliser des variables globales à cet effet (l'automate émetteur modifie la variable globale lors de l'envoi du signal, l'automate récepteur lit cette même variable à la réception du signal). Dans le premier cas, il s'agit d'informations liées à la mise au point du modèle, et qui peuvent donc être supprimées sans difficulté lors du passage à l'implantation. Dans le second cas, il s'agit d'un envoi de message entre agents, pour lequel on substitue la modification de variable globale par un paramètre transmis avec le message. Au moment de l'implantation, nous n'autorisons que ce deuxième type d'utilisation de variable globale.

Notre compilateur identifie tout d'abord l'ensemble des données globales (horloges et variables) présentes dans le modèle et alerte le concepteur des problèmes potentiels liés à ces données. Le concepteur a alors le choix de modifier le modèle ou de continuer la phase d'implantation. Dans ce dernier cas, le compilateur identifie tous les signaux de synchronisation entre les automates. Puisque chaque automate modélise un agent, les signaux peuvent être assimilés à des messages échangés entre les agents. Dans le cas de notre implantation avec la plate-forme JADE, il s'agit de messages asynchrones de type ACL (Agent Communication Language). Lorsque les arcs porteurs de ces signaux sont également

associés à des affectations de variables globales, ces variables sont transmises comme paramètres du message, avant d'être lues par le destinataire.

De manière plus globale, l'automate temporisé est transcrit sous forme d'un automate à états finis. Les places dans lesquelles il est nécessaire de laisser s'écouler le temps (places associées à un invariant de type $t <= t_{max}$, avec une transition sortante associée à une garde de type $t >= t_{min}$) sont supposées correspondre à un traitement. Cela signifie que l'agent n'est pas censé attendre dans la place un temps compris entre t_{min} et t_{max} mais qu'il est censé effectuer un traitement dont on a estimé la durée entre t_{min} et t_{max}. L'appel au module de traitement doit être ajouté manuellement à l'issue de la phase de génération automatique.

6.3. *Validation*

L'étape de validation consiste à s'assurer que le système une fois implanté possède un comportement similaire à celui du modèle, tel qu'on a pu le simuler. En l'occurrence, le comportement obtenu apparaît sensiblement différent. Hormis le fait que la plate-forme d'exécution n'est pas temps-réel, la principale différence provient du fait que les signaux synchrones et instantanés du modèle sont remplacés par des messages asynchrones, dont l'acheminement prend du temps. Une solution pourrait être d'adopter une plate-forme temps-réel pour laquelle la gestion des messages est optimisée, ce qui réduirait le temps d'acheminement. Ce temps ne pourrait cependant jamais être nul, et cela ne serait pas conforme à l'esprit de ce travail qui vise à l'implantation du modèle sur des plates-formes agent respectant au maximum les normes de communication et d'interaction agent. Il apparaît donc nécessaire de prendre en compte ce mode de communication asynchrone à l'intérieur du modèle.

Pour ce faire, une solution possible consiste à associer, à chaque échange de signal entre automates, un automate *messager* : l'automate *expéditeur* envoie un signal à l'automate *messager* ; ce dernier bascule dans une place où il restera pendant un temps correspondant au temps d'acheminement du message (estimé d'après l'exécution du système implanté) ; à l'issue de ce temps, l'automate messager revient dans l'état initial, en envoyant un signal à l'automate *destinataire*. Avec cette modification, nous avons pu valider que le système une fois implanté se comportait qualitativement de manière très similaire à la simulation du système modélisé.

7. Conclusion

Nous avons montré dans cet article l'apport de l'utilisation des automates temporisés dans le processus de modélisation d'un système d'agents, à la fois sous l'angle du traitement de données et sous l'angle de la reconfiguration dynamique de la chaîne de traitement. Plus précisément, l'approche proposée permet de contrôler le compromis de qualité entre des contraintes antagonistes, en choisissant de manière adaptative les modules de traitement à utiliser par un agent et les

connexions entre les agents. Pour ce faire, nous avons montré qu'il était possible de modéliser de façon modulaire un contrôleur d'agent, pour raisonner et prendre des décisions en fonction des objectifs assignés. Le formalisme se prête bien par ailleurs à l'utilisation de techniques de model-checking et de simulation, ce qui permet d'obtenir une connaissance théorique du comportement du système et de tester a priori différents comportements d'agents. Même si le model-checking est limité à la vérification de systèmes de petite taille, les techniques de simulation resteront applicables quelle que soit la taille du système. Enfin, le formalisme peut constituer le support de la génération semi-automatique du squelette d'un premier prototype.

A plus long terme, l'objectif est de proposer, en utilisant le formalisme décrit dans cet article, des patrons de conception et des outils de composition pour faciliter la modélisation d'un système entier, dans une perspective de prototypage rapide. Ces patrons de conception seront couplés à des techniques d'apprentissage et d'exploration de l'espace des paramètres pour optimiser automatiquement les comportements des agents lorsque le modèle est plus complexe. Pour réaliser l'objectif d'une chaîne complète de conception et de développement, l'approche présentée pourra être complétée en amont par une méthodologie permettant l'expression des besoins. Symétriquement, il est prévu de développer un protocole expérimental pour valider, sur le prototype réel, les propriétés observées sur le modèle. Dans cet esprit, le travail présenté, s'il ne constitue qu'une étude préliminaire, démontre cependant la faisabilité de l'approche et laisse présager favorablement le développement d'outils puissants et complets pour la modélisation et l'implantation de systèmes multi-agents réactifs.

8. Bibliographie

Abdeddaïm Y., Asarin E., Maler O., "Scheduling with Timed Automata", in *Theoretical Computer Science*, 2004.

Alur R., Dill D. L., "A Theory of Timed Automata", in *Theoretical Computer Science*, Vol. 126, No. 2, April 1994, pp. 183-236, 1994.

Atkins E. M., Abdelzaher T. F., Shin K. G., Durfee E. H., "Planning and Ressource Allocation for Hard Real-Time, Fault Tolerant Plan Execution", in *Journal of Autonomous Agents and Multi-Agent Systems*, Vol. 4, No. 1/2, pp. 57-78, March 2001.

Attoui A., *Les systèmes multi-agents et le temps-réel*, Eyrolles, Paris, 1997.

Bellfemine F., Poggi A., Rimassa G., "JADE – A FIPA-compliant agent framework", in *Proceedings of Practical Applications of Intelligent Agents and Multi-Agent Technology*, London, pp. 97-108, 1999.

Blazewicz, J., Ecker K., Schmidt G., Weglarz, J., *Scheduling in computer and manufacturing systems*, Springer-Verlag, Berlin, 1993.

Foundation for Intelligent Physical Agents, `http://www.fipa.org`, 1996-2005.

Hutzler G., Gortais B., Joly P., Orlarey Y., Zucker J.-D., « *J'ai dansé avec machine* ou comment repenser les rapports entre l'homme et son environnement », in *JFIADSMA'2002*, pp.147-150, Hermès Science, Paris, 2002.

Jéron T., Marchand H., Rusu V., Tschaen V., « Synthèse de contrôleurs pour une relation de conformité », in *4ième Colloque Francophone sur la Modélisation des Systèmes Réactifs, MSR'03,* Metz, France, Octobre 2003.

Larsen K. G., Pettersson P., Yi W., "UPPAAL in a Nutshell", in *Springer International Journal of Software Tools for Technology Transfer*, 1(1-2), pp. 134-152, 1998.

Maler O., On Optimal and Sub-optimal Control in the Presence of Adversaries, in *Proc. WODES'04*, 2004.

Marc F., Degirmanciyan-Cartault I., El Fallah-Seghrouchni A., « Modélisation et synchronisation de plans multi-agents contraints, application aux missions aériennes », in *JFSMA 2003*, pp. 143-157, Hermès-Lavoisier, Paris, 2003.

Musliner D. J., Durfee E. H., Shin K. G., "CIRCA: A Cooperative Intelligent Real-Time Control Architecture", in *IEEE TSMC*, 23(6), pp. 1561-1574, 1993.

Musliner D. J., Goldman R. P., Krebsbach K. D., "Deliberation Scheduling Strategies for Adaptive Mission Planning in Real-Time Environments", in *Proc. Third International Workshop on Self Adaptive Software*, 2003, http://www.cs.umd.edu/users/musliner/papers/safer03.ps.gz.

Occello M., Demazeau Y., Baeijs C., "Designing Organized Agents for Cooperation with Real-Time Constraints", in CRW'98, Paris, pp. 25-37, Springer-Verlag, 1998.

Ramadge P.J., Wonham W.M., "Supervisory control of a class of discrete-event processes", in *SIAM Journal of Control and Optimization*, 25:206--230, 1987.

Smith S. F., et Becker M., "An ontology for constructing scheduling systems", in *Working Notes of 1997 AAAI Symposium on Ontological Engineering*, AAAI Press, 1997.

Soler J., Julian V., Rebollo M., Carrascosa C., Botti V., "Towards a Real-Time Multi-Agent System Architecture", in *COAS, AAMAS'2002*, Bologne, 2002, www.agentcities.org/Challenge02/Proc/Papers/ch02_22_soler.pdf.

Tripakis S., *L'analyse formelle des systèmes temporisés en pratique*, Thèse de l'Université Joseph Fourier, Grenoble, 1998.

Wolfe V. F., DiPippo L. C., Cooper G., Johnston R., Kortman P., Thuraisingham B. M., "Real-Time CORBA", in *IEEE TPDS*, v. 11, no. 10, pp. 1073-1089, Oct. 2000.