

Abstraction de systèmes cognitifs

Jacques Calmet (1) et Pierre Maret (2)

1: Université de Karlsruhe (TH), calmet@ira.uka.de

2: INSA-Lyon, pierre.maret@insa.fr

Le désir d'abstraction

Comment abstraire un processus informatique? Cette question toute simple a été et est toujours une sorte de recherche du Graal pour les informaticiens. En effet, abstraire un processus signifie qu'il est complètement compris et maîtrisé. Des réponses diverses existent en informatique théorique. Elles sont étroitement liées à l'origine de l'informatique. En effet, les scientifiques programmaient en Fortran tandis que Cobol était le langage du monde économique. Le concept d'algorithme était alors central. Un algorithme était souvent interprété comme l'image d'une preuve sinon comme un théorème, et les modèles ou théories de calculabilité conduisent directement à abstraire les concepts requis. Une abstraction existe aussi pour les langages de programmation. Elle est bien connue, en principe, de tous et s'appelle la programmation orientée objet (OOP). L'approche OOP s'est imposée aujourd'hui comme un standard, certainement par la continuité qu'elle offre de l'abstraction à l'écriture du code.

En intelligence artificielle (IA) la situation est beaucoup plus complexe. D'un côté il peut sembler que l'utilisation d'une logique engendre une abstraction presque triviale. Ceci est en fait faux. Cela est démontré par le cas de la démonstration automatique. Tous les démonstrateurs de théorèmes, sauf en géométrie, sont construits autour d'une certaine logique. Mais, le concept de raisonnement logique ne peut être abstrait que lorsqu'on formalise le processus de raisonnement. On doit introduire un concept de théorie, ensuite un niveau de contrôle permet "d'utiliser" cette théorie tandis qu'un 3e niveau formalise l'intégration dans un environnement donné. Ceci a été étendu sous le nom de OMSCS (Open Mechanized Symbolic Computation Systems) aux problèmes calculatoires (Ber99). C'est à ce jour le seul modèle abstrait en IA qui soit théoriquement correct. Il existe en effet des preuves qu'OMSCS produit des résultats corrects.

D'un autre côté, une caractéristique de l'IA est qu'une de ses composantes de base est la représentation des connaissances. Les connaissances peuvent être structurées ou non, complètes ou non, certaines ou non par exemple. Au cours des 10 dernières années des progrès spectaculaires ont été faits dans ce domaine. La technologie des systèmes médiateurs a conduit à définir des structures de données communes et sémantiquement bien définies. Des méthodes de résolutions de conflits en sont issues et ont facilité nombres d'applications. Aujourd'hui le concept de mobilité est omniprésent et nécessite un traitement dynamique des systèmes cognitifs. En même temps le e-business s'est imposé et l'avenir de l'économie européenne est vu depuis Bruxelles à travers une société des connaissances qui devrait remplacer l'industrie classique.

Ceci implique que les aspects de sécurité et de confiance sont devenus incontournables.

La méthodologie des systèmes multi-agents a été introduite, il y a 15 ans environ pour traiter les questions d'IA distribuée. Elle s'est rapidement imposée comme méthodologie de gestion des systèmes distribués depuis une structure hospitalière jusqu'à un modèle d'échange P2P. Le but de cette note est d'examiner comment un concept d'abstraction pour les systèmes multi-agents peut être introduit. Les premiers modèles ont été centrés sur OOP.

La programmation orientée objet

Il est très facile de résumer le concept d'OOP au travers des 5 points suivants:

- Les objets,
- Les classes,
- L'héritage,
- L'encapsulation,
- Le polymorphisme.

Il est à remarquer que certaines définitions de OOP introduisent l'abstraction en classes comme caractéristique de base alors que d'autres omettent la structure de classe. En fait l'abstraction est une propriété plus qu'une caractéristique. Pour comprendre l'OOP on doit penser "objet". Un objet est une collection de données et de méthodes pour manipuler ces données. Cette collection est aussi définie par les messages que cet objet peut recevoir et émettre. Les méthodes qui manipulent ces messages sont l'unique interface des objets avec le monde extérieur. L'idée de classe est essentiellement un concept introduit pour faciliter l'action du compilateur. En fait, c'est un prototype pour un objet. Cela explique simplement pourquoi un objet est une instance d'une classe, qu'il connaisse sa classe mais qu'une classe ne connaisse pas tous les objets qu'elle instancie. Les variables et méthodes de classes restent des facilités de programmation, non présentes dans les fondements de l'approche. L'encapsulation signifie que les données d'un objet ne sont accessibles que par cet objet et pas par les autres objets. On peut donc interpréter un objet comme une boîte noire. Cela implique que les invariants qui caractérisent un programme ne peuvent être brisés.

Le polymorphisme signifie que les collections d'objets et les références à des objets peuvent concerner des objets de différents types. L'héritage organise et facilite le polymorphisme et l'encapsulation au moyen des classes qui sont organisées en arbre ou en réseau. Ce réseau de relations entre objet explique comment un programme fonctionne.

OOP est un paradigme en ce sens que de nos jours c'est la manière quasiment unique de concevoir un langage de programmation. Shoham (Sho93) a été le premier à transposer ce modèle d'abstraction aux systèmes multi-agents sous le nom de AOP (Agent-Oriented Programming). Son modèle (qui n'est pas arrivé à s'imposer dans le domaine multi-agents comme OOP en programmation) est en fait un modèle BDI (Belief, Desire, Intention) de représentation cognitive. Il a donné lieu à de nombreuses discussions qui aboutissent à la conclusion suivante: ce modèle n'est pas une véritable abstraction pour les systèmes multi-agents (MAS).

Une abstraction orientée agents

Le challenge est donc de définir une abstraction de systèmes cognitifs qui s'applique aux MAS. Le modèle (Cal04) que nous introduisons ici n'est sûrement pas la réponse définitive à ce challenge. Mais, il montre qu'il existe des solutions. Il repose sur quelques définitions. La première consiste à définir un agent. Il n'existe pas de définition unanimement acceptée. Celle-ci est inspirée à la fois par l'IA distribuée et par les approches e-business.

Définition 1: Un agent consiste en des connaissances annotées couplées à un mécanisme de prise de décisions.

Une décision est évaluée à travers son facteur d'utilité. Ce dernier peut avoir pour origine la théorie des jeux créée par Morgenstein et Von Neumann vers 1930 ou un modèle statistique de classification de données ou encore un modèle spécifique utilisé soit en économie soit en physique par exemple.

Définition 2: Les décisions sont le mécanisme au travers duquel un agent peut atteindre les buts qui lui sont assignés. Il est basé sur les connaissances disponibles chez un agent. Il est caractérisé par un facteur d'utilité.

Les classes de connaissance que nous devons pouvoir considérer sont diverses et variées. Elles peuvent être structurées ou non, complètes ou non, certaines ou non pour ne citer que les cas les plus connus. À cette fin on peut introduire un concept d'annotation des connaissances. L'idée d'annotation est semblable à celle que l'on trouve pour la logique annotée ou pour annoter les listes de propriétés dans le langage de programmation LISP.

Définition 3: La connaissance est annotée en classes (ou types). Celles-ci structurent la connaissance disponible chez un agent ou accessible par un agent.

Une structure de classes est également introduite pour évaluer la qualité des décisions prises. Cette structure correspond aux diverses possibilités de mesure de l'utilité des décisions (facteur d'utilité déjà évoqué plus haut). Cela conduit à la définition suivante:

Définition 4: L'utilité du mécanisme de décision est une mesure de l'efficacité de ce mécanisme. Il est structuré en classes d'utilités.

L'aspect sociétal est indispensable pour rendre compte de l'organisation d'un système multi-agents. Ceci est communément associé au rapport entre les agents et leur environnement. En d'autres termes, on doit spécifier le niveau de contrôle que le système de gestion impose sur les agents. On peut se référer à des concepts de sociologie. Notre modèle est inspiré de la théorie de Weber qui

dit que les actions déterminent la société et qui peut aussi être adopté pour développer des modèles mathématiques montrant que la micro-économie engendre la macro-économie.

Définition 5: Une société d'agents est l'organisation sociale résultant des actions exécutées par les agents individuels appartenant à l'univers des agents actifs pour résoudre un problème donné.

La dernière définition requise doit exprimer le fait que diverses méthodes sont disponibles pour approcher la problématique des organisations en agents. Au niveau supérieur, on peut avoir par exemple des architectures logiques ou BDI ou en couches. On peut simuler la communication et la négociation entre agents par des protocoles synchrones ou non, par des modèles issus du monde des affaires comme les enchères. Les diverses facettes des systèmes multi-agents sont très bien introduites dans (Wes99).

Définition 6: Une spécialisation est l'implantation logicielle de classes abstraites pour la connaissance ou les fonctions d'utilité.

Nous avons très brièvement introduit une d'abstraction orientée agent dont la vertu principale est de montrer que l'abstraction est possible en intelligence artificielle distribuée. Pour ce domaine, cette approche est plus adaptée que les approches OOP et AOP. Les principales caractéristiques d'OOP (encapsulation, héritage, polymorphisme) peuvent être cependant exploitées, à travers les classes d'annotations. Nos travaux se poursuivent dans la définition formelle de l'abstraction, et dans son application à la problématique des organisations en agents : systèmes pervasifs, questions de sécurité et de confiance, connaissances en entreprises distribuées (Mar04).

Bibliographie

(Ber99) P.G. Bertoli, J. Calmet, F. Giunchiglia, K. Homann *Specification and Integration of Theorem Provers and Computer Algebra Systems*. Fundamenta Informaticae, Vol. 39, No. 1-2, 39-57, 1999

(Sho93) Y. Shoham. *Agent-oriented programming*. Artificial Intelligence 60, pp. 51—92, 1993.

(Cal04) J. Calmet, P. Maret, R. Endsuleit. *Agent-Oriented Abstraction*. RACSAM (Revista Real Academia de Ciencias, Serie "A" de Matemáticas) Vol. 98 (1), 2004

(Wei99) G. Weiss. *Multiagent systems: a modern introduction to distributed artificial intelligence*, MIT Press. 1999

(Mar04) P. Maret, J. Calmet. *Modeling Corporate Knowledge within the Agent Oriented Abstraction*. International Conference on Cyberworlds (CW), IEEE Computer Society, November 18–20, 2004, Tokyo, Japan.