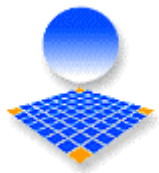
Le modèle STROBE et ses évolutions : Interprétation des interactions

GT Modèles Formels pour l'Interaction
du GDR I3

5 décembre 2003

Clément Jonquet

LIRMM
Laboratoire
d'Informatique
de Robotique
et de Microélectronique
de Montpellier



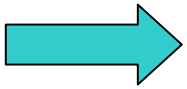
Où se situe notre modèle ?

- IAD/SMA : C'est un modèle de représentation et de communication agent
- IHM : C'est un modèle centré utilisateur qui tente de faire abstraction du type des agents (AH/AA)
  - EIAH : Apprentissage => Interaction
  - LA : STROBE est intimement lié à Scheme
- Globalement on parle de communication agent et d'interprétation des langages

Au sujet de la communication agent

- La simple mise en commun de plusieurs agents ne suffit pas à former un SMA
- La communication permet la coopération et la coordination entre agents

- Modéliser la communication est difficile
- Le domaine de prédilection des agents est le Web… le GRID…



Développer de nouvelles architectures et de nouveaux langages conçus pour les agents sur le Web

Pré requis sur la communication agent

- Autonomie et adaptation pendant la communication

Réflexivité et travail sur l'interprétation pour faire évoluer dynamiquement nos agents

- La représentation qu'un agent se fait de son interlocuteur

Les Environnements Cognitifs de STROBE

Déroulement de la présentation

- Le modèle STROBE - Concept fondamentaux
  - Les Environnements Cognitifs
  - Les agents comme des interpréteurs
  - Protocole de communication
- Évolutions du modèle - Intégration du méta-niveau
  - Les 3 niveaux d'abstraction et d'apprentissage
  - Ajout des interpréteurs dédiés
- Exemples et expérimentations
  - Apprentissage au méta-niveau par la communication
  - Spécification dynamique de problème
- Conclusion et perspectives

Le modèle STROBE

- STReam + OBject + Environment
- Basé sur les ACL classiques (KQML, FIPA-ACL…)
- Dialogue = ensemble de message échangés sous forme de flots
- Le concept de mémoire via des objet (sous forme de procédure)
- Utilisation du concept de Dynamic Scheduler
- Apprentissage comme effet secondaire de la communication

Les Environnements Cognitifs

- KQML ne gère pas la modélisation de l'utilisateur
- 2 pré requis [Cerri 96]:
  - Environnement de première classe
  - Environnement multiples pour un même objet
- Structure des liaisons conservant l'historique
- Comportement multiples, émergents des différentes interactions
- Les environnements sont modifiés pendant la conversation

Les agents comme des interpréteurs

- Boucles d'interpréteurs REPL (*Read–Eval –Print–Listen*)
- Même interpréteur pour le message et son contenu
- Les messages sont interprété par une procédure concrète d'évaluation (**`evaluate`**)
- Les effets de l'interprétation d'un message sont répercutés dans un environnement cognitif dédié à l'agent dont provient le message

Protocole de communication

- STROBE est « orienté » acte de langage (performatif)
- Format des messages :

  (kqmlmsg performative sender receiver content)

- Performatifs :
  - ***assertion*** rép: ***ack***
  - ***request*** rép: ***answer***
  - ***order*** rép: ***executed***

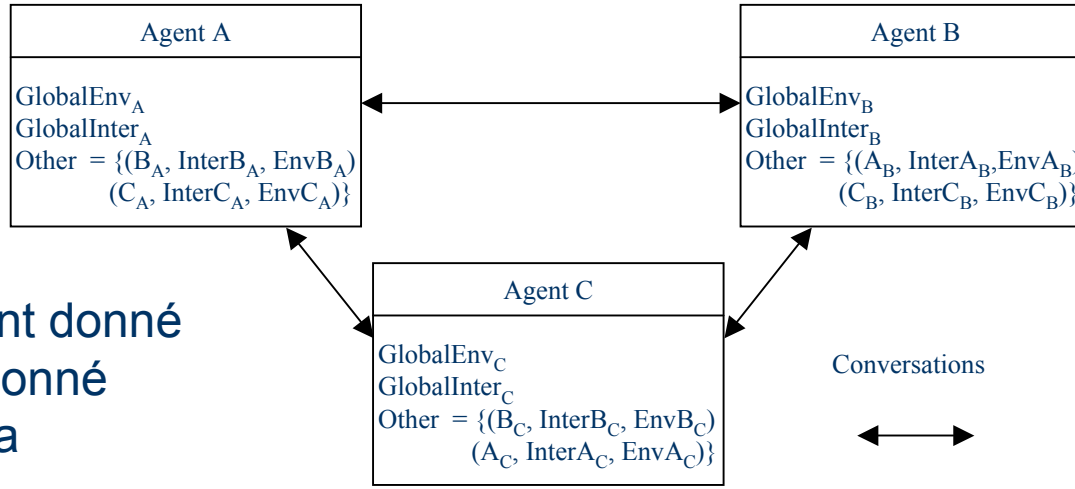
3 niveaux d'abstraction et d'apprentissage

Donnée	Contrôle	Interpréteur
`(set! a 3)` `(define b 4)`	`(define square…)`	Modifier `(evaluate e r k)` en ajoutant une forme spéciale

- Le challenge se situe au niveau interpréteur – le méta-niveau
- Comment faire évoluer STROBE de façon à ce que ce méta-niveau soit accessible ?
- Comment réaliser un apprentissage à ce méta niveau par communication ?

Ajout des interpréteurs dédiés

- Ajouts dans les Environnements Cognitifs d'interpréteurs dédiés
- Pour chacun de ses interlocuteurs un agent possède un environnement (et donc un interpréteur) dédié. Ce couple lui sert de modèle du partenaire



L' interprétation des messages est faite :
- dans un environnement donné
- avec un interpréteur donné

tous les deux dédiés à la conversation courante

Le comportement de nos agents

LISTEN
- Retirer le premier message de la file des messages en entrée
- Si ce message est nul, renvoyer(pas de message)
- Sinon transmettre ce message à READ

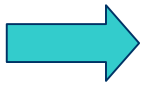
READ
- Transmettre ce message à EVAL

EVAL
- Récupérer l'évaluateur et l'environnement dédié à la conversation
- Évaluer ce message avec l'évaluateur et l'environnement choisi
- Transmettre via la continuation le résultat à PRINT

PRINT
- Si le résultat est un message alors envoyer ce message
- Sinon si le résultat est une liste de message alors envoyer ces messages
  Sinon traiter la réponse
- Revenir à LISTEN

Et le méta-niveau alors ?

- Agents = Entités autonomes contrôlé par une procédure concrète d'évaluation (**`evaluate`**)
- Ces interpréteurs correspondent au comportement d'un agent, ainsi qu'aux langages qu'il reconnaît.
- Réifier le niveau contrôle au niveau interpréteur pour rendre le méta-niveau accessible
- Alors ces interpréteurs correspondent à la connaissance d'un agent et son évolution dans le temps



Nous proposons une architecture (Scheme) qui permet de faire évoluer ces interpréteurs dynamiquement au fur et à mesure des conversations

A propos de Scheme ?

- Scheme = puissance et simplicité

- Modèle de mémoire (environnements de première classe)
- Modèle de contrôle (procédures et les continuations de première classe)

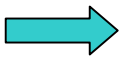
- La modification dynamique d'un interpréteur :
  - Méta-évaluateur réflexif proposant le mécanisme des *reifying procedures* qui permet à un programme d'accéder à son contexte d'évaluation et de le modifier

  - Utiliser un schéma d'évaluation du type (eval (apply 'eval …))

Première expérimentation

- Apprentissage du type « apprendre en étant dit » (learning-by-being-told)

- Dialogue type « professeur – élève » où un agent **apprend à l'issu de la conversation un nouveau performatif**.



Apprentissage au méta niveau (fonction d'interprétation des messages qui est modifiée)

Détails du dialogue

TEACHER	STUDENT
Voici la définition de la procédure square : `(kqmlmsg 'assertion teacher student '(define (square x) (* x x)))`	Ok, je connais maintenant cette procédure : `(kqmlmsg 'ack student teacher '(*.*))`
Diffuse à tous tes correspondants : `(kqmlmsg 'broadcast teacher student '(order (square 3)))`	Désolé, je ne connais pas ce performatif : `(kqmlmsg 'answer student teacher `'(no-such-performative broadcast))`

Ok, voilà comment ajouter broadcast à la liste des performatifs que tu reconnais :
Voilà le code que tu devras générer et ajouter à ta procédure evaluate-kqmlmsg :
`(kqmlmsg 'assertion teacher student learn-broadcast-code-msg)`
 	
Exécute cette procédure :
`(kqmlmsg 'order teacher student '(set! evaluate-kqmlmsg learn-broadcast-code))`
	Ok, j'ai rajouté ce code dans une variable de mon environnement :
	`(kqmlmsg 'ack student teacher '(*.*))`
	
	Ok, J'ai modifié mon évaluateur :
	`(kqmlmsg 'executed student teacher '(*.*))`

Diffuse à tous tes correspondants : `(kqmlmsg 'broadcast teacher student '(order (square 3)))`	Ok, je diffuse… `(kqmlmsg 'order student … '(square 3))`

Deuxième expérimentation

- Scénario de type e-commerce de réservation de billet de train.

  ⇒ **Spécification dynamique d'un problème** (utile pour la génération dynamique de service)

- Considérer les agents comme des interpréteurs non déterministes

- Intéressant pour la programmation par contraintes

  ⇒ Construire des programmes à contraintes par communication

  ⇒ Spécifier et coder au même moment

Dialogue e-commerce

CLIENT	SNCF

Je voudrais un billet de Montpellier à Paris
`(require (eq? depart montpellier))`
`(require (eq? dest paris))`
	Début de la construction de `find-ticket` :
	`(define (find-ticket)`
	`(let ((depart(amb *ens-ville*))`
	`(dest (amb *ens-ville*))`
	`(prix (amb *ens-prix*)`
	`(date (amb *ens-date*)`
	`(require (not (eq? depart dest)))`
	`(require (eq? depart montpellier))`
	`(require (eq? dest paris))`
	`(list (list 'depart depart)`
	`(list 'destination dest)`
	`(list 'prix prix)`
	`(list 'date date))))`
	Pour quand ?

Demain avant 10H du matin
`(require (< date *demain10H*))`
Pourriez vous me faire une proposition ?
`(find-ticket)`
	Modification de la fonction `find-ticket` en lui ajoutant la nouvelle contrainte. Puis exécution de cette fonction.
	`((depart montpellier) (destination paris) (prix 150) (date *dem9H30*))`
	Voilà, train 34170, départ demain 9H30, Montpellier en direction de Paris, 150€.

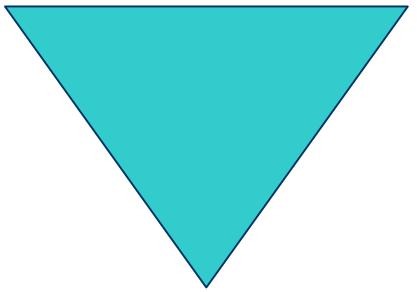
Vous n'auriez pas à moins de 100 € ?
`(require (< prix 100))`
`(find-ticket)`
	idem.
	`((depart montpellier) (destination paris) (prix 95) (date *dem8H41*))`
	Voilà, train 34730, départ demain 8H41, Montpellier en direction de Paris, 95€.

Une autre proposition s'il vous plait ? `(try-again)`	Execution de `find-ticket` : *Voilà, train 34392, départ demain 9H15, Montpellier en direction de Paris, 98€* `((depart montpellier) (destination paris) (prix 98) (date *dem9H15*))`
Ok, celui-ci me va	

Conclusion

Simplicité

Fonctionnalité



Apprentissage au méta niveau par la communication

Potentiel applicatif important (Grid, e-commerce, Web..)

Spécification dynamique d'un programme pour la génération dynamique de service

Approche centrée dialogue

Références

- Pour le modèle STROBE :
  - Cerri S.A., “Cognitive Environments in the STROBE Model”, Presented at *EuroAIED: the European Conference in Artificial Intelligence and Education*, Lisbon, Portugal, 1996.
  - Cerri S.A., “Shifiting the Focus from Control to communication: The STReams OBject Environments (STROBE) model of communicating agents”, In Padget, J.A. (ed.) *Collaboration between Human and Artificial Societies, Coordination and agent-Based Distributed Computing*, Berlin, Heidelberg, New York: Springer-Verlag, Lecture Notes in Artificial Intelligence, pp 71-101, 1999.
  - Cerri S.A., Sallantin J., Castro E., Maraschi D., “Steps towards C+C: a Language for Interactions”, In Cerri, S. A., Dochev, D. (eds), *AIMSA2000: Artificial Intelligence: Methodology, Systems, Applications*, Berlin, Heidelberg, New York: Springer Verlag, Lecture Notes in Artificial Intelligence, pp 33-46, 2000.
- Et plus récemment :
  - Clement Jonquet, Stefano A. Cerri, **Apprentissage issu de la communication pour des agents cognitifs**, *11ème Journées Francophones sur les Systèmes Multi-Agents, JFSMA'03*, Hammamet, Tunisie, Novembre 2003
  - Clement Jonquet, Stefano A. Cerri, **Cognitive Agents Learning by Communicating**, *7ème Colloque Agents Logiciels, Coopération, Apprentissage & Activité humaine, ALCAA'03*, Bayonne, France, Septembre 2003
- Pour l’aspect langage
  - Abelson H., Sussman G.J., Sussman J., *Structure and Interpretation of Computer Programs*, Second Edition, MIT Press, Cambridge, Massachusetts, 1996.
  - Friedman D.P., Jefferson S., “A Simple Reflective Interpreter”, *IMSA'92: International*