Description, programmation et validation d'interactions par *Coupled Augmented Transition Network* (CATN)

Christian Lemaître† Xavier Prat⋆ Laurent Magnin⋆ Arnaud Dury⋆
cll@lania.mx lmagnin@crim.ca adury@crim.ca

⋆CRIM – Centre de recherche informatique de Montréal
550, rue Sherbrooke Ouest
Montréal H3A 1B9 (Québec) – CANADA

† LANIA – Laboratorio Nacional de Informática Avanzada
Rébsamen 80 A.P. 696 C.P. 91090 Xalapa, Veracruz – MÉXICO

Résumé :
Bien que le concept d'interaction entre agents soit au cœur du paradigme multiagent, il reste encore difficilement manipulable. C'est pour cela que nous introduisons ici non seulement un nouveau modèle de description d'interactions, mais bien un modèle de programmation d'interactions, sous forme de machines à transitions d'états, appelé CATN (*Coupled Augmented Transition Network*). Deux implémentations de ce langage ont déjà été réalisées, à partir des agents FIPA-OS et *Guest*. L'étape suivante de notre travail vise à générer automatiquement à partir de ces descriptions d'interactions des *patterns* servant à valider l'exécution réelle de ces interactions.

Mots-clés : Modèle d'interaction, programmation par interaction, validation, *model-checking*, systèmes multiagents

Abstract:
The concept of interaction is at the heart of the multiagent paradigm, but is still hard to manipulate : few models exists to describe, code and verify a set of interactions. We introduce in this paper not only a new model for the description of interactions, but also the implementation model associated with it, using states transition machines and called CATN for *Coupled Augmented Transition Network*. Two different implementations have been made of this model, using the FIPA-OS and the *Guest* platforms. The next step of our work concerns the automated generation of patterns of behaviour, derived from these CATN interaction description, and allowing a semi-automated validation of the real system, implemented with these tools and concepts.

Keywords: Interaction model, interaction programming, validation, model-checking, multiagents systems

1 Introduction

Bien que les interactions entre agents soient au coeur de la problématique multiagent, elles sont rarement explicitées en tant que telles. En effet, ces interactions sont générées (« émergent ») lors de l'exécution des agents à travers un ensemble d'actions programmées pour chaque agent de façon indépendante. Dès lors, ces interactions ne peuvent être qu'observées, et non pas manipulées par le concepteur de l'application. Certes, il existe des modèles d'interactions entre agents (par exemple AUML [2]), mais ceux-ci disparaissent lors de la phase de programmation, et plus encore lors de l'exécution du code. Ce qui pose de nombreuses difficultés en terme de conception, de mise en œuvre, de maintenance ainsi que validation. Comme exemple, il suffit de penser à l'ajout d'un protocole de compression entre agents : où retrouver le code de tous les agents qui seront potentiellement affectés ? Comment passer en cours d'exécution et en une seule étape d'un mode normal à un mode compressé, et vice-versa ? Quelle procédure utiliser pour vérifier que ce nouveau service est correctement effectué ?

La solution à ce problème consiste à réifier de telles interactions, lesquelles seront décrites indépendamment des agents impliqués, et ce en utilisant un langage exécutable (ou interprétable). C'est ainsi que nous introduisons le formalisme CATN (*Coupled Augmented Transition Network*) qui se présente sous la forme d'une machine à transitions d'états représentant l'exécution en parallèle des différents agents impliqués dans l'interaction, lesquels sont reliés entre eux par des arcs *ad hoc*.

Cette modélisation sous forme de CATN a été implémentée et intégrée à la plate-forme FIPA-OS [9] ainsi qu'au *framework* agent « Guest » [14, 15], lesquels agents peuvent indifféremment s'exécuter et migrer sur les plates-formes Aglets, Corba, Grasshopper, Jade et Voyager.

Les interactions étant dès lors représentées formellement sous la forme de CATN, il est possible d'en extraire des informations utiles à leur validation. Pour cela, nous employons l'approche *Goal* [10, 11] basée sur la détection de *patterns* et d'*anti-patterns*.

2 Le formalisme *Coupled ATN*

2.1 Des ATN aux CATN

Un CATN est une machine à transitions d'états à laquelle on associe un but (ou signification) particulier. Il possède également la caractéristique d'être récursif : un CATN peut être décomposé lui-même en sous-CATN, et ce avec autant de niveaux que nécessaires.

Chacun de ces composants est un CATN à part entière, ayant son propre but. Tous les sous-CATN qui dépendent d'un CATN principal héritent de l'agent et du rôle correspondant. Un CATN peut être appelé par différents CATN en tant que sous-CATN dans des contextes différents et avec des agents-rôles différents.

Les composants d'un CATN sont reliés entre-eux par l'intermédiaire de transitions *ad hoc* appelées « transitions d'interaction ». Parmi celles-ci, nous distinguons les transitions d'interaction non terminales de celles, terminales, correspondant à des actes de langages (entre agents) ou à des actions privées des agents. Cet aspect récursif des CATN permet dès lors une approche de conception descendante, du comportement le plus abstrait d'un groupe d'agents jusqu'à leurs actions les plus concrètes (actions terminales individuelles et communications - au travers des transitions d'interaction -).

Les CATN sont une extension des ATN (*Augmented Transition Network*), lesquels ont été conçus originellement par W. Woods afin de *parser* du langage naturel[1]. Les ATN appartiennent à l'ensemble des grammaires libres de contexte, tout en étant étendus, au sens où ils sont capables de manier des caractéristiques comme le genre et le nombre afin d'assurer les concordances entre les groupes nominaux sujets et les groupes verbaux, ce qui leur donne davantage de puissance.

Les modifications introduites pour passer des ATN aux CATN sont de trois types :
- le formalisme CATN a été conçu afin de supporter des comportements d'agents constitués de toutes les actions que peuvent effectuer ces agents, en particulier les actions privées (SY 94) et les communications (en émission - *i.e.*, actes de langage - ainsi qu'en réception de messages) ;
- l'introduction de nouveaux types de transitions tels que le « OU » et le « AND », ainsi que la prise en compte possible de l'heure locale afin de définir des contraintes temporelles pour chaque transition ;
- la définition de transitions d'interaction correspondant à des communications entre deux CATN dépendants de deux agents différents.

En tant que machines à transitions d'états, les CATN sont particulièrement puissants du fait de leur aspect récursif et de l'existence pour chaque état d'un espace de travail dans lequel sont sauvegardées les variables, les fonctions et les conditions de test ; ces dernières servant à choisir la transition la plus appropriée en cours d'exécution. Ces propriétés proviennent de celles des ATN traditionnels qui se trouvent être aussi puissants que des machines de Turing. Avec l'introduction des transitions d'interactions, nous obtenons une puissance comparable à celles des *Interaction Machines* de Peter Wegner [20].

Au sein des transitions d'interactions nous distinguons différents niveaux d'abstraction. Par exemple, dans le cas du commerce électronique, une transition peut indiquer, au niveau le plus abstrait, que l'agent vendeur veut interagir avec l'agent acheteur à un instant donné ; tandis qu'au niveau le plus bas, l'émission - réception d'un acte de langage concret, échangé afin de réaliser la vente, est également représenté par une transition.

Un autre point important concernant les CATN est le fait que chaque agent peut exécuter de façon concurrente plusieurs CATN en fonctions des tâches qu'il a à réaliser ; tout en permettant à un même CATN d'être exécuté en même temps par plusieurs agents (en fait, il s'agit ici d'instanciations différentes du même CATN).

2.2 Le formalisme CATN

Un CATN représente une tâche que doit exécuter un agent jouant un rôle spécifique. Un agent peut exécuter simultanément plusieurs CATN.

Les éléments qui composent un CATN sont :
- *1 ou plusieurs états* : graphiquement représenté par un cercle, chaque état représente l'exécution partielle d'un plan ou d'une tâche. Quand le CATN est exécuté, un seul de ses états (celui qui est en cours d'exécution) correspond à l'état courant ;
- *0 ou plus transitions*, lesquelles connectent les états, permettant ainsi le passage de l'état courant d'un état donné à un autre. Chaque transition est composée d'un ou de plusieurs arcs en fonction de son type :

– *Transitions simples*. Sont composées d'un seul arc qui doit pouvoir être franchi afin que l'état courant puisse être modifié,
– *Transitions XOR (OU exclusif)*. Sont composées de deux ou plus arcs, dont l'un au moins doit pouvoir être franchi afin que l'état courant puisse être modifié. Tous les arcs d'une transition XOR doivent avoir pour origine le même état tout en ayant pour destination des états différents,
– *Transitions OR (OU)*. Sont composées de deux ou plus arcs, dont l'un au moins doit pouvoir être franchi afin que l'état courant puisse être modifié. Tous les arcs d'une transition OR doivent avoir pour origine le même état tout en ayant pour destination un autre état également unique,
– *Transitions AND (ET)*. Sont composées de deux ou plus arcs, dont tous doivent pouvoir être franchis afin que l'état courant puisse être modifié. Tous les arcs d'une transition AND doivent avoir pour origine le même état tout en ayant pour destination un autre état également unique

Graphiquement représenté par un arc au trait continu connectant deux états, chaque arc appartient à un des types suivants :

– *CATN*. Ce type d'arc représente des actions non-terminales, lesquelles sont spécifiquement dédiées à l'invocation d'un CATN (lequel peut par ailleurs être celui là-même qui effectue l'invocation) ;
– *Actions privées*. Ce type d'arc représente des actions terminales qu'un agent peut exécuter. Ces actions se retrouvent sous la forme de fonctions qui encapsulent une action en particulier (telle qu'imprimer un message, écrire dans un fichier, etc.) ;
– *CATNMS (CATN Message Sender)*. Ce type d'arc représente l'envoi d'un message de la part de l'agent propriétaire du CATN à destination d'un autre CATN détenu par un autre agent. Il s'agit aussi d'une action terminale ;
– *CATMR (CATN Message Receiver)*. Ce type d'arc représente la réception par l'agent propriétaire du CATN d'un message en provenance d'un autre CATN détenu par un autre agent ;
– *UserMR (User Message Receiver)*. Ce type d'arc représente la réception par l'agent exécutant le CATN d'un message en provenance de l'utilisateur ;
– *Arc JUMP*, lequel sera franchi sans qu'aucune action ne soit effectuée.

Les arcs peuvent être liés à des contraintes, lesquelles spécifient les valeurs qu'un ou plusieurs registres doivent avoir lorsque l'arc sera franchi. En d'autres termes, c'est seulement lorsque tous les registres ont comme valeurs celles spécifiées par les contraintes que l'arc peut être franchi.

2.3 Interactions entre CATN

Le formalisme CATN permet la représentation d'interactions entre CATN de différents agents. De telles interactions sont représentées graphiquement par un arc vertical, qui partant d'un arc d'un CATN abouti à un autre arc dépendant d'un autre CATN. Notre formalisme propose deux types d'interactions entre CATN :

– *Interactions non terminales*[1], lesquelles relient deux arcs de type CATN (de deux CATN différents). Pour ce type d'interaction, il n'y a pas d'échange explicite de messages : ils servent à indiquer que lors de l'exécution des deux CATN inter-reliés il y aura, éventuellement, échange de messages entre eux ;
– *Interactions terminales*[2], lesquelles relient un arc de type CATNMessageSender à un arc CATNMessageReceiver (de deux CATN différents).

2.4 Exemple de CATN : « Offre de service »

L'exemple de CATN que nous vous proposons ici (cf. fig. 1) décrit un protocole entre 1 agent demandeur et 0 ou plusieurs agents fournisseurs de service. Il met en œuvre des sous-CATN, l'envoi de message par *broadcast* et l'envoi de message à un agent particulier.

Le comportement attendu du système est le suivant :

1. un message `request` est broadcasté par l'agent exécutant le CATN `applicant.catn` ;
2. les agents exécutant le CATN `serviceProviderListener.catn` calculent alors le coût du service, puis envoient un message `inform` contenant ce coût au demandeur ;
3. le demandeur attend 1 seconde pour recevoir un message, s'il reçoit un message dans cet intervalle il attend une seconde supplémentaire ;

[1] L'arc représentant ce type d'interaction sera hachuré, avec un flèche à une seule extrémité lorsque l'envoi des message ne pourra se faire que dans une sens, avec deux flèches dans le cas contraire

[2] L'arc représentant ce type d'interaction sera plein et unidirectionnel

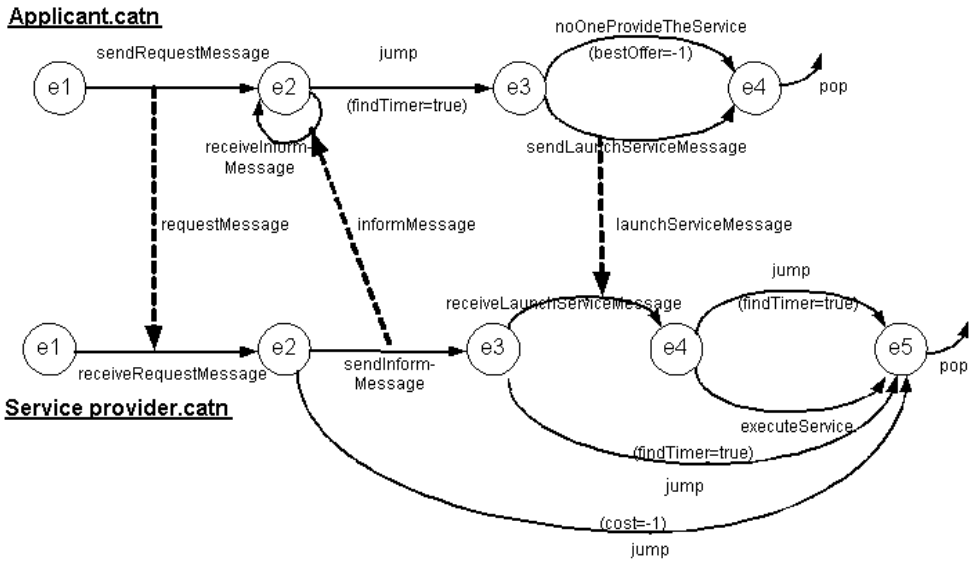

FIG. 1 – Schéma représentant le CATN d'offre de service

4. une fois qu'il a reçu les messages il envoi un message `LaunchService` à l'agent proposant le coût le plus faible ;
5. s'il n'a reçu aucun message au bout d'une seconde il affichera un message *ad hoc* ;
6. l'agent recevant le message `LaunchService` va exécuter ce service ;
7. les autres agents attendront 10 secondes puis afficheront un message indiquant qu'ils n'ont pas été sélectionnés.

2.5 Un formalisme comparable : les Réseaux de Petri

Parmi les formalismes de réseaux de transitions les plus proches des CATN se trouvent les Réseaux de Petri (RdP), dans leurs diverses variantes ; lesquels RdP ont déjà été appliqués à des systèmes multiagents [7, 3, 18, 6]. Les principales différences entre les différents types de RdP et les CATN se situent essentiellement au niveau de leurs représentations graphiques.

La première différence tient au fait que les RdP sont des graphes bipartis comprenant deux types de nœuds (les places et les transitions), alors que les CATN sont des graphes ayant un seul type de nœud (lesquels représentent les états des agents en cours d'exécution d'une tâche). Les RdP permettent de modéliser des systèmes distribués complets ; l'état du système étant représentés par l'ensemble des marques, également appelées jetons, distribuées dans les différentes places du système. Le fait d'avoir dans le cas des RdP deux types de nœuds complique au niveau graphique la compréhension d'un système de moyenne dimension. À l'inverse, les CATN permettent une représentation explicite des actions et des états des différents agents en tout moment.

La deuxième différence est liée au fait que les CATN introduisent deux types d'arcs, les arcs entre états du CATN d'un même agent et les interactions entre CATN d'agents associés à la résolution d'une tâche commune. Dès lors, on obtient une plus grande expressivité graphique, en permettant d'analyser séparément, soit l'exécution des actions de chaque agent en suivant ses CATN respectifs, soit les interactions entre agents.

Parmi les variantes des RdP utilisées pour les SMA on trouve les RdP Colorés, RdPC [3, 18], qui bien qu'ayant la même puissance de description que les RdP, essayent de mieux synthétiser les informations, particulièrement celles des jetons distribuées dans les places du RdPC. Malheureusement cette synthétisation ne se reflète pas dans une plus grande lisibilité des diagrammes des RdPC, ce qui n'aide pas a ce que cette représentation puisse être plus largement utilisée.

Une autre variante est celle des RdP Récursifs [6], RdPR, qui a été appliquée pour modeler la fusion de plans, leur coordination dynamique et

leur exécution. Les RdPR introduisent les notions d'action, de transition et de méthode abstraites qui permettent une définition de RdPR de hauts niveaux chaque fois plus détaillés jusqu'à arriver à des transitions simples. Cette capacité de description de plans ou de processus qui vont de modèles abstraits à des modèles toujours plus plus concrets correspond parfaitement à la démarche naturelle d'utilisation des CATN.

3 CATN pour agents FIPA-OS

Le formalisme CATN a donné lieu à une première implémentation, entièrement réalisée au LANIA, qui utilise les deux premiers composants de la plate-forme FIPA-OS de Nortel Networks[9], MTS et MTP, lesquels fournissent les fonctionnalités qui permettent aux agents d'envoyer et de recevoir des messages.

3.1 Le module CATNAgentToolkit

Ce module est décomposé en deux *packages* : CATNAgent et CATNManager ; chacun consistant en un ensemble de classes et fournissant un ensemble de fonctionnalités reliées qui sont utilisées pour la programmation d'agents basés sur les CATN.

Le composant CATNAgent. Le composant CATNAgent encapsule les fonctionnalités de base d'un agent FIPA-compatible et d'un agent basé sur le formalisme CATN. Les agents qui sont développés en utilisant le module CATNAgentToolkit devront étendre cette classe.

Un agent CATNAgent est composé de :
- *un CATNListener* (lequel est facultatif), qui est le CATN responsable de la réception des messages correspondant à des conversations non entamées par l'agent et à lancer les CATN qui s'occuperont de traiter ces messages. L'ensemble des CATN qui sont lancés par le CATNListener représentent les demandes dont l'agent est capable de s'occuper ainsi que les actions qu'il effectue en s'occupant d'elles ;
- *un ou plusieurs CATN* qui sont activés au début de l'exécution de l'agent. Cet ensemble de CATN représente les actions que l'agent effectue automatiquement.

Le composant CATNManager. Dans le CATNAgentToolkit chaque agent est associé à un composant CATNManager. Les fonctionnalités de celui-ci sont de contrôler l'exécution des CATN auxquels l'agent se trouve associé et de gérer les conversations dans lesquelles intervient cet agent. Dans les sections suivantes il est brièvement expliqué comment il effectue chacune de ces tâches.

Contrôle de l'exécution des CATN Pour chaque CATN qu'un agent est en train d'exécuter (en incluant le CATNListener), le CATNManager maintient un arbre d'exécution spécifique. Pour chaque branche, les noeuds depuis le noeud racine jusqu'à l'avant-dernier niveau correspondent à des actions non terminales alors que les feuilles de l'arbre correspondent à des actions terminales.

Gestion des conversations Tous les messages font partie d'un protocole d'interaction. Pour gérer les conversations de l'agent auxquelles il se trouve associé, le CATNManager fait usage du paramètre Conversation-ID défini dans le langage FIPA ACL[8]. Un identificateur de conversation est créé (de façon unique) pour chaque paire de CATN qui s'envoie des messages pour ensuite être utilisé pendant tout le flux de messages ou durant toute la durée du protocole qu'ils suivent. Pour les fonctions d'envoi et de réception de messages le CATNManager utilise les services de l'objet MTS de FIPA-OS. Pour contrôler le flux de messages, le CATNManager définit et utilise trois tableaux :
- *le tableau de conversations actives*, lequel associe l'identificateur de conversation avec le CATN associé à cette conversation ;
- *le tableau des CATN en attente de message*, lequel associe l'identificateur du CATN attendant le message, le nom de l'action terminale qui s'exécutera en le recevant (laquelle est définie comme une fonction) et le nom du fichier où on trouve la fonction correspondant à cette action terminale ;
- *le tableau de messages reçus dont le CATN récepteur n'a pas encore été activé*, lequel associe l'identificateur du CATN auquel est destiné le message et le message lui-même.

4 CATN pour agents *Guest*

4.1 Les agents *Guest*

Malgré des efforts de normalisation (essentiellement Masif et Fipa), les plates-formes multiagents actuelles ne sont pas conçues pour permettre l'exécution et la migration d'agents iden-

tiques sur chacune d'elles. C'est ce qui a amené le CRIM à proposer et à implémenter un modèle d'agents « universels » [14, 15], appelés *Guest*, capables de s'exécuter et de migrer, sans recompilation, entre plusieurs plates-formes hétérogènes[3].

Nous offrons également aux agents *Guest* des fonctionnalités supplémentaires non offertes par les plates-formes utilisées : *plug-ins* permettant l'ajout et le retrait de code des agents, diverses formes de hiérarchies, une interface graphique unique pour tous les systèmes sous-jacents, etc. C'est dans ce contexte que nous avons intégré les CATN aux agents *Guest* ; à la fois pour en faciliter la programmation, mais également pour aider à la validation des interactions entre ceux-ci (cf. section 5).

4.2 L'implémentation CATN / *Guest*

L'implémentation des agents *Guest* n'impose pas de modèle comportemental particulier. Au contraire, ils ont été conçus afin de permettre une grande diversité dans ce domaine, pour autant que leur programmation soit réalisée en Java. C'est dans ce contexte qu'a été réalisée une première implémentation d'agents *Guest* spécifiques dont le comportement est décrit par un CATN. La seule obligation, pour qu'un tel agent puisse exécuter un CATN est qu'il possède les méthodes correspondant aux noms spécifiés dans le fichier décrivant ce CATN. À noter cependant que cette première version est encore limitée et expérimentale : impossibilité d'offrir des comportements mixtes (CATN et non-CATN) et de remplacer dynamiquement un type comportement par un autre, sans oublier leur mobilité ou leur sauvegarde sur disque qui n'ont pas été pris en compte[4].

Cette réalisation s'est appuyée sur la couche de la communication entre agents fournie par *Guest*. Ceci nous a notamment permis d'intégrer au formalisme CATN deux nouvelles fonctionnalités ; lesquelles sont le *broadcast* des messages et la communication sur canal nommé. Le *broadcast* permet de créer des CATN sans *a priori* sur l'environnement dans lequel se trouvera l'agent, condition indispensable à la réalisation de systèmes multiagents dynamiques. Les canaux de communication nommés, quant à eux, servent surtout à simplifier la réalisation de CATN pour lesquels un message doit être envoyé à un groupe donné d'autres CATN.

En dehors de ces aspects de communication propres à *Guest*, l'implémentation actuelle des CATN est totalement générique. C'est ainsi que le parseur de fichier CATN de même que le moteur d'exécution des CATN pourraient parfaitement être réutilisés tels quels par d'autres plates-formes multiagents basées sur Java.

4.3 Les modules du moteur de CATN

Le parseur construit en mémoire le graphe du CATN à partir d'un fichier XML le décrivant. C'est à ce niveau que sont effectuées les premières vérifications de cohérence : unicité des noms de registre et des noms d'état, conformité des valeurs d'initialisation des registres, existence des méthodes qui sont spécifiées pour les passages d'arc ainsi que la connexité du graphe CATN.

Le manager de CATN a pour rôle de charger, de décharger et d'exécuter un ou plusieurs CATN sur un agent. La principale difficulté rencontrée a été de gérer correctement la remonté des erreurs. En effet nous devons garantir qu'une erreur sur un CATN entraîne le déchargement de ce CATN mais également, en cascade, celui de tous les CATN fils et de tous les CATN ancêtres.

Le manager de message a pour tâche la gestion d'une file de message en attente de traitement et d'un file d'arcs en attente de message. De plus, ce module doit attribuer des identifiants de conversation unique localement à chaque CATN pour permettre un bon aiguillage des messages pour les réponses.

5 Validation d'interactions entre agents

Un des défis du domaine des systèmes multiagents est celui de la validation, rendue complexe par le nombre et l'autonomie des composants en interaction. Notre but est de proposer un modèle de développement de SMA capable de s'interfacer avec un modèle existant et robuste de validation de systèmes distribués. Pour ce faire, nous adaptons le modèle CATN à l'approche GOAL [10, 11] de validation par *patterns*. Nous montrons que l'utilisation des CATN, qui rendent

[3] Actuellement, sont supportées les plates-formes multiagents suivantes : Aglets, Concordia (partiellement), Grasshopper, Jade et Voyager ainsi que Corba tel que fourni par Java 1.4.

[4] À noter qu'une implémentation de méta-CATN pouvant manipuler des CATN de plus bas niveaux est en cours. Ceci afin de permettre leur modification dynamique.

explicites les interactions définies au sein du SMA, permet de manipuler comme une seule entité un processus de communication à plusieurs agents, et donc d'approcher la validation des SMA sous l'angle interactionnel. Nous nous contentons dans cette première phase d'une approche *post-mortem*, basée sur l'analyse de la trace d'un système une fois son exécution terminée. Cette approche, en raison de la complexité des traitements effectués par les *model-checkers*, ne s'applique pas à une validation en temps réel des systèmes.

5.1 Validation *post-mortem*

La validation *post-mortem* se base sur l'étude, à la fin de l'exécution du programme, du fichier de *log* que celui-ci a généré. Ce fichier est construit au fur et à mesure de l'exécution[5], de façon à contenir l'ensemble des données nécessaires à la procédure de validation : la description des messages échangés, la description des événements survenus dans le système et la définition d'un ordre partiel sur ceux-ci. Cette validation est réalisée à l'aide d'un environnement de validation de traces d'exécution, utilisant le moteur ObjectGeode. L'utilisation de cet outil nécessite de produire des *patterns* définissant les propriétés que l'on cherche à valider. Ces *patterns* sont définies sous forme d'observateurs dans le langage GOAL. GOAL est un langage d'automates à états finis, dont certains états sont des états d'acceptation du *pattern* reconnu. Un observateur en GOAL est décrit en terme d'entités (objets et signaux échangés entre les objets). Les communications entre agents sont directement accessibles aux observateurs, et les états internes des objets (variables et états) peuvent l'être en utilisant des sondes particulières reliées à ces objets.

L'intérêt majeur de l'approche CATN dans ce cadre est la possibilité d'exprimer une transformation semi-automatique du formalisme de définition de comportements vers le formalisme GOAL de définition des *patterns* de validation. En effet, connaissant précisément le protocole d'interaction utilisé dans la description du CATN, on peut extraire de celui-ci plusieurs observateurs GOAL correspondant aux différents chemins possibles d'exécution de l'interaction (par exemple : le plus court chemin menant à un noeud donné, le plus long, etc.).

[5]Pour cela nous utilisons les outils de gestion centralisée de trace de *Guest*.

5.2 Modèle de transformation

Le modèle de validation sur lequel nous travaillons demande à ce que les informations suivantes soient disponibles : succès ou échecs associés aux états finaux, des liens de communication de type *broadcast* ainsi que des informations sur l'arité des liens de *broadcast*. Ce dernier besoin provient de la non-disponibilité dans GOAL de la notion de *broadcast*, nous obligeant à décrire chaque *brodcast* d'un agent vers tous les autres comme étant en fait une succession d'échanges point à point entre agents.

À l'aide de ces informations supplémentaires, nous développons un algorithme semi-automatique capable, à partir d'un SMA dont le comportement est défini sous forme de CATN, de produire un ensemble de *patterns* de validation pertinentes pour le système considéré. Le choix des *patterns* effectivement utilisés reviendra dans un premier temps à un utilisateur humain, et pourra par la suite être confié à des heuristiques intelligentes. Une heuristique très simple serait par exemple de privilégier l'expression des *patterns* décrivant les plus courts chemins menant à des nœuds terminaux de l'interaction identifiés comme des succès (afin de s'assurer que ces chemins, les meilleurs possibles, sont le plus souvent empruntés), ainsi que les plus longs chemins menant à des nœuds terminaux de l'interaction identifiés comme des échecs (car cela représente le pire cas).

6 Conclusion

La possibilité de pouvoir non seulement décrire, mais également programmer et manipuler des interactions entre agents ouvre de nombreuses perspectives. À la fois en terme de conception de systèmes multiagents, mais également, comme nous avons pu le voir, par les facultés apportées en terme de validation.

Une étape supplémentaire sur laquelle nous avons commencé à travailler concerne la possibilité offerte aux agents basés sur les CATN de pouvoir observer et manipuler, en cours d'exécution, leurs propres interactions. C'est ainsi que deux agents pourraient décider de remplacer de façon synchrone leurs communications usuelles par des communications cryptées en effectuant une simple mise à jour de leur CATN commun. Notre but ultime étant de permettre à de tels agents de détecter automatiquement des défaillances dans l'exécution de

leurs CATN, pour ensuite pouvoir les modifier afin de contourner les causes de ces échecs. En d'autres termes, de rendre les agents adaptatifs, non seulement individuellement, mais également collectivement à travers leurs interactions.

Remerciements

Le travail initial concernant les CATN a été partiellement financé par le Conseil National de la Science et de la Technologie du Mexique, (Consejo Nacional de Cienca y Tecnología, CONACYT) à travers le projet de recherche numéro 31827A. Le projet *Guest* est pour sa part largement soutenu par le CRIM, sous la forme d'un financement interne de type « patrimoine ». C'est ainsi que durant l'été 2002 Xavier Prat a pu effectuer au CRIM son stage de DESS GLA de l'université Paris VI. Ce projet a également bénéficié de l'aide financière accordée à titre individuelle à Laurent Magnin par le CRSNG (Conseil de recherches en sciences naturelles et en génie du Canada).

Nous tenons également à remercier tout particulièrement Laura Zavala et Araceli Carreño, anciens étudiants au LANIA, pour l'implémentation de CATNAgentToolKit, ainsi que Nourchène Elleuch et Hesham Hallal du CRIM pour l'étude de la validation des CATN à travers l'approche GOAL.

Références

[1] J. Allen, *Natural Language Understanding*, Benjamin/Cummings. 1987

[2] *AUML Website*, http ://www.auml.org/

[3] I. Bakam, F. Kordon, C. Le Page, F. Bousquet, *Formalisation of a Spatialized Multiagent Model Using Coloured Petri Nets for Study of an Hunting Management System*, In J.L. Rash et al. (Eds), FAABS 2000, LNAI 1871, Springer, 2001

[4] I. Bourke, Nortel Network Co, *FIPA-OS Agent Tasks*, http ://fipa-os.sourceforge.net/docs/presentations/FIPAOS_Agent_Tasks.pdf

[5] M. Daconta and A. Saganích, *XML development with Java 2*. Sams Publishing. 2000.

[6] A. El Fallah Seghrouchni, S. Haddad, *A recursive Model for Distributed Planning*, In Mario Tokoro(Ed) Proceedings ICMAS-96, AAAI Press, 1996.

[7] J. Ferber. *multiagent Systems, an introduction to distributed artificial Intelligence*, Addison-Wesley, 1999

[8] *FIPA Website*, http ://www.fipa.org/

[9] *FIPA-OS Website*, http ://www.nortelnetworks.com/fipa-os

[10] H. Hallal, A. Petrenko, A. Ulrich, S. Boroday, *Using SDL Tools to Test Properties of Distributed Systems*, Workshop on Formal Approaches to Testing of Software (FATES) in affiliation with CONCUR 2001 ; Aalborg, Denmark ; BRICS Technical Report NS-01-4, August 2001.

[11] H. Hallal, S. Boroday, A. Ulrich, A. Petrenko, *An Automata-based Approach to Property Testing in Event Traces*, In Proc. IFIP TestCom : Int'l Conference on Testing Communication System, Sophia-Antipolis, France, May 26-29, 2003.

[12] C. Lemaître, C. Excelente. *multiagent Organization Approach*.
In Proceedings of the II Iberoamerican Workshop on DAI-MAS, Toledo, Spain, 1998.

[13] C. Lemaître, C. Excelente. *multiagent Network for Cooperative Work*.
Expert System with Applications : An international Journal. Elsevier Science, 1998.

[14] L. Magnin, E. H. Alikacem, *GenA : Multiplatform Generic Agents* In First International Workshop on Mobile Agents for Telecommunication Applications (MATA'99). Ottawa, Canada, October 4-6, 1999.

[15] L. Magnin, T. V. Pham, A. Dury, N. Besson and A. Thiefaine, *Our Guest Agents are Welcome to Your Agent Platforms* In Proceedings of the ACM Symposium on Applied Computing (SAC 2002), pp. 107-114. Madrid, Spain, March 10-14, 2002.

[16] Nortel Networks Co, *FIPA-OS Developers Guide*, http ://fipa-os.sourceforge.net/docs/Developers_Guide.pdf

[17] D. North, *Institutions, Institutional Change and Economic Performance*, Cambgridge University Press, 1990

[18] R. Scott, Y. Chen, T. Finin, Y. Labrou, Y. Pen, *Modeling Agent Conversations with Colored Petri Nets*, Third Conference on Autonomous Agents (Agents-99), Workshop on Agent Conversation Policies, Seattle, WA (May 1999)

[19] Y. Shoham, *Agent Oriented Programming : an overview of the framework and a summary of recent research*, Knowledge Representation and Reasoning Under Uncertainty pp. 123-9, 1994.

[20] P. Werner. *The Paradigm Shift from Algorithms to Interaction*, CACM,40 :80-91. 1997

[21] W. A. Woods. *Transition Network Grammars for Natural Language Analysis*. Communications of the ACM. Vol 13, No 10,1970

[22] *Extensible Markup Language (XML)*. W3C Website, http ://www.w3.org/xml

[23] L. Zavala, A. Careño, C. Lemaître. *CATNAgent Toolkit una plataforma para el diseño y ejecución de sistemas multiagentes*. In Proceedings of the Mexican International Conference on Computer Science, INEGI, Mexico, 2001.