CRAT – Vers une Programmation Conversationnelle

EMMANUEL CASTRO, JEAN SALLANTIN, STEFANO A. CERRI

LIRMM
151 rue Ada, Montpellier, France
34392 Cedex 5
EURIWARE
12,14 rue du Fort de Saint Cyr
78180 Montigny le Bretonneux

Résumé :
Dans cet article nous montrons que les communications entre des agents peuvent être décrites en termes d'environnements dynamiques d'évaluation. Nous mettons en évidence cette idée à travers un scénario de commerce électronique, puis nous proposons un protocole de communication nommé CRAT qui rend compte des communications au sein d'un couple d'agents en terme d'environnements d'évaluation et de scripts, et qui permet d'introduire la notion d'environnement dynamique dans le domaine des agents. Ensuite nous montrons que l'utilisation d'environnements dynamique permet de résoudre certains problèmes d'accès à l'information de la part des agents. Ayant posé cela, nous faisons le constat que l'utilisation d'environnements dynamiques renforce la nécessité d'annoter sémantiquement les informations échangées par les agents si l'on veut qu'ils puissent faire face à des situations imprévues.

Mots-clés :
Agents, Conversation, Messages, Environnement Dynamique.

Abstract:
In this article we show that the communications between agents can be described in terms of dynamic environments of evaluation. We first make this idea clear through an electronic commerce scenario, and then we propose a communication protocol named CRAT that describes the communications between two agents in terms of evaluation environments and scripts. That allows us to introduce dynamic environments in the domain of agents. After that, we show that the use of dynamic environments helps solving some problems occurring when agents have to access data. Finally we observe that the use of dynamic environment make stronger the need to annotate semantically the information exchanged between agents so that the agents can handle unexpected situations.

Keywords:
Agents, Conversation, Messages, Dynamic Environment.

1Introduction

Depuis son apparition, le Web a profondément modifié la manière de faire de l'informatique. Il permet de mettre en communication des ordinateurs et des agents logiciels qui ne sont pas contrôlés par les mêmes personnes ou les mêmes organisations. Cela apporte des problèmes d'interopérabilité entre ces agents. Nous présentons un problème de commerce électronique dans lequel il est réaliste que ces problèmes arrivent, même si des efforts de standardisation des communications ont été entrepris. Ensuite nous proposons de résoudre en partie ces problèmes en introduisant la notion d'environnement dynamique d'évaluation dans la communication entre les agents au moyen du protocole de communication CRAT.

2Exemple : location d'appartement sur le Web

Un *Étudiant*, qui possède un *Assistant Électronique*, cherche à louer un appartement à Montpellier. Cet *Assistant Électronique* contient le profil utilisateur de l'*Étudiant* et il va aussi lui servir d'intermédiaire avec le Web. Par ailleurs, notre étudiant connaît un agent *Moteur de Recherche* compétent dans le domaine immobilier. Ce dernier connaît, entre autre, un *Agent Immobilier de Montpellier* qui peut proposer des appartements à ses clients. La figure 1 montre les communications établies entre les différents agents. Nous supposerons dans un premier temps, pour plus de simplicité, que les quatre agents (l'Étudiant, l'Assistant Électronique, le Moteur de Recherche et l'Agent Immobilier de Montpellier) partagent le même vocabulaire. Notons que le Moteur de Recherche aurait pu mettre en communication directe l'Agent Immobilier et l'Assistant Électronique, mais nous avons choisi de ne pas le faire dans cet article pour ne pas alourdir notre propos.

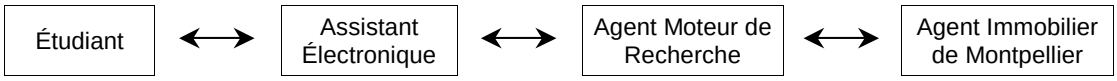
Voyons l'exemple, d'abord à travers les différentes étapes du scénario, puis par les dialogues entre les agents.

2.1Scénario

L'Étudiant demande à son Assistant de l'aider à réaliser une tâche en utilisant son profil utilisateur. Ce dernier répond en lui demandant quelle tâche faire. L'Étudiant lui dit de contacter le service de recherche d'appartement du Moteur de Recherche, alors l'Assistant le contacte et lui demande de trouver un appartement.

Le Moteur de Recherche demande à l'Assistant dans quelle ville il veut cet appartement. L'Assistant trouve dans le profil utilisateur de l'Étudiant que ce dernier veut son appartement à Montpellier, il l'indique donc au Moteur de Recherche. Nous ne nous étendrons pas sur la manière dont il a eu cette information, il a pu l'obtenir dans ses précédentes conversations avec l'Étudiant, ou encore en analysant ses courriers électroniques.

Le Moteur de Recherche contacte l'Agent Immobilier de Montpellier et lui demande de trouver un appartement. Comme on pouvait s'y attendre, l'Agent Immobilier ne demande pas dans quelle ville devra être l'appartement, car toutes les requêtes qui lui sont faites concernent des biens immobiliers à Montpellier, d'autant plus que le terme Montpellier apparaît dans son nom. Il cherchera seulement à savoir le prix que le futur locataire voudra bien y mettre. Cette question est posée au Moteur de Recherche, qui ne sait pas. Ce dernier la pose à l'Assistant, qui ne sait pas non plus, enfin l'Assistant la pose à l'Étudiant, qui répond qu'il veut bien y mettre 260 € par mois. La réponse est retransmise jusqu'à l'Agent Immobilier. Ensuite, l'Étudiant donne, de sa propre initiative, une information qui n'avait pas été demandée. Il dit à tout hasard, à son assistant, qu'il veut bien être en sous-location.



Étudiant	Assistant	Agent Moteur de Recherche	Agent Immobilier de Montpellier
Étudiant – Bonjour, veuillez m'aider à faire une tâche en utilisant mon profil utilisateur			
Assistant – Que faut-il faire ?			
É – Il faut trouver un appartement avec l'Agent Moteur de Recherche Yahoo	**Assistant** – Bonjour, veuillez me trouver un appartement		
	Agent Moteur de Recherche – Savez-vous dans quelle ville ?		
	A – À Montpellier		
			Agent Moteur de Recherche – Ag. Im. de Montpellier, je recherche un appartement.
			Agent Immobilier – À quel prix voulez-vous cela ?
A – À quel prix voulez-vous cela ?	M – À quel prix voulez-vous cela ?		
É – 260	A – 260		
É – Notez que j'accepte d'être en sous-location			M – 260
A – Cela concerne-t-il un étudiant ?	M – Cela concerne-t-il un étudiant ?		I – Cela concerne-t-il un étudiant ?
É – Oui, c'est pour un étudiant	A – Oui, c'est pour un étudiant		
			M – Oui, c'est pour un étudiant
A – Je vous propose l'appartement *XYZ*	M – Je vous propose l'appartement *XYZ*		I – Je vous propose l'appartement *XYZ*

Figure 1 – Dialogue entre les agents

Ce qui n'était pas prévu, ni par le Moteur de Recherche, ni par l'Assistant, ni même par l'Étudiant, c'était que l'Agent Immobilier allait demander si le locataire de l'appartement allait être un étudiant ou pas, car il a un appartement pas cher à proposer, mais il est réservé aux seuls étudiants. L'Étudiant répond que oui, et la réponse suit le même chemin que pour le prix. Finalement, l'Agent Immobilier répond à l'agent Moteur de Recherche, qui transmet l'information jusqu'à l'Étudiant. – Fin du scénario –.

2.2Dialogues

La figure 1 montre le scénario du point de vue des conversations en faisant voir le contenu des dialogues ayant lieu entre chacun des agents. Les dialogues sont exprimés en langue naturelle. Dans la section suivante, nous exposerons nos propositions de résolution des problèmes d'interopérabilité que pose l'échange d'informations avec des agents dont on ne connaît pas grand-chose *a priori*, par exemple quand l'Agent Immobilier pose la question au sujet du statut du locataire, étudiant ou non.

3Accès imprévus à l'information et environnement dynamique

Dans le scénario présenté dans la partie précédente, nous avons soulevé le problème de l'obtention d'une information – provoqués par la question *« Cela concerne-t-il un étudiant ? »* – auprès d'un agent – l'Agent Moteur de Recherche – alors que celui-ci n'avait pas prévu qu'on lui demande cette information. Pour résoudre ce problème, nous proposons de réutiliser un outil bien connu en programmation, mais peu utilisé de nos jours : les variables dynamiques et les environnements dynamiques [Quei94, §2.5][Dami97]. Ces outils ont été développés dans un cadre de programmation classique

(processus d'exécution unique), mais pas du tout dans un cadre de programmation par agents (plusieurs processus d'exécution). Aussi, nous proposons le cadre CRAT, un cadre de communications entre agents permettant aux agents de bénéficier des environnements dynamiques.

Avant d'aller plus loin, commençons par rappeler brièvement ce qu'est un environnement en général, et un environnement dynamique en particulier.

3.1Environnement

En programmation classique, un environnement est un objet qui est utilisé pour évaluer des expressions, par exemple des expressions en langages LISP ou SCHEME. Il fournit les valeurs associées aux noms des variables de l'expression que l'on veut évaluer [Quei94, §1] [Abel96, §3.2]. Techniquement, un environnement est un empilement de blocs (en anglais *frames*) qui contiennent les liaisons entre les noms des variables et leurs valeurs. L'ajout d'un bloc permet d'étendre un environnement, c.-à-d. d'y définir de nouvelles variables. Ensuite, l'accès à la valeur d'une variable se fait en parcourrant la pile des blocs à la recherche de celui qui contient une liaison concernant cette variable.

1.1Environnement lexical

On distingue deux sortes d'environnements (lexical ou dynamique) suivant la manière dont on accède à ceux-ci. Sans entrer dans les détails, disons que dans un environnement lexical, une fonction n'a accès qu'aux variables qu'elle définit elle-même plus celles déjà présentes dans l'environnement dans lequel la fonction a été construite. L'utilisation de ce type d'environnements a une propriété très intéressante : il permet par un simple examen du code source de savoir quelles sont les variables qui sont définies, ce qui simplifie énormément la vie du programmeur. C'est probablement ce qui explique que l'environnement lexical est celui qui a été choisi dans les grands langages de programmation modernes (SCHEME, JAVA, C++).

Il y cependant a un inconvénient à cela : lorsqu'une fonction A appelle une fonction B, B n'a accès qu'aux informations que A lui a explicitement fournies à travers les paramètres d'appel de B. Dans notre scénario, si nos agents avaient un tel comportement, cela les empêcherait d'avoir accès aux informations dont ils ont besoin, car au moment où on leur demande un service (c.-à-d. qu'on leur demande de faire quelque chose), aucune information ne leur est fournie (cf. fig. 1). Ces informations ne sont fournies que plus tard, dans la suite du dialogue. Nous pourrions évidemment prévoir une liste d'informations à fournir lors d'une demande de service, comme cela est le cas pour les appels de fonctions, avec la liste des paramètres de la fonction, mais alors le problème de l'accès non prévu à une information *(« Cela concerne-t-il un étudiant ? »*) ne pourrait être résolu.

3.2Environnement dynamique

Dans un environnement dynamique, une fonction a accès à l'ensemble des variables définies, non plus lors de la construction de la fonction, mais lors de son appel. Ainsi, une fonction représentant l'Agent Immobilier, si elle était appelée par la fonction représentant l'Agent Moteur de Recherche, aurait accès à toutes les informations que possède ce dernier. C'est bien le comportement que nous décrivons dans notre scénario.

Il nous reste maintenant à décrire le cadre CRAT qui va nous permettre d'apporter aux agents l'équivalent des environnements dynamiques de la programmation traditionnelle.

4CRAT – Actes de langage de base pour la communication entre agents

Le cadre CRAT est composé de deux niveaux : d'une part, un niveau permettant de modéliser la communication entre des paires d'agents, ce qui nous permet d'introduire la notion d'environnement[1] d'évaluation dans la programmation avec des agents ; d'autre part, un niveau permettant d'enchaîner des paires d'agents, et dont nous verrons qu'il fournit aux agents des mécanismes analogues à ceux des environnements dynamiques.

Nous présenterons tout d'abord notre manière de modéliser les échanges entre tous types de programmes avec 4 types de messages directement inspirés du processus d'évaluation des expressions SCHEME. Dans notre approche, quand deux agents communiquent, ou plus généralement quand deux morceaux de programmes communiquent, nous cherchons à modéliser leur relation comme une relation Agent Environnement–Agent Script (c.f. §) dans laquelle les deux agents échangent des messages de type CRAT (CALL, REPLY, ASK, TELL) (c.f. §). Ces messages peuvent être explicitement transmis par des chaînes de bits à travers un réseau comme dans une simple relation client-serveur, ou implicitement comme dans une relation fonction appelante–fonction appelée dans laquelle l'envoi du message se fait par le transfert du contrôle d'exécution du programme entre la fonction appelante et la fonction appelée.

Ensuite, nous montrerons comment l'enchaînement de plusieurs couples Agent Environnement–Agent Script présente des caractéristiques communes avec les Environnements Dynamiques (c.f. §). Pour finir, nous illustrerons CRAT sur le scénario présenté au §.

4.1Rôles environnement et script

Dans le paragraphe précédent, nous avons évoqué deux agents, l'Agent Environnement et l'Agent Script. En fait, il s'agit d'abus de langage pour désigner, d'une part, l'agent ayant le rôle Environnement, et d'autre part, l'agent ayant le rôle Script, chaque agent pouvant avoir les deux rôles à la fois.

Lors d'un dialogue entre deux agents, l'Agent Environnement fournit des valeurs à partir de descriptions d'informations. L'Agent Script, lui, exécute une suite d'instructions – un script, d'où son nom. Un agent prend le rôle d'Agent Environnement quand il demande un Service à un autre agent. Ce dernier prend alors le rôle d'Agent Script et l'Agent Environnement s'engage à lui fournir les informations nécessaires à la réalisation du Service. On peut comparer cela au mécanisme d'application d'une fonction dans lequel on passe des paramètres à cette fonction pour qu'elle puisse travailler avec. Si on regarde de plus près les mécanismes d'évaluation d'expression LISP dont nous avons parlé au §, on constate que l'application d'une fonction commence aussi

[1] Dans les systèmes multi-agents, le terme environnement désigne principalement l'ensemble des choses qui sont « autour » de l'agent : son « terrain de jeux » ainsi que les autres agents qui s'y trouvent et avec qui il communique. Ce n'est évidemment pas le sens que nous utilisons ici. Cela dit, dans notre approche, un tel « terrain de jeux » serait accessible à travers un environnement d'évaluation (agent environnement du §).

par créer un environnement contenant les paramètres à passer à la fonction [Abel96, §3.2.1], et que c'est seulement ensuite que le corps de la fonction (le Script) est évalué dans cet environnement.

Quand un Agent Script *B* exécute les instructions lui permettant de réaliser le Service que lui a demandé un Agent Environnement *A*, il peut très bien être amené à demander à un Agent *C* un autre Service. Dans ce cas, en plus du rôle d'Agent Script qu'il avait déjà envers *A*, il prend un rôle d'Agent Environnement envers *C*. Illustrons cela avec l'exemple de la location d'appartement (c.f. fig. 2) : l'Étudiant est Agent Environnement vis-à-vis de son Assistant Électronique car le premier fournit au second les informations qu'il demande (*prix de l'appartement*, *statut du locataire (étudiant ?)*). En retour, l'Assistant Électronique est Agent Script vis-à-vis de l'Étudiant, mais il est aussi Agent Environnement vis-à-vis de l'Agent Moteur de Recherche.

Voyons maintenant en détails les types de messages échangés entre un Agent Environnement et un Agent Script.

4.2Messages CRAT

Pour gérer la communication entre un Agent Environnement et un Agent Script, nous proposons 2 couples de types de messages : CALL – REPLY et ASK – TELL, gérés de la manière suivante :

- Ouverture et fermeture de la communication :
  - **CALL**(*service_description*) : l'Agent Environnement demande à l'Agent Script de lui fournir le Service décrit par *service_description*. L'Assistant Électronique s'engage à fournir les informations nécessaires à la réalisation du Service au moyen de messages TELL.
  - **REPLY**(*reply*) : l'Agent Script indique qu'il a fini son traitement, avec ou sans succès, et renvoi son résultat (*reply*). Cela ferme aussi la communication.
- Échange d'informations :
  - **ASK**(*data_request*) : l'Agent Script demande à l'Agent Environnement une information nécessaire à l'exécution de son script. L'information demandée est décrite par la Requête de Donnée *data_request*. L'Agent Script a le droit de suspendre son exécution jusqu'à obtention d'un message TELL satisfaisant à sa demande, mais ce n'est pas une obligation. Cela incite fortement l'Agent Environnement à répondre à ses demandes, s'il veut que le Service aboutisse.
  - **TELL**(*data_description*, *value*) : l'Agent Environnement envoie une information à l'Agent Script, soit motivée par un message ASK de l'Agent Script, soit de sa propre initiative. L'information est représentée par une Description de Donnée (*data_description*) et par sa valeur (*value*).

Data_request et *data_description* servent à annoter et à décrire les donnés que l'on recherche et celles que l'on propose. Nous les appelons des Annotations. Ce terme sera utilisé quand il sera inutile de distinguer Requête et Description. Nous reviendrons dessus plus en détails au §, nous reviendrons aussi sur la manière de calculer si un TELL (avec sa Description de Donnée) satisfait à la demande formulée par un ASK (avec sa Requête de Donnée) (c.f. §).

Pour récapituler, nous avons 4 types messages :

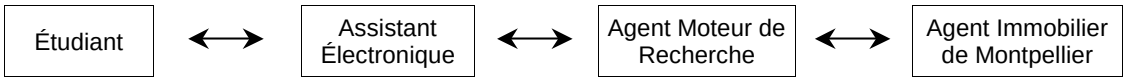
- 2 types de messages émis par l'Agent Environnement : CALL et TELL
- 2 types de messages émis par l'Agent Script : ASK et REPLY

4.3Chaînes d'Agent et Environnement Dynamique

L'utilisation des messages CRAT impose quelques contraintes dans le comportement des Agents :

- L'Agent Environnement envoie un message CALL, ce qui ouvre un canal de communication sur lequel il écoute les messages ASK de l'Agent Script jusqu'à ce qu'il reçoive un message REPLY. Dans le même temps, il émet ses messages TELL, correspondant ou non à des messages ASK.
- L'Agent Script est en permanence en attente de messages CALL. Quand il en reçoit un, il se met aussi en attente des messages TELL correspondant aux CALL. Dans le même temps il envoie ses messages ASK. Quand il a fini son traitement, il envoie le résultat et ferme le canal de communication avec un message REPLY.

Cependant, cela ne nous dit pas comment l'Agent Environnement procède pour obtenir les valeurs qu'il envoie dans ses messages TELL. Deux possibilités s'offrent à lui : soit il les calcule lui-même à partir des informations qu'il possède, soit il les demande à l'extérieur. Dans se cas, l'Agent Environnement s'inscrira dans une autre relation Agent-Environnement–Agent-Script dans laquelle il prendra le rôle de Script. Pour toute information qui lui sera demandée il peut soit chercher la calculer lui-même (le « calcul » le plus simple étant de lire directement sa mémoire interne), soit la demander à son Agent Environnement.



É → CALL(**utilise-profil-utilisateur**)		
A ← ASK(*action*)		
É → TELL(*action*, **moteur-trouve-appart**)	A → CALL(**moteur-trouve-appart**)	
	M ← ASK(*ville*)	
	A → TELL(*ville*, Montpellier)	M → CALL(**trouve-à-montpellier**)
A ← ASK(*prix*)	M ← ASK(*prix*)	I ← ASK(*prix*)
É → TELL(*prix*, 260)		
É → TELL(accepte-sous-location, oui)	A → TELL(*prix*, 260)	
	M ← ASK(*étudiant*)	M → TELL(*prix*, 260)
A ← ASK(*étudiant*)		I ← ASK(*étudiant*)
É → TELL(*étudiant*, oui)	A → TELL(*étudiant*, oui)	
	M ← REPLY(appartement XYZ)	M → TELL(*étudiant*, oui)
A ← REPLY(appartement XYZ)		I ← REPLY(appartement XYZ)

Figure 2 – Messages CALL, REPLY, ASK, TELL entre les agents

Si ce même dispositif se répète sur plusieurs niveaux, comme c'est le cas dans notre scénario, on peut observer que la mémoire interne de chaque Agent Environnement fait office de *bloc*. Ces blocs s'empilent lors de « l'appel » de nouveaux Agents Script et de l'établissement de relations Environnement–Script. Pour l'Agent Script qui se trouve au bout de la chaîne, l'environnement d'évaluation dans lequel il se trouve ne contient pas seulement le bloc de son Agent Environnement, il contient aussi l'ensemble des blocs présents de la chaîne d'agents, formant ainsi l'équivalent, dans le domaine des agents, d'un Environnement Dynamique (cf. fig. 3).

4.4Illustration sur le scénario – Dialogues

La figure 2 montre le scénario exposé au §, en terme de messages CRAT (CALL, REPLY, ASK, TELL). Vous pouvez les comparer aux dialogues de la figure 1. Voici le dictionnaire des symboles qui y sont utilisés :

- Descripteurs de Service (écrits en gras) :
  - o**utilise-profil-utilisateur** : designe le service de l'Assistant Électronique qui propose de faire faire une tâche à un autre agent en mettant à sa disposition le profil utilisateur de l'Étudiant.
  - o**moteur-trouve-appart** : désigne le service de l'Agent Moteur de Recherche qui propose de servir d'intermédiaire avec des agents immobiliers.
  - o**trouve-à-montpellier** : désigne le service de l'Agent Immobilier de Montpellier qui propose des appartements.

  Tous ces services renvoient la description d'un appartement, dans notre scénario c'est *appartement XYZ*.
- Descriptions et Requêtes de Données (écrits en italique) :

  Dans un soucis de simplicité, toutes les annotations de notre exemple (les Requêtes et les Descriptions de Données) sont représentées dans le même langage. Elles sont encodés avec des identificateurs semblables à ceux des langages de programmation, ainsi un TELL satisfait un ASK s'ils contiennent tous les deux la même annotation. Ainsi: *TELL(prix,*260*)* satisfait à la demande formulée par *ASK(prix)*.
  - *action :* décrit un Descripteur de Service. Le Service en question est n'importe quel service pour lequel l'utilisateur de l'Assistant Électronique (c.-à-d. l'Étudiant) veut que son profil utilisateur soit utilisé. Dans l'exemple, l'Étudiant choisit comme action *moteur-trouve-appart*.
  - *prix* : décrit un nombre. Ce nombre doit être un prix en Euros.
  - *accepte-sous-location* : décrit la signification d'une valeur booléenne (oui/non). *Oui* indique l'acceptation d'une sous-location. *Non* indique le contraire.
  - *étudiant* : décrit la signitication d'un booléen. *Oui* indique qu'on parle d'un étudiant, *Non* indique le contraire.
  - *ville* : décrit un nom. Ce nom doit être le nom d'une ville.

On pourra remarquer que le fait d'indiquer quelque chose qui n'a pas été demandé (*TELL(accepte-sous-location,* oui)) se fait avec le même type de messages que les réponses à des questions Ask. Pour le moment, nous n'étudierons pas comment traîter [illegible]és, car des problèmes se posent si un agent tr[illegible]sollicités aux autres [illegible] Age[illegible]finir par être subme[illegible]n-soll[illegible]

4.5Illus[illegible]nviro[illegible]

L[illegible]s form[illegible]e chacun des agents [illegible]lescr[illegible]iée la valeur propos[illegible]ée par l'agent. Sur la gauche d[illegible]ement auquel a accès l[illegible]tant d[illegible]

5Actes d[illegible] mixt[illegible]

5.1Actes

Les actes de langage [Sear69] [McCa92] cherchent à donner un statut aux informations échanges (*messe*) lors d'une

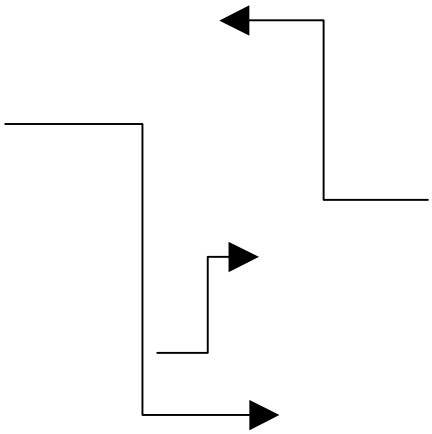


Figure 3 – Environnement formé par les autres agents au profit de l'Agent Immobilier

communication. L'idée a été portée en informatique dans le domaine des agents et a abouti aux actes de communication des langages KQML [Fini94] et ACL[FIPA00]. KQML et ACL proposent un grand catalogue d'actes de communication. CRAT a pour but de proposer un noyau d'actes de communication encore plus restreint que celui proposé par STROBE [Cerr99]. Contrairement à KQML où le comportement des agents est défini par des automates, CRAT laisse le comportement des agents beaucoup plus libre (notamment pour l'agent script). Des communications aussi complexes que celles de KQML peuvent être rendues en composant plusieurs communications CRAT au sein d'un même couple d'agents.

5.2Initiative mixte

Cependant, nous pensons que malgré sa simplicité, CRAT peut rendre compte de communications complexes avec une part d'asynchronisme : en

effet, des messages TELL peuvent intervenir à n'importe quel moment, sans sollicitation préalable par un message ASK. Lorsque nous avons conçu CRAT, nous avons eu le souci de pouvoir aussi modéliser l'initiative mixte introduite par la présence d'utilisateurs [Mara01a] [Mara01b]. Notre parti était de suivre les idées déjà introduites dans le modèle STROBE [Cerr99] et dans son évolution C+C [Cerr00], qui proposent de représenter les agents au moyen d'interpréteurs Scheme et de leur boucle REPL (*read-eval-print-listen*). En plus de chercher à modéliser les dialogues entre les différents agents de notre scénario, nous avons nous avons cherché le cadre de communication le plus simple qu'il soit possible de faire en se basant sur la manière dont les programmes informatiques sont évalués (interprétés / exécutés).

5.3STROBE

Dans ce contexte, il nous semble intéressant de montrer en quoi CRAT se distingue de son inspirateur STROBE. Le modèle STROBE propose 6 types d'actions de base, fondées sur SCHEME, pour représenter les communications :

- *assertion – acknowledge* : l'*assertion* sert à envoyer une nouvelle information en tentant de modifier l'environnement de l'interlocuteur, sans sollicitation particulière de cet interlocuteur. L'acquittement (*acknowledge*) sert à indiquer si l'action à réussi ou pas.
- *request – answer* : la requête (*request*) demande une information en tentant d'accéder à une variable de l'environnement de l'interlocuteur. La réponse (*answer*) envoie une information, comme le fait *assertion*, mais uniquement sur sollicitation.
- *order – executed* : l'ordre (*order*) demande l'application d'une fonction. L'action *exécuté* (*executed*) renvoi la valeur résultant de cette exécution.

Contrairement à STROBE qui met en œuvre l'environnement de chacun des deux agents en communication, nous ne considérons que l'environnement de l'Agent Environnement, et ce seulement en lecture.

CALL et REPLY sont semblables à *order* et *executed*. Nous ne définissons pas d'action pour modifier les environnements, mais, comme nous l'avons vu plus haut, nous avons un message commun pour les envois d'informations sollicitées (*answer*) et non-sollicitées (*assertion*) : c'est TELL. Ne faisant pas d'affectation, nous n'avons pas besoin d'en recevoir l'acquittement, nous n'avons donc aucun message semblable à *acknowledge*. Il reste ASK qui est semblable à *request*. Enfin, à la différence des messages *assertion* et *request* qui travaillent explicitement avec des noms de variables, ASK et TELL travaillent avec des annotations, qui sont une extension de la notion de variable.

6Interopérabilité, annotation et ambiguïté

Pour aider à rendre interopérables des programmes, nous avons besoin de descriptions sémantiques des données qu'ils échangent, d'autant plus lorsque certains de ces programmes ont des comportements non prévus par les autres, comme c'est le cas avec l'Agent Immobilier lorsqu'il émet le message ASK(étudiant). Les annotations (requête et description de données) servent justement à porter cette information sémantique. Au §, nous présentions ces annotations comme une extension de la notion de variable. Nous ajoutons que c'est aussi une notion proche de la notion de type. Nous développons dans cette section ces deux idées un peu iconoclastes.

6.1Annotation vs. variables

Pour comparer les annotations aux variables, il faut d'abord donner une définition de ce qu'est une variable en programmation. Les variables sont ce qui permet d'utiliser des noms pour faire référence à des objets informatiques [Abel96, §1.1.2]. C'est bien ce que nous avons fait dans notre exemple au § avec les annotations. Elles y ont permis de faire référence à l'Agent Moteur de Recherche (*moteur-trouve-appart*), au prix proposé (*prix*), *et cætera*. Voyons maintenant en quoi les annotations sont aussi des types.

6.2Annotation vs. type

Les types sont des informations utilisables par les machines, décrivant la structure des données et les opérations qu'il est possible de faire dessus. Ils permettent de s'abstraire de la manière dont sont rangés les octets représentant des objets informatiques. Dans les langages de programmation on peut observer deux tendances, non incompatibles, sur l'utilisation des types. La première, le typage dynamique, est la pratique qui consiste à associer le typage aux valeurs (Lisp, Smalltalk) ; la seconde, le typage statique, consiste à associer le typage aux variables (ML, C). Cela est à rapprocher du découpage fait dans les annotations entre Descriptions de Données et Requêtes de Données.

6.3Annotation, requête / description

Dans le typage dynamique, chaque valeur se voit adjoindre un type [Adab89] ce qui forme un couple (*Type, Valeur*). Ce couple n'est pas sans rappeler les valeurs annotées transmises par les messages TELL (*Description de Donnée*, *Valeur*). Contrairement au type, qui décrit la structure d'une donnée, la description de donnée cherche seulement à indiquer l'usage pour lequel cette donnée est faite.

Dans le typage statique, les types sont associés aux variables. Ils ont deux utilisations. Ils sont d'abord utilisés par les compilateurs pour vérifier la cohérences des programmes et produire le meilleur code compilé possible. La deuxième est la description sémantique des informations que l'on met dans les variables, et surtout dans les paramètres des fonctions. Actuellement, elle ne sert qu'au programmeur humain pour choisir les données qu'il va utiliser lorsqu'il va écrire un appel de fonction. De la même façon, les requêtes de données des messages ASK doivent permettre à un agent comme l'Agent Moteur de Recherche de choisir les données à transmettre à l'Agent Immobilier de Montpellier quand celui ci veut savoir si l'appartement qu'on lui demande est destiné à un étudiant ou pas ?

6.4Contenu des annotations

Au-delà des simples chaînes de caractères, il est possible d'utiliser des structures beaucoup plus riches pour encoder les annotations. On peut envisager d'utiliser des listes d'identificateurs, des formules de logique propositionnelle, des formules de logique des prédicats, et bien évidemment des ontologies. Quand on parle d'annotation et d'ontologie, on pense bien évidemment à l'annotation des pages Web, qui sont des données comme les autres, et pour lesquelles des systèmes d'annotation sont développés [Hefl00][Hill00]

6.5Ambiguïté

Avec des annotations très riches, il n'est plus possible de définir strictement la correspondance entre une requête de donnée et une description de donnée. On cherchera plutôt la valeur annotée dont la description est la plus proche de la requête. Mais souvent se posera le problème de choisir entre deux valeurs annotées équidistantes par rapport à la requête. Par exemple, un agent cherchant un véhicule pour transporter des personnes pourra se voir proposer d'abord un avion, puis un autobus. Dans ce cas, il doit pouvoir ne pas s'arrêter obligatoirement à la première réponse valide, et doit pouvoir faire un choix entre deux réponses, quitte à demander l'aide d'un autre agent, éventuellement humain. Là, il devient nécessaire d'avoir des stratégies de détection des ambiguïtés [Cast00], et de résolution de ces ambiguïtés. La première de ces stratégies de résolutions d'ambiguïtés est l'interaction avec les utilisateurs intéressés à ce que les agents travaillent correctement. Ces utilisateurs pourront aussi aider à construire, en contexte, les correspondances entre des requêtes et des descriptions de données de deux agents ne partageant pas le même langage de description. Cette souplesse permettra de faciliter l'interopérabilité entre deux agents, lorsque le comportement de l'un d'entre eux n'est pas complètement prévu par l'autre. Cela est fait en relâchant la nécessité d'un accord parfait entre les requêtes et les descriptions. La nécessité de gérer des informations ambiguës à l'intérieur des programmes en est le prix à payer.

7Conclusion

Nous avons montré que les communications entre des agents pouvaient être décrites en termes d'environnements dynamiques d'évaluation. Le scénario de commerce électronique que nous avons présenté nous a permis de mettre ces idées en évidence. Nous avons proposé un protocole de communication nommé CRAT rendant compte des communications au sein d'un couple d'agents en terme d'environnements d'évaluation et de scripts. Ce protocole est basé sur quatre types de messages matérialisant les échanges d'informations se produisant lors de l'utilisation d'environnements dynamiques. Il nous a ainsi permis d'introduire les environnements dynamiques dans le domaine des agents. Ce protocole permet en outre de prendre en compte un phénomène comme l'initiative mixte, indispensable lorsqu'un utilisateur humain doit intervenir dans un système. Ensuite nous avons montré que dans les cas où des agents avaient besoin d'obtenir des informations que leurs interlocuteurs directs n'avaient pas prévu de fournir, le mécanisme d'environnement dynamique leur permettait tout de même de les obtenir. Ayant posé cela, nous avons constaté la nécessité d'annoter sémantiquement les informations échangées par les agents pour qu'ils puissent trouver ces informations non prévues.

8Bibliographie

Abadi, M. et al. (1989) *Dynamic Typing in a Statically-Typed Language*. Proceedings of Sixteenth Annual Symposium on Principles of Programming Languages, p. 213-227.

Abelson, H. et al (1996) *Structure and Interpretation of Computer Programs, second edition*. Cambridge : The MIT Press.

Castro, E. (2000) Misunderstanding Detection using a Constraint Based Mediator. *Proceedings of ALCAA 2000*, Biarritz. http://www.lirmm.fr/~castro/Paper/ALCAA2000

Cerri, S.A. (1999) Shifting the focus from control to communication: the STReams OBjects Environments model of communicating agents. *Collaboration between Human and Artificial Societies, Coordination and Agent-Based Distributed Computing*, J.A. Padget Ed., Lect. Notes in Artificial Intelligence vol. 1624, pp. 71-101, Springer-Verlag.

Cerri, S.A. et al. (2000) Steps towards C+C: a Language for Interactions*. Proceedings of Artificial Intelligence: Methodology, Systems, Application, AIMSA 2000*, S.A. Cerri and D. Dochev Eds., Lecture Notes in Artificial Intelligence vol 1904, pp. 33-46, Springer-Verlag.

Dami, L. (1998) A Lambda-Calculus for Dynamic Binding. *Theoretical Computer Science*, 192 (2) pp. 201-231.

Finin, T. et al. (1994) KQML: An Information and Knowledge Exchange Protocol*. Knowledge Building and Knowledge Sharing*, Kazuhiro Fuchi and Toshio Yokoi (Ed.), Ohmsha and IOS Press.

FIPA (2002) FIPA Communicative Act Library Specification. FIPA Agent Management Specification, XC00037.

Heflin J. et al. (2000) Semantic Interoperability on the Web*. Proceedings of Extreme Markup Languages 2000*. Graphic Communications Association, 2000. pp. 111-120

Hillmann, D. (2000) *Using Dublin Core, Dublin Core Metadata Initiative working draft*. http://dublincore.org/documents/usageguide/

Maraschi, D. et al. (2002) Relation entre les technologies de l'apprentissage humain et les agents. *Revue cognitique, n° spécial agents logiciels–coopération–apprentissage–activité humaine~~, pp. ???~~*

Maraschi, D. et al. (2001) *A Conversational, Constructive view of Web knowledge: the Symbol Level*. Rap. de Recherche LIRMM.

McCarthy, J. (1992) *Elephant 2000: A Programming Language Based on Speech Acts*. http://www-formal.stanford.edu/jmc/elephant.pdf

Queinnec C. (1994) *Les langages Lisp*. Paris : InterÉditions.

Searle, J. (1969) *Speech Acts*. Cambridge: Cambridge University Press.