HUTZLER G., KLAUDEL H., WANG D. Y., « Automates temporisés et systèmes multi-agents temps-réel », *Journées Francophones sur les Systèmes Multi-Agents 2004*, pp. 69-82, Hermès Science, Paris, 2004.

Automates temporisés et systèmes multi-agents temps-réel

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. 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 correction 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 satisfait les mêmes propriétés.*

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. We use the timed automata formalism for the design of the agents' behaviour. This allows evaluating beforehand the properties of the system (regarding logical correctness and timeliness), 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 satisfying the properties of the model.*

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). 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.

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 correction logique souhaitée, de la correction temporelle mesurée 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.

Nous nous plaçons dans une démarche de génie logiciel, en proposant 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 montrons que ce formalisme permet la 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 (section 4). Nous illustrons ensuite 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.

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.

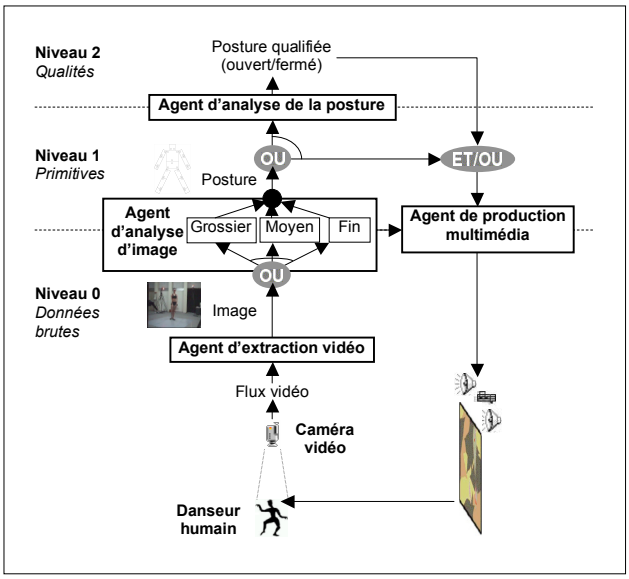

Figure 1. *Architecture globale de la chaîne de traitement pour la modalité visuelle dans le projet « J'ai dansé avec Machine »*

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).

L'objectif de la modélisation est 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 doit ainsi permettre de 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 théoriques d'appréciation de la qualité du compromis effectué entre correction logique (la qualité du traitement effectué est-elle bonne ?) et correction temporelle (le système tient-il la cadence ?).

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 où chaque sommet correspond à un état du système et les arcs aux transitions entre ces états. Les contraintes temporelles s'expriment au travers des *contraintes d'horloges* et peuvent porter sur les états et sur les transitions. 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 d'*états* parmi lesquels on distingue un état dit *initial*. Dans chaque état, l'écoulement du temps est exprimé par la progression (uniforme) des valeurs d'horloges. Ainsi, dans un état, à tout instant, la valeur d'une horloge x correspond au temps écoulé depuis la dernière remise à zéro de x. A chaque état est associée une contrainte d'horloge, appelée *invariant*, qui doit être vérifiée pour que le système puisse être dans l'état correspondant. Les transitions entre les états 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.

Le comportement d'un système complexe peut être représenté par un unique automate temporisé qui résulte du produit synchronisé de plusieurs autres. L'ensemble des états de cet automate résultant est le produit cartésien des états 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 états 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.

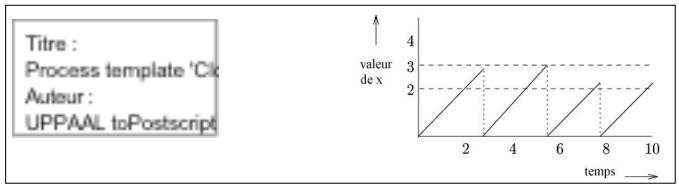

Figure 2. *Exemple d'un automate temporisé où x est une horloge. La garde x>=2 et l'invariant x<=3 impliquent que la transition ne pourra se déclencher qu'après 2 et avant 3 unités de temps passées dans l'état.*

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.

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. Typiquement, il peut 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 reste-t-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 et doit être effectué de manière répétée. La modélisation correspondante apparaît sur la figure 3 à gauche.

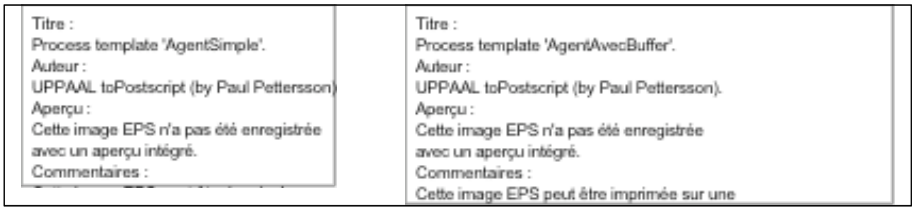

Figure 3. *Modèle d'agent simple (à gauche) et avec buffer (à droite) dédié au traitement de données au sein d'une chaîne de traitements*

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'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`.

A ce stade, il nous reste à modéliser le fait que plusieurs modules sont disponibles (correspondant à différentes précisions dans le traitement) pour effectuer l'analyse de la posture du danseur. Une première approche consiste à dupliquer l'agent de traitement correspondant, en y associant des constantes de temps de traitement différentes. Lorsqu'une donnée sera disponible, elle sera transmise à l'un ou l'autre des agents de traitement de manière non-déterministe. Il est donc nécessaire d'ajouter un module de contrôle dans l'agent pour lui permettre de choisir le module de traitement à utiliser dans chaque cas. C'est ce que montre la figure 4.

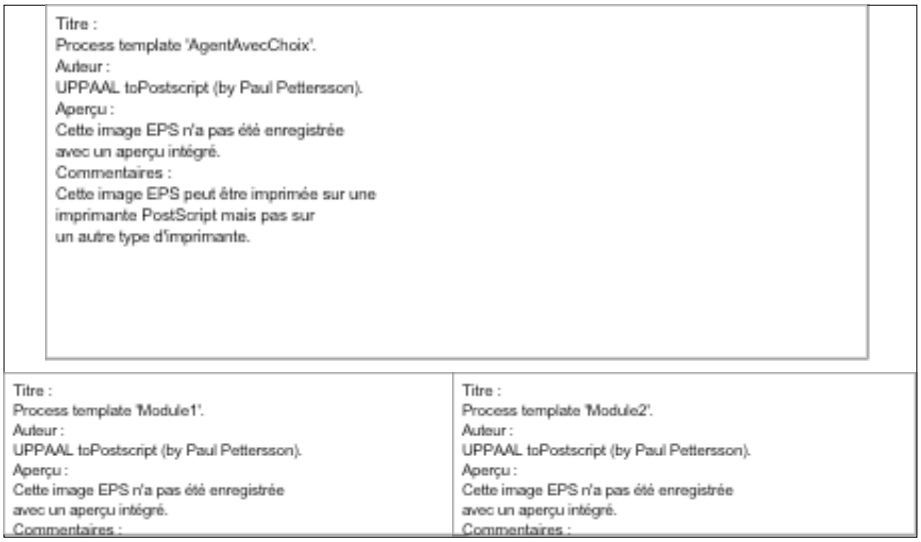

Figure 4. *Modèle d'agent avec un module de contrôle (en haut) et 2 modules de traitement (en bas).*

Lorsqu'une donnée est prête pour être traitée, le module de contrôle de l'agent passe dans l'état `Choix`. En fonction de la valeur de l'expression booléenne `condition_sur_clk`, l'agent choisira d'effectuer le traitement en utilisant l'un ou l'autre des deux modules de traitement. Lorsque le module de traitement a terminé son travail, il informe le module de contrôle grâce au signal `Free`. Le module de contrôle peut alors informer l'agent suivant dans la chaîne de traitement.

L'exemple ci-dessus 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.

5. Vérification et Simulation

Le modèle de choix 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. Nous présentons dans ce paragraphe trois 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 5.

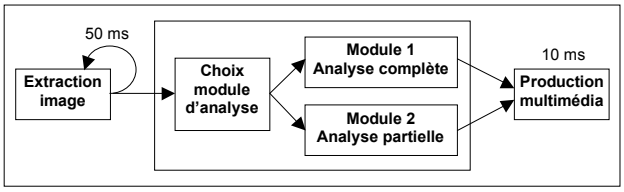

Figure 5. *Schéma de la chaîne de traitement simplifiée*

L'agent d'extraction d'image produit 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 ($t_{traitement2} < t_{traitement1}$). Le module de choix doit être conçu de manière à concilier deux critères potentiellement contradictoires : d'une part analyser toutes les images sans en perdre (critère de correction temporelle) ; d'autre part maximiser le nombre d'analyses complètes (critère de correction logique).

5.1. Stratégies de choix

La première stratégie n'en est pas réellement une mais est donnée à titre de référence. Elle consiste uniquement à alterner systématiquement les deux modules.

La deuxième stratégie vise à 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 deuxième stratégie, le module 1 sera alors choisi si et seulement si : $t_{traitement1} < t_{perte} - t_{choix}$.

Si l'on souhaite, dans une troisième stratégie, maximiser le nombre d'analyses complètes, on peut relâcher la contrainte précédente en autorisant l'utilisation du module1 même si l'on sait que son exécution entraînera nécessairement une perte d'image : $t_{traitement1} < (t_{perte} - t_{choix}) * tolérance$.

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, exprimées par des formules de logique temporelle, sont soumises au moteur de vérification d'UPPAAL, ce qui nous a permis de prouver, dans certains cas :

– qu'il n'y a pas de deadlocks : `A[] not deadlock` ;

– 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`)

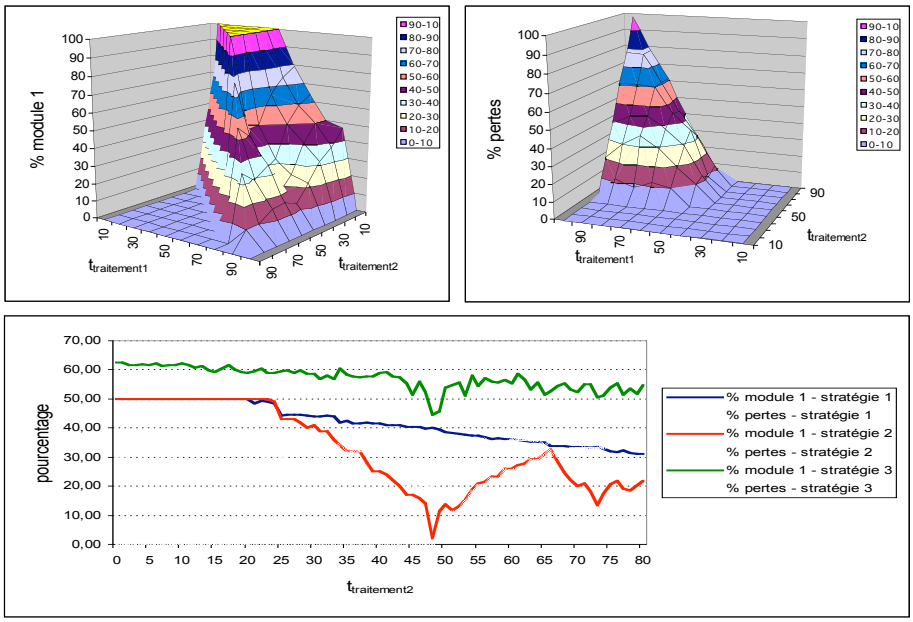

Figure 6. *En haut : pourcentage d'images analysées avec le module 1 (à gauche) et pourcentage d'images perdues (à droite) obtenus avec la 2ème stratégie pour différentes valeurs de temps de traitement pour les modules 1 et 2. En bas : comparaison des 3 stratégies pour* $t_{traitement1}$ *= 80 ms, tolerance = 1,25 et en faisant varier* $t_{traitement2}$.

L'utilisation de la simulation permet par ailleurs d'obtenir une évaluation empirique des performances du système en termes de correction 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 en figure 6. Cela 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.

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. 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 en effectuant un certain nombre d'optimisations pour minimiser la taille de l'automate résultant. Si l'on fait, par exemple, le produit synchronisé des automates modélisant l'agent de traitement reproduit à la figure 4, l'automate résultant comporte 24 états et 96 transitions. Du fait notamment des propriétés des états instantanés, on sait cependant que certaines transitions ne pourront jamais être franchies : lorsque l'on fusionne deux états dont l'un seulement est un état instantané, seules les transitions sortant de l'état instantané sont autorisées, les autres n'ayant aucune chance d'être franchies. Ces dernières peuvent donc être supprimées, ce qui peut, dans certains cas, conduire à des états qui ne sont plus connectés au reste de l'automate et peuvent eux aussi être supprimés. D'autres optimisations sont encore possibles, qui consistent, dans certains cas, à fusionner plusieurs états instantanés successifs. Après optimisation, l'automate résultant ne comporte plus que 8 états et 13 transitions, ainsi que le montre la figure 7.

Chaque agent est alors modélisé par un automate unique que l'on peut traduire sous forme exécutable en plusieurs étapes. Dans un premier temps, l'automate temporisé est transcrit sous forme d'automate à états fini. Les places dans lesquelles il est nécessaire de laisser s'écouler le temps sont supposées correspondre à un traitement. Notre compilateur traduit ceci pour l'agent sous forme d'état dans lequel ce dernier fait une pause (à remplacer par l'appel au module de traitement lorsqu'il sera disponible). Enfin, les signaux de synchronisation entre automates sont associés à des communications entre les agents correspondants.

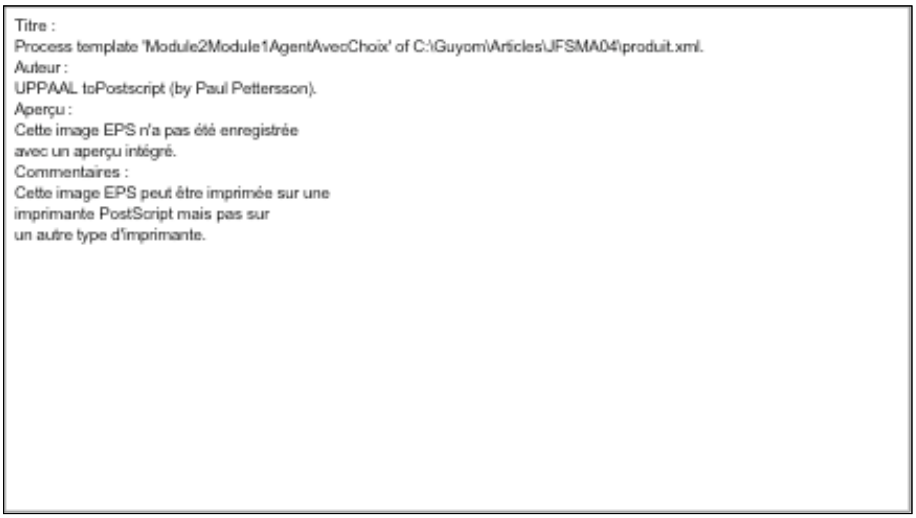

Figure 7. *Automate obtenu en effectuant puis en optimisant le produit synchronisé des trois automates de la figure 4.*

7. Conclusion

Nous avons décrit dans cet article une démarche complète de conception de systèmes multi-agents réactifs, qui intègre les principales étapes du cycle de vie logiciel, depuis les phases de conception jusqu'aux phases de test. Au cœur de cette démarche se situent les automates temporisés, qui constituent un modèle unificateur. Nous avons montré en effet comment ce formalisme, de par sa capacité à modéliser des processus concurrents ayant des propriétés temporelles, pouvait s'adapter à la représentation de systèmes à base d'agents. Nous avons également montré qu'il était possible de modéliser un contrôleur d'agent sous forme modulaire, 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. Enfin, le formalisme peut constituer le support de la génération semi-automatique du squelette d'un premier prototype.

L'objectif est finalement 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. 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

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.

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.

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.

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.

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.

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.