République Algérienne démocratique et populaire
Ministère de l'enseignement supérieur et de la recherche scientifique

Université MENTOURI Constantine
Faculté des Sciences de l'Ingénieur
Département d'Informatique

Direction de la Post-Graduation et de la Recherche Scientifique
École Doctorale de l'Est en Informatique
Pôle de CONSTANTINE

Mémoire pour obtenir le diplôme de

Magister

Spécialité : Informatique
Option : Intelligence Artificielle

Thème

Vers une méthode pour la conception et l'adaptation de situations d'apprentissage collaboratif

par

Sara BOUTAMINA

Devant le jury :

Président :	Mr. Mahmoud Boufaïda, Professeur, Université Mentouri (Constantine)
Rapporteur :	Mme. Hassina Seridi, MC, Université Badji Mokhtar (Annaba)
Examinateur :	Mme. Habiba Belleili, MC, Université Badji Mokhtar (Annaba)
Examinateur :	Mr. Nacereddine Zarour, MC, Université Mentouri (Constantine)

2009

à mes parents,
à mon frère Amin,
à ma sœur Lilia dite Racha.

Remerciements

J’adresse mes vifs remerciements à Mme. Hassina SERIDI, Maître de Conférences, Université Badji Mokhtar (Annaba) pour son soutien constant, ses encouragements continus, sa gratitude et sa qualité humaine qu’elle m’a manifestés durant la réalisation de ce travail.

J’exprime mes remerciements à Mr. Mahmoud BOUFAÏDA, Professeur, Université Mentouri (Constantine) d’avoir accepter la présidence du jury, ainsi que les docteurs Mme. Habiba BELLEILI, Maître de Conférences, Université Badji Mokhtar (Annaba) et Mr. Nacereddine ZAROUR, Maître de Conférences, Université Mentouri (Constantine) d’avoir accepter de juger ce mémoire.

Je remercie également Mr. Abdelkader GOUAÏCH, Maître de Conférences, Université Montpellier II (France), pour ses conseils, sa disponibilité et son intérêt pour le travail effectué.

Je tiens à remercier tous ceux qui m’ont aidé de près ou de loin pour la réalisation de ce travail.

Je remercie enfin ma famille et plus particulièrement mes parents, qui m’ont toujours soutenue.

Résumé

L’apprentissage collaboratif n’est pas toujours efficace. Ses effets dépendent de la richesse et de l’intensité des interactions entre étudiants pendant la collaboration (Dillenbourg et al., 1996). La conception des scénarios pédagogiques n’est pas facile et elle nécessite des informations sur les apprenants et sur leurs interactions.
En effet, toute conception qui a pour cible des apprenants humains requiert une évaluation auprès de ces derniers. Les démarches de conception font rarement appel à un retour expérimental pour améliorer la conception initiale. Pour guider une conception, il faut mettre en jeu une évaluation expérimentale du scénario réalisé, et prendre en compte le comportement et les interactions des apprenants pour améliorer le scénario.
Dans notre travail, nous avons présenté une approche de construction de scripts de collaboration incrémentale dédiée à l’apprentissage collaboratif avec prise en compte des interactions des apprenants. Il s’agit de suivre le comportement des apprenants en respectant le scénario prescrit par le concepteur et fournir des retours sur l’exécution de ce dernier pour revoir ou affiner la conception préliminaire.
Pour cela, on suggère l'utilisation d'un ensemble d'agents ayant les rôles suivants: décision, interprétation, exécution, observation, et suivi de trace. Le travail des agents commence par l'interaction des différents apprenants avec le système. En se basant sur les profils de ces apprenants, un script sera sélectionné (adéquat aux profils et non pas aux comportements/collaboration des apprenants) par un agent de décision. Dans une deuxième phase, celui-ci (le script) sera traduit et interprété par un ensemble d’agents. Finalement un autre ensemble d’agents l'exécutera en tenant compte des différents profils de ces apprenants. L’exécution du scénario sera contrôlée par un observateur (un groupe d'agents). Des agents de trace sont responsables du suivi des traces des apprenants.
Pour la mise en œuvre de ces agents, nous avançons qu’un langage de programmation orienté agent est requis. Nous proposons d’aller vers la proposition d’un langage suite à une analyse détaillée des langages existants afin d’introduire un certain nombre d’aspects requis en général par ce type de langages et dûment appliqué dans notre cas. Pour ce faire, nous apportons une grammaire décrite en notation LBNF, ainsi que le compilateur.

Table des matières

3. Les Langages de Programmation Agent : Étude Comparative

4. Conception des Scripts par Retour expérimental

5. Vers un langage de Programmation d'Agents Pour la Conception des scripts

La liste des figures :

La liste des tableaux :

Introduction Générale

L'apprentissage collaboratif est un terme général pour une variété d'approches dans l'éducation, qui impliquent un effort intellectuel commun par les apprenants et les enseignants. Des groupes d'apprenants travaillent ensemble afin de résoudre les problèmes et atteindre leurs objectifs.

Devant les progrès de la technologie, les chercheurs se sont intéressés ces dernières années à l'apprentissage collaboratif assisté par ordinateur. Ce domaine de recherche est connu sous l'appellation CSCL (Computer-Supported Collaborative Learning).

Le CSCL représente un nouveau paradigme pluridisciplinaire dans le domaine de l'Apprentissage basé sur les Technologies (TEL: Technology Enhanced Learning), dans lequel les technologies de l'information et la communication (TIC) sont employées en vue d'améliorer et de contribuer au développement des activités d'apprentissage collaboratives.

Dans le CSCL, Les interactions sociales jouent un rôle important et essentiel dans l'apprentissage. Les effets de ce dernier dépendent de l'émergence de certaines interactions riches telles que l'explication, la résolution des conflits,…etc. Toutefois, ces interactions ne se produisent pas de la libre collaboration (Dillenbourg, 2002). Une façon d'améliorer l'efficacité de l'apprentissage collaboratif est d'engager les apprenants dans des interactions structurées faisant partie de scénarios bien définis. Le CSCL est basé sur cette notion de scénario.

Certains auteurs préfèrent à cette notion l'utilisation d'expressions telles que : conception pédagogique, méthode pédagogique, approche pédagogique et technique pédagogique. En fait, chaque expression se réfère à peu près à l'idée d'un scénario pédagogique ou d'apprentissage

Dans le domaine de CSCL, la notion de scénario est appelée "script de collaboration " ou "script " tout court. Un script est une séquence de phases bien définies ayant plusieurs attributs. Ces phases servent à réguler la collaboration afin qu'elle puisse aboutir à un apprentissage que l'on souhaite efficace. Ces scripts structurent l'apprentissage en définissant la composition des groupes, la distribution des rôles et des ressources ainsi que la coordination des activités qui composent le processus d'apprentissage (Dillenbourg, 2002).

La structuration de ce processus d'apprentissage est très importante. Néanmoins elle reste primordiale et complexe. Afin de faciliter cette tâche plusieurs méthodes de conception de scripts ont été proposées.

Cadre et motivation

Notre travail porte sur la conception des scripts (Scripting) et le suivi des apprenants (tracking) dans des situations d'apprentissage collaboratif. Un script collaboratif d'une manière générale est une séquence de phases qui permet l'organisation de l'apprentissage en spécifiant comment les apprenants doivent former des groupes, collaborer et résoudre un problème.

La conception des scripts n'est jamais une tâche facile, elle nécessite des informations sur les apprenants et leurs interactions. Afin de faciliter cette tâche, on trouve qu'une approche de conception incrémentale est plus adaptée pour concevoir des scripts adéquats aux situations d'apprentissage.

Une fois conçu, celui-ci doit être exécuté par le système qui doit suivre les traces des interactions des apprenants. La distribution d'un tel système et la possibilité d'utilisation de plusieurs entités pour résoudre les problèmes (par exemple la collecte des traces…etc.) nous a amené à utiliser le paradigme des systèmes multi-agents (SMA). Ces systèmes sont un domaine relativement récent, apparu dans les années 80 et issu de la rencontre de l'Intelligence Artificielle et des Systèmes Distribués.

Le paradigme d'agent permet d'offrir des abstractions pour l'analyse et la modélisation des complexités des organisations. Il permet de voir les humains et les systèmes logiciels comme des entités qui interagissent et collaborent afin de réaliser leurs buts, et exécuter leurs tâches.

Néanmoins, proposer un nombre adéquat d'agents n'est pas suffisant pour autant dans notre proposition. Nous nous sommes particulièrement intéressés aux aspects touchant au développement et au déploiement des agents artificiels. Ainsi une part centrale de la thèse est consacrée aux langages de programmation agents et aux concepts à prendre en compte en général afin de pouvoir programmer des agents pour un tel système.

Objectifs et contribution

Notre objectif est de proposer une approche pour la conception incrémentale des scripts de collaboration fondée sur les agents intelligents en tenant compte des apprenants.

Une variété de méthodes de conception de scénarios a été proposée mais aucune ne prend en compte les interactions des apprenants et les utilise de façon incrémentale dans le processus de conception des scripts. En fait, on a besoin de prendre en compte les interactions des apprenants présentés par leurs traces afin d'améliorer progressivement la conception initiale. Ceci est possible par un retour expérimental suite à l'exécution des scripts conçus par des groupes d'apprenants pour d'éventuels ajustements.

Le paradigme des systèmes multi-agents est un paradigme puissant permettant la modélisation des complexités des systèmes et adéquat pour notre approche de conception des scripts de collaboration.

La mise en œuvre de ces agents requiert l'utilisation d'un langage de programmation agent permettant d'exprimer facilement les différentes fonctions que doivent effectuer les agents impliqués par cette conception.

Ainsi notre contribution se résume en :

- Une étude des théories d'apprentissage collaboratif, des approches de conception des scripts et de suivi de traces dans le domaine des EIAH (Environnement Informatique d'Apprentissage Humain).
- Une étude détaillée et comparative des langages de programmation agents.
- Proposition des concepts généraux à prendre en compte par le langage de programmation.
- Proposition de modèles généraux pour la conception des scripts et le suivi des apprenants implémentés à l'aide du paradigme des agents et répondant aux différents concepts proposés.
- Proposition d'une grammaire pour le langage en question.
- Implémentation d'un compilateur pour le langage proposé.

Organisation du mémoire :

Ce manuscrit est composé de cinq chapitres :

1. Le premier chapitre présente une vue générale sur l'apprentissage collaboratif en présentant les scripts de collaboration et leurs intérêts pour l'organisation de celui-ci et les traces des apprenants. Il présente également les travaux portant sur la conception des scripts de collaboration et le suivi des traces des apprenants.

2. Le deuxième chapitre a pour objectif de donner la définition des agents artificiels, les théories, les architectures et les langages de programmation agents.

3. Dans le troisième chapitre, nous présentons une étude comparative entre les différents langages de programmation agents et frameworks.

4. Dans le quatrième chapitre, nous décrivons notre approche de conception des scripts de collaboration. Cette approche repose sur deux modèles, basés sur l'utilisation des agents: le premier introduit la conception des scripts de collaboration d'une manière générale et le deuxième représente le suivi des traces des apprenants.

5. Dans le cinquième chapitre, nous décrivons le langage proposé en présentant sa grammaire et les différents composants du compilateur associé à ce langage.

Nous terminons par une conclusion et les perspectives de ce travail.

Chapitre 1

L'Apprentissage Collaboratif

1.1. Introduction

L'apprentissage collaboratif offre des avantages aux apprenants, mais cette collaboration doit être structurée pour mieux profiter. Cette structuration est basée sur la notion de script qui représente d'une manière générale un ensemble de phases qui permet l'organisation de la collaboration afin qu'elle puisse offrir un apprentissage que l'on souhaite efficace.

Pour mieux structurer cette collaboration, on a besoin d'avoir une idée sur les apprenants et leurs comportements. Le suivi des traces des apprenants nous permet d'avoir une information précise et adéquate sur leurs états, leurs évolutions individuelles et collectives.

Dans ce chapitre, nous allons présenter l'apprentissage collaboratif en introduisant ses différentes approches. Dans une deuxième phase on présentera les scripts en vue de structurer cet apprentissage et enfin le suivi des traces des apprenants.

1.2. Environnement Informatique d'Apprentissage Humain

Un Environnement Informatique d'Apprentissage Humain (**EIAH**) est défini selon (Tchounikine, 2002) comme suit:

> « U*n environnement qui intègre des agents humains (e.g. élève ou enseignant) et artificiels (i.e., informatiques) et leur offre des conditions d'interactions, localement ou à travers les réseaux informatiques, ainsi que des conditions d'accès à des ressources formatives (humaines et/ou médiatisées) locales ou distribuées* »

Un EIAH est un environnement informatique conçu afin de 'favoriser l'apprentissage humain', c'est-à-dire la construction de connaissances chez un apprenant. Il permet à ses utilisateurs d'interagir avec d'autres personnes ou avec des agents artificiels, d'accéder à des éléments formatifs de toutes natures, locales ou distribuées.

1.3. L'apprentissage collaboratif

1.3.1. Définition de l'apprentissage collaboratif

Henri, F. et Lundgren-Cayrol, K. (Henri & Lundgren-Cayrol, 1998) soulignent la définition suivante de l'apprentissage collaboratif:

> *« L'apprentissage collaboratif est une démarche active et centrée sur l'apprenant. Au sein d'un groupe et dans un environnement approprié, l'apprenant exprime ses idées, articule sa pensée, développe ses propres représentations, élabore ses structures cognitives et fait une validation sociale de ses nouvelles connaissances. La démarche collaborative reconnaît les dimensions individuelles et collectives de l'apprentissage, encourage l'interaction et exploite les cognitions réparties au sein de l'environnement. Le groupe, acteur principal et ressource première de la collaboration joue un rôle de soutien et de motivation. Il contribue à l'atteinte, pour chaque apprenant, d'un but commun et partagé. La collaboration qui s'y développe est faite de communication entre apprenants, de coordination de leurs actions et d'engagement de chacun face au groupe. »*

Ils pensent que l'apprentissage collaboratif n'est pas une théorie d'apprentissage mais une démarche d'apprentissage en vue de la construction progressive des connaissances. Pour eux, l'apprentissage collaboratif a des origines théoriques dans le courant constructiviste. Selon ce courant, une personne peut réussir à "construire" ses connaissances dans le cadre d'un processus d'interactions entre lui et son entourage.

Une autre définition est celle de (Roschelle & Teasley, 1995) qui ce base sur l'approche de la cognition partagée. Selon cette approche,

> *« la collaboration est un processus de construction et de maintien d'une conception partagée d'un problème. »*

L'attention est portée sur les conceptions émergentes au plan social, analysées en tant que productions du groupe.

1.3.2. Les approches de l'apprentissage collaboratif

Il existe trois approches théoriques de l'apprentissage collaboratif (Dillenbourg et al., 1996) :

- L'approche socioconstructiviste.
- L'approche socioculturelle.
- L'approche de la cognition partagée

Ces approches apportent un regard différent et complémentaire à travers trois niveaux de compréhension : individuel, interindividuel et social.

L'approche socioconstructiviste

Cette approche est inspirée de la théorie de Piaget, "the individual-centred theory of development". Elle se concentre sur l'influence des interactions sociales sur le développement cognitif individuel et non pas sur les actions elles mêmes.

Ce développement est vu comme une spirale de causalité : pour participer à une interaction sociale, l'individu doit avoir un certain niveau cognitif. Une fois cette confrontation effectuée, l'individu acquiert de nouvelles compétences et peut participer à des interactions sociales plus sophistiquées etc.

Il est décrit en utilisant des concepts empruntés de la théorie de Piaget, qui sont : le conflit et les points de vue des apprenants (le terme utilisé pour ces points de vue est 'centration'). L'idée est d'induire un conflit sociocognitif entre les apprenants, c'est à dire un conflit entre différentes réponses basées sur différentes centrations.

Le paradigme expérimental utilisé consiste en deux phases supposées individuelles (pré et post-test) séparées par une session d'apprentissage par paires (ou en solitaire dans les conditions de contrôle). Cette dernière phase était caractérisée comme *conflit sociocognitif.* Cette théorie attache de l'importance au degré de différence entre les paires.

Les principaux tenants de cette approche admettent ces derniers temps que leurs travaux étaient probablement trop mécanistes (Perret-Clermont et al., 1991) et que le conflit socio-cognitif n'était pas le seul facteur de développement cognitif.

L'approche socioculturelle

Cette approche est basée principalement sur les travaux de Vygotsky (1978). Elle s'oriente autour de *l'activité sociale intériorisée par l'individu et qui conduit à l'apprentissage***.** Elle insiste sur la relation causale entre l'interaction sociale et les changements cognitifs (et non sur le développement individuel dans le contexte de l'interaction sociale).

Le développement est le résultat du processus d'internalisation qui représente le passage du plan social (interaction avec les autres) au plan interne (raisonnement) (Dillenbourg, 1999). Bref, il apparaît qu'une capacité se développe de prime abord de façon interpersonnelle pour ensuite s'intérioriser et devenir partie intégrante du fonctionnement cognitif personnel.

Elle se concentre sur les différences entre les apprenants. Ainsi, pour Vygotsky, il existe une zone proximale de développement chez chaque individu qui est définie comme suit :

> *« La distance entre son niveau de développement actuel, en regard de sa capacité à résoudre individuellement un problème, et son niveau de développement potentiel déterminé par la résolution guidée par un adulte ou en collaboration avec des pairs plus compétents ».*

Le mécanisme par lequel la participation à une résolution de problème commune peut modifier la compréhension du problème est nommé appropriation ou intériorisation.

L'approche de la cognition partagée

Elle est étroitement liée à la théorie de la *cognition située* qui met l'accent sur l'environnement. Ce dernier est considéré comme une partie intégrante de l'activité cognitive.
Elle explique l'apprentissage comme une activité faisant participer l'apprenant à un monde réel. Ainsi, l'apprentissage nécessite un contexte social et matériel authentique.

1.4. Les scénarios de collaboration

1.4.1. Définition d'un scénario

Le mot "scénario" est souvent utilisé dans les domaines artistiques, notamment cinématographiques. Dans le domaine de l'audiovisuel ou du théâtre, scénariser une histoire consiste à lui donner vie, c'est-à-dire à créer du mouvement, à passer à une vision multidimensionnelle pour faire vivre une expérience par le spectateur.

Pour Roche et Taranger (2001) (Henri et al., 2007),

> *« Le terme, emprunté à l'italien, désigne le récit verbal qui correspond à un film: ce que le film raconte avec des images et des sons, le scénario le raconte avec des mots. »*

Son utilisation est plus récente dans le domaine de l'éducation où le terme est complété par l'adjectif pédagogique, ou le complément de nom d'apprentissage.

1.4.2. Définition d'un scénario pédagogique

Un scénario pédagogique, selon Paquette, Crevier et Aubin (Paquette et al., 1998), est définit comme :

> *«La conjonction d'un scénario d'apprentissage et d'un scénario de formation qui lui est associé avec l'expression des liens qui les unissent* ».

Un scénario d'apprentissage est

> « *l'ensemble des activités destinées aux apprenants et organisées en un tout cohérent ; à ces activités, on greffe les instruments offerts comme supports aux activités (instruments-intrants) et les instruments à être réalisés par les apprenants (produits)* ».

Un scénario de formation est

> « *l'ensemble des activités destinées au formateur et organisées en un tout cohérent ; à ces activités, on greffe les instruments offerts comme supports aux activités (instruments-intrants) et les instruments à être réalisés par le formateur (produits)* ».

Ces définitions séparent les activités des apprenants de celles des formateurs, ce qui ne correspond pas toujours à la réalité, car ces activités sont liées le plus souvent au sein des scénarios pédagogiques.

1.4.3. Cycle de vie d'un scénario pédagogique

Certains travaux mettent l'accent sur le cycle de vie du scénario qui se décompose selon Pernin et Lejeune (Pernin et Lejeune, 2004) en :

- **La phase de conception initiale :** permet de définir d'une manière générale la structure d'une situation d'apprentissage. Le résultat de cette phase est un scénario abstrait qui ne tient pas précisément compte des conditions de mise en oeuvre (par exemple : la distribution des rôles à des personnes physiques, …etc.).

- **La phase de contextualisation :** permet de définir les conditions de mise en œuvre d'un scénario abstrait dans une situation précise et concrète de formation. Parmi les taches effectuées dans cette phase on trouve : l'affectation des rôles, la planification des activités, la médiatisation (création, réutilisation ou adaptation des ressources de manipulation de connaissances nécessaires à la réalisation des activités), l'instrumentation (création, réutilisation ou adaptation des outils et services nécessaires à la réalisation des activités), la localisation (consiste à rendre les ressources, outils et services concrets développés, réutilisés ou adaptés accessibles aux acteurs) et La concrétisation des composants abstraits. Le résultat de cette phase est un scénario contextualisé.

- **La phase d'exploitation :** correspond à la mise en œuvre du scénario contextualisé en situation effective d'apprentissage. Le résultat de cette phase peut être un scénario adapté ou descriptif. Un scénario adapté résulte des modifications faites au scénario initial contextualisé au cours du déroulement de la situation d'apprentissage.

- **La phase de retour d'usage :** correspond à l'évaluation des résultats obtenus lors de la phase d'exploitation des scénarios. Les objectifs principaux de cette phase sont l'évaluation de l'efficacité du scénario en termes didactiques et pédagogiques et l'extraction des conditions de sa réutilisation.

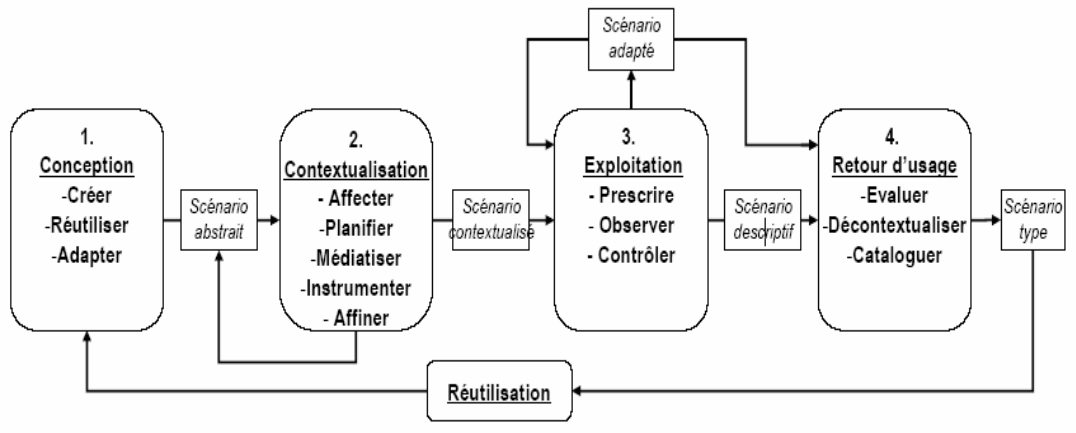


Fig.1. Les principales phases du cycle de vie des scénarios (Pernin et Lejeune, 2004).

1.4.4. Les Scripts de collaboration

Nous avons donc choisi la définition de Dillenbourg (Dillenbourg, 2002) qui présente un script collaboratif comme :

> *« un ensemble d'instructions décrivant comment les apprenants doivent former des groupes, doivent collaborer et comment ils doivent résoudre un problème ».*

Il inclut des activités individuelles (lecture, écriture,…), des activités de groupe (résoudre un problème avec un autre apprenant…) et des activités de classe (discussion...) (Dillenbourg, 2006).

Ces scripts sont groupés en deux classes: les macros scripts (the ArgueGraph) et les micros scripts (the social script) (Dillenbourg, 2002). Les macros scripts assument que les apprenants sont capables d'argumenter sans l'intervention des autres, alors que les micros scripts supportent le séquencement des différentes activités, le temps associés à chacune d'elles et le changement de rôle entre les apprenants.

Un exemple d'un script collaboratif: "Le script ArgueGraph "

ArgueGraph (Dillenbourg, 2002) est un "macro-script", c'est-à-dire son principe de base est de mettre en place des paires de façon particulière pour favoriser l'argumentation. Il a été souvent utilisé pour un enseignement face à face (en classe). Il engage les apprenants à trois niveaux d'activités: individuel, de groupe et de classe.

Il est basé sur un questionnaire à choix multiples élaboré par l'enseignant et se rapportant à un thème particulier. Pour chaque réponse de chaque question, l'enseignant détermine des valeurs X et Y qui seront utilisées pour présenter l'avis des étudiants dans un espace de deux dimensions. Ce script comprend les cinq étapes suivantes:

- **Phase 1 :** Chaque étudiant répond à un questionnaire de choix multiple en ligne. Il doit effectuer, pour chaque choix, un argument dans une zone de saisie de texte libre.

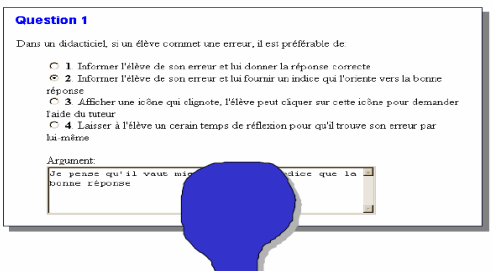


Fig.2. phase 1 du script (Schneider, 2007).

- **Phase 2 :** Dès que tous les étudiants ont répondu à toutes les questions, le système produit un graphe dans lequel tous les étudiants sont positionnés en fonction de leurs réponses. Un score horizontal et vertical est associé à chaque réponse du quiz et les positions des étudiants sont représentées par la somme de ces valeurs. Les étudiants observent le graphe et discutent ses résultats d'une manière informelle. Le système ou le tuteur forme des paires des étudiants qui ont la plus grande distance sur le graphe (c'est-à-dire : ceux qui présentent des opinions les plus divergents).

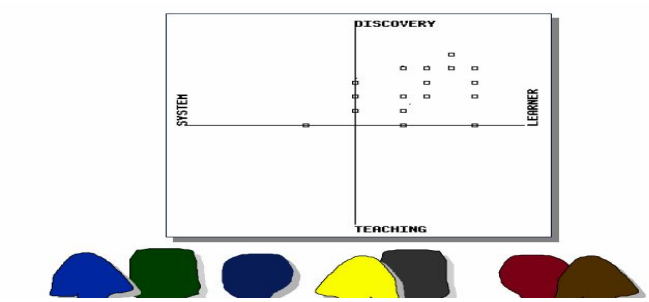


Fig.3. phase 2 du script (Schneider, 2007).

- **Phase 3 :** Les paires répondent au même questionnaire (comme dans la phase 1) et fournissent de nouveau des arguments. Ils peuvent lire leur réponse précédente

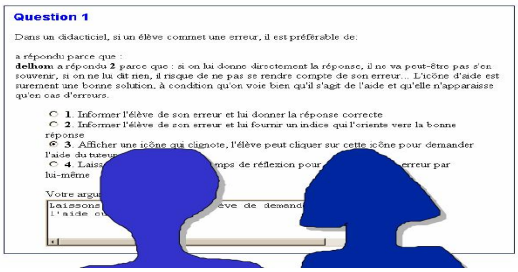


Fig.4. phase 3 du script (Schneider, 2007).

- **Phase 4 :** Dès que toutes les paires d'étudiants ont répondu à toutes les questions, le système calcule, pour chaque question, les réponses données individuellement (phase 1) et en collaboration (phase 3). L'enseignant / tuteur utilise ces résultats au cours d'une session de débriefing face-à-face et il demande aux étudiants de commenter leurs arguments. L'ensemble des arguments couvre plus ou moins le contenu du cours mais il est complètement non structuré. Ainsi le rôle de l'enseignant est d'organiser les arguments des étudiants en théorie, de les relier, de clarifier les définitions, en d'autres termes, de structurer les connaissances émergentes.

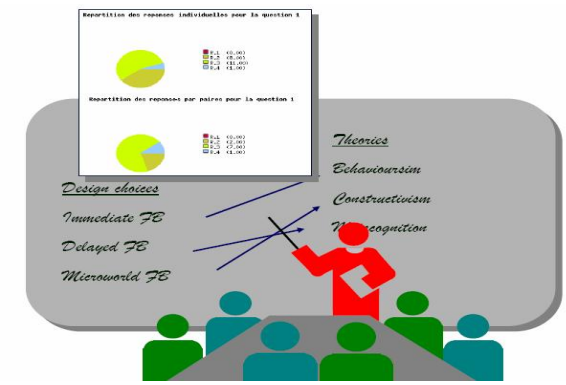


Fig.5. phase 4 du script (Schneider, 2007).

- **Phase 5 :** Chaque élève écrit une synthèse de tous les arguments collectés pour une question précise. La synthèse doit être structurée selon le cadre théorique mis en place par l'enseignant lors de la session de débriefing (phase 4).

Réflexions de Dillenbourg (2002)

Ce script a été utilisé avec succès à TECFA pour enseigner la relation entre les théories d'apprentissage et la conception de logiciels éducatifs. Il peut être généralisé aux domaines conceptuels dans lesquels de multiples théories co-existent.

• Le script intègre des activités face-à-face et en ligne.

• Il n'est pas collaboratif à 100%: il comprend une phase d'interaction entre paires (3), des phases individuelles (1 et 5) et une phase collective (4).

• Le but de la conception de ce script est de créer des conflits entre les étudiants et impliquer ces derniers dans des interactions pour résoudre le conflit.

• Ils ont testé deux versions de ce script, l'une où tous les étudiants étaient dans la salle informatique et l'autre où le système est utilisé à distance pour les phases 1 à 3. Les deux versions utilisent des environnements CSCL différents. L'environnement utilisé pour la deuxième version ne fonctionne pas très bien pour deux raisons. La première est que l'interface utilisée pour la phase 3 permet aux étudiants d'éviter la résolution des conflits en calculant leur degré associé à chacune des propositions au lieu d'être forcés à choisir une et une seule proposition. La deuxième raison est que les paires qui ont argumenté leurs propositions (phase 3) tôt avant la session de débriefing (phase 4) sont moins impliqués à cette session par apport à ceux qui ont fait leurs argumentations juste avant la session.

Remarque : L'efficacité de ce script n'est pas seulement influencée par le choix des activités, mais aussi par des facteurs tels que l'ergonomie de l'environnement (Dillenbourg, 2002), le calendrier des phases ainsi que la qualité du questionnaire.

1.5. Définition du concept de scénarisation

La notion de scénarisation est apparue assez récemment dans les travaux de recherche. Elle est considérée comme le processus d'élaboration d'un scénario pédagogique destiné à être utilisé et manipulé dans un contexte d'apprentissage, soit par un autre enseignant, soit par des apprenants.

France Henri (Henri et al., 2007) définit la scénarisation comme

> *« La scénarisation est avant tout un travail de conception de contenu, d'organisation des ressources, de planification de l'activité et des médiations pour induire et accompagner l'apprentissage, et d'orchestration, c'est-à-dire d'intégration des contributions des différents spécialistes qui travaillent à la conception et à la réalisation du scénario dans l'environnement. »*

Selon Gilbert Paquette (Paquette, 2007) La scénarisation pédagogique est

> *« Ce processus central en ingénierie pédagogique qui vise à définir une organisation des activités d'apprentissage et de facilitation des apprentissages. Le produit de la scénarisation, le scénario pédagogique ou d'apprentissage, est*

fondé sur un modèle en relation multiple avec divers concepts donc ceux de méthode d'apprentissage, de stratégie et tactique d'enseignement, de plan de cours, de processus de travail pour l'apprentissage. »

Le concept de scénarisation emprunte des termes à des domaines artistiques (cinéma, musique etc …). Elle est associée à la dimension créative, composante essentielle du travail de l'enseignant et vise à la conception de situation d'enseignement et d'apprentissage.

1.6. Approches de Conception des scripts de Collaboration

La conception des scripts de collaboration est un processus complexe. Afin d'aider le concepteur dans sa tâche, plusieurs approches ont été proposées. Parmi ces dernières on présente, une approche basée sur les patterns pour la conception des scripts de collaboration et la méthode des pléiades.

1.6.1. Une approche basée sur les patterns d'apprentissage pour la conception des scripts de collaboration

Le travail de Davinia Hernández-Leo et ses collègues (Hernández-Leo et al ,2006) positionne et présente les différents types de patterns qui peuvent être utilisés pour générer des scripts de collaboration. Ces patterns peuvent être représentés et interprétés par des systèmes de gestion d'apprentissage (*Learning Management Systems (LMSs)).*

Afin de faciliter le processus de compréhension et d'application de ces patterns pour la conception des scripts, une structure hiérarchique de ces derniers est proposée. Cette structure lie les différents types de patterns en fonction de leur granularité et leur intérêt. Cette structure est représentée par la figure suivante Fig.6.

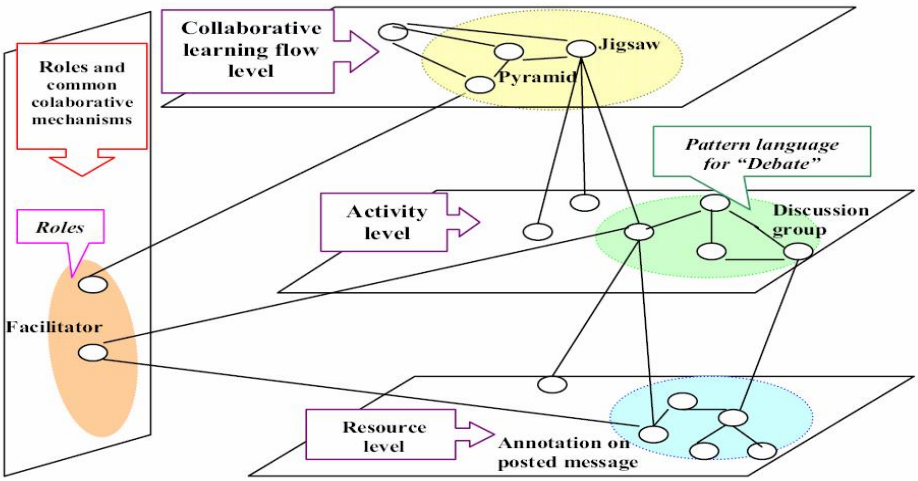


Fig.6. La structure hiérarchique des patterns de conception (Hernández-Leo et al ,2006).

Il y a trois types de granularité qui correspondent aux trois niveaux horizontaux de la structure proposée. Les patterns de ces niveaux sont liés verticalement. Toutefois il y a

certains aspects qui peuvent être connectés directement à certains patterns à n'importe quel niveau de granularité. Ces aspects représentent les rôles et les mécanismes communs de collaboration, tel que : la formation de groupe, la distribution des rôles / ressources, la sensibilisation…etc. Par exemple, les rôles peuvent être définis au niveau du flux d'apprentissage, dans les activités et / ou dans les outils de collaboration. Ainsi, les patterns qui introduits ces aspects sont représentés à un niveau vertical.

Les niveaux horizontaux de cette structure sont comme suit :

- **Le niveau de flux d'apprentissage collaboratif :** représente la séquence des activités du processus d'apprentissage. Parmi les patterns de ce niveau, on trouve Jigsaw, pyramide et CLFPs.
- **Le niveau des activités :** représente les activités elle mêmes. Un exemple de pattern de ce niveau est le Groupe de discussion.
- **Le niveau de ressources :** comprend les ressources (matériaux et outils) nécessaires pour soutenir les activités. Un exemple de patterns est l'annotation sur les messages envoyés.

1.6.2. La méthode des Pléiades

Elle repose sur la métaphore astronomique des Pléiades. Elle vise à soutenir la conception des scénarios structurés et réutilisables (David et al., 2007).

Les Pléiades sont un regroupement d'étoiles visuellement identifiables, construisant une forme cohérente qu'on peut nommer et lui attribuer un sens.
L'intérêt de cette image est qu'elle offre un cadre conceptuel pour la construction d'un scénario pédagogique selon différents niveaux d'activités. Ainsi les adeptes de cette méthode ont défini trois niveaux de granularité (David et al., 2007):

- **Une étoile :** représente une activité élémentaire.
- **Une pléiade :** représente un regroupement de plusieurs activités.
- **Une constellation :** représente un regroupement de plusieurs pléiades.

Cette méthode repose sur le principe sous-jacent de regroupement signifiant d'activités. Ainsi, un scénario pédagogique est vu comme un ou plusieurs regroupements d'activités qui peut être nommé et qui a un sens par rapport à la situation d'apprentissage et aux intentions pédagogiques des enseignants.

Ces regroupements sont reliés entre eux par des liens de précédence, d'hiérarchie, de simultanéité, de résonance, etc. A chaque regroupement est associé un ensemble de propriétés auxquelles il est possible d'attribuer des valeurs. Ces propriétés permettent de décrire certaines caractéristiques de l'entité concernée (pléiade ou constellation) telles que : la granularité (par ex., pléiade), le famille (par ex., évaluation diagnostique), le statut dans le scénario (optionnel), les activités constituantes, les stratégies d'enseignement et d'apprentissage (stratégie interactive/coopération-gestion de conflit) et les éléments de connaissances.
Une autre partie des propriétés permet d'organiser le scénario de façon dynamique autour d'un pivot central : les activités « constituants ». Elle permet d'envisager «l'orchestration » (avec ordre, sans ordre etc.), « les conditions de clôture » (choix de l'apprenant, choix de

l'enseignant, contrainte temporelle) et la « distribution » entre les acteurs (élèves, enseignants, élèves-élèves, élèves-expert), et entre les éléments supports (ressources utilisées, ressources produites, outils utilisés).

1.7. Le suivi des traces

1.7.1. Définition

Dans le contexte des EIAH, les traces sont des données issues de l'observation directe ou indirecte permettant la régulation, le contrôle, l'analyse et la compréhension de l'activité d'apprentissage. Une trace est définie comme :

« Une séquence temporelle d'observés ».

La *séquence temporelle consiste en* l'existence d'une relation d'ordre organisant les données de la trace par rapport à un repère de temps.

Une autre définition (Settouti et al., 2005) décrit les traces comme :

> *« Des données issues d'observations directes ou indirectes de marques laissées durant l'activité d'apprentissage par un apprenant ou un groupe d'apprenants permettant la régulation, le contrôle, l'analyse et la compréhension de l'activité d'apprentissage. »*

1.7.2. Pourquoi suivre les traces des apprenants

Le suivi des apprenants joue un rôle important pour l'apprentissage humain. Il existe plusieurs usages des traces parmi lesquels on trouve (Settouti et al., 2005):

- L'analyse des traces permet le contrôle et la régulation de l'activité d'apprentissage.
- Les traces offrent aux tuteurs et apprenants la possibilité d'être conscients de l'activité d'apprentissage.
- la compréhension de la réussite ou de l'échec des apprenants afin que le tuteur et l'apprenant comprennent leurs activités.
- L'observation en se basant sur des traces est aussi un important facteur pour la qualité du scénario d'apprentissage.
- Les traces sont utilisées pour améliorer le scénario pédagogique, elles permettent l'évaluation du scénario prédictif par rapport au scénario effectif décrit par ces dernières.
- Elles sont utilisées pour la réingénierie des EIAH dans de nombreux travaux.

1.8. Les approches de suivi des traces des apprenants

Parmi les approches de suivi des traces, on présente : l'approche MUSETTE et le système à base de traces.

1.8.1. L'approche MUSETTE (Modelling USEs and Tasks for Tracing Experience)

Dans cette approche (Champin et al., 2004), la trace d'utilisation d'un système informatique est structurée en état-transition et est exploitée afin d'être réutilisée selon le principe du Raisonnement à Partir de Cas.
Les états représentent les entités, et les transitions représentent les évènements observés entre deux états.

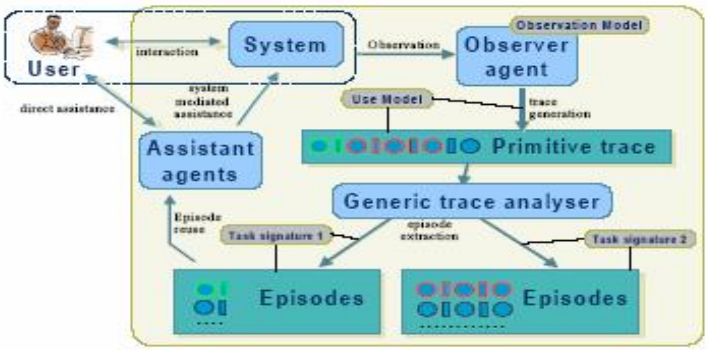


Fig.7. Approche Musette (Champin et al., 2004).

Les composants principaux de cette approche sont :

- L'agent observateur
- Un analyseur générique de trace
- Agents assistants

L'agent observateur

Il permet de générer à partir des interactions des utilisateurs avec l'environnement (système) une trace primitive. Il utilise un modèle d'utilisation et un modèle d'observation.

Un modèle d'utilisation est l'ensemble des éléments (entités, évènements et relations) permettant la construction d'une trace sur la base des interactions avec l'environnement informatique. Ce modèle est construit à partir de l'observation des utilisateurs dans leurs pratiques de l'environnement.

Les entités sont des objets utilisés par l'utilisateur pendant son interaction ; les évènements sont des objets qui 'ont lieu' durant cette même interaction ; les relations peuvent lier indifféremment des entités et/ou des évènements.

Un modèle d'observation décrit les règles nécessaires pour déterminer les données pertinentes (issues de l'environnement) et pour construire la trace primitive. Il n'est pas spécifié dans l'approche Musette et doit être adapté à chaque environnement observé.

Un analyseur générique de trace

Il permet d'extraire les épisodes significatifs de la trace primaire en utilisant les signatures de tâches expliquées (SiTEx).

Un épisode est une partie de la trace, qui correspond à une expérience spécifique en effectuant une tache spécifique. Cette partie peut être réutilisée dans une situation similaire. Les signatures de tâches expliquées permettent de déterminer les parties qui sont considérées comme des épisodes.

Agents assistants

Ils (ré) utilisent ces épisodes pour assister les apprenants directement ou indirectement (l'assistance est via le système). Dans le deuxième cas, l'agent observateur peut observer les interactions de cette assistance.

1.8.2. Un système à base de traces (SBT)

Les systèmes à base de trace découlent d'une approche plus générale que MUSETTE permettant de représenter un cadre pour l'exploitation des traces. Un système à base de traces (SBT) est un système informatique permettant et facilitant la manipulation des traces (Settouti et al., 2005).

Une trace dans un SBT est considérée comme une séquence temporelle d'observés accompagnée de son modèle de trace. Ce modèle est un ensemble d'objets étiquetés représentant le vocabulaire de la trace (par exemple dans l'approche MUSETTE, le modèle d'utilisation de la trace primitive est une ontologie exprimée en RDF structurant le vocabulaire de la trace).

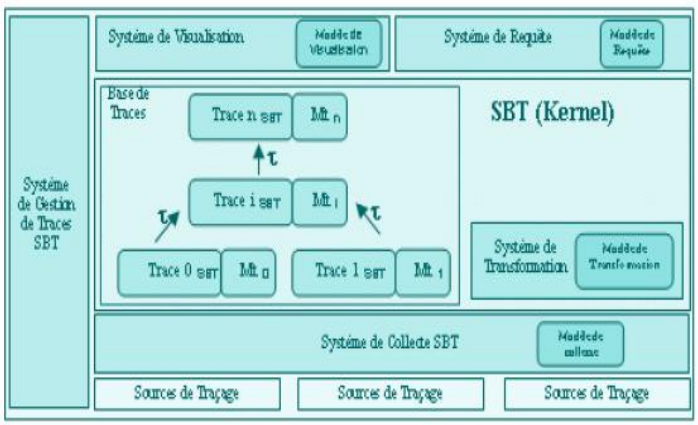


Fig.8. Architecture du système à base de traces (Settouti et al., 2005).

Ce système est constitué des composants suivants :

- Le SBT (Kernel).
- Le système de collecte.

- Le système de requêtes.
- Le système de visualisation.
- Le système de gestion des traces.

Le SBT (Kernel)

Il représente le noyau du système. Il est constitué d'un système de transformation et d'une base de traces. Cette base permet le stockage des traces du SBT et de leurs modèles.

Le système de transformation permet la modification des données des traces, la modification de leurs modèles, la mise à jour de la base de traces ou bien de faire des transformations automatiques en utilisant un modèle de transformation (un ensemble de règles formelles).

Le système de requêtes

Ce système permet d'interroger la base de traces. Les requêtes peuvent concerner une ou plusieurs traces. Il permet de partager des modèles de requêtes entre différents utilisateurs.

Le système de collecte

C'est un ensemble structuré de processus en contact direct avec les sources de traçage. Une source de traçage est un fichier ou un flux de données dans un format explicite quelconque. Dans le SBT, ces sources deviennent des traces lorsque elles sont associées avec un modèle de trace. Ce système utilise des outils pour convertir les données en traces qui sont souvent obtenues d'une manière itérative afin de l'améliorer.

Le système de visualisation

Il permet de visualiser les traces et les traces transformées afin de faciliter leur analyse et interprétation, ainsi que l'accès aux sources de traçage à partir des traces pour interroger celles-ci plus directement.

Le système de gestion des traces

Il gère les différents modèles du système (modèles de trace, de transformation, de requête). Il permet l'ajout, la suppression des traces, ainsi que leurs conservations et leurs administrations.

1.9. Utilisation des agents et les systèmes multi-agents dans les domaines de formation

La complexité inhérente à la conception des EIAH a induit l'utilisation de différents paradigmes dont celui des agents. Selon Ferber un système multi-agents (Ferber, 1995) est un système composé d'un environnement, d'un ensemble d'objets qui peuvent être perçus, créés, détruits et modifiés par les agents, d'un ensemble d'agents qui représentent les entités actives du système, d'un ensemble de relations qui unissent des objets/des agents entre eux, d'un ensemble d'opérations permettant aux agents de percevoir, produire,

consommer, transformer et manipuler des objets et des opérateurs chargés de représenter l'application de ces opérations et la réaction du monde à cette tentative de modification (appelés lois de l'univers).

L'utilisation des agents et systèmes multi-agents dans les domaines de formation, l'enseignement et l'apprentissage est intéressante et prometteuse. Ces agents sont capables de s'organiser, coopérer et coordonner leurs actions afin de supporter l'apprentissage.

Les SMA ont été utilisées dans le domaine de l'enseignement à distance (Pesty, 2003). Certains travaux se sont intéressés aux agents 'assistants' dits aussi 'compagnons' des acteurs (apprenant, tuteur, auteur) dont la tâche est de faciliter l'accès et l'usage du système. Des agents dits tuteurs ont été proposés pour l'adaptation des activités aux profils des apprenants. Des agents de support à la collaboration dans l'apprentissage en groupe pour encourager la participation des apprenants et facilitent la discussion entre les membres du groupe.

1.10. Conclusion

La collaboration présente des avantages certains pour l'apprentissage. La recherche sur l'apprentissage collaboratif traite notamment des interactions entre apprenants. Ces interactions peuvent avoir lieu entre les apprenants ou bien entre les apprenants et les environnements informatiques pour l'apprentissage humain (EIAH).
Ces environnements informatiques sont conçus de façon à favoriser l'apprentissage humain. Ce type d'environnement permet à ses utilisateurs l'interaction avec d'autres utilisateurs ainsi que l'interaction avec des agents artificiels.

Afin de tirer un meilleur profit de cette collaboration, on utilise les scripts de collaboration qui permettent de mieux l'organiser et de la structurer. Ainsi le suivi des apprenants permet de comprendre les comportements des apprenants afin d'assurer un meilleur apprentissage.

Pour cela on pense que l'utilisation des agents et les SMA peuvent apporter un plus à la conception des scripts de collaboration.

Chapitre 2

Les Agents Artificiels

2.1. Introduction

La recherche sur les Systèmes Multi-agent (SMA) est un domaine important et évolue rapidement dans les domaines des systèmes distribués et l'intelligence artificielle.

La recherche sur les systèmes à base d'agents peut être classée en général en deux volets (Iglesias et al., 1999), le premier volet représente la recherche dans les SMA, qui se concentre sur le système dans son ensemble. Par exemple, la conception du SMA a donné naissance à diverses méthodologies (Gaia, TROPOS). Le deuxième volet concerne les agents comme des individus : il se concentre sur l'agent comme individu, sur son architecture interne et sur les différentes approches à mettre en œuvre, pour son implémentation. L'une des oeuvres intéressantes de ce courant est le développement des langages de programmation agents.

Afin de développer les systèmes multi-agents, de nombreux langages de programmation ont été proposé pour implémenter des agents individuels, ainsi que leur environnement et leurs interactions. Dans ce chapitre on présente la notion d'agent, les différentes théories, architectures, langages de programmation agents et quelques langages qui ont été proposés.

2.2. Agents

2.2.1. Définitions

Michael Wooldridge and Nicholas R. Jennings (Wooldridge & Jennings, 1995a) proposent deux définitions générales pour l'agent intelligent : une définition faible et l'autre forte.

La définition faible

La définition faible, proposée par N.R. Jennings, K.Sycara and M. Wooldridge (Jennings et al., 1998), est considérée comme la plus générale. Elle décrit un agent comme :

> *« ... un système informatique, situé dans un environnement, et qui agit d'une façon autonome et flexible pour atteindre les objectifs pour lesquels il a été*

conçu. Par 'situé', l'agent est capable d'agir sur son environnement à partir des entrées sensorielles qu'il reçoit de ce même environnement. 'Autonome', l'agent est capable d'agir sans l'intervention directe des humains (ou d'autres agents) et de contrôler ses propres actions ainsi que son état interne. […] Par 'flexible': le système est :
- Réactif: l'agent doit être capable de percevoir son environnement et élaborer une réponse dans les temps requis.
- Proactif: l'agent doit exhiber un comportement proactif et opportuniste, tout en étant capable de prendre l'initiative au bon moment.
- Social: l'agent doit être capable d'interagir avec les autres agents (logiciels et humains) quand la situation l'exige. »

La définition forte

Pour certains chercheurs, et spécialement pour ceux qui travaillent dans le domaine de l'intelligence artificielle, un agent intelligent est un système informatique qui a les propriétés définies précédemment dans la définition faible et qu'il est conçu ou implémenté en utilisant des concepts qui sont le plus souvent appliquée à l'homme (Wooldridge & Jennings, 1995a). Par exemple, en utilisant des notions comme la connaissance, la croyance, l'intention, et l'obligation. D'autres auteurs comme Bates (1994) sont allés plus loin en envisageant des agents émotionnels.

2.2.2. Autres attributs d'agents

Les chercheurs ont parfois recours à d'autres attributs des agents (Wooldridge & Jennings, 1995a) pour atteindre les objectifs de leurs travaux. Par exemple:

- Mobilité: l'agent est capable de se déplacer dans un réseau électronique.
- Véracité: l'agent ne communique pas consciemment des informations erronées.
- Bénévolat: les agents n'ont pas de conflits d'objectifs et chaque agent essaye toujours de satisfaire la demande des autres.
- La rationalité: l'agent agira en vue d'atteindre ses objectifs et d'éviter toutes les actions qui l'empêchent de les atteindre dans la mesure où ses croyances le permettent.

2.3. Théories d'agents

2.3.1. Définition

La théorie des agents (Wooldridge & Jennings, 1995a) est essentiellement une spécification pour l'agent. Elle concerne la conception d'agents, leurs propriétés, et les formalismes mathématiques. Ces derniers sont utilisés pour avoir une représentation formelle et permettre un raisonnement sur les propriétés de l'agent.

Les théoriciens ont développés des formalismes, puis d'utiliser ceux-ci afin de développer des théories représentatives des propriétés que l'on associe aux agents.

2.3.2. Classification des théories d'agents

Selon M.Wooldridge, NR. Jennings (Wooldridge & Jennings, 1995b), les théories des agents sont classés en quatre catégories qui sont :

- Les agents en tant que systèmes autonomes rationnels.
- Les agents en tant que systèmes basé–actions.
- Les agents en tant que systèmes de haut niveau.
- Les agents en tant que systèmes intentionnels

Les agents en tant que systèmes autonomes rationnels

Les agents sont des entités actives qui affectent d'une manière autonome et rationnelle leur environnement. L'autonomie signifie généralement que l'agent agit sans l'intervention des humains ou d'autres agents. La rationalité signifie que l'agent agit d'une manière à maximiser ses performances en utilisant une fonction d'évaluation.

Cette théorie est considérée comme faible car elle est fondée sur un critère faible, acceptant ainsi, une large classe d'objets comme des agents. Selon cette définition, un transistor (la forme la plus simple d'un commutateur électronique) peut être considéré comme un agent autonome et rationnel.

Les agents en tant que systèmes basé–actions

Cette théorie se concentre sur les différentes actions des agents. Sa principale difficulté est la notion d'action qui peut être décrite selon différentes manières où chacune semble être valide.

Afin de simplifier cette notion, les pionniers de cette théorie essayent de décrire les actions en termes de liens de causalité. Cette proposition introduit un problème de régression infinie.
Pour expliquer ces liens et cette régression infinie, on considère l'exemple suivant : lorsque on fait un geste du bras à un ami, on soulève le bras (l'action est causée par la contraction des muscles).Cette contraction est le résultat des actions de certains neurones et ainsi de suite.

Les agents en tant que systèmes de haut niveau

Shoham pense que le terme "agent" dans l'IA est souvent utilisé pour désigner les systèmes de haut niveau. Un tel système emploie des représentations symboliques et peut utiliser des fonctions cognitives (comme le raisonnement logique explicite).
Il implique que les agents possèdent des ressources de calcul (un nombre fini de ressources).

Toutefois, de nombreux chercheurs pensent que cette théorie discrimine des systèmes qui n'emploient pas explicitement ces fonctions. Ainsi ce n'est pas le meilleur moyen pour présenter l'IA. Brooks, un des protagonistes de cette théorie, argumente que les agents non qualifiés de 'haut niveau' peuvent exécuter des tâches importantes de l'IA.

Les agents en tant que systèmes intentionnels

Cette théorie est largement répandue. Elle considère un agent comme une entité "qui semble faire l'objet de croyances, de désirs, etc " (Seel, 1989, p. 1), (Wooldridge & Jennings, 1995b). Le philosophe Dennett a inventé le terme système intentionnel pour désigner de tels systèmes (Wooldridge & Jennings, 1995b).

On s'est alors légitimement demandé si il était pertinent ou utile d'attribuer des croyances, des désirs, etc. à des agents artificiels. McCarthy (1978) (Wooldridge & Jennings, 1995b) fut parmi ceux qui considérèrent que cela pouvait être approprié notamment pour des entités dont la structure n'est que partiellement connue. Il est possible, bien que cela reste un peu superficiel, d'attribuer de telles attitudes même à des entités rudimentaires dont le comportement peut être décrit par des automates. Il apparaît plus approprié de décrire un système complexe d'un point de vue intentionnel ; les notions intentionnelles devenant des abstractions utiles nous fournissant des moyens pratiques et familiers pour décrire, expliquer et prédire le comportement de ces systèmes. Il convient alors de se demander quelles attitudes sont les plus appropriées pour les agents.

Certaines des plus importantes théories intentionnelles sont présentées dans les sous sections suivantes :

- **Moore – connaissance et action**

La préoccupation principale de Moore est l'étude des pré-conditions pour les actions. Il met l'accent sur les connaissances qu'un agent a besoin de savoir afin de pouvoir faire une action.
Il formalisa un modèle de capacité basé sur une logique dotée d'une modalité pour les connaissances et sur une logique dynamique pour la modélisation de l'action. Ce formalisme permet à un agent d'accomplir ses tâches, en ayant des informations incomplètes sur la manière d'atteindre certains objectifs et de réaliser des actions.

- **Cohen et Levesque - intention**

Au début, Cohen et Levesque étaient intéressé par le développement d'un formalisme qui a servi à élaborer la théorie de l'intention (comme dans «J'ai l'intention de…»), qui est utilisée comme un pré-requis pour une théorie des actes de langage. Cependant, la logique s'étant révélée utile pour raisonner sur les agents, elle a été utilisée pour l'analyse des conflits et la coopération dans les dialogues entre agents.

Ils identifient sept propriétés qui doivent être satisfaites par une théorie raisonnable de l'intention:

1. les intentions posent des problèmes aux agents, qui doivent déterminer les moyens pour les atteindre.
2. les intentions constituent des filtres pour l'adoption d'autres intentions qui ne doivent pas entrer en conflit
3. les agents surveillent la satisfaction de leurs intentions, et sont tentés d'insister lorsqu'elles échouent
4. les agents croient que leurs intentions sont possibles.

5. les agents ne croient pas qu'ils ne pourront pas satisfaire leurs intentions
6. sous certaines circonstances, les agents croient qu'ils vont satisfaire leurs intentions
7. les agents n'ont pas besoin d'avoir des intentions portant sur les effets secondaires attendus de leurs intentions.

- **Rao et Georgeff – les architectures BDI (belief, desire, intention)**

Rao et Georgeff ont développé leur théorie sur la base de trois modalités primitives : les croyances, les désirs et les intentions. Leur formalisme est basé sur un modèle de branchement temporel, dans lequel la croyance, le désir et l'intention sont eux-mêmes des structures de branchement temporels.

Ils se sont particulièrement préoccupés par la notion de réalisme, c'est-à-dire la question de savoir comment les croyances d'un agent influent ses désirs et ses intentions. Dans un autre travail, ils ont également envisagé la possibilité d'ajouter à leur formalisme des plans (sociaux).

- **Singh**

Singh a développé une famille de logiques pour représenter les intentions, les croyances, les connaissances, le savoir faire et la communication dans un framework basé sur les branchements temporels.
Le formalisme de Singh, bien qu'il soit complexe, est extrêmement riche, et des efforts considérables ont été consacrés à la mise en place de ses propriétés.

- **Werner**

Werner a présenté les bases d'un modèle général de l'agence en s'appuyant sur les travaux en économie, la théorie des jeux, théorie des automates situés, sémantique de situation et la philosophie. Toutefois, les propriétés de ce modèle n'ont pas été étudiées en profondeur.

- **Wooldridge – modélisation des systèmes multi-agent**

Wooldridge vise à construire des formalismes qui pourraient être utilisés dans la spécification et la vérification de systèmes multi-agents réalisables. Il a développé un modèle simple et général de système multi-agents, et a montré comment l'historique de l'exécution de tels systèmes pouvait être utilisé comme le fondement sémantique d'une famille de logiques linéaire et de branchement temporels.
Il a ensuite donné des exemples d'utilisation de ces logiques dans la spécification et la vérification de protocoles de coopération.

2.4. Architectures d'agents

2.4.1. Définition

Les architectures agents (Wooldridge & Jennings, 1995a) représentent le passage de la spécification à l'implémentation. Elles sont utilisées pour construire des systèmes qui

satisfont les propriétés spécifiées par les théoriciens. Elles s'intéressent à l'identification des logiciels appropriés et /ou aux structures matérielles à utiliser.

Maes (1991) definit une architecture d'agent (Wooldridge & Jennings, 1995a) comme suit:

> *« Une architecture spécifie comment l'agent peut être décomposé en un ensemble de modules et comment ces modules peuvent interagir. L'ensemble de ces modules et de leurs interactions doit fournir une réponse à la question suivante : comment à partir des données de ses senseurs et de son état interne courant l'agent détermine ses actions et son état interne futur ? Une architecture comprend des techniques et des algorithmes visant à répondre à cette problématique. »*

Pour Kaelbling (1991) (Wooldridge & Jennings, 1995a), une architecture d'agent est:

> *« Une collection spécifique de modules logiciels (ou matériels), généralement représentés par des boîtes et des flèches indiquant les donnés et les flux de contrôles entre ces modules. Une vision plus abstraite d'une architecture est celle d'une méthodologie générale permettant de concevoir des décompositions modulaires particulières pour des tâches particulières.»*

2.4.2. Architectures concrètes d'agents

Wooldridge (Wooldridge, 1999) classe les architectures agent en quatre catégories:

- les architectures basées sur la logique
- les architectures réactives
- les architectures BDI (Belief-Desire-Intention)
- les architectures en couche

Les architectures basées sur la logique

Ces architectures sont basées sur l'approche " traditionnelle " de la construction des systèmes artificiels intelligents. Cette approche suggère une représentation symbolique du comportement d'agent et de son environnement, ainsi qu'une manipulation syntaxique de cette représentation. La prise de décision est le fruit d'une déduction logique

Dans cette approche la représentation symbolique est un ensemble de formules logiques et la manipulation syntaxique correspond à la déduction logique ou à la preuve du théorème.

- **Avantages**

- Les approches basées sur la logique sont élégantes et ont une sémantique propre.

- **Inconvénients**

Les approches basées sur la logique ont beaucoup de problèmes.

- La complexité de la preuve du théorème a une influence sur l'efficacité de l'agent, en particulier dans les environnements à fortes contraintes temporelles.
- L'utilisation de l'hypothèse de rationalité pour la prise de décision signifie que le monde ne changera pas de manière significative quand l'agent est en train de décider de ce qu'il faut faire, et qu'une action qui est rationnelle lors de la prise de décision, sera rationnelle quand elle se termine.
- Les problèmes liés à la représentation des environnements complexes, dynamiques, et physique ainsi que le raisonnement sur ces derniers sont essentiellement non résolus.

Les architectures réactives

La complexité de la première classe d'architectures a conduit de nombreux chercheurs à rejeter les approches fondées sur la logique, en plus certains thèmes nécessitent:

- Le rejet de représentations symboliques, et de la prise de décision fondée sur la manipulation syntaxique de ces représentations.
- L'idée que l'intelligence de l'agent est vue comme étant le résultat de ses interactions avec l'environnement car les agents sont situés dans un environnement, et ils ne sont pas dissociés de ce dernier.
- L'idée que le comportement intelligent émerge de l'interaction de divers comportements simples.

Dans ces architectures, la prise de décision est mise en oeuvre dans une forme directe du passage de la situation à l'action.

- **Avantages**

- Simplicité, économie, robustesse contre l'échec, et l'élégance.

- **Inconvénients**

- Les agents purement réactifs prennent des décisions fondées sur des informations locales, (c'est-à-dire, des informations sur l'état actuel des agents). Ils ne tiennent pas compte des informations non locales.
- Si les agents n'ont pas de modèles de l'environnement, ils doivent avoir des informations suffisantes, disponibles dans leurs environnements locaux pour pouvoir déterminer une action acceptable.
- La prise en compte des expériences dans la conception de ces agents afin d'améliorer leurs performances au fil du temps.

Les architectures BDI (Belief – Desire - Intention)

Les architectures BDI sont des architectures de raisonnement pratique, dans lequel la prise de décision dépend de la manipulation des croyances, des désirs, et des intentions de l'agent. La décision ressemble au raisonnement humain.

Les éléments de base d'une architecture BDI sont des structures de données représentant les croyances, les désirs, et les intentions de l'agent, ainsi que les fonctions qui représentent

ses délibérations (décider de l'intention à prendre) et le raisonnement moyens-fins (décider comment faire).

- **Avantages**

- Intuitif : on connaît le processus quoi faire et comment le faire. On a une idée de ce que sont les notions de croyances, désirs et intentions
- Une décomposition fonctionnelle claire indiquant quels sont les sous systèmes nécessaires pour construire l'agent.

- **Inconvénients**

- L'inconvénient majeur est comment implémenter d'une manière efficace ces fonctions.

Les architectures en couches

Dans cette classe d'architecture, les différents sous-systèmes sont organisés en une hiérarchie de couches en interaction. Généralement, il y aura au moins deux couches, l'une pour le comportement réactif et l'autre pour le comportement pro-actif.

En se basant sur le flux de contrôle au sein de ces architectures, on peut identifier deux catégories:

• **Architecture en couches horizontales** : Dans cette architecture (voir Fig.9.), les couches logicielles sont chacune reliées aux entrées des capteurs et aux sorties des actions. En effet, chaque couche se comporte comme un agent et produit des propositions sur le type d'actions à exécuter.

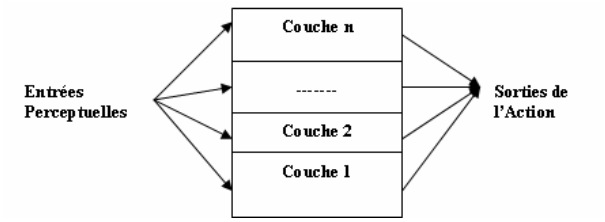


Fig.9. Architecture en couches horizontales (Wooldridge, 1999).

• **Architecture en couches verticales** : Dans cette architecture l'entrée sensorielle et la sortie des actions sont pris en charge par, au plus un agent (voir Fig.10.).

Ces architectures peuvent être classées en deux catégories:

- **Architectures verticales à une passe**: dans ces architectures, le flot de contrôle passe séquentiellement par chaque couche, jusqu'à ce que la couche finale génère l'action de sortie.
- **Architectures verticales à deux passes**: dans ces architecture, les informations circulent dans un sens (de bas en haut : une passe) et le flot de contrôle revient en sens inverse.

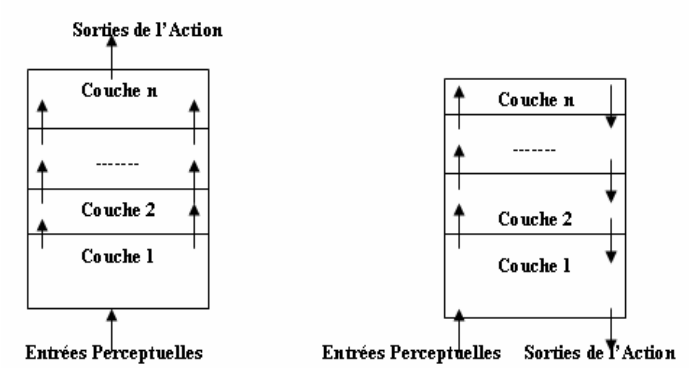


Fig.10. Architecture en couches verticales (Wooldridge, 1999).

- **Avantages**

- Les architectures en couches sont populaires et disponibles.
- Une décomposition naturelle des fonctionnalités: il est facile de voir comment les comportements réactif, pro-actif et social sont générés par les couches réactive, pro-active et sociale.

- **Inconvénients**

- Ces architectures n'ont pas la même clarté sémantique et conceptuelle des architectures sans couches (plates).
- L'interaction entre les couches (partiellement résolu par l'architecture en couches verticale à deux passes comme dans INTERRAP) nécessite la prise en considération de toutes les possibilités d'interaction entre ces couches.

2.5. Langages d'agents

2.5.1. Définition

Les langages agent (Wooldridge & Jennings, 1995a) sont des systèmes logiciels utilisés pour la programmation des agents. Ces langages devraient offrir les primitives appropriées pour cette tâche, et ils peuvent aussi inclure des principes proposés par les théoriciens.

2.5.2. Les caractéristiques des langages de programmation d'agents

M. Martelli, V. Mascardi, et S. Sterling dans (Martelli et al., 1995) pense que les langages de programmation devraient fournir au moins certains éléments, qui sont: le temps, la perception, la communication, la concurrence, le non-déterminisme, la modularité, et la sémantique.

- **Le temps**: Les agents doivent réagir aux changements qui se produisent dans leur environnement et planifier leur futur dans un temps raisonnable.
- **La perception:** les agents doivent être capables de percevoir leurs environnement.

- **Communication:** les agents peuvent communiquer avec d'autres agents ou êtres humains.
- **Concurrence :** les langages de programmation devrait supporter la concurrence des threads internes du même agent, la concurrence entre les agents ou bien les deux en même temps.
- **Non-déterminisme:** dans les systèmes multi-agent, on ne peut pas déterminer la succession des événements.
- **Modularité:** Structurer le programme d'agent en modules ou en procédures est plus simple et utile pour le développeur.
- **Sémantique:** l'utilisation d'une sémantique clair permet au développeur de comprendre la signification des primitives du langage afin de l'exploiter de manière utile.

2.5.3. Classification des langages d'agents

Les langages de programmation agent peuvent être classés en trois catégories (Bordini et al., 2005) :

- Les langages déclaratifs.
- Les langages Impératifs.
- Les langages hybrides.

Les langages déclaratifs

Les langages déclaratifs (Bordini et al., 2005) sont généralement caractérisés par leur nature formelle, basée sur la logique ou sur un autre formalisme.
Flux, Minerva, Dali sont des exemples de langages de programmation déclaratifs basés respectivement sur les logiques suivantes: Le Flux Caculus, MDLP et Kaboul, Least Herbrand Models.

Comme exemple de langages fondés sur d'autres formalismes, on peut citer CLAIM qui se base sur le calcul des ambiants.

Dans ce qui suit on présente les deux langages : AGENT-0 et CLAIM.

- **AGENT-0**

En 1993, Shoham (Shoham, 1993) définit la programmation orientée-agent (POA) comme une spécialisation de la programmation orientée objet. Pour Shoham, un système d' POA complet devrait se composer de:

- un langage formel avec une syntaxe et une sémantique claires permettant de décrire les états mentaux.
- un langage de programmation des agents
- un « agentifieur »permettant de convertir une entité neutre en un agent programmable.

L'état mental d'un agent de ce paradigme est composer de: croyances (beliefs), obligations (obligations or *commitment*), décision, et capacités (Shoham, 1993).

- Les croyances (Beliefs): Elles représentent les propositions que l'agent croit vrai à un moment donné. Ces propositions sont représentées par un opérateur modal B. La forme générale d'une croyance est comme suit : B_a^t ρ signifiant: ' « À l'instant t l'agent a croit ρ ».

- Obligations : Elles représentent les actions que l'agent s'est engagé à faire. La représentation formelle de ces dernieres est :
  $OBL^t_{a;b}$ ρ signifiant: « À l'instant t l'agent a est engagé auprès de b à ρ ».
- Décisions: Elles sont considérées comme des obligations faites à soi-même. Plus formellement :
  $DEC^t_a \rho = OBL^t_{a;a} \rho$.
- Capacités (Capabilities): Elles représentent les actions que l'agent peut exécuter. Formellement, elles sont représentées par :
  CAN^t_a ρ, , signfiant: « À l'instant t l'agent a est capable de ρ ».

Shoham a proposé un langage de programmation capable de manipuler ces concepts : AGENT-0 (Martelli et al., 1995). La syntaxe de ce langage contient :

- Les faits (Facts) : Ils spécifient le contenu des actions et leurs conditions temporelles associées.
- Les actions privées (Private actions) : Ils représentent les actions à exécuter.
  Exemple : (*((DO t p-action)*) signifie de faire l'action privée p-action au temps t.
- Les actions communicatives (Communicative actions) : Agent-0 offre quatre types de routines de communication :
  - (*((INFORM t a fact)*,
  - *(REQUEST t a action)*,
  - *(UNREQUEST t a action)*),
  - *((REFRAIN action)*).
- Les conditions mentales (Mental conditions): ceux sont des combinaison des patterns mentaux de la forme : *(B fact)* ou *((CMT a) action)*).
- Les actions conditionnelles (Conditional action) : Elles permettent de spécifier des actions à effectuer sous certaines conditions portant sur l'état mental de l'agent.
  Example: (*((IF mntlcond action)*).
- Les conditions de message (Message conditions): ceux sont des combinaison des patterns de message de la forme: *(From Type Content)*
  From represente le nom de *…, Type* est *INFORM, REQUEST* or *UNREQUEST* et *Content* représente soit un fait ou une action.
- Les règles d'engagement (The *commitment rules*) : Les règles d'engagement ont la forme suivante:

  (COMMIT msgcond mntlcond (agent action))*

  msgcond et *mntlcond* representent, respectivement, les conditions de messages et conditions mentales. a*gent* est le nom d'agent, *action* represente l'action à effectuer et '***' signifie une répétition (de zéro à ..).

Un programme Agent-0 est une séquence de règles d'engagement précédé par la définition des capacités de l'agent, ses croyances initiales et la définition d'un time grain

(Martelli et al., 1995). Le comportement d'un agent est contrôlé par un cycle des étapes suivantes (Shoham, 1993):

- lecture du message reçu et mise à jour de son état mental (à partir de ses croyances initiales et de ses règles d'engagement).
- L'exécution des actions qui doivent l'être à chaque pas de temps et peut être la mise à jour de son état mental.

- **CLAIM**

CLAIM (Computational Language for Autonomous Intelligent and Mobile Agents) est un langage de programmation orienté agent de haut niveau. Il combine des aspects cognitifs spécifiques aux agents intelligents, des primitives de communication et de mobilité inspirée du calcul des ambiants (El Fallah-Seghrouchni & Suna, 2003).

Il fait partie d'un framework (cadre) unifié appelé Himalaya (Hierchical Intelligent Mobile Agents for building Large-scale and Adaptative sYstèmes based on Ambients), dans lequel les systèmes multi-agents mobiles sont déployés sur un ensemble d'ordinateurs connectés. Par conséquent, Il est supporté par un système multi-plateforme d'agents (SyMPA) (Bordini et al., 2005).

Un agent CLAIM est une entité autonome, intelligente et mobile considérée comme une place délimitée où elle contient (Bordini et al., 2005):

- Des composants mentaux (par exemple, des connaissances, des objectifs, des capacités).
- Des processus : Un processus peut être une composition parallèle ou une séquence de processus, une instruction, une instanciation de variable, une fonction définie dans un autre langage de programmation, programmation, la création d'un nouvel agent, une opération de mobilité ou la transmission d'un message.
- Des sous-agents (facultatif).

Les agents CLAIM sont organisés en hiérarchies ou chaque nœud d'une hiérarchie représente un agent qui peut se déplacer au sein de celle-ci ou à une autre (Bordini et al., 2005). Un agent se déplace comme un ensemble, avec tous ses composants (par exemple : les processus en cours, les sous-agents …) (El Fallah-Seghrouchni & Suna, 2003).

Le langage CLAIM offre deux types de raisonnement pour les agents (El Fallah-Seghrouchni & Suna, 2003):

- raisonnement en avant: Il est utilisé quand des messages arrivent. Donc, l'agent commence par choisir un message parmi ces derniers, puis cherche les capacités qui peuvent être activé par ce message, vérifier leurs conditions et enfin activer les capacités dont les conditions sont satisfaites.
- raisonnement en arrière: Il est utilisé pour atteindre un but sélectionné. L'agent commence par choisir un but dans la liste des buts, puis trouver les capacités qui permettent d'atteindre ce but, ensuite vérifier leurs conditions et enfin activer les capacités dont les conditions sont satisfaites.

La sémantique formelle de CLAIM est basée sur l'approche opérationnelle et structurelle de Plotkin qui définit un système de transition comme suit (Plotkin, 1981) :

> *« Un système de transition (ts) est (juste!) une structure <Γ,→> où Γ est un ensemble (des éléments γ appelés configurations) et γ → γ'est inclut dans Γ× Γ est une relation binaire (appelée une relation de transition). γ → γ' signifie qu'il y a une transition de la configuration γ à la configuration γ'. »*

Le principe de cette approche est que l'exécution d'une opération automatique change l'état du programme d'un état initial à un autre.

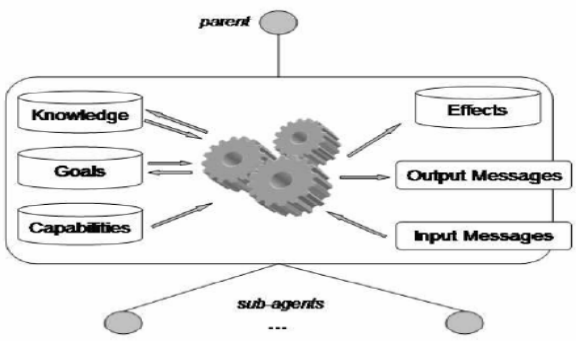


Fig.11. Agent ClAIM (Suna, 2005).

Les langages Impératifs

Les Langages purement impératifs sont moins fréquents, principalement du fait que les concepts relatifs aux agents sont fortement déclaratifs par nature (Bordini et al., 2005). Toutefois, de nombreux programmeurs utilisent les langages impératifs existant pour mettre en oeuvre des systèmes multi-agents, et par conséquent, les agents sont implémentés de manière ad-hoc.

Comme exemple d'un langage orienté- agent purement impératif, on introduit JAL (JACK Agent language).

- **JAL**

JAL est un langage de programmation orienté agent développé pour la plate-forme JACK (Busetta et al., 2000). Il étend le langage Java dans sa syntaxe et il est utilisé pour l'implémentation des agents BDI.

Il étend la syntaxe Java avec des ajouts qui peuvent être classés comme suit (Busetta et al., 2000):

- Un petit nombre de mots clés utilisé pour identifier les composantes principales d'un agent (comme : agent, plan, événement et capacité).
- Un ensemble des primitives utilisé pour la déclaration des attributs et d'autres caractéristiques des composants d'agent (par exemple, l'information contenue dans les croyances).

- Un ensemble de primitives utilisé pour définir les relations statiques (par exemple, quel plan qui peut être adopté pour réagir à un certain événement).
- Un ensemble de primitives utilisé pour manipuler l'état d'un agent (par exemple, les ajouts de nouveaux objectifs ou sous-objectifs à atteindre, des changements de croyances, l'interaction avec d'autres agents).

Les composants d'un agent sont les suivants (Winikoff, 2005):

Agent: Ils sont spécifiés en définissant les évènements qu'ils peuvent gérer et envoyer, leurs données dont notamment leur base de croyances ainsi que les plans et les capacités qu'ils peuvent utiliser.

Une base de croyances (Beliefset): est une petite base de données relationnelle stockée en mémoire. Les bases de croyances peuvent également envoyer des évènements par exemple lorsqu'elles sont modifiées.

Les Vues (View): sont des bases de croyances virtuelles calculées à partir d'autres bases de croyances.

L'évènement (Event): est changement qui intervient dans le temps et qui exige une réponse. Les évènements sont utilisés pour modéliser les messages reçus, l'adoption de nouveaux buts ainsi que des informations provenant de l'environnement.

Plan: un plan est une « recette » qui gère un type d'évènement donné. Les plans incluent :

- Une indication sur l'évènement: représente l'événement qu'ils manipulent.
- Une condition sur le contexte: contexte qui décrit les situations dans lesquels le plan peut être utilisé.
- Le corps du plan : qui inclut du code Java ainsi que du code JACK et qui est exécuté par le système.

Les capacités (Capability): elles sont composées de plans et de croyances et spécifient quels évènements elles supportent et envoient.

Remarque: Le programmeur peut utiliser les primitives de Java pour définir des agents (Busetta et al., 1999). Les composants en JAL sont transformés en des classes de java à l'aide d'un compilateur.

JAL n'a pas de sémantique formelle, et il offre aussi des structures permettant de spécifier des équipes d'agents (Busetta et al., 1999). Une équipe est une entité contenant des plans, des capacités, des données et qui peut être composée de sous-équipes.
Les agents qui effectuent les calculs courts ou qui partagent la plupart de leurs codes ou de leurs données, peuvent être regroupés.

Les langages hybrides

Les langages hybrides combinent les caractéristiques déclaratives et impératives. Il s'agit généralement de langages déclaratifs qui fournissent en même temps des constructions

impératives ou qui offrent des possibilités pour l'intégration du code implémenté dans des langages de programmation impératifs.

Comme exemple, on introduit dans ce qui suit les langages 3apl et ConGolog.

- **3apl**

3apl (An Abstract Agent Programming Language « triple-a-p-l ») est un langage de programmation pour l'implémentation des agents cognitifs. C'est une combinaison de programmation impérative et logique (Hindriks et al., 1999).

De la programmation impérative, le langage hérite les procédures récursives et la notion de calcul à base d'état (Hindriks et al., 1999). De la programmation logique, le langage hérite la preuve pour la manipulation de la base de croyances de l'agent (Hindriks et al., 1999).

NB: Les états des agents sont des bases de croyances qui sont différentes des affectations de variable de la programmation impérative (Hindriks et al., 1999).

La syntaxe de ce langage inclut (Hindriks et al., 1999) :

- Les croyances (beliefs) d'un agent 3APL décrivent la situation dans laquelle se trouve l'agent. Chaque agent se voit donc doté d'une base de croyances contenant les informations qu'il possède sur le monde.
- Buts et actions: Ils représentent ce que l'agent veut réaliser ou atteindre. Ils sont regroupés dans une base de buts. Un but peut être simple ou complexe.
  - But simple: un but simple peut être :
    - Actions de base: Spécifient les capacités d'un agent pour atteindre ses objectifs.
    - Le but à atteindre (Achievement goal): sont des propositions logique atomique
    - Le but de test (Test goal): il permet de déterminer si une formule bien formée peut-être dérivé de la base de croyances de l'agent.
  - Le but complexe : est composé de buts simples en utilisant les constructeurs de composition séquentielle et du choix non-déterministe.
- Les règles de raisonnement pratique (Practical reasoning rules): elles sont utilisées pour opérer sur les buts et permettre à l'agent une manipulation facile de ces derniers. Formellement elles sont représentées comme suit :

$$\pi_h \leftarrow \varphi \mid \pi_b$$

Cette relation signifie que si un agent a adopté un but π_h (la tête de la règle) et une croyance φ (the *guard* of the rule), alors il peut adopter π_b (le corps de la règle) comme un nouveau but. Il y a des règles avec des têtes vide ($\leftarrow \varphi \mid \pi_b$) utilisés pour créer de nouveaux buts indépendamment des buts actuels, et des règles avec des corps vide ($\pi_h \leftarrow \varphi$) pour omettre les buts.

Un agent intelligent en 3apl est représenté par un triplet <Π,σ,Γ> où

- Π: la base des buts.
- σ: la base des croyances.
- Γ : la base des règles de raisonnement pratique.

Le cycle (Fig.12.) de délibération permet à un agent de déterminer les plans à exécuter. Il représente une boucle itérant les phases suivantes (Hindriks et al., 1999):

1. Trouver les règles de raisonnement dont la tête correspond à un but.
2. Trouver les règles de raisonnement dont la garde est satisfaite par les croyances.
3. Choisir les règles de raisonnement à appliquer aux buts.
4. Appliquer ces règles.
5. Trouver des buts à exécuter.
6. Sélectionner un but.
7. Exécuter le but.

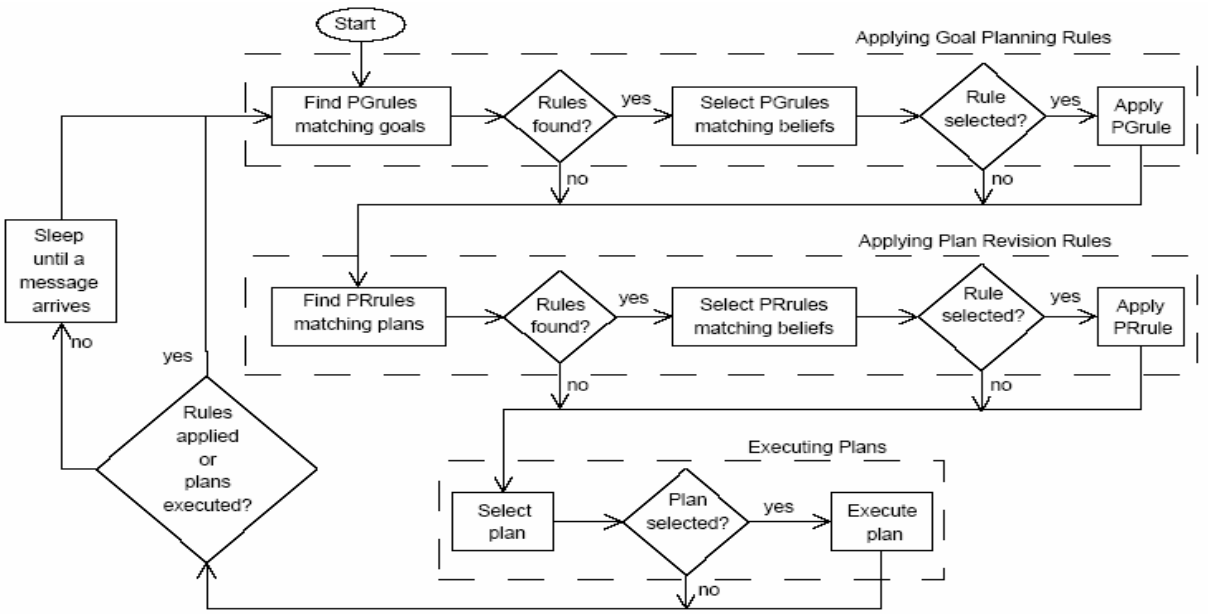


Fig.12. Le cycle de délibération pour un agent 3apl (Triple-apl, 2008).

3apl est basée sur une sémantique opérationnelle définie en utilisant les systèmes de transition (Plotkin, 1981). Un système de transition est utilisé pour transformer une configuration à une autre en utilisant un ensemble d'axiomes et de règles de dérivation (une définition de systèmes de transition de Plotkin est présentée dans la section des langages déclaratifs/ CLAIM). Une configuration 3apl consiste en trois composants :

- La base des buts de l'agent.
- La base des croyances de l'agent.
- Une substitution utilisée pour enregistrer des valeurs associées avec des variables du premier ordre.

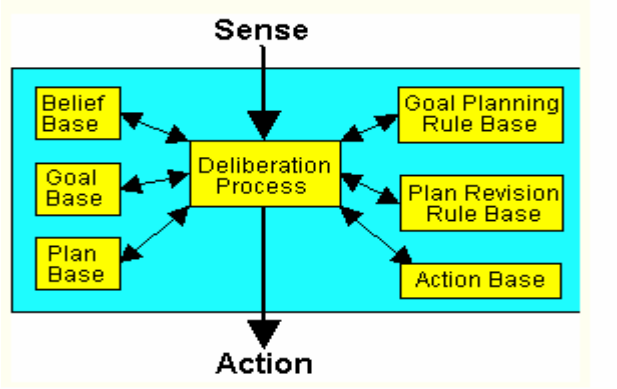


Fig.13. Agent 3apl (Triple-apl, 2008).

3apl offre une forte séparation entre les données (attitudes mentales des agents) et le processus de délibération qui manipule ces données.

De nombreuses extensions ont été proposées pour 3apl, parmi eux, l'ajout des primitives de communication pour permettre de décrire les systèmes multi-agents.

- **ConGolog**

ConGolog est un langage de programmation concurrent orienté agent basé sur le calcul des situations. Il offre des facilités pour (De Giacomo et al., 2000) :

- donner la priorité à l'exécution en cours.
- interrompre une exécution quand certaines conditions deviennent vraies.
- Le traitement des actions exogènes).

ConGolog est une extension de Golog (un langage de programmation orienté agent basé sur la logique), qui introduit la notion d'action exogène et la notion de concurrence. Cette concurrence traite:

- Des processus concurrents avec peut-être des priorités différentes,
- Des interruptions de haut niveau,
- les actions exogènes arbitraires.

Une action exogène est une action primitive qui n'a pas été prévue dans le programme mais qui doit être exécutée à cause de l'environnement de l'agent.

Golog inclut les primitives suivantes (De Giacomo et al., 2000):

a	action primitive.
Φ?	attend la condition Φ.
$(\delta_1;\delta_2)$	séquence d'actions.
$(\delta_1\|\delta_2)$	choix non déterministe entre actions
$\pi^{v.\delta}$	choix non déterministe des arguments.
δ^*	itération non déterministe
$\{\text{proc } P_1(\vec{v}_1)\delta_1 \text{ end};\ldots\text{proc } P_n(\vec{v}_n)\delta_n \text{ end };\delta\}$	procédures.

En plus des primitives précédentes, ConGolog inclut aussi (De Giacomo et al., 2000) :

If Φ then δ_1 else δ_2	Structure conditionnelle synchronisée.
While Φ? do δ	boucle synchronosée.
$(\delta_1 \| \delta_2)$	exécution concurrente.
$(\delta_1 >> \delta_2)$	priorité au niveau de la concurrence, δ_2 ne peut être exécuté que si $\delta_1$est bloqué ou terminé.
$\delta^{\|}$	Itération concurrente.
$< \Phi \rightarrow \delta >$	interruption, le programme δ sera déclenché si la condition Φ est satisfaite.

La sémantique de Golog et ConGolog est présentée sous forme de transition (De Giacomo et al.,). Celle de ConGolog utilise deux prédicats : Final(δ,s) et Trans(δ,s,δ′,s′).

Trans et Final spécifient, respectivement, les transitions possibles entre les configurations, et quand une configuration peut être terminée.

- Final(δ,s): représente l'état dans lequel un programme δ peut se terminer dans une certaine situation s.
- Trans(δ,s,δ′,s′) : représente la transformation du programme δ dans la situation s en δ' dans une situation s' après l'exécution d'une étape.

2.6. Frameworks d'agents

2.6.1. JADE

JADE (Java Agent Development framework) est un environnement logiciel pour le développement des systèmes multi-agents. Il est conforme à la FIPA. JADE offre les caractéristiques suivantes (Bellifemine et al., 2007) (cette référence est utilisée pour toute la section ci dessous):

- La plate-forme multi-agents compatible FIPA inclut le Système de Gestion d'Agents (AMS), le Facilitateur d'Annuaire (DF), et le Canal de Communication entre Agents (ACC).
- Une plate-forme d'agents distribuée implémentée en java. Cette plate-forme peut être distribuée sur plusieurs hôtes, à condition qu'il n'y ait pas de pare-feu entre ces hôtes.
- Une interface graphique utilisateur pour gérer plusieurs agents et agent conteneur en partant d'un agent unique. L'activité de chaque plate-forme peut être supervisée et enregistrée.
- Un certain nombre de DF (Facilitateurs d'Annuaire) compatibles FIPA qui peuvent être activés quand on lance la plate-forme pour exécuter les applications multi-domaines, où un domaine est un ensemble logique d'agents.
- Une interface de programmation pour simplifier l'enregistrement de services d'agents avec un ou plusieurs domaines de type DF.
- Le mécanisme de transport et l'interface pour l'envoi et la réception des messages.

- Le protocole IIOP compatible avec le document FIPA97 pour connecter des plates-formes multi-agents différentes.
- Une API en java pour la transmission de messages ACL entre les agents. Dans le but de simplifier la transmission, les messages internes (sur la même plate-forme) sont transférés codés comme des objets Java. Quand l'expéditeur ou le récepteur n'appartient pas à la même plate-forme, le message est automatiquement converti à/du format de chaîne de caractères spécifiés par la FIPA. De cette façon, la conversion est cachée au programmeur d'agents, qui a seulement besoin de traiter la classe d'objets Java.
- Une bibliothèque de protocoles d'interaction compatibles FIPA.
- L'enregistrement et l'annulation automatique d'agents dans le Système de Gestion d'Agents (AMS).
- Un service d'attribution de noms compatible FIPA. Chaque agent a un identificateur unique (Globally Unique Identifier - GUID).
- Des outils de débogage.
- JADE supporte la mobilité des agents.
- Le modèle de comportement (behaviour) pour permettre l'exécution des activités multiples, parallèles et concurrentes des agents.
- Une bibliothèque pour gérer les ontologies et les langages définis par l'utilisateur.
- Les applications externes ont la capacité de lancer des agents autonomes en utilisant l'interface InProcess.

La classe 'Agent'

La classe 'Agent' représente une classe de base commune pour tous les agents définis par l'utilisateur. Un agent JADE est une instance d'une classe java défini par l'utilisateur (cette classe étend la classe de base). Par conséquent, les agents héritent des caractéristiques d'interaction de base (tels que: l'enregistrement, la configuration, …) et d'un ensemble de méthodes qui peuvent être appelées pour implémenter les tâches spécifiques à l'agent (par exemple, envoyer / recevoir des messages, l'utilisation de protocoles standard d'interaction…).

Chaque service fourni par un agent devrait être implémenté comme un ou plusieurs comportements qui peuvent être exécutés simultanément. Ces comportements sont gérés par un scheduler hérité par l'extension de la classe Agent.

Interaction entre agents

L'interaction des agents JADE est faite par les messages ACL. Lorsque un agent reçoit un message trois situations peuvent avoir lieu:

1. Les agents sont en interaction dans le même conteneur de la même plate-forme, donc, les événements java sont utilisés et le message est tout simplement copié.
2. L'un des agents en interaction sur un autre conteneur, mais toujours de la même plate-forme: Java RMI est utilisé, le message est numéroté à l'expéditeur, une méthode à distance est appelée et le numéro du message est supprimé au récepteur.
3. L'éxpéditeur et le destinataire résident sur des plates-formes différentes. Dans cette situation, l'IIOP est utilisé. Le message est envoyé par le biais d'un appel à distance CORBA.

2.6.2. NetLogo

NetLogo est un langage de programmation multi-agents et un environnement de modélisation pour la simulation de phénomènes complexes (Tisue & Wilensky, 2004).

Son principe est le suivant : "à seuil bas, pas de plafond." Par ‘Seuil bas’ en sous entend la facilité d'utilisation du langage même si l'utilisateur n’a jamais programmé, et par ‘ pas de plafond’, le langage ne devrait pas être limité pour les utilisateurs avancés.

Les utilisateurs peuvent utiliser "la bibliothèque des modèles" (Tisue & Wilensky, 2004), qui contient différents types développés à l’aide de Netlogo. Ces modèles sont généralement bien documentés. Il suffit d’utiliser l'onglet "information" dans programme pour prendre connaissance de leurs utilisations. De plus, ils peuvent soit modifier les paramètres du modèle en utilisant “sliders”, ou le code du modèle en cliquant sur l’onglet "procédures".

Il existe quatre types d’agents NetLogo (NetLogo, 2008) : turtles, patches, links, et observer.
Turtles sont des agents qui se déplacent et agissent dans leur environnement. Ce dernier est constitué d’une grille de patches qui représentent des "zones", des portions de l’environnement. links sont des agents qui relient deux Turtles. Observer ne possède pas une position, mais on peut l'imaginer situé en dehors du monde des Turtles et des patches.

Les comportements des Turtles et des patches peuvent être contrôlés par des commandes primitives (pré-défini) spécifiques à chaque type d'agents. Il est possible de contrôler les deux à la fois en utilisant les "procédures" (une procédure est une séquence de commandes NetLogo).

La commande ‘Ask’

Les utilisateurs ont la possibilité de donner des commandes uniquement aux Turtles, aux patches et aux links. Par exemple:

ask turtles [...]

Habituellement, l'observateur utilise la commande ‘ask’ pour demander à toutes les turtles, à tous les patches ou à tous les links d’exécuter des commandes. On peut également l’utiliser avec une tortue, un patch ou un link.

La primitive ‘agentset’

Un agentset est un ensemble d’agents, qui contient un seul type d’agents à la fois (soit des turtles, soit patches ou des links).

La primitive ‘Breeds’

NetLogo permet la définition de "races" pour les turtles ou les links. Lorsque les races sont définies, l'utilisateur peut les faire évoluer selon la situation désirée. Par exemple, il peut créer des races moutons et loups, et les loups essayent de manger les moutons.

Pour chaque race définie, un agentset est automatiquement créé. Ainsi, les turtles peuvent changer leurs races. Un loup peut ne pas rester loup toute sa vie.

L'ordre des déclarations des races a un impact important sur l'ordre dans lequel ils sont représentés. Ainsi, les races définies ultérieurement apparaîtront au-dessus des races définies plus tôt.

2.7. Conclusion

Les agents sont un domaine en pleine expansion avec des applications fructueuses dans plusieurs domaines. Pour bénéficier de ce paradigme intéressant, on doit mettre en oeuvre des agents individuels, leurs environnements et leurs interactions. Par conséquent, de nombreux langages de programmation ont été proposés.

La plupart des recherches sur les langages d'agent sont fondées sur des approches déclaratives et en particulier celles basées sur la logique. Les langages purement impératifs sont moins fréquents, en raison de la nature déclarative et le niveau élevé des abstractions de la conception des systèmes à base d'agent (Bordini et al., 2005). D'autres langages préfèrent la combinaison des deux approches pour bénéficier de leurs avantages. Parfois un langage de programmation agent est intégré dans un framework d'agents qui offre des composants utiles pour leurs implémentations, ainsi que des mécanismes qui facilitent leurs communications.

Ces langages se différencient par le degré d'importance associé à certains aspects par rapport à d'autres. Ainsi une comparaison entre les différents langages s'impose afin d'avoir une idée générale sur les différent aspects qui peuvent être utilisés pour présenter les agents impliqués dans le domaine de la conception des scripts de collaboration. . Le chapitre suivant fera l'objet d'une présentation des langages de programmation d'agents les plus connus et les plus référencés ainsi qu'une discussion de quelques aspects.

Chapitre 3

Les Langages de Programmation Agent : Étude Comparative

3.1 Introduction

La programmation orientée agents a été proposée par Yoav Shoham en 1993 comme un nouveau paradigme de programmation. Dans ce paradigme les agents sont les éléments importants et centraux du langage.

En réalité, on peut utiliser n'importe quel langage de programmation pour implémenter les agents, mais il est souhaitable d'utiliser un langage de programmation qui offre des constructions spéciales pour la construction des agents, de leurs environnements et les moyens d'interaction entre eux.

Il y a une grande diversité de langages orientés agents pour implémenter les agents, chaque langage favorisant une vision particulière de la notion d'agent intelligent et offrant des constructions spéciales pour leurs développements.

Ce chapitre traite les langages de programmation présentés dans le chapitre précédent en les comparant en premier lieu par rapport à des aspects généraux puis par rapport à d'autres aspects qu'on propose comme critères de comparaison. Une discussion des résultats de cette comparaison sera présentée.

3.2. Étude comparative des langages de programmation d'agents

Cette section présente une comparaison entre les différents langages présentés dans le chapitre précédent par rapport aux caractéristiques (Martelli et al., 1995) présentés dans la section intitulée: «Les caractéristiques des langages de programmation agent».
Ces caractéristiques sont: le temps, la perception, la communication, la concurrence, le non-déterminisme, la modularité et la sémantique.

Temps

Dans Agent-0 le temps est inclus dans toutes les constructions du langage (Martelli et al., 1995). Les opérations autorisées sur les variables de temps sont seulement les opérations mathématiques (des sommes et des différences).

Dans ConGolog, le temps est directement liée aux situations (Martelli et al., 1995), par exemple: s_0 est la situation de l'agent au temps t = 0 et do([a1,…,an], s_0) est sa situation au temps n. Il n'existe pas d'opérateurs spécifiques pour le gérer.
Pour 3apl, le temps est directement lié aux états de l'agent. Par exemple: la base de buts initiale représente la base de buts au moment t = 0. Le même principe est adopté pour CLAIM et JAL.

Perception

Les agents devraient percevoir tous les changements qui surviennent dans leur environnement. Pour 3apl, quand un agent exécute une action qui change l'état de l'environnement, ses résultats mettent immédiatement à jour sa base de croyances sans vraiment percevoir ces changements. Par conséquent, cet agent n'observe pas le plus souvent son environnement. ConGolog et Agent-0 n'ont pas de constructions explicites pour la perception de l'environnement (Martelli et al., 1995).

Communication

AGENT-0 supporte la communication, il offre un ensemble de structures représentant les actions communicatives (par exemple : INFORM, REQUEST, UNREQUEST, …etc.). Ces structures peuvent être utilisés pour construire n'importe quel type de communication (Martelli et al., 1995). Cependant, il y a une forte limitation liée à la syntaxe de l'action demandée (Martelli et al., 1995): par exemple, si un agent A demande une action à l'agent B, ce dernier n'a aucun moyen pour comprendre son contenu afin de l'effectuer, par conséquent, les agents doivent spécifier les actions demandées en utilisant la même syntaxe.

3apl permet la communication entre les agents à l'aide de primitives de communication ayant une sémantique opérationnelle formelle.
Les agents CLAIM sont aptes à communiquer point-à-point ou par diffusion (El Fallah-Seghrouchni & Suna, 2003).

ConGolog n'offre pas des primitives de communication, le développeur doit définir ses propres primitives (Martelli et al. , 1995).

Pour JAL, différentes infrastructures de communications peuvent être fournies par l'utilisation de méthodes appropriées (Busetta et al., 1999).

Concurrence

ConGolog supporte les deux types de concurrence (la concurrence des actions dans le même agent et/ou la concurrence entre les agents) (Martelli et al., 1995): il propose

différentes constructions pour l'exécution simultanée des processus internes du même agent ou la représentation de l'exécution simultanée des différents agents.

3apl permet la concurrence entre les agents, ainsi que l'exécution d'un ensemble de buts à l'intérieur du même agent. CLAIM supporte les deux types de concurrence (El Fallah-Seghrouchni & Suna, 2003).
Pour JAL, le noyau de JACK prend en charge les deux types de concurrence et permet aussi la combinaison des deux. Toutefois, AGENT-0 ne prend pas en charge la concurrence (Martelli et al., 1995).

Non déterminisme

Un programme est déterministe si, pour n'importe quel état, il y a exactement un seul état prochain possible. Les programmes en CLAIM sont implicitement non-déterministes en raison de la concurrence (El Fallah-Seghrouchni & Suna, 2003). Dans plusieurs situations, un programme peut évoluer vers plusieurs états à partir d'un état donné.

ConGolog permet un choix non déterministe entre les actions, un choix non déterministe des arguments et une itération non déterministe (Martelli et al., 1995).

3apl et JAL supportent le non-déterminisme (par exemple, un agent 3apl choisit les règles correspondantes aux buts, puis il sélectionne une à exécuter et ainsi de suite), tandis qu'AGENT-0 ne supporte aucun comportement non déterministe (Martelli et al., 1995).

Modularité

ConGolog permet la définition de procédures. Elles sont définies par des macros dans les formules de calcul de situations (Martelli et al., 1995). AGENT-0 ne prend pas en charge la définition des procédures.

Un agent 3apl est composé d'un certain nombre d'éléments (tels que la base de croyance, la base des buts,… etc.), mais cette décomposition ne représente pas la modularité parce que leurs fonctionnements sont étroitement liés et ils ne peuvent pas être considérés comme des modules distincts (Van Riemsdijk et al., 2006). Un agent CLAIM ne peut supporter la modularité que si les sous agent sont considérés comme des modules distincts avec des fonctionnalités indépendantes.
Java supporte la modularité et comme JAL est une extension de celui-ci, on conclue qu'il la supporte aussi.

Sémantique

Tous les langages discutés ont une sémantique formelle, sauf AGENT-0 et JAL.

Le tableau ci-dessous représente une comparaison entre les différents langages.

	Langages Déclaratifs		Impératif	Langages Hybrides	
Langages	AGENT-0	Claim	JAL	3apl	ConGolog
Temps	✓	≈	≈	≈	≈
Perception	x	x	x	x	x
Communication	✓	✓	✓	✓	≈
Concurrence	x	✓	✓	✓	✓
Non déterminisme	x	✓	✓	✓	✓
Modularité	x	✓	✓	x	✓
Sémantique	x	✓	x	✓	✓

✓ : l'aspect est assuré. x : l'aspect est non assuré. ≈ : l'aspect est assuré partiellement.

Tab.1. Comparaison des langages présentés selon des critères généraux.

Ce tableau montre les points forts et faibles des langages de programmation orienté-agent selon quelques caractéristiques. Ces dernières sont choisies selon le travail à développer ou pour donner une vue générale de ces langages.

La comparaison présentée ci-dessus donne une idée générale des langages présentés. Les caractéristiques utilisées sont très importantes et utiles afin d'avoir une vue globale du langage, mais il y a aussi d'autres aspects qui devraient être examinés.

3.3. Les six aspects

Parmi les aspects qui devraient être examinés, on suggère: l'autonomie et le comportement d'agent, l'organisation, l'interaction et la coordination, l'environnement (perception, action), l'intégration des entités non agent et la gestion des ressources.

Cette partie concerne la définition des aspects et l'analyse des différents langages et frameworks selon ces aspects.

3.3.1. Autonomie et comportement

Selon Tim Smithers (1992) (Gouaïch, 2005), l'autonomie est décrite comme:

> *« L'idée centrale de la notion d'autonomie est identifiée dans l'étymologie du terme:* ***autos*** *(soi) et* ***nomos*** *(règle ou loi). Il a été utilisé pour la première fois en Grèce afin de spécifier l'état des citoyens qui formulent leurs propres lois, par opposition à la vie selon des lois imposées par un pouvoir externe. Il est utile de différencier le concept d'autonomie de celui d'un système automatique. Le sens d'automatique provient de l'étymologie du terme* ***'cybernétique'****, qui dérive du terme grec équivalent à 'auto-contrôle'. En d'autres termes, les systèmes automatiques contrôlent leurs activités afin de satisfaire des lois qui leurs sont imposées ou construits en eux, tout en rectifiant et compensant les effets des perturbations externes. Alors que, les systèmes autonomes développent pour eux-mêmes, les lois et les stratégies selon lesquelles ils contrôlent leurs comportements. »*

D'après la définition ci-dessus, il apparaît que l'autonomie d'un système est son aptitude à développer ses propres lois et de les suivre, alors qu'un système automatique doit suivre un ensemble de lois imposées extérieurement ou construites intérieurement.

Cette description de l'autonomie est générale et il existe de nombreuses interprétations de celle-ci. Gouaïch (Gouaïch, 2003) présente deux principales interprétations à savoir l'auto-gouvernance et l'indépendance, alors que Parunak (Verhagen, 2000) introduit l'autonomie dynamique et déterministe.

- **Les interprétations de Gouaïch**

L'autonomie en tant qu'auto-gouvernance

Un agent autonome (Castelfranchi, 1995) est un logiciel capable de produire ses propres lois et se comporter selon celles-ci afin d'atteindre ses objectifs.

L'autonomie en tant qu'indépendance

L'autonomie d'un agent est liée à son contexte social (Sichman et al., 1994). Les partisans de cette idée introduisent le réseau de dépendance sociale (SDN) (Sichman et al., 1994) afin de permettre aux agents d'avoir des descriptions extérieures représentant les autres agents, par conséquent, les agents ont la possibilité de raisonner sur leur société artificielle.

On distingue trois formes d'autonomie.

- **A-autonomie:** un agent est un a-autonome s'il ne dépend que de ses actions pour atteindre un certain objectif en fonction d'un plan choisi.

- **R-autonomie:** un agent est r-autonome s'il ne dépend que de ses ressources pour atteindre un certain objectif en fonction d'un plan choisi.

- **S-autonomie:** un agent est s-autonome s'il est à la fois un a-autonome et r-autonome.

- **Les interprétations de Parunak**

L'autonomie dynamique

Un agent présente une autonomie dynamique s'il a la capacité d'entamer l'exécution en se basant sur sa structure interne.

L'autonomie déterministe

Un agent présente une autonomie déterministe s'il n'y a pas d'accès à ses ressources, évitant ainsi l'influence sur sa prévisibilité.

Selon Parunak, ceci est déterminé par un observateur extérieur. Cette autonomie ne représente pas l'autonomie de l'agent, mais plutôt une indication qu'un observateur extérieur a un modèle incomplet sur le modèle interne de l'agent.

Remarque

Gouaïch (Gouaïch, 2005) est en accord avec l'idée de Parunak concernant l'observateur extérieur et pense que ses deux interprétations sont plus absolues et il est plus intéressant de donner une définition relative à l'autonomie, c'est-à-dire selon un observateur extérieur.

3.3.2. Communication et interaction

La communication (Gouaïch, 2005) est le processus d'échange de données entre les différentes entités logicielles en utilisant un moyen de communication.

L'interaction (Gouaïch, 2005) est un processus spécial de communication où les données échangées ont la possibilité de modifier l'état des entités en communication. Ces données influent sur leurs futurs comportements. L'échange se fait par le biais d'un moyen d'interaction.

La coordination selon Malone et Crowston (Malone & Crowston, 1994) est décrite comme la gestion des dépendances entre les activités afin d'éviter les conflits. Ces dépendances peuvent être classées en quatre catégories:

- Ressource partagée.
- La relation producteur / consommateur.
- La contrainte de simultanéité.
- Tâche / sous-tâche.

Ressource partagée: cette dépendance est due au nombre limité de ressources partagées. Lorsque différentes activités ont besoin des mêmes ressources en même temps, il sera nécessaire de gérer ces allocations.

La relation producteur / consommateur: elle est introduite quand une activité (le producteur) produit quelque chose qui va être utilisée par une autre (le consommateur). Cette dépendance introduit les sous-dépendances suivantes:

- **Contrainte sur les prérequis**: le travail de l'activité consommatrice est lié à la disponibilité des ressources à consommer, c'est à dire quand les travaux de l'activité productrice sont achevés.
- **Transfert:** Pour consommer les ressources produites, il est nécessaire de les transférer du point de production au point de consommation.
- **Utilisation:** Les ressources produites doivent être utilisées correctement par le consommateur. Pour cela, le producteur et le consommateur doivent être en accord sur des standards et de déterminer les conditions qui rendent la ressource consommable.

La contrainte de simultanéité: Cette dépendance est liée essentiellement au temps d'exécution de certaines activités. Certaines activités doivent être exécutées en même temps, alors que d'autres ne peuvent pas se produire en même temps.

Tâche / sous tâche: une activité ou une tâche peut être divisée en plusieurs sous-activités. Ainsi, l'activité principale dépend de ses sous-activités et se termine seulement après la fin de celles-ci.

3.3.3. Organisation

Le sens de l'organisation (Boissier et al., 2007b) varie souvent selon les deux points de vue suivants:

(i) Une entité collective représenté par (mais pas identique à) un groupe d'agents en interaction. Cette entité a une identité.
(ii) Un pattern stable (ou une structure) des activités partagées qui peut influencer les interactions entre les agents afin d'atteindre leurs objectifs.

Les deux points de vue ne sont pas en général mutuellement exclusifs. Pour avoir une idée plus générale de l'organisation, les auteurs (Boissier et al., 2007a) la présentent selon deux dimensions: le processus de définition de l'organisation des agents et sa "représentation" au sein des agents.

- **Point de vue basé-agent vs. Point de vue basé-Organisation**

Ces points de vue proposés dans (Lemaître & Excelente, 1998) sont :

- Point de vue basé-agent.
- Point de vue basé-organization.

Le premier point se concentre plus sur les agents et les considère comme le "moteur" de l'organisation.
L'organisation est un phénomène observable émergent qui présente une vision globale du modèle de coopération entre les agents (voir première ligne dans la Fig.14.-a-b).

Dans le cas (a), l'organisation est le résultat du comportement collectif des agents dans un environnement commun et dynamique. Ce type d'organisation est le plus approprié pour les agents réactifs. Alors que, dans le cas (c) le comportement collectif est le résultat d'un modèle de coopération qui structure les activités collaboratives des agents.

Le deuxième point présente l'organisation comme une entité explicite du système (voir la deuxième rangée dans la Fig.14.-c-d). Il insiste sur l'utilisation de primitives qui sont différentes des agents ". Le modèle de coopération est réglé par les concepteurs (ou par des agents eux-mêmes dans les systèmes auto-organisés).

- **Agents connaissent vs. Agents ne connaissent pas l'organisation**

Cette dimension est basée sur les capacités des agents à représenter et à raisonner sur leur organisation.

Dans la première colonne de la Fig.14., Les agents ne connaissent rien au sujet de l'organisation.
Dans le cas (a), ceux-ci ne sont pas conscients qu'ils font partie d'une organisation, mais l'observateur est au courant de celle-ci. Ainsi dans le cas (c), bien qu'elle soit définie et formalisée, les agents ne la connaissent pas et ils n'en tiennent pas compte. Ils lui obéissent comme si les contraintes organisationnelles sont implantées dans leurs codes.

Dans la deuxième colonne, les agents *ont une représentation explicite de l'organisation.* Dans le cas (b), chaque agent a une représentation interne et locale du pattern de coopération qu'il utilise pour prendre sa décision. Cette représentation locale est obtenue soit par la perception, la communication ou le raisonnement explicite (en raison de la première dimension : point de vue basé- agent).

Dans le cas (d), l'organisation est définie (en raison de: point de vue basé-organisation) et chaque agent a une représentation explicite de celle-ci. Ils sont capables de l'utiliser pour leurs raisonnements et prendre l'initiative de coopérer avec d'autres agents dans le système.

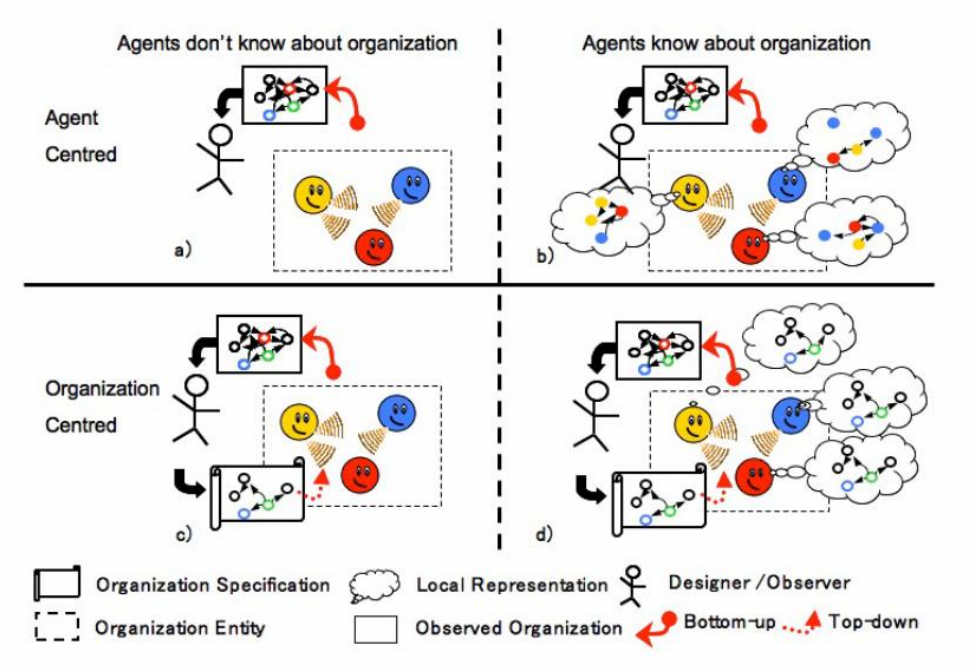


Fig.14. Les deux dimensions des organisations dans les SMA (Boissier et al., 2007a).

Remarque

Dans la littérature, certaines approches de l'organisation d'agents sont fondées sur un cas spécifique indiqué à la Fig.14. ; tandis que d'autres sont basées sur des cas multiples. Par exemple, les approches de réorganisation pour les organisations formelles combinent les cas (b) et (d) permettant aux agents d'utiliser leurs mécanismes internes pour s'adapter à l'organisation imposée. La manipulation de l'organisation peut être soit endogène (réalisée par les agents appartenant à l'organisation) ou exogène (réalisée par un concepteur ou par d'autres agents en dehors de l'organisation).

3.3.4. Environnement

Le terme environnement (Weyns et al., 2005) est utilisé pour se référer à:

- L'entité logique des SMA, dans laquelle les agents et autres objets / ressources sont intégrés.
- L'infrastructure logicielle sur laquelle le SMA s'exécute.

- L'infrastructure matérielle sur laquelle le SMA fonctionne.

Les fonctionnalités de l'environnement sont souvent traitées implicitement ou d'une manière ad-hoc parce que les chercheurs en SMA ne traitent pas l'environnement comme une entité de première classe. Cette dernière (Weyns et al., 2005) est définie comme suit:

> *« Un bloc de programme, une partie indépendante de logiciel qui fournit [...] une abstraction ou un mécanisme caché d'information afin qu'une mise en œuvre d'un module peut être changé sans exiger le changement d'autres modules. »*

Ainsi en général, l'environnement dans les SMA n'est pas considéré comme une entité indépendante avec des responsabilités claires.

- **Point de vue de Russell et Norvig**

Russell et Norvig présentent la relation entre l'agent intelligent et son environnement (Weyns et al., 2005):

> *« Un agent est ce qui peut être compris comme percevant son environnement à travers des senseurs et comme agissant sur cet environnement par l'intermédiaire des effecteurs. »*

Ils proposent de classer les environnements en :

– Accessible vs. Inaccessible : un environnement accessible est un environnement dans lequel l'agent peut obtenir une information complète sur l'état de cet environnement.

– Déterministe vs. Non déterministe: un environnement déterministe est un environnement dans lequel les actions ont effet. Il n'y a pas une certitude sur l'état qui résultera de l'action en cours.

– Statique vs. Dynamique: un environnement dynamique est un environnement qui peut être changé (c'est-à-dire, aucun autre processus n'agit sur lui) lorsque un agent est en train de s'exécuter.

– Discret vs. Continu: un environnement est discret s'il a un nombre fini de perceptions et d'actions possible.

Les classes les plus complexes d'environnement sont celles qui sont inaccessibles, non déterministes, dynamiques et continues.

- **Point de vue de J. Ferber**

Un environnement, selon J. Ferber (Weyns et al., 2005), peut être représenté comme un environnement centralisé ou comme un environnement distribué (un ensemble de cellules assemblées dans un réseau).

Dans le premier cas, tous les agents ont accès à la même structure. Tandis que dans le deuxième cas, chaque cellule agit comme un environnement centralisé en miniature, tout en présentant les différences suivantes:

(1) L'état d'une cellule dépend des cellules voisines.
(2) Les agents ont la capacité de percevoir les autres cellules.
(3) La gestion de la liaison de l'agent avec les cellules en cas de mouvement d'une cellule à l'autre.
(4) La gestion de la propagation des signaux sur le réseau des cellules.

Il suggère que :
- les environnements peuvent être représentés par des modèles "généralisés" ou "spécialisés". Dans un modèle généralisé, les agents peuvent effectuer tout type d'actions (les actions possibles), alors que dans un modèle spécialisé, ils ne peuvent effectuer que des actions bien définies.
- les SMA peuvent être purement communicatifs (les agents ne peuvent communiquer que par le transfert de messages), purement situé (ils ne peuvent agir que dans l'environnement) ou une combinaison des deux.

Le concept central du modèle de l'environnement de Ferber est le modèle des actions. Il différencie entre les influences et les réactions aux influences. Les influences sont des actions effectuées par les agents afin de modifier l'environnement. Les réactions sont produites par l'environnement en combinant les influences de tous les agents.

- **Point de vue de J. Odell et ses collègues**

Selon J. Odell et ses collègues (Weyns et al., 2005),

> *« Un environnement offre les conditions dans lesquelles une entité (agent ou objet) existe. »*

Les auteurs considèrent l'environnement physique et l'environnement de communication. L'environnement physique se concentre sur l'existence physique des agents et des objets, par conséquent, un ensemble de lois, de règles, de contraintes et de stratégies sont fournis (exemple de loi fournie: deux agents ne sont pas autorisés à occuper le même lieu en même temps) .

L'environnement de communication se concentre sur l'échange d'informations, les fonctions et les structures utilisées (tel que les rôles, les groupes et les protocoles d'interaction entre eux).

Ils définissent aussi l'environnement social d'un agent comme :

> *« Un environnement de communication dans lequel les agents interagissent de manière coordonnée. »*

Cet environnement se compose de groupes, de rôles et de membres. Les agents sont les membres qui jouent des rôles dans les groupes sociaux.

Un groupe est une unité sociale qui peut être vide (il n'y a aucun agent participant) ou un ensemble d'agents (contenant un seul agent participant ou plusieurs agents). Les groupes ont une identité unique dans le système. Un rôle est une représentation abstraite de la fonction d'un agent ou une identification au sein d'un groupe.

- **Point de vue de Jennings, Sycara et Wooldridge**

Wooldridge, Sycara et Jennings proposent la définition suivante d'un agent (Jennings et al., 1998) :

> *« Un système informatique, situé dans un environnement, et qui agit d'une façon autonome et flexible pour atteindre les objectifs pour lesquels il a été conçu. »*

La plupart des approches se concentrent sur les agents cognitifs, et traitent l'environnement d'une manière abstraite en accordant peu d'attention à celui-ci. En conséquence, celles-ci se concentrent sur l'analyse et le processus de conception de l'agent au sein du système. Les préoccupations liées à l'environnement sont traitées au niveau des frameworks, middleware, et d'autres constructions d'implémentation (Klein & Holger, 2006).

La recherche dans le domaine des agents réactifs insiste sur la situation des agents et accorde de l'importance à l'environnement et à la manière dont ceux-ci le perçoivent et l'affectent (Klein & Holger, 2006).

- **Environnements des agents mobiles**

Dans les paragraphes ci-dessus, les chercheurs insistent sur le fait que les agents sont statiques alors que d'autres peuvent être mobiles. Ces derniers se déplacent d'une façon autonome à travers un réseau. Par conséquent, leurs environnements doivent prendre en compte la contrainte de mobilité.

3.3.5. Intégration des entités non agent

Les entités non agent sont des objets qui peuvent être utilisés par les agents. Elles peuvent appartenir à l'environnement.

3.3.6. La gestion de ressources

On propose que les ressources soient toutes les structures de données et les espaces mémoire qui peuvent être utilisés par les agents. Dans le cas où ces ressources appartiennent à l'environnement, celui-ci est responsable du contrôle et de la gestion de leur accès, sinon ce rôle incombe à l'agent lui-même.

Dans le cas des ressources partagées, il est nécessaire de coordonner les activités d'accès à celles-ci. Ceci peut être fait par l'environnement ou par les agents en utilisant l'interaction afin de résoudre les différents conflits.

3.4. Discussion

3.4.1. Les tables d'analyse

Le premier tableau représente l'analyse des langages de programmation ; tandis que le deuxième représente l'analyse des framework

	AGENT-0	**CLAIM**	**JAL**	**3apl**	**ConGolg**
Autonomie et Comportement	- Le comportement d'un agent est contrôlé par un cycle	- L'agent a deux types de comportement: réactive et proactive. - L'autonomie de créer et supprimer des agents pendant l'exécution du système	- L'agent a deux types de comportement: réactive et proactive	- Le comportement d'un agent est implémenté par un cycle de délibération	- Le comportement des agents est spécifié par des procédures
Organisation	Implicite: - Elle concerne l'ensemble des gents.	Implicite: - Elle concerne l'ensemble des agents.	Implicite: - Elle concerne l'ensemble des agents. - Il enrichit le noyau de JACK par des concepts comme: 'Team' et 'Role'.	Implicite: - Elle concerne l'ensemble des agents.	Implicite: - Elle concerne l'ensemble des agents
Interaction Et Coordination	Inclut des primitives de communication	Permet la communication asynchrone	Les agents JACK ne sont liés à aucun langage de communication spécifique.	La communication est conforme aux standards FIPA.	Il n'y a pas de primitives de communication.

Environnement (perception, action)	Implicite: - Aucune capacité de percevoir l'environnement. - Les croyances sont utilisées pour représenter l'état du monde et des autres agents.	Implicite: - Aucune capacité de percevoir l'environnement. - L'environnement est l'ensemble des environnements des agents.	Implicite: - Aucune capacité de percevoir l'environnement.	Implicite: - Aucune capacité de percevoir l'environnement. - Les croyances représentent l'environnement à partir du point de vue de l'agent	Implicite: - Aucune capacité de percevoir l'environnement.
Intégration des entités non agent.	Aucun support des entités non agent.	Il est possible d'utiliser les objets java.	Il est possible d'utiliser les objets java.	Aucun support des entités non agent.	Aucun support des entités non agent.
Gestion des resources	- Structures de données (exp : les croyances.)	- Structures de données (exp : les croyances). - Les sous agents (facultatif). - La file d'attente de messages d'agent (privée).	- Les croyances sont représentées dans une base de données relationnelle, qui est enregistrée en mémoire.	- Structures de données (exp : les croyances, les buts, les plans et les règles de raisonnement.) - Des constructeurs de programmation pour manipuler ces structures.	- Des structures de données.

Tab.2. Comparaison des langages présentés selon les six aspects.

	NetLogo	**JADE**
Autonomie Et Comportement	- Comportement réactif.	- Les agents sont des objets actifs avec au moins un thread java. - Les comportements des agents sont représentés en utilisant la notion du 'Behaviour'.
Organisation	- L'organisation est le résultat du comportement collectif des agents. - Il est possible de définir des groupes d'agents (des turtles et des links) en utilisant : 'agentsets' et 'breeds'.	- Implicite: concerne l'ensemble des agents.
Interaction et Coordination	- Les agents Links sont utilisés pour l'interaction de deux turtles.	- JADE permet la communication peer-to-peer.
Environnement (perception, action)	- Aucune capacité de percevoir l'environnement	- Aucune capacité de percevoir l'environnement - Réduit l'environnement à un système de transport des messages.
Intégration des entités non agent	- Aucun support des entités non agent.	- L'utilisation des objets java pour le transfert des messages.
Gestion des resources	- L'utilisation de variables globales, des variables des 'turtles' ou de variables de 'patch'.	- Les structures de données. - JADE utilise des files d'attente pour stocker les messages reçus des agents (chaque agent a une file privée) et une bibliothèque pour gérer les ontologies définies par l'utilisateur.

Tab.3. Comparaison des frameworks présentés selon les six aspects.

3.4.2. Analyse des travaux

Autonomie et comportement

Pour AGENT-0, le comportement des agents (Suna, 2005) est contrôlée par un cycle itérant les étapes suivantes:
- Lire les messages courants et mettre à jour l'état mental.
- Exécuter les engagements (obligations) pour le temps actuel et peut-être mettre à jour l'état mental.

Pour un agent 3apl, le comportement est basé sur un cycle penser-réagir. La première phase du cycle (Penser) correspond à la phase du raisonnement pratique à l'aide de règles qui permettent aux agents de modifier leurs objectifs de manière arbitraire. Tandis que, la deuxième phase (réagir) correspond à une phase d'exécution dans laquelle l'agent effectue une action.

Le langage CLAIM offre deux types de raisonnement pour les agents qui sont le raisonnement en avant et le raisonnement en arrière. Il permet deux types de comportement: réactive et proactive (Suna, 2005). Les agents CLAIM ont la possibilité de créer et supprimer des agents au cours de l'exécution du système de façon autonome. Un agent peut créer un nouvel agent en utilisant le processus 'newAgent', l'agent ainsi créé devient un sous-agent de l'agent créateur. Il peut aussi supprimer complètement l'un de ses sous-agents, en utilisant les primitives 'Kill'. L'agent supprimé disparaît complètement du SMA avec tous ses sous-d'agents.

Le comportement d'un agent JAL est exprimé en terme de plans. Ce langage fournit des événements normaux afin de permettre la mise en œuvre du comportement réactif, et des événements BDI pour la mise en oeuvre du comportement proactif (Tagni & Jovanovic, 2006).

Dans ConGolog, le comportement des agents est spécifié par des procédures.

Pour JADE (Bellifemine et al., 2007), les agents sont des objets actifs avec au moins un thread Java. Les comportements des agents sont représentés en utilisant la notion de 'Behaviour' qui met en œuvre les tâches et les intentions d'un agent.

Netlogo est un langage pour la simulation de phénomènes complexes, par conséquent, il se concentre sur les agents réactifs c'est-à-dire sur le comportement réactif.

Organisation

Il n'existe pas de représentation explicite de l'organisation et tous les langages discutés ci-dessus l'introduisent implicitement comme l'ensemble des agents.

L'organisation est généralement considérée soit comme une entité collective (représentée par un groupe d'agents en interaction) ou comme un pattern stable (ou structure) des activités partagées des agents. Dans notre cas, on la présente comme la capacité d'organiser les agents dans des groupes. On utilise pour l'analyse de celle-ci les différents concepts utilisés pour former ces groupes.

Ces concepts sont généralement utilisés pour la définition des environnements sociaux. Dans notre cas, pour l'aspect d'environnement, l'accent sera mis sur la perception et l'action des agents sur ce dernier. Ainsi, ces concepts sont utilisés dans l'aspect organisation plutôt que dans celui de l'environnement.

NetLogo fournit les concepts de 'Agentsets' et de 'breeds' (breeds de turtles et breeds de links). L'intérêt du groupement est la possibilité de le traiter comme un seul ensemble (une unité) au lieu de traiter chaque agent.
JAL comprend les notions d'équipe et de rôle.

Interaction et coordination

Le mot 'interaction' est souvent utilisé pour représenter un processus de communication. La communication est un des aspects utilisés dans la première comparaison qui montre qu'elle est supportée par tous les langages présentés sauf ConGolog. En ConGolog, le développeur (de spécification) doit définir ses propres primitives de communications. JADE (Bellifemine et al., 2007) support un modèle de communication 'peer-to-peer', obtenu par un transfert asynchrone de messages. En Netlogo, l'interaction de deux 'turtles' se fait par le biais des agents 'links'.

Environnement (perception, action)

Il n'existe pas de représentation explicite de l'environnement par ces langages. Ils considèrent les agents comme des entités actives situées dans cet environnement, capables de le percevoir et d'effectuer des actions.

Les agents intelligent 3APL (Hindriks et al., 1999) sont des entités qui représente leur environnement par le biais de leurs croyances et le contrôlent par des actions qui font partie de leurs objectifs ou plans. Les agents dans AGENT-0 utilisent aussi leurs croyances pour représenter l'état du monde et les autres agents. Un agent JAL a différentes bases de croyances (beliefsets) qui stockent des croyances sur les différents aspects du monde.

Pour CLAIM (Tagni & Jovanovic, 2006), l'environnement est représenté par l'ensemble des environnements des agents. Chaque agent a son propre environnement contenant des éléments qui ne commandent pas des processus à l'intérieur de l'agent, bien qu'ils puissent évoluer au cours de son exécution et influent sur son comportement. Ces éléments sont l'autorité de l'agent actuel, le nom de la classe utilisée pour l'instancier, la base de connaissances, la liste des capacités, les intentions et le but à atteindre.

Les frameworks populaire tels que Jade, Jack réduisent l'environnement à un système de transport de messages (Weyns et al., 2005).

La première comparaison montre qu'aucun des agents, spécifiés par ces langages, n'est en mesure de percevoir son environnement. L'action sur ce dernier se résume à mettre à jour les croyances de ces agents en fonction de leurs actions.

Intégration des entités non agent

Dans JADE (Bellifemine et al., 2007), les messages sont transférés codés comme des objets Java au lieu de chaînes de caractères. Lorsque les messages transférés arrivent aux frontières de la plate-forme, ils sont automatiquement convertis en une syntaxe conforme à la norme FIPA. Il est possible d'utiliser des objets Java avec CLAIM et JAL.

La gestion de ressources

Tous les langages et les frameworks présentés utilisent différentes structures de données pour stocker leurs propres informations (les croyances (AGENT-0), des croyances, des objectifs, des plans et des règles de raisonnement (3apl),…). Ils offrent des constructions de programmation afin de les manipuler.

JAL représente l'ensemble des croyances (beliefset) comme une base de données relationnelle qui est stockée en mémoire. Pour économiser les ressources du système, les agents qui effectuent des calculs courts, partagent la plupart de leur code ou de leurs données, peuvent être regroupés (en utilisant les équipes) (Howden et al., 2003).

Un agent CLAIM peut avoir comme ressource des sous-agents (facultatif) et une file d'attente de messages. Ces sous-agents offrent à leurs parents l'intelligence et des composants de calcul (Suna, 2005). Les messages reçus par un agent sont mis dans sa file d'attente et traités dans l'ordre d'arrivée. Ainsi, les agents JADE utilisent des files d'attente pour stocker les messages reçus.

On peut utiliser les ontologies comme des ressources pour les agents. Ainsi JADE (La dernière version) permet aux programmeurs de créer leurs propres langages de contenu, leurs ontologies ainsi qu'une bibliothèque pour les gérer.
Le programmeur en CLAIM (Suna, 2005) peut définir ses propres ontologies des informations sur le monde. Ces informations sont représentées comme des propositions du premier ordre contenant un nom et une liste d'arguments.

NetLogo utilise des variables qui sont des espaces pour stocker les valeurs (comme les nombres). Une variable peut être globale (de 'turtle' ou du 'patch'). Chaque agent peut accéder à une variable globale qui a une seule valeur.

3.5. Conclusion

Les chercheurs ont proposé de nombreux langages pour la programmation des agents. Ces langages orientés agent sont plus appropriés que les langages généraux quand le problème est spécifié comme un système multi-agents (Bordini et al., 2005). L'utilisation des constructions spéciales pour construire nos agents facilite la programmation et permet le développement rapide de nos applications.

Ces langages doivent offrir au moins les caractéristiques principales d'un système multi-agent (tel que l'autonomie des agents). Toutefois, ils se différencient par le degré d'importance associé à certains concepts par rapport à d'autres.

Devant la difficulté de conception d'un langage offrant des constructions adéquates et efficaces pour l'implémentation d'un agent quelconque, il est important de spécifier certains aspects dès le début. Ces derniers sont choisis le plus souvent selon le domaine d'application. Dans notre cas, on est intéressé par la conception des scripts de collaboration en utilisant les agents artificiels. Afin de développer ces agents on a besoin d'un langage de programmation agent adéquat et qui offre certains aspects permettant leurs réalisations.

Dans notre travail, on s'est intéressé aux aspects mentionnés précédemment à savoir: l'autonomie et le comportement, l'organisation, l'interaction et la coordination, l'environnement (perception, action), l'intégration des entités non agent et la gestion des ressources.

Chapitre 4

Conception des Scripts par Retour Expérimental

4.1. Introduction

La conception d'un script n'est pas une tâche facile car elle a pour cible un groupe d'humain. Ainsi cette conception requiert une évaluation auprès des apprenants.

Le suivi des apprenants joue un rôle important pour l'apprentissage humain. L'enjeu dans ce domaine est de comprendre le comportement de l'apprenant ou d'un groupe d'apprenants, qui utilise un environnement d'apprentissage. Ceci permet à l'enseignant ou au formateur d'avoir une information précise et adéquate pour ses besoins propres sur l'évolution individuelle et collective des apprenants.

L'exploitation des traces réalisées par les enseignants, formateurs, ou apprenants, permet non seulement de produire des éléments intéressants pour la modélisation comportementale ou conceptuelle. Il est nécessaire dans ce contexte d'aider les utilisateurs de la trace à collecter, transformer et analyser les traces issues des observations d'apprentissage humain.

4.2. Méthodologie et problématique

Diverses méthodes de conception des scénarios d'apprentissage ont été proposées mais aucune ne prend en compte le retour expérimental et l'incrémentalité dans le processus de conception des scripts (scénarios) de collaboration.
La plupart des travaux sur la conception des scripts portent sur la description du résultat plutôt que sur la démarche que suit le concepteur et ne font pas jouer un rôle important au retour expérimental. Ainsi les facteurs humains (apprenants) ne sont pas assez pris en compte.

Le scénariste pédagogique ne peut juger d'un choix de conception que s'il en évalue les conséquences en situation réelle à partir d'un retour des vrais utilisateurs. Aussi, nous préconisons une démarche itérative dans laquelle les résultats de l'évaluation du scénario produit sont analysés et interprétés en vue d'une adaptation, d'une remise en cause ou d'une amélioration de ce dernier.

L'exécution du script doit se décliner dans un modèle de tâches et non à partir d'un modèle de contenu des ressources mises à la disposition des apprenants. L'idée est de permettre au concepteur d'exprimer ses choix et de ne pas se contenter uniquement de la description du script.

Le cadre de notre conception repose sur six idées principales :

1. Le processus de conception est de nature incrémentale, basé sur une boucle (Scripting/Spécification/Exécution/Evaluation). Le scripting consiste à écrire les règles de collaboration pour un groupe d'apprenants en langage naturel, de le spécifier ensuite avec un formalisme, de l'exécuter et enfin de l'évaluer par l'analyse des traces des apprenants (Fig.15.).

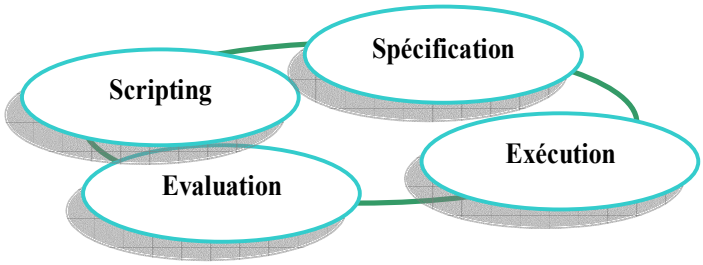


Fig.15. La boucle du processus de conception.

2. Le Scripting doit être considéré dans son ensemble et pas uniquement au travers de son résultat. Il s'agit de prendre en compte :
   - Le point de départ : les données sources.
   - Le point d'arrivée : le produit cible.
   - La transformation faisant passer de l'un à l'autre.

3. Une méthode de conception permettant de guider sans contrainte le concepteur qui doit prendre en compte les facteurs humains intervenant dans l'exécution du script. Le modèle d'exécution du script doit se déduire d'un modèle de tâches ou d'activités et non d'un modèle de données.

4. Les facteurs humains jouent un rôle central dans le processus de conception. Les concepteurs ont besoin des informations sur les apprenants et de leurs collaborations afin de favoriser les interactions souhaitables. Pour cette raison, nous proposons le suivi des apprenants.

5. Un formalisme de description qui permettra au scénariste d'exprimer ses choix et pas uniquement d'en décrire le résultat.

6. Un environnement intégré est souhaitable pour la conception des scripts de collaboration afin de faciliter une communication « sans discontinuité » entre les divers « espaces d'activités » dans lesquels le concepteur évolue. En effet, dans une approche incrémentale de conception de scripts intégrant un retour sur l'exécution amène le concepteur à évoluer sans cesse d'un espace à l'autre.

4.3. La conception incrémentale des scripts de collaboration

Le scénariste pédagogique ne peut juger d'un choix de conception que s'il en évalue les conséquences en situation réelle à partir d'un retour expérimental des vrais utilisateurs.
Il s'agit de suivre le comportement des apprenants en respectant le scénario prescrit dans un premier temps par le concepteur et fournir des retours sur l'exécution de ce dernier pour revoir ou affiner la conception préliminaire.

Nous proposons une conception incrémentale afin d'insister à la fois sur l'importance des interactions des apprenants qui devront être prises en charge dans le cadre d'un retour expérimental et sur l'affinage progressif du script tout au long de sa conception. Ce processus de conception, comme le processus de conception en génie logiciel est de nature itérative et incrémentale. Ainsi, des retours peuvent également être nécessaires pendant la phase de conception.

Afin de faciliter la tâche aux concepteurs, nous proposons un ensemble d'espaces d'activités dans lesquels ils évoluent.

Quels espaces d'activités et comment évoluer d'un espace à un autre ?

Les notions d'étapes doivent être distinguées de celle d'espaces d'activité dans une démarche de conception. En effet, une étape caractérise un état spécifique du produit et les méthodes de conception sont décrites en termes d'étapes. Alors que l'espace d'activité caractérise un état de l'activité du développeur. Nous avons identifié et caractérisé les espaces d'activités dans lesquels évolue le processus de conception et qui sont présentés dans la figure suivante (Fig.16.) ainsi que les liaisons entre les différents espaces.
Nous pouvons identifier sept espaces d'activités utiles pour le concepteur répartis sur deux phases à savoir la phase de conception et d'évaluation. Chaque espace d'activité correspond à un point de vue identifiable du concepteur sur sa tâche de conception.
Pour la phase de conception, nous avons identifié quatre espaces d'activités qui sont :

- **L'espace d'acquisition** des données et des activités dans lesquelles les informations sont collectées : ressources pédagogiques, profils, activités d'apprentissage,…

- **L'espace de modélisation et de description** qui permet au concepteur de disposer d'abstractions clefs et une vision claire et précise des informations qu'il va utiliser.

- **L'espace d'élaboration de la structure des activités** : l'activité principale du concepteur est l'élaboration d'un modèle d'activités ou des ressources pédagogiques qui découle de celui de la tâche pour laquelle est conçu le script et qui est différent du modèle des données.

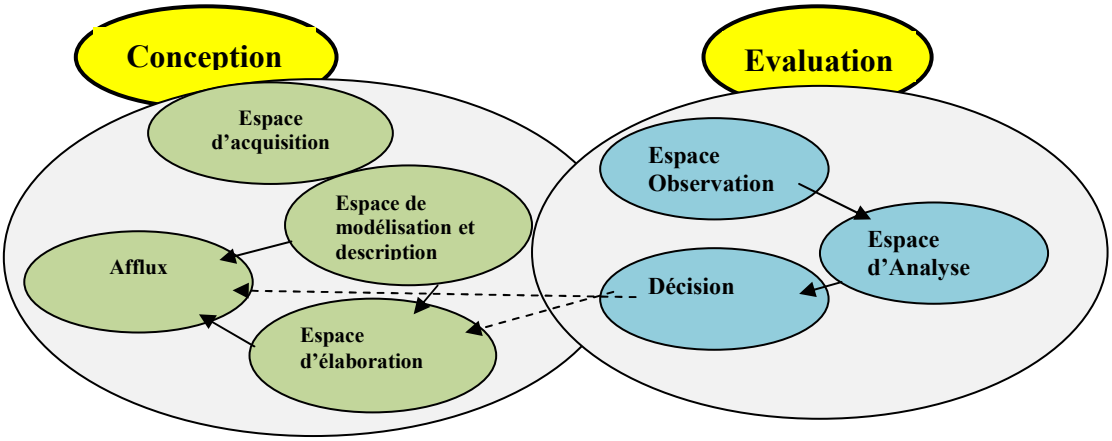
- **L'espace d'afflux** est de donner le moyen au concepteur de spécifier comment passer de la spécification de la structure du script à la structure instanciée du script cible.

Pour la phase d'évaluation, nous avons retenu les trois espaces suivants qui sont inspirés du processus de collecte de la trace:

- **L'espace d'observation :** Le principal souci est d'observer la progression des apprenants en accord avec leurs besoins et profils.

❖ **L'espace d'analyse :** actions et interactions des apprenants dans leurs groupes sont analysés à partir de l'observation livré dans l'espace précédent afin d'avoir des informations synthétisées sur la progression des apprenants dans le groupe et sur le script conçu.

❖ **L'espace décision**: Le principal souci est d'avoir une décision sur le scénario qui sera présenté au concepteur du script afin de l'affiner plus tard.



.

Fig.16. les différents espaces d'activités.

Les contraintes d'une telle approche sont la génération automatique du script qui ne peut être possible que par une formalisation du résultat de la conception et une assistance par une société d'agents autonomes qui activeront dans les différents espaces d'activités proposés ci-dessus et conserver d'un cycle à l'autre ce qui concerne l'activité du concepteur.

4.4. Pourquoi les agents pour la conception des scripts et le suivi des traces

Jennings et ces collègues (Jennings et al., 1998) affirment que l'utilisation des agents est devenue attractive grâce à leurs capacités de caractériser naturellement et facilement une variété d'applications, et aussi la possibilité de représenter les entités actives d'un domaine ou d'un système.

Le paradigme d'agent est le plus puissant paradigme à offrir des abstractions pour l'analyse et la modélisation des complexités des organisations. Il permet de voir les humains et les systèmes logiciels comme des entités qui interagissent et collaborent afin de réaliser leurs buts, et exécuter leurs tâches.

Aussi la planification des actions et réactions d'un groupe d'apprenants sont difficiles sans le recours à des outils dits " intelligents" qui exécuteront le scénario, suivront la trace des acteurs de ce dernier (apprenants et/ou tuteurs) et fourniront certaines interprétations aux concepteurs.

Il est indéniable que ce paradigme apporte des solutions en matière de résolution de problèmes, d'analyse et de conception logicielle lorsque des critères comme la distance, la coopération entre des entités hétérogènes ou l'intégration de logiciels préexistants sont à prendre en considération (Pesty et al., 2003). Ces critères ont un rôle important dans le domaine des EIAH. Les apprenants et /ou les tuteurs utilisent des environnements d'apprentissage qui sont souvent situés à distance et qui peuvent intégrer d'autres systèmes (par exemple des systèmes spécifiques utilisés pour l'obtention des éléments de traces de plus haut niveau).

Dans une situation d'apprentissage normale (non assistée par ordinateur), l'enseignant cherche toujours des indices afin qu'il puisse comprendre le niveau d'un apprenant, son état de compréhension … etc. Alors que l'utilisation d'un environnement d'apprentissage rend la tâche plus compliquée d'où la nécessité d'un suivi des apprenants. Le suivi lui-même est difficile parce qu'il est persistant dans le temps et il se concentre sur l'observation des apprenants qui sont différents. Pour ces raisons, il est plus intéressant d'utiliser des agents spécialisés selon le type des traces des apprenants, capables de coopérer et de coordonner leurs actions pour fournir le meilleur appui à l'apprentissage.

4.5. Modèles de conception de scripts et de suivi de traces basés sur les systèmes multi-agents

Dans cette section on présente nos modèles utilisés pour la conception des scripts de collaboration et le suivi des apprenants en utilisant des agents artificiels.

4.5.1. Présentation générale du modèle suggéré pour la conception des scripts de collaboration

La conception des scripts de collaboration est une tâche difficile. Pour cette raison on suggère l'utilisation d'un ensemble d'agents ayant les rôles suivants: décision, interprétation, exécution, observation, et suivi de traces des apprenants.

Le travail des agents commence par l'interaction des différents apprenants avec le système.

Une interface graphique d'utilisateur (GUI) est utilisée pour faciliter les interactions des apprenants avec le système. Chaque apprenant doit présenter son profil (nom, prénom, niveau, groupe, …etc.) en utilisant celle-ci. Ces profils sont stockés dans la "Base de Profils" par "l'Agent Décision".

En se basant sur les profils de ces apprenants, un script sera sélectionné (adéquat aux profils et non pas aux comportement/collaboration des apprenants) par "l'Agent Décision". Dans une deuxième phase, celui-ci (le script) sera traduit et interprété par un ensemble d'agents ("Agent Interprétation") qui doit réécrire le script dans un format compréhensible par les autres agents. Finalement un autre ensemble d'agents ("Agent Exécution") l'exécutera en tenant compte des différents profils. L'exécution du scénario sera contrôlée par un observateur (un groupe d'agents). "Les Agents de Suivi de traces" sont responsables du suivi des traces des différentes actions des apprenants. Les traces collectées sont stockées dans une "Base de Traces".

De cette manière, le concepteur peut modifier son script et l'adapter en fonction des apprenants.

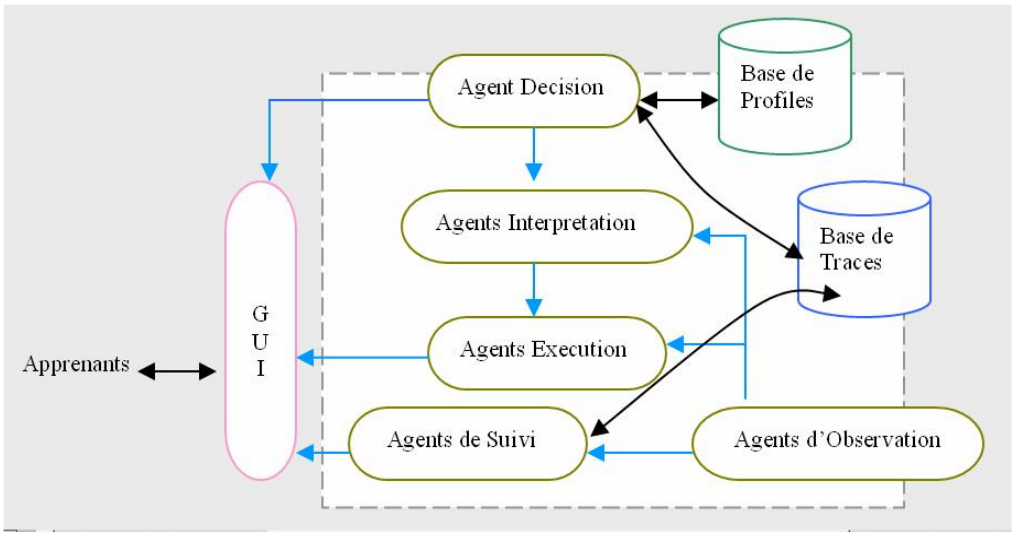


Fig.17. le schéma général du modèle suggéré pour la conception des scripts.

Agent Décision : S'appuyant sur les profils des apprenants, l'agent décision choisit un script adéquat pour être exécuté. Afin de fournir au concepteur les informations nécessaires, cet agent peut accéder directement aux bases des profils et des traces.

Agent Interprétation : Le script est spécifié en utilisant un format qui est différent de celui utilisé par les agents, par conséquent, ce script sera interprété par ces agents afin de le rendre compréhensible pour les autres agents.

Agents Exécution : Ces agents exécutent le script interprété en tenant compte de la diversité des profils des apprenants.

Agents de suivi : Ils recueillent les différentes traces des apprenants au cours de leurs interactions et collaborations. On suggère la classification des traces suivante :

- **Les traces brutes :** Elles sont directement issues de l'activité de l'apprenant avec l'environnement informatique. par exemple un fichier "logs" enregistré par le système, un enregistrement vidéo de l'apprenant pendant la session, les messages postés dans un forum,…etc.

- **Les traces de production :** Elles sont issues principalement des travaux des apprenants destinés à être évalués, et aussi des travaux des enseignants, exemple un rapport sur la qualité de l'activité.

- **Les traces additionnelles :** Elle sont issus de la situation d'apprentissage. Elles sont de nature variée, comme la méta-donnée d'une ressource, une taxonomie décrivant le domaine d'apprentissage, une ontologie, un curriculum académique,… etc.

Remarque: Un fichier log (Log, 2008) est un fichier regroupant l'ensemble des événements survenus sur un logiciel, une application, un serveur ou tout autre système informatique. Il se présente sous la forme d'un fichier texte classique, reprenant de façon

chronologique, l'ensemble des événements qui ont affecté un système informatique et l'ensemble des actions qui ont résulté de ces événements.

Agents d'Observation : L'exécution du script sera contrôlée par ces agents qui surveillent le travail des autres agents afin de fournir des informations générales sur l'exécution du script.

De manière générale, le concepteur utilisera les différentes informations introduites par les apprenants (leurs profiles, …etc.) ou collectés par les agents de suivi dans les différents espaces de la phase de conception. Dans la phase d'évaluation, on se base essentiellement sur le travail des agents de suivi des traces et d'observation afin d'évaluer l'exécution du script et offrir une décision sur ce dernier au concepteur.

4.5.2. Présentation du modèle suggéré pour le suivi des traces des apprenants

En général, le processus de collecte et d'utilisation des traces d'apprentissage comprend trois phases successives (David et al., 2005) qui sont représentés par le schéma ci-dessous :

- la collecte des traces, qui peut par exemple se faire à partir de fichiers "log" issus de l'application dans laquelle se déroule l'activité.
- la structuration des traces. Après leur collecte, les traces sont exprimées dans un format en vue de leur exploitation.
- l'analyse (régulation, évaluation, observation, etc.) des traces structurées, aussi appelées indicateurs d'apprentissage.

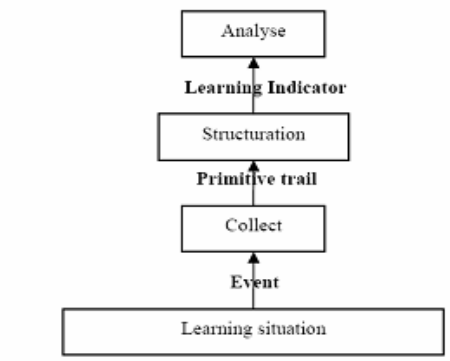


Fig.18.Le processus de collecte et d'utilisation des traces d'apprentissage (David et al., 2005)

D'après ce schéma, on remarque que les phases représentent des composants distribués, hétérogènes et autonomes. Par analogie aux systèmes développés en intelligence artificielle distribuée, l'utilisation des systèmes à base d'agent, semble bien adaptée à assurer le rôle de chaque phase. En effet, le principal intérêt de ces systèmes réside dans la distribution des agents logiciels, entités communicantes, autonomes, réactives et compétentes (Ferber, 1997), susceptibles de réaliser des tâches ou de résoudre des problèmes collectivement.

Pour cette raison on suggère l'utilisation d'un ensemble d'agents ayant les rôles suivants: collecte, structuration, analyse et visualisation.
Le travail des agents commence par l'interaction des différents apprenants avec le système.

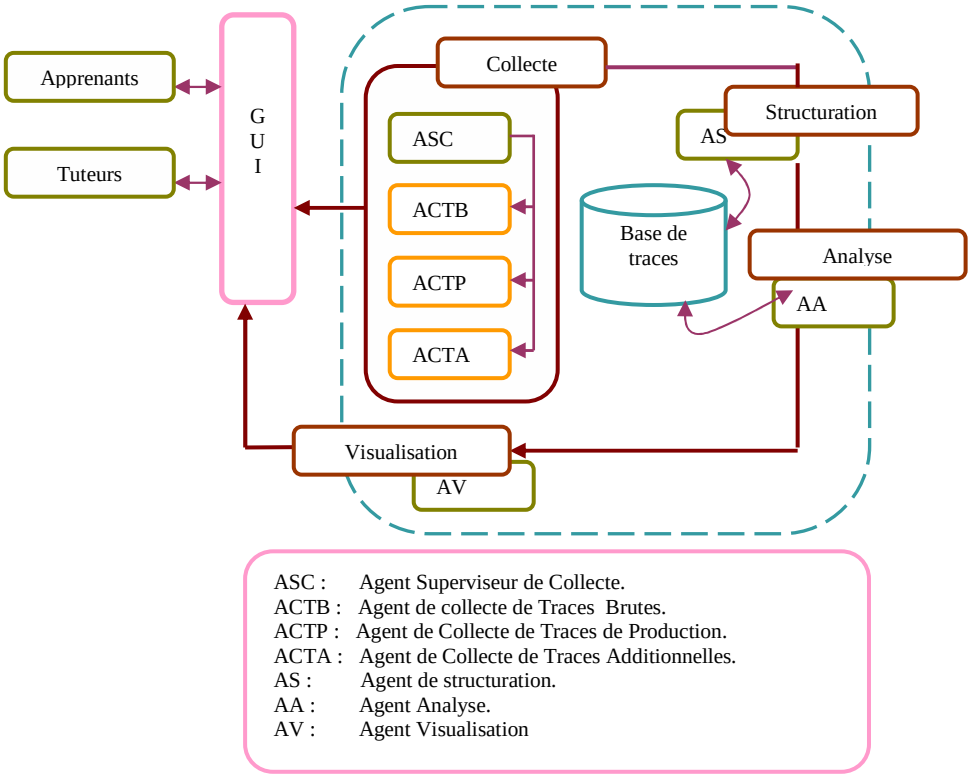


Fig.19. le schéma du modèle suggéré pour le suivi des traces des apprenants.

L'agent superviseur de collecte : Cet agent communique avec trois agents, à savoir, l'agent de collecte des traces brutes, l'agent de collecte des traces de production et l'agent de collecte des traces additionnelles. Les traces obtenues par ces agents sont des "traces primaires" car elles sont obtenues directement et non pas à l'aide d'autres traces ou données. Il permet de créer d'autres traces à partir des indices ou des traces collectées par ces agents : "les traces dérivées". Ces dernières peuvent être également obtenues à l'aide d'autres données dérivées.

Aucune interprétation n'est faite sur les données collectées pendant cette étape.

- **L'agent de collecte des traces brutes :** Il permet la capture des traces brutes, puis extrait de ces dernières des données qualifiées de brutes, représentées dans un format indépendant du dispositif d'apprentissage. Elles peuvent être collectées avant, pendant ou après la session d'apprentissage par le dispositif d'enseignement.

- **L'agent de collecte des traces de production** : Cet agent collecte les traces produites volontairement par les apprenants ou les enseignants (tuteurs).

- **L'agent de collecte des traces additionnelles :** Cet agent collecte les traces additionnelles qui sont liées à la situation pédagogique et elles sont connues à l'avance.

L'agent structuration : Il permet de regrouper les données collectées et les présenter suivant un format spécifié (par exemple format CSV, format texte etc.…) afin de les rendre compréhensibles. Ces traces spécifiées sont enregistrées dans une base de traces.

L'agent analyse : Il permet l'extraction des traces susceptibles d'intéresser le concepteur, afin de déduire des informations sur le comportement des apprenant, sur leurs connaissances et leurs difficultés, ainsi que sur leurs interactions.

L'agent visualisation : Il offre à l'apprenant une visualisation de son état d'avancement ou une représentation des activités effectuées durant la résolution d'un problème.

4.6. Application des modèles suggérés au script 'ArgueGraph'

4.6.1. Application du modèle suggéré pour la conception des scripts de collaboration

Quand les apprenants sont présents, l'agent de décision informe les agents d'interprétation en vue de réécrire le script dans un format compréhensible par les autres agents. Ce script est exécuté par un ensemble d'agents d'exécution.

Le travail des agents, selon les différentes phases du script **'ArgueGraph'**, se compose des étapes suivantes :

Phase1: Les agents d'exécution offrent aux apprenants le questionnaire. Chaque apprenant répond aux questions et argumente ses choix. Les différentes interactions avec le système sont recueillies par les agents de suivi.

phase2: Selon les réponses, l'agent de décision produit le graphe correspondant. L'enseignant ou cet agent forme des paires d'apprenants qui ont des réponses conflictuelles au cours de la phase 1.

phase3 : Les agents d'exécution fournissent le même questionnaire de la phase 1 où chaque paire doit répondre et fournir des arguments. L'agent de décision permet aux apprenants de voir leurs réponses et leurs justifications précédentes (de la phase 1). Ces apprenants sont suivis par les agents de suivi (cette tâche sera expliqué en appliquant le modèle de suivi des traces).

phase4 : L'agent de décision évalue pour chaque question la réponse apportée individuellement et en collaboration. Les résultats sont utilisés dans une session de compte rendu où les apprenants sont invités à commenter leurs arguments.

phase5 : Chaque apprenant écrit un résumé de tous les arguments concernant une question précise. Le résumé doit être structuré selon le cadre utilisé dans la session de débriefing.

De cette façon, le concepteur peut modifier son script et l'adapter sur la base des apprenants en utilisant un ensemble d'agents artificiels.

4.6.2. Application du modèle suggéré pour le suivi des traces des apprenants

Le suivi des apprenants commence par l'interaction de ces derniers par le système, en commençant par :

Phase 1: Une fois que l'apprenant aura répondu au questionnaire proposé et argumenté ces choix, l'agent de collecte des traces de production capture les différentes traces laissées par cet apprenant, puis transfert ces dernières à l'agent superviseur de collecte qui peut créer d'autres traces dérivées à partir de ces traces collectées.

L'agent superviseur de collecte envoie les traces résultantes à l'agent structuration afin qu'il les organise et les transforme en un format spécifié. Ces traces transférées sont stockées dans une base de traces. L'agent analyse extrait de la base de traces celles qui représentent les réponses des apprenants et les envoie à l'agent visualisation.

Cet agent visualisation utilise ces traces pour offrir à l'apprenant la possibilité de voir son état d'avancement par rapport au questionnaire (les questions qui ont des réponses, les questions restantes, les questions argumentés,…etc.) et la possibilité de voir aussi les activités effectuées.

Phase 2 : Le système produit un graphe dans lequel tous les étudiants sont positionnés en fonction de leurs réponses. Les étudiants observent le graphe et discutent ses résultats d'une manière informelle. Le système ou le tuteur forme des paires des étudiants différents.

Phase 3 : L'agent de collecte des traces de production suie les paires qui répondent au même questionnaire (comme dans la phase 1) et fournissent de nouveau un argument. L'agent de collecte des traces brutes essaye de capturer les différentes traces des apprenants qui consultent leurs réponses précédentes. Ces traces sont envoyées à l'agent superviseur de collecte.

L'agent superviseur de collecte envoie les traces (primaires et/ou dérivées) à l'agent structuration qui organise ces traces et les réécrit sous un format spécifié puis les enregistre dans la base de traces.

Phase 4 : Pour chaque question, l'agent analyse extrait les traces collectées des activités individuelles (phase 1) et des activités en collaboration (phase 3). Ces traces sont transférées à l'agent visualisation qui permet aux apprenants de voir leurs états d'avancement et les activités effectuées.

Le système calcule les réponses données individuellement et en collaboration. L'enseignant / tuteur utilise ces résultats au cours d'une session de débriefing face-à-face et il demande aux étudiants de commenter leurs arguments.

Phase 5 : Chaque élève écrit une synthèse de tous les arguments collectés pour une question précise. La synthèse doit être structurée selon les instructions de l'enseignant lors de la session de débriefing (phase 4).

Remarque : Par le mot 'système', on désigne l'environnement informatique d'apprentissage humain (EIAH), et notre modèle représente une partie à intégrer à cet environnement d'apprentissage.

4.7. La spécification des scripts

On a besoin d'un formalisme afin de spécifier les scripts et les présenter sous un format bien défini. On suggère l'utilisation du IMS Learning Design (IMS LD) car il est flexible (Burgos et al., 2005). Il prend en compte une grande variété de scripts. Il est utilisé pour transformer les situations d'apprentissages en unités d'apprentissage (UOL) décrites de manière formelle et pouvant être exécutées avec un éditeur IMS - LD basé sur un moteur tel que Coppercore.

4.7.1. La spécification IMS LD

La spécification IMS LD est inspirée des travaux de R. Koper sur les langages de modélisation pédagogique. Les unités d'apprentissage sont exprimées sous forme de documents structurés au format XML.

Une unité d'apprentissage (UOL) est une unité complète de travail pédagogique organisée selon une approche conceptuelle de l'apprentissage et qui assemble les ressources liées, les liens web et plusieurs matériaux et services d'apprentissage dans un dossier ZIP unique. C'est un fichier compressé avec : (a) un « manifeste » XML qui décrit la « méthode », la pièce, les actes, les rôles, les activités, l'environnement, les propriétés, les conditions et ou les notifications de la spécification, qui indique en outre les ressources qui lui sont liées ; et (b) le groupe de documents ou les ressources mentionnées dans le « manifeste » XML.

De manière générale, la spécification IMS LD utilise une métaphore selon laquelle une unité d'apprentissage est décrite comme une pièce de théâtre, organisée en actes dans lesquels des activités sont proposées à des rôles dans un environnement composé de services (chat, forum, messagerie, …) ainsi que de ressources.

Pour faciliter la tâche de spécification, on utilise un éditeur pour la conception des situations d'apprentissage. Dans notre cas on a utilisé l'éditeur : "ReCourse" (voir Fig.20. et Fig .21.), qui permet de créer des unités d'apprentissage spécifiées par l'IMS LD (ReCourse, 2008).

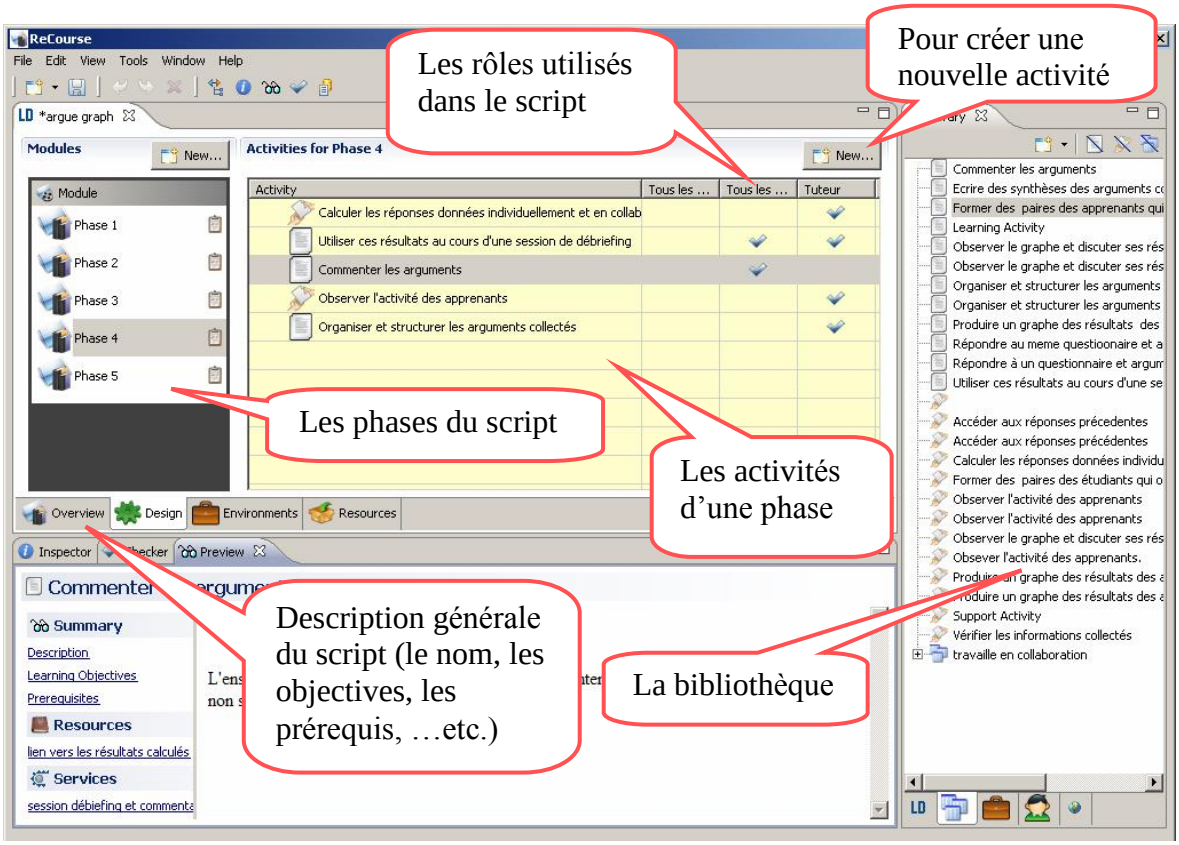


Fig.20. L'éditeur 'ReCourse' (ReCourse, 2008).

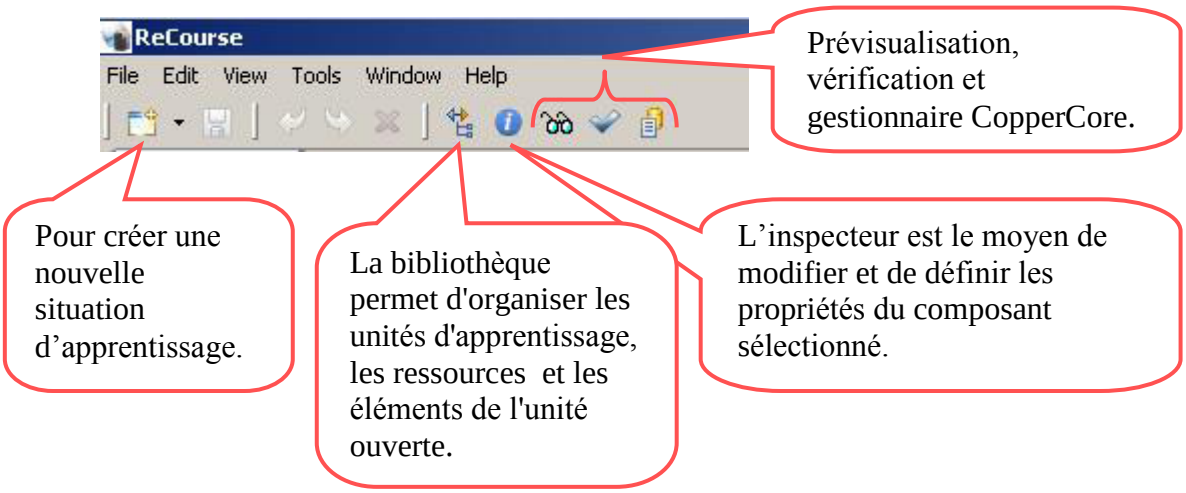


Fig.21. Description des fenêtres de 'ReCourse' (ReCourse, 2008).

'ReCource' offre la possibilité de prévisualiser les activités d'apprentissage, activités de soutien, les environnements et les ressources d'une situation d'apprentissage (exemple voir Fig.22.).

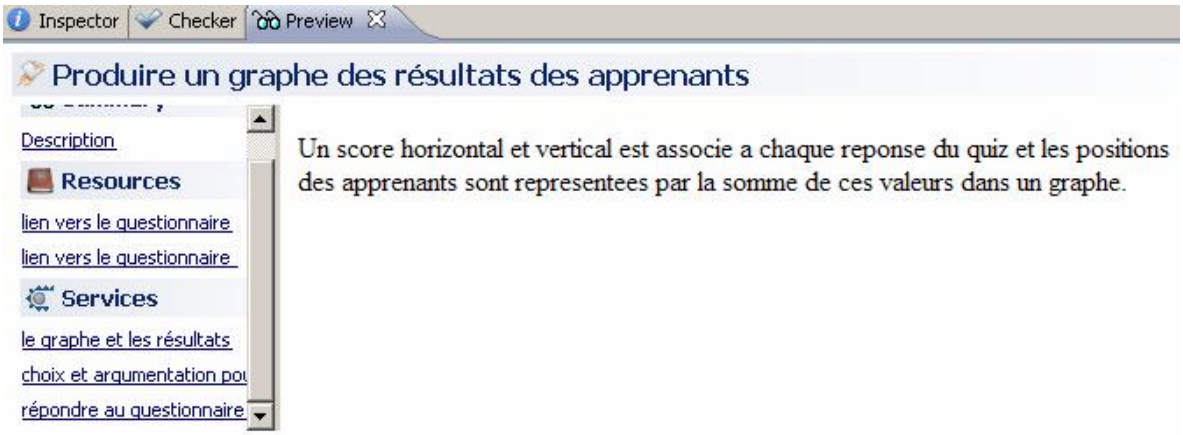


Fig.22. Prévisualisation de l'activité ' Produire un graphe des résultats des apprenants'.

4.7.2. Edition du script 'ArgueGraph' par 'ReCourse'

La description du script et ses phases a été présentée dans le premier chapitre. Dans ce qui suit on ne cite que les activités, les rôles, et les règles de terminaison utilisés pour chaque phase. La proposition de ces derniers est inspirée de la description du script. La description détaillée des phases, activités, rôles, environnements et ressources est présentée comme un fichier HTML. Les environnements et leurs relations ainsi que les ressources et le questionnaire sont présentés à part.

Phase 1: La première phase consiste d'une activité d'apprentissage ('Répondre à un questionnaire et argumenter vos choix') et d'une activité de soutien ('Observer l'activité des apprenants') (voir Fig.23.). Les rôles impliqués pour cette phase sont tous les apprenants et le tuteur. Cette phase se termine quand ces activités sont achevées (voir Fig.24.).

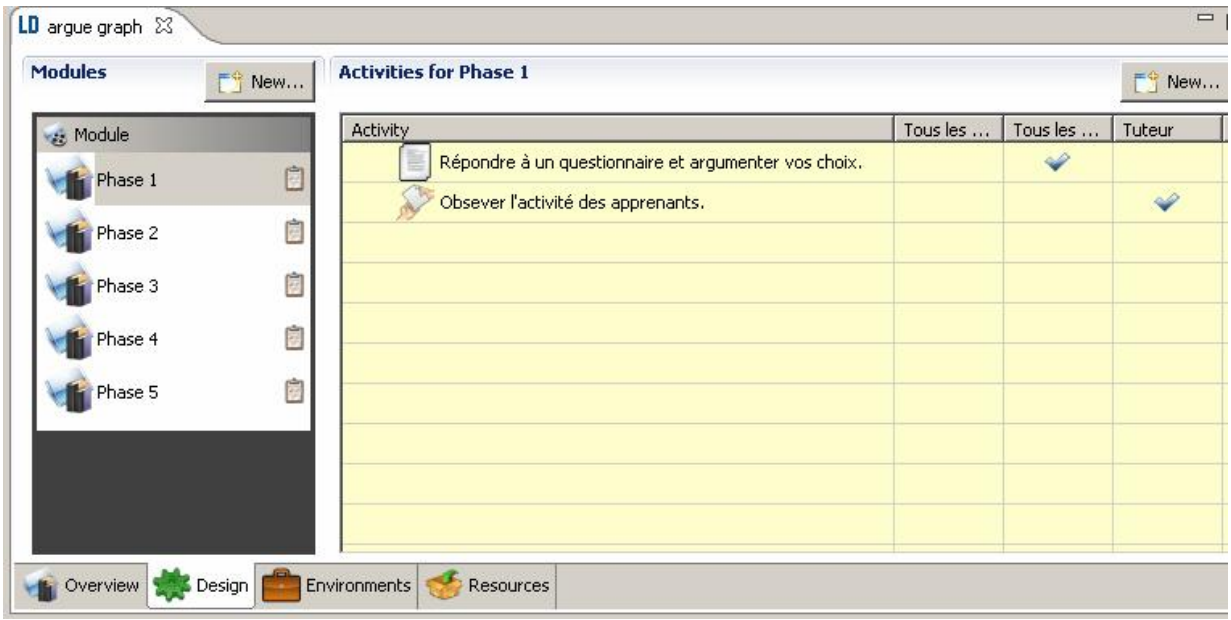


Fig.23. Les activités et la distribution des rôles de la phase 1.

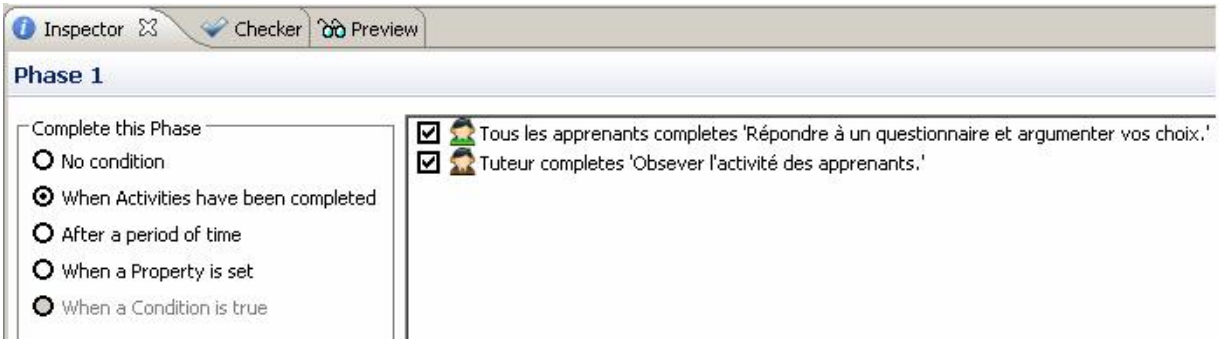


Fig.24. Les règles de terminaison et les rôles de la phase 1.

Phase 2: Dans cette phase on a une activité d'apprentissage ('Observer le graphe et discuter ses résultats') et deux activités de soutien ('Produire un graphe des résultats des apprenants 'et 'Former des paires d'étudiants qui ont des opinions les plus convergentes') qui sont assurées par un tuteur (Fig.25.). Cette phase se termine à la fin de ces activités (Fig.26.).

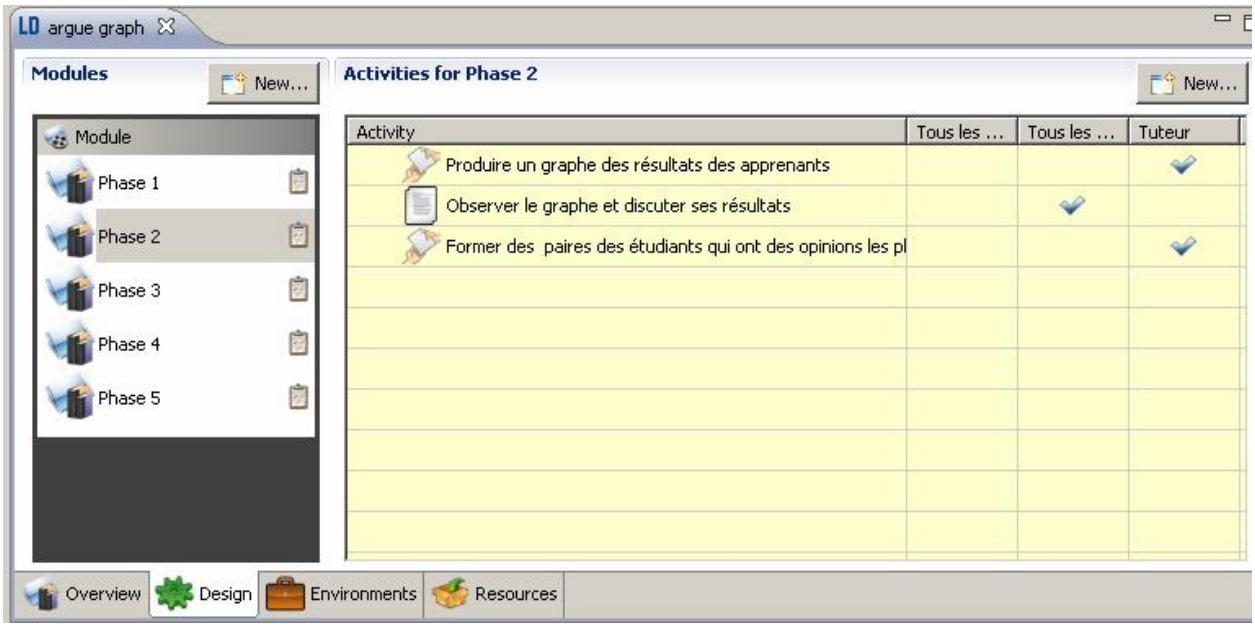


Fig.25. Les activités et la distribution des rôles de la phase 2.

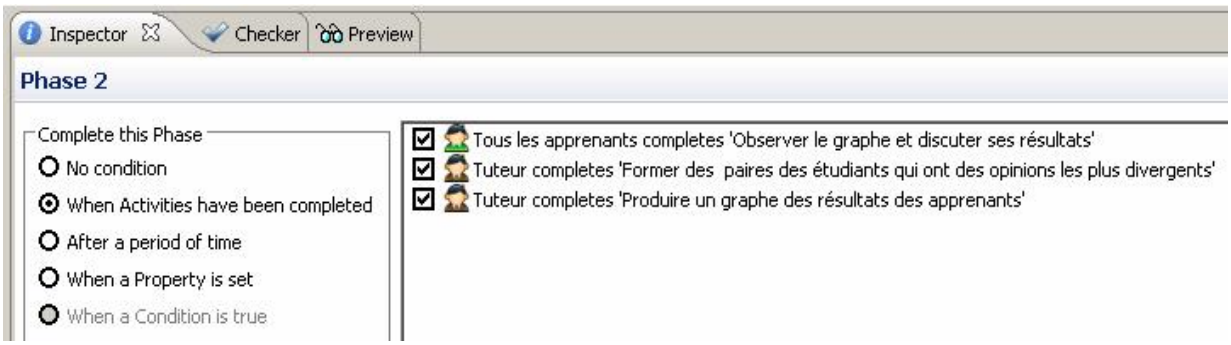


Fig.26. Les règles de terminaison et les rôles de la phase 2.

Phase 3: Dans cette phase (voir Fig.27.), on a une activité collaborative qui est présentée par une activité d'apprentissage ('répondre au même questionnaire et argumenter vos choix'). On a besoin aussi de deux activités de soutien ('Accéder aux réponses précédentes' et 'Observer l'activité des apprenants'). Les rôles utilisés sont : tous les paires et le tuteur. La phase se termine quand tous les rôles complètent leurs activités (Fig.28.).

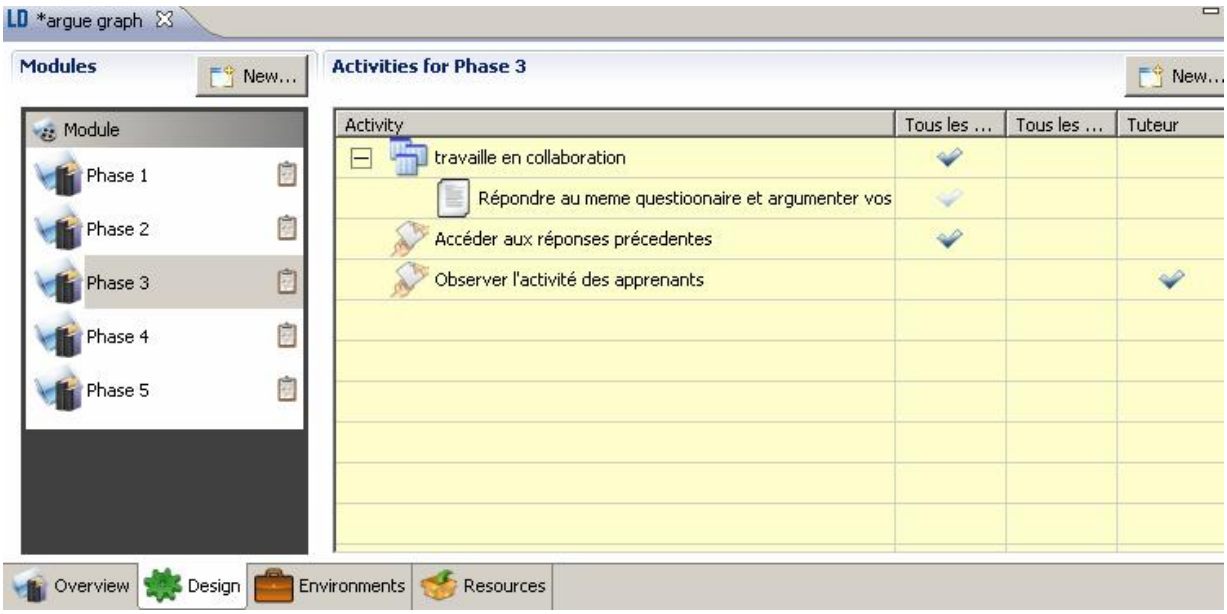


Fig.27. Les activités la distribution des rôles de la phase 3.

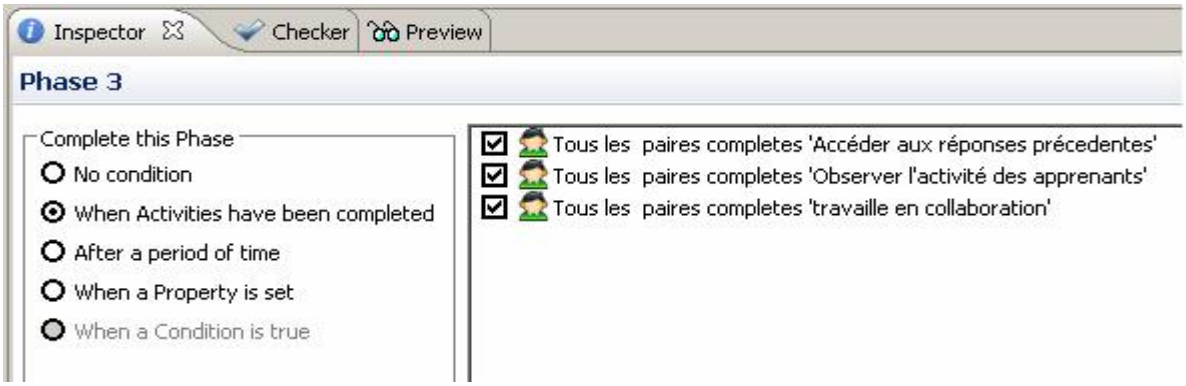


Fig.28. Les règles de terminaison et les rôles de la phase 3.

Phase 4: Le tuteur joue un rôle central dans cette phase. Il est le responsable des activités d'apprentissage ('Utiliser ces résultats au cour d'une session de débriefing' et 'Organiser et structurer les arguments collectés') et des activités de soutien ('Calculer les réponses données individuellement et en collaboration' et 'Observer les activités des apprenants'). Les apprenants participent aux activités ('Commenter les arguments') et ('Utiliser ces résultats au cour d'une session de débriefing') (voir Fig.29.). Les règles de terminaisons sont présentées par la figure Fig.30.

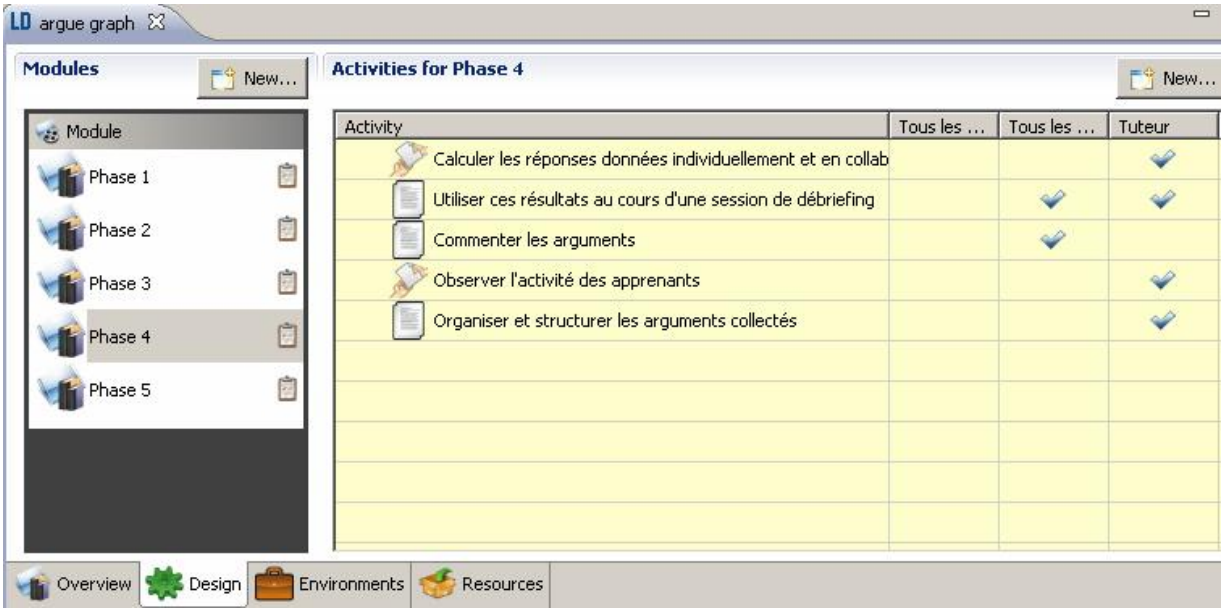


Fig.29. Les activités et la distribution des rôles de la phase 4.

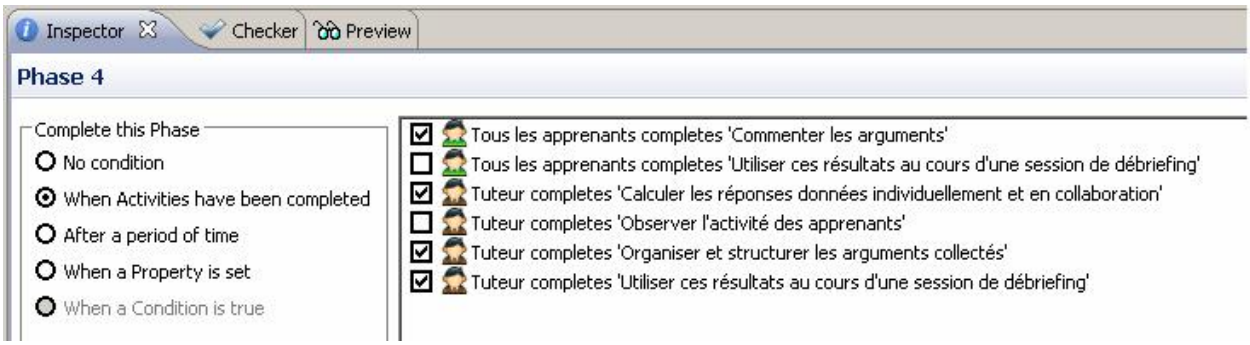


Fig.30. Les règles de terminaison et les rôles de la phase 4.

Phase 5: Dans cette phase, chaque apprenant doit rédiger une synthèse en utilisant les arguments collectés pour une question précise et le tuteur doit vérifier ces informations. Ces activités sont présentées par (‘écrire des synthèses des arguments collectés’ et ‘Vérifier les informations collectées’) (voir Fig.31.). Les règles de terminaison sont présentées par la figure Fig.32.

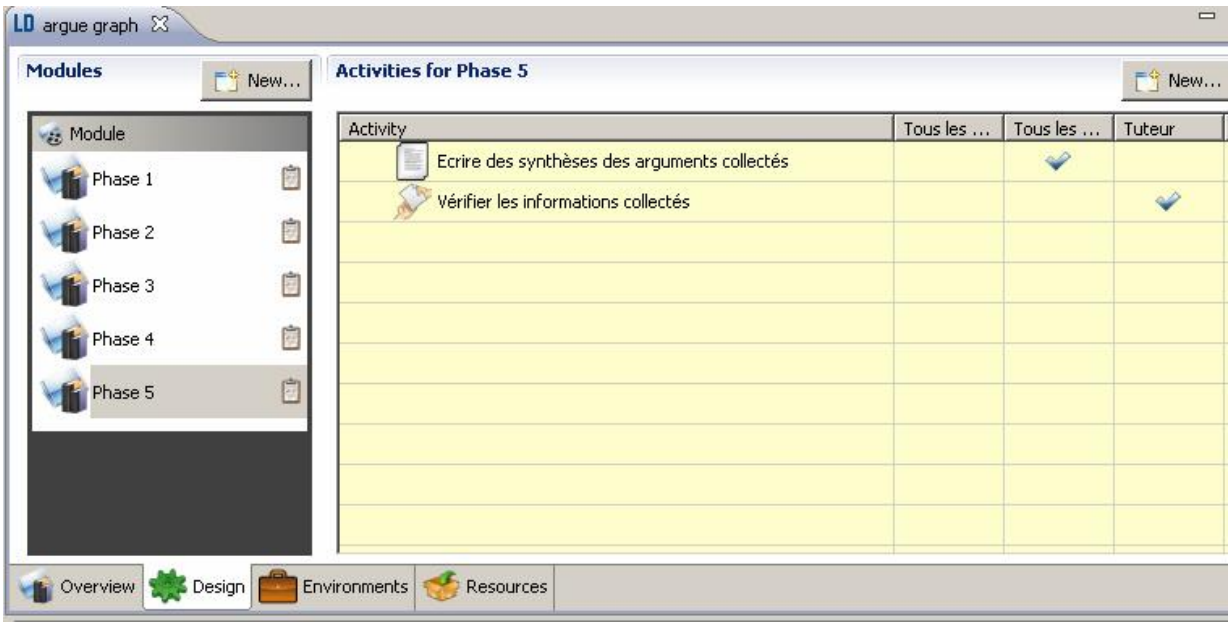


Fig.31. Les activités et la distribution des rôles de la phase 5.

Fig.32. Les règles de terminaison et les rôles de la phase 5.

4.7.3. Vérification de la conception de cette situation d'apprentissage

Cet éditeur nous permet de vérifier les éléments manquants et invalides dans une conception d'une situation d'apprentissage. Les résultats de notre cas sont présentés par la figure ci-dessous.

Fig.33. La vérification du script 'ArgueGraph'.

4.7.4. Les environnements

Les environnements sont des collections de ressources et de services qui sont utilisés dans les activités d'apprentissage et les activités de soutien. 'ReCourse' offre un éditeur graphique qui facilite la création des environnements. L'environnement du 'ArgueGraph' est présenté par la figure Fig .34.

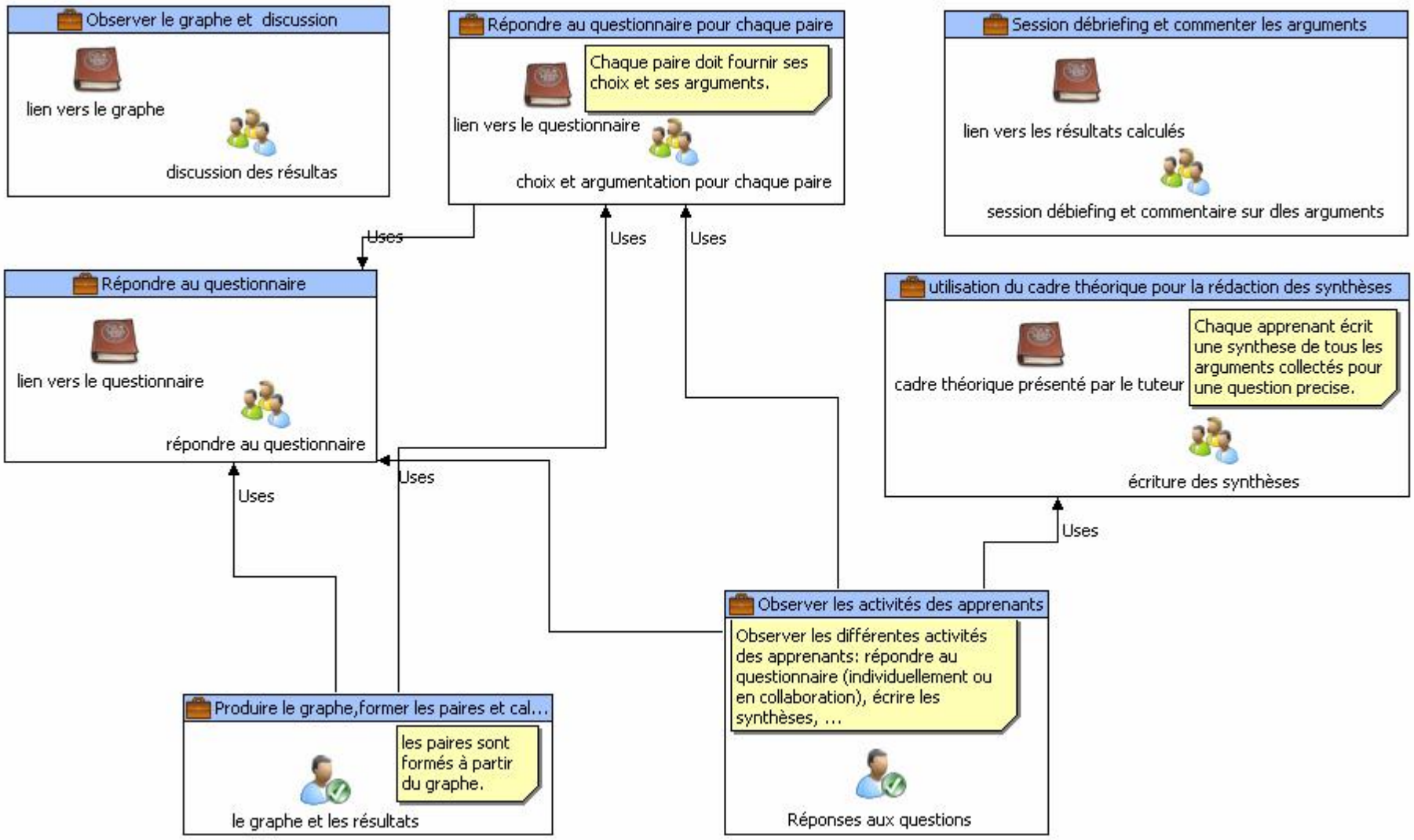


Fig.34. Les environnements utilisés pour le script ‘ArgueGraph’.

4.7.5. Les ressources

Les ressources se réfèrent à des adresses Web. Elles peuvent être créées, modifiées et organisées en utilisant l'éditeur de ressources.

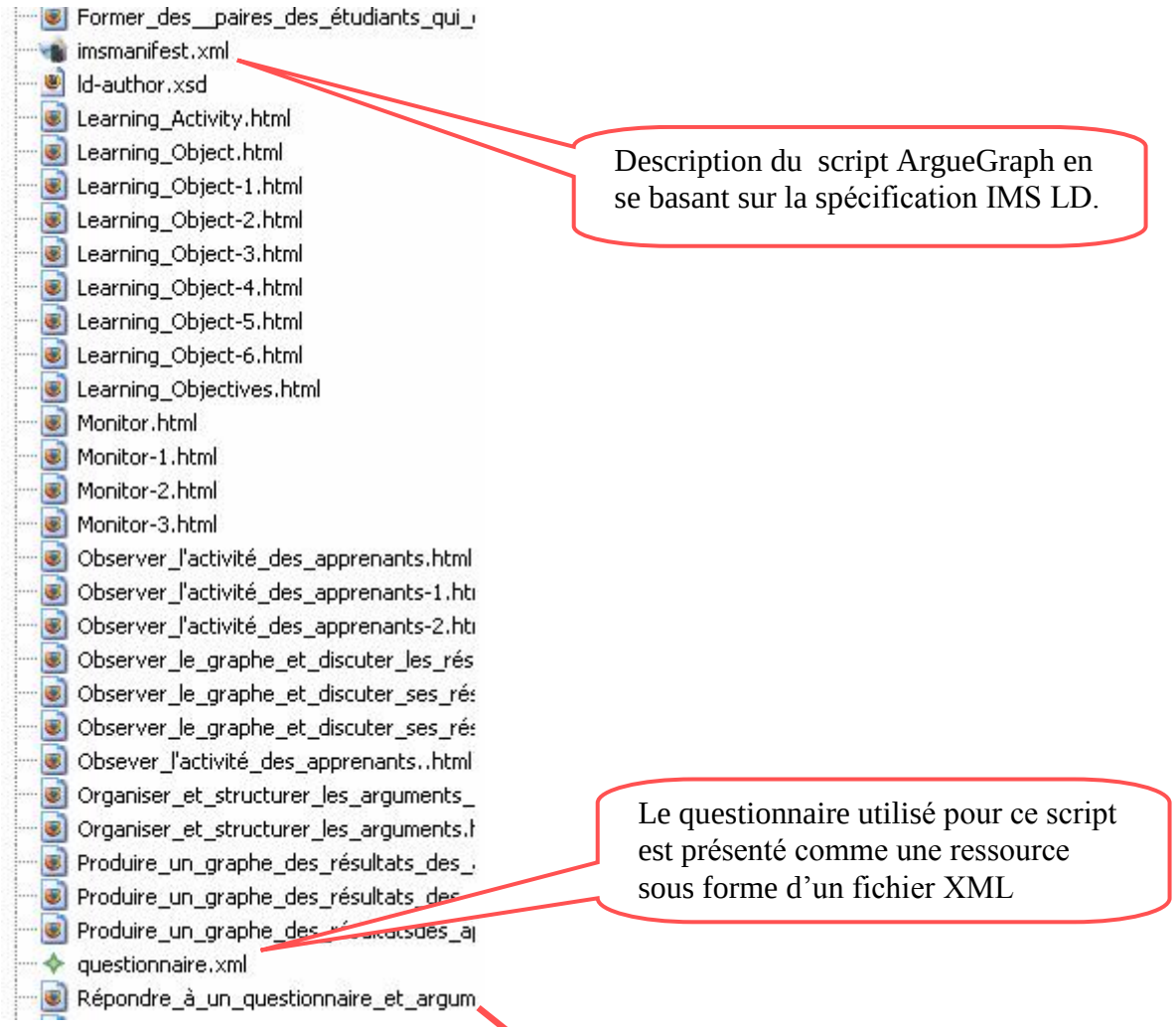


Fig.35. Une partie des ressources utilisées.

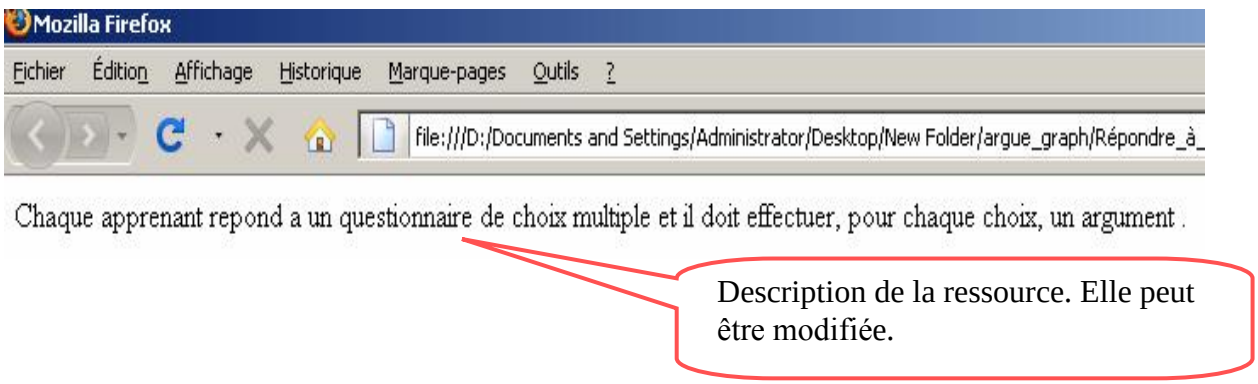


Fig.36. Une partie du fichier représentant la ressource indiquée.

4.7.6. Le questionnaire

Pour créer un nouveau questionnaire, on choisit nouveau Test QTI (Question and Test Interoperabilty). L'éditeur de test offre la possibilité de choisir plusieurs types de test (question /réponse, …etc.). Dans notre cas on choisit un questionnaire à choix multiples.

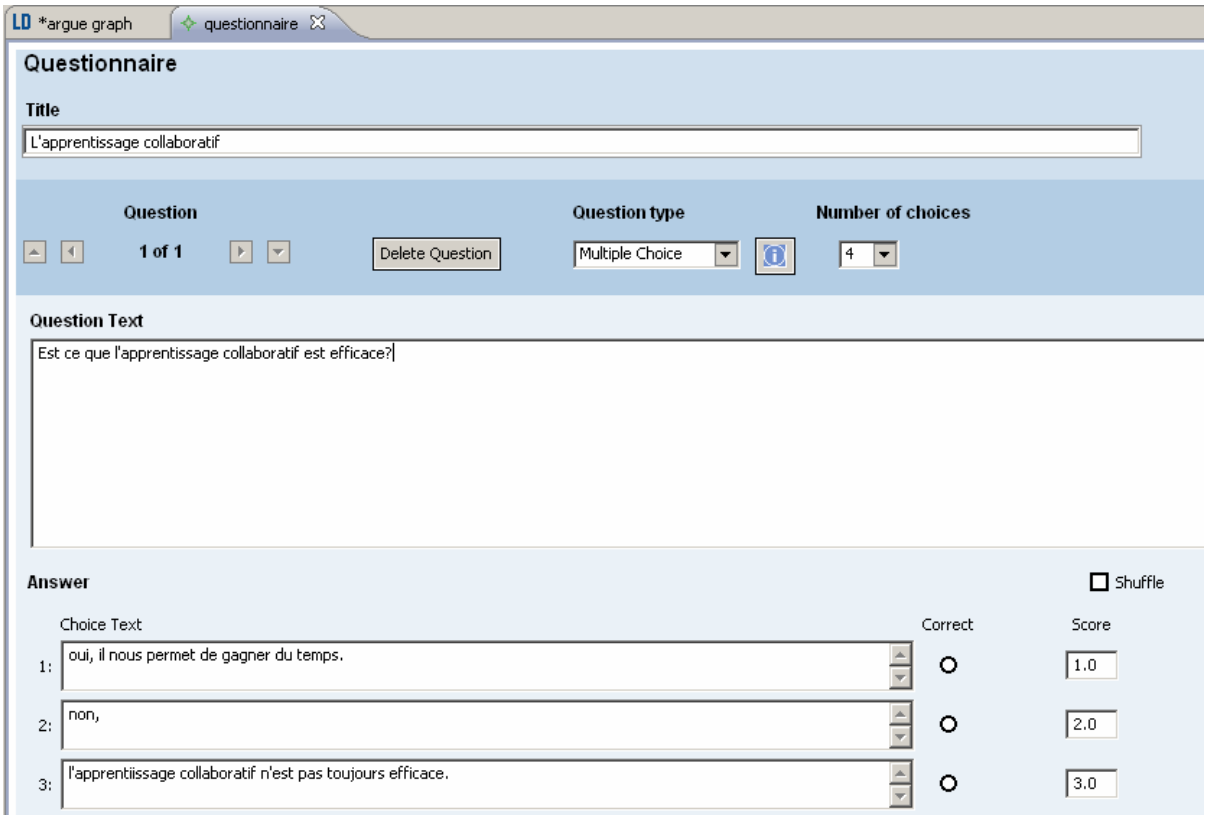


Fig.37. L'édition du questionnaire utilisé.

Une fois le questionnaire écrit, il se présente sous forme d'un fichier XML. Une partie de ce fichier est présentée ci-dessous :

```
<?xml version="1.0" encoding="UTF-8"?>
<!--This file was created with the TENCompetence QTI Assessment Item Editor on Wed
Feb 04 22:09:40 CET 2009-->
<assessmentItem xmlns="http://www.imsglobal.org/xsd/imsqti_v2p0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.imsglobal.org/xsd/imsqti_v2p0 imsqti_v2p0.xsd"
identifier="AI-4837b4a2-c440-4de3-abfd-0aa6a645c64c-6" title="" adaptive="false"
timeDependent="false">
<outcomeDeclaration identifier="SCORE" cardinality="single" baseType="integer" />
<responseDeclaration identifier="RESPONSE" cardinality="single"
baseType="identifier">
<mapping lowerBound="0.0" upperBound="4.0" defaultValue="0.0">
<mapEntry mapKey="SC-14c97d6e-88b7-4b70-a1c6-f010a76af666-9"
mappedValue="3.0" />
<mapEntry mapKey="SC-b53d478a-bf56-4429-b1ff-94d580e2340b-7"
```

```
mappedValue="1.0" />
<mapEntry mapKey="SC-e81a3266-fee8-48d5-b625-ccdaa9b6362a-8"
mappedValue="2.0" />
</mapping>
</responseDeclaration>
<itemBody class="textBasedMultipleChoice"><choiceInteraction
responseIdentifier="RESPONSE" shuffle="false" maxChoices="1">
<prompt>Est ce que l'apprentissage collaboratif est efficace?</prompt>
<simpleChoice identifier="SC-b53d478a-bf56-4429-b1ff-94d580e2340b-7">oui, il nous
permet de gagner du temps.</simpleChoice>
<simpleChoice identifier="SC-e81a3266-fee8-48d5-b625-ccdaa9b6362a-8">non,
</simpleChoice>
<simpleChoice identifier="SC-14c97d6e-88b7-4b70-a1c6-f010a76af666-
9">l'apprentiissage collaboratif n'est pas toujours efficace.</simpleChoice>
</choiceInteraction>
</itemBody>
….
```

4.7.7. La Spécification du script ‘ArgueGraph’

L’éditeur ‘ReCourse’ nous permet de générer la spécification IMS LD du script sous forme d’un fichier XML. Une partie de ce fichier est représentée ci-dessous.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Edited with ReCourse-->
<!--Created - Wed Jan 28 09:15:13 CET 2009-->
<!--Modified - Thu Feb 05 11:39:44 CET 2009-->
<manifest xmlns="http://www.imsglobal.org/xsd/imscp_v1p1"
xmlns:imsld="http://www.imsglobal.org/xsd/imsld_v1p0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ldauthor="http://www.tencompetence.org/ldauthor"
xsi:schemaLocation="http://www.imsglobal.org/xsd/imscp_v1p1
http://www.imsglobal.org/xsd/imscp_v1p1.xsd http://www.imsglobal.org/xsd/imsld_v1p0
http://www.imsglobal.org/xsd/IMS_LD_Level_A.xsd
http://www.tencompetence.org/ldauthor ld-author.xsd" identifier="manifest-5aa72d87-
1d00-4e22-8490-723467d61db0">
  <organizations>
    <imsld:learning-design identifier="ld-2cc221e5-c5c5-4274-9038-81f344664a33"
version="1.0.0" level="A" uri="http://www.yourURI.here/2390a467-53db-4ea7-b139-
9d2ead66b426" sequence-used="false">
      <imsld:title>argue graph</imsld:title>
      <imsld:learning-objectives>
        <imsld:item identifier="item-c8927c34-55ff-4dc8-8da8-c3e10cf09268"
isvisible="true" identifierref="Learning_Objectives">
          <imsld:title>Learning Objectives</imsld:title>
        </imsld:item>
      </imsld:learning-objectives>
      <imsld:components>
        <imsld:roles>
```

```
        <imsld:learner identifier="role-458270df-9517-4b21-b0ef-248d603bc2d4" min-
persons="6" max-persons="24">
          <imsld:title>Tous les apprenants</imsld:title>
        </imsld:learner>
        <imsld:learner identifier="role-6a2403a3-af3b-4b4d-be21-8f6db3623738" min-
persons="3" max-persons="12">
          <imsld:title>Tous les  paires</imsld:title>
        </imsld:learner>
        <imsld:staff identifier="role-d36df94f-d3ca-4f12-ac23-8032cd9b7120" min-
persons="1" max-persons="2">
          <imsld:title>Tuteur</imsld:title>
        </imsld:staff>
…
```

4.8. Réalisation des modèles suggérés

Pour la réalisation de ce modèle, on suggère l'utilisation d'un langage de programmation orienté agents parce que les langages orienté agent sont plus appropriés que les langages généraux quand le problème est spécifié comme un système multi-agents (Bordini et al., 2005). Nicholas R. Jennings pense qu'un problème complexe peut être considéré comme un système multi-agents (Jennings, 2001).

Le langage utilisé doit assurer quelques aspects importants. Parmi ces aspects, on suggère les six aspects présentés au chapitre 3 et qui sont : l'autonomie et le comportement, l'environnement (perception, action), l'intégration des entités non agent et la gestion des ressources, l'organisation, l'interaction et la coordination.

L'autonomie et le comportement : Le suivi des apprenants est persistant dans le temps et il se concentre sur l'observation des différents apprenants. Pour ces raisons on a besoin d'agents autonomes qui interagissent sans intervention d'autres agents (humains ou logiciels) pour qu'ils puissent suivre les apprenants d'une manière efficace. Par exemple les agents de suivi de traces doivent collecter les différentes traces des apprenants d'une manière autonome sans attendre l'ordre d'autres agents pour commencer leur travail.

L'environnement (perception, action) : Ces agents sont situés dans un environnement où ils évoluent et interagissent. Ils sont capables de le percevoir et d'agir sur lui. Il est considéré comme une partie intégrante à l'environnement d'apprentissage humain (EIAH).

L'intégration des entités non agent et la gestion des ressources : Afin d'accomplir leurs tâches et atteindre les objectifs de leurs conception, ils utilisent différentes ressources (par exemple : des structures de données,…) et même des entités non agents (des objets). Ces dernières (ressources et entités non agents) peuvent appartenir à l'environnement des agents qui sera dans ce cas le responsable de la gestion de leur accès.

L'organisation : Bien qu'ils forment d'une manière implicite une organisation, des groupes d'intérêts peuvent se former afin de permettre la coopération entre les agents d'un même groupe afin de résoudre d'éventuels problèmes. L'agent superviseur de collecte et les agents de collecte (les agents de collecte des traces brutes, les agents de collecte des

traces de production et les agents de collecte des traces additionnelles) forment un groupe qui a pour objectif la capture des différents indices des apprenants.

L'interaction et la coordination : Les agents ont besoin de communiquer afin d'accomplir leurs tâches. Par exemple, l'agent superviseur de collecte doit être en contact avec l'agent structuration : une fois la collecte terminée, les agents de collecte informent et envoient les traces collectées à l'agent superviseur qui doit être en contact avec l'agent structuration (par exemple il lui envoie un message qui l'informe du types de traces envoyées.)
La coordination est très importante pour gérer les dépendances entre ces agents. En effet, différents types de traces peuvent être captées de la même source et par conséquent les agents de collecte doivent coordonner leurs activités afin d'éviter la perte de quelques indices

4.9. Conclusion

La conception des scripts de collaboration n'est pas facile, pour cette raison, on suggère l'utilisation d'un modèle de conception des scripts de collaboration en tenant compte des comportements des apprenants et de leurs interactions. Notre approche de conception est incrémentale, basée sur l'utilisation des agents artificiels. Dans cette approche les résultats de l'évaluation du scénario produit sont analysés et interprétés en vue d'une adaptation, d'une remise en cause ou d'une amélioration de ce dernier. L'exécution du script doit se décliner dans un modèle de tâches et non à partir d'un modèle de contenus des ressources mises à la disposition des apprenants.

Le comportement de l'apprenant est observé au moyen de traces laissées par son activité et provenant de plusieurs sources. Dans notre approche on propose d'utiliser les agents pour réaliser les différentes étapes constituant le processus général de collecte et d'utilisation des traces d'apprentissage. Ces dernières sont classées en : traces brutes, de production et additionnelles.

La réalisation de nos modèles nous a amené à penser à l'utilisation d'un langage de programmation d'agents et à privilégier certains aspects (les six aspects) afin de développer les agents impliqués dans ces modèles. Toutefois, les langages d'agents existants ne couvrent pas tous les six aspects en même temps. Afin de mettre en pratique ces agents on compte proposer un langage de programmation qui fera l'objet d'une présentation au chapitre suivant.

Chapitre 5

Vers un langage De Programmation d'Agents Pour la Conception des Scripts

5.1. Introduction

La recherche dans le domaine des systèmes multi-agents (SMA) conduit au développement de nombreux outils et langages de programmation afin de faciliter leurs mises en œuvre et l'implémentation d'agents. Toutefois ces développements manquent encore de maturité et actuellement il n' y a pas de langage universellement adopté.

Dans ce chapitre, on propose une version initiale d'un langage de programmation afin que les agents à développer pour la conception des scénarios de collaboration, assurent les six aspects présentés précédemment. Pour faciliter la conception de ce langage on utilise le convertisseur BNF.

Ce convertisseur est un outil de construction des compilateurs multi-lingue. Il utilise des grammaires écrites par la notation LBNF (Labelled BNF), et génère les composants d'un compilateur front-end. Parmi les composants générés, un fichier squelette qui peut être utilisé comme point de départ pour la construction du compilateur back-end.

5.2. Généralités

5.2.1. Un compilateur

Un compilateur (Comp, 2008) est un programme, qui traduit un programme écrit dans un langage de programmation L (appelé langage source) à un programme équivalent écrit dans un autre langage L'(appelé langage cible ou sortie). En général L est un langage évolué et L'est un langage moins expressif.

5.2.2. Le Compilateur front-end

Le compilateur front-end comprend les phases suivantes (Comp, 2008):

1. Analyse lexicale (lexing): elle décompose un code source en des unités élémentaires appelés: unités lexicales ou tokens (par exemple : des mot clés, des identifiants,…etc.) et les coder.
2. Analyse syntaxique (parsing): elle identifie les structures syntaxiques du code source (l'ordre des tokens) et elle détermine si une suite de tockens peut être engendrée par les règles de la grammaire.
3. Analyse sémantique: elle identifie le sens du programme et commence à préparer la sortie. Elle permet une vérification de type et l'introduction de la plupart des erreurs de compilation.
4. Génération du code intermédiaire : elle permet de créer un programme équivalent à l'orignal dans un langage intermédiaire.

5.2.3. Le Compilateur back-end

Bien qu'il existe des applications où seul le compilateur front-end est nécessaire, comme les outils de vérification de langage statique, un véritable compilateur utilise la représentation intermédiaire générée par le front-end dans le back-end, afin de produire un programme fonctionnel équivalent dans la langue de sortie (Comp, 2008). Ceci se fait en plusieurs étapes:

1. Analyse du compilateur: C'est le processus de collecte d'information sur le programme à partir de la représentation intermédiaire des fichiers sources d'entrée.
2. Optimisation: la représentation intermédiaire du langage est transformée en des formes fonctionnellement équivalentes mais plus courtes.
3. génération du Code : Le langage intermédiaire transformé est traduit dans le langage de sortie (généralement le langage machine du système).

5.2.4. Un pretty-printer

Un pretty-printer est un programme qui sert à redisposer des données ou des programmes sous une forme agréable et facile à lire. Par exemple, un pretty-printer spécialisé pour le langage C pourrait recevoir ce programme mal disposé :

```
#include <stdio.h>
int main(int argc,char*argv[]){printf(
"Hello world!\n");return 0;}
```

Et produire le même programme sous cette forme qui est bien plus lisible :

```
#include <stdio.h>
int main(int argc, char *argv[])
{
printf("Hello world!\n");
return 0;
}
```

5.3. LBNF

5.3.1. Définition

Une grammaire LBNF (**Labelled Backus Naur Form**) (Forsberg & Ranta, 2003) est un ensemble de règles étiquetées. C'est une grammaire BNF où chaque règle a une étiquette.

Cette étiquette est utilisée pour la construction d'un arbre syntaxique où les non-terminaux de la règle, pris dans le même ordre, introduisent les sous arbres.

Une règle LBNF a la forme suivante (exprimée par les expressions régulière; Annexe A donne une définition complète de la notation BNF) (Forsberg & Ranta, 2003):

Ident "." Ident "::=" (Ident | String)* ";"

À l'exception du premier identifiant représentant l'étiquette de la règle, celle-ci est une règle BNF ordinaire, avec les symboles terminaux écrits entre guillemets et non-terminaux écrits sans les guillemets.
Le deuxième identifiant qui suit l'étiquette est le symbole de catégorie. La flèche de production est représentée par le symbole (::=), introduisant la liste des éléments de production. Pour satisfaire Haskell, C et Java, l'étiquette et le symbole de catégorie sont imposées comme des suites de lettres non vides et commençant par une majuscule.

5.3.2. Pourquoi le choix de LBNF et non BNF?

En général les langages de programmation sont définis par une grammaire BNF, qui est considérée comme le format standard pour la spécification et la documentation de ces langages (Ranta, 2005). Mais ce n'est pas toujours le cas pour deux raisons.

La première raison est qu'un langage peut nécessiter des méthodes plus puissantes (par exemple, les langues avec des règles layout). La deuxième est que, lors de l'analyse syntaxique, on veut faire autres choses (par exemple : le contrôle de types, etc.)

5.3.3. Les conventions de LBNF

Pour éviter de générer une syntaxe abstraite trop détaillée, sans détruire la nature déclarative et la simplicité de LBNF, les auteurs dans (Forsberg & Ranta, 2003) (qui sera la référence principale pour l'introduction de ces conventions) enrichissent cette dernière avec quatre conventions ad hoc, qui sont décrites dans les sous sections suivantes:

- Les types de base prédéfinis.
- Semantic dummies.
- Les niveaux de priorité.
- Les listes Polymorphiques.

Les types de base prédéfinis

La grammaire LBNF permet la définition des types de base, tels que entier et caractère, par exemple:

```
Char_a. Char ::= "a" ;
Char_b. Char ::= "b" ;
```

Toutefois, cela est très lourd et inefficace. C'est pour cette raison que la LBNF est étendue avec des types de base prédéfinis, et représente leurs grammaire comme une partie de la structure lexicale. Ces types sont les suivants, tels que définis par les expressions régulières du LBNF:

Integer of integers, defined
digit+
Double of floating point numbers, defined
digit+ '.' digit+ ('e' '-'? digit+)?
Char of characters (in single quotes), defined
'\'' ((char - ["'\\"]) | ('\\' ["'\\nt"])) '\''
String of strings (in double quotes), defined
'"' ((char - ["\"\\"]) | ('\\' ["\"\\nt"]))* '"'
Ident of identifiers, defined
letter (letter | digit | '_' | '\'')*

Semantic dummies

Une règle sémantique est considérée comme 'semantic dummy', si elle n'apporte aucune différence sémantique. Par exemple, une règle BNF qui permet à l'analyseur syntaxique d'accepter des points virgule en plus après des déclarations:

Stm ::= Stm ";" ;

Donc, afin d'éviter sa représentation dans la syntaxe abstraite, dans LBNF, l'étiquette peut être remplacée par un trait de soulignement ' _. ' qui n'ajoute rien à l'arbre syntaxique. Ainsi, la règle précédente peut être écrite (en LBNF) comme suit

_ . Stm ::= Stm ";" ;

Semantic dummies ne laissent pas de trace dans le pretty-printer.

Les niveaux de priorité

Le niveau de priorité réglemente l'ordre de l'analyse syntaxique, y compris l'associativité. Les parenthèses, aussi, permettent d'élever le niveau d'une expression à un autre plus élevé.

Ils sont exprimés en BNF en utilisant des variantes indexées de catégories, par exemple les règles suivantes:

Exp3 ::= Integer ;
Exp2 ::= Exp2 "*" Exp3 ;
Exp ::= Exp "+" Exp2 ;
Exp ::= Exp2 ;
Exp2 ::= Exp3 ;
Exp3 ::= "(" Exp ")" ;

L'étiquetage de ces règles crée une grammaire souhaitée, mais elle est encombrée par des distinctions de type (entre Exp, Exp2, et Exp3) et des constructeurs (pour les trois dernières règles) sans contenu sémantique. Comme solution, le BNFC différencie les symboles de catégorie de ceux qui ne sont que des variantes indexés de ces derniers. Un symbole de catégorie peut se terminer par un indice entier (par exemple : 2 et 3 dans Exp2,

Exp3), et est ensuite traité comme un type synonyme correspondant au symbole non indexé (Exp). Donc, Exp2 et Exp3 sont des variantes indexées de Exp.

Les transitions entre les variantes indexées sont considérées comme des 'Semantic dummies' et, par conséquent, ces variantes sont représentées par des traits de soulignement. L'exemple de grammaire ci-dessus peut maintenant être étiqueté comme suit:

```
EInt.    Exp3 ::=  Integer ;
ETimes.  Exp2 ::=  Exp2 "*" Exp3 ;
EPlus.   Exp  ::=  Exp "+" Exp2 ;
_.       Exp  ::=  Exp2 ;
_.       Exp2 ::=  Exp3 ;
_.       Exp3 ::=  "(" Exp ")" ;
```

Les listes Polymorphiques

Dans LBNF, Les types de listes monomorphes sont définis comme suit:

```
NilDef. ListDef ::= ;
ConsDef. ListDef ::= Def ";" ListDef ;
```

LBNF permet l'utilisation des constructeurs de listes de Haskell comme des étiquettes de règles et les crochets dans les symboles de catégorie afin de définir des listes polymorphes.

```
[]. [Def] ::= ;
(:). [Def] ::= Def ";" [Def] ;
```

Comme règle générale, on a:

[C], la catégorie des listes de type C.
[] et (:), les étiquettes des règles, respectivement, Nil et Cons,
(:[]), l'étiquette de la règle pour une liste d'un seul élément.

5.3.4. Les Pragmas LBNF

Certaines caractéristiques ne peuvent pas être exprimées naturellement dans une grammaire BNF. Ainsi l'introduction de la notion de pragmas dans la notation LBNF permet de résoudre ce problème. Ces pragmas sont les suivants:

- Pragma du commentaire.
- Pragmas internes.
- Pragmas du Tokens.
- Pragma des Points d'entrée.

Pragma du commentaire

Ce pragma montre le type de commentaires que le langage possède. Il est représenté par l'expression régulière suivante

"comment" String String? ";"

La première chaîne de caractères indique le symbole d'introduction du commentaire, tandis que la deuxième (facultatif) marque sa fin (en cas d'absence de cette dernière, la fin est marquée par une nouvelle ligne.)

Pragmas internes

Ils montrent les structures qu'on veut inclure dans la syntaxe abstraite, mais ne font pas partie de la syntaxe concrète. Par conséquent, ils ne seront pas pris en compte par l'analyse syntaxique.

Ces pragmas sont définis comme suit :

"internal" Rule ";"
(Rule indique une règle LBNF normale).

Pragmas de Tokens

Ils sont utilisés pour définir de nouveaux types de token (mot clé). Par exemple, pour faire une distinction entre les lettres minuscules et les lettres majuscules, on introduit deux nouveaux types de tokens : LIdent et UIdent.

token UIdent (upper (letter | digit | '_')*) ;
token LIdent (lower (letter | digit | '_')*) ;

Pragma des Points d'entrée

Il permet à l'utilisateur de définir l'analyseur syntaxique effectivement exporté, car le BNFC génère, par défaut, un analyseur syntaxique pour chaque catégorie dans la grammaire. Le format est le suivant :

entrypoints (Ident ",")* Ident ;

5.3.5. Les inconvénients de l'utilisation de LBNF

L'approche LBNF présente les inconvénients suivants (Forsberg & Ranta, 2003) :

1. On ne peut pas définir complètement certains langages.
2. Les modules générés ne sont pas aussi bons que ceux écrits par les programmeurs.

5.4. Le convertisseur BNF

5.4.1. Définition

Le convertisseur BNF (BNFC) est un outil de construction des compilateurs (Pellauer et al., 2003). Il est utilisé pour la définition des langages formels. Il accepte en entrée une grammaire spécifiée par des descriptions LBNF pour générer les composants nécessaires

du compilateur front-end du langage cible (par exemple : l'analyseur lexical et l'analyseur syntaxique).

Les composants générés sont les suivants:

• Les structures de données de l'arbre syntaxique abstrait.
• Les spécifications de l'analyseur lexical (Lexer) et l'analyseur syntaxique (parser).
• Pretty printer et un fichier squelette (traversal skeleton).
• Un fichier pour le Test et Makefile
• Documentation du language

Le BNFC est enrichi par une couche abstraite permettant la génération de code multilingue [tec rapport]. Il est possible d'avoir un compilateur front-end en Haskell, Java, C ou C + + (à partir de la version 2.0).

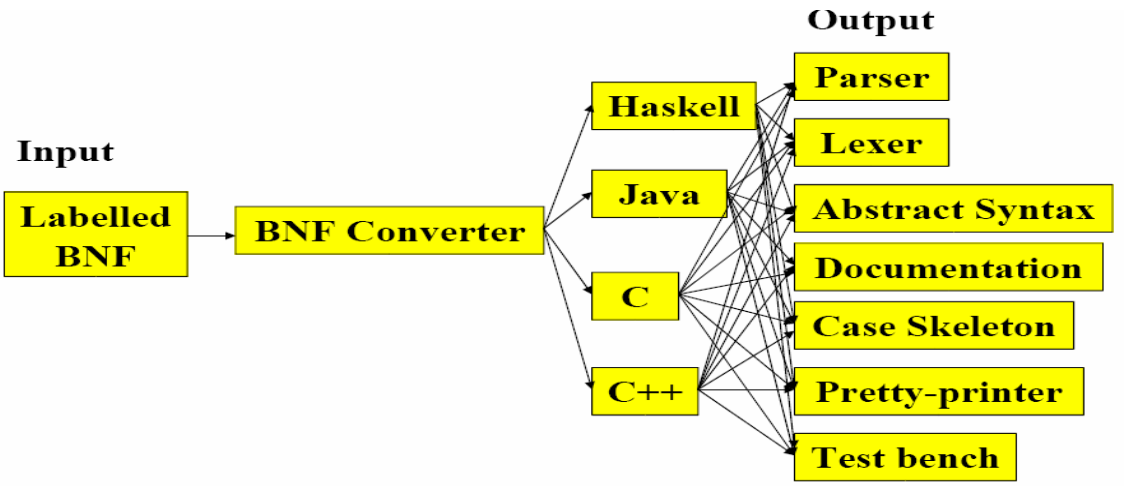


Fig.38. Schéma du BNFC (Forsberg & Ranta, 2004).

5.4.2. Les conditions d'utilisation de BNFC

Pour avoir les résultats désirés (les modules mentionnés), le langage doit satisfaire les conditions suivantes (Pellauer et al., 2003) :

1. La structure lexicale du langage doit être décrite par une expression régulière. Cette condition est déjà satisfaite dès lors que le mécanisme de l'analyseur lexical est un automate à états finis.
2. Le langage doit avoir un analyseur syntaxique LALR (1) (sauf pour la génération en Haskell avec le support GLR). Toutefois, cette exigence est inhérente aux outils qui sont utilisés pour produire l'analyseur syntaxique.
3. La construction des modules du compilateur front-end est séquentielle. L'implémentation du langage peut être séparé en différentes étapes (par exemple analyse lexicale, analyse syntaxique…) et le résultat d'une étape est utilisé pour l'étape suivante.

5.4.3. Les avantages de BNFC

Les avantages de BNFC sont comme suit (Pellauer et al., 2003):

1. L'augmentation du niveau d'abstraction permet au BNFC la vérification de la grammaire

des problèmes, plutôt que d'essayer de vérifier le code écrit directement dans un langage de programmation (comme le langue C).

2. Les composants sont générés pour interagir correctement sans aucun travail supplémentaire de l'utilisateur.
3. Les paquets générés (par exemple : la syntaxe abstraite, pretty-printer, …etc.) peuvent être utilisés pour encourager l'utilisation du nouveau langage.

5.5. Vers un langage de programmation orienté agent

5.5.1. La grammaire proposée

En utilisant le formalisme LBNF, on suggère la grammaire (APL) suivante:

```
A_Agent. Agent ::= "agent" Ident "{" "environments:" [Environment] "behaviours:"
                        [Behaviour] "internal memory:" [Ident]"}";

A_Environment. Environment ::= "environment" Ident "{" "Inbox_matrix:" [Ident]
                                    "Outbox_matrix:" [Ident] "}";

A_Behaviour. Behaviour ::= "behaviour" Ident "{" Program "}";

terminator Behaviour "";
terminator Environment "";

Progr.  Program ::= [External_dec] ;

(:[]).  [External_dec] ::=  External_dec;
(:).    [External_dec] ::=  External_dec [External_dec];

Afunc.  External_dec ::=  Func_def ;
Global. External_dec ::=   Dec ;

OldFunc.       Func_def ::= [Dec_specifier]Declarator[Dec]Compound_stm;
NewFunc.       Func_def ::= [Dec_specifier] Declarator Compound_stm ;
OldFuncInt.    Func_def ::= Declarator [Dec] Compound_stm ;
NewFuncInt.    Func_def ::= Declarator Compound_stm ;

NoDeclarator.  Dec ::= [Dec_specifier] ";";
Declarators.       Dec ::= [Dec_specifier] [Init_declarator] ";" ;

(:[]).  [Dec] ::= Dec;
(:).    [Dec] ::= Dec [Dec];

(:[]). [Dec_specifier]  ::= Dec_specifier ;
(:).   [Dec_specifier]  ::= Dec_specifier [Dec_specifier] ;

Type.        Dec_specifier ::=  Type_specifier ;
```

```
Storage.    Dec_specifier ::=  Storage_class_specifier ;

(:[]). [Init_declarator] ::= Init_declarator ;
(:).    [Init_declarator] ::= Init_declarator "," [Init_declarator] ;

OnlyDecl. Init_declarator  ::= Declarator ;
InitDecl.    Init_declarator ::=  Declarator "=" Initializer;

Tvoid.       Type_specifier  ::=  "void";
Tchar.       Type_specifier ::=  "char";
Tshort.      Type_specifier ::=  "short";
Tint.        Type_specifier  ::=  "int";
Tlong.       Type_specifier ::=  "long";
Tfloat.      Type_specifier ::=   "float";
Tdouble.     Type_specifier ::=  "double";
TBool.       Type_specifier  ::=  "bool" ;
TString.     Type_specifier ::=   "string"
Tsigned.     Type_specifier ::=  "signed";
Tunsigned. Type_specifier ::=  "unsigned";
Tstruct.     Type_specifier ::=   Struct_or_union_spec;
Tenum.       Type_specifier ::=   Enum_specifier;
Tname.       Type_specifier ::=  "Typedef_name";

MyType.            Storage_class_specifier  ::= "typedef";
GlobalPrograms. Storage_class_specifier ::=  "extern";
LocalProgram.    Storage_class_specifier ::=  "static" ;
LocalBlock.       Storage_class_specifier ::=  "auto" ;
LocalReg.          Storage_class_specifier ::=  "register" ;

Tag.        Struct_or_union_spec  ::=  Struct_or_union Ident "{" [Struct_dec] "}" ;
Unique.    Struct_or_union_spec ::=  Struct_or_union "{" [Struct_dec] "}";
TagType. Struct_or_union_spec  ::=  Struct_or_union Ident ;

Struct.  Struct_or_union ::= "struct";
Union.  Struct_or_union ::= "union";

(:[]). [Struct_dec] ::= Struct_dec ;
(:).    [Struct_dec] ::= Struct_dec [Struct_dec] ;

Structen. Struct_dec ::= [Spec_qual] [Struct_declarator] ";" ;

(:[]). [Spec_qual] ::= Spec_qual ;
(:).    [Spec_qual] ::= Spec_qual [Spec_qual];

TypeSpec. Spec_qual ::= Type_specifier ;
QualSpec.  Spec_qual ::= Type_qualifier;

(:[]). [Struct_declarator] ::=  Struct_declarator;
(:).    [Struct_declarator] ::=  Struct_declarator "," [Struct_declarator];
```

```
Decl.        Struct_declarator ::=  Declarator;
Field.       Struct_declarator ::=  ":" Constant_expression;
DecField.  Struct_declarator ::=   Declarator ":" Constant_expression ;

EnumDec.     Enum_specifier ::=  "enum" "{" [Enumerator] "}" ;
EnumName.  Enum_specifier ::=  "enum" Ident "{" [Enumerator] "}" ;
EnumVar.      Enum_specifier ::=  "enum" Ident ;

(:[]). [Enumerator] ::= Enumerator ;
(:).    [Enumerator] ::= Enumerator "," [Enumerator] ;

Plain.       Enumerator  ::=  Ident ;
EnumInit.  Enumerator ::=  Ident "=" Constant_expression ;

BeginPointer. Declarator ::=  Pointer Direct_declarator ;
NoPointer.      Declarator ::=  Direct_declarator ;

Name.            Direct_declarator ::=   Ident ;
ParenDecl.      Direct_declarator ::=   "(" Declarator ")" ;
InnitArray.      Direct_declarator ::=  Direct_declarator "[" Constant_expression "]" ;
Incomplete.     Direct_declarator ::=  Direct_declarator "[" "]" ;
NewFuncDec. Direct_declarator ::=  Direct_declarator "(" Parameter_type ")" ;
OldFuncDef.   Direct_declarator ::=  Direct_declarator "(" [Ident] ")" ;
OldFuncDec.   Direct_declarator ::=  Direct_declarator "(" ")" ;

Point.               Pointer ::=  "*" ;
PointQual.         Pointer ::=  "*" [Type_qualifier] ;
PointPoint.        Pointer ::=  "*" Pointer ;
PointQualPoint.  Pointer ::=  "*" [Type_qualifier] Pointer ;

(:[]). [Type_qualifier] ::=  Type_qualifier ;
(:).    [Type_qualifier] ::=  Type_qualifier [Type_qualifier] ;

AllSpec. Parameter_type ::=  Parameter_declarations ;
More.     Parameter_type ::=  Parameter_declarations "," "..." ;

ParamDec.           Parameter_declarations ::=  Parameter_declaration ;
MoreParamDec.    Parameter_declarations ::= Parameter_declarations ","
                                                Parameter_declaration ;

OnlyType.         Parameter_declaration ::=  [Declaration_specifier] ;
TypeAndParam. Parameter_declaration ::=  [Declaration_specifier] Declarator ;
Abstract.           Parameter_declaration ::=  [Declaration_specifier] Abstract_declarator;

(:[]). [Ident] ::=  Ident ;
(:).    [Ident] ::=  Ident "," [Ident] ;

InitExpr.       Initializer ::=  Exp2 ;
InitListOne.  Initializer ::= "{" Initializers "}" ;
InitListTwo. Initializer ::=  "{" Initializers "," "}" ;
```

```
AnInit.     Initializers ::=  Initializer ;
MoreInit. Initializers ::= Initializers "," Initializer ;

PlainType.       Type_name ::=  [Spec_qual] ;
ExtendedType. Type_name ::=  [Spec_qual] Abstract_declarator ;

PointerStart.      Abstract_declarator ::=  Pointer ;
Advanced.          Abstract_declarator ::=  Dir_abs_dec ;
PointAdvanced.  Abstract_declarator ::=  Pointer Dir_abs_dec ;

WithinParentes.  Dir_abs_dec ::=  "(" Abstract_declarator ")" ;
Array.               Dir_abs_dec ::=  "[" "]" ;
InitiatedArray.    Dir_abs_dec ::=  "[" Constant_expression "]" ;
UnInitiated.        Dir_abs_dec ::=  Dir_abs_dec "[" "]" ;
Initiated.           Dir_abs_dec ::=  Dir_abs_dec "[" Constant_expression "]" ;
OldFunction.      Dir_abs_dec ::=  "(" ")" ;
NewFunction.     Dir_abs_dec ::=  "(" Parameter_type ")" ;
OldFuncExpr.     Dir_abs_dec ::=  Dir_abs_dec "(" ")" ;
NewFuncExpr.    Dir_abs_dec ::=  Dir_abs_dec "(" Parameter_type ")" ;

SDecl.          Stm ::=     Type Ident ";"  ;
SDeclT.         Stm ::=     Type [Ident] ";"  ;
LabelS.         Stm  ::=    Labeled_stm ;
CompS.          Stm  ::=    Compound_stm ;
ExprS.          Stm  ::=    Expression_stm ;
SelS.           Stm  ::=    Selection_stm ;
IterS.          Stm  ::=    Iter_stm ;
JumpS.          Stm  ::=    Jump_stm ;
SPrint.         Stm ::=     "print" Exp  ";" ;
SRead.          Stm ::=     "read" Exp  ";" ;

(:[]).   [Ident] ::= Ident ;
(:).     [Ident] ::= Ident "," [Ident] ;

SlabelOne.    Labeled_stm ::=  Ident ":" Stm ;
SlabelTwo.   Labeled_stm ::=  "case" Constant_expression ":" Stm ;
SlabelThree. Labeled_stm ::=  "default" ":" Stm;

ScompOne.    Compound_stm  ::=  "{" "}" ;
ScompTwo.   Compound_stm ::=  "{" [Stm] "}" ;
ScompThree. Compound_stm ::=  "{" [Dec] "}" ;
ScompFour.   Compound_stm ::=  "{" [Dec] [Stm] "}" ;

SexprOne.  Expression_stm  ::=  ";" ;
SexprTwo.  Expression_stm  ::=  Exp ";" ;

SselOne.     Selection_stm    ::=  "if" "(" Exp ")" Stm ;
SselTwo.    Selection_stm    ::=  "if" "(" Exp ")" Stm "else" Stm ;
SselThree.  Selection_stm    ::=  "switch" "(" Exp ")" Stm ;
```

```
SiterOne.      Iter_stm ::=   "while" "(" Exp ")" Stm;
SiterTwo.      Iter_stm ::=   "do" Stm "while" "(" Exp ")" ";" ;
SiterThree.    Iter_stm ::=   "for" "(" Expression_stm Expression_stm ")" Stm ;
SiterFour.     Iter_stm ::=   "for" "(" Expression_stm Expression_stm Exp ")" Stm;

SjumpOne.    Jump_stm ::=  "goto" Ident ";" ;
SjumpTwo.    Jump_stm ::=  "continue" ";" ;
SjumpThree. Jump_stm ::=   "break" ";" ;
SjumpFour.   Jump_stm ::=   "return" ";" ;
SjumpFive.   Jump_stm ::=   "return" Exp ";" ;

(:[]).       [Stm] ::=      Stm ;
(:).         [Stm] ::=      Stm [Stm];

token Unsigned ["123456789"] digit * ('u'|'U');

token Long ["123456789"] digit * ('l'|'L');

token UnsignedLong ["123456789"] digit * (('u"l')|('U"L'));

token Hexadecimal '0' ('x'|'X') (digit | ["abcdef"] | ["ABCDEF"])+;

token Octal '0'["01234567"]*;

token CDouble (((digit+ '.')|('.' digit+))(('e'|'E') ('-')? digit+)?)|
          (digit+ ('e'|'E') ('-')? digit+)|(digit+ '.' digit+ 'E' ('-')? digit+);

token CFloat (((digit+ '.' digit+)|(digit+ '.')|('.' digit+))(('e'|'E')('-')? digit+)?
                    ('f'|'F'))|((digit+ ('e'|'E')('-')? digit+)('f'|'F'));

token CLongDouble (((digit+ '.' digit+)|(digit+ '.')|('.' digit+))(('e'|'E')('-')?
              digit+)?('l'|'L'))|((digit+ ('e'|'E')('-')? digit+)('l'|'L'));

Ecomma.      Exp   ::=     Exp "," Exp2;
Eassign.     Exp2  ::=     Exp15 Assignment_op Exp2;
Econdition.  Exp3  ::=     Exp4 "?" Exp ":" Exp3;
Elor.        Exp4  ::=     Exp4 "||" Exp5;
Eland.       Exp5  ::=     Exp5 "&&" Exp6;
Ebitor.      Exp6  ::=     Exp6 "|" Exp7;
Ebitexor.    Exp7  ::=     Exp7 "^" Exp8;
Ebitand.     Exp8  ::=     Exp8 "&" Exp9;
Eeq.         Exp9  ::=     Exp9 "==" Exp10;
Eneq.        Exp9  ::=     Exp9 "!=" Exp10;
Elthen.      Exp10 ::=     Exp10 "<" Exp11;
Egrthen.     Exp10 ::=     Exp10 ">" Exp11;
Ele.         Exp10 ::=     Exp10 "<=" Exp11;
Ege.         Exp10 ::=     Exp10 ">=" Exp11;
Eleft.       Exp11 ::=     Exp11 "<<" Exp12;
Eright.      Exp11 ::=     Exp11 ">>" Exp12;
```

```
Eplus.          Exp12 ::=   Exp12 "+" Exp13;
Eminus.         Exp12 ::=   Exp12 "-" Exp13;
Etimes.         Exp13 ::=   Exp13 "*" Exp14;
Ediv.           Exp13 ::=   Exp13 "/" Exp14;
Emod.           Exp13 ::=   Exp13 "%" Exp14;
Etypeconv.      Exp14 ::=   "(" Type_name ")" Exp14;
Epreinc.        Exp15 ::=   "++" Exp15;
Epredec.        Exp15 ::=   "--" Exp15;
Epreop.         Exp15 ::=   Unary_operator Exp14;
Ebytesexpr.     Exp15 ::=   "sizeof" Exp15;
Ebytestype.     Exp15 ::=   "sizeof" "(" Type_name ")";
Earray.         Exp16 ::=   Exp16 "[" Exp "]" ;
Efunk.          Exp16 ::=   Exp16 "(" ")";
Efunkpar.       Exp16 ::=   Exp16 "(" [Exp2] ")";
Eselect.        Exp16 ::=   Exp16 "." Ident;
Epoint.         Exp16 ::=   Exp16 "->" Ident;
Epostinc.       Exp16 ::=   Exp16 "++";
Epostdec.       Exp16 ::=   Exp16 "--";
Evar.           Exp17 ::=   Ident;
Econst.         Exp17 ::=   Constant;
Estring.        Exp17 ::=   String;

Efloat.             Constant ::=    Double;
Echar.              Constant ::=    Char;
Eunsigned.          Constant ::=    Unsigned;
Elong.              Constant ::=    Long;
Eunsignlong.        Constant ::=    UnsignedLong;
Ehexadec.           Constant ::=    Hexadecimal;
Eoctal.             Constant ::=    Octal;
Ecdouble.           Constant ::=    CDouble;
Ecfloat.            Constant ::=    CFloat;
Eclongdouble.       Constant ::=    CLongDouble;
Eint.               Constant ::=    Integer;
internal Edouble.   Constant ::=    Double;
Especial.           Constant_expression ::= Exp3 ;

_.    Exp    ::=    Exp2 ;
_.    Exp2   ::=    Exp3 ;
_.    Exp3   ::=    Exp4 ;
_.    Exp4   ::=    Exp5 ;
_.    Exp5   ::=    Exp6 ;
_.    Exp6   ::=    Exp7 ;
_.    Exp7   ::=    Exp8 ;
_.    Exp8   ::=    Exp9 ;
_.    Exp9   ::=    Exp10 ;
_.    Exp10 ::=    Exp11 ;
_.    Exp11 ::=    Exp12 ;
_.    Exp12 ::=    Exp13;
_.    Exp13 ::=    Exp14 ;
_.    Exp14 ::=    Exp15 ;
```

```
_.    Exp15 ::=    Exp16 ;
_.    Exp16 ::=    Exp17 ;
_.    Exp17 ::=    "(" Exp ")" ;

 Address.        Unary_operator ::=    "&" ;
 Indirection.    Unary_operator ::=    "*" ;
 Plus.           Unary_operator ::=    "+" ;
 Negative.       Unary_operator ::=    "-" ;
 Complement.     Unary_operator ::=    "~" ;
 Logicalneg.     Unary_operator ::=    "!" ;

 (:[]).          [Exp2] ::=            Exp2 ;
 (:).            [Exp2] ::=            Exp2 "," [Exp2];

 Assign.         Assignment_op ::=     "=" ;
 AssignMul.      Assignment_op ::=     "*=" ;
 AssignDiv.      Assignment_op ::=     "/=" ;
 AssignMod.      Assignment_op ::=     "%=" ;
 AssignAdd.      Assignment_op ::=     "+=" ;
 AssignSub.      Assignment_op ::=     "-=" ;
 AssignLeft.     Assignment_op ::=     "<<=" ;
 AssignRight.    Assignment_op ::=     ">>=" ;
 AssignAnd.      Assignment_op ::=     "&=" ;
 AssignXor.      Assignment_op ::=     "^=" ;
 AssignOr.       Assignment_op ::=     "|=" ;

 comment "/*" "*/" ;
 comment "//";
 comment "#";
```

5.5.2. La génération de code

On propose de générer les différents composants du compilateur front-end en Java (mode1.5).Cette génération nécessite l'utilisation du JLex et CUP.

Jlex (Berk, 2000) est un générateur des analyseurs lexicaux pour Java. Un analyseur lexical accepte en entré une suite de caractères et produit en sortis des tokens. L'utilité de JLex est basée sur le modèle du générateur de l'analyseur lexical Lex (développé pour le système UNIX). JLex prend un fichier de spécification similaire à celui utilisé par Lex, puis crée un fichier source en Java pour l'analyseur lexical cible.

Cup (Hudson, 1999) est un générateur d'analyseurs syntaxiques pour Java. Il est utilisé pour générer des analyseurs LALR à partir de spécifications simples. CUP est écrit en Java et produit des analyseurs implémentés en Java.

Le BNFC utilise le nom de la grammaire comme nom de paquet du code Java généré. Les importants composants générés sont les suivants:

Le nom du fichier Description

Absyn/*.java	Le packet de l'arbre syntaxique abstrait.
APL.cup	Fichier CUP (la spécification d'analyseur syntaxique).
Ylex	Fichier JLex (la spécification d'analyseur lexical).
PrettyPrinter.java	Un Pretty Printer pour l'arbre syntaxique abstrait.
VisitSkel.java	Le code du squelette utilisé pour visiter l'arbre syntaxique. Il utilise le Design Pattern de Visiteur.
Test.java	Un programme test.
Makefile	Makefile utilisée pour compiler le fichier Test d'une manière facile.
Grammar document	Un fichier Latex.

5.5.3. La compilation du compilateur

Pour compiler le code Java généré, on utilise 'Makefile' pour obtenir les composants suivants:

Fichier	Description
Absyn/*.class	Les fichiers classes de la syntaxe abstraite.
Ylex.java	L'analyseur lexical généré par JLex
Ylex.class	La classe compilée de L'analyseur lexical.
parser.java	L'analyseur syntaxique généré par CUP.
sym.java	Les symboles de tokens de CUP.
parser.class, sym.class, CUP$parser$actions.class	Les classes compilées de l'analyseur syntaxique.
PrettyPrinter.class	Pretty-printer compilé.
Test.class	Le fichier test compilé.

5.5.4. Le test du compilateur Front-End

La classe Java Test.java peut être utilisée pour tester le résultat de la génération (Bringert, 2005). On commence par l'écriture d'un fichier dans le langage cible, puis on attribue ce fichier à la classe Test. Si l'analyse syntaxique est correcte, l'arbre de syntaxe abstraite (présentée sous forme Haskell) et le résultat du pretty_printer est visualisé.

.

```
$ java APL.Test Helloword.apl

Parse Succesful!

[Abstract Syntax]

(A_Agent "Helloword" [] [(A_Behaviour "start" (Progr [(Afunc (NewFunc [(Type Tvo
id)] (NoPointer (NewFuncDec (Name "init") (AllSpec (ParamDec (OnlyType [(Type Tv
oid)])))))) (ScompTwo [(ExprS (SexprTwo (Efunkpar (Evar "printf") [(Estring "Hell
o word")])))])))] ["mem"])

[Linearized Tree]

agent Helloword
{
  environments: behaviours: behaviour start
  {
    void init (void)
    {
      printf ("Hello word");
    }
  }
  internal_memory: mem}
```

Fig.39. le résultat du test du compilateur.

5.5.5. La construction de notre compilateur

La construction du compilateur spécifique au langage défini est basée sur l'utilisation de Design Pattern de visiteur. Ainsi, le fichier généré à partir de BNFC, VisitSkel.java, peut être utilisé comme base pour cette construction (Ranta, 2005). Ce fichier représente le squelette initial du design pattern de visiteur associé à cette grammaire sans codes qu'il appartient au programmeur de compléter avec un ensemble de procédures et de fonctions écrites en Java pour construire le compilateur. La figure présente une partie du fichier généré.

```
package APL;
import APL.Absyn.*;
/*** BNFC-Generated Visitor Design Pattern Skeleton. ***/
/* This implements the common visitor design pattern.
   Tests show it to be slightly less efficient than the
   instanceof method, but easier to use.
   Replace the R and A parameters with the desired return
   and context types.*/

public class VisitSkel
{
  public class AgentVisitor<R,A> implements Agent.Visitor<R,A>
  {
    public R visit(APL.Absyn.A_Agent p, A arg)
    {
      /* Code For A_Agent Goes Here */

      //p.ident_;
      for (Environment x : p.listenvironment_) {
      }
      for (Behaviour x : p.listbehaviour_) {
      }
      for (String x : p.listident_) {
      }

      return null;
    }

  }
  public class EnvironmentVisitor<R,A> implements Environment.Visitor<R,A>
  {
    public R visit(APL.Absyn.A_Environment p, A arg)
    {
      /* Code For A_Environment Goes Here */

      //p.ident_;
      for (String x : p.listident_1) {
      }
      for (String x : p.listident_2) {
```

Fig.40. une partie du fichier VisitSkel.java

Avant de présenter le compilateur désiré, on commence par définir les designs patterns en général puis et le design pattern visiteur.

Design Patterns

La définition d'un design pattern selon Gamma (Gamma et al., 1995) est la suivante :

> *« Un design pattern nomme, rend abstrait, et identifie les principaux aspects d'un design commun pour le rendre utile afin de créer des designs orienté-objet réutilisables. »*

Celle-ci est la plus acceptée par la communauté informatique, elle stipule qu'un design pattern définit précisément ce qui est un pattern (modèle) et comment il devrait être utilisé.

Il existe 23 modèles qui sont classés en trois catégories (Arnout, 2004):

- Les designs patterns de création.
- Les designs patterns structurels.
- Les designs patterns comportementaux.

➢ **Les designs patterns de création** : ils permettent plus de souplesse dans le processus d'instanciation.

➢ **Les designs patterns structurels** : ils sont utilisés pour assembler des éléments logiciels en structures plus large. Certains utilisent l'héritage pour avoir un lien permanent et statique entre les classes, tandis que d'autres s'intéressent à la composition dynamique des objets.

➢ **Les designs patterns comportementaux** : ils traitent les algorithmes, l'attribution des responsabilités entre les objets, et la communication entre ces derniers. Parmi ces patterns, on cite le design pattern visiteur qui est présenté ci-dessous.

- **Design Pattern de Visiteur :** Le Design Pattern Visiteur (Gamma et al., 1995) permet d'ajouter à des instances de classes différentes ayant un ancêtre commun, des opérations non prévues au départ sans surcharger la structure de base de l'objet. Il est souvent utilisé pour visiter les éléments d'un arbre syntaxique abstrait (AST).

Compilateur

➢ **Les types primitifs :** Dans cette version, notre compilateur assure les types suivant : entiers, doubles, caractère, string et booléen.

➢ **Les opérateurs et les expressions**

Dans la plupart des langages on trouve des expressions formées à l'aide d'opérateurs, et des instructions pouvant intervenir des expressions (comme l'instruction d'affectation).

- **Les opérateurs d'affectation**

Exemple 1 :

```
int i;
i = 7;
print i;
i+= 5;
print i;
i -= 3 ;
print i;
i *= 11;
print i ;
i /= 2 ;
i %= 6 ;
print i ;
```

```
$ java Mini/Interpret < ex.apl
7
12
9
99
1
```

Fig.41. le résultat de l'exemple 1.

- **Les opérateurs d'incrémentation et de décrémentation**

Exemple 2 :

```
int b ;
b = 2 ;
```

Exemple 3 :

```
double b ;
b = 2.25 ;
```

```
b = ++b ;
print "la valeur de ++b est: ";
print b ;
b = --(--b) ;
print "la valeur de --(--b) est: ";
print b ;
b = --b + 5;
print "la valeur de --b+5 est: ";
print b;
b = b-- + 5;
print "la valeur de b-- +5 est: ";
print b;
b= ++b*2+5 ;
print "la valeur de ++b*2+5 est: ";
print b;
```

```
$ java Mini/Interpret < ex.apl
la valeur de ++b est:
3
la valeur de --(--b) est:
1
la valeur de --b+5 est:
5
la valeur de b-- +5 est:
10
la valeur de ++b*2+5 est:
27
```

Fig.42. le résultat de l'exemple 2.

```
b = ++b ;
print "la valeur de ++b est: ";
print b ;
b = --(--b) ;
print "la valeur de --(--b) est: ";
print b ;
b = --b + 4.98;
print "la valeur de --b+4.98 est: ";
print b;
b = b-- + 4.98;
print "la valeur de b-- +4.98 est: ";
print b;
b= ++b*1.99+5.0 ;
print "la valeur de ++b*1.99+5 est: ";
print b;
```

```
$ java Mini/Interpret < ex.apl
la valeur de ++b est:
3.25
la valeur de --(--b) est:
1.25
la valeur de --b+4.98 est:
5.23
la valeur de b-- +4.98 est:
10.21
la valeur de ++b*1.99+5 est:
27.3079
```

Fig.43. le résultat de l'exemple 3.

- **Les opérateurs arithmétiques**

Exemple 4 :

```
int a, b, c, d, e ;

a = 5 + 8 ;
print "le resultat de a = 5+8 est: " ;
print a ;
b = 15 - a;
print "le resultat de b = 15-a est: " ;
print b;
c = b * a ;
print "le resultat de c = b*a est: " ;
print c ;
d = a % c ;
print "le resultat de d = a%c est: "    ;
print d ;
e = d / 3 ;
print "le resultat de e = d/3 est: " ;
print e ;
```

Exemple 5 :

```
double a, b, c, d, e ;

a = 5 + 8.75 ;
print "le resultat de a = 5+8.75 est: " ;
print a ;
b = 15.5 - a;
print "le resultat de b = 15.5-a est: " ;
print b;
c = b * a ;
print "le resultat de c = b*a est: " ;
print c ;
d = a % c ;
print "le resultat de d = a%c est: "    ;
print d ;
e = d / 3 ;
print "le resultat de e = d/3 est: " ;
print e ;
```

```
$ java Mini/Interpret < ex.apl
le resultat de a = 5+8 est:
13
le resultat de b = 15-a est:
2
le resultat de c = b*a est:
26
le resultat de d = a%c est:
13
le resultat de e = d/3 est:
4
```

Fig.44. le résultat de l'exemple 4.

```
$ java Mini/Interpret < ex.apl
le resultat de a = 5.5+8.75 est:
14.25
le resultat de b = 15.5-a est:
1.25
le resultat de c = b*a est:
17.8125
le resultat de d = a%c est:
14.25
le resultat de e = d/3 est:
4.75
```

Fig.45. le résultat de l'exemple 5.

Exemple 6 :

```
int a, b, c ;

b = 2 ;
c = 19 ;
a = (b + c) * 5 ;
print "le resultat de a = (b+c)*5 est: " ;
print a ;
a = b + c * 5 ;
print "le resultat de a = b+c*5 est: " ;
print a ;
a = 5*b + 5*c ;
print "le resultat de a = 5*b + 5*c est: " ;
print a ;
```

Exemple 7 :

```
double a, b, c ;

b = 2.25 ;
c = 19.0 ;
a = (b - c) / 5.0 ;
print "le resultat de a = (b-c)/5 est: " ;
print a ;
a = b - c / 5.0 ;
print "le resultat de a = b-c/5 est: " ;
print a ;
a = b/5.0 - c/5.0 ;
print "le resultat de a = b/5+ c/5 est: " ;
print a ;
```

```
$ java Mini/Interpret < ex.apl
le resultat de a = (b+c)*5 est:
105
le resultat de a = b+c*5 est:
97
le resultat de a = 5*b + 5*c est:
105
```

Fig.46. le résultat de l'exemple 6.

```
$ java Mini/Interpret < ex.apl
le resultat de a = (b-c)/5 est:
-3.35
le resultat de a = b-c/5 est:
-1.5499999999999998
le resultat de a = b/5+ c/5 est:
-3.3499999999999996
```

Fig.47. le résultat de l'exemple 7.

- **Les opérateurs relationnels**

Exemple 8 :

```
int a ,b, c ;
bool f ;

b = 2 ;
c = 19 ;
f = true ;
print " la valeur de f est: "  ;
print f ;
a = (b + c) * 5 ;
print " le resultat de a = (b+c)*5 est: " ;
print a ;
f = a < b ;
print " le resultat de a<b est:  ";
print f ;
f = a > b ;
print " le resultat de a>b est:  ";
```

```
print f ;
f = a < 105 ;
print " le resultat de a<105 est:  ";
print f ;
f = a <= 105 ;
print " le resultat de a<=105 est:  ";
print f ;
f = a > 105 ;
print " le resultat de a>105 est:  ";
print f ;
f = a >= 105 ;
print " le resultat de a>=105 est:  ";
print f ;
f = a != 66 ;
print " le resultat de a != 66  est:  ";
print f;
f = a == 66 ;
print " le resultat de a == 66  est:  ";
print f ;
```

```
$ java Mini/Interpret < ex.apl
 la valeur de f est:
true
 le resultat de a = (b+c)*5 est:
105
 le resultat de a<b est:
false
 le resultat de a>b est:
true
 le resultat de a<105 est:
false
 le resultat de a<=105 est:
true
 le resultat de a>105 est:
false
 le resultat de a>=105 est:
true
 le resultat de a != 66  est:
true
 le resultat de a == 66  est:
false
```

Fig.48. le résultat de l'exemple 8.

- **Les opérateurs logiques**

Exemple 9 :

```
int a, b, c ;
bool f, g ;

b = 2 ;
c = 19 ;
f = true ;
print " la valeur de f est: "  ;
print f ;
a = (b + c) * 5 ;
print " le resultat de a = (b+c)*5 est: ";
print a ;
g = true ;
f = f && g ;
print " la valeur de f && g est: " ;
print f ;
f = g && (a < 100) ;
print " la valeur de g && (a<100) est: " ;
print f ;
f = g && (a > 100) ;
print " la valeur de g && (a>100) est: " ;
print f ;
f = g || ( a < 100 ) ;
print " la valeur de g || (a<100) est: " ;
print f ;
f = ( b > 3) || ( a < 100 ) ;
print " la valeur de ( b > 3) || ( a < 100 ) est: " ;
print f ;
```

```
$ java Mini/Interpret < ex.apl
 la valeur de f est:
true
 le resultat de a = (b+c)*5 est:
105
 la valeur de f && g est:
true
 la valeur de g && (a<100) est:
false
 la valeur de g && (a>100) est:
true
 la valeur de g || (a<100) est:
true
 la valeur de ( b > 3) || ( a < 100 ) est:
false
```

Fig.49. le résultat de l'exemple 9.

- **Les instructions de contrôle**

  - **Les instructions if et if …else**

Exemple 10:

```
double b, c ;
bool f;
b = 2.25 ;
f = false ;
b= ++b*1.99+5.0 ;
print "la valeur de (++b*1.99+5) est: ";
print "la valeur de b est : " ;
print b;
if ( b == 10.5 )
{ b = ++b ;
}
print "la valeur de b est : " ;
print b ;
if ((b < 15.5) && (b != 10.5 ))
{ f = true ;}
print "la valeur de f est : " ;
print f;
```

```
$ java Mini/Interpret < ex.apl
la valeur de (++b*1.99+5) est:
la valeur de b est :
11.467500000000001
la valeur de b est :
11.467500000000001
la valeur de f est :
true
```

Fig.50. le résultat de l'exemple 10.

Exemple 11:

```
double b, c ;
bool f;
b = 2.25 ;
f = false ;
print " le meme exemple en utilisant if ...else: ";
b= ++b*1.99+5.0 ;
print "la valeur de (++b*1.99+5) est: ";
print "la valeur de b est : " ;
print b;

if ( b == 10.5 )
{ b = ++b ; }
else
{if ((b < 15.5))
{ f = true ;}
}
print "la valeur de b est : " ;
print b ;
print "la valeur de f est : " ;
print f;
```

```
$ java Mini/Interpret < ex.apl
 le meme exemple en utilisant if ...else:
la valeur de (++b*1.99+5) est:
la valeur de b est :
11.467500000000001
la valeur de b est :
11.467500000000001
la valeur de f est :
true
```

Fig.51. le résultat de l'exemple 11.

  - **Les instructions while, do...while et for**

Exemple 12:

```
int n , f ;

n = 4;
f= n;
print " la boucle while ";
while (n >= 2)
{   f = f * (n - 1);
   print n ;
   n = n - 1;
}
print " le factorielle de n=4 est :" ;
print f;
```

Exemple 13:

```
int n , f ;

n = 4;
f= n;
print " la boucle do...while ";
 do {
   f = f * (n - 1);
   print n ;
   n = n - 1;
}
while ( n >= 2) ;
print " le factorielle de n=4 est :" ;
print f;
```

```
$ java Mini/Interpret < ex.apl
 la boucle while
4
3
2
 le factorielle de n=4 est :
24
```

Fig.52. le résultat de l'exemple 12.

```
$ java Mini/Interpret < ex.apl
 la boucle do...while
4
3
2
 le factorielle de n=4 est :
24
```

Fig.53. le résultat de l'exemple 13.

Example 14 :

```
int i,j;

for ( i=1;j=2;  ; i<5 ;  i= i+1; j+=i;)
 {   print "les valeurs de i et j sont: " ;
     print i;
     print j;
  }
```

```
les valeurs de i et j sont:
1
2
les valeurs de i et j sont:
2
4
les valeurs de i et j sont:
3
7
les valeurs de i et j sont:
4
11
```

Fig.54. le résultat de l'exemple 14.

- **L'instruction de branchement inconditionnel : break**

Exemple 15:

```
int i,n;
i = 1;
n = 1;
while(i<15)
{   n = n*2;
    print n;
    i = i++;
    if (n>10) {break;}
}
```

```
$ java Mini/Interpret < ex.apl
2
4
8
16
```

Fig.55. le résultat de l'exemple 15.

- **Les commentaires :** Ils représentent des textes explicatifs destinés aux lecteurs du programme.

Exemple 16:

```
int i;
//  i est initialisé à 7
i = 7 ;
print i;
/* i est initialisé à  13 */
i = 13 ;
print i;
```

```
$ java Mini/Interpret < ex.apl
7
13
```

Fig.56. le résultat de l'exemple 16.

- **Les exceptions**

Exemple 17:

```
int x ;
double y ;
x = 6 ;
```

```
y = x + 3.5 ;
print y ;
```

```
$ java Mini/Interpret < ex.apl
Exception in thread "main" java.lang.RuntimeException: 3.5 is not an integer.
        at Mini.Interpreter$Value.getInt(Interpreter.java:121)
        at Mini.Interpreter$ExpEvaluator.visit(Interpreter.java:1060)
        at Mini.Interpreter$ExpEvaluator.visit(Interpreter.java:798)
        at Mini.Absyn.Eplus.accept(Eplus.java:8)
        at Mini.Interpreter.evalExp(Interpreter.java:791)
        at Mini.Interpreter.access$500(Interpreter.java:14)
        at Mini.Interpreter$StmExecuter.visit(Interpreter.java:436)
        at Mini.Interpreter$StmExecuter.visit(Interpreter.java:404)
        at Mini.Absyn.SAss.accept(SAss.java:10)
        at Mini.Interpreter.execStm(Interpreter.java:397)
        at Mini.Interpreter.access$200(Interpreter.java:14)
        at Mini.Interpreter$ProgramExecuter.visit(Interpreter.java:369)
        at Mini.Interpreter$ProgramExecuter.visit(Interpreter.java:353)
        at Mini.Absyn.Prog.accept(Prog.java:8)
        at Mini.Interpreter.execProgram(Interpreter.java:349)
        at Mini.Interpreter.access$100(Interpreter.java:14)
        at Mini.Interpreter$BehaviourExecuter.visit(Interpreter.java:341)
        at Mini.Interpreter$BehaviourExecuter.visit(Interpreter.java:332)
        at Mini.Absyn.A_Behaviour.accept(A_Behaviour.java:9)
        at Mini.Interpreter.execBehaviour(Interpreter.java:328)
        at Mini.Interpreter.interpret(Interpreter.java:32)
        at Mini.Interpret.main(Interpret.java:21)
```

Fig.57. le résultat de l'exemple 17.

- **Les agents**

Exemple 18:

```
agent  Helloword {
        environments:
        behaviours: behaviour start {
                        print "Hello";
                       }
        internal_memory: mem
       }
```

```
$ java Mini/Interpret < Helloword.apl
Hello
```

Fig.58. le résultat de l'exemple 18.

Exemple 19:

```
agent Fact {
        environments:
        behaviours: behaviour start {
                        int n, f ;

                        n = 4;
                        f = n;
                        do {
                          f = f * (n - 1);
                          n = n - 1;
                        }
                        while ( n >= 2) ;
                        print " le factorielle de n=4 est :" ;
                        print f;
                        }
        internal_memory: mem
       }
```

```
$ java Mini/Interpret < Fact.apl
 le factorielle de n=4 est :
24
```

Fig.59. le résultat de l'exemple 19.

5.6. Conclusion

Devant la difficulté de concevoir un langage de programmation agents capable d'assurer tous les aspects dès le départ, on a proposé une version où on suggère l'utilisation de constructions spéciales aux agents (par exemple : Agent, Behaviour,…) afin de faciliter leurs déploiement.

Dans notre travail on propose une grammaire (APL) pour spécifier notre langage. Celle-ci est définie par la notation LBNF et se présente comme un ensemble de règles étiquetées. Afin de faciliter le développement de notre compilateur on a utilisé le convertisseur BNF.

Le BNFC utilise la notation LBNF pour la description du langage désiré. Il nous a permis la construction d'un compilateur front-end ainsi que le choix du langage pour la génération de ses composants. Il offre un fichier squelette à base de design pattern de visiteur qui peut être utilisé pour la construction du compilateur back-end.

On a proposé une version initiale de notre langage de programmation agents qui sera amélioré par la suite pour avoir une version plus complète assurant tous les aspects mentionnés précédemment.

Actuellement on peut réaliser des agents individuels avec des comportements simples. Afin d'implémenter ces derniers on a développé un ensemble d'instructions et d'expressions à savoir les instructions de contrôle, les opérateurs et les expressions, ….etc. L'aspect de communication et d'interaction n'est pas pris en compte pour le moment ainsi que l'aspect organisation qui est représenté implicitement dans cette version par l'ensemble des agents réalisés. Pour l'environnement on n'a spécifié que deux structures qui peuvent être utilisées par chaque agent (inbox et outbox) afin de spécifier les objets qu'il utilise. Comme ressources, dans cette version, les agents peuvent utiliser les différentes structures de données prédéfinies.

Conclusion

Les scripts de collaboration constituent une manière prometteuse de mettre en œuvre des situations d'apprentissage collaboratives qui encouragent certaines interactions riches entre les apprenants et de fournir une structuration de ces situations.

Dans notre travail on a cherché à trouver un moyen pour faciliter la tâche de conception des scripts en prenant en compte les comportements des apprenants. Notre travail se compose de deux volets. Le premier consiste à élaborer les différents modèles utilisés pour la conception des scripts tandis que le second concerne l'utilisation des systèmes multi-agents.

Notre approche de conception est incrémentale basée sur la boucle: scripting, spécification, exécution et évaluation. Un script une fois écrit et spécifié, sera exécuté sur un environnement d'apprentissage afin de favoriser les interactions visées entre les apprenants. La phase d'évaluation nous permet d'avoir une idée sur le déroulement de la situation d'apprentissage et le comportement des apprenants.

La meilleure façon d'avoir des informations sur les apprenants est de suivre leurs traces pendant leurs interactions (les interactions entre eux ou avec le système). Afin d'avoir une idée claire et précise sur ces interactions, nous proposons que les traces collectés soient groupées en trois classes à savoir : les traces brutes, additionnelles et de production.

Pour la mise en œuvre de notre approche, on suggère l'utilisation des agents artificiels. En effet, durant ces dernières années, les travaux de recherche dans le domaine des systèmes multi-agents ont conduit au développement d'outils et de langages de programmation visant à faciliter le déploiement de ces systèmes et l'implémentation des agents. Le choix d'un langage de programmation est généralement guidé par des contraintes. Dans notre cas ces contraintes sont liées au domaine des agents lui-même, aux domaines de conception de scripts et de suivi des traces des apprenants et à la disponibilité de ces langages.

Toutefois, les langages existants ne satisfont pas à nos besoins. Afin de résoudre ce problème, nous avons proposé un langage en tenant compte en premier lieu de certaines contraintes liées au domaine d'agents. Celui-ci sera amélioré par la prise en compte d'autres contraintes liées aux domaines de conception de scripts et de suivi des traces des apprenants.

Pour la mise au point de notre langage, nous avons conçu une grammaire décrite en notation LBNF, ainsi qu'un compilateur. Bien que celui-ci offre actuellement certaines fonctions minimales, il reste sujet à de futures améliorations afin de le rendre plus performant.

Comme perspectives, nous prévoyons dans une première phase l'amélioration de nos modèles de conception et de suivi ainsi que la finalisation de la mise au point du langage de programmation. La deuxième phase concernera l'implémentation des agents impliqués dans ce travail en utilisant ce langage.

Références

(Arnout, 2004) Arnout Karine Marguerite Alice. (2004) From Patterns to Components. PHD thesis.

(Bellifemine et al., 2007) Bellifemine Fabio., Caire Giovanni., Trucco Tiziana., Rimassa Giovanni., Mungenast Roland. (2007) JADE Administrator's Guide. Last update: 18-June-2007. JADE 3.5

(Berk, 2000) Berk Elliot., (2000) JLex: A lexical analyzer generator for Java(TM) . http://www.cs.princeton.edu/~appel/modern/java/JLex/current/manual.html

(Bordini et al., 2005) Bordini Rafael H. , Braubach Lars , Dastani Mehdi, El Fallah Seghrouchni Amal, Gomez Sanz Jorge J., Leite João, O'Hare Gregory, Pokahr Alexander, Ricci Alessandro (2005)A Survey of Programming Languages and for Multi-Agent Systems. Informatica 30(2006)3344.

(Bringert,2005) Bringert Björn., (2005) BNF Converter Java 1.5 Mode. http://www.cs.chalmers.se/Cs/Research/Languagetechnology/BNFC/doc/BNF_ Converter_Java1_5_Mode.html. Consulté en (2008).

(Boissier et al., 2007a) Boissier Olivier., Hübner Jomi Fred., and Sichman Jaime Simão. (2007) Organization Oriented Programming: From Closed to Open Organizations. G. O'Hare et al. (Eds.): ESAW 2006, LNAI 4457, pp. 86–105, 2007. Springer-Verlag Berlin Heidelberg 2007.

(Boissier et al., 2007b) Boissier, O., Coutinho, L., Sichman, J.S. (2007) Modeling dimensions for multi-agent systems organizations. In: Dignum, V., Dignum, F., Edmonds, B., Matson, E. (eds.) Agent Organizations: Models and Simulations (AOMS), Workshop held at IJCAI 07, 2007.

(Burgos et al., 2005) Burgos Daniel, Arnaud Michel, Neuhauser Patrick, Koper Rob (2005) IMS Learning Design la flexibilité pédagogique au service des besoins de l'e-formation. http://www.epi.asso.fr/revue/articles/a0512c.htm

(Busetta et al., 1999) Busetta P., Ronnquist R., Hodgson A., and Lucas A. (1999) "JACK intelligent agents – components for intelligent agents in Java," AgentLink News Letter, vol. 2, 1999.

(Busetta et al., 2000) Busetta Paolo, Howden Nicholas, Rönnquist Ralph, and Hodgson Andrew (2000) Structuring BDI Agents in Functional Clusters. N.R. Jennings and Y. Lespérance (Eds.): Intelligent Agents VI, LNAI 1757, pp. 277–289, 2000. Springer-Verlag Berlin Heidelberg 2000.

(Castelfranchi, 1995) Castelfranchi C. (1995) Guarantees for Autonomy in Cognitive Agent Architecture.Intelligent Agents: Theories, Architectures, and Languages, 890, 56–70.

(Champin et al., 2004) Champin P.-A., Prié Y., Mille A., (2004) MUSETTE: a Framework for Knowledge Capture from Experience, EGC'04, Clermont Ferrand, Jan, 2004.

(Cleaveland,1988) Cleaveland J. C. (1988) Building application generators. IEEE Software, pages 25-33, July 1988.

(Comp, 2008) http://www.compilers.net/paedia/compiler/index.htm (2008). Consulté en (2008)

(David et al. ,2005) David, J.P., Lejeune, A., Luengo, V., Pernin, J.P., Diagne, F., Adam, J.M., et Choquet. C., (2005) State of art of tracking and analysing usage. Technical report, Kaleidoscope, rapport du JEIRP DPULS, avril 2005.

(David et al., 2007) David, J.P., George, S., Godinet, H., Villiot-Leclerq, E. (2007) Scénariser une situation d'apprentissage collectif instrumentée : réalités, méthodes et modèles, quelques pistes.

(De Giacomo et al., 2000) De Giacomo Giuseppe, Lesperance Yves, and Levesque Hector J. (2000) ConGolog a concurrent pogramming language based on the situation calculus. language and implementation.

(Dillenberg et al., 1996) Dillenberg, P., Baker, M., Blaye, A.. and O'Malley, C. (1996) The evolution of research on collaboratve learning .In E.Spada and P.Reiman (Ads)Learning in humans and machine . Towards an interdisciplinary learning science. (Pp. 189_211). Oxford: Elsevier.

(Dillenberg, 1999) Dillenbourg, Pierre. (1999) What do we mean by collaborative learning?. In Dillenbourg P. (Ed) Collaborative-learning Cognitive and computational Approaches. (pp.1-19). Oxford: Elsevier.

(Dillenberg, 2002) Dillenbourg, P. (2002) Over-scripting CSCL: the risks of blending collaborative learning with instructional design. In P. A. Kirschner(Ed) Three worlds of CSCL. Can we support CSCL (Heerlen , Open UniversiteitNed.61-91. Netherland

(Dillenberg, 2006) Dillenbourg, P. (2006) Orchestrating integrated learning scenarios. Proceedings of the 23rd annual ascilite conference: who's learning?whose technology?

(El Fallah-Seghrouchni & Suna, 2003) El Fallah-Seghrouchni Amal, and Suna Alexandru (2003) CLAIM: A Computational Language for Autonomous, Intelligent and Mobile Agents. M. Dastani, J. Dix, A. El Fallah-Seghrouchni (Eds.): PROMAS 2003, LNAI 3067, pp. 90–110, 2004. Springer Verlag Berlin Heidelberg 2004.

(Ferber, 1995) Ferber, Jacques. (1995) "Les systèmes multi-agents : vers une intelligence collective", 1995.

(Ferber, 1997) Ferber, J. (1997). Les systèmes multi-agents : un aperçu général. Technique et Science Informatiques, 16, 8, 979-1012.

(Forsberg & Ranta, 2003) Forsberg, Markus. and Ranta, Aarne. (2003) Labelled BNF: A High-Level Formalism for Defining Well-Behaved Programming Languages. NWPT'03, Proceedings of the Estonian Academy of Sciences: Physics and Mathematics, Special issue on programming theory edited by J. Vain and T. Uustalu, 2003, volume 52, p. 356–377.

(Forsberg & Ranta, 2004) Forsberg Markus and Ranta Aarne (2004) Tool Demo: BNF Converter. Haskell Workshop 2004. Chalmers University of Technology, Sweden.

(Gamma et al., 1995) Gamma Erich., Helm Richard., Johnson Ralph., and Vlissides John. (1995): Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

(Gouaïch, 2003) Gouaïch Abdelkader. (2003) Requirements for achieving software agents autonomy and defining their responsibility. In: The First International Workshop on Computational autonomy – Potential, Risks, Solutions (autonomy 2003).

(Gouaïch et al., 2005) Gouaïch Abdelkader. , Michel Fabien. , and Guiraud Yves. (2005) MIC*: a Deployment Environment for Autonomous Agents. D. Weyns et al. (Eds.): E4MAS 2004, LNAI 3374, pp. 109–126, 2005. Springer-Verlag Berlin Heidelberg 2005. LNAI Vol. 3374, pages 109-126, 2005.

(Gouaïch, 2005) Gouaïch Abdelkader. (2005) Movement, Interaction, Calculation as Primitives for Everywhere & Anytime Computing. PhD Thesis.

(Henri & Lundgren-Cayrol, 1998) Henri, France. et Lundgren-Cayrol, Karin. (1998) Apprentissage collaboratif et nouvelles technologies.

(Henri et al., 2007) Henri France, Compte Carmen, Charlier Bernadette (2007) la

scénarisation dans toutes ses débats. Revue internationale des technologies en pédagogie universitaire, 4(2) www.profetic.org/revue

(Hernandez-Leo et al., 2006) Hernandez-Leo, D.; Villasclaras-Fernandez, E.D.; Asensio-Perez, J.I.; Dimitriadis, Y.A.; Retalis, S.; (2006) CSCL Scripting Patterns: Hierarchical Relationships and Applicability. Advanced Learning Technologies, 2006. Sixth International Conference on 5-7 July 2006 Page(s):388 - 392 Digital Object Identifier 10.1109/ICALT.2006.1652452

(Hindriks et al., 1999) Hindriks Koen V., De Boer Frank S., Van Der Hoek Wiebe. , and Meyer John-Jules CH. (1999) Agent Programming in 3APL. Autonomous Agents and Multi-Agent Systems, 2,357]401 1999. Kluwer Academic Publishers. Manufactured in The Netherlands.

(Hudson, 1999) Hudson Scott E. (1999) CUP User's Manual. http://www.cs.princeton.edu/~appel/modern/java/CUP/manual.html. Consulté en (2008)

(Iglesias et al., 1999) Iglesias Carlos A., Garijo Mercedes and González José C. (1999) A Survey of Agent-Oriented Methodologies. Muller et al. (Eds.): ATAL'98, LNAI 1555, pp. 317-330, 1999. Springer- Verlag Berlin Heidelberg 1999 .

(Jennings et al., 1998) Jennings Nicholas R., Sycara K. and Wooldridge Michael (1998) A Roadmap of Agent Research and Development. Autonomous Agents and Multi-agent Systems, 1:7-38, 1998.

(Jennings, 2001) Jennings Nicholas R. (2001) An Agent-Based Approach For Building Complex Software Systems. COMMUNICATIONS OF THE ACM April 2001/Vol. 44, No. 4.

(Klein & Holger, 2006) Klein Florian.and Holger Giese. (2006) Grounding Social Interactions in the Environment. D. Weyns, H. Van Dyke Parunak, and F. Michel (Eds.): E4MAS 2005,LNAI 3830, pp. 139–162, 2006._Springer-Verlag Berlin Heidelberg 2006.

(Lemaître & Excelente, 1998) Lemaître, C., Excelente, C.B. (1998) Multi-agent organization approach. In: Garijo, F.J., Lemaître, C. (eds.) Proceedings of II Iberoamerican Workshop on DAI and MAS 1998.

(Log, 2008) http://www.dicodunet.com/annuaire/def-352-fichier-log.htm. Consulté en (2008)

(Malone & Crowston,1994) Malone, Thomas W., & Crowston, Kevin. (1994) The interdisciplinary study of coordination. ACM Computing Surveys (CSUR), 26(1), 87–119.

(Martelli et al., 1995) Martelli Maurizio, Mascardi Viviana, and Sterling Leon (1995) How Agents Do It In Stream Logic Programming.?.

(Mernik et al., 2005) Mernik Marjan, Heering Jan, and Sloane Anthony M. (2005) When and How to Develop Domain-Specific Languages. ACM Computing Surveys, Vol. 37, No. 4, December 2005, pp. 316–344.

(NetLogo, 2008) NetLogo 4.0.2 User Manual. Consulté en (2008)

(Howden et al., 2003) Howden Nick, Rönnquist Ralph, Hodgson Andrew, Lucas Andrew (2003) JACK Intelligent Agents ^{TM} – Summary of an Agent Infrastructure. Agent Oriented Software Pty. Ltd., Melborne, Australia, 2003.

(Parunak, 1998) Parunak H. V. D. (1998) Engineering Artifacts for Multi-Agent Systems. Slides fromMAAMAW99 invited presentation to be found at www.erim.org/~van/Presentations/presentations.htm,1999.

(Paquette et al., 1998) Paquette, G., Crevier, F. et Aubin, C. (1998). Méthode d'Ingénierie d'un Système d'Apprentissage (MISA). Initiation à la formation/conseil en milieu de travail. Sainte- Foy, Québec, Téléuniversité.

(Paquette, 2007) Paquette Gilbert (2007) Scénarisation pédagogique : vers une instrumentation cognitive. Actes du colloque Scénario 2007, le 2e colloque international sur les scénarios pédagogiques **:** Scénariser les activités de l'apprenant : une activité de modélisation. 14 et 15 mai 2007. Montréal.

(Pellauer et al., 2003) Pellauer Michael., Forsberg Markus., and Ranta Aarne. (2003) BNF Converter: Multilingual Front-End Generation from Labelled BNF Grammars.

(Pernin & Lejeune, 2004) Pernin Jean-Philippe., Lejeune Anne.(2004) Le cycle de scénario : modèles pour la réutilisation de scénarios d'apprentissage**.** TICE Méditerranée, Nice, 2004.

(Perret-Clermont et al., 1991) Perret-Clermont A.N., Perret F.-F. and Bell N. (1991) The social construction of meaning and cognitive activity in elementary school children. In L. Resnick, J. Levine & S. Teasley (Eds) Perspectives on Socially Shared Cognition (pp.41-62). Hyattsville, MD: American Psychological Association.

(Pesty et al., 2003) Pesty, S., Webber, C. and Balacheff, N. (2003) Baghera: une architecture multi-agents pour l'apprentissage humain. Cognitique, (P. Aniorte, S. Gouarderes eds), Cepadeus Edition, Toulouse, 2003.

(Plotkin, 1981) Plotkin G. D. (1981). A Structural Approach to Operational Semantics.University of Aarhus_ Denmark

(Ranta, 2005) Ranta Aarne. (2005) BNF Converter Tutorial. http://www.cs.chalmers.se/Cs/Research/Language-technology/BNFC/doc/tutorial/bnfc-tutorial.html

(ReCourse, 2008) ReCourse learning design editor: http://www.tencompetence.org/ldauthor/. (2008).

(Roschelle & Teasley, 1995) Roschelle J. & Teasley S. (1995) The construction of shared knowledge in collaborative problem solving. In C.E. O'Malley (Ed) Computer-supported collaborative learning. Heidelberg: Springer-Verlag.

(Schneider, 2007) Schneider, Daniel K. (2007) ArgueGraph Script. In http://edutechwiki.unige.ch/en/ArgueGraph_Script. Consulté en (2008).

(Settouti et al., 2005) Settouti, Prié and Mille, (2005) Systèmes à base de trace pour l'apprentissage humain.

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

(Sichman et al., 1994) Sichman Jaime Simão., Conte Rosaria., Castelfranchi Cristiano., and Demazeau, Yves. (1994) A Social Reasoning Mechanism Based On Dependence Networks. Pages 188–192 of: Cohn, A. G. (ed), Proceedings of the Eleventh European Conference on Artificial Intelligence. Chichester: John Wiley & Sons.

(Suna, 2005) Suna Alexandru (2005).CLAIM et SyMPA : Un environnement pour la programmation d'agents intelligents et mobiles. PhD Thesis.

(Tagni & Jovanovic, 2006) Tagni Gaston Eduardo. and Jovanovic Dejan. (2006) Comparison of Multi-Agent Systems JACK vs 3APL. Agents course held by Prof. Joao Alexandre Leite. New University of Lisbon. January 12, 2006.On http://www.tagni.com.ar/projects.html(2008).

(Tchounikine, 2002) Tchounikine Piere (2002) Quelques éléments sur la conception et l'ingénierie des EIAH. In: Actes du GDR I3, p. 233-245, Cepadues Editions.

(Tisue & Wilensky, 2004) Tisue Seth.and Wilensky Uri. (2004) NetLogo: Design and Implementation of a Multi Agent Modelling Environment.

(Triple-apl, 2008) http://www.cs.uu.nl/3apl/ (2008).

(Van Deursen & Klint, 1998) Van Deursen A. and Klint E. (1998). Little languages: Little maintenance? Journal of Software Maintenance, 10:75-92, 1998.

(Van Deursen et al., 2000) Van Deursen A., Klint P., and Visser J. (2000) Domain-specific languages: An annotated bibliography. ACM SIGPLAN Notices, 35(6):26–36, June 2000.

(Van Riemsdijk et al., 2005) Van Riemsdijk M. Birna , de Boer Frank S., and Meyer John-Jules Ch. (2005) Dynamic Logic for Plan Revision in Intelligent Agents. J. Leite and P. Torroni (Eds.): CLIMA V, LNAI 3487, pp. 16–32, 2005. Springer-Verlag Berlin Heidelb erg 2005.

(Van Riemsdijk et al., 2006) Van Riemsdijk, M. B., Dastani, M., Meyer, J.-J. Ch., and de Boer, F. S. (2006). Goal-oriented modularity in agent programming. In Proceedings of the fifth international joint conference on autonomous agents and multiagent systems (AAMAS'06), pages 1271-1278, Hakodate.

(Verhagen, 2000) Verhagen Henricus J.E. (2000) Norm Autonomous Agents. PhD Thesis.

(Wang et al., 2005) Wang Mengqiu, Nowostawski Mariusz, and Purvis Martin (2005) Declarative Agent Programming Support for a FIPA-Compliant Agent Platform. R.H.Bordini et al. (Eds.):ProMAS2005, LNAI3862, pp.252–266, 2006. Springer-Verlag Berlin Heidelberg 2006

(Weyns et al., 2005) Weyns Danny. , Parunak H. Van Dyke. , Michel Fabien. , Holvoet Tom. , and Ferber Jacques. (2005) Environments for Multiagent Systems State-of-the-Art and Research Challenges. D. Weyns et al. (Eds.): E4MAS 2004, LNAI 3374, pp. 1–47, 2005. Springer-Verlag Berlin Heidelberg 2005.

(Winikoff, 2005) Winikoff Michael (2005) JACK ™ Intelligent Agents: An Industrial Strength Platform. R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors. Multi-Agent Programming: Languages, Platforms and Applications. Number 15 in Multiagent Systems, Artificial Societies and Simulated Organizations. Springer, 2005. pages 175–193.

(Wooldridge, 1999) Wooldridge Michael (1999) Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Modern Approach to Artificial Intelligence, edited by Weiss Gerhard 1999. MIT Press, Cambridge, MA, USA.

(Wooldridge & Jennings, 1995a) Wooldridge Michael, and Jennings Nicholas R. (1995) Intelligent Agents: Theory and Practice. Knowledge Engineering Review 10 (1995) 115–152.

(Wooldridge & Jennings, 1995b) Wooldridge Michael, and Jennings Nicholas R. (1995) Agent Theories, Architectures, and Languages: A Survey. Intelligent Agents, 1995.

ملخص—التعلم التعاوني ليس دائما فعال.آثاره مرتبطة بثراء وكثافة التفاعلات بين الطلاب اثناء تعاونهم. تصميم سيناريوهات التعلم ليس سهلا و يتطلب معلومات عن المتعلمين وتفاعلاتهم.
لتوجيه التصميم يجب تقييم السيناريو المنجز وأخذ بعين الاعتبار سلوك المتعلمين وتفاعلهم مع بعضهم البعض لتحسينه. في عملنا ، قدمنا نهجا تدريجيا لتصميم سيناريوهات التعلم التعاوني مع مراعاة التفاعل بين المتعلمين. وهذا بمراقبة سلوك الطلاب اثناء التعلم مع احترام السيناريو المحدد،و تقديم معلومات عن اداء هذا السيناريو لتعديل التصميم الأولي.
لهذا نقترح استخدام مجموعة من الوكلاء الذين يقومون بالأدوار التالية : القرار ،التفسير ،التنفيذ ، المراقبة و تتبع الاثر. عمل الوكلاء يبدأ بالتفاعل الفردي للمتعلمين مع النظام. على أساس المعلومات المقدمة للنظام وكيل القراريختار سيناريو (الاختيار ليس على اساس سلوك او تعاون الطلاب). في مرحلة ثانية ، هذا السيناريويترجم من قبل مجموعة من وكلاء التفسير. وأخيرا مجموعة أخرى من الوكلاء تقوم بالتنفيذ مع مراعاة المعلومات الخاصة بالمتعلمين. تنفيذ السيناريو يكون مراقب من قبل مجموعة من الوكلاء(المراقبة). وكلاء تتبع الاثر يقومون برصد آثار الدارسين.
تطويرالوكلاء يتطلب استخدام لغة برمجة خاصة بهم . حيث قمنا بوضع قواعد اللغة باستعمال LBNF وكذلك قمنا بوضع compilateur خاص بها.

Résumé—L'apprentissage collaboratif n'est pas toujours efficace. Ses effets dépendent de la richesse et de l'intensité des interactions entre étudiants pendant la collaboration. La conception des scénarios pédagogiques n'est pas facile et elle nécessite des informations sur les apprenants et sur leurs interactions.
Pour guider une conception, il faut mettre en jeu une évaluation du scénario réalisé, et prendre en compte le comportement et les interactions des apprenants pour améliorer le scénario.
Dans notre travail, nous avons présenté une approche de construction incrémentale de scripts de collaboration avec prise en compte des interactions des apprenants. Il s'agit de suivre le comportement des apprenants en respectant le scénario prescrit par le concepteur et fournir des retours sur l'exécution de ce dernier pour revoir ou affiner la conception préliminaire.
Pour cela, on suggère l'utilisation d'un ensemble d'agents ayant les rôles suivants: décision, interprétation, exécution, observation, et suivi de trace. Le travail des agents commence par l'interaction des différents apprenants avec le système. En se basant sur les profils de ces apprenants, un script sera sélectionné (adéquat aux profils et non pas aux comportements/ collaboration des apprenants) par un agent de décision. Dans une deuxième phase, celui-ci (le script) sera traduit et interprété par un ensemble d'agents. Finalement un autre ensemble d'agents l'exécutera en tenant compte des différents profils de ces apprenants. L'exécution du scénario sera contrôlée par un observateur (un groupe d'agents). Des agents de trace sont responsables du suivi des traces apprenants.
La mise au point de ces agents nécessite l'utilisation d'un langage de programmation orienté agent qu'on a mis au point à savoir une grammaire décrite en notation LBNF, ainsi qu'un compilateur.

Abstract— Collaborative learning is not always efficient. Its effects depend on the richness and intensity of the learners interactions during their collaboration. The design of learning scenarios is not easy and it requires information on learners and on their interactions.
To guide a design, we have to evaluate the scenario, and to take into account the learners behaviours and interactions to improve it.
In our work, we presented an approach to build scripts incrementally taking into account the learners interactions. These include monitoring the behaviour of learners within the scenario specified by the designer and provide feedback on the performance of the latter to revise or refine the preliminary design.
For this, we suggest the use of a set of agents with the following roles: decision, interpretation, execution, observation and tracking learners' traces. The work of agents starts by the learners individual interactions with the system. On the basis of the profiles of these learners, the decision agent selects a script (adequate to the profiles and not to the behaviours/collaboration of learners). In a second phase, it (the script) will be translated and interpreted by a set of agents. Finally another set of agents execute it taking into account the different profiles of these learners. The execution of the scenario will be monitored by an observer. Tracking agents are responsible for collecting the learners' traces.
The development of these agents requires the use of an agent oriented programming language. For this reason, we proposed a grammar described using the LBNF notation and a compiler.