Construction modulaire d'agents et de systèmes multi-agents adaptatifs en DIMA

Zahia Guessoum, Thomas Meurisse et Jean-Pierre Briot

Laboratoire d'informatique de Paris VI (LIP6)
UPMC 4 Place Jussieu
Case 169, 4 Place Jussieu, 75252 Paris Cedex 05
{Zahia.Guessoum, Thomas.Meurisse, Jean-Pierre.Briot}@lip6.fr

RÉSUMÉ. *Cet article présente une approche de construction d'agents par assemblage et raffinement de composants, mis en œuvre dans la plate-forme DIMA. Le point de départ est la conception d'un composant minimal de pro-activité, étendant la notion d'objet vers la notion de composant pro-actif. Ce composant est ensuite spécialisé incrémentalement de manière à représenter différents types d'architectures d'agents (agents réactifs, agents adaptatifs, …). Nous présentons ensuite comment concevoir une organisation multi-agent adaptative par réification du réseau de dépendances entre agents.*

ABSTRACT. *This paper presents an approach to build agents by combining and specializing components. This approach is implemented in DIMA. The starting point is the design of a minimal proactive component that extends the reactive object notion to proactive component. This component is then incrementally specialized to represent various agent architectures (reactive agent, adaptive agent,...). The paper shows then how to design an adaptive multiagent organization where an agent-dependence net is reified.*

MOTS-CLES *: objet, modèles d'agents, modèles multi-agents*
KEYS-WORDS *: object, agent models, multiagent models.*

1. Introduction

Cette dernière décennie a vu fleurir un très grand nombre de modèles, de langages et d'architectures dans le but de développer des systèmes à plusieurs composantes autonomes (appelées agents) pouvant coopérer entre elles.

Ainsi, de nombreux travaux sur les architectures d'agents ont tenté d'identifier les différentes fonctions et composantes (perception, communication, délibération,...) d'un agent et la façon dont elles sont organisées pour fournir le comportement global d'un agent. Ces différentes composantes sont ensuite réifiées pour faciliter la spécification d'un agent. Les architectures d'agents proposées offrent une solution à une partie des problèmes que posent le développement des applications multi-agents. En effet, le problème lié à la difficulté d'implémenter les systèmes multi-agents reste entier notamment par un non-spécialiste des systèmes multi-agents.

La difficulté de réalisation d'un système multi-agent s'explique principalement de par la diversité, la richesse des paradigmes multi-agents, et la complexité des concepts sous-jacents (coordination, interaction, organisation, etc.). Cette complexité rend l'utilisation de la majorité des outils existants toujours difficile pour ceux qui ne sont pas les concepteurs et souvent presque impossible pour les non-spécialistes des systèmes multi-agents.

L'objectif de ce papier est de montrer que la modularité et la réutilisabilité peuvent apporter une solution à ces problèmes. Nous proposons une plate-forme basée sur une architecture modulaire et une bibliothèque de composants. Elle est fondée sur l'extension des facilités de modélisation et d'implémentation de langages à objets. Nous voulons rendre un environnement à objets plus approprié aux problèmes de systèmes multi-agents en lui incorporant des représentations spécifiques et des structures de contrôle. Plus concrètement, dans cet article nous décrivons l'extension d'un modèle d'objets en modèles d'agents génériques et modulaires (incarnés par la plate-forme DIMA –Développement et Implémentation de Systèmes Multi-Agents). Nous introduisons ensuite un nouveau modèle organisationnel pour les systèmes multi-agents adaptatifs.

L'article comprend cinq sections. La deuxième section présente un modèle d'architecture d'agents et montre que ce modèle offre les principales propriétés des agents. La troisième section introduit une architecture de systèmes multi-agents adaptatifs. Cette architecture est illustrée par un exemple d'application, les systèmes tolérants aux pannes. La quatrième section donne une vue générale de la plate-forme DIMA. Nous terminons l'article par une discussion pour souligner l'intérêt et les caractéristiques des modèles proposés ainsi que celles de la plate-forme DIMA et quelques perspectives.

2. Architecture d'agents

Les systèmes multi-agents existants sont classés en général en deux groupes : les systèmes cognitifs et les systèmes réactifs. Les systèmes multi-agents cognitifs sont fondés sur la coopération d'agents capables, à eux seuls, d'effectuer des opérations complexes [Dem 90] [Hay 95] [Jen 92]. Chaque agent dispose d'une capacité de raisonnement, d'une aptitude à traiter des informations diverses liées au domaine d'application, et d'informations relatives à la gestion des interactions avec d'autres agents et l'environnement.

Dans un système multi-agent réactif [Bro 86] [Fer 92], le comportement complexe du système émerge de la coexistence et de la coopération d'agents au comportement simple. Les agents réactifs ne disposent que d'un protocole de communication réduit. Ils répondent uniquement à une loi de type stimulus/réponse.

Récemment, une synthèse s'est amorcée entre ces deux orientations et on parle alors d'agents hybrides [Fer 95] [Mül 94]. Ces agents combinent des propriétés cognitives et réactives généralement logées dans des modules différents. Ce type d'architecture pose le problème du contrôle à deux niveaux : la coordination classique entre agents, et la coordination entre modules au sein même de l'agent.

Les architectures hybrides apportent une solution au problème de l'intégration des aspects réactifs et cognitifs. Ces architectures présentent des qualités indéniables de génie logiciel. Leur organisation modulaire permet entre autre l'intégration de capacités très hétérogènes en terme de programmation. Cette hétérogénéité est nécessaire pour répondre à l'intégration des tâches très diversifiées qu'a à remplir un agent depuis le raisonnement jusqu'à l'interaction qui peuvent faire appel à des concepts de système des plus spécialisés. L'intérêt est de pouvoir réutiliser des méthodes existantes notamment des méthodes d'intelligence artificielle [Gue 01].

La modularité introduit beaucoup de souplesse dans la gestion des capacités en autorisant à moindre coût l'échange de modules dans un but d'évolutivité ou de test. Une architecture modulaire fait de l'agent un système ouvert [Gue 01].

Cependant, les architectures hybrides sont complexes et ne seraient pas appropriées à la modélisation d'agents dont le comportement est très simple. Par exemple, la conception d'un agent avec *Interrap* (voir [Mül 94]) nécessite la conception des trois modules correspondant aux trois couches (comportement, planification, coopération) de l'architecture. Il est donc difficile d'utiliser cette architecture pour concevoir un agent dont le comportement est très simple (par exemple, basé sur le principe de l'éco-résolution [Fer 92]). Ces architectures hybrides ne résolvent pas le problème de granularité. Elles ne permettent pas la conception de systèmes multi-agents dont la granularité des agents est variable.

2.1. Un modèle d'architecture qui dépasse la dichotomie classique

Le modèle d'architecture d'agents DIMA propose de décomposer chaque agent en différents composants dont le but est d'intégrer des paradigmes existants notamment des paradigmes d'intelligence artificielle. Un agent peut ainsi avoir un ou plusieurs composants qui peuvent être réactifs ou cognitifs (voir Figure 1). Ce modèle permet de dépasser la dichotomie classique réactif/cognitif en permettant de définir des agents hétérogènes au sein d'une même application. Chaque agent est composé d'un ou de plusieurs composants.

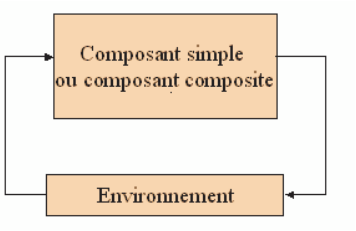

Figure 1 : Architecture d'agents

La modularité offre ainsi plusieurs avantages à cette architecture :

- Possibilité de définir des agents à granularité variable.
- Possibilité de définir des agents à structure adaptative. Chaque agent peut dynamiquement changer ces composants ainsi que les relations entre ces différents composants.
- Possibilités d'intégrer différents modèles d'agents.
- Possibilité de définir des bibliothèques de composants réutilisables.
- Etc.

Le modèle d'architecture d'agents DIMA est donc fondé sur la conclusion suivante : les travaux sur les architectures d'agents ont engendré plusieurs résultats intéressants. Il est très difficile de comparer ces architectures dans le but de trouver la meilleure architecture. Chacune est bien appropriée à une catégorie d'agents. Une bonne plate-forme intègre les différentes architectures d'agents existantes ainsi que d'éventuelles nouvelles architectures. Ainsi, le modèle d'architecture d'agents proposé par DIMA peut être vu comme un modèle 'ouvert', une proposition d'agent minimal est faite permettant, par raffinements successifs, d'ajouter des fonctionnalités fournies par les différentes bibliothèques de la plate-forme. La force de DIMA réside donc à la fois dans sa proposition de modèle d'architecture d'agents modulaire, mais également dans les différentes bibliothèques disponibles. Chaque agent est un composant simple ou composant composite [Meu 01] qui gère l'interaction de l'agent avec son environnement et qui représente son comportement interne (voir Figure 1). L'environnement regroupe tous les autres agents ainsi que les entités qui ne sont pas des agents.

Dans les sections suivantes, nous présentons d'abord les composants proactifs, la brique de base de cette architecture. Ensuite nous décrivons les différents modèles d'agents qui sont fondés sur ces composants proactifs.

2.2. Les Composants proactifs

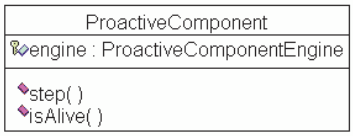

Figure 2 : Structure générale d'un composant proactif

Un composant proactif représente une entité autonome et proactive. Le noyau de base de DIMA est un *framework* de composants proactifs. Il est composé de plusieurs classes et de leurs méthodes. Nous avons choisi un ensemble minimal de classes et de méthodes définissant les fonctionnalités des composants proactifs. Ces fonctionnalités peuvent être étendues dans les sous-classes. Ce noyau minimal est principalement composé de la classe *ProactiveComponent* (voir Figure 2 [1]).

Contrairement aux objets dont l'activité est restreinte au traitement des messages envoyés par les autres objets, le composant proactif a différentes compétences et son activité n'est pas restreinte à l'envoi et à la réception de messages. Une instance de la classe *ProactiveComponent* décrit :

- Le but du composant proactif, il est implicitement ou explicitement décrit par la méthode *isAlive()*.
- Les compétences de base du composant proactif, une compétence est une suite d'actions qui permettent de changer l'état interne de l'agent ou envoyer des messages aux autres agents. Dans notre implémentation une compétence pourrait être implémentée sous forme d'une méthode Java.
- Le comportement, il définit la manière dont les compétences sont sélectionnées, séquencées et activées en fonction du but.

La Table 1 décrit les principales méthodes de la classe *ProactiveComponent*.

Principales Méthodes de la classe ProactiveComponent	**Description**
public abstract boolean isALive()	Teste si le composant proactif n'a pas atteint son but.
public abstract void step()	Décrit un cycle du comportement du

[1] Les différents diagrammes de cet article sont des schémas UML générés avec Rational Rose

	composant proactif.
void proactivityLoop()	Représente le comportement du composant proactif. *public void proactivityLoop() {* *while (this.isAlive())* *{ this.preActivity();* *this.step();* *this.postActivity();* *}}*
public void startUp()	Initialise et active le comportement. *public void startUp() {* *this.proactivityInitialize();* *this.proactivityLoop();* *this.proactivityTerminate();* *}*

Table 1 : Principales méthodes de *ProactiveComponent*

Différents paradigmes (Automate, règles de production,...) sont réutilisés pour définir de nouvelles classes de composants proactifs. Par exemple, le comportement d'un composant proactif peut être modélisé par un ATN (Augmented Transition Network) dont les états correspondent aux différents états d'un composant proactif. A chaque état est associé une ou plusieurs transitions. Une transition est définie par une condition et une action. L'action de transition est l'activation d'une compétence.

La Figure 3 donne des exemples de composants proactifs. Dans la classe *ATNBasedProactiveComponent*, la méthode *step()* permet d'activer une transition dont la condition est vérifiée.

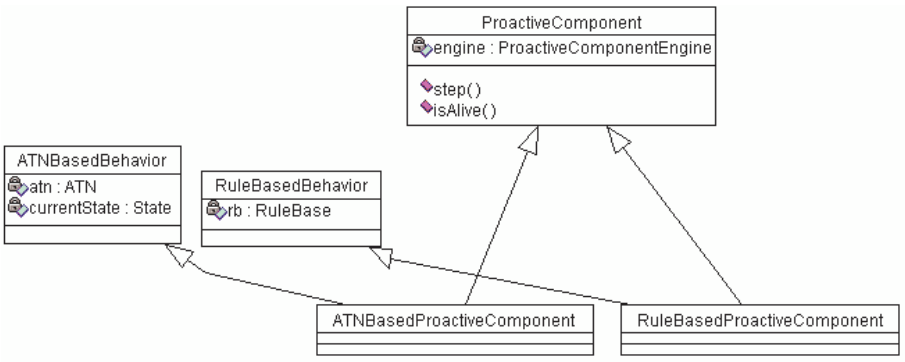

Figure 3 : Exemple de composants proactifs

```
public void step() {
currentState = currentState.crossTransition(this);
}
```

Pour construire un composant proactif, on doit décrire son comportement et ses compétences de base en sous classant la classe *ProactiveComponent*. On doit également décrire la méthode *isAlive()*. Par exemple, dans le cas où le comportement du composant proactif serait décrit par un ATN, cette méthode teste si l'état final n'est pas atteint.

```
public boolean isAlive() {
     return !(currentState.isFinal());
 }
```

La définition d'un composant proactif ou l'introduction d'un nouveau type nécessite de définir une sous classe des classes de la hiérarchie de la Figure 3.

Figure 4 : Modèle du gestionnaire de composants proactifs

Une application à base de composants proactifs intègre notamment un ensemble de composants proactifs et éventuellement un gestionnaire (instance de la classe *ProactiveComponentManager*). Cette classe (voir Figure 4) offre le choix d'activer le même ensemble de composants proactifs en simulation ou avec des *threads* en choisissant respectivement une des méthodes *startAll()* ou *startAllWithThreads()*.

```
public void startAllWithThreads () {
Vector pv = proactiveComponents;
for (int i=0; i<pv.size();i++)
{ProactiveComponent pao =
     (ProactiveComponent) pv.elementAt(i);
ThreadedProactiveComponentEngine paoEngine =
     new ThreadedProactiveComponentEngine(pao);
paoEngine.startUp();
}}
```

Dans les sections suivantes, nous allons montrer comment ces composants proactifs sont réutilisés pour définir différents modèles d'agents.

2.3. Des composants proactifs aux agents réactifs

Le composant proactif peut être considéré comme une bonne brique de base pour construire des agents. Par exemple, nous pouvons implémenter avec les ATNBasedProactiveComponent des agents réactifs tels qu'ils sont décrit par J. Ferber et A. Drogoul [Fer 92]. Le comportement d'un agent réactif est, dans ce cas, décrit par un ensemble de règles :

Si stimulus et état alors action

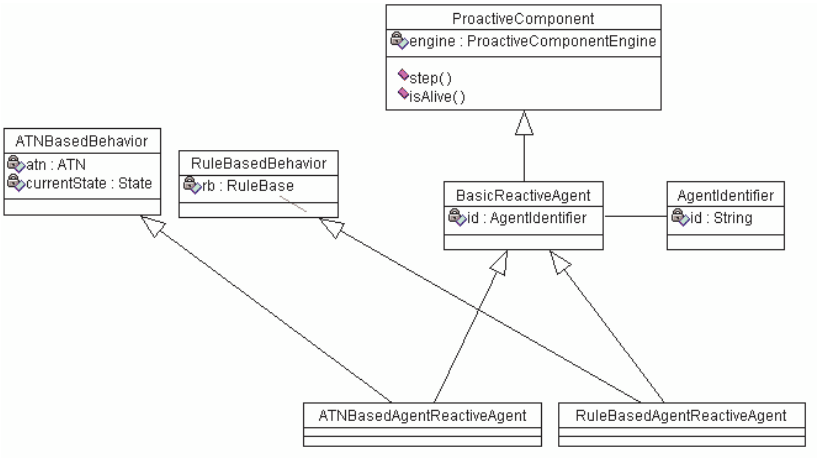

Figure 5 : Modèle d'agents réactifs

Les classes de la hiérarchie sont ainsi réutilisées pour définir des classes d'agents (voir Figure 5). Par exemple la classe ATNBasedReactiveAgent définit des agents réactifs dont le comportement est décrit par des ATN.

Ces agents réactifs sont ensuite enrichis pour définir des agents interactifs (voir figure 6). Les compétences de ces agents intègrent deux types d'actions :

- des actions liées aux concepts d'agents (envoi et réception de messages, …),
- des actions liées au domaine.

Ces différentes classes d'agents peuvent décrire des agents réactifs ou cognitifs selon le paradigme choisi et selon les connaissances représentées. Par exemple, les ATN sont souvent utilisés pour décrire des agents réactifs et les bases de règles sont souvent utilisées pour décrire des agents cognitifs.

De plus, la classe *CommunicationComponent* est sous classée pour intégrer un langage de communication tel que KQML ou FIPA-ACL[2] [FIP97] .

Ainsi, la définition d'un agent s'opère d'une façon relativement simple. La conception d'un agent réactif minimaliste, puis d'un agent interactif, peut se faire

[2] KQML : Knowledge Query and Manipulation Knoweldge
ACL : Agent Communication Language

par raffinements successifs de la classe de base proposée. Cette simplicité d'utilisation de la plate-forme est héritée de la programmation par objets.

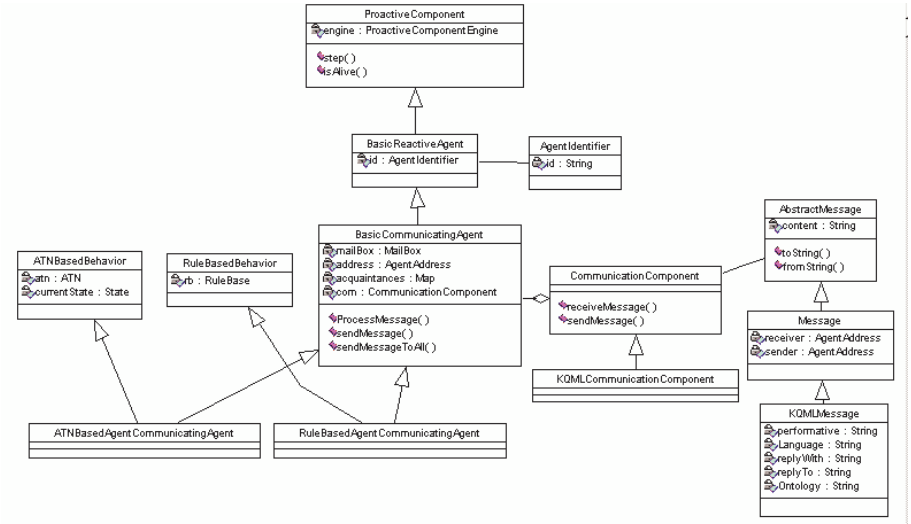

Figure 6 : Exemple de modèle d'agents interactifs

2.4. Des agents adaptatifs

Depuis peu, les travaux en adaptation ont fait apparaître l'opposition réactif/cognitif sous un nouveau jour. Il ne s'agit plus de choisir l'un ou l'autre, mais de cumuler les avantages de chacun en utilisant des couches réactives, pour réagir rapidement aux stimuli connus, et des couches cognitives, pour adapter les réactions de la couche réactive aux variations imprévues de l'environnement.

Il faut donc pouvoir doter l'agent d'un système de décision principal, appelé système comportemental, faisant le lien entre les données reçues en entrée et les données qu'il doit fournir en sortie. Si ce système comportemental est suffisant pour permettre à l'agent d'interagir dans son environnement, il ne permet pas pour autant (du moins, dans le cadre des agents réactifs, qui nous intéressent ici) de s'adapter. Pour permettre une adaptation de l'agent, il devient nécessaire de lui greffer un ou plusieurs autres systèmes décisionnels qui s'en chargent. Ces autres systèmes, bien que pouvant gérer directement le comportement de l'agent, sont surtout là pour mesurer, tester et améliorer la réactivité de l'agent. Pour ce faire, ils doivent disposer d'informations sur l'agent et son environnement, mais surtout d'informations concernant le fonctionnement du système comportemental, des méta-informations. Ils doivent de plus adapter ce système comportemental par le biais de méta-actions et, ainsi, adapter l'agent à son environnement.

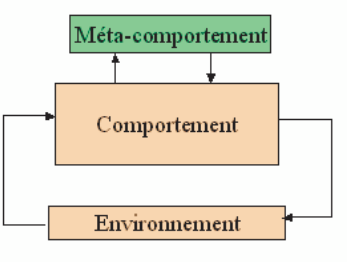

Figure 7 : Architecture d'agents adaptatifs

Par ailleurs, plusieurs chercheurs ont souligné l'intérêt de la représentation explicite et séparée du contrôle ou de l'aspect réflexif des architectures de méta-niveaux (voir par exemple [Rao91] et [Pit 90]). En adoptant, cette représentation explicite et séparée du contrôle, nous proposons d'introduire un méta-comportement dans notre architecture d'agents adaptatifs (voir Figure 7). Ce méta-comportement donne à chaque agent la capacité de prendre des décisions appropriées au sujet de contrôle ou d'adapter son comportement avec le temps à des nouvelles circonstances. Il fournit à l'agent un mécanisme d'auto-contrôle pour adapter dynamiquement ses comportements selon son état interne et celui de son monde.

Le méta-comportement se base sur les données de l'agent lui-même, son univers, et le système de décision utilisé.

Le comportement et le méta-comportement sont basés sur deux types d'éléments : *actions* et *conditions*, et ils ont un système de décision.

Pour implémenter cette structure, nous réutilisons la classe *ProactiveComponent*. La nouvelle classe a les principaux composants suivants :

- un ensemble de conditions destinées à être utilisées par le méta-comportement pour tester l'état du comportement,
- un ensemble d'actions qui peuvent être utilisées par le méta-comportement pour modifier le comportement.

Cette structure est très générale et peut être utilisée avec n'importe quel système décisionnel (ATN, base de règles, etc.), il suffit qu'il fournisse un lien entre les conditions et les actions.

Actuellement DIMA intègre trois types de méta-comportements : *RuleBasedMetaBehavior*, *GeneticBasedMetaBehavior*, *ClassifierBasedMeta-Behavior*. Chacun de ces méta-comportements est utilisé avec deux types de comportements : *RuleBasedProactiveComponent* et *ATNBasedProactiveComponent*.

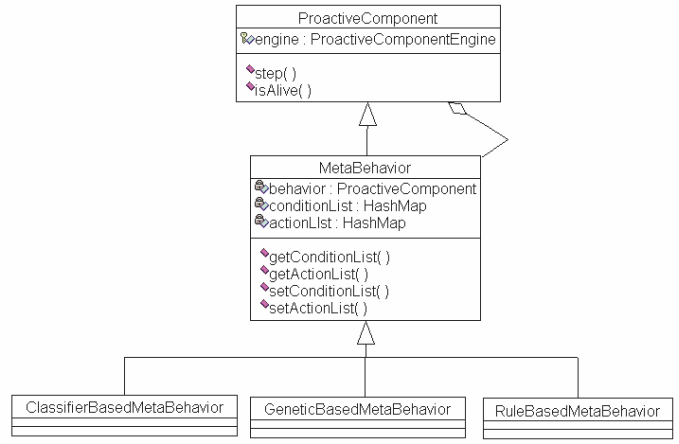

Figure 8 : Hiérarchie des classes représentant les agents adaptatifs

Ce méta-comportement a deux principaux rôles :

- Adaptation structurelle : il s'agit d'adapter la structure de l'agent à l'évolution de son environnement. Par exemple, l'arrivée de nouveaux agents hétérogènes dans le système ou tout simplement dans la liste des accointances d'un agent peut nécessiter le remplacement de son composant CommunicationComponent (basé sur des simples envois de messages) par KQML CommunicationComponent ou l'ajout de ce dernier.
- Adaptation comportementale : il s'agit d'adapter le processus de décision de l'agent à l'évolution de son environnement. Par exemple, dans le cas d'un agent économique le choix d'une stratégie dépend fortement de l'évolution de la compétition. Il serait donc intéressant de doter cet agent d'un ensemble de méta-règles qui permettent de mettre à jour les données sur l'évolution de la compétition qui permettant de choisir une stratégie.

Ces deux rôles peuvent être combinés pour avoir à la fois des agents qui adaptent leur structure et leur comportement.

Ce modèle peut être utilisé hors-ligne pour faciliter la conception d'un agent. Il suffit donc de simuler l'évolution de l'environnement des agents et d'attendre que les bases de règles se stabilisent. Les agents peuvent ensuite être activés avec la base de règles ainsi obtenue.

2.5. Principales caractéristiques

Grâce à sa modularité, l'architecture proposée aide à décomposer le comportement arbitrairement complexe d'un agent en un ensemble de petits comportements spécialisés, et éventuellement en dirigeant ces divers comportements par un méta-comportement. Les avantages de cette architecture correspondent aux différentes propriétés recherchées d'un système multi-agent :

- **Multi-granularité** : des agents de diverses granularités (taille, comportements internes, connaissance) peuvent être implémentés avec l'architecture proposée. Cette propriété est très importante pour la conception des systèmes complexes. Elle permet de dépasser la dichotomie classique entre les agents réactifs et cognitifs.
- **Dynamicité et ouverture** : des agents peuvent être dynamiquement créés et/ou détruits. Ils peuvent intégrer de nouveaux comportements et changer leurs connaissances en fonction de l'information qu'ils reçoivent et/ou qu'ils perçoivent. De nouveaux comportements peuvent être créés et intégrés par le méta-comportement d'un agent.
- **Réflexion** : notre architecture met en application un modèle d'agents adaptatifs basé sur la réflexion dans lequel chaque agent a son propre méta-comportement qui régit ses divers comportements, par exemple afin de prendre des décisions appropriées au sujet de contrôle ou adapter ses comportements à de nouvelles circonstances.
- **Hétérogénéité** : l'hétérogénéité est une propriété très importante dans les systèmes multi-agents. Cependant, la majorité des systèmes existants ne fournissent pas la possibilité de réaliser des agents hétérogènes.

Beaucoup de travaux (voir par exemple [Dem 98] et [Ode 99]) soulignent qu'un système multi-agent est plus qu'un ensemble d'agents qui interagissent. Dans notre architecture, les agents intègrent une composante interaction.

Dans la section suivante, nous présentons différents modèles de systèmes multi-agents.

3. Des agents adaptatifs aux systèmes multi-agents adaptatifs

La majorité des méthodologies de conception de systèmes multi-agents proposées (voir par exemple Gaia [Woo 00]) est bien adaptée aux domaines qui ont les caractéristiques suivantes :

- structures organisationnelles statiques,
- capacités des agents statiques,
- petit nombre d'agents.

Les systèmes multi-agents ainsi conçus sont un ensemble d'agents qui interagissent. Chaque agent peut avoir un ou plusieurs rôles [Fer 98] et les règles d'interaction entre les différents rôles sont données par le concepteur en basant sur des protocoles d'interaction bien définis.

Les différentes applications peuvent être facilement modélisées avec les différents modèles d'agents proposés dans la section précédente. Cependant, le cas d'organisations dynamiques et adaptatives a rarement été abordé. Dans la majorité des systèmes multi-agents existants, l'adaptation des structures organisationnelles est fondée sur l'adaptation des comportements des agents et il n'est pas bien montré

comment émerge le comportement complexe du système à partir des simples règles locales des agents.

Cette section décrit d'abord un exemple d'application : systèmes multi-agents tolérants aux pannes. Il présente ensuite le modèle multi-agent adaptatif proposé.

3.1. Systèmes multi-agents tolérants aux pannes

Comme les systèmes répartis, les systèmes multi-agents sont exposés à des cadences plus élevées des pannes de leurs composants matériels et/ou logiciels : un exemple symptomatique serait un système de gestion de crises. Par ailleurs, la panne d'un composant peut entraîner la panne de tout le système. Pour éviter ce problème, la solution consiste à répliquer ces composants. Ces réplicats peuvent en effet remplacer les composants de base en cas de panne. Cependant, on ne peut pas répliquer l'ensemble des composants à cause du manque de ressources d'une part et de la performance du système d'autre part. Le problème étant de déterminer les composants à répliquer.

D'autre part, la majorité des plates-formes multi-agents n'abordent pas du tout la question de tolérance aux pannes, parce que ces plates-formes sont souvent utilisées pour développer des simulations locales et de courte durée fonctionnant souvent sur un seul ordinateur.

Nous considérons un système multi-agent où les agents sont distribués et avec un nombre de ressources limité. Nous distinguons deux cas possibles :

- La structure organisationnelle est statique : dans ce cas, les agents les plus critiques peuvent être identifiés à la conception du système. Ils peuvent en effet être répliqués, à la conception, en fonction des ressources disponibles (nombre de machines et leurs capacités par exemples).
- La structure organisationnelle est dynamique : dans ce type de système, habituellement, nous avons plusieurs structures dynamiques. Par exemple, C. Castelfranchi définit la structure d'interdépendances, la structure de communications, etc. (voir [Cas 90]). Le mécanisme de réplication est, dans ce cas, basé sur la structure d'interdépendances. Le problème est comment définir et comment mettre à jour cette structure dynamique ?

La section suivante apporte une réponse à cette question. Elle propose un modèle multi-agent à deux niveaux.

3.2. Un modèle multi-agent adaptatif

Nous proposons de considérer l'organisation comme la propriété la plus importante d'un système multi-agent. Nous adoptons la définition de Les Gasser : "... *an organization is a particular set of settled and unsettled questions about beliefs*

and actions through which agents view other agents. So, the organization relies on: mutual commitments, global commitments, and mutual beliefs ... » [Gas 90].

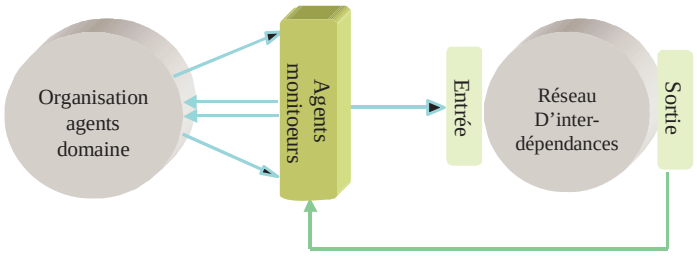

Figure 9 : Différents composants d'un SMA adaptatif

Nous proposons de réifier la partie adaptative de l'organisation [Car 00]. Un système multi-agent se compose donc (voir Figure 9) :

- D'une composante *organisation d'agents* (niveau micro) qui regroupe l'ensemble des agents du domaine dans un réseau classique d'accointances. Ce réseau définit la communication explicite avec des messages spécialisés explicites entre agents (par exemple, des messages de KQML ou de ACL), et les connaissances de domaine. Chaque agent résout une partie du problème global.
- D'une *composante adaptative* (niveau macro) qui peut-être définie par un ensemble de nœuds reliés et qui forment le réseau d'interdépendances. Elle réifie l'ensemble des tendances du système. Comme cela a été souligné par C. Castelfranchi, "this part represents the glue of groups, it links the agent with joint goal and the common solution, it links members with each other" [Cast 98]. Chaque nœud de ce réseau correspond à un agent du domaine et les connexions entre ces nœuds représentent les dépendances entre ces agents. Ce réseau peut-être donc représenté par une matrice W où $W_{i,j}$ décrit la dépendance entre les agents du domaine $Agent_i$ et $Agent_j$.
- D'un ensemble d'*agents moniteurs* qui observent continuellement la première composante, établissent un état du système et fournissent cette information à la composante adaptative. Ils peuvent également contrôler l'organisation d'agents du domaine.

3.2.1. Activité d'un système multi-agent adaptatif

Un système multi-agent adaptatif a trois types d'activités :

- l'activité de l'organisation d'agents du domaine,
- le monitoring qui est l'activité des agents moniteurs,
- l'adaptation qui est l'activité du réseau d'interdépendances.

Dans cette section nous présentons ces trois activités. L'exemple que nous avons choisi pour les illustrer est la tolérance aux pannes (voir la section 3.1).

Activité de l'organisation d'agents du domaine

Les agents du domaine sont reliés en fonction des caractéristiques de l'application. Ils agissent continuellement en fonction de leurs buts, l'évolution de leur environnement et éventuellement en fonction de l'évolution du réseau d'interdépendances [Cas 98]. La fin de l'activité de ces agents entraîne la fin de l'activité du système.

Monitoring

L'activité de monitoring dépend de l'activité de l'organisation d'agents du domaine et de leur environnement. Il s'agit d'observer les agents du domaine, analyser les événements observés et contrôler ces agents. Par exemple, pour les systèmes multi-agents tolérants aux fautes, nous associons un agent moniteur à chaque machine et nous considérons ainsi :

- le nombre de messages envoyés par un agent,
- le type et le contenu des messages envoyés par un agent,
- la quantité d'information envoyée par un agent,
- le(s) rôle(s) de l'agent,
- l' historique de l'activité de l'agent,
- etc.

Cet ensemble de données est déduit de l'ensemble des événements observés.

Chaque agent moniteur définit alors un vecteur $D_{i,\ i=1,n}$ pour chaque agent $Agent_i$ qu'il observe. Par exemple, $D_i[1]$ peut représenter le nombre de messages envoyés par l'agent $Agent_i$ à l'agent $Agent_1$. Ce vecteur est mis à jour à chaque intervalle de temps Δt par la procédure d'analyse. Les données ainsi définies représentent les attributs qui caractérisent l'état du système multi-agent durant cet intervalle.

Un autre rôle de ces agents moniteurs est l'interprétation du réseau d'interdépendances fournit par la composante adaptative. Ce réseau est un graphe orienté représenté par une matrice W. Chaque $W_{i,j}$ décrit la dépendance entre $Agent_i$ et $Agent_j$. Le réseau d'interdépendances est alors analysé pour déterminer les poids des agents. Le poids w_i de chaque agent $Agent_i$ peut être calculé en fonction de ses dépendances $W_{i,j}$ comme suit :

$$w_i = \Sigma_{j=1,n} W_{i,j}$$

Dans un système multi-agent tolérant aux pannes, les agents sont ensuite répliqués par les agents moniteurs en fonction de leurs poids (les mécanismes de réplications sont décrit dans [Mar 01]).

Ainsi, l'organisation des agents qui réalise le monitoring maintient deux types d'informations : la tendance du comportement des agents et la variation du réseau d'interdépendances. Toute modification des agents du domaine est alors perçue par

les agents moniteurs et propagée au réseau d'interdépendances qui est adapté en conséquence. D'autre part, toute modification du réseau d'interdépendances est également interprétée par les agents moniteurs afin d'agir sur les agents du domaine (mettre à jour le nombre de réplicats).

Dans certaines applications telles que les simulations économiques, les données d'interdépendances pourraient être utilisées par les agents pour raisonner sur les autres agents.

Adaptation

Les modifications du réseau d'interdépendances sont contrôlées par les connaissances organisationnelles. Le but des connaissances organisationnelles est d'offrir au système :

- *une capacité d'adaptation* pour permettre au système de tenir compte des nouvelles contraintes et de nouvelles données des agents.
- *Une capacité d'auto-organisation* pour assurer la stabilité du système en tant que système dynamique. Cette capacité présente un intérêt déterminant pour tous les problèmes évolutifs.
- *une capacité de généralisation* en se basant sur les exemples de problèmes que l'on peut résoudre. Cette propriété permet de palier le problème que pose l'acquisition de connaissances. Le but est de retrouver les règles qui permettent de résoudre le problème à partir d'un ensemble d'exemples.

Pour la représentation de ces connaissances, nous pouvons, par exemple, utiliser les réseaux de Kohonen [Koh 90]. Ces réseaux offrent toutes les propriétés citées ci-dessus ainsi que de nombreux autres avantages tel que la distribution.

Les réseaux de Kohonen proposent une structure très générale pour représenter ce type de problème. Dans ces réseaux, chaque neurone est caractérisé par sa fonction d'activation. Le réseau d'interdépendances (matrice W) est ainsi représenté par une carte de Kohonen où un neurone représente un agent.

La structure proposée par Kohonen tient compte à la fois des données externes et des connexions internes. Elle permet de décrire en fonction des deux types de connexions (connexions entre les éléments d'entrées et celles entre les éléments de sorties) des équations d'état dynamiques pour décrire le fonctionnement et l'apprentissage du réseau. Dans notre cas, les données externes représentent les variations des agents durant l'intervalle de temps Δt. Par ailleurs, les données internes représentent les données des différents agents et leurs dépendances ($W_{i,j}$). Notons que $W_{i,j}$ et différent de $W_{j,i}$, la fonction de dépendance n'est pas réciproque.

Les $W_{i,j}$ sont mis à jour par l'algorithme proposé par T. Kohonen :

```
1. Initialiser :
    - les poids W_{i,j},
```

- Le coefficient d'apprentissage (η =1),
- le voisinage V_i.

2. Evaluer la corrélation,
3. Activer le réseau,
4. Déterminer le neurone i_o dont la sortie (w_i) est maximale,
5. Mettre à jour les poids :
   $W_{i,j} = W_{i,j} + \eta\ (C_{io} - W_{i,j})$
   pour chaque V_i,
6. Mettre à jour le coefficient d'apprentissage
   $\eta = \eta - 0.001\ \Delta\ t$
7. Aller à 2

Le concepteur de l'application peut choisir plusieurs méthodes pour initialiser les poids initiaux ($W_{i,j}$) du graphe. Il peut :

- choisir des valeurs par défaut (0 par exemple),
- ou définir les valeurs.

4. La plate-forme DIMA

4.1. Mise en œuvre de l'architecture par la plate-forme DIMA

DIMA est un environnement de développement de systèmes multi-agents dont le développement a débuté en 1993. La première version de DIMA a été implémentée en Smalltalk-80 et a été ensuite portée en JAVA (voir http://www-poleia.lip6.fr/~guessoum/dima.html).

DIMA est principalement caractérisée par son architecture d'agents modulaire qui permet de dépasser la dichotomie classique en offrant la possibilité de développer des applications multi-agents dont la granularité des agents est variable. DIMA offre en effet des bibliothèques (classes JAVA) offrant les briques de base pour construire des modèles d'agents divers. Ces différentes briques ont pour but d'offrir à l'utilisateur une grande variété de paradigmes (par exemple automate, règle de production, etc.) d'une part, et d'autre part, une implémentation des différentes propositions conceptuelles introduites par la communauté multi-agents (BDI, KQML, ACL, etc.). Les librairies de DIMA regroupent des classes qui peuvent être réutilisées et/ou adaptées pour construire facilement des agents. Ces classes peuvent être instanciées ou sous-classées pour implémenter un comportement.

DIMA offre également un ensemble de frameworks (sous forme de packages) pour les différents paradigmes (règles, ATN, classeurs, ...). Par exemple, un framework (l'ensemble de classes du package Gdima.tools.automata) d'ATN est proposé. Pour implémenter un ATN avec ce framework, il suffit de spécifier les différents états et les différentes transitions.

Pour la distribution des systèmes multi-agents [VER01], DIMA se base sur le framework Comet [PES 99]. Ce dernier fournit un support logiciel en JAVA à un protocole, centralisé ou non, pour une exécution distribuée tout en garantissant, en s'appuyant sur TCP, des communications sûres (pas de perte de messages), respectant l'ordonnancement des messages entre deux composants (deux messages envoyés successivement entre deux composants arrivent dans le même ordre) et un mode de communication asynchrone entre les entités du système.

DIMA a été utilisée pour développer plusieurs applications réelles (Ventilation artificielle [Gue 96], simulation des modèles économiques [Dur 99], simulation d'un marché électrique, etc.) par les auteurs mais également par d'autres personnes n'ayant pas participé à son développement. Ces applications peuvent être des simulations, des résolutions de problèmes ou des systèmes de contrôles ayant éventuellement des contraintes temps réel. Par ailleurs des petits exemples (proies/prédateurs, ventes aux enchères, …) sont proposés pour illustrer l'utilisation des différents modèles.

4.2. Conception d'un système multi-agent

Dans DIMA, un systèmes multi-agents est un ensemble agents (domaine ou moniteur) et un ensemble d'objets représentant le domaine. La modélisation et l'implémentation d'un système multi-agents nécessite :

- la définition du domaine,
- la conception/implémentation des agents du domaine,

et

- la conception/implémentation des agents moniteurs,
- la conception/implémentation des connaissances organisationnelles.

Pour les systèmes multi-agents adaptatifs.

La modélisation du domaine est similaire aux processus d'élaboration de modèles conceptuels dans les méthodes objets. Son implémentation nécessite la définition d'un ensemble de classes JAVA.

Les principales étapes pour la conception et l'implémentation d'un agent sont les suivantes :

- Détermination de l'ensemble de ses compétences,
- Détermination du type de l'agent : réactif ou adaptatif. Si l'ensemble des stimuli auxquels l'agent doit réagir sont connus à la conception ainsi que l'ensemble des règles de réaction (*Si stimulus et état alors action),* l'agents est réactif, sinon ils est adaptatif.
- Choix de la classe (voir Figues 6 pour les agents réactifs et Figure 7 pour les agents adaptatifs),
- Implémentation des ses compétences, de son comportement et éventuellement son méta-comportement,

– Implémentation de l'agent en instanciant ou en sous classant la classe choisie à l'étape précédente,

Après avoir modéliser le domaine et l'ensemble des agents,

– Initialisation du réseau de communication (accointances),
– Activation des agents.

Les agents moniteurs ont le même modèle que les agents du domaine. Ce sont des agents réactifs dont le comportement consiste à :

– observer les événements,
– analyser les événements,
– interagir avec les autres agents moniteurs.

Leurs compétences sont souvent liées au domaine d'application. La conception et l'implémentation d'un système multi-agent adaptatif nécessitent la conception et l'implémentation de ces trois composantes contrairement aux systèmes non adaptatifs qui n'ont qu'une composante (l'organisation d'agents du domaine). Pour pouvoir implémenter facilement ces systèmes et profiter de leurs caractéristiques, il est en effet important d'avoir des architectures d'agents simples et faciles à utiliser qui permettent de définir différents types d'agents. Pour cela, nous nous appuyons sur le modèle d'architecture modulaire de DIMA.

4.3. Vers un environnement de développement

La richesse des bibliothèques proposées par DIMA permet aux utilisateurs de développer des applications à base de systèmes multi-agents de façon intuitive en se basant sur un modèle de conception solide. De nombreux chercheurs ont contribué à l'élaboration du noyau de base, et également des bibliothèques de la plate-forme tout en essayant de conserver la « philosophie » DIMA : permettre la conception incrémentale d'agents en partant d'un modèle relativement simple. En cela, DIMA peut être vue comme un environnement de conception de système multi-agents simple à prendre en main (le point de départ étant un modèle d'agent minimal) mais pouvant se révéler très puissant via l'exploitation des différents modèles d'agents et/ou des options fournies par la plate-forme (bibliothèques de composants, architectures d'agents, modèles de communications, etc.).

Pour améliorer le processus de développement de systèmes multi-agents avec la plate-forme DIMA, un outil de méta-modélisation est en cours de développement (une exemple est décrit dans [Les 99]). L'intérêt de l'approche par méta-modélisation provient notamment de la diversité des paradigmes en multi-agents (différents modèles d'agents, différents modèles organisationnel, ...) qu'il convient de combiner. Cette approche permettra de définir une méthodologie intégrant les différentes étapes du cycle de base de génie logiciel. Son avantage est qu'elle est basée sur une plate-forme opérationnelle.

5. Conclusion

Aujourd'hui, l'intérêt d'une méthodologie de conception et d'implémentation de systèmes multi-agents n'est plus à démontrer. De nombreux travaux de recherche continuent à explorer les possibilités que fournirait une méthodologie [Dem 98] [Dro 98] [Fer 98] [Woo 00]. Toutefois, la plupart de ces propositions se basent sur une architecture d'agent purement 'conceptuelle', l'idée n'étant pas de fournir un moyen à partir d'une spécification d'un problème donné, de déduire une solution opérationnelle, mais de se restreindre à l'agentification du problème.

Nous pensons que ces approches, aussi intéressantes soient-elles, limitent leurs potentialités (au sens utilisation potentielle) via l'esquisse d'une trop grande généricité (qui a d'ailleurs beaucoup de mal à être atteinte). Proposer une méthodologie de conception de systèmes multi-agents générique (i.e. qui permet de potentiellement agentifier un problème posé, sans se soucier de sa partie opérationnelle) est louable à de nombreux points de vue. Mais cette généricité apporte plus d'inconvénients, qu'elle ne propose de solutions. Ce type de méthodologie force à proposer des solutions dans lesquelles les hypothèses d'implémentation restent trop vagues, et où peu de réelles caractéristiques des entités logicielles générées sont clairement spécifiées.

L'approche que nous proposons ici peut être qualifiée de *bottom-up* : plutôt que de partir sur un modèle d'agent générique couvrant l'ensemble des domaines mais restant hautement conceptuelle, nous avons commencé par proposer une première solution opérationnelle minimale. Via cette proposition et sa confrontation avec un ensemble d'utilisateurs et de chercheurs, le modèle s'est affiné afin de répondre aux besoins spécifiques de différents domaines d'applications, tout en conservant une cohérence globale. Les bibliothèques et frameworks composants DIMA en font un environnement de conception de systèmes multi-agents riche et des solutions opérationnelles associées cohérentes.

6. Bibliographie

[Bra 87] C. Braganza and L. Gasser. *MACE Multi-Agent Computing Environment, Version 6.0.* Release Note 1.0, Technical Report CRI 87-16, University of Southern, California, Etats Unis, .

[Bri 89] J. P. Briot. *Actalk: a testbed for classifying and designing actor languages in the smalltalk-80 environment*. In Proc. of ECOOP'89, volume LNAI 1069, pages 109--129, Nottingham, 1989.

[Gas 92] L. Gasser and J.-P. Briot, *Object-Oriented Concurrent Programming and Distributed Artificial Intelligence*, in N. A. Avouris and L. Gasser (eds.), Distributed Artificial Intelligence: Theory and Praxis, Kluwer Academic Publisher, pp. 81-108, 1992.

[Bro 86] R. A. Brooks. *A Robust layered control system for mobile robot.* IEEE Journal of Robotics and Automation, volume RA-2 (1), April 1986.

[Car 00] A. Cardon, Z. Guessoum. *Système multi-agents adaptatifs*. JFIADSMA'2000, pp. 99-101, oct. 2000.

[Cas 94] C. Castelfranchi. *Guarantees for autonomy in cognitive agent architecture*. In M. Wooldridge and N.R. Jennings, editors, Intelligent Agents: Theories, Architectures and Languages (LNAI Volume 890), pp. 56-70, Springer Verlag, Heidelberg, Germany, 1994.

[Cas 98] C. Castelfranchi, M*odelling Social Action for AI agents*, Artificial Intelligence, vol. 103, pp.157-182, 1998.

[Dem 90] Y. Demazeau, J-P Müller. *Decentralized Artificial Intelligence (1)*. Yves Demazeau and Jean-Pierre Müller (Eds.), Elsevier Science Publisher B.V. (North-Holland), pp. 3-16, 1990.

[Dem 98] Y. Demazeau, *Multi-Agent Systems Methodology*, 14th Brazilian Symposium on Artificial Intelligence, SBIA'98, Porto Alegre, November 1998.

[Dur 99] R. Durand, A. Cool and Z. Guessoum, *Resource accumulation and subtainability of competitive advantage simulation and empirical analysis*, Academy of management Conference, August 1999, Chicago.

[Fer 92] J. Ferber and A. Drogoul. *Using Reactive Multi-Agent Systems in Simulation and Problem Solving*. in Distributed Artificial Intelligence: Theory and Praxis, N. M. Avouris and L. Gasser (eds.), Kluwer Academic Publishers, London, 1992.

[Fer 95] I. A. Ferguson. *On supporting rational behavior in real-time multi-agent domains*. Proc. of AAAI Full Symposium on Rational Agency: concepts, theories, models and applications, Cambridge, MA, November , pp. 61-65, 1995.

[Fer 98] J. Ferber and O. Gutknecht, *Alaadin: a meta-model for the analysis and design of organizations in mutli-agent systems*, ICMAS'98, Y. Demazeau (ed.), pp. 128-135, Paris, 1998.

[FIP97] Foundation for Intelligent Physical Agents. FIPA 97 Specification. Part 2, Agent Communication Language. http://www.fipa.org, 1997.

[Gas 90] L. Gasser, *Conceptual Modeling in Distributed Artificial Intelligence*, Journal of the Japanese Society of Artificial Intelligence, vol. 5, 1990.

[Gue 96] Guessoum Z., *Un environnement de développement de systèmes multi-agents*, Thèse d'université, Paris 6, 1996.

[Gcr 99] Guessoum Z., Cardon A., Ramdani A., *Toward self-adaptive multi-agent systems*, Maamaw'97, Valencia, juin 1999. (Poster)

[Gue 99] Guessoum Z. and Briot J.P., From Active Objects to Autonomous Agents, IEEE Concurrency. 7(3): 68-76, November 1999.

[Gue 01] Z. Guessoum et M. Occello. *Environnements de développement*. In "Principes et architecture des systèmes multi-agents", Jean-Pierre Briot et Yves Demazeau (eds.), Collection IC2. Hermès Science Publications, Paris, France, automne 2001 à paraître.

[Hay 95] B. Hayes-Roth. *An architecture for adaptive intelligent systems*. Artificial Intelligence, 72(1-2), pp. 329-365, January 1995.

[Jen 92] N. R. Jennings. Specification and implementation of a belief desire joint-intention architecture for collaborative problem solving. Proc. of the Tenth National Conference of AI, San Joe, California. pp. 269-275, 1992.

[Koh] Kohonen T., *The self-organizing map*, Proceedings of the IEEE, vol. 78, n° 9, p. 1464 - 1480, 1990.

[Lac 00] G. Lacotte, J.-P. Briot, Z. Guessoum and P. Sens. *Towards Fault-Tolerant Agents*. ECOOP'2000 Workshop on Distributed Objects Programming Paradigms, Cannes, juin 2000.

[Les 99] Bruno Lesueur, Gerson Sunyé, Z. Guessoum, G. Blain et Jean-François Perrot. *La métaphore du dossier*. In Proc. Inforsid'99. pp. 279-299. La Garde, June 1999.

[Mar 01] O. Marin, P. Sens, J.-P. Briot and Z. Guessoum, Towards Adaptive Fault-Tolerance For Distributed Multiagent Systems, in *Proceedings of ERSADS'2001*, Bertinoro, Italy, pp. 195-201, May 2001.

[Mar 90] T. Maruichi, M. Ichikawa and M. Tokoro. *Modeling Autonomous Agents and their Groups*. Yves Demazeau and Jean Pierre Müller (Eds.), Elsevier Science Publisher B.V. (North-Holland), pp. 215-234, 1990.

[Meu 01] T. Meurisse and J.-P. Briot, *Une approche à base de composants pour la conception d'agents.* , TSI vol. 20 n°4, Numéro thématique Réutilisabilité, 2001.

[Mül 94] J-P. Müller and M. Pischel. *Modelling reactive behaviour in vertically layered agent architectures*. Proc. of ECAI'94, Amsterdam, (NL), pp. 709-713, 1994.

[Ode 98] J. Odell , *Agents and Beyond : a Flock is not a Bird*, Distributed computing, April 1998.

[Ode 99] J. Odell , *Objects and Agents : Is There Room for Both ?*, Distributed computing, November 1999.

[Pes 99] F. Peschanski. *COMET : A Component-based Reflective Architecture for Distributed Programming*, First OOPSLA Workshop on Object-Orientation for Software Engineering, Denver, USA, November 1999

[Pit 90] J. Pitrat, An intelligent system must and can observe it's own behavior, Cognitiva, pp.119-128, 1990.

[Rao 91] R. Rao, *From the Board Notion of Reflection to the Engineering Practice of Object-Oriented Metalevel Architectures*, in Proc. OOPSLA-91 Workshop on Reflection and Metalevel Architectures, pp. 27-31, 1991.

[Ver 01] O. Vernin. *Déploiement de systèmes multi-agents*. Rapport de stage de maîtrise. A paraître comme rapport interne LIP6.

[Wei 99] G. Weiß and P. Dillenbourg, *What is 'multi' in multiagent learning ?*, In P. Dillenbourg (Ed.), Collaborative learning. Cognitive and computational approaches (Chapter 4, pp. 64--80). Pergamon Press. 1999.

[Woo 00] M. Wooldridge N. R. Jennings, D. Kenny, *Gaia methodology for agent-oriented analysis and design*, *jaamas2000*, Kluwer academic publishers, Boston, pp. 1-27.