# IAD ET SYSTÈME MULTI-AGENTS

## 2.1 Introduction

L'Intelligence Artificielle Distribuée est au carrefour de plusieurs disciplines, telles que la psychologie cognitive, la sociologie, la biologie, et l'informatique. Ces études sont utilisées entre autre pour modéliser et construire des systèmes informatiques dans lesquels les capacités de traitement, de représentation et de raisonnement sont distribuées dans un ensemble de sous systèmes appelés agents. L'ensemble de ces agents constitue une société appelée Système Multi-Agents.
Dans ce chapitre, nous présentons un aperçu de ce que sont les systèmes d'intelligence artificielle distribuée et multi-agents.

## 2.2 Intelligence artificielle distribuée

L'I.A.D est une branche de l'I.A qui s'intéresse à la modélisation de comportement intelligent par la coopération d'un ensemble d'entités autonomes (ou semi autonomes), qui interagissent entre elles et avec leur environnement. [14]
L'I.A.D recouvre en réalité trois axes fondamentaux de recherche :
**La résolution distribuée des problèmes (RDP) :** elle s'intéresse à la manière de diviser un problème particulier sur un ensemble d'entités distribuées et coopérantes. Elle s'intéresse aussi à la manière de partager la connaissance du problème et d'en obtenir la solution. [19]
**L'intelligence artificielle parallèle(IAP) :** elle concerne le développement de langages et d'algorithmes parallèles pour l'I.A. L'intelligence artificielle parallèle vise l'amélioration des performances des systèmes d'intelligence artificielle sans, toutefois, s'intéresser à la nature du raisonnement ou au comportement intelligent d'un groupe d'agents. Cependant, le développement de langages concurrents et d'architectures parallèles peut avoir un impact important sur les systèmes d'I.A. [19]
**Les systèmes multi-agents (SMA) :** il s'agit de faire coopérer un ensemble d'entités, appelées "agent", dotées d'un comportement intelligent et de coordonner leurs buts et leurs plans d'action pour la résolution d'un problème. [19]

## 2.2.1 Raisonnement semi-distribué

Dans un système où le raisonnement est distribué les MCs (Modules de connaissances) ne communiquent pas directement entre eux mais par l'intermédiaire d'une structure de données partagé.

### 2.2.1.1 La méthode du tableau noir :

**a) L'origine des systèmes de tableau noir :**
Le premier domaine dans lequel on a utilisé le modèle de tableau noir est celui de la reconnaissance vocale. Les travaux réalisés dans ce domaine à Carnegie-Mellon University ont aboutit au système Hearsay. D'après Carver et Lesser, ce sont les difficultés du problème traité et sa nature même (la reconnaissance vocale) qui ont conduit à développer une architecture de tableau noir. En effet, ce type d'application est caractérisé par la taille importante de l'espace de recherche, le caractère incomplet et/ou erroné des données d'entrée, ainsi que le caractère imprécis et/ou incomplet de la connaissance utile à la résolution du problème. Il est aussi caractérisé par la nécessité de faire intervenir des agents bien identifiés : acoustique, phonétique, lexical, syntaxique, etc .. Ces caractéristiques imposent un modèle de résolution de problème qui permette le développement incrémentaI de la solution ainsi que l'utilisation de différentes expertises, et qui puisse adapter la stratégie de résolution en fonction du cas particulier traité. La flexibilité dans la structuration du processus de résolution, le développement incrémentaI de la solution du problème, et la modularité sont des caractéristiques importantes du modèle de tableau noir. [2]
**b) Principe de tableau noir :**
Le tableau noir est le nom donné à la base de faits globale partagée par les MC(s)(Modules de connaissances). Il est accessible par un protocole uniforme et systématique. Il contient l'état courant du problème et permet ainsi d'avoir une cohérence de résolution. Les éléments d'information qu'il contient sont appelés des hypothèses ou des unités. [13]
Chaque MC a accès au tableau noir en lecture ou en écriture, pour ajouter, modifier ou supprimer des hypothèse. La communication et les interactions entre MC se font exclusivement par l'intermédiaire de cette structure de données. [13]
Les MC sont totalement indépendants dans le sens où ils ne s'appellent pas direc-

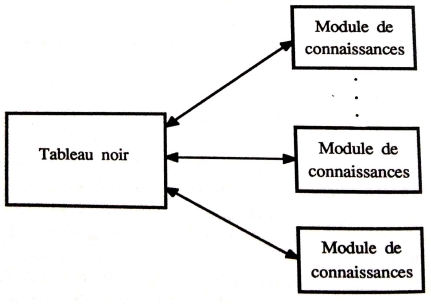


FIGURE 2.1 – Communication par tableau noir général

tement (chacun ignore la présence des autres dans le système) et où l'intervention d'un MC dans la résolution est provoquée par une modification dans le tableau noir. Ainsi, la communication se fait de manière implicite. [13]
Cette méthode de communication schématisée Figure2.1, impose aux experts concepteurs des différents MC d'utiliser le même langage et donc de s'exprimer avec le même vocabulaire. Cette contrainte peut entraîner un appauvrissement de leur langage dans l'expression de leurs connaissances et la conception du SME(systèmes multi-experts) avec les experts en est fortement compliquée. Les interactions entre les experts humains sont donc nombreuses, ce qui nuit à la modularité du système. De plus, dans certains domaines d'application tels que la compréhension de la parole, un même objet traité à différents niveaux peut avoir plusieurs représentations. [13]
Pour pallier cela, le tableau noir peut être partitionné en niveaux, comme cela est représentée Figure 2.2. Chaque niveau spécifie une représentation différente de l'espace du problème. La suite des niveaux forme une structure hiérarchique où les éléments de chaque niveau peuvent être approximativement considérés comme des abstractions des éléments du niveau inférieur. Cette décomposition peut être vue à priori, comme la structure d'un plan pour résoudre le problème. Chaque niveau est une étape générique dans le plan. Dans cette configuration, les MC travaillent sur un ou deux niveaux adjacents. [13]
Un des inconvénients de cette méthode de communication est le temps d'accès aux

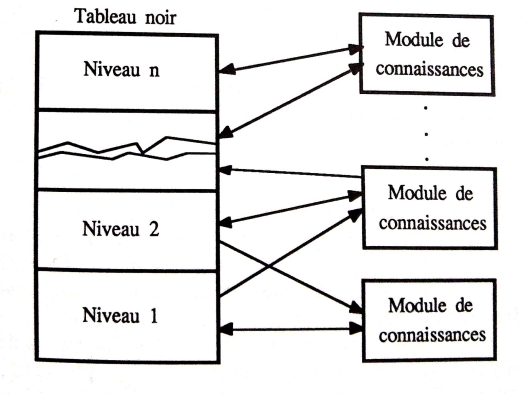


FIGURE 2.2 – Communication par tableau noir partitionné en niveaux

informations qui peut devenir prohibitif. La structure de données globale présente aussi le risque d'accumuler des données inutiles du fait de l'accès libre à toutes les données que le tableau noir contient. Par conséquent, la recherche d'informations prend de plus en plus de temps au fur et à mesure que le tableau noir s'enrichit. [13]
Le partage d'informations dans une structure de données globales s'apparente à celui qui a lieu dans une base de faits d'un SE dans lequel chaque granule de connaissance (règle) va puiser la donnée qui l'intéresse. L'activation d'un MC correspond, dans cette analogie, à la phase de restriction d'un paquet de règles pour éviter de raisonner avec de trop nombreuses connaissances. Vu les contraintes de cette méthode, elle ne peut réellement se justifier que si les interactions inter-modules sont très fortes, ce qui entraîne une fréquence de communication très élevée. [13]

### 2.2.2 Le raisonnement totalement distribué

La distribution du contrôle parmis un ensemble d'agents induit la nécessité d'un comportement globalement cohérent et finalisé. Le système doit se comporter comme un tout. Pour cela, il est nécessaire de définir les stratégies permettant la conduite de la résolution. Elles permettent au système d'aboutir à un résultat et lui assurent un comportement global cohérent [20].
À l'inverse des systèmes à tableau noir, les systèmes d'acteurs relèvent d'une distribution totale des connaissances et du contrôle. Ces systèmes sont caractérisés par un traitement local c'est-à-dire un agent ne peut manipuler que sa base de connaissances locale, envoyer des messages aux autres qu'il connaît (accointance). De ce fait, les connaissances et le contrôle sont distribués entre les agents, qui effectuent des tâches en parallèle et de manière indépendante.
**Le mécanisme de contrôle :** La distribution du contrôle entre un ensemble d'agents induit la nécessité d'un comportement globalement cohérent et finalisé. Le système doit se comporter comme un tout. Pour cela, il est nécessaire de définir les stratégies permettant la conduite de la résolution. Elles permettant au système d'aboutir à un résultat et lui assurent un comportement global cohérent [20]. En effet l'efficacité de la distribution du contrôle est en fonction des solutions apportées.

## 2.3 Les systèmes multi-agents(SMA)

Selon Ferber, un agent est une entité réelle ou virtuelle, évoluant dans un environnement, capable de le percevoir et d'agir dessus, qui peut communiquer avec d'autres agents, qui exhibe un comportement autonome, lequel peut être vu comme la conséquence de ses connaissances, de ses interactions avec d'autres agents et des buts qu'il poursuit. Un système multi-agents (SMA) et une organisation d'agents (société). [8]
Selon Demazeau, un agent est une entité réelle ou virtuelle, dont le comportement est autonome, évoluant dans un environnement, qu'il est capable de percevoir, sur lequel il est capable d'agir et d'interagir avec les autres agents. [15]
Selon Wooldridge, un agent est un système informatique capable d'agir de manière autonome et flexible dans un environnement. Flexibilité signifie réactivité, pro-activité, adaptativité et capacités sociales. [15]
Selon Wooldridge, Jennings et Sykara : "Un agent est un système informatique situé dans un certain environnement et qui est capable d'agir de manière autonome et flexible dans cet environnement afin d'atteindre ses objectifs de conception ". [15]

**Définition formelle de l'agent :** Qu'est ce qu'un agent ?
Comme dans tous les domaines porteurs, le terme agent est utilisé de manière assez vague. Cependant on peut dégager une définition minimale commune qui est approximativement la suivante [8] :
On appelle agent une entité physique ou virtuelle
a. qui est capable d'agir dans un environnement,
b. qui peut communiquer directement avec d'autres agents,
c. qui est mue par un ensemble de tendances (sous la forme d'objectifs individuels ou

d'une fonction de satisfaction, voire de survie, qu'elle cherche à optimiser),
d. qui possède des ressources propres,
e. qui est capable de percevoir (mais de manière limitée) son environnement,
f. qui ne dispose que d'une représentation partielle de cet environnement (et éventuellement aucune),
g. qui possède des compétences et offre des services,
h. qui peut éventuellement se reproduire,
i. dont le comportement tend à satisfaire ses objectifs, en tenant compte des ressources et des compétences dont elle dispose, et en fonction de sa perception, de ses représentations et des communications qu'elle reçoit.

Chacun des termes de cette définition est important. Une entité physique est quelque chose qui agit dans le monde réel : un robot, un avion ou une voiture sont des exemples d'entités physiques. En revanche, un composant logiciel, un module informatique sont des entités virtuelles, car elles n'existent pas physiquement. Les agents sont capables d'agir, et non pas seulement de raisonner comme dans les systèmes d'IA classique. L'action, qui est un concept fondamental pour les systèmes multi-agents, repose sur le fait que les agents accomplissent des actions qui vont modifier l'environnement des agents et donc leurs prises de décision futures. Ils peuvent aussi communiquer entre eux, et c'est d'ailleurs là l'un des modes principaux d'interaction existant entre les agents. Ils agissent dans un environnement, sauf, comme nous le verrons, pour les agents purement communicants pour lesquels toutes les actions se résument à des communications. [8]

Les agents sont doués d'autonomie. Cela signifie qu'ils ne sont pas dirigés par des commandes venant de l'utilisateur (ou d'un autre agent), mais par un ensemble de tendances qui peuvent prendre la forme de buts individuels à satisfaire ou de fonctions de satisfaction ou de survie que l'agent cherche à optimiser. On pourrait dire ainsi que le moteur d'un agent, c'est lui-même. C'est lui qui est actif. Il a la possibilité de répondre par l'affirmative ou le refus à des requêtes provenant des autres agents. Il dispose donc d'une certaine liberté de manœuvre, ce qui le différencie de tous les concepts semblables, qu'ils s'appellent "objets", "modules logiciels" ou "processus". Mais l'autonomie n'est pas seulement comportementale, elle porte aussi sur les ressources. Pour agir, l'agent a besoin d'un certain nombre de ressources : énergie, CPU(Central processing unit), quantité de mémoire, accès à certaines sources d'informations, etc. Ces ressources sont à la fois ce qui rend l'agent non seulement dépendant de son environnement, mais aussi, en étant capable de gérer ces ressources, ce qui lui donne une certaine indépendance vis-à-vis de lui. L'agent est ainsi à la fois un système ouvert (il a besoin d'éléments qui lui sont extérieurs pour survivre) et un système fermé (car les échanges qu'il a avec l'extérieur sont très étroitement réglementés). [8]

Les agents n'ont qu'une représentation partielle de leur environnement, c'est-à-dire qu'ils n'ont pas de vision globale de tout ce qui se passe. C'est d'ailleurs ce qui se passe dans les réalisations humaines d'envergure (la fabrication d'un Airbus par exemple) dans lesquelles personne ne connaıt tous les détails de la réalisation, chaque spécialiste n'ayant qu'une vue partielle correspondant à son domaine de compétence. [8]

L'agent est ainsi une sorte "d'organisme vivant" dont le comportement, qui se résume à communiquer, à agir et, éventuellement, à se reproduire, vise à la satisfaction de ses besoins et de ses objectifs à partir de tous les autres éléments (perceptions, représentations, actions, communications et ressources) dont il dispose. [8]

**Définition d'un système multi-agents SMA :**
On appelle système multi-agent (ou SMA), un système composé des éléments suivants [3] :
1. Environnement E : espace disposant souvent d'une métrique.
2. Ensemble d'objets situés O : objets passifs pouvant être perçus, créés, détruits et modifiés par les agents (possibilité à un moment donné d'associer une position dans E).
3. Ensemble d'agents A : les entités actives du système.
4. Ensemble de relations R : unissant les objets et les agents.
5. Ensemble d'opérations Op : permettant aux agents A de percevoir, produire, consommer, transformer et manipuler des O

Lorsque les agents sont situés, E est généralement un espace métrique, et les agents sont capables de percevoir leur environnement, c'est-à-dire de reconnaître les objets situés dans l'environnement en fonction de leurs capacités perceptives, et d'agir, c'est-à-dire de transformer l'état du système en modifiant les positions et les relations existant entre les objets. [3]

### 2.3.1 Domaines d'applications :

Les domaines d'application des systèmes multi-agents sont particulièrement riches. Nous en citerons seulement les principales directions, toute recherche d'exhaustivité aboutissant à une sclérose a priori d'un domaine de recherche en pleine évolution. On peut considérer qu'il existe cinq grandes catégories d'applications des systèmes multi-agents : la résolution de problèmes au sens large, la robotique distribuée, la simulation multi-agents, la construction de mondes hypothétiques et la conception kénétique de programmes (Figure 2.3). [8]

### 2.3.2 La résolution des problèmes par des SMA :

La résolution de problèmes au sens large concerne en fait toutes les situations dans lesquelles des agents logiciels accomplissent des tâches utiles aux êtres humains. Cette catégorie s'oppose aux applications de robotique distribuée en ce sens que les agents sont purement informatiques et n'ont pas de structure physique réelle. [8]

#### La résolution distribuée de problèmes

Toutes les applications relevant de la résolution distribuée de problèmes supposent qu'il est possible d'effectuer une tâche complexe en faisant appel à un ensemble de spécialistes disposant de compétences complémentaires et donc, lorsque c'est l'expertise ou le mode de résolution qui sont distribués, sans que le domaine le soit. Lorsque l'expertise est en effet si vaste et complexe qu'elle ne peut appartenir qu'à une seule personne, il faut faire

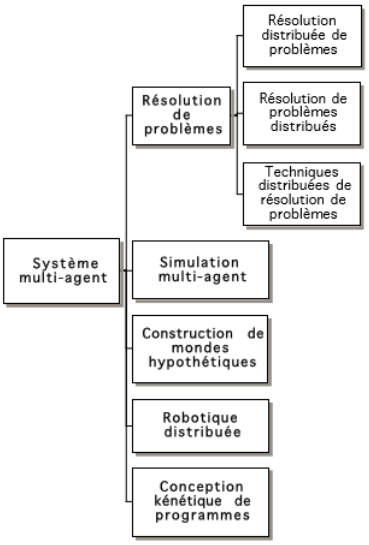


FIGURE 2.3 – Une classification des différents types d'application des systèmes multi-agents

appel à plusieurs spécialistes qui doivent travailler ensemble à la poursuite d'un objectif commun. [8]

**Résolution distribuée de problèmes distribués**

Si le domaine est lui aussi distribué, on parle alors de résolution (distribuée) de problèmes distribués. Il s'agit essentiellement d'applications telles que l'analyse, l'identification, le diagnostic et la commande de systèmes physiquement répartis, pour lesquelles il est difficile d'avoir une vision totalement centralisée. Par exemple, s'il s'agit de contrôler un réseau de communication ou d'énergie, le domaine, qui est représenté par le réseau lui-même, constitue un système réparti qu'il s'agit de surveiller, voir de superviser, en décentralisant au maximum les tâches de surveillance au sein des nœuds du réseau. La perception distribuée, telle qu'elle a été initiée par DVMT(Distributed Vehicle Monitoring Test), ou la surveillance de réseaux d'énergie ou de télécommunications dans lesquels la supervision est répartie sur chacun des nœuds constitue un bon exemple de résolution de problèmes distribués. [8]

**Résolution par coordination**

Les agents peuvent aussi servir d'une manière beaucoup plus élémentaire à résoudre des problèmes au sens classique du terme, c'est-à-dire à tenter de trouver une solution à quelque chose dont l'énoncé est bien posé et dont l'ensemble des informations est entièrement disponible, comme, par exemple, trouver une affectation de tâches pour une machine-outil ou définir un emploi du temps pour un collège, donner la suite des actions à accomplir pour sortir d'un labyrinthe ou pour faire partir une fusée, assembler des cubes ou des composants mécaniques, résoudre un casse-tête ou démontrer un théorème. Dans ce cas, le domaine n'est pas distribué et l'expertise ne l'est pas non plus. Et pourtant l'approche multi-agent peut apporter un mode de raisonnement nouveau en décomposant le

problème de manière totalement différente. Par exemple, s'il s'agit d'empiler des cubes ou d'assembler des pièces mécaniques, on peut considérer que les cubes ou les pièces sont des agents qui doivent satisfaire des buts précis donnés par le plan imposé par le concepteur et que les liaisons sont des contraintes que les agents doivent respecter. [8]

### 2.3.3 Différents types d'agents

Les systèmes multi-agents sont en général classés en deux principales familles : les systèmes cognitifs et les systèmes réactifs. Contrairement aux systèmes réactifs, les systèmes cognitifs se rapprochent le plus du modèle de sociétés d'experts. Chaque agent cognitif a une représentation explicite de croyances, d'intentions, d'actes de langages, de modèles des autres agents, etc.. [16]

#### 2.3.3.1 Agent cognitif vs réactif

Les systèmes multi-agents font la distinction entre « agents cognitifs » et « agents réactifs » : les agents cognitifs disposent d'une base de connaissances comprenant les diverses informations liées à leurs domaines d'expertise et à la gestion des interactions avec les autres agents et leur environnement. Les agents sont généralement « intentionnels » c'est-à-dire qu'ils possèdent des buts et des plans explicites leur permettant d'accomplir leurs buts. Dans ce cadre, comme le précise J. Ferber, les problèmes de coopération ressemblent étonnamment à ceux de petits groupes d'individus, qui doivent coordonner leur activité, et sont parfois amenés à négocier pour résoudre leurs conflits. [16]

Les agents réactifs au contraire ne sont pas « intelligents » pris individuellement. Ils ne peuvent que réagir à des stimuli simples provenant de leur environnement, et leur comportement est alors simplement dicté par leur relation à leur entourage sans que ces agents ne disposent d'une représentation des autres agents ou de leur environnement. Cependant, du fait, de leur nombre, ces agents réactifs peuvent résoudre des problèmes qualifiés de complexes (tab 2.1 qui résume ces différences). Les travaux sur ces agents s'intéressent plus à la modélisation d'une société d'agents qu'à l'agent lui-même. Les analogies que les chercheurs ont établi sont celles de la vie artificielle, de l'éthologie (la fourmilière, la termitière, la ruche d'abeille), etc.. [16]

| AGENTS COGNITIFS | AGENTS REACTIFS |
|---|---|
| Représentation explicite de l'environnement | Pas de représentation explicite |
| Peut tenir complexes de son passé | Pas de mémoire locale |
| Agents complexes | Fonctionnement stimulus/action |
| Nombre d'agents réduit | Nombre d'agents élevé |

Table 2.1 – Les agents cognitifs vs réactifs

#### 2.3.3.2 Agent hybride

En général, la différence entre des agents réactifs et des agents cognitifs peut être expliquée par le compromis efficacité/ complexité. La complexité des systèmes réactifs exige le développement de nouvelles théories dans le domaine de la coopération, de la communication et de la compréhension de nouveaux phénomènes telles que l'émergence. Toutefois, il est maintenant possible de concevoir des systèmes hétérogènes comportant les deux types de comportements (cognitif et réactif) : on parlera alors d'agents hybrides. [16]

Dans ce sens, nous pouvons citer les travaux de [Ferguson, 92 ; Muller & Pischel, 94 ; Bussman & Demazeau, 94]. La majorité des modèles d'agents hybrides présentés par ces auteurs, propose de décomposer chaque agent en différents modules réactifs et cognitifs avec un module spécifique qui contrôle l'activation des autres modules. Cette approche est intéressante et semble apporter une solution adéquate pour modéliser les systèmes complexes dont l'environnement est dynamique, mais elle ne résout pas clairement le problème d'interaction entre les différents modules. Le problème, pour M.J. Wooldridge et N.R. Jennings [Wooldridge & Jennings, 95], est alors de définir les mécanismes et les stratégies du module de contrôle interne de l'agent pour soit gérer les interactions entre ses différents modules, soit imposer un séquencement temporel global interne à l'agent. [16]

### 2.3.4 Communication des SMA

La communication désigne l'ensemble des processus physiques et psychologiques par lesquels s'effectue l'opération de mise en relation d'un émetteur avec un ou plusieurs récepteurs, dans l'intention d'atteindre certains objectifs [Anzieu & Martin, 68]. Elle est considérée comme une forme d'action particulière qui, au lieu de s'appliquer à la transformation de l'environnement, agit sur (modifie) les représentations mentales des agents(buts, croyances, etc..). Comme le confirme J. Erceau et J. Ferber, la communication dans l'univers multi-agents n'est plus une simple tâche d'entrée-sortie, mais doit être modélisée comme un acte pouvant influer sur l'état des autres agents. [16]

De ce fait, les processus physiques désignent les mécanismes d'exécution des actions (l'envoi et la réception de messages), les processus psychologiques se rapportent aux transformations opérées par les communications sur les buts et les croyances des agents. [16]

Au sens de Ferber, la communication est un moyen ou une méthode de coopération (d'interaction), à côté de la :
- collaboration qui s'intéresse à la manière de répartir le travail et les ressources entre plusieurs agents ;
- coordination d'actions qui analyse la manière dont les actions des différents agents doivent être organisées dans le temps et l'espace de manière à atteindre les objectifs ;
- résolution de conflit par arbitrage et négociation en établissant par exemple des compromis, etc... [16]

Voyons maintenant, pourquoi les agents communiquent ? quand ? comment ? et avec qui ?

**Pourquoi communiquer ? :** Les agents communiquent et interagissent pour synchroniser leurs actions et pour résoudre des conflits, qui sont des conflits de ressources, de buts ou d'intérêts. Ils communiquent également pour s'aider mutuellement ou, comme le souligne J. Ferber, pour suppléer aux limites de leurs champs de perception. En effet, un agent ne peut être en relation avec tous les autres, ni équipé de tous les capteurs nécessaires à la connaissance de l'environnement. [16]
**Quand et avec qui communiquer ? :** Pour répondre à cette question, il faut identifier les situations qui vont nécessiter la communication des agents. En général, les agents communiquent lorsqu'ils sont face à un problème qu'ils ne savent pas résoudre (soit par manque de compétences ou de ressources), lorsqu'il est nécessaire de coordonner leurs actions, ou encore lorsqu'il y a un conflit entre plusieurs agents et que le conflit ne peut pas être résolu de façon déterministe. Les communications peuvent être diffusées à l'ensemble des agents ou à des agents particuliers (des agents susceptibles d'être intéressés par le message). [16]
**Comment communiquer ? :** Les procédures de communication pour véhiculer les messages (qui sont porteurs d'informations ou d'actions) entre agents sont la communication par envoi de messages, la communication par partage d'informations. [16]
**Communication par partage d'informations :** C'est historiquement, le premier modèle de communication qui est apparu au début des années 60 [Newell, 62]. Le parangon des structures centralisées est le « tableau noir » [Hayes-Roth 85], où la mémoire partagée est vue comme un tableau sur lequel les agents écrivent, trouvent des réponses partielles, des informations. Le tableau noir est divisé en niveaux. Les agents travaillant à un niveau particulier d'abstraction ont accès à un niveau correspondant dans le tableau. Un dispositif de contrôle gère les conflits d'accès au tableau, les agents faisant les demandes d'accès de manière autonome (Figure 2.4). [16]

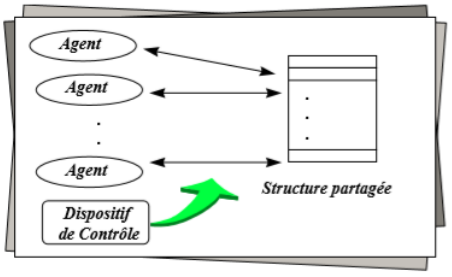


FIGURE 2.4 – Communication par partage d'informations

**Communication par envoi de messages :** Si les agents communiquent par envoi de messages ils se rapprochent du modèle d'acteur proposé par Hewitt [Hewitt, 77]. Ce premier modèle définissait un acteur comme une entité active et autonome qui a une vue partielle de l'univers. Cet acteur était décrit au moyen de deux éléments : des accointances (qui correspondent aux acteurs connus d'un autre acteur), et un comportement décrit par un script, ensemble de méthodes qui indiquent les différentes actions que peut accomplir cet acteur en réponse aux messages qu'il reçoit. L'acteur pourra adopter différents comportements pour répondre à un message : soit il le traite, soit il délègue la

tâche à ses accointances ; il peut également créer de nouveaux acteurs puis disparaître. L'envoi de messages est le seul mode de communication entre acteurs. Tous les messages sont asynchrones et placés dans une mémoire-tampon (sans attente de réponse). Chaque message contient une continuation, à qui doit être retournée la réponse à la requête. Il y a distribution des connaissances (chaque acteur possède un comportement réparti entre ses accointances) et distribution du contrôle (chaque acteur possède un script qui définit sa réaction aux messages qu'il reçoit) [Ouzrout, 96]. [16]

### 2.3.5 Organisation des SMA

Avec le développement des systèmes multi-agents, différents paradigmes organisationnels ont été développés. Ces organisations établissent un cadre pour les relations et interactions entre les agents. Nous allons présenter ici les principales :

**Hiérarchies :** Dans ce modèle, les agents sont hiérarchisés selon la structure d'un arbre, dans lequel chaque nœud représente un agent, et possède un lien d'autorité sur ses nœuds-fils. Ce modèle permet de décomposer la tâche globale du système. [15]

**Holarchies :** L'holarchie se rapproche de la hiérarchie, mais il existe quand même une différence majeure. En effet, il n'y a pas de relation d'autorité entre un agent et son sous-groupe, mais les agents du sous-groupe constituent "physiquement" leur sur-agent. Pour illustrer cette notion, on peut prendre l'exemple d'une ville, constituée de bâtiments. Les bâtiments sont des agents, et la ville est un agent constitué de ces agents bâtiments. On peut également avoir, à l'échelle supérieure, un agent région qui sera constitué d'agents villes. De même, un banc de poissons ressemble parfois à un poisson plus gros que les poissons qui le composent, le banc comme les poissons sont alors des agents, organisés en holarchie. [15]

**Coalitions :** Une coalition est une alliance temporaire d'agents qui s'unissent et collaborent car leurs intérêts individuels se rencontrent. La valeur de la coalition doit être supérieure à la somme des valeurs individuelles des agents la composant. Pour illustrer cette notion, imaginons que nous ayons des agents qui ont chacun besoin d'un gâteau. Le gâteau individuel coûte 5€, et le lot de 6 coûte 24€. Si six agents forment une coalition pour acheter un lot, chacun pourra repartir avec son gâteau pour seulement 4€. La coalition leur a donc permis d'optimiser leurs intérêts individuels. [15]

**Équipes :** Les agents constituant l'équipe travaillent ensemble à la réalisation d'objectifs communs. À la différence des agents d'une coalition, les agents d'une équipe cherchent à maximiser les intérêts de l'équipe plutôt que leurs intérêts personnels. [15]

Congrégations : Les congrégations sont assez similaires aux coalitions et aux équipes. Cependant, elles sont destinées à être permanentes et ont généralement plusieurs objectifs à réaliser. De plus, les agents peuvent entrer et sortir des congrégations, et appartenir à plusieurs congrégations en même temps. [15]

**Sociétés :** La société est un ensemble d'agents variés, qui interagissent et communiquent. Ils possèdent différents objectifs, n'ont pas le même niveau de rationalité, ni les mêmes capacités, mais sont tous soumis à des lois communes (normes). [15]

**Fédérations :** Les agents d'une fédération cèdent une partie de leur autonomie au délégué de leur groupe. Les agents d'un groupe n'interagissent qu'avec leur délégué, qui lui-même interagit avec les délégués des autres groupes. [15]

**Marchés :** Des agents vendeurs proposent des objets à la vente, sur lesquels des agents

acheteurs peuvent enchérir. Ce genre d'organisation permet, par exemple, de simuler des marchés réels et/ou de comparer différentes stratégies de négociation. [15]
**Matrices :** Les agents d'une organisation en matrices sont hiérarchisés. Cependant, à la différence de la hiérarchie présentée plus haut, où un agent n'était soumis qu'à l'autorité d'au plus un seul autre agent, les agents dans une organisation matricielle peuvent être soumis à plusieurs autres agents. [15]
**Combinaisons :** Une organisation combinée mélange plusieurs des styles présentés ci-dessus (ou d'autres qui auraient été oubliés dans cette liste). Cela peut être, par exemple, une fédération de coalitions ou une hiérarchie d'équipes. [15]

## 2.4 Conclusion

Dans ce chapitre, nous avons présenté un aperçu de ce que sont les systèmes d'intelligence artificielle distribuée et multi-agents. Nous insisterons sur des aspects importants pour ces systèmes tels que l'architecture des agents, l'interaction et coopération, la communication, l'organisation, etc...
Dans le chapitre suivant, nous allons présenter les architectures que nous avons proposées et les algorithmes que nous avons choisis, adapté et implémenté pour la réalisation du SEGM.

# CHAPITRE 3

# CONCEPTION

## 3.1 Introduction

Dans les deux chapitres précédents nous avons présenté les systèmes experts, les limites de ces systèmes sont liées à la taille de la base de connaissances et la complexité du raisonnement, plusieurs tentatives d'amélioration de tels systèmes sont proposées dans la littérature, nous avons choisi l'approche multi-agents.
Pour résoudre le problème d'incohérence dans la base de connaissance, nous allons proposer d'intégrer un module de contrôle de cohérence particulièrement, les redondances et les contradictions c'est un SMV qui va accompagner le moteur d'inférence suite aux différents cycles de raisonnement.
Le but de notre travail est de concevoir et réaliser un système expert didacticiel résolveur d'exercices de la géométrie mathématique en montrant l'apport de la distribution du raisonnement pour un système à base de connaissance.
Dans ce chapitre nous allons représenter l'architecture générale de système. On va détailler l'ensemble des outils conceptuels et algorithmiques que nous avons proposé, adapté et utilisé pour réaliser le système proposé.

## 3.2 Base de connaissances du système

Comme nous l'avons déja vu dans le premier chapitre, la base de connaissances d'un système est constituée de deux parties qui sont la base de faits et la base de règles.
**Remarque :**
Nous proposons dans ce travail la conception et la réalisation d'un système à base de connaissances dédié au domaine de la géométrie mathématique et que nous appelons SEGM (Système expert de la géométrie mathématique).

### 3.2.1 La base de faits

La base de faits est composé des prédicat de la logique d'ordre 1.
**Exemples :**
point(a,1,2) a est le nom du point, 1 et 2 sont ses coordonnés, carré(a,b,c,d)

### 3.2.2 Base de règles

Nous avons réalisé une BR(base de règles) en modélisant et en formalisant un ensemble de théorèmes sur la géométrie dans le plan de deux dimensions
**Exemples :**
1 - Trois angles A,B,C de 60° forme un triangle équilatéral.
2 - Un rectangle qui à deux cotés consécutifs égaux est un carrée.
3 - Deux triangles rectangles [ABC] et [ADC] forme un rectangle ABCD.

## 3.3 Compilateur de la base de connaissances

La première étape dans la conception d'un compilateur est de concevoir ensuite générer une grammaire du type 2. Cette grammaire permet à l'expert du domaine de translater les définitions et les théorèmes de la géométrie à des règles de production. La deuxième étape est de réaliser un éditeur de saisi de règles. Avant d'être ajoutée à la base de connaissances une règle de production est compilée.

### 3.3.1 Grammaire géométrique

En informatique, une grammaire est une description d'un ensemble de règles permettant de générer, à partir d'un ensemble de symboles, les suites ordonnées des symboles (chaînes) constituant les phrases autorisées dans le langage correspondant.
Grammaire proposée :

Grammaire Géométrique(Vt, Vn, règle, P)

Vt :vocabulaire terminal c'est l'ensemble des lexèmes autorisés dans le langage

Vt={si, alors, et, point, somme,...Ect}∪{==, !=,<,>,+,-,*,/}∪{„ ;,(,)}∪{A-Z,a-z,0-9}

**Remarque :**

Vt contient tous les noms des prédicats et des fonctions qui sont utilisées par l'expert.

Vn :vocabulaire non terminal

Vn={règle, conditions, une condition, prédicat, comparaison, arguments, un argument,

conclusion, fonction, expression, AS, terme, MS, facteur, operateur, argument_F, variable,

constante, numéro }

P={

<règle> → si <conditions> Alors <conclusion>

<conditions> → <une condition> | <une condition> Et <conditions>

<une condition> → <prédicat> | <comparaison>

<prédicat> → identifient (<arguments>)

<comparaison> → <expression> <operateur> <expression>

<arguments> → <un argument> , <arguments> | <un argument> | $\varepsilon$

<un argument> → <constante> | <fonction> | <expression> | <variable>

<conclusion> → identifient (<arguments>)

<fonction> → nom fonction [<arguments>]

<expression> → <terme> <AS>

<AS> → + <terme> <AS> | $\varepsilon$ | - <terme> <AS>

<terme> → <facteur> <MS>

<MS> → * <facteur> <MS> | / <facteur> <MS> | $\varepsilon$

<facteur> → <constante> | <variable> | (<expression>)

<operateur> → == | != | > |<

<argument_F> → <un argument> ; <argument_F> | <un argument>| $\varepsilon$

<variable> → A...Z

<constante> → a...z | <numéro>

<numéro> → $(0|1|2|3|4|5|6|7|8|9)^{+}$

}

### 3.3.2 Format des règles

Une règle de production possède la forme suivante :

si hypothèse alors conclusion

Une hypothèse est soit un prédicat ou une conjonction des prédicats une conclusion est un prédicat nous présentons quelques exemples :
R1 : si droite(A,B) et droite(B,C) et droite(A,C) et ¬alignés(A,B,C) alors triangle(A,B,C)
R2 : si angle(A,B,C,X) et angle(B,C,A,Y) et angle(C,A,B,Z) et X== 90 alors triangle_rectangle(A,B,C)
R3 : si angle(A,B,C,X) et angle(B,C,A,Y) et angle(C,A,B,Z) et X==Y alors triangle_isocèle(A,B,C)
R4 : si triangle_rectangle(A,B,C) et triangle_rectangle(D,C,B) et largeur[AB]==largeur[BC]

alors carré(A,B,C,D)
R5 : si carré(A,B,C,D) alors rectangle(A,B,C,D)

### 3.3.3 L'analyse syntaxique

L'analyse syntaxique est dite « phase de compilation » au cours de laquelle la conformité, par rapport à une grammaire donnée, est vérifiée. L'étape précédente est l'analyse lexicographique.
L'analyse syntaxique est parfois appelée « analyse grammaticale ».

L'algorithme de l'analyse syntaxique :

**Algorithme 4:** AnalyseSyntaxique

```
48 R : est une règle;
49 Règle_Correct ← vrai;
50 début
51     si (R.StartsWith("si",0)) alors
52         R← R.supprimer("si");
53         si (R.Contien("Alors")) alors
54             Condition_Conclusion[ ] ← R.Split("Alors");
55             pour (i=0 à Condition.longeurs et Règle_Correct = vrai) faire
56                 si ( !predicat(condition[i]) et !comparaison(condition[i])) alors
57                     Règle_Correct← faux;
58                     Afficher("erreur syntaxique, cause : partie conditions");
59                 fin
60             fin
61             si (Règle_Correct == vrai) alors
62                 si ( ! pridicat(Condition_Conclusion[1])) alors
63                     Règle_Correct← faux;
64                     Afficher("erreur syntaxique, cause : partie conclusion");
65                 fin
66             fin
67         sinon
68             Règle_Correct← faux;
69             Afficher("erreur syntaxique, cause : absence de alors");
70         fin
71     sinon
72         Règle_Correct← faux;
73         Afficher("erreur syntaxique, cause : absence de si");
74     fin
75     si (Règle_Correct == vrai) alors
76         Afficher("La règle est bien écrit");
77     fin
78 fin
```

### 3.3.4 Analyse sémantique

Nous avons suggéré une analyse sémantique pour assurer que chaque fait est une instance d'un prédicat, ceci est assurer par la vérification des propriétés suivantes : nom, nombre des paramètres et les types des paramètres de chaque prédicat. Dans cette partie, nous avons utilisé un dictionnaire qui contient les prédicats du SEGM avec le nombre et les types des paramètres de chaque prédicat.

**Exemple :**
prédicat point(A,X,Y) est réprésenter comme suit :
point 3 String Float Float

**Algorithme 5:** AnalyseSémantique

```
79  f :prédicat instancé;
80  Table ← f.split("(");
81  Nom ← Table[0];                                          /* nom de f */
82  début
83      si (Nom !=dictionnaire[0]) alors
84          b=faux;
85          Afficher("Erreur sémantique : prédicat de ce nom n'éxiste pas");
86      fin
87      Argument[ ] = Table[1].split(")");                   /* supprimer ) */
88      si (b = vrai) alors        /* Argument.longeur : nomber de paramètre */
89          si (Argument.longeur != dictionnaire[1]) alors
90              b ← faux;
91              Afficher("Erreur sémantique : nombre de paramètres");
92          fin
93      fin
94      si (b == vrai) alors                          /* vérification des types */
95          pour (i=0 à Argument.longeur) faire
96              si (Argument[i] != dictionnaire[i+2]) alors
97                  b = faux;
98                  Afficher("Erreur sémantique :type des paramètres");
99              fin
100         fin
101     fin
102 fin
```

**Saisie automatique :**
Nous avons utilisé un algorithme qui permet à l'utilisateur ou l'éxpert de visualiser les prédicats similaires à celui en cours de saisie afin de minimiser les erreurs et faciliter l'édition des règles de production.

**Algorithme 6:** SaisieAutomatique

```
103 L : liste des noms des prédicats;
104 début
105     tant que non stop faire
106         S : chaîne de caractères saisie dans la zone d'écriture;
107         i←0;
108         tant que i<L.size faire
109             si L.get(i).startWithe(S) alors
110                 Liste_des_choix.add(L.get(i));
111                 i←i+1;
112             fin
113             si cliquez sur un élément de la liste des choix alors
114                 S← élément choisi;
115             fin
116         fin
117     fin
118 fin
```

## 3.4 Contrôle de cohérence

Il existe deux types de contrôle de cohérence qui sont :

### 3.4.1 Contrôles locaux

Le contrôle local est l'analyse d'une règle isolée, sans considération de conflit avec d'autres règles. Dans cette étape nous allons vérifier trois cas d'incohérence locales qui peuvent apparaitre dans la syntaxe d'une règle (Figure3.1).

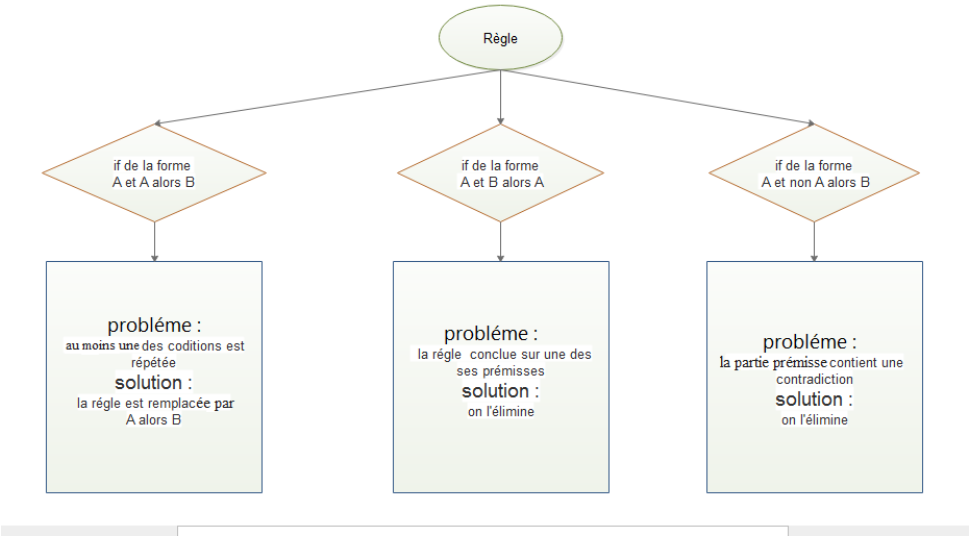


FIGURE 3.1 – Contrôles locaux

Algorithme proposé pour le maintient de la cohérence locale :

**Algorithme 7:** ContrôleLocaux

```
Entrée : Règle;
début
    String [ ] conditions_conclusion = Règle.séparer("Alors");
    String conclusion = conditions_conclusion[1];
    String [ ] conditions = conditions_conclusion[0].séparer("et");
    si conditions.contiens(conclusion) alors
        Ecrire("La forme A et B alors A et détecté");
        La règle est pas ajouter a BR;
    fin
    si ("¬"+condition[i])==condition[j] et i != j alors
        Ecrire("La forme A et ¬A est détecté);
        La règle est pas ajouter à la BR;
    fin
    si condition[i]==condition[j] alors
        Ecrire("La forme A et A alors B est détecté");
        Suppression les répététion des condition;
    fin
fin
```

### 3.4.2 Contrôles globaux

Le contrôle global est l'analyse d'une règle avec considération de conflit avec les autres règles de la base de connaissances, ce conflit provoque une incohérence dans le raisonnement du système ce qui peut produire des contradictions et des anomalies dans la base de connaissances : parfois, on dit que le système déconne dans cette étape nous allons vérifier trois conflits possibles.

#### 3.4.2.1 Contrôle de redondance

Afin d'éviter la répétition inutile des connaissances particulièrement du savoir faire dans la base de connaissances, nous utilisons l'algorithme suivant :

**Algorithme 8:** ContrôleDeRedondance

```
BR : base de règles;
R : la nouvelle règle;
début
    si R ∉ BR alors
        Ajouter R a BR;
    sinon
        Afficher("R est existe déja");
    fin
fin
```

#### 3.4.2.2 Contrôle d'inclusion

Un ensemble des règles est peut-être inclus dans une seule règle ceci est un cas particulier de redondance, qui peut provoquer l'exécution de plusieurs règles inutilement, on dit qu'une règle r : P → Q est redondante s'il existe une règle r1 : P1 → Q1 Tel que : P1 ⊂ P et Q ⊂ Q1
L'algorithme suppression d'inclusion permet de traiter cette anomalie :

**Algorithme 9:** SuppressionD'inclusion

```
Entrée : Base de règles;
début
    pour toute R ∈ BR faire
        String [ ] conditions_conclusion = R.séparer("Alors");
        String conclusion = conditions_conclusion[1];
        String [ ] conditions = conditions_conclusion[0].séparer("et");
        pour toute R1 ∈ BR et R != R1 faire
            String [ ] conditions_conclusion1 = R1.séparer("Alors");
            String conclusion1 = conditions_conclusion1[1];
            String [ ] conditions1 = conditions_conclusion1[0].séparer("et");
            si pour toute c ∈ conditions → c ∈ conditions1 alors
                Supprimer R1;
            fin
        fin
    fin
fin
```

#### 3.4.2.3 Format des cycles

Un n_uplet des règles $R_1 : P_1 \rightarrow Q_1$,..., $R_n : P_n \rightarrow Q_n$ forment un cycle si ∀ i tel que 1<i<n $Q_i \subset P_{i+1}$ et $Q_n \subset P_i$
L'algorithme Cycle permet de traiter cette anomalie :

**Algorithme 10:** Cycle

```
R : la nouvelle règle;
BR : la base de règles;
début
    L_C_R ← hypothèses de R;        /* L_C_R:liste des conditions de R */
    C_R ← la conclusion de R;                  /* C_R: conclusion de R */
    pour toute règle R1 ∈ BR faire
        L_C_R1 ← les hypothèses de R1;  /* L_C_R1: liste des conditions de
        R1 */
        C_R1 ← la conclusion de R1;             /* C_R1: conclusion de R1 */
        si L_C_R.contiens(C_R1) et L_C_R1.contien(C_R) alors
            la règle R n'est pas ajoutée à BR;
            afficher(" R n'est pas ajoutée ");
        fin
    fin
fin
```

## 3.5 Architecture du SEGM

Nous avons proposé un système divisé en deux parties, une pour l'expert et l'autre pour l'utilisateur (Figure 3.2) :
1- Pour l'expert : SEGM possède un compilateur des règles comme nous avons déjà vu dans ce chapitre, et une interface qui permet à l'expert ou cogniticien d'enrichir la base de connaissances.
2- Pour l'utilisateur : SEGM possède une interface pour l'interaction avec l'utilisateur (l'insertion des faits, poser des questions, etc..).
Pour la distribution du raisonnement nous avons proposé une approche multi-agents pour un raisonnement semi distribué qui consiste à utiliser deux types d'agents (agent statique, agent dynamique) le système va générer un agent statique et plusieurs agents dynamiques qui vont être activés en même temps.
SEGM possède un système de maintent de vérité pour résoudre le problème d'incohérence dans la base de connaissances.

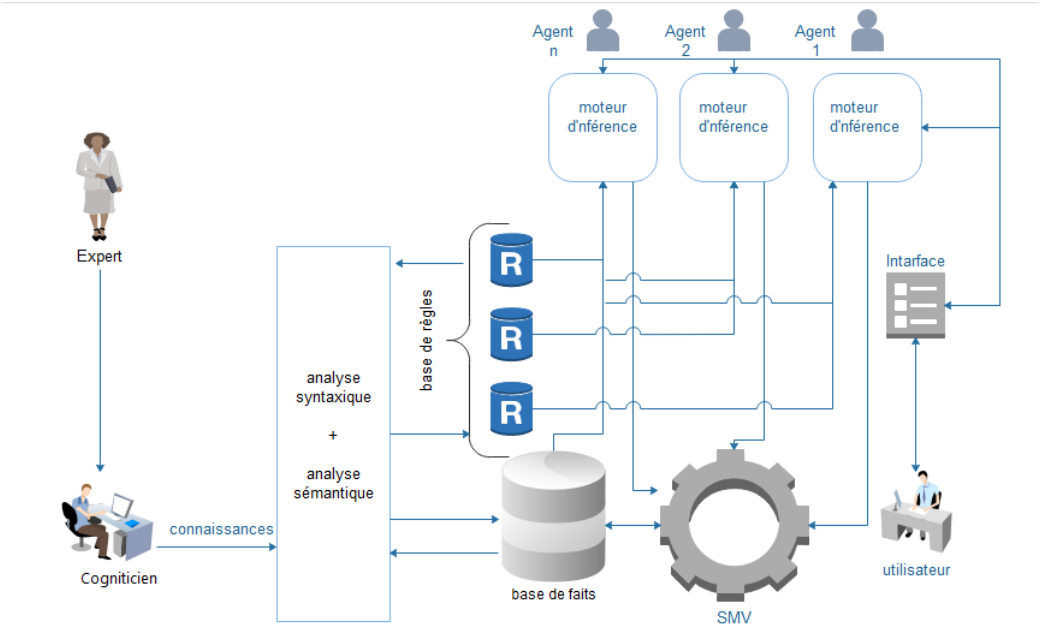


FIGURE 3.2 – Architecture générale du SEGM

### 3.5.1 Les agents du SEGM

Le système multi-agents utilise deux types d'agents (statique et dynamique) (Figure 3.3)
1- Agent statique (principal) :
L'agent principal possède :
- Module de contrôle grammatical (compilateurs du SEGM)
- Module de génération des agents dynamiques
- Module de communication avec les agents dynamiques
- Deux interfaces (interface utilisateur, interface expert, etc..)
- Un module de contrôle de cohérence
- Tableau noir
2- Agent dynamique :
L'agent dynamique possède :
- Une base de règles locales contient un nombre maximal de règles qui sera fixé selon les conditions des tests
- Un moteur d'inférence adapté fonctionnant en chaînage avant avec et sans but
- Un module de communication pour communiquer avec l'agent principal
- Une liste des agents accointances

**Remarque :**
Un agent $A_i$ est un agent accointance de l'agent $A_j$ s'il existe une règle de $A_i$ qui conclu sur un des prémisse d'une règle de $A_j$.

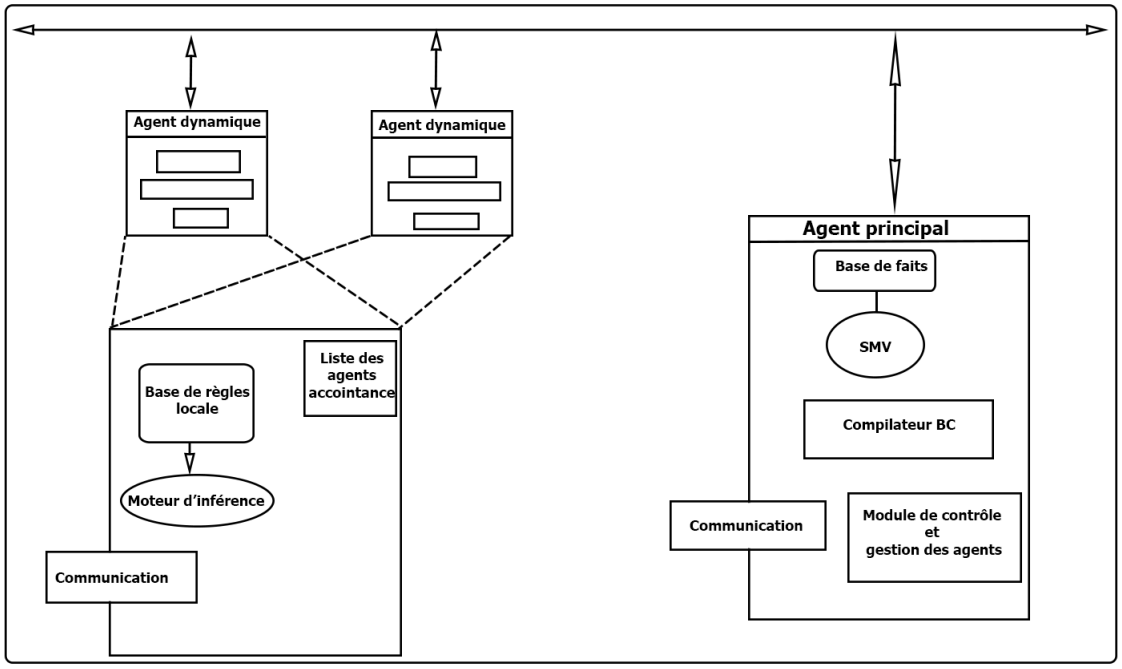


FIGURE 3.3 – Les agents du SEGM

### 3.5.2 Fonctionnemnet de SEGM

Nous avons proposé un système semi distribué (Figure 3.4) qui fonctionne comme suit :
- Lorsque l'utilisateur lance une requête d'exécution l'agent principal génère plusieurs (selon la taille de la base de règles) agents dynamiques avec l'état actif et distribue ou réparti la base de règles entre eux
- Chaque agent dynamique du SEGM possède un MR, ce dernier lui permet de raisonner en utilisant les connaissances locales (sous base de règles de production locale) et les connaissances globales (la base de faits globale) ainsi les nouveaux faits inférés sont soumis au contrôle de cohérence par le SMV avant d'être insérés dans la mémoire de travail.
- Tous les agents raisonnent en même temps sur la même base de faits, tell que chaque agent possède deux états possibles :
1- Attendre : lorsque l'ensemble de conflit ne change pas
2- Actif : l'agent est entrain de raisonner
- Si tous les agents sont à l'état attendre ou but trouvé alors l'agent principal arrête tous les agents dynamiques
- Un agent dynamique est passé de l'état attendre à l'état actif lorsqu'un des agents accointances ajoute une nouvelle conclusion à la base de faits
- Un agent dynamique est passé de l'état actif à l'état attendre lorsque la base de faits ne change pas après deux cycles de raisonnement successifs
- Les agents dynamiques communiquent entre eux à travers l'agent principal qui contient BF
**- Les messages de communication :**
- état(agent$_i$, attendre) : l'état de l'agent$_i$ est attendre

- état (ACC(agent$_i$),état) : l'état de l'agent accointance de l'agent$_i$

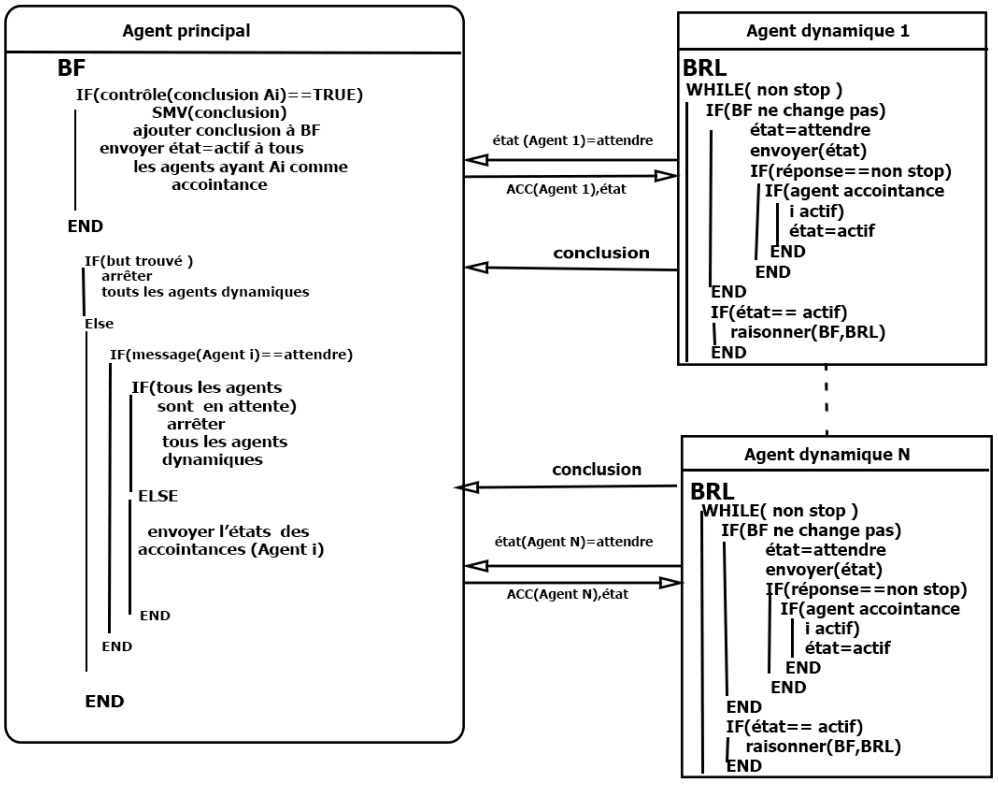


FIGURE 3.4 – Fonctionnement général du SEGM

## 3.6 Moteur d'inférence

Comme nous avons déja vu dans le premier chapitre, le moteur d'inférence est l'équivalent de l'esprit logique de l'expert humain qui raisonne sur des faits connus pour en sortir de nouvelles vérités (Figure3.5).

Pour réaliser un moteur d'inférence, nous avons adapté et proposé ensuite implémenté un ensemble d'algorithmes que nous allons présenter.

### 3.6.1 Algorithme d'unification (unificateur)

Une substitution s unifie deux termes si elle les rend identiques : s unifie t et t' si (t)s=(t')s.
Un unificateur d'un ensemble fini d'équations entre termes E={t1=t'1, ..., tn=t'n} est une substitution qui unifie les deux termes de chaque équation

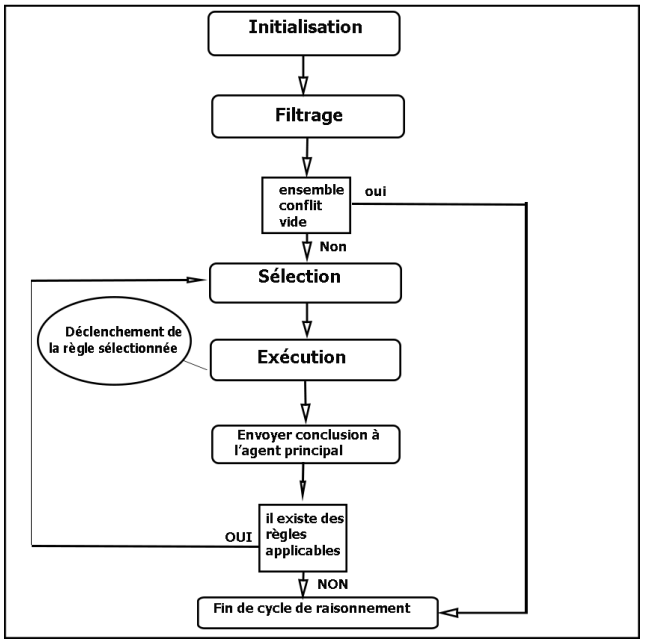


FIGURE 3.5 – Fonctionnement de moteur d'inférence adapté

**Algorithme 11:** AlgorithmeD'unification

entrée : un ensemble fini E d'équation entre termes;
sortie : ou bien échec, ou bien un upg de E;
**début**
  **tant que** (E n'est pas résolu) **faire**
    choisir une équation de E;
    appliquer une des règles suivantes à cette équation :
    **si** elle est de la forme t = t **alors**
      la supprimer;
    **fin**
    **si** elle est de la forme $f(t_1,...,t_n) = g(t'_1,...,t'_m)$ et f et g sont différentes **alors**
      échec;
    **fin**
    **si** elle est de la forme $f(t_1,...,t_n)=f(t'_1,...,t'_n)$ **alors**
      la remplacer par n équations $t_1=t'_1,...,t_n=t'_n$;
    **fin**
    **si** elle est de la forme t = x et t n'est pas une variable **alors**
      la remplacer par x=t;
    **fin**
    **si** elle est de la forme x = t et x apparaît dans t **alors**
      échec;
    **fin**
    **si** elle est de la forme x = t et x n'apparaît pas dans t **alors**
      remplacer x par t partout ailleurs dans E;
    **fin**
  **fin**
  rendre la subsitution $s_E$ associé à E;
**fin**

**Remarque :**
Un cas spécial : après l'opération d'unification, il faut remplacer les fonctions déjà unifiées par leurs valeurs. Ceci est possible grâce à l'utilisation d'une liste qui contient les noms de toutes les fonctions utilisées dans le système, une fois le nom de la fonction est trouvé dans la liste précédente, la procédure correspendante est déclenchée

**Algorithme 12:** CasSpécial

```
entrée : R une règle unifie;
donnée : LF liste des fonctions;
début
    pour toute condition C de R faire
        si C.contien une fonction f et f ∈ LF alors
            appeller la fonction correspondante f et la remplacer par sa valeur;
        fin
    fin
fin
```

### 3.6.2 Ensemble de conflit

Dans cette étape nous avons sélectionné les règles qui peuvent effectivement être déclenchées, ou bien c'est la production d'un sous ensemble de règles appelé ensemble conflits voici l'algorithme

**Algorithme 13:** AlgorithmeDeFiltrage

```
E{}=ensemble conflit;
BR=base de règles;
début
    pour toute règle R de la base de règles faire
        pour toute hypothèse de R qui appartient à la base de faits faire
            UPG(hypothèse, fait);
            si unification alors
                E=E+R;
                BR=BR-R;
            fin
        fin
    fin
fin
```

### 3.6.3 Sélection

Le but de cette étape est de choisir la règle à déclencher parmi les règles de l'ensemble de conflit, pour cela, nous avons utilisé une heuristique
voici l'algorithme de sélection

**Algorithme 14:** AlgorithmeDeSélection

```
E={ensemble de règles applicables};
début
    pour toute règle de l'ensemble E faire
        sélectionner une règle selon l'heuristique proposée;
    fin
fin
```

Nous avons proposé une heuristique pour le choix de la règle à éxécuter parmi l'ensemble des règles déclanchables.
L'idée de l'heuristique que nous avons proposé est de choisir la première règle qui possède un nombre minimal des hypothèses
voici l'algorithme d'heuristique

**Algorithme 15:** Heuristique

```
BRD : ensemble de règles déclenchables de l'agent;
R : La règle choisie;
R← BRD(0);
début
    pour i=1 à BRD.size faire
        /* R.PG: partie gauche de la règle R                    */
        si R1.PG.size<R.PG.size alors
            R← R1;
        fin
    fin
    return R;
fin
```

### 3.6.4 Algorithme de chaînage avant

Le chaînage avant commence par des données connues et procède de l'avant avec ces données avec certaines propriétés :
- Les règles avec la partie gauche vraie sont déclenchées
- Les règles déclenchées ajoutent des nouvelles données dans la base de faits
- Une règle peut seulement être activée une fois
- Le cycle s'arrête lorsqu'il n'y a plus de règles à déclencher (chaînage avant sans but)
- Le cycle s'arrête lorsqu'il n'y a plus de règles à déclancher ou le but chercher trouvé (chaînage avant avec but)

Voici l'algorithme de chaînage avant sans but :

**Algorithme 16:** ChaînageAvantSansBut

```
Entree : BF,BR;
début
    tant que (∃ R ∈ BR) faire
        (R applicable);
        - choisir une règle applicable R;
        BR = BR - R;
        Envoyer conclusion de R;
    fin
fin
```

Voici l'algorithme de chaînage avant avec but :

**Algorithme 17:** ChaînageAvantAvecBut (B)

```
Entree : BF,BR;
début
    tant que (B ∉ BF et ∃ R ∈ BR) faire
        (R applicable);
        - choisir une règle applicable R;
        BR = BR - R;
        Envoyer conclusion de R;
    fin
    si (B ∈ BF) alors
        B est établi;
    sinon
        B n'est pas établi;
    fin
fin
```

## 3.7 système de maintien de vérité (SMV)

SMV est un processus de maintien de cohérence attaché a un moteur d'inférence, ce processus doit vérifier toute nouvelle insertion ou suppression dans la base de connaissances.

Avant d'expliquer ce processus, nous allons rappeler quelques concepts de base que nous avons présenté au chapitre 1 :

**1- Un noeud :** dans un SMV toute connaissance est représentée par un noeud constitué de :

- identificateur : nom du fait généré par MR
- Etat (IN/OUT)
- liste des justifications

**2- Justification :** une justification est une connaissance qui permet de déterminer l'état de noeud soit IN ou OUT, c'est à dire le noeud appartient au contexte ou non.

Une justification comporte deux listes : la liste IN et la liste OUT.

**3- Contexte :** est constitué de l'ensemble des noeuds déductibles (état = IN)

### 3.7.1 Traitement des contradictions :

Pour détecter une contradiction, nous avons utilisé un dictionnaire des contradictions qui contient les connaissances qui sont cause de contradiction.
Nous avons modélisé ce dictionnaire par un fichier, ce dictionnaire est chargé par l'expert de domaine.
**Exemples :**
Triangle_rectangle(A,B,C) $\perp$ Triangle_équilatérale(A,B,C)
Losange(A,B,C,D) $\perp$ rectangle(A,B,C,D)

#### 3.7.1.1 Propagation

Si une contradiction est détectée alors, le processus de propagation se déclenche pour enlever la contradiction et propager les changement d'états des différents noeuds considérés dans le chemin de la propagation, elle s'opère si le support d'un noeud devient invalide et aucune autre justification du noeud n'est valide.
Différents cas se présentent :
- Si le noeud invalidé appartient à la liste IN d'une justification support, alors le noeud supporté est invalidé si aucun autre support ne peut être trouvé.
- Si le noeud invalidé appartient à la liste OUT d'une justification d'un noeud non valide, celle-ci est réévaluée OUT $\leftarrow$ IN, s'opère si un support est trouvé pour un noeud
- Si le noeud validé appartient à la liste OUT d'une justification support d'un noeud, alors le noeud supporté est invalidé, si aucun autre support ne peut être trouvé.
- Si le noeud validé appartient à la liste IN d'une justification d'un noeud non valide, alors celle-ci est réévaluée. En général lorsqu'un noeud change d'état tous les noeuds qu'il justifie sont mis à jour.

**Algorithme 18:** Propagation

```
N : Noeud;
BN : base de noeuds;
début
    si N.état == IN alors
        N.état ← OUT;
        pour toute N_k ∈ BN faire
            si N_k.état == IN alors
                                /* J est une justification valide de noeud N_k */
                si N ∈ liste_IN de J alors
                    si ∄ J_i tel que J_i valide alors
                        N_k.état ← OUT;
                    fin
                fin
            sinon
                si N ∈ liste_out de J alors
                    N_k.état ← réevaluer();
                fin
            fin
        fin
    sinon
        N.état← IN;
        pour toute N_k ∈ BN faire
            si N_k.état == IN alors
                si N ∈ liste_OUT de J et J est valide alors
                    si ∄ J_i tel que J_i valide alors
                        N_k.état ← OUT;
                    fin
                fin
            sinon
                si N ∈ liste_IN de J alors
                    N_k.état ← réevaluer();
                fin
            fin
        fin
    fin
fin
```

**Algorithme 19:** Réevaluer

```
300 L_J : liste des justifications d'un noeud N;
301 début
302 |  N.état ← IN;
303 |  pour toute j ∈ L_J faire
304 |  |  pour toute noeud M ∈ liste_IN de J faire
305 |  |  |  si M.état == OUT alors
306 |  |  |  |  N.état ← out;
307 |  |  |  |  Exit() ;
308 |  |  |  fin
309 |  |  fin
310 |  |  pour toute noeud M ∈ liste_out de J faire
311 |  |  |  si M.état == IN alors
312 |  |  |  |  N.état ← out;
313 |  |  |  |  Exit() ;
314 |  |  |  fin
315 |  |  fin
316 |  fin
317 fin
```

## 3.8 Conclusion

Nous avons détaillé dans ce chapitre les principaux outils conceptuels et algorithmiques que nous avons utilisé pour concevoir et réaliser le correcteur géométrique multi-agents que nous avons proposé dans ce projet. Notre travail est orienté selon cinq axes :
- La construction d'une base de connaissances formalisant les définitions et les théorèmes de la géométrie à deux dimensions.
- Réalisation d'un moteur d'inférence d'ordre 1 et pour améliorer son éfficacité nous avons proposé de le concevoir et le réaliser par une approche semi distribuée.
- Des algorithmes pour la vérification syntaxique et sémantique de la base de connaissances.
- Afin d'assurer une cohérence minimale et éviter la présence de contradictions dans la base de connaissances, nous avons intégré un processus procédant comme un SMV avec le moteur d'inférence.

Dans le chapitre suivant nous présentons l'implémentation des différents algorithmes. Nous allons tester le système dans différents cas ensuite, nous présentons et nous interprétons les résultats obtenus.