REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'enseignement supérieur et de la recherche scientifique

Université Mentouri de Constantine

Faculté des sciences de l'ingénieur

Département d'Informatique

Numéro d'ordre : 199/ Mag/ 2010

Numéro de série : 004/ inf / 2010

Mémoire en vue de l'obtention du diplôme Magistère en informatique

Option : Système distribué

Présenté par :

Abderrezak Samira

Atelier de génie logiciel à base d'agents

Soutenue le 13 / Juin / 2010

Devant le jury :

Président: Pr. N. Zarour Professeur à l'université Mentouri Constantine

Rapporteur: Dr. R. Maamri Maître de conférence à l'université Mentouri Constantine

Examinateur : Dr. A. Chaoui Maître de conférence à l'université Mentouri Constantine

Examinateur : Dr. S. Merniz Maître de conférence à l'université Mentouri Constantine

Remerciements

Mr Ramdane MAAMRI, Maître de Conférence à l'université de Constantine, a pris la responsabilité de diriger ce travail dans ses moments délicats ; qu'il trouve ici l'expression de ma profonde gratitude et reconnaissance pour son soutien constant, ses encouragements continus, sa patience et sa qualité humaine qu'il m'a manifesté durant la réalisation de ce mémoire.

Le sujet de ce travail a été proposé par le professeur Zaidi SAHNOUN de l'université de Constantine, sa réalisation a été initiée avec lui ; Je lui adresse tous mes remerciements pour ces précieux conseils et ces bonnes orientations.

Je remercie vivement Mr Nacereddine ZAROUR, Professeur à l'université de Constantine, pour l'honneur qu'il m'a fait en acceptant de présider de jury de ce mémoire.

Je remercie vivement Mr Allaoua CHAOUI, Maître de Conférence à l'Université de Constantine, et Mr Salah MERNIZ, Maître de conférence à l'Université de Constantine, de m'avoir fait l'honneur d'accepter de participer au jury de ce mémoire pour le juger.

Je souhaite adresser mes vifs remerciements à mon mari pour son aide, son soutien et sa patience.

Je remercie tous ceux qui m'ont aidé dans la réalisation de ce travail et j'exprime particulièrement toute ma profonde reconnaissance à Naouel Ouafek et Mme Imen Bousbough pour leur soutien et leur aide.

Je tiens à remercier infiniment mes parents pour leurs encouragements et leur soutien aux moments de joie et de détresse.

Je remercie enfin tous les membres de ma famille chacun par son nom.

A Hidaya, Yahya

et Oussama

Résumé

Les ateliers de génie logiciel ont un rôle essentiel dans l'amélioration de la productivité et la qualité des logiciels. Leur objectif majeur est d'automatiser au maximum le processus de développement du logiciel. La réalisation de cet objectif implique une assistance dans les différentes phases du cycle de vie du logiciel.

Une assistance intelligente active au processus de développement du logiciel peut améliorer significativement le rendement des outils en leurs donnant plus de souplesse et de flexibilité.

Ce travail donne une approche basée sur l'utilisation d'un Système Multi Agents pour aider le développeur dans la réalisation de ses logiciels par la modélisation UML, en lui fournissant une assistance au développement du processus logiciel. Le processus utilisé est le Rational Unified Process.

Mots Clés

Atelier de génie logiciel, Assistance intelligente, RUP, Système multi agent.

Abstract

CASE tools have a critical role in improving productivity and software quality. Their major goal is to automate the process of software development to the maximum. Achieving this objective involves assistance in the different phases of the software lifecycle. Active intelligent assistance for the conduct of the software process can significantly improve the performance of the tools in their giving more flexibility and supplest. This work gives an approach based on the use of a system Multi Agents to assist the developer in the achievement of its software with the UML modelling, by providing assistance to the way of working throughout the process. The process used is the rational Unified Process.

Key words

CASE tools, intelligent assistant, multi agent system, RUP

ملخص

ورشات هندسة البرمجيات لها دور حاسم في تحسين إنتاجية العمل الهندسي و جودة البرمجيات . هدفها الرئيسي هو جعل عملية تطوير البرمجيات آلية إلى أبعد حد . تحقيق هدا الهدف يتطلب تقديم المساعدة في مختلف مراحل دورة الحياة الكاملة للبرمجيات و يمكن جعل هده المساعدة أثناء أداء الورشات ذكية و فعالة بإعطائها مرونة أكبر.

هذا البحث يتناول منهجية تعتمد على إستعمال نظام متعدد الوكلاء لمساعدة مطوري البرامج الآلية في مراحل إنجاز برامجهم بإستعمال النمذجة UML.المنهج المستعمل من أجل تحقيق دلك هو RUP.

كلمات مفتاحية

ورشات عمل هندسة البرمجيات, المساعدة الذكية, نظام متعدد الوكلاء, المنهج RUP

Sommaire

INTRODUCTION GENERALE 1

CHAPITRE 1 4

LES ATELIERS DE GENIE LOGICIEL 4

1. INTRODUCTION 4
2. LE GÉNIE LOGICIEL 4
3. LE PROCESSUS LOGICIEL 5
   - ***3.1. Le rôle du processus de développement*** *5*
   - ***3.2. Les différentes activités du processus logiciel*** *6*
   - ***3.3. Modèles de Cycle de vie*** *6*
     - 3.3.1. Modèle en cascade 7
     - 3.3.2. Modèle en V 7
     - 3.3.3. Modèle en spirale 8
     - 3.3.4. Modèle par incrément 8
     - 3.3.5. Modèle itératif et incrémental 9
4. NÉCESSITÉ D'UN ENVIRONNEMENT DE PRODUCTION DE LOGICIEL 10
5. ATELIERS DE GÉNIE LOGICIEL OU OUTILS CASE : 10
   - ***5.1. L'origine de "CASE"*** *10*
   - ***5.2. Définitions*** *10*
   - ***5.3. Objectifs et Avantages des ateliers CASE*** *11*
   - ***5.4. Constituants d'un atelier de génie logiciel*** *12*
     - 5.4.2. Les méthodes 13
     - 5.4.3. Les outils 13
   - ***5.5. Intégration des outils CASE*** *14*
     - 5.5.1. Intégration par l'interface : 15
     - 5.5.2. Intégration par les données : 15
     - 5.5.3. Intégration par le contrôle 16
     - 5.5.4. L'intégration des procédés 16
   - ***5.6. Les différents types d'AGL*** *17*
     - 5.6.1. Outils CASE à base de l'étendue de leur support 18
     - 5.6.2. Outils CASE à base de cycle de vie 19
       - 5.6.2.1. Outils CASE orientés conception (Upper CASE Tools) 19
       - 5.6.2.2. Outils CASE orientés réalisation (Lower CASE Tools) 20
6. COMPARAISON DE QUELQUES AGL EXISTANT 21
   - ***6.1. ArgoUml*** *21*
   - ***6.2. IBM Rational XDE*** *21*
   - ***6.3. Visible Analyst*** *21*
   - ***6.4. Enterprise Architect v5.0" by Sparx Systems*** *22*
   - ***6.5. Rational Rose*** *22*
   - ***6.6. NATURE*** *23*
   - ***6.7. CPCE*** *24*
   - ***6.9. Le Projet ARGO*** *24*
   - ***6.10. Le projet ALF (1987 - 1992)*** *24*
   - ***6.11. WayPointer*** *24*
   - ***6.12. OOExpertProcessModel*** *25*
   - ***6.13. UCDA (Use Case Driven Assistant)*** *25*
   - ***6.14. ISPMS (Intelligent Software Process Management System)*** *25*
   - ***6.15. Analyse*** *25*

7. CONCLUSION 27

CHAPITRE 2 28

L'APPROCHE AGENTS 28

1. INTRODUCTION 28
2. CARACTÉRISTIQUES D'UN AGENT 30
3. ARCHITECTURES D'UN AGENT 31
3.1. Agent réactif *31*
3.1.1. Agent à réflexes simples 32
3.1.2. Agent conservant une trace du monde 32
3.2. Agents délibératifs *33*
3.2.1. Agent ayant des buts 33
3.2.2. Agent utilisant une fonction d'utilité 34
3.2.3. Agent BDI (***Belief, Desire, Intentions)*** 35
3.3. Agent hybride *36*
4. TYPOLOGIE D'AGENTS 37
4.1. Les agents collaboratifs *37*
4.2. Les agents d'interface *38*
4.3. Les agents mobiles *38*
4.4. Les agents d'information *39*
4.5. Les agents logiciels réactifs *39*
4.6. Les agents hybrides *40*
5. SYSTÈME MULTI-AGENTS 40
5.1. Définition *40*
5.2. Utilité des systèmes multi agents *41*
5.3. L'interaction *42*
5.3.1. Composantes des interactions 43
5.3.1.1. Compatibilité des buts 43
5.3.1.2. Disponibilité des ressources 43
5.3.1.3. Capacités des agents par rapport aux tâches 43
5.4. La coopération *44*
5.4.1. Les méthodes de coopération 44
5.4.1.1. Le regroupement et la multiplication 44
5.4.1.2. La spécialisation 44
5.4.1.3. La collaboration par partage de tâches et de ressources 45
5.4.1.4. La coordination d'actions 45
5.4.1.5. La résolution de conflit par arbitrage et négociation 45
5.5. Communication dans les SMA *46*
5.5.1. Communication par partage d'information 46
5.5.2. Communication par envoi de message 47
5.5.3. Les actes de langage 48
5.5.4. Les conversations 48
5.5.5. Langage de Communication Agent (LCA) 49
5.5.6. KQML 49
6. DOMAINES D'APPLICATION DES AGENTS 50
7. CONCLUSION 51

CHAPITRE 3 : 52

CONCEPTION SMA D'UN ATELIER 52

DE GENIE LOGICIEL 52

1. INTRODUCTION 52

2. CHOIX DU RUP 53
3. PRÉSENTATION DU RUP (RATIONAL UNIFIED PROCESS) 53
3.1. Modèle d'un processus RUP *53*
3.1.1. Les rôles 54
3.1.2. Les activités 54
3.1.3. Les artefacts (Work products) 54
3.1.4. Les workflows 55
3.1.5. Les disciplines 55
.3.1.6 Des éléments additionnels du processus 56
3.2. Le cycle de vie du RUP *56*
4. L'ASSISTANCE AU PROCESSUS LOGICIEL 57
5. L'APPROCHE PROPOSEE 58
5.1. L'identification de buts et le diagramme d'hiérarchie de buts *59*
5.2. Les cas d'utilisation *60*
5.3. Le diagramme de séquence *61*
5.4. Les rôles du système *62*
5.5. Le modèle d'ontologie *64*
5.5.1. SPEM 66
5.5.1.1. Le méta modèle SPEM 66
5.5.1.2. SPEM comme profil UML 66
5.5.2. Ontologie de domaine 67
5.5.2.1. Les concepts 67
5.5.2.2. Relation entre concepts et diagramme des relations binaires 70
5.5.3. Ontologie de tâches 71
5.5.3.1. Conceptualisation 71
5.5.3.1.1. L'expression de besoins 71
a. Groupes d'activités (Workflow details) 71
b. Activités 72
5.5.3.1.2. L'analyse et la conception 73
a. Groupes d'activités 73
b. Activités 74
5.5.3.1.3. L'implémentation 75
a. Groupe d'activités 75
b. Activités 75
5.5.3.2. Formalisation 76
5.5.3.3. Représentation des relations entre concepts 77
5.5.3.4. La relation Rôle-Activité-Artefact 79
5.6. Les agents du système *81*
5.7. Les classes d'agents *82*
5.8. Le diagramme de classes *82*
5.9. Structure Interne des agents *83*
5.9.1. L'agent Superviseur 83
a. Le module de connaissance 84
b. Le module de raisonnement 84
c. Le module d'action 84
5.9.2. Les agents_Rôle 86
a. Une base de connaissance 86
b. Un module raisonnement 86
c. Un module Action 87
5.9.3. Les agents Activité 87
a. Un module connaissance 87
b. Un module raisonnement 87
c. Un module Action 88
5.10. Interaction entre les agents Rôle *88*
5.11. L'architecture du système *91*

6. EXEMPLE D'UN OUTIL CASE UTILISANT LA METHODE OMCP 93
6.1. Le comportement de quelques agents du système *96*
6.1.1. L'Agent Analyste Système 96
6.1.2. Le comportement de l'agent Spécificateur des cas d'utilisation 97
6.1.3. Le comportement de l'agent Analyste des cas d'utilisation 98
7. CONCLUSION 99

CHAPITRE 4 100

DESCRIPTION DU FONCTIONNEMENT 100

DE L'OUTIL D'ASSISTANCE A LA MODELISATION 100

1. INTRODUCTION 100
2. L'ONTOLOGIE 100
3. QUELQUES SCÉNARIOS POSSIBLES DE FONCTIONNEMENT 102
a. Scénario de fonctionnement du système *102*
b. Scénario de fonctionnement par demande de réalisation des artefacts *104*
c. Scénario de fonctionnement par demande d'adoption d'un rôle *105*

CONCLUSION GENERALE ET PERSPECTIVES 106

GLOSSAIRE 108

RÉFÉRENCES 109

Liste des figures

Figure 1 : Modèle en V du cycle de vie 7
Figure 2 : Le modèle de la spirale de BOEHM [6] 8
Figure 3 : Le modèle itératif et incrémental 9
Figure 4 : Modèle d'architecture d'AGL (ECMA) [15] 12
Figure 5 : Intégration des outils CASE 17
Figure 6 : Structure de l'AGL Rational Rose [37] 20
Figure 7 : Caractérisation des agents 31
Figure 8 : Schéma d'un agent à réflexes simples 32
Figure 9 : Schéma d'un agent conservant une trace du monde 33
Figure 10 : Schéma d'un agent ayant des buts [65]. 34
Figure 11 : Schéma d'un agent basé sur l'utilité 35
Figure 12 : Diagramme d'une architecture BDI 35
Figure 13 : Architectures d'agents en couches [75] 37
Figure 14 : Communication par partage d'information 47
Figure 15 : Communication par envoi de message 47
Figure 16 : La Boite Noire du RUP 54
Figure 17 : Exemples Rôle, activités, et artefact 55
Figure 18 : Le cycle de vie du RUP 57
Figure 19 : Diagramme d'hiérarchie de buts 60
Figure 20 : Diagramme de cas d'utilisation 61
Figure 21 : Le diagramme de séquence 62
Figure 22 : Le diagramme de rôle de l'agent Superviseur 63
Figure 23 : Le diagramme de rôle de l'agent Rôle 63
Figure 24 : Le diagramme de rôle de l'agent Superviseur 63
Figure 25 : Le diagramme de rôle 64
Figure 26 : Les éléments essentiels du SPEM et leurs relations [102] 66
Figure 27 : Correspondance entre SPEM et les éléments du méta modèle d'UML [103] 67
Figure 28 : Diagramme des relations binaires 70
Figure 29 : Modèle de relation entre les tâches dans RUP 76
Figure 30 : Diagramme d'activités pour les groupes d'activités de l'expression de besoin 77
Figure 31 : Diagramme d'activités pour les groupes d'activités de l'implémentation 78
Figure 32 : La hiérarchie des classes d'agents 82
Figure 33 : Diagramme de classes 83
Figure 34 : Structure interne de l'agent Superviseur 86
Figure 35 : Structure interne de l'agent Activité 88
Figure 36 : Interaction entre les agents rôle 88
Figure 37 : Architecture proposée 91
Figure 38 : Liste des artefacts [55] 93
Figure 39 : L'interaction entre les agents Rôle 94
Figure 40 : Structure interne de l'agent responsable de la méthode TrouverActeur dont l'agent Analyste Système est responsable 96
Figure 41 : Le comportement de l'agent TrouverActeur 97
Figure 42 : Le comportement de l'agent Spécificateur pour un cas d'utilisation 97
Figure 43 : Le comportement de l'agent Analyste de cas d'utilisation 98
Figure 44 : Une partie du code OWL de l'ontologie de domaine de RUP 101
Figure 45 : Une partie du code OWL de l'ontologie de tâche de RUP 102
Figure 46 : Automates à état finis pour le scénario de communication b 105

Liste des tables

Tableau 1 : Tâches de développement dans la classification de Forte et McCully **14**
Tableau 2 : Classes des outils CASE [28] **19**
Tableau 3 : Comparaison de quelques outils CASE **23**
Tableau 4 : Relation Rôle-Activité-Artefacts **81**
Tableau 5 : Croyances, Objectifs et Plans de l'agent Superviseur **85**
Tableau 6 : Croyances, Objectifs et Plans de chaque agent de type Rôle **87**
Tableau 7 : Les agents rôle et activité de l'Analyse et la conception **91**
Tableau 8 : La collaboration entre les agents rôle **95**

Introduction générale

Alors que le matériel informatique a fait des progrès très rapides, le logiciel, l'autre ingrédient de l'informatique, traverse toujours une crise qui dure depuis la fin des années 60 du siècle passé. Cette crise peut se percevoir à travers des symptômes tels que : le coût de développement d'un logiciel qui est généralement très élevé avec un délai de livraison rarement respecté ; la qualité du produit livré ne satisfait pas souvent les besoins de l'utilisateur ; la maintenance du logiciel est difficile, coûteuse et souvent à l'origine de nouvelles erreurs …etc. La raison de fond de la crise du logiciel réside dans le fait qu'il est beaucoup plus difficile de créer des logiciels que le suggère notre intuition. Comme les solutions informatiques sont essentiellement constituées de composants immatériels on sous-estime facilement leur complexité. Déjà la taille des programmes montre que cette complexité est souvent bien réelle: un million de lignes pour un logiciel de commande et de navigation d'un avion moderne, le décuple pour une station orbitale.

Pour maîtriser la complexité des systèmes logiciels, il convient de procéder selon une démarche bien définie, de se baser sur des principes et méthodes, et d'utiliser des outils performants. Pour réaliser un produit, il faut suivre un procédé qui décrit la suite des actions et opérations à entreprendre pour le développement et la maintenance du logiciel. Les actions et opérations utilisent des méthodes, techniques et outils. Pour évaluer un procédé, ou un produit, on effectue des mesures. Un procédé répond donc aux questions: qui doit faire le travail? Que faut-il faire? Comment faut-il faire le travail? Quand faut-il le faire? Et à quel niveau de détail faut-il faire le travail? Il s'agit d'un processus variable (selon le type d'application) et complexe, composé de différentes phases interdépendantes.

Afin de tenter de résoudre la crise du logiciel, ce processus a fait l'objet de différentes modélisations. Historiquement, le premier modèle de développement proposé est celui dit ``de la cascade'', au début des années 70. Ce modèle a été assez largement mis en œuvre, mais on s'est rapidement aperçu qu'il n'est pas toujours approprié. Sa vision simpliste du processus sous-estime le coût des retours en arrière dans le cycle de vie. Ainsi, plusieurs alternatives au modèle de la cascade ont été proposées, basées notamment sur le prototypage et l'assemblage de composants réutilisables.

A partir des années 90 à nos jours, les ateliers de génie logiciel (AGL) sont apparus comme un moyen plus efficace pour apporter une solution réelle à certains problèmes du génie logiciel et contribuent nettement à l'amélioration de la productivité et de la qualité du logiciel, notamment en faisant le suivi des différentes phases du processus logiciel et en offrant un cadre cohérent et uniforme de production.

Un AGL ou atelier CASE (Computer Aided Software Engineering) est un logiciel permettant lui-même de produire des logiciels de manière industrielle assistée par ordinateur. Il intègre des outils adaptés aux différentes phases de la production d'un

logiciel et se base sur des méthodologies pour formaliser le processus logiciel et chacune des phases internes qui le composent. L'application des méthodologies de développement est donc nécessaire pour garantir la qualité des logiciels produits.

De nombreuses méthodologies ont été développées durant les deux dernières décennies, mais, les méthodologies les plus complètes présentent elles mêmes une complexité certaine due au nombre élevé d'artefacts exigés aux règles de la méthodologie et à leurs processus généralement compliqués à mettre en œuvre. Il ne fait plus de doute que le développeur a toujours besoin de plus d'assistance face à la complexité grandissante des problèmes à résoudre ou à celle des méthodologies de développement utilisées dans les différents projets.

Les ateliers CASE existant proposent une assistance à la façon de modéliser mais peu d'entre eux s'intéressent au déroulement des activités du processus. Ce déroulement est essentiellement guidé par une longue série de documents fournis par les auteurs de la méthodologie. Les développeurs ont donc besoin d'une assistance intelligente au processus logiciel qui consiste à lier les outils avec les connaissances pour leurs permettre de participer activement à la réalisation du logiciel. Les outils doivent se comporter comme des participants actifs au développement du logiciel, plus ou moins autonomes et peuvent prendre de l'initiative.

Pour répondre à cette problématique, nous allons introduire le concept d'agent où un AGL sera considérée comme un ensemble d'agents servant à guider le développeur. Ces agents ayant divers rôles, différentes connaissances et compétences. Ils ne travaillent pas de manière isolée ni séquentielle, et ont besoin de communiquer, de coordonner leurs actions et, de manière plus large, de coopérer pour mener à bien le développement de logiciels. Dans ce travail, nous proposons un système multi-agents qui assiste l'utilisateur pendant le développement de son logiciel en utilisant la méthodologie RUP connue par ses « best practices ». L'outil sera considéré comme un système multi agents, où les agents collaborent ensemble par communication pour atteindre l'objectif commun, qui est la réalisation du logiciel. Chaque agent a donc son objectif local et l'objectif global. La collaboration se fait selon le processus logiciel itératif RUP. Les agents utilisent la connaissance sur toutes les entités du logiciel (outils à invoquer, méthodes à appliquer,…etc) pour assister activement le développeur.

Pour atteindre notre objectif, nous avons besoin de décrire une base d'ontologies qui contient deux ontologies, de domaines et de tâches. Une ontologie de domaine, représente la description du domaine, c'est à dire les concepts clés de RUP utilisés par les agents du système. Et une ontologie de tâches qui décrit quoi réaliser et comment le réaliser. Elle décrit les tâches du processus générique RUP, et leur enchaînement logique suivant le processus RUP.

Les agents vont simuler le travail d'un AGL, ils disposent chacun des règles et des connaissances qui vont aider l'agent à assister intelligemment le développeur. C'est-à-dire, guider le développeur, lui donner des recommandations, et même prendre des initiatives. Ainsi, les agents interagissent entre eux en collaborant par communication suivant le processus afin de réaliser la modélisation de logiciel.

Plan de Travail

Ce mémoire est composé en quatre chapitres, les deux premiers chapitres fixent le contexte de notre travail et les deux derniers constituent la proposition et quelques scénarios de fonctionnement de l'approche proposée. Nous les détaillerons comme suit :

Chapitre 1 : Présente quelques concepts de base liés au génie logiciel, aux modèles de processus logiciel et aux ateliers de génie logiciel, en citant l'état de l'art de quelques approches de développement des ateliers de génie logiciel et l'assistance au processus logiciel. Ensuite, une analyse de l'état de l'art est faite, et elle a permis d'introduire la problématique et de motiver la contribution.

Chapitre 2 : Est consacré à l'étude des systèmes multi agents, en mettant l'accent sur le concept d'agent et ses mécanismes essentiels. Les cinq problématiques des agents : L'interaction, la coopération, la communication, la négociation et la coordination sont discutés, ainsi que quelques domaines d'application.

Chapitre 3 : Dans ce chapitre, nous avons proposé une solution multi-agents à l'assistance à la modélisation. Nous avons commencé par introduire une présentation brève de RUP. Nous donnons ensuite une analyse du problème à résoudre en suivant la méthodologie MASE qui servira à définir une conception de la solution. La conception d'une ontologie de domaine était nécessaire pour définir les concepts de RUP et les relations entre eux et une ontologie de tâches pour définir les tâches et les liens entre eux et ainsi définir un vocabulaire commun entre les agents. L'analyse est terminée par donner une architecture finale du système.

Chapitre 4 : Afin d'éclaircir l'idée du travail réalisé, quelques scénarios de réalisation sont décrites ainsi que le code OWL des deux ontologies de domaine et de tâche ont été donnés.

Finalement, nous terminons le mémoire par un bilan du travail effectué et nous donnons quelques perspectives et les extensions possibles du travail proposé.

Chapitre 1

Les ateliers de génie logiciel

1. Introduction

Au cours des trois dernières décennies, les outils d'ingénierie logicielle assistée par ordinateur ont émergé comme l'une des innovations les plus importantes dans le développement de logiciels pour gérer la complexité des projets de développement logiciel. L'efficacité de ces outils, généralement appelés outils « CASE », apparaît nettement dans l'augmentation de la vitesse avec laquelle le logiciel est développé, l'amélioration de la qualité du système développé [1] et la réduction des coûts de développement de logiciel. En plus, les outils CASE offrent aux développeurs du logiciel / système une plateforme uniforme pour présenter l'information et la connaissance d'une manière compacte pour faciliter la communication [2] [3] [4]. Les outils CASE sont donc apparus pour soulager les développeurs des activités d'ingénierie logicielle trop ordinaires en leur laissant plus de temps pour se concentrer sur les tâches non triviale, exigeant plus de perspicacité et de créativité.

En effet, les outils CASE aident activement à produire et maintenir des logiciels complexes. Ils appliquent des méthodologies de développement de logiciels pour garantir un minimum de qualité des logiciels produits. Pour répondre à ces recommandations, de nombreuses méthodologies ont été développées depuis les années 80 du siècle passé. Cependant, les méthodologies les plus complètes présentent elles mêmes une complexité certaine due au nombre élevé d'artefacts exigés, aux règles de la méthodologie et à leurs processus plutôt compliqués à mettre en œuvre. Il ne reste plus de doute que le développeur a besoin aujourd'hui de plus d'assistance qu'avant pour faire face à la complexité grandissante des problèmes à résoudre et des méthodologies de développement utilisées dans les différents projets.

Pour cette raison, nous commençons par introduire dans ce premier chapitre un état de l'art global concernant les ateliers CASE appelés aussi ateliers de génie logiciel (AGL) terminé par une analyse comparative entre quelques AGL existant dans la littérature, dans le but de donner une motivation à la problématique proposée.

2. Le génie logiciel

Le terme ***génie logiciel*** a été introduit à la fin des années 60. Depuis, cette discipline n'a pas cessé d'évoluer. Son objectif est de développer des outils et des méthodes de production assistée par ordinateur afin de garantir la qualité et la productivité des logiciels.

Le génie logiciel, définit par Patrick Jaulent [5], est l'ensemble des activités de conception et de mise en œuvre des produits et des procédures tendant à rationaliser la production du logiciel et son suivi.

La discipline du génie logiciel (GL) représente un ensemble de méthodes, techniques et outils pour la production et la maintenance des systèmes d'information. En d'autres termes, le génie logiciel est l'art de spécifier, concevoir, réaliser et faire évoluer avec des moyens et dans des délais raisonnables; des programmes, des documentations et des procédures de qualité pour résoudre certains problèmes à l'aide d'un système informatique.

Le génie logiciel ne concerne pas seulement la réalisation de produits, mais surtout la façon la plus efficiente de les produire. Pour qu'un logiciel soit bon; il doit être, en plus de ses fonctionnalités requises : maintenable, fiable, efficace et doit offrir une interface utilisateur appropriée [6].

Et si le génie logiciel est l'art de produire de bons logiciels, il est nécessaire de fixer les critères de qualité d'un logiciel. On peut séparer ces qualités en deux catégories qui sont rapportées à son utilisation et à sa maintenance [7].

- Les qualités du logiciel pendant son utilisation qui sont : La fiabilité (correction et robustesse), l'adéquation aux besoins, l'ergonomie (simplicité, rapidité d'emploi, personnalisation), l'efficacité, la convivialité, le faible coût et respect des délais…etc.

- les qualités du logiciel lors de sa maintenance: un logiciel doit pouvoir être maintenu pour le corriger, l'améliorer, l'adapter aux changements de son environnement, ...etc. Pour cela, il doit être flexible (utilisation du paramétrage, de la généricité, de l'héritage), portable (éviter l'assembleur et les langages trop confidentiels), structuré (utilisation de modules ou de classes, de procédures ou de fonctions) avec une indépendance maximum entre les structures (utilisation de l'abstraction) et doit être, bien sûr, documenté.

Ces différentes qualités ne sont pas toujours compatibles ni même réalisables et il est nécessaire de trouver des compromis. Dans tous les cas, les objectifs de qualité doivent être définis pour chaque logiciel et la qualité du logiciel doit être contrôlée par rapport à ces objectifs.

3. Le processus logiciel

3.1. Le rôle du processus de développement

Un processus de développement logiciel définit le qui, le quoi, le quand et le comment du développement de logiciels. L'importance des processus logiciels est directement liée à l'observation, communément acceptée, que la qualité du produit est un résultat de la qualité du processus [8]. Ainsi, le processus logiciel est intéressant dans le développement d'un système informatique à plus d'un titre [9] à savoir :

- Chaque membre de l'équipe de développement comprend quel est son rôle dans le développement du produit.
- Les développeurs comprennent mieux leur participation dans le développement et leur contribution par rapport aux autres développeurs.
- Les dirigeants et responsables comprennent mieux l'avancement du travail de chacun grâce aux différents modèles produits.
- Le processus permet de mesurer lors de la phase de lancement les risques liés au projet et de mesurer à chaque itération les écarts entre ce qui est prévu et la réalité.

3.2. Les différentes activités du processus logiciel

La notion de cycle de vie de logiciel modélise l'enchaînement de différentes activités du processus technique de développement du logiciel [10].

Les étapes suivantes permettent de décrire, en général, le cycle de développement du logiciel :

- ***Analyse des besoins (Expression des besoins du produit).***
- ***Spécification globale (Conception préliminaire).***
- ***Conception architecturale et détaillée.***
- ***Programmation (Implémentation ou phase de codage).***
- ***Gestion de Configuration et Intégration.***
- ***Validation et Vérification.***
- ***Maintenance et Assistanc***e.

Et le problème qui se pose ici est : comment rendre la réalisation de ces étapes plus facile? Comment assurer la cohérence entre les différentes activités ? La réponse à ces questions appartient au domaine des ateliers de génie logiciel.

3.3. Modèles de Cycle de vie

Tous les modèles de cycle de vie ne sont pas les mêmes, mais l'idée est toujours la même: le logiciel est développé en phases discrètes, chacune ayant un résultat défini et un critère de terminaison défini, et chaque phase est achevée avant que ne commence la suivante. Le modèle classique de cycle de vie comprend les phases suivantes: analyse, conception, implémentation, test, et éventuellement phase d'installation. D'autres noms sont parfois donnés à ces phases et un découpage plus fin est souvent utile.

Chaque phase a des entrées et des sorties, qui sont généralement des documents et parfois des produits. Toute sortie d'une phase servira d'entrée à une phase ultérieure, souvent celle qui suit immédiatement, mais pas toujours. Certaines activités déployées pendant une phase lui sont donc spécifiques et d'autres sont destinées à préparer les phases qui suivent.

3.3.1. Modèle en cascade

Dans ce modèle le principe est très simple. Chaque phase du modèle ne se termine que par la production de certains documents ou logiciels. Les résultats d'une phase sont obtenus sur la base des interactions entre ses étapes, ils sont soumis à une revue approfondie et on ne passe à la phase suivante que lorsqu'ils sont jugés satisfaisants. Le modèle original ne comportait pas la possibilité de retour en arrière. Celle-ci a été rajoutée ultérieurement sur la base qu'une étape ne remet en cause que l'étape précédente, cependant, il s'est avéré que ce retour reste insuffisant dans la pratique.

3.3.2. Modèle en V

Dérivé du modèle de la cascade, le modèle en V du cycle de développement (fig. 1) montre non seulement l'enchaînement des phases successives, mais aussi les relations logiques entre phases plus éloignées. Ce modèle décrit un enchaînement purement séquentiel des différentes activités, en mettant toutefois en évidence la correspondance entre activités amont (analyse, conception) et aval (intégration, validation).

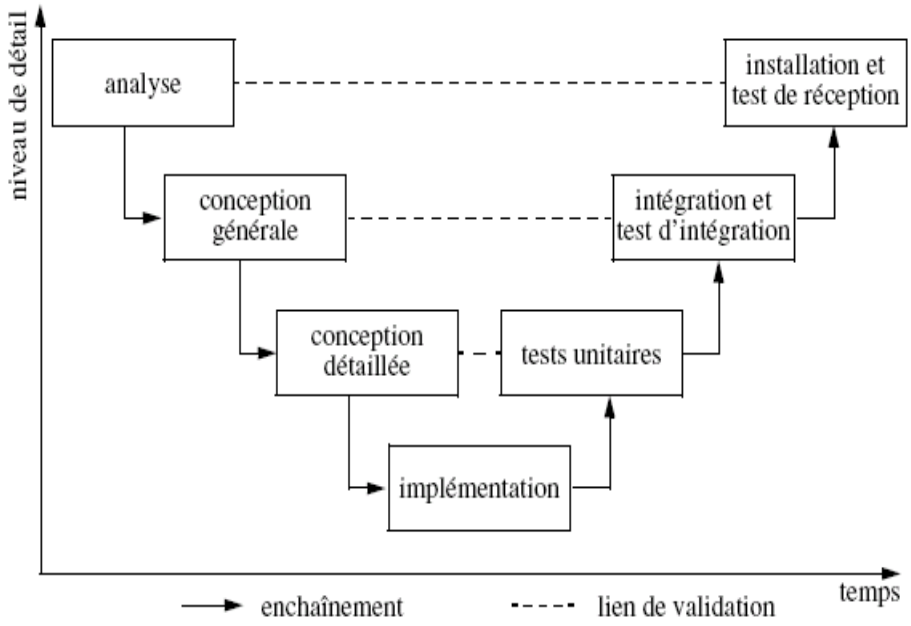

Figure 1 : Modèle en V du cycle de vie

L'application stricte du modèle en phases prescrit qu'on complète entièrement une phase avant de passer à la suivante. Dans la pratique, il arrive cependant qu'au cours d'une phase on découvre des erreurs commises dans une phase antérieure ou même l'absence d'éléments essentiels qui auraient dû être fournis par une phase antérieure. Il peut donc être nécessaire de revenir sur une phase précédente. Si tel est le cas, il faut parcourir à nouveau toutes les phases à partir de la phase révisée pour répercuter partout les modifications.

3.3.3. Modèle en spirale

Proposé par B. Boehm en 1988 [6], ce modèle est beaucoup plus général que les précédents. Il met l'accent sur l'activité d'analyse des risques : chaque cycle de la spirale se déroule en quatre phases :

- Détermination, à partir des résultats des cycles précédents ou de l'analyse préliminaire des besoins, des objectifs du cycle avec les alternatives et les contraintes possibles pour atteindre ces objectifs.
- Analyse des risques, évaluation des alternatives avec proposition d'un maquettage éventuel.
- Développement et vérification de la solution retenue en utilisant un modèle classique (en Cascade ou en V).
- Revue des résultats et vérification du cycle suivant.

L'analyse préliminaire est affinée au cours des premiers cycles. Le modèle utilise des maquettes exploratoires pour guider la phase de conception du cycle suivant. Le dernier cycle se termine par un processus de développement classique.

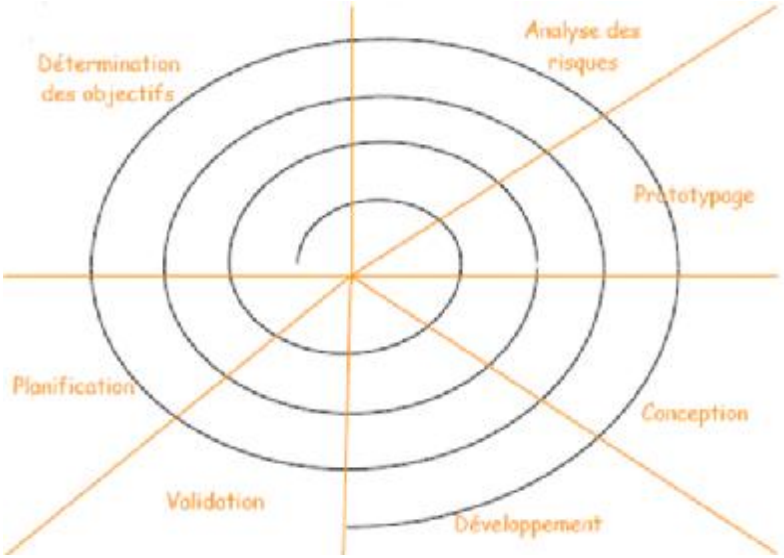

Figure 2 : Le modèle de la spirale de BOEHM [6]

3.3.4. Modèle par incrément

Dans les modèles précédents un logiciel est décomposé en composants développés séparément et intégrés à la fin du processus.

Dans les modèles par incrément, un seul ensemble de composants est développé à la fois : des incréments viennent s'intégrer à un noyau de logiciel développé au préalable. Chaque incrément est développé selon l'un des modèles précédents.

Les avantages de ce type de modèle sont les suivants :

- Chaque développement est moins complexe.
- les intégrations sont progressives.
- il est ainsi possible de livrer et de mettre en service chaque incrément.
- il permet un meilleur lissage du temps et de l'effort de développement grâce à la possibilité de recouvrement (parallélisation) des différentes phases.

Les risques de ce type de modèle sont les suivants :

- Remettre en cause les incréments précédents ou dans le cas pire, le noyau.
- Ne pas pouvoir intégrer de nouveaux incréments.

Les noyaux, les incréments ainsi que leurs interactions doivent donc être spécifiés globalement, au début du projet. Les incréments doivent être indépendants fonctionnellement et sur le plan du calendrier du développement, le plus que possible.

3.3.5. Modèle itératif et incrémental

Dans un cycle de vie itératif et incrémental, le système est développé en une série d'itérations jusqu'au système final.

Chaque itération englobe une ou plusieurs phases de processus : modélisation métier, besoins, analyse, conception, implémentation, tests et déploiement. Au début du cycle, les développeurs se gardent de croire que tous les besoins sont connus mais on s'attend à des changements à toutes les phases.

Ce type de cycle a pour objet d'atténuer les risques. Les risques techniques sont évalués et classés par priorité très tôt et sont revus à chaque itération de développement. Ils sont définis de sorte que le développement réussi d'une itération élimine les risques qui lui étaient attachés (voir figure 3). Les versions sont planifiées pour assurer le traitement prioritaire des risques les plus importants. Ceux-ci sont ainsi identifiés et traités précocement, ce qui se traduit par un coût moindre [11].

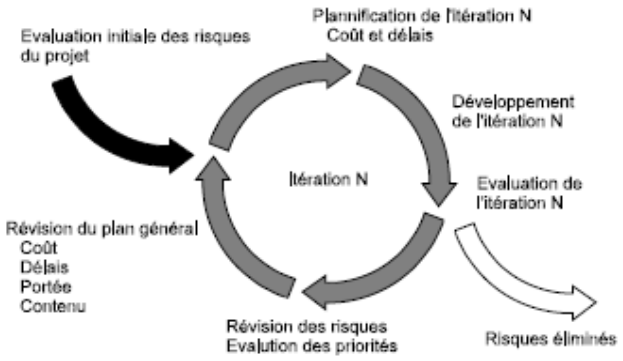

Figure 3 : Le modèle itératif et incrémental

4. Nécessité d'un environnement de production de logiciel

L'objectif de cet environnement est de définir un cadre de production dans lequel le logiciel ne serait plus conçu de façon artisanale mais bien de manière industrielle. La fabrication du logiciel doit être de plus en plus automatisée ainsi que de nombreux outils d'aide à la réalisation d'une tâche particulière du cycle de vie du logiciel ou spécifique à un domaine existant, tels que les outils d'aide à l'analyse, à la conception …etc. En plus, d'autres outils non liés spécifiquement à une phase de projet, comme les outils de gestion de projet ou d'aide à la documentation et à la validation, doivent être aussi automatisés.

5. Ateliers de Génie Logiciel ou Outils CASE :

5.1. L'origine de "CASE"

Le terme CASE « Computer Aided Software Engineering » a été d'abord appliqué à des outils qui fournissent un support pour les phases d'analyse et de conception du cycle de développement logiciel et la plupart des premiers outils étaient destinés à automatiser les méthodes structurées qui étaient disponibles.

La vision actuelle de CASE est celui d'un ensemble d'outils interdépendants qui prennent en charge tous les aspects du processus de développement logiciel. Elle inclue des outils qui prennent en charge des phases spécifiques du cycle de vie telles que les outils d'analyse et de conception, outils de génération de code, outils de test et les outils qui fournissent des fonctionnalités durant le cycle de vie tels que les outils de gestion de projet, de gestion de configuration et de documentation. [12]

5.2. Définitions

Le terme Atelier de Génie Logiciel (AGL) ou atelier CASE (Computer Aided Software Engineering) a été introduit au cours des années 80, pour désigner un logiciel aidant à la réalisation de logiciels. Il s'agit donc d'un système utilisé pour le développement logiciel assisté par ordinateur. Un AGL intègre des outils adaptés aux différentes phases de la production d'un logiciel et facilite la communication et la coordination entre ces différentes phases. Il est basé sur des méthodologies qui formalisent le processus logiciel et chacune des phases internes qui le composent. Donc, Un rôle important d'un AGL dans le développement de logiciels est de servir comme un compagnon de méthodologie en fournissant un support à la méthodologie.

Une autre définition donnée par R. J. Noran [13] et qui fait ressortir l'aspect modélisation est : "Un AGL est un ensemble intégré d'outils qui permet aux développeurs de logiciel de documenter et modéliser un système d'information dès la spécification initiale des besoins jusqu'au projet et son implantation; en passant par l'application de tests de cohérence, complétude et conformité aux spécifications proposées".

Il faut noter ici que les AGLs sont seulement des aides et ne permettent pas de donner une solution totale à touts les problèmes de développement du logiciel. Ils sont des outils de gestion pour le développement de logiciel [14]. Un apport essentiel de l'AGL est de permettre de documenter automatiquement un logiciel et maintenir en permanence cette documentation à jour tout au long de sa conception.

5.3. Objectifs et Avantages des ateliers CASE

Le développement des logiciels complexes fiables et efficaces demande l'effort et la collaboration de beaucoup de gens spécialistes et peut prendre des années pour l'accomplir. Des petites erreurs dans la logique des programmes peuvent avoir des conséquences énormes pour le développeur. Et ainsi les outils CASE ont été développés dans le but d'aider les développeurs dans la production de systèmes logiciels et de produits de haute qualité.

L'objectif principal des AGL est l'automatisation maximale de tout le processus ou d'une partie du processus de développement du logiciel. Cet objectif, limité par la réalité du terrain, demande d'impliquer une assistance aux différentes phases du cycle de vie du logiciel.

Les outils CASE constituent un moyen indispensable pour résoudre les problèmes du développement d'application et de l'entretien des logiciels. Ils changent de manière significative le temps pris par chaque phase et la distribution du coût dans le cycle de vie de logiciel. Les concepteurs du logiciel s'intéressent plus sur l'analyse et la conception. Une grande partie du code peut être générée automatiquement avec le développement des spécifications détaillées. Ces améliorations rendues possibles par l'utilisation des outils CASE montrent des réductions importantes et très bénéfiques en coûts de développement et de maintenance.

La puissance des outils CASE se situe dans leur répertoire central qui contient des descriptions de tous les composants centraux du système. Ces descriptions sont employées à toutes les étapes du cycle : la création des concepts d'entrée-sortie, la génération automatique de code …etc. Les tâches suivantes continuent à s'ajouter pour construire ce répertoire de sorte que dans la conclusion du projet il contient une description complète du système entier. C'est un dispositif puissant qui n'était pas faisable avant l'introduction des outils CASE.

En résumé, les outils CASE servent pour :

1 - Améliorer la qualité du processus en:

_ Augmentant la productivité des équipes.

_ Favorisant la standardisation de la production.

_ Accroissant la prédictibilité des développements.

_ Améliorant la visibilité des projets.

_ Augmentant le confort et la créativité des développeurs.

2 - Augmenter la conformité des produits en :

_ Aidant à appliquer les plans et les normes d'assurance qualité,

_ Permettant de mener à bien des projets complexes et importants en volume et en taille d'équipe,

_ Améliorant le travail coopératif,

_ Obtenant et mesurant un niveau de qualité défini.

Fondamentalement, les outils CASE:

- aident le développeur à créer les principaux modèles d'un système d'information.
- vérifient que les modèles sont complets et compatibles avec d'autres modèles.
- permettent de générer le code à partir des modèles.

5.4. Constituants d'un atelier de génie logiciel

Un atelier efficace doit être constitué d'un ensemble cohérent de méthodes, une base de données (***Repository***) qui gère l'ensemble des objets crée au cours de chacune des étapes de production, un ensemble d'outils nécessaire au développement du logiciel et un ensemble intégré de mécanismes d'interface entre chacune des phases du cycle de vie. Cependant, le doublet (méthodes, outils) est considéré comme l'ensemble des ressources nécessaires à tout AGL. La figure 4 présente le modèle standard d'un AGL proposé par ECMA.

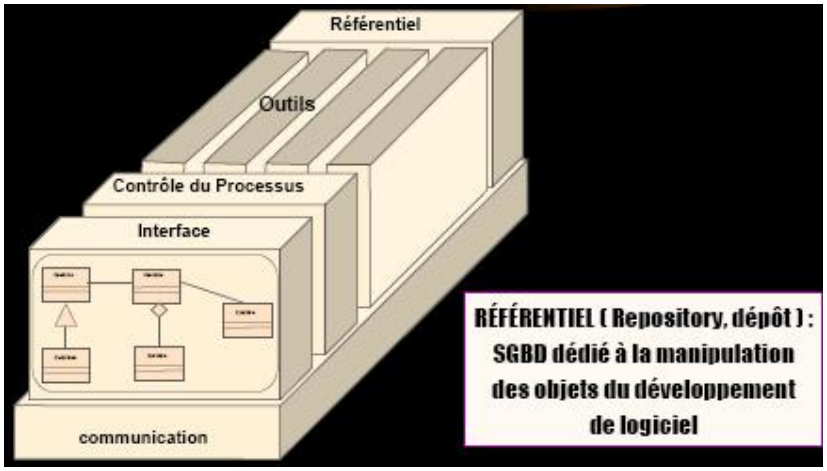

Figure 4 : Modèle d'architecture d'AGL (ECMA) [15]

5.4.1. Le référentiel

Le cœur d'un système CASE bien conçu est un référentiel (***Repository,dépôt***) [16], qui est utilisé comme une base de connaissances pour stocker des informations sur l'organisation, sa structure, les modèles d'entreprise, les fonctions, les procédures, les modèles de données …etc. Le sens représenté par leurs fenêtres de détail et de diagrammes est stocké dans le référentiel. Le référentiel accumule constamment les informations relatives à la planification, l'analyse, la conception, la construction et la maintenance des systèmes.

Il stocke des informations sur le système, dont des modèles des descriptions et de références servant à lier les modèles ensembles. L'outil CASE permet de vérifier les modèles pour assurer qu'ils sont complets et suivent les bonnes règles de conceptualisation. Il peut aussi confronter un modèle à un autre pour attester leur cohérence. Si l'on pense à tout le temps qu'un analyste consacre à créer, vérifier et réviser des modèles, puis à certifier qu'ils s'emboîtent tous, l'utilité des outils CASE est indéniable. Si l'information est stockée dans un référentiel, l'équipe de développement peut l'utiliser de diverses manières. Chaque fois qu'un membre de l'équipe ajoute de l'information, elle devient disponible pour tous.

5.4.2. Les méthodes

Un AGL est basé sur des méthodologies pour formaliser le processus logiciel et chacune des phases qui le composent. Il peut supporter des méthodes structurées, des méthodologies orientées objet ou les deux. La recherche actuelle sur l'évaluation des outils CASE se concentre plus sur les méthodes orientées objet.
On distingue quatre types de méthodes:

- Méthodes de planification stratégique du développement des systèmes applicatifs ;
- Méthodes de développement ; c'est à dire de spécification, conception, réalisation, installation et maintenance des systèmes logiciels ;
- Méthodes de conduite de projet ;
- Méthodes d'assurance de la qualité.

5.4.3. Les outils

Un AGL intègre des outils adaptés aux différentes phases de la production d'un logiciel, et facilite la communication et la coordination entre ces différentes phases. Ce sont des différents outils d'aide au développement de logiciels.

Forte, G. et McCully, K. (1991) [17] proposent une taxonomie d'outils CASE à deux niveaux. Le premier classifie les outils CASE par domaines comme les domaines d'application visés par l'outil, les tâches soutenues dans la méthode et les notations de cycle de vie de développement, la plateforme de matériel, les logiciels d'exploitation, les bases de données et les sous-ensembles de traitement transactionnel soutenus par l'outil …etc. Le deuxième niveau spécifie les attributs de chaque domaine en deux classes (Outils horizontaux et Outils verticaux) suivant les tâches de développement de la figure.

Outils horizontaux : interviennent durant la totalité du processus logiciel (Service pour l'ensemble du cycle de vie), exemples : Éditeurs de texte, Gestion de projet, Gestion du dictionnaire de données, Administration et droits d'accès, Gestion des configurations, Documentation, Service de communication.

Outils verticaux : Ces différents outils interviennent lors d'une ou plusieurs phases du cycle de vie du logiciel : Spécification, Conception, Génération de code, IDE,

Compilateurs, Génération d'interfaces homme-machine, Génération de tests, Validation, Prototypage, Maintenance.

Le tableau 2 suivant présente la classification de Forte et McCully.

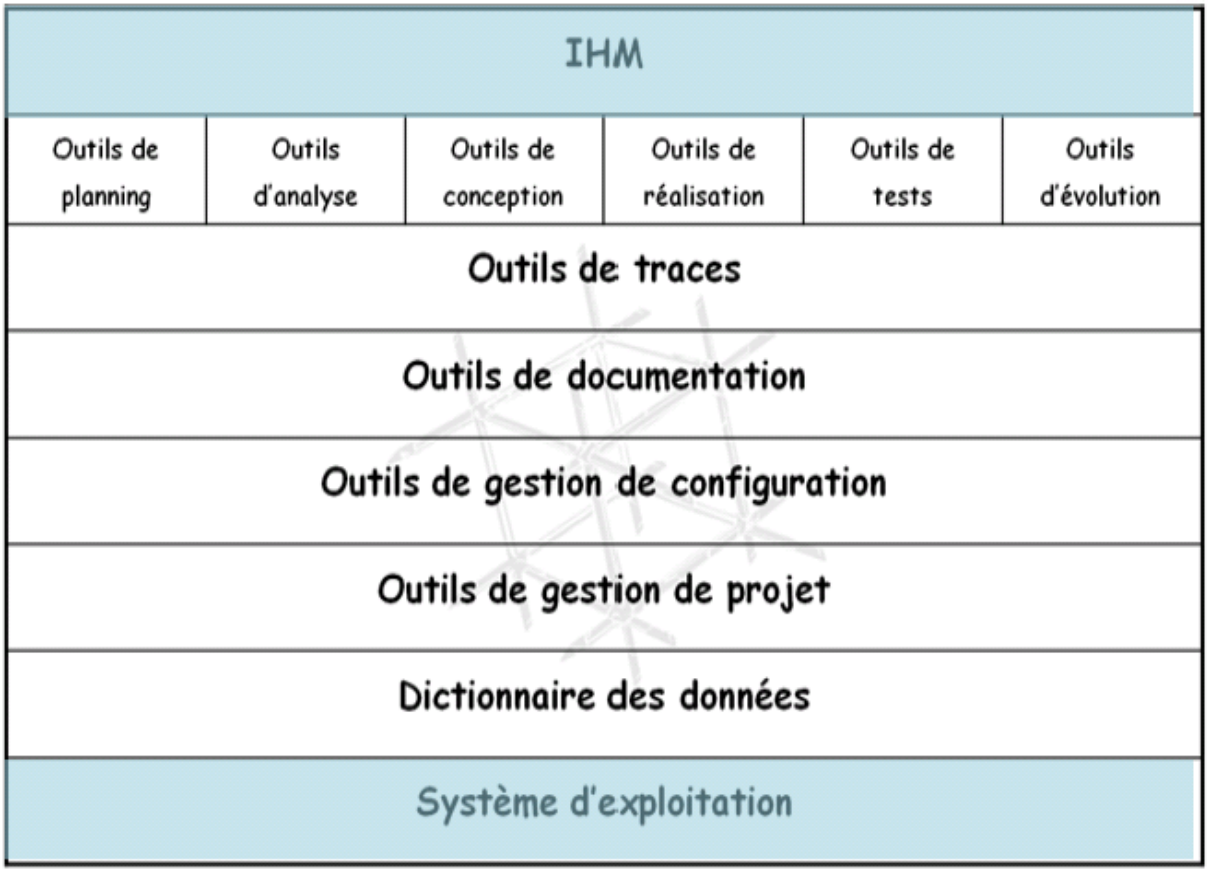

Tableau 1 : Tâches de développement dans la classification de Forte et McCully

5.5. Intégration des outils CASE

L'objectif de l'intégration (Figure 5) est de faire une collection d'outils qui fonctionne comme un seul outil avec les caractéristiques : Une interface utilisateur commune assurant une transition sans heurt entre les outils ; transfert sémantique de données de sorte que les données produites par un outil sont utilisables par un autre outil ; navigation simple entre les outils avec une exécution d'outil contrôlée de façon uniforme et les fonctions, qui suivent un processus prédéfini, sont de sorte que la fonction d'un outil suit ou précède directement la fonction d'un autre outil. [18]

Le modèle de référence de l'ECMA (European Computer Manufacturers Association) [15] pour les environnements assistés (CASE, Computer Assisted Environments) qui se veut un cadre conceptuel et fonctionnel pour décrire et comparer les systèmes, identifie huit dimensions pour l'intégration : la base d'objets, l'intégration par les données, les outils, la gestion des tâches, les messages ou communication entre les outils, l'interface

utilisateur, la sécurité, et l'administration et la configuration de la plateforme support de l'environnement.

Chaque dimension est analysée selon une perspective conceptuelle, un point de vue externe (vision de l'utilisateur) et une perspective interne (point de vue du système). Pour chaque dimension, le modèle définit des règles, des opérations, la variété de types de données requises, ainsi que les relations entre les dimensions.

En effet, il existe plusieurs modèles d'intégration, parmi elle le modèle conceptuel d'AGL proposé par Wasserman [19] qui propose les quatre dimensions essentielles suivantes :

5.5.1. Intégration par l'interface :

C'est la manière d'utiliser un AGL. L'objectif majeur de l'intégration par l'interface est d'améliorer les performances et l'efficacité d'utilisation d'un AGL par ses utilisateurs.
L'intégration par l'interface est supportée par des systèmes de gestion de fenêtre.
On dit qu'un AGL est bien intégré par rapport à cet axe lorsqu'un utilisateur qui a déjà appris à utiliser un outil d'AGL dépense un minimum d'énergie pour apprendre à utiliser un autre outil. Les outils qui composent l'AGL doivent donc utiliser le même paradigme d'interaction.

5.5.2. Intégration par les données :

Concerne le contrôle de l'utilisation des données par les outils qui composent l'AGL. Les données d'un projet dans l'ensemble sont réparties sur les outils étant adoptés. L'intégration d'outils essai de concevoir un environnement de développement intégré qui offre l'accès uniforme aux données des différents outils et les maintient conformés.
Toute l'information manipulée par l'environnement doit être gérée comme un tout cohérent. Les mécanismes qui rendent possible l'intégration des données prennent en compte la représentation, la conversion et le stockage des données.
Par rapport à cet axe d'intégration, Thomas et Nedjmeh [20] ont identifié les propriétés d'intégration suivantes:

Inter-operabilité : Le but est de minimiser les conversions de données entre deux outils qui coopèrent.
Non-redondance : Le but est de stocker le minimum d'objets qui peuvent être dérivés de manière automatique ou de limiter le plus possible la duplication des données.
Cohérence : Pour expliquer ce critère, supposons que nous avons deux outils A et B manipulant respectivement les objets OA et OB liés sémantiquement. Un AGL satisfait ce critère, lorsque l'outil A informe des actions effectuées sur OA et des effets provoqués afin que l'outil B puisse appliquer correctement des contraintes sur OB.

Échange de données : On cherche à diminuer le travail nécessaire à la transformation des données échangées entre les outils, que celles ci soient persistantes ou non.
Synchronisation par les données : Lorsque les outils sont capables de s'échanger des informations à propos des modifications de valeurs effectuées sur des objets partagés.
Dans le domaine de l'intégration par les données, PCTE+ [15] est considéré comme un standard d'intégration des données au niveau européen adopté par ECMA.

5.5.3. Intégration par le contrôle

Afin de supporter des combinaisons harmonieuses de fonctions, les outils composant un AGL doivent partager et échanger des services.

L'outil fournissant les services ne doit pas être directement concerné par le fait que d'autres outils utilisent ses services. Afin de rendre possible le partage de services entre les outils, un AGL doit être intégré par le contrôle, donc fournir des mécanismes pour permettre la communication entre les outils. En général, l'intégration par le contrôle est supportée par des gestionnaires de messages, qui fournissent essentiellement trois types de communication : outil - outil, outil - service et service - service.

Il y'a beaucoup des AGL qui sont basé sur l'intégration de contrôle, le premier parmi eux est le système SoftBench [21] qui est une version commerciale et révisée de l'AGL nommé FIELD [22] considéré comme un des premiers AGL basé sur l'intégration par le contrôle. FIELD est bâti sur un serveur de communication, qui permet le partage d'informations par des échanges de messages. Dans le but d'améliorer le processus de contrôle, FOREST [23] a perfectionné le serveur de messages de FIELD en y introduisant un mécanisme basé sur des règles de sélection. L'utilisation de telles règles rend possible la description du décodage des messages échangés entre les outils.

Le mécanisme de déclenchement (***"trigger"***) a été également utilisé pour contrôler le partage et l'échange des services des outils dans un AGL, comme par exemple, les AGL : Adèle 2 [24] et Alf [25]. D'autres systèmes, comme par exemple Oikos [26], utilisent la technique dite des tableaux noirs (***"blackboard"***) dérivé du domaine de l'intelligence artificielle pour contrôler l'échange de messages entre les outils.

Une autre stratégie qui a été utilisé est celle de wrapping, elle a été utilisée par Valetto et Kaiser [27] pour l'incorporation des outils CASE dans un environnement centré processus, nommé Oz.

5.5.4. L'intégration des procédés

C'est la possibilité de modéliser, à priori, la façon dont devrait se dérouler un développement et de contrôler l'exécution du modèle. Cette dimension d'intégration mesure le degré auquel les outils participent dans le processus de logiciel global et commun.

Par ailleurs, il est à noter que ces dimensions ne sont pas isolées l'une de l'autre. L'intégration par le contrôle peut nécessiter des services de la dimension donnée dans la mesure où, par exemple, la structure d'un message reçu par un outil ne se conforme pas à celle admise par l'outil. D'autre part, les mécanismes associés à l'intégration des procédés font recours à la fois à la dimension contrôle (par exemple, pour activer simplement un outil) et à la dimension données.

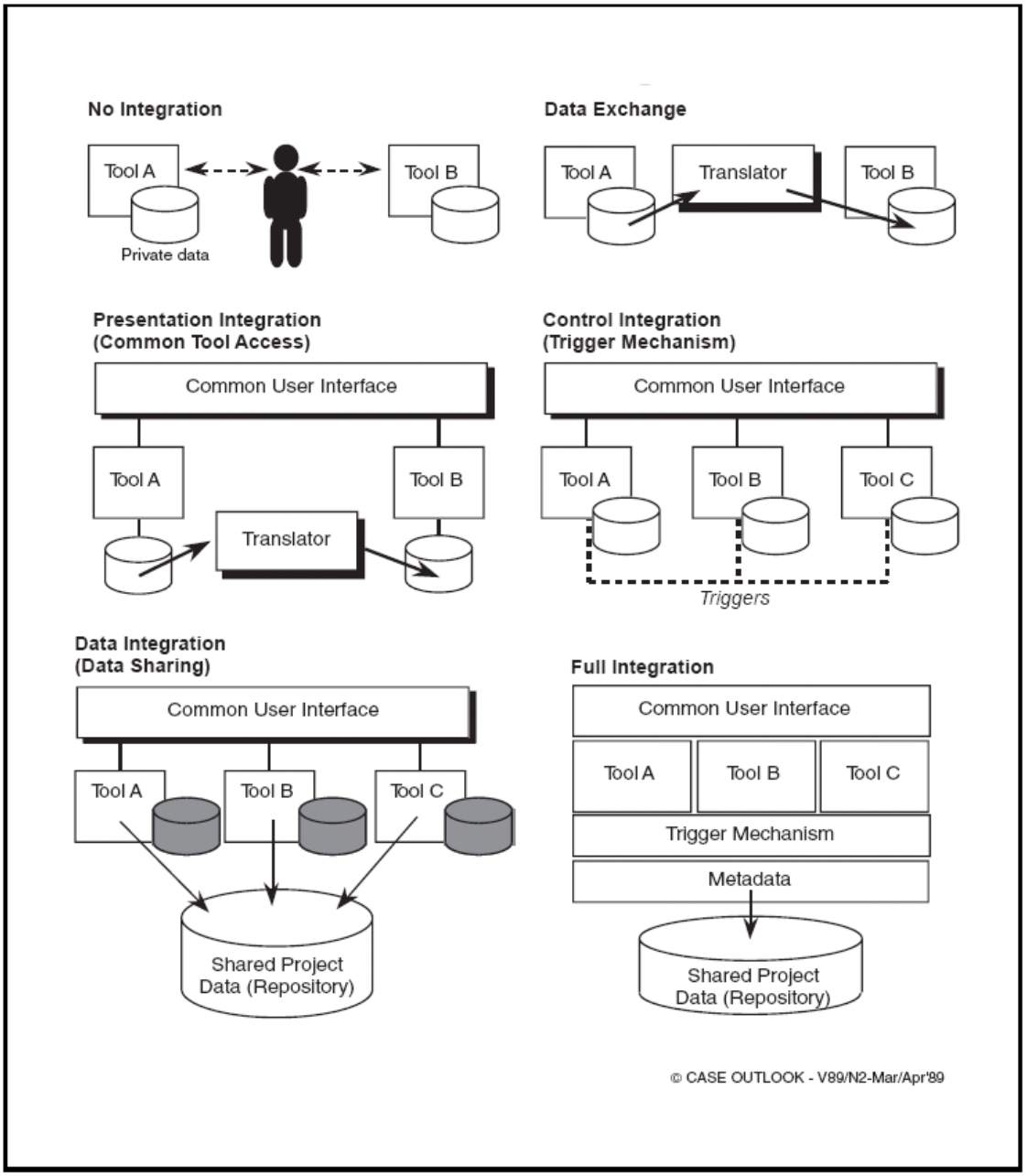

Figure 5 : Intégration des outils CASE

5.6. Les différents types d'AGL

Les AGL peuvent être classés selon plusieurs aspects tels que :

- La richesse du support : ensemble d'outils, outils intégrés, aide à la démarche.
- Le type de problèmes : logiciels embarqués, temps réel, "business applications", applications métiers …
- L'ampleur du projet: complexité, nombre de participants, durée...

- Gestion des ressources du projet : les considérations managériales des ressources mises en œuvre dans le projet sont elles prises en compte? (Planification, ordonnancement, …).
- Phase du cycle de développement prises en compte: conception et/ou développement.

Nous citerons ici les classifications les plus connues.

5.6.1. Outils CASE à base de l'étendue de leur support

Alfonso Fuggetta [28] classifie les produits CASE (Table 2) suivant l'étendue de leur support. Ils partitionnent les outils CASE en trois catégories : Outils, Ateliers (Workbenches***)*** et Environnements.

La première catégorie supporte des tâches spécifiques dans le processus de développement du logiciel.

La deuxième catégorie «***Workbenches*** » supporte une ou plusieurs activités par application. Cette catégorie est partitionnée en huit classes de W***orkbenches*** qui sont :

1. La planification et la modélisation du métier

2. L'analyse et la conception

3. Le développement de l'interface utilisateur

4. La programmation

5. La vérification et la validation

6. La maintenance et l'ingénierie inversée (reverse engineering)

7. La gestion de configuration

8. La gestion de projet

Des détails concernant certaines classes citées sont donnés dans la table 1 ci-dessous.

Class	Subclass
Editing	Graphical editors, Textural editors
Programming	Coding and debugging, Code generators, Code restructurers
Verification & Validation	Static/Dynamic analyzers, Comparators, Symbolic executors, Emulators/Simulators, Correctness proof assistants, Test case generators, Test management tools
Configuration management	Configuration- and version management tools, Configuration builders, Change-control monitors, Librarians
Metrics & measurement	Code analyzers, Execution monitors/timing analyzers
Project management	Cost-estimation tools, Project-planning tools, Conference desk, E-mail, Bulletin boards, Project agenda, Project notebooks
Miscellaneous tools	Hypertext systems, Spreadsheets

Tableau 2 : Classes des outils CASE [28]

La troisième catégorie, ***Environments***, supporte tout ou au moins une partie substantielle du processus du logiciel avec une collection d'outils et des Workbenches. Cette catégorie est divisée en cinq classes : les Toolkits, les environnements centrés-Langage, les environnements intégrés, les environnements de quatrième génération et les environnements centrés-Processus.
Les outils peuvent être donc des produits autonomes ou des composants des ateliers et des environnements.

5.6.2. Outils CASE à base de cycle de vie

Cette dimension permet de classer les outils CASE sur la base des activités qu'elles prennent en charge dans le cycle de vie des systèmes d'informations. Ils peuvent être classés comme outils CASE Upper ou Lower.

5.6.2.1. Outils CASE orientés conception (Upper CASE Tools)

Ces ateliers visent surtout l'assistance des phases initiales du processus de développement de logiciel. Ils intègrent généralement : Des outils pour l'édition de diagrammes (avec vérification syntaxique), des dictionnaires de données, des outils pour l'édition de rapports, des générateurs de (squelettes de) code, des outils pour le prototypage,... Ils supportent les langages de diagrammes traditionnels comme Diagrammes ER, Diagrammes de flots de données, Cartes structurées, Arbres de décision, Tables de décision, …etc. Ils sont fortement basé sur des paradigmes (Orienté Objet), des méthodes de conception et les formalismes associés (ex : RUP/UML, Merise/E-R, ...), et proposent des outils d'éditions graphiques de ces formalismes. Ils proposent aussi une assistance pour la génération de documentation. Ils peuvent proposer un outil de prototypage (génération automatique partielle de code) et éventuellement de ***reverse engineering*** (création de représentations graphique dans un formalisme donné à partir de code source existant). Voici quelques AGL Orientés Conception :

- Win'Design [29] de CECIMA

- PowerDesigner [30] de Sybase [31], basés sur Merise et UML (spécialisé dans le développement de système d'information).
- Oracle Designer [32] d'Oracle Corporation.
- Rational Suite AnalystStudio [33], Rational Rose [34] (Présenté dans la figure 6), basés sur UML et supporte la méthodologie Rational Rose Unified Process [35].
- Objecteering [36] de SoftTeam, basé sur UML.

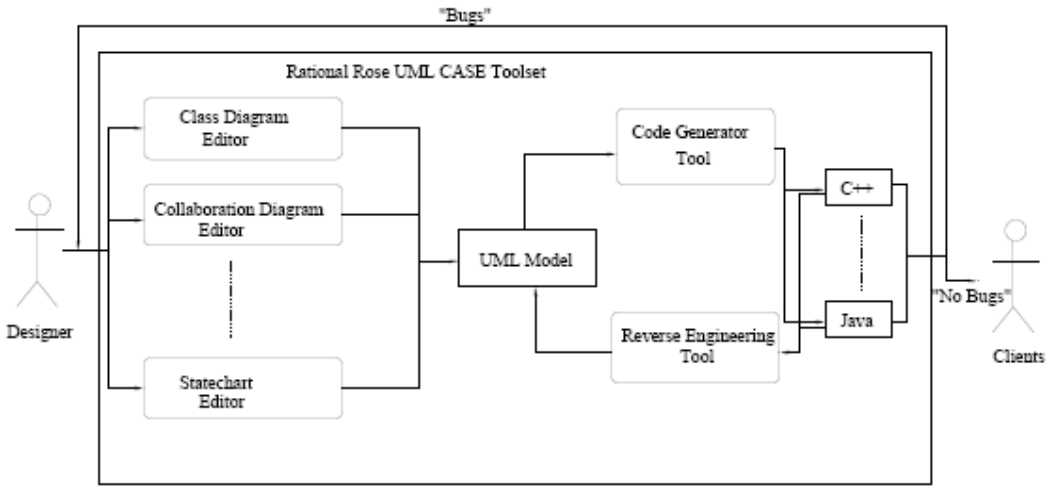

Figure 6 : Structure de l'AGL Rational Rose [37]

5.6.2.2. Outils CASE orientés réalisation (Lower CASE Tools)

Ceux sont les outils qui se concentrent plus sur les dernières activités de cycle de vie du logiciel et donc prendre en charge des activités telles que la conception physique, débogage, construction, tests, intégration de composants logiciels, maintenance, les activités de réingénierie et de reverse engineering. On peut les classifier selon le niveau d'assistance en:

- Outils de développement : éditeur, compilateur, debugger, profiler, gestion de version, multi-utilisateurs.
- Environnements de Développement Intégré : idem mais regroupés au sein d'une seule interface et intégrés entre eux. ***Ex : Turbo C++.***
- Environnement de Développement Rapide : idem avec facilité d'automatisation de certaines tâches de programmation (e.g. interfaces graphiques). ***Ex : Visual x, JBuilder, Sun One, ...***
- Atelier de Génie Logiciel : idem avec support étendu aux autres phases du cycle de développement du logiciel (spécification, conception, déploiement …). ***Ex: WinDev***

Parmi les AGL Orientés Réalisation on peut citer :

- Windev [38] de PCSoft qui est un atelier de génie logiciel édité par la société française PC SOFT et conçu pour développer rapidement des applications, principalement orientées données. C'est un ***outil RAD*** plus avancé car il permet à partir d'une analyse Merise ou UML de produire un applicatif final et opérationnel. WinDev peut générer des applications en Java le long avec des applications standard ou des applications pour

Plateforme de .NET. Il supporte le paradigme procédural aussi bien que le paradigme de programmation orientée objet.

- PowerBuilder [39] de Sybase (PowerSoft) utilise une approche orienté objet et conçu pour les applications orientées données.
- Oracle Developer [40] de Oracle Corporation conçu pour les applications orientées données.
- SafeBuild [41] de TNI-Valiosys, basé sur UML destiné au développement d'applications ***temps-réel***.
- Rational Suite Development Studio [42] de Rational Software, basé sur UML et aux applications OO.

6. Comparaison de quelques AGL existant

6.1. ArgoUml

ArgoUML [43] est un projet Open Source fondé par Jason Robbins à l'Université de Californie en Irvinie. ArgoUML offre un guide dans le workflow d'un processus qui n'a pas encore été défini ou plutôt qui manque de standard. Il s'apparente également à un outil CASE UML avec une surcouche assistance essentiellement axée sur les ***critiques dans la manière de modéliser***. Il offre les critiques de conception (Design Critics), inclue étroitement l'assistance intelligente intégrée qui apporte sur les erreurs de conception, incomplètes fournies de la conception, ou de l'incompatibilité de l'interface. En outre, ils sont ciblés pour suggérer d'autres conceptions ; par exemple en raison de problèmes dans la génération du code à partir de la conception ; ou offrir des conseils heuristiques ; par exemple un collègue qui a signalé un problème. Il vise à long terme une assistance complète au développeur pendant le processus de développement.

Poseidon for UML reprend et étend les concepts de ArgoUML, c'est la version commerciale d'ArgoUML.

6.2. IBM Rational XDE

IBM Rational XDE [44] est une famille de produits Rational Rose, qui fournit l'essentiel de la modélisation UML dont la plupart de ses produits offre une assistance au processus RUP. Rose XDE Modeler et Rose Data Modeler par exemple ne supportent pas une configuration RUP. Rose XDE Modeler offre des capacités spéciales pour assister la modélisation. En outre, Rational PurifyPlus, Rational Rose XDE Developer Plus, offrent un ensemble d'outils d'analyse d'exécution qui détectent les erreurs de mémoire, mettre en évidence les performances des applications goulot et identifient le code non testé. Ces fonctionnalités permettent de trouver des problèmes dans le code et de les corriger avant leur surface dans l'environnement du client.

6.3. Visible Analyst

Visible Analyst [45] est un AGL développé par Visible. Il supporte deux méthodes, Structurée et orientée objet. Il intègre les stratégies de la planification, modélisation des

données, BPMN (Business Process Modeling), Modélisation UML et la modélisation d'analyse et conception structurée.

6.4. Enterprise Architect v5.0" by Sparx Systems

Enterprise Architect v5.0 [46] est un AGL de modélisation UML, supporte toutes les phases de spécification des besoins jusqu'au Déploiement. Ingénierie de code dans plus de 10 langues. Référentiel évolutif, basée sur l'équipe. L'esprit de cadres de l'entreprise, cartes, BPMN et plus...

6.5. Rational Rose

Rational Rose [33] est un AGL UML de IBM, il assiste l'utilisateur dans la modélisation UML. Ibm Rational Rose et ses produits liés offrent une assistance à la modélisation à l'aide des Explorateurs de projets, des palettes d'outils, des barres d'action contextuelles, de l'édition directe dans les diagrammes, des vues de propriété, d'héritage, d'ensemble (***outline***), …etc ; et de l'aide étendue : antisèches, exemples, didacticiels, …etc. Il offre aussi une assistance au processus à l'aide des patrons de la structure initiale des modèles ; ensemble de patrons pour RUP [47]; exemples de catégories : Analyse et conception, Modélisation d'activité... Les patrons sont partageables entre les membres de l'équipe.
La vérification de la consistance (Consistency checking) est ***contrôlée par l'utilisateur***. Rational Rose® inclut un module complémentaire document automatisé, SoDA, qui permet la génération automatique de rapports et documents de cycle de vie du projet. Son support de travail d'équipe est limité à l'intégration du référentiel (repository) distribué avec gestion de configuration et granularité de verrouillage flexible.
Le tableau suivant (Tableau 3) va résumer les caractéristiques des AGL décrits ainsi.

	AGL					
Critères	Rational Rose	Enterprise Architect v5.0	Visible Analyst	Rose XDE Developer for Java	Rose XDE Modeler	ArgoUml
méthodologie suivie	Orienté Objet	Orienté Objet	Structuré et orientée objet	Orientée Objet	Orientée Objet	Orientée Objet
Processus D'assistance	RUP	Inconnu		RUP	Inconnu	Inconnu
Modèles de Cycle de vie	Itératif et incrémental			Itératif et incrémental	Itératif et incrémental	Itératif et incrémental
Supporte le travail en	+	+	+	+	-	-

équipe						
Support de diagramme Uml	+	Diagrammes UML2.x	+	+	-	-
Fournir une assistance à la modélisation	+	+	+	+	+	Les critiques de la conception
Transformation XMI	+	+	+	+	-	-
Vérification de la consistance	Contrôlée par l'utilisateur	Contrôlée par l'utilisateur	+	+	Contrôlée par l'utilisateur	Critiques
Couverture des étapes du cycle de vie	+	+	+	Business Modeling, Architecture logiciel& conception, Codage / Construction et Modélisation des données	Sauf Le business Modeling, l'architecture de logiciel et la conception	-
Configuration de processus	+	-	-	+	-	-

Tableau 3 : Comparaison de quelques outils CASE

On peut mentionner encore d'autres travaux qui ont abordé le sujet des outils CASE et l'assistance intelligente :

6.6. NATURE

Les premiers travaux sur l'assistance aux méthodes de modélisation ont débuté avec le projet ***NATURE*** PR [48] et son environnement ***MENTOR*** SRG [49] qui démontre la faisabilité de la modélisation orientée décision pour le support aux méthodes. NATURE utilise une approche d'utilisateur unique en faisant abstraction de l'espace de travail des rôles et de tous les aspects organisationnels liés au développement.

6.7. CPCE

L'approche ***CPCE*** [50] du laboratoire LORIA de l'***Université Nancy*** avec Jacques Lonchamp. Lonchamp a visé de construire un noyau d'environnement paramétré par des modèles de procédés qui fournit assistance et guidage actif à ses utilisateurs. Grâce à une approche hybride orientée tâche au niveau global et orientée décision au niveau local le système peut s'adapter à une large gamme d'applications de conception collective. Le premier prototype de CPCE permet de piloter essentiellement des inspections collectives de code.

Le projet CPCE propose une assistance et guidage stratégique au niveau du flot d'activités et une assistance et guidage tactique concernant les décisions à prendre pendant une activité spécifique. Il propose également un approche intégrant quatre vues : la vue orientée tâche (***workflow view***), la vue orientée décision (***problem solving view***), la vue oriente produit (***artifact view***) et la vue organisationnelle. Ce premier prototype va faire évoluer le projet vers un nouvel autre qui le fait passer sous le giron du CNRS (***Centre National de Recherches Scientifique français***), le projet Argo qui généralise CPCE pour l'assistance à la méthode et à la coopération.

6.9. Le Projet ARGO

L'objectif général du projet ***ARGO*** [51] est de construire un noyau d'environnement centré processus fournissant une assistance et une guidance active à ses utilisateurs. Il utilise une approche hybride de modélisation orientée tâche et orientée décision, cette approche peut être paramétrée pour supporter une plus large gamme d'activités de conception. Il préconise comme son prédécesseur quatre vues : la vue tâche, la vue organisation, la vue produit et la vue processus.

6.10. Le projet ALF (1987 - 1992)

N. Boudjlida et ***al.*** de l'Université de Loria présentent dans ALF [52] un framework pour construire des environnements de génie logiciel centrés processus. Ce projet avait pour objectif de fournir des facilités pour supporter les activités de développement de logiciels. Il se propose de rompre avec la mécanique des environnements de génie logiciel en proposant des éléments pour des environnements beaucoup plus orientés activités et objectifs de l'utilisateur. Il fournit deux fonctionnalités remarquables à savoir une assistance intelligente à l'utilisateur et la possibilité pour l'utilisateur de paramétrer son environnement.

6.11. WayPointer

WayPointer [53] est défini par JAB comme une application personnelle de productivité basée sur les agents intelligents qui supportent un développeur individuel dans la modélisation avec le langage UML. WayPointer permet de définir les objectifs en termes d'artefacts qui peuvent être spécifiés ou affinés à la demande et il fournit un support proactif et des conseils sur la réalisation des objectifs définis. WayPointer assiste le développeur dans la modélisation des cas d'utilisation, l'analyse et la conception.

6.12. OOExpertProcessModel

L' OOExpertProcessModel [54] Utilise un système d'agents intelligents qui vise à aider les concepteurs lors de la conception orientée objet du logiciel en automatisant les tâches mal définies dans le processus de création du modèle objet, y compris l'identification des objets, les associations entre les objets identifiés, les attributs, le comportement et l'organisation des objets avec l'héritage. Le système OOExpert propose d'abord des modèles formels du processus de création du modèle objet. Et ensuite, il formule les modèles de conception et les règles de résolution des problèmes rencontrés dans le modèle et les stocke dans des bases de connaissances de l'agent.

6.13. UCDA (Use Case Driven Assistant)

UCDA [55] décrit un outil CASE assistant au développeur qui propose une méthode semi automatique qui assiste le développeur dans la modélisation à partir des use cases jusqu'au modèle de classes en suivant RUP et la méthode OMCP (***object model creation process***). La méthode est basée sur la spécification des cas d'utilisation à partir des phrases en langage naturel.

6.14. ISPMS (Intelligent Software Process Management System)

Zhao, Keith Chan et Mingshu Li [56] aperçoivent que les technologies traditionnelles de génie logiciel appliquées au SPMs et PSEEs ne conviennent pas pour modéliser les processus logiciels qui sont centrés humain, et proposent une approche agents dans laquelle les processus logiciels sont considérés comme la collaboration d'un groupe d'agents de processus qui savent gérer les activités de développement de logiciels et peuvent agir de la même façon que les développeurs de logiciels ont l'intention de le faire sur la planification, le déroulement et la réflexion concernant leur travail.

6.15. Analyse

Les projets CPCE et Argo se sont particulièrement intéressés à la collaboration. Ils définissent des méthodes d'assistance et de guidage stratégiques et tactiques assez intéressantes. Néanmoins, les prototypes fournis restent encore incomplets et ne sont pas à la hauteur de la modélisation ou plutôt des objectifs à atteindre. De plus ils utilisent des méthodologies propres qui sont encore moins répandues dans le génie logiciel.

Le projet ALF, resté au niveau de la spécification des méthodes d'assistance et de paramétrage, n'est pas appliqué à un processus particulier. Il demeure un framework qui peut s'adapter à toutes les méthodologies.

ArgoUML et Poseidon for UML s'apparentent à des outils CASE UML avec une assistance consistant à critiquer le travail du développeur. Ils disposent également de vues destinées à répartir le travail du développeur. Ces projets n'en sont pas encore à leur phase terminale qui consistera à pousser un peu plus loin pour assister le développeur dans la réalisation d'un logiciel avec un processus clairement défini. Néanmoins ils n'utilisent pas un Système Multi Agents.

WayPointer est beaucoup plus intéressant du fait qu'il utilise un SMA pour assister le développeur pendant les phases d'expression des besoins et d'analyse conception. Mais, au niveau de son processus, qui est une adaptation du processus générique RUP, n'est pas assez représentatif, il ne dispose que de sept activités sur la trentaine des disciplines concernées. De plus, la version disponible ne gère pas les rôles du processus et surtout ne fournit pas de format d'échange standard si ce n'est qu'avec les outils de Rational Software.

Le projet OOExpertProcessModel et UCDA semble plus intéressant du fait que le premier utilise un système multi agents, mais ils ne couvrent pas tout le cycle de développement. Le dernier travail (ISPMS) utilise une assistance intelligente avec les agents et il donne une solution très importante avec les agents adaptatifs mais il est conçu pour les AGL centré processus.

D'après ce que nous avons vu précédemment (table 3), on remarque que le domaine des AGL reste généralement encore ouvert pour l'amélioration, car les AGL existant ne sont pas complets. Le cycle de vie de logiciel n'est pas totalement couvert par touts les AGLs. Le travail d'équipe n'est pas supporté par touts les AGLs. En plus, ces derniers ne sont pas totalement automatisés…etc.

D'une autre part, nous remarquons aussi que les AGL qui existent s'intéressent plus à l'assistance de la modélisation, c'est à dire à la façon de modéliser. Cependant, peu d'entre eux s'intéressent au déroulement des activités du processus c'est-à-dire à la façon de travailler, et même les AGLs qui suivent un processus de génie logiciel déterminé ne fournissent pas une assistance automatisée. L'assistance au déroulement des activités du processus proposée par ces AGLs est essentiellement guidée par une suite de documents fournis par les auteurs de la méthodologie. Les méthodologies qui existent sont souvent complexes et nécessitent elles aussi une assistance automatisée afin de faciliter l'utilisation de l'outil et le rendre un participant actif au développement du logiciel.

7. Conclusion

L'art de fabriquer des logiciels n'est pas un métier facile. Pour métriser la complexité des systèmes logiciels il convient de se baser sur des principes et méthodes et utiliser des outils performants pour supporter les activités du processus de développement. Dans ce chapitre, nous avons présenté les AGLs appelés encore outils CASE comme un moyen très efficace pour encadrer les développeurs dans tout le cycle de vie du logiciel. Nous avons abordé les AGLs en précisant d'abord leurs objectifs, leurs types et leur apport pour les développeurs pour produire un logiciel de qualité. Nous avons terminé par une étude comparative entre les différents AGLs déjà existant. Nous retenons que, malgré que ces AGLs ont atteint un niveau beaucoup élevé que ce soit dans la couverture des différentes phases du processus logiciel ou dans la qualité du logiciel développé, ils restent encore sujet de controverse et ils sont loin d'être totalement automatisables et doivent donner plus d'assistance et plus de flexibilité aux développeurs. En effet, peu d'AGLs s'intéressent au processus logiciel dont il est acquis que sa qualité conditionne la qualité du logiciel à réaliser. Et encore plus, peu d'outils réalisés utilisent des agents, une technologie dont les avantages sont indiscutables.

L'introduction d'une couche d'assistance active et automatisable au processus logiciel modélisée avec un système multi agents est donc nécessaire. Une approche agents sera donc le sujet de notre chapitre suivant dans le but d'éclaircir le concept d'agent et d'introduire leurs caractéristiques.

Chapitre 2

L'Approche Agents

1. Introduction

Bien que le sujet des systèmes multi agents (SMA) ne soit pas récent, il reste toujours un domaine de recherche très vaste et très actif. C'est une discipline qui lie entre plusieurs domaines de recherche en informatique tels que l'intelligence artificielle, les systèmes informatiques distribués, le génie logiciel …etc. Elle s'intéresse aux comportements collectifs produits par les interactions entre plusieurs entités autonomes et flexibles appelées agents, que ce soit, ces interactions tournent autour de la coopération, la concurrence ou la coexistence entre ces agents. [57]

Définition

Dans la littérature, on trouve plusieurs définitions des agents. Elles se ressemblent toutes et diffèrent seulement par le type d'application pour laquelle l'agent est conçu.

Nous retenons ici la définition proposée par J. Ferber [58] :

On appelle agent une entité physique ou virtuelle

a. ***qui est capable d'agir dans un environnement,***
b. ***qui peut communiquer directement avec d'autres agents,***
c. ***qui est mue par un ensemble de tendances (sous la forme d'objectifs individuels ou d'une fonction de satisfaction, voire de survie, qu'elle cherche à optimiser),***
d. ***qui possède des ressources propres,***
e. ***qui est capable de percevoir (mais de manière limitée) son environnement,***
f. ***qui ne dispose que d'une représentation partielle de cet environnement (et éventuellement aucune),***
g. ***qui possède des compétences et offre des services,***
h. ***qui peut éventuellement se reproduire,***
i. ***dont le comportement tend à satisfaire ses objectifs, en tenant compte des ressources et des compétences dont elle dispose, et en fonction de sa perception, de ses représentations et des communications qu'elle reçoit.***

Chacun des termes de cette définition est important. Une ***entité physique*** est quelque chose qui agit dans le monde réel: un robot, un avion ou une voiture sont des exemples d'entités physiques. En revanche, un composant logiciel, un module informatique sont des ***entités virtuelles***, car elles n'existent pas physiquement.

Les agents sont ***capables d'agir***, et non pas seulement de raisonner comme dans les systèmes d'IA classique. L'action, qui est un concept fondamental pour les systèmes multi-agents, repose sur le fait que les agents accomplissent des actions qui vont modifier l'environnement des agents et donc leurs prises de décision futures. Ils peuvent aussi ***communiquer entre eux***, et c'est d'ailleurs là l'un des modes principaux d'interaction existant entre les agents. Ils agissent dans un ***environnement***, sauf pour les agents purement communicants pour lesquels toutes les actions se résument à des communications.

Les agents sont doués d'***autonomie***. Cela signifie qu'ils ne sont pas dirigés par des commandes venant de l'utilisateur (ou d'un autre agent), mais par ***un ensemble de tendances*** qui peuvent prendre la forme de buts individuels à satisfaire ou de fonctions de satisfaction ou de survie que l'agent cherche à optimiser. On pourrait dire ainsi que le moteur d'un agent, c'est lui-même. C'est lui qui est actif. Pour être autonome, un agent doit [30, 31, 39, 40] ; Créer ses propres objectifs à la base des événements internes et externes ; Prendre des décisions sur les instanciations buts ; avoir ses propres processus de contrôle ; et Créer les objectifs proactifs sans intervention externe directe (proactivité). Il a la possibilité de répondre par l'affirmative ou le refus à des requêtes provenant des autres agents. Il dispose donc d'une certaine liberté de manœuvre, ce qui le différencie de tous les concepts semblables, qu'ils s'appellent "objets", "modules logiciels" ou "processus". Mais l'autonomie n'est pas seulement comportementale, elle porte aussi sur les ressources. Pour agir, l'agent a besoin d'un certain nombre de ressources: énergie, CPU, quantité de mémoire, accès à certaines sources d'informations, …etc. Ces ressources sont à la fois ce qui rend l'agent non seulement dépendant de son environnement, mais aussi, en étant capable de gérer ces ressources, ce qui lui donne une certaine indépendance vis-à-vis de lui. L'agent est ainsi à la fois un système ouvert (il a besoin d'éléments qui lui sont extérieurs pour survivre) et un système fermé (car les échanges qu'il a avec l'extérieur sont très étroitement réglementés).

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

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

C'est la présence de tous ces attributs dans une seule entité logicielle qui procure la force au paradigme agent et qui le distingue des autres paradigmes logiciels tels que les systèmes orientés objets, les systèmes distribués et les systèmes experts. [59]

2. Caractéristiques d'un agent

De cette définition ou d'autres qui apparaissent dans la littérature [60, 61, 62, 63,64] nous pouvons identifier plusieurs propriétés qui caractérisent les agents :

• ***Autonomie*** : Les agents peuvent travailler sans l'intervention directe de l'usager et ont un certain contrôle de leurs actions et état interne.

• ***Réactivité*** : Les agents peuvent percevoir leur environnement, qui peut inclure le monde physique (un usager derrière une interface graphique, des applications sur le réseau) ou d'autres agents et répondre à des changements qui se produisent dans celui-ci.

• ***Initiative*** : Le comportement des agents est déterminé par les buts qu'ils poursuivent et par conséquent ils peuvent produire des actions qui ne sont pas seulement des réponses à leur environnement.

• ***Habilité sociale*** : Pour satisfaire ses buts un agent peut demander la collaboration d'autres agents à qui il délègue ou avec lesquels il partage la réalisation de tâches. Pour cela, il a besoin de se coordonner, négocier, en définitive, interagir.

• ***Raisonnement*** : Un agent peut décider quel but poursuivre ou à quel événement réagir, comment agir pour accomplir un but, ou suspendre ou abandonner un but pour se dédier à un autre.

• ***Apprentissage*** : L'agent peut s'adapter progressivement à des changements dans des environnements dynamiques grâce à des techniques d'apprentissage.

• ***Mobilité*** : Dans des applications déterminées, il peut être intéressant de permettre aux agents de migrer d'un nœud à un autre dans un réseau tout en préservant leur état lorsqu'ils sautent entre les nœuds.

Il est intéressant de noter, comme cela est montré dans la figure 7 [64], qu'il y a trois lignes de recherche dans l'aire des agents, selon qu'on donne plus ou moins de l'importance aux aspects cognitifs (raisonnement, apprentissage), sociaux (communication et coopération) ou de mobilité (agents mobiles).

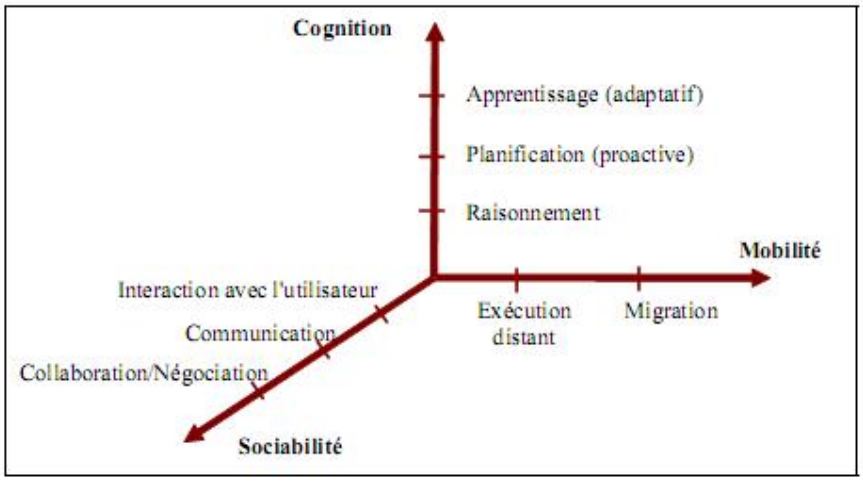

Figure 7 : Caractérisation des agents

Il est classique de confondre les objets avec les agents. En effet, comme les objets, les agents ont des comportements hérités et des données encapsulées. Seulement, les agents sont aussi des entités actives avec des connexions avec leur environnement. Elles possèdent un modèle interne du monde, des bases de connaissance et des facilités de communication [19]. La notion d'agent englobe celle d'objet mais possède des caractéristiques que les objets n'ont pas : Les agents ont une capacité d'autonomie plus large que les objets…

3. Architectures d'un agent

Un agent se caractérise essentiellement par la manière dont il est conçu et par ses actions, en d'autres termes par son architecture et par son comportement [58].

Il existe plusieurs manières de concevoir des agents, mais peu importe l'architecture adoptée, un agent peut toujours être vu comme une fonction liant ses perceptions à ses actions. Plus précisément, ***un agent perçoit l'environnement à l'aide de ses capteurs et il agit sur son environnement à l'aide de ses effecteurs***. Ce qui différencie les différentes architectures d'agents, c'est la manière dont les perceptions sont liées aux actions. [59]

Les auteurs Russel et Norvig [65] regroupent les architectures d'agents en trois types, à savoir : Les agents réactifs et qui regroupent les agents à réflexes simples et les agents conservant une trace du monde et Les agents délibératifs qui regroupent les agents ayant des buts et les agents utilisant une fonction d'utilité et les agents hybrides.

3.1. Agent réactif

Un agent réactif ne fait réagir qu'aux changements qui surviennent dans l'environnement. Autrement dit, un tel agent ne fait ni délibération ni planification, il se contente simplement d'acquérir des perceptions et de réagir à celles-ci en appliquant certaines

règles prédéfinies. Etant donné qu'il n'y a pratiquement pas de raisonnement, ces agents peuvent agir et réagir très rapidement.

3.1.1. Agent à réflexes simples

Ce type d'agent agit uniquement en se basant sur ses perceptions courantes. Il utilise un ensemble de règles prédéfinies, du type Si ***condition*** alors ***action***, pour choisir ses actions ; ces règles permettent d'avoir un lien direct entre les perceptions de l'agent et ses actions. Le comportement de l'agent est donc très rapide, mais peu réfléchi. A chaque fois, l'agent ne fait qu'exécuter l'action correspondant à la règle activée par ses perceptions.

La figure 8 [65] montre l'architecture d'un agent à réflexes simples. Les rectangles représentent l'état interne de l'agent dans son processus de décision et les ovales représentent les informations qui sont utilisées dans le processus. L'agent se bâtit une représentation du monde à l'aide de ses perceptions lui venant de ses capteurs. Par la suite, il utilise ses règles pour choisir l'action qu'il doit effectuer selon ce qu'il perçoit de l'environnement.

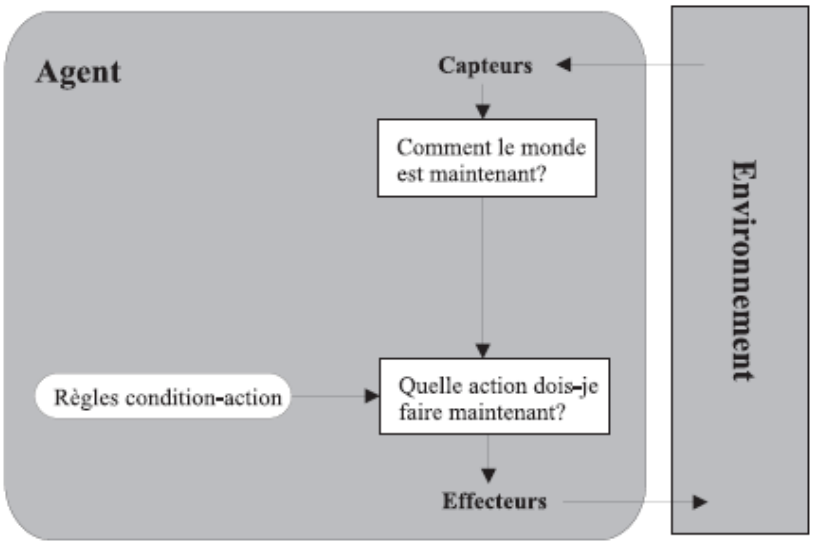

Figure 8 : Schéma d'un agent à réflexes simples

3.1.2. Agent conservant une trace du monde

Le problème posé par le type d'agents décrit précédemment est que les capteurs de l'agent ne fournissent pas une vue ***complète*** du monde. Pour régler ce problème, l'agent doit maintenir des informations internes sur l'état du monde dans le but d'être capable de distinguer deux situations qui ont ***des perceptions identiques, mais qui, en réalité, sont différentes.***

Pour que l'agent puisse faire évoluer ses informations internes sur l'état du monde, il a besoin de deux types d'information. Tout d'abord, il doit avoir des informations sur la

manière dont le monde évolue, indépendamment de l'agent. L'agent doit avoir ensuite des informations sur la manière dont ses propres actions affectent le monde autour de lui.

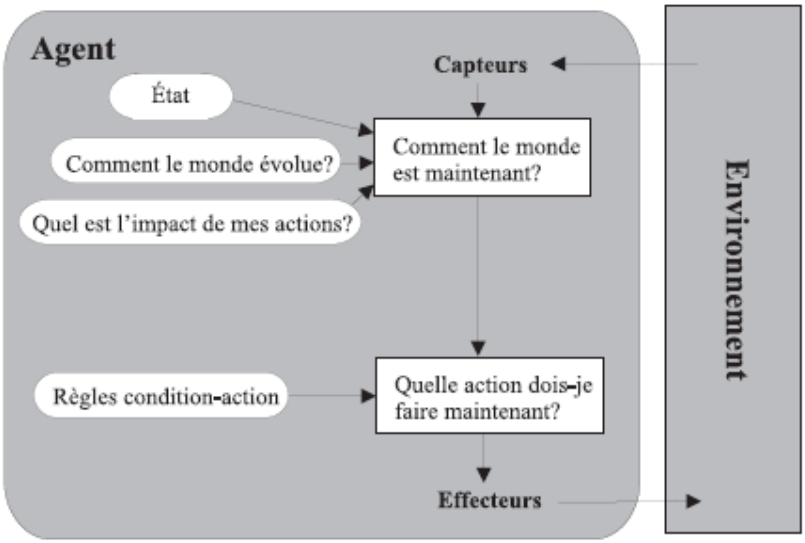

Figure 9 : Schéma d'un agent conservant une trace du monde

On peut voir, sur la figure 9 [65], la structure d'un agent conservant une trace du monde. Il utilise ses informations internes (état précédent du monde, l'évolution du monde et l'impact de ses actions) pour mettre à jour ses perceptions actuelles. Par la suite, il choisit ses actions en se basant sur cette perception « améliorée » du monde qui l'entoure.

3.2. Agents délibératifs

Les agents délibératifs sont des agents qui effectuent une certaine délibération pour choisir leurs actions. Une telle délibération peut se faire en se basant sur les buts de l'agent ou sur une certaine fonction d'utilité. Elle peut prendre la forme d'un plan qui reflète la suite d'actions que l'agent doit effectuer en vue de réaliser son but.

3.2.1. Agent ayant des buts

Les agents réactifs utilisent leurs connaissances sur l'état actuel de l'environnement pour choisir leurs actions mais, cela peut s'avérer insuffisant pour prendre une décision sur l'action à effectuer. Donc, l'agent a besoin, en plus de la description de l'état actuel de son environnement, de certaines informations décrivant ses buts qui peuvent être vus comme des situations désirables pour l'agent.

Les agents délibératifs (voir Figure 10), qui raisonnent sur les buts, tiennent compte d'une certaine projection dans le futur, pour voir l'impact de leurs actions et choisirent leurs actions en se basant sur leurs buts. Bien entendu, l'agent raisonnant sur ses buts prend, en

général, beaucoup plus de temps à agir qu'un agent réactif. Il offre en revanche beaucoup plus de flexibilité.

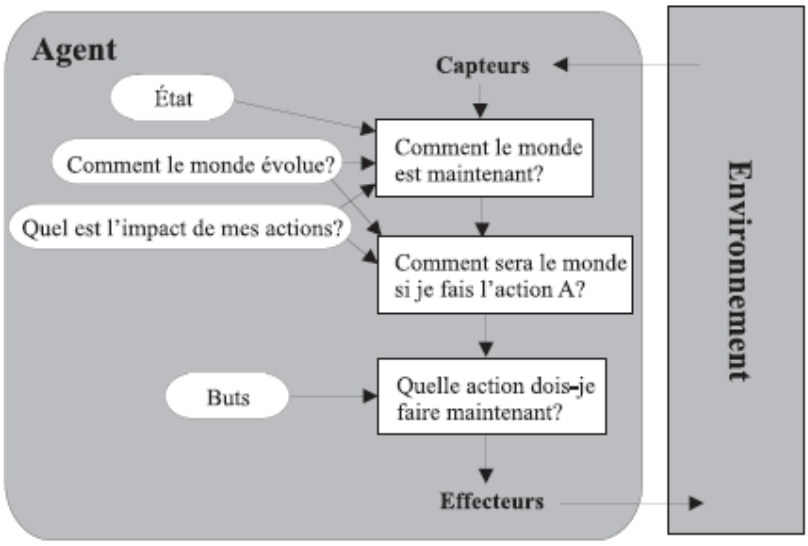

Figure 10 : Schéma d'un agent ayant des buts [65].

3.2.2. Agent utilisant une fonction d'utilité

Dans plusieurs situations, les buts ne sont pas suffisants pour générer un comportement de haute qualité. Par exemple, s'il y a plusieurs chemins possibles pour atteindre un port, certains seront plus rapides et d'autres plus dangereux. Dans cette situation, l'agent raisonnant uniquement sur ses buts n'a pas de moyens pour choisir le meilleur chemin, son seul but étant de se rendre à destination. Cela se produit car les buts ne procurent qu'une simple distinction entre les états où l'agent est satisfait ou non. En fait, l'agent doit plutôt s'appuyer sur une manière plus fine dans l'évaluation des états pour être en mesure de reconnaître pour chacun des états son degré de satisfaction. Pour cela, on dit que l'agent va ***préférer*** un état à un autre si son utilité est plus grande dans le premier état que dans le deuxième.

Généralement, l'utilité est une fonction qui attribue une valeur numérique à chacun des états. Plus l'état a une grande valeur, plus il est désirable pour l'agent. Dès lors, la spécification d'une fonction d'utilité permet à l'agent de prendre des décisions rationnelles dans deux cas de situations où le raisonnement sur les buts échoue. Par exemple, lorsqu'il y a des buts conflictuels qui ne peuvent pas être satisfaits en même temps (par exemple, la vitesse et la sécurité), la fonction d'utilité spécifie le compromis approprié entre les différents buts. De même, lorsqu'il y a plusieurs buts possibles et qu'aucun d'eux ne peut être atteint avec certitude, la fonction d'utilité permet de pondérer la chance de succès avec l'importance de chacun des buts.

La figure 11 montre le schéma d'un agent basé sur l'utilité. On peut voir que l'agent utilise la fonction d'utilité pour évaluer la pertinence d'une action. Il choisit donc les actions qui l'amèneront dans les états ayant la plus grande valeur d'utilité pour lui.

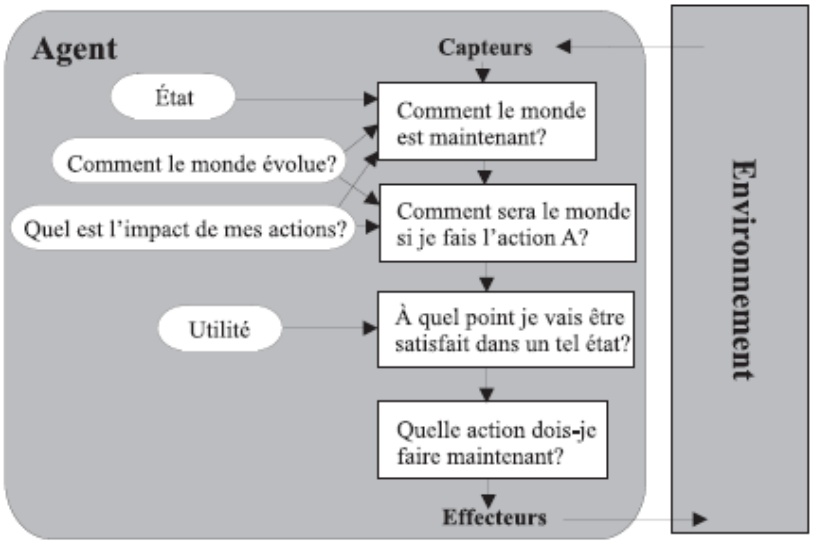

Figure 11 : Schéma d'un agent basé sur l'utilité

3.2.3. Agent BDI (*Belief, Desire, Intentions)*

C'est une architecture bâtie autour d'un ***raisonnement pratique*** [66]. Ces agents sont généralement représentés par un "état mental" ayant les attitudes mentales suivantes : croyances, désirs et intentions pour choisir leurs actions. Dans ce cadre, Wooldrige [67] propose une architecture à sept composants, telle qu'elle présentée sur la figure 12 :

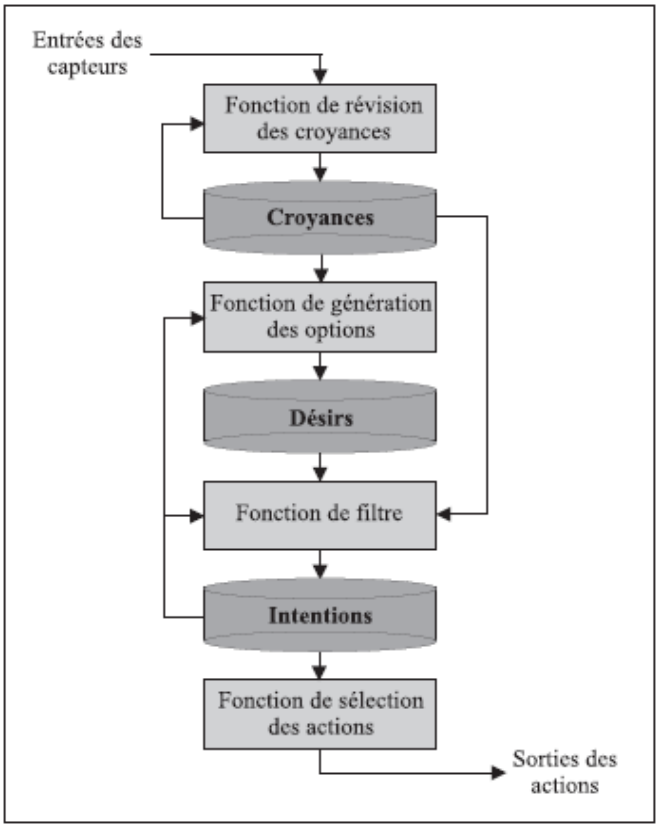

Figure 12 : Diagramme d'une architecture BDI

– Un ensemble de ***croyances*** courantes, représente les informations que l'agent possède à propos de son environnement courant ;

– Une ***fonction de révision des croyances***, prend les entrées des capteurs et les croyances actuelles de l'agent et détermine ensuite un nouvel ensemble de croyances ;

– Une ***fonction de génération des options***, détermine les options disponibles pour l'agent (ses désirs), en se basant sur les croyances courantes de l'agent à propos de son environnement et sur ses intentions courantes ;

– Un ensemble de ***désirs***, représente les options disponibles à l'agent ;

– Une ***fonction de filtre***, représente le processus de délibération de l'agent. Elle détermine les intentions de l'agent en se basant sur ses croyances, ses désirs et ses intentions courantes ;

– Un ensemble d'***intentions*** courantes, représente le centre d'attention actuel de l'agent, c'est-à-dire les buts envers lesquels il s'est engagé et envers lesquels il a engagé des ressources ;

– Une ***fonction de sélection des actions***, détermine l'action à effectuer en se basant sur les intentions courantes de l'agent.

Alors, un agent BDI doit donc mettre à jour ses croyances avec les informations qui lui proviennent de son environnement, décider quelles options lui sont offertes, filtrer ces options afin de déterminer de nouvelles intentions et poser ses actions en se basant sur ses intentions.

3.3. Agent hybride

Chacune des architectures vues précédemment est appropriée pour un certain type de problème. Cependant ; Pour la majorité des problèmes posés, ni une architecture complètement réactive, ni une architecture complètement délibérative, n'est appropriée.

Dans ce cas, une architecture conciliant à la fois des aspects réactifs et délibératifs est requise. On parle alors d'architecture hybride [68, 69, 70, 71, 72, 73, 74], dans laquelle on retrouve généralement plusieurs couches logicielles. Les couches peuvent être arrangées verticalement (seulement une couche a accès aux capteurs et aux effecteurs) ou horizontalement (toutes les couches ont accès aux entrées et aux sorties).

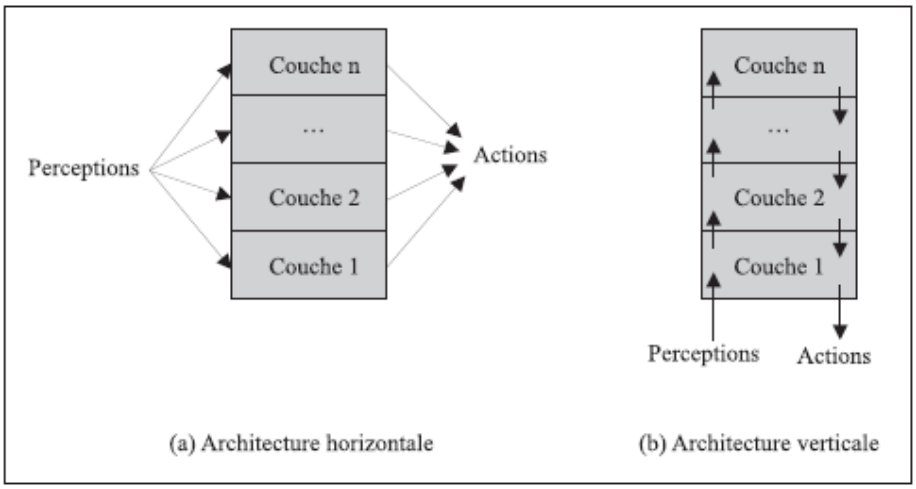

Figure 13 : Architectures d'agents en couches [75]

Un agent est composé de plusieurs couches arrangées selon une hiérarchie (voir figure 13). Au plus bas niveau de l'architecture, on retrouve habituellement une couche purement réactive qui prend ses décisions en se basant sur des données brutes en provenance des senseurs. La couche intermédiaire fait abstraction des données brutes et travaille plutôt avec une vision qui se situe au niveau des connaissances de l'environnement. Finalement, la couche supérieure se charge des aspects sociaux de l'environnement, c'est à dire du raisonnement tenant compte des autres agents.

4. Typologie d'agents

Nwana [76] propose une typologie des agents à partir de plusieurs critères de classification :

- Mobilité : statique ou mobile ;
- Présence d'un modèle de raisonnement symbolique : délibératif ou réactif ;
- Existence d'un objectif et de propriétés initiales comme l'autonomie, la coopération et l'apprentissage. A partir de ces propriétés, Nwana déduit quatre types d'agents : collaboratifs, collaboratifs et apprenants, d'interface et intelligents ;
- Rôles : recherche d'informations ou travail sur Internet
- Philosophies hybrides : combinaison entre deux ou plusieurs approches dans un seul agent.

De cette typologie, Nwana propose sept catégories d'agents : agents collaboratifs, agents d'interface, agents mobiles, agents d'information ou d'Internet, agents réactifs, agents hybrides et agents intelligents.

4.1. Les agents collaboratifs

Les agents collaboratifs ont davantage des caractéristiques d'autonomie et de coopération avec les autres agents dans la réalisation de leurs objectifs. Ils doivent pouvoir négocier afin d'arriver à des compromis acceptables. La faculté d'apprendre peut exister mais n'est

pas une caractéristique fondamentale requise pour ce type d'agents. Plusieurs chercheurs en intelligence artificielle leur prêtent des états mentaux comme les croyances, les désirs et les intentions. Ils sont alors qualifiés d'agents collaboratifs de type BDI (Beliefs, Desires, Intentions). Les propriétés caractéristiques des agents collaboratifs sont : l'autonomie, la réactivité, le dynamisme (capacité à initier des actions) et la sociabilité. Les principales raisons pour lesquelles on peut être amené à implémenter les agents collaboratifs relèvent de l'intelligence artificielle distribuée ; elles sont les suivantes :

- résoudre des problèmes trop importants pour un seul agent à cause des limitations des ressources et des risques de systèmes centralisés ;
- permettre l'interconnexion et l'interopérabilité de plusieurs legacy systems comme les systèmes experts, les systèmes décisionnels …etc. ;
- résoudre des problèmes fondamentalement distribués comme le contrôle du trafic aérien ou des problèmes inhérents aux systèmes d'information distribués ;
- résoudre des problèmes pour lesquelles l'expertise disponible est distribuée ;
- encourager la modularité, la vitesse d'exécution et la flexibilité.

4.2. Les agents d'interface

Les agents d'interface disposent d'une autonomie et d'une capacité à apprendre. Ils sont caractérisés par la métaphore d'assistant personnel collaborant avec l'utilisateur dans le même environnement. Alors que les agents collaboratifs collaborent avec les autres agents, les agents d'interface collaborent avec l'utilisateur. La collaboration avec l'utilisateur ne nécessite pas l'existence explicite d'un langage de communication entre agents comme dans le premier cas. Les agents d'interface apprennent les préférences des utilisateurs afin de mieux les assister. Ils apprennent :

- en observant et en imitant leurs utilisateurs ;
- à travers le feed-back des utilisateurs suite à leurs actions ;
- en recevant des instructions de l'utilisateur ;
- en demandant les conseils des autres agents.

La collaboration avec les autres agents lorsqu'elle existe se limite à la demande de conseils, et diffère ainsi du type de collaboration qui a cours entre les agents collaboratifs. Un agent d'interface est doté d'un minimum de connaissances au départ et acquiert lui-même, par la suite, la connaissance dont il a besoin pour assister l'utilisateur. La principale raison qui amène à considérer de tels agents est le souci de déléguer certaines tâches répétitives et fastidieuses (il faut d'ailleurs signaler que la répétitivité de certaines actions ou attitudes dans le comportement de l'utilisateur est le gage de la réussite de tels agents, autrement, ils ne pourraient rien apprendre de ce comportement).

4.3. Les agents mobiles

Les agents mobiles sont des processus capables de se déployer à travers de grands réseaux d'information comme Internet, interagissant avec différents hôtes, recueillant des informations pour leurs propriétaires et accomplissant des tâches qui leur sont confiées. Ces tâches vont de la réservation de vols d'avions à l'administration de réseaux de télécommunications. Il faut dire que la mobilité n'est pas une caractéristique des agents,

les agents mobiles sont des agents du fait qu'ils sont autonomes et coopèrent (certes différemment des agents collaboratifs). Un agent mobile peut ainsi communiquer ou coopérer avec un autre agent chargé d'informer les autres agents de la localisation de ses attributs et méthodes ; ceci dispense cet agent de rendre publique toutes les informations et données le concernant. Les raisons qui poussent à recourir aux agents mobiles sont les suivantes :

- réduction des coûts de communication : en permettant à l'agent d'aller s'exécuter là où se trouvent les informations brutes, on évite ainsi de les rapatrier sur son système local pour n'en utiliser qu'une petite partie ;
- l'insuffisance des ressources locales : les capacités de traitement et de stockage peuvent être limitées au point de justifier l'usage des agents mobiles qui iraient s'exécuter sur des systèmes distants plus performants ;
- une coordination plus facile : il est plus simple de coordonner un nombre de requêtes distantes et d'en collecter simplement les résultats ;
- un traitement asynchrone : on peut initialiser ses agents mobiles et faire autre chose pendant que ceux-ci s'exécutent ;
- une architecture distribuée plus flexible : les agents mobiles offre une architecture distribuée unique.

4.4. Les agents d'information

Les agents d'information ou agents d'Internet sont apparus du fait de l'explosion de l'information et du besoin de disposer d'outils de manipulation de ces informations. Les agents d'information ont pour rôle d'administrer, manipuler ou collecter les informations à partir de plusieurs sources d'informations distribuées. Du fait de l'avènement et du succès d'Internet, la distinction entre les autres types d'agents, tels que les agents d'interface ou de collaboration, et les agents d'information est subtile : en effet, tous manipulent des informations. Il faut néanmoins garder à l'esprit que les agents d'information sont définis par ce qu'ils font, alors que les autres sont définis par ce qu'ils sont, c'est-à-dire par leurs attributs et propriétés caractéristiques. Les raisons qui président à la mise en œuvre des agents d'information sont de deux ordres. D'une part, il y a la nécessité de faire face à l'explosion des sources d'informations et donc de répondre à un besoin similaire à celui rempli par les moteurs de recherche. D'autre part, sur le plan financier, de tels agents rapporteraient à leur éditeur un succès comparable à celui de Netscape avec son navigateur sur Internet.

4.5. Les agents logiciels réactifs

Les agents réactifs sont une catégorie spéciale d'agents ne disposant pas de modèles internes et symboliques de leur environnement : ils ne font que réagir aux stimuli provenant de l'environnement. Les agents réactifs sont relativement simples et interagissent avec les autres agents de façon basique. La mise au point de tels agents n'exige pas une spécification complète de leur comportement. Les agents réactifs peuvent en outre être considérés comme une collection de modules autonomes chargés chacun d'une tâche spécifique. La communication entre les différents modules est minimale. Les agents réactifs opèrent sur des données brutes provenant des capteurs, contrairement aux précédents types d'agents qui utilisent des représentations symboliques de haut niveau.

L'existence et la mise en œuvre d'agents réactifs se justifient dans les situations où l'on ne veut prendre en compte que des hypothèses basées sur l'environnement physique. Ces agents réactifs sont supposés aussi être plus robustes et plus tolérants aux pannes que les autres types d'agents, la perte d'un tel agent n'entraîne pas de conséquences catastrophiques. Ce sont des agents situés qui n'ont pas besoin d'avoir une représentation de leur environnement, ils ne réagissent qu'aux sollicitations du moment.

4.6. Les agents hybrides

Nous avons passé en revue cinq types d'agents : collaboratifs, d'interface, mobiles, d'information et réactifs. Le débat sur l'opportunité d'avoir tel ou tel type d'agent est davantage un débat théorique et plutôt stérile. Chaque type d'agent possède ses points forts et faibles, ses avantages et ses inconvénients. Une approche consiste donc souvent dans la réalité à bâtir des agents hybrides. Un agent hybride consiste en la combinaison de plusieurs caractéristiques au sein d'un même agent ; ces caractéristiques concernent la mobilité, la collaboration, l'autonomie, la capacité à apprendre, …etc. L'utilisation d'agents hybrides reste toujours guidée par le souci de minimiser les faiblesses et d'augmenter les forces de tel ou tel type d'agents que l'on mettrait en œuvre dans une application.

5. Système multi-agents

Dans la plupart des applications, un agent n'est pas seul dans son environnement, il y a presque toujours d'autres agents présents autour de lui. Les agents doivent, par conséquent, être capables d'interagir entre eux. Ils peuvent soit ***coexister***, ***coopérer*** ou être en ***compétition*** [69], [70], [78]. S'ils ne font que coexister, alors chaque agent ne fait que considérer les autres agents comme des composantes de l'environnement. S'ils coopèrent, alors les agents doivent pouvoir communiquer et se coordonner pour agir efficacement ensemble. S'ils sont en compétition, alors les agents doivent être en mesure de négocier. Un système où évoluent plusieurs agents est appelé un système multi agents et il possède généralement plusieurs caractéristiques intéressantes, comme le parallélisme, la robustesse et l'extensibilité. [70]

5.1. Définition

Un système multi-agent est un système distribué composé d'un ensemble d'agents. Contrairement aux systèmes d'IA, qui simulent dans une certaine mesure les capacités du raisonnement humain, les SMA sont conçus et implantés idéalement comme un ensemble d'agents interagissant, le plus souvent, selon des modes de ***coopération***, de ***concurrence*** ou de ***coexistence*** [69], [70], [77].

Un SMA est généralement caractérisé par [75] :

– chaque agent a des informations ou des capacités de résolution de problèmes limitées et ainsi, chaque agent ne possède qu'un point de vue partiel,

– il n'existe aucun contrôle global du système multi agents,

– les donnés sont décentralisées,

– le calcul est asynchrone.

Les SMA peuvent être vus comme l'intersection de plusieurs domaines scientifiques :

- L'intelligence artificielle pour les aspects prise de décision de l'agent
- L'intelligence artificielle distribuée pour la distribution de l'exécution
- Les systèmes distribués pour les interactions
- Le génie logiciel pour l'approche ***agents*** et l'évolution vers des composants logiciels de plus en plus autonomes

Dans un SMA, les activités individuelles des agents font tendre le système vers un comportement global qui n'est défini par aucun agent ou une règle globale : On parle ***d'émergence de comportement***.

Lors de la conception d'un SMA, il s'agit de définir des comportements individuels faisant émerger des comportements globaux qui feront converger le système vers l'objectif global de l'application.

De nombreuses expérimentations sont généralement nécessaires avant de définir les comportements individuels qui font émerger le comportement global souhaité car un comportement émergent est difficilement prévisible. [78]

5.2. Utilité des systèmes multi agents

Les systèmes multi agents s'adaptent bien aux situations où les domaines requièrent l'utilisation de plusieurs entités. Par exemple, les systèmes qui sont géographiquement distribués comme la coordination entre plusieurs frégates, le contrôle aérien, les bases de données coopératives distribuées, …etc.

Tous ces domaines sont par définition distribués et par conséquent, les systèmes Multi agents procurent une façon facile et efficace pour les modéliser.

Une autre situation, où les systèmes multi agents sont sollicités, est lorsque les différents systèmes et les données qui s'y rattachent appartiennent à des organisations indépendantes qui veulent garder leurs informations privées et sécurisées pour des raisons concurrentielles. Par exemple, les missions maritimes qui se tiennent en collaboration avec plusieurs pays.

Même si le domaine ne requiert pas l'utilisation des systèmes multi agents, il existe tout de même de bons avantages à utiliser ce type de système. Ainsi, ils peuvent s'avérer bien utiles pour des problèmes possédant de multiples méthodes de résolution, de multiples perspectives et/ou de multiples résolveurs. En particulier, les systèmes multi agents sont très utiles pour modéliser le raisonnement humain à l'intérieur de grandes simulations de combats aériens [79].

Ils possèdent également les avantages traditionnels de la résolution distribuée et concurrente de problèmes [80]:

– La ***modularité*** permet de rendre la programmation plus simple. Elle permet aussi aux systèmes multi agents d'être facilement extensibles car, il est plus facile d'ajouter de nouveaux agents à un système multi agent que d'ajouter de nouvelles capacités à un système monolithique.

– La ***vitesse*** est principalement due au parallélisme, car plusieurs agents peuvent travailler en même temps pour la résolution d'un problème.

– La ***fiabilité*** peut être également atteinte, dans la mesure où le contrôle et les responsabilités étant partagés entre les différents agents, le système peut tolérer la défaillance d'un ou de plusieurs agents. Si une seule entité contrôle tout, alors une seule défaillance de cette entité fera en sorte que tout le système tombera en panne.

Finalement, les systèmes multi agents héritent aussi des bénéfices envisageables du domaine de l'intelligence artificielle comme le traitement symbolique (au niveau des connaissances), la facilité de maintenance, la réutilisation et la portabilité.

5.3. L'interaction

D'après J. Ferber [81], une interaction est une mise en relation dynamique de deux ou plusieurs agents par le biais d'un ensemble d'actions réciproques.

Les agents interagissent à travers un ensemble d'évènements pendant lesquels les agents sont en relation les uns avec les autres soit directement, soit par le biais de l'environnement.

Ainsi, la notion d'interaction suppose :

1. La présence d'agents capables d'agir et/ou de communiquer.

2. Des situations susceptibles de servir de point de rencontre entre agents : collaboration, déplacement de véhicules amenant à une collision, utilisation de ressources limitées, régulation de la cohésion d'un groupe.

3. Des éléments dynamiques permettant des relations locales et temporaires entre agents : communication, choc, champ attractif ou répulsif, …etc.

4. Un certain "jeu" dans les relations entre les agents leur permettant à la fois d'être en relation, mais aussi de pouvoir se séparer de cette relation, c'est-à dire de disposer d'une certaine autonomie. Si des agents sont totalement liés par un couplage fixe, leur interaction devient rigide et ils n'interagissent plus au sens plein du terme.

En fait, l'aspect pluriel des Systèmes Multi Agents a apporté une dimension supplémentaire de l'individu et son intelligence : l'intelligence ne peut être comprise d'un point de vue individuel, c'est le résultat d'un effort commun d'échanges, de partages et d'interdépendances d'un ensemble d'agents : dans l'absence de l'interaction, un agent devient isolé dépourvu d'aptitudes à l'adaptation.

Les circonstances dans lesquelles des agents entrent dans un processus interactif définissent et donnent des critères de comparaison entre les différents types d'interaction. Ces circonstances permettant l'interaction entre les agents sont appelées situations d'interaction dont voici la définition :

On appellera situation d'interaction [81] un ensemble de comportements résultant du regroupement d'agents qui doivent agir pour satisfaire leurs objectifs en tenant compte des contraintes provenant des ressources plus ou moins limitées dont ils disposent et de leurs compétences individuelles.

5.3.1. Composantes des interactions

Chaque agent peut être caractérisé suivant trois dimensions : ses buts, ses capacités à réaliser certaines tâches et les ressources dont il dispose. Les principales situations d'interaction dans lesquelles se trouvent les agents, sont classées selon trois critères principaux : la compatibilité des objectifs, la disponibilité des ressources et la capacité des agents par rapport aux tâches.

5.3.1.1. Compatibilité des buts

Selon J.Ferber [81], la compatibilité des buts est définit comme suit : Le but d'un agent A est incompatible avec celui d'un agent B si les agents A et B ont comme buts respectifs d'atteindre les états décrits respectivement par p et q et que p $-\!\!\rightarrow$eq, c'est-à-dire que : satisfait(but(A,p)) $-\!\!\rightarrow$ not(satisfait(but(B,q)).

A partir de cette définition, nous aboutissons à une première classification : deux agents sont dans une situation de coopération ou à la limite d'indifférence si leurs buts sont compatibles et dans une situation d'antagonisme, sinon.

5.3.1.2. Disponibilité des ressources

On entend par les ressources tous les éléments environnementaux et matériels utiles à la réalisation d'une action. Ces ressources, nécessairement limitées, sont des sources de conflits : chaque agent perçoit l'autre comme gêneur quant à l'accomplissement de ses actions.

5.3.1.3. Capacités des agents par rapport aux tâches

Dans plusieurs cas, un agent ne peut accomplir ses tâches et réaliser ses buts tout seul : ses seules compétences ne lui permettent pas de s'en sortir tout seul. Dans ce cas, l'agent est obligé d'interagir avec les autres en se groupant pour accomplir cette tâche ensemble.

Les interactions des agents d'un SMA sont motivées par l'interdépendance des agents selon ces trois dimensions : leurs buts peuvent être compatibles ou non; les agents peuvent désirer des ressources que les autres possèdent; un agent X peut disposer d'une capacité nécessaire à un agent Y pour l'accomplissement d'un des plans d'action de Y [57].

Les types d'interaction sont multiples et variés et sont divisés en trois catégories : l'indifférence, l'antagonisme et la coopération.

En réalité, la coopération est la catégorie la plus sollicitée et la plus étudiée lors de la conception d'un SMA.

5.4. La coopération

La coopération est une forme d'interaction. Elle consiste à établir qui fait quoi, avec quels moyens, de quelle manière et avec qui. Cela sous-entend qu'il faut trouver des solutions aux différents sous-problèmes que constitue la collaboration par répartition de tâches, la coordination d'actions et la résolution de conflits. La coopération se résume donc par la formule :

Coopération = collaboration + coordination d'actions + résolution de conflits

La résolution de conflit supposant une négociation pour lever ce conflit. Plusieurs agents coopèrent ou sont dans une situation de coopération si l'une des conditions suivantes est vérifiée :

• L'ajout d'un nouvel agent permet d'accroître différentiellement les performances du groupe

• L'action des agents sert à éviter ou à résoudre des conflits potentiels ou actuels

5.4.1. Les méthodes de coopération

Ce sont les moyens offerts aux agents pour coopérer, ils sont :

5.4.1.1. Le regroupement et la multiplication

Il consiste à se regrouper pour former un groupe homogène sinon dense. Dans le contexte multi agents, ce regroupement offre une fiabilité accrue au système –réaction aux pannes par exemple- ainsi qu'une possibilité de spécialisation des agents qui le composent. En effet, un plus grand nombre d'agents permet à chacun des composants à se concentrer sur certaines activités plutôt que d'autres. Cette multiplication du nombre de participants et ce principe de vie en communauté ainsi que leurs avantages peuvent être étudiés analogiquement avec ceux de la vie réelle en société.

5.4.1.2. La spécialisation

La spécialisation est le processus par lequel des agents deviennent de plus en plus adaptés à leurs tâches. Il est souvent difficile de faire des agents qui soient spécialisés dans toutes les tâches. La réalisation performante d'une tâche suppose souvent des

caractéristiques structurelles et comportementales qui ne peuvent permettre d'effectuer d'autres tâches avec efficacité. La spécialisation n'est pas nécessairement le fruit d'un choix à priori. Des agents initialement totipotents peuvent se spécialiser progressivement dans l'accomplissement de leur tâche, ce qui revient à leur associer des rôles temporaires par une sorte d'adaptation individuelle. Cette spécialisation peut être bénéfique à la collectivité en augmentant la capacité du groupe à résoudre plus rapidement un problème semblable. A. Drogoul [82] a montré que, si l'on utilise des agents capables de renforcer leur tendance à s'occuper préférentiellement d'une tâche, alors une société spécialisée est plus efficace qu'une société d'agents totipotents.

5.4.1.3. La collaboration par partage de tâches et de ressources

La collaboration est l'ensemble des techniques permettant à des agents de se répartir des tâches, des informations et des ressources afin de réaliser une tâche commune. Cependant, il existe de nombreux moyens de répartition de tâches et de moyens. Pour les agents cognitifs, il s'agit soit d'avoir pour les agents des représentations mutuelles de leurs capacités (réseaux d'accointances) [83], soit d'utiliser des techniques d'appel d'offre où les agents sont à la fois demandeurs et offreurs. La technique d'appel d'offre la plus connue est le "contract net") [84].

5.4.1.4. La coordination d'actions

C'est le processus par lequel l'agent raisonne sur ses actions locales et les actions (anticipées) des autres agents pour s'assurer que la communauté agit d'une manière cohérente [85].

Gérer un ensemble d'agents suppose l'exécution d'un certain nombre de tâches supplémentaires qui ne sont pas directement productives mais qui servent simplement à faire en sorte que les actions productives puissent s'accomplir dans les meilleures conditions. Lorsqu'il s'agit de système monolithique, ces tâches supplémentaires font partie du système organisationnel, mais dans le cas d'un système multi-agent, c'est-à-dire d'un travail nécessairement distribué, on parle de ***tâches de coordination***. Celles-ci sont indispensables dès lors que l'on se trouve en présence d'un ensemble d'agents autonomes qui poursuivent leurs propres buts, la réalisation de tâches productives entraînant avec elles tout une série de tâches de coordination sans lesquelles les premières ne peuvent être accomplies.

La phase de coordination d'actions est directement impliquée dans la définition de l'ordre des actions à effectuer.

5.4.1.5. La résolution de conflit par arbitrage et négociation

Les activités des agents dans un système multi-agents sont souvent interdépendantes et conflictuelles. L'arbitrage et la négociation sont deux moyens utilisés pour résoudre ces conflits. L'arbitrage permet de définir les règles de comportements, dont le résultat global limite des conflits et préserve les individus ainsi que la société d'agents [58].

Dans une situation conflictuelle concernant l'objectif ou la ressource, les agents peuvent chercher eux-mêmes un accord bilatéral pendant le processus de la négociation. Cette dernière assure la coopération constructive à l'intérieur d'un groupe d'agents indépendants, ayant leurs propres buts [86].

D'après Bouron T. [87], la négociation définit une stratégie de résolution qui utilise le dialogue pour obtenir un accord concernant les conflits de croyances ou les conflits de buts. Les conflits de croyances sont les résultats de contradictions entre les croyances des différents agents, possédant des connaissances incomplètes. En ce qui concerne les conflits de buts, ils apparaissent à cause de l'incompatibilité des buts des différents agents.

Pendant le processus de négociation entre les agents, on peut distinguer les situations suivantes [88] :

- compromis pour améliorer les capacités du système total.
- les plans sont reçus en un seul plan global. Les conflits potentiels entre les agents sont supprimés soit en ordonnant leurs actions, soit en déterminant la synchronisation;
- ***La planification distribuée,*** où chaque agent planifie les actions par rapport à ses propres buts.

5.5. Communication dans les SMA

En communiquant, les agents peuvent coopérer, négocier, échanger des informations et coordonner leurs activités. Dans les SMA deux stratégies principales ont été utilisées pour supporter la communication entre agents : les agents peuvent échanger des messages directement ou ils peuvent accéder à une base de données partagées (appelée tableau noir ou “blackboard”) dans laquelle les informations sont postées. Les communications sont à la base des interactions et de l’organisation sociale d’un SMA.

5.5.1. Communication par partage d’information

Cette communication consiste à définir une structure de données commune à tous les agents, centralisent les éléments de la solution d’un problème. Cette structure de données convient beaucoup plus pour les agents qui n’ont pas à se connaître mutuellement. On peut donc parler de ***communication implicite***. Le modèle du tableau noir ou “blackboard” constitue l’archétype des systèmes de cette catégorie [89].

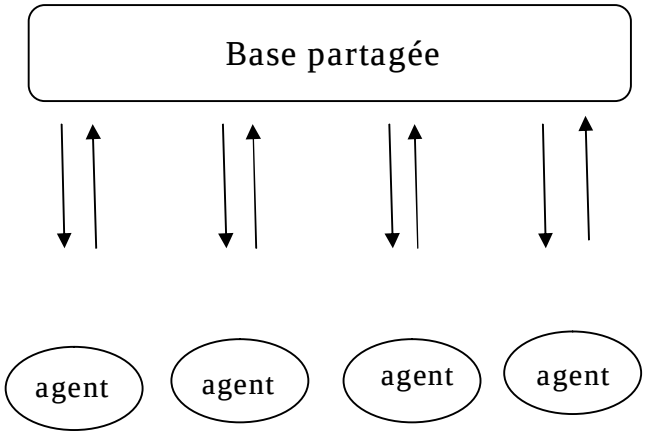

Figure 14 : Communication par partage d'information

Ils sont organisés autour de trois composants, comme il est présenté dans la figure 14 :

- Les agents (souvent appelés source de connaissance ou spécialistes).
- La base partagée (le tableau noir) qui comprend les états partiels d'un problème en cours de résolution et d'une manière générale toutes les informations que s'échangent les agents.
- Un dispositif de contrôle qui gère les conflits d'accès entre les agents. Ces derniers, intervenant de manière « opportuniste » c'est-à-dire être déclenchés effectivement par un système de contrôle.

5.5.2. Communication par envoi de message

A l'inverse des techniques de communication par partage d'information, qui suppose l'existence d'une structure commune. Les systèmes fondés sur la communication par envoi de messages, comme il est présenté dans la figure 15, relèvent d'une distribution totale à la fois des connaissances, des résultats partiels et des méthodes utilisées pour la résolution du problème.

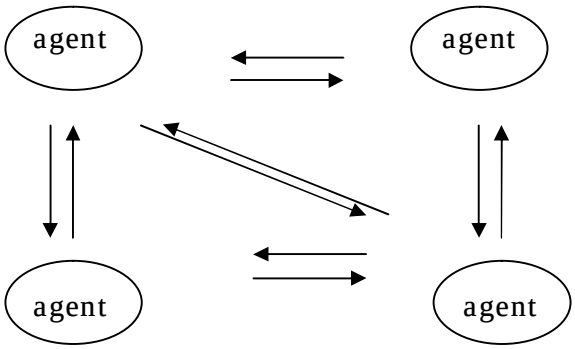

Figure 15 : Communication par envoi de message

On distingue deux grandes catégories dans ce mécanisme [90] :

a. L'appel associatif (pont à pont) : Le message destiné à un agent contient un mot clé auquel est associée chez le récepteur, une action déclenchée par la réception d'un message.
b. L'appel sélectif par filtrage : Un agent n'est sensible à un message que si son filtre associé lui permet de détecter que ce message le concerne. C'est un mécanisme que l'on rencontre en général dans les systèmes d'acteurs.

Il existe deux formes de communication :

a. Les communications intentionnelles mettant en contact des agents cognitifs par le biais d'envoi de message.
b. Les communications réactives prenant la forme des signaux transmis entre agents réactifs.

5.5.3. Les actes de langage

Les actes de langage désignent l'ensemble des actions intentionnelles effectuées au cours d'une communication. Il existe plusieurs types d'actes de langage.

Austin a distingué trois différents aspects des actes du discours (catégorie qui tient compte surtout de la structure conversationnelle) : ce sont les actes locutoires, illocutoires et perlocutoires. [91] :

1. La composante locutoire concerne la génération matérielle des énoncés i.e. le mode de production de phrases à l'aide d'une grammaire et d'un lexique donnés.

2. La composante illocutoire se rapporte à la réalisation de l'acte effectué par le locuteur sur le destinataire de l'énoncé. Elle est composée d'une force illocutoire F et d'un contenu propositionnel P qui est l'objet concerné. Ils peuvent être représentés sous la forme F(P) (exemple : affirmer (la guerre est finie)).

3. La composante perlocutoire concerne l'effet des actes illocutoires sur l'état interne du destinataire.

On voit que ces actes concernent trois volets d'expressions et non seulement aux actes illocutoires.

5.5.4. Les conversations

La théorie des actes de langage ne prend en compte que des actes isolés. Ceci n'est pas réaliste puisque, par analogie avec la société humaine, l'émission et la réception de messages a des effets sur l'état interne des intervenants, ce qui engage d'autres émissions et d'autres réceptions en formant une conversation. Dans les Systèmes Multi Agents, la modélisation de ces conversations passe notamment par des protocoles i.e. des séquences valides de messages. Les modélisations les plus courantes de ces protocoles se basent sur les automates à états finis et sur les réseaux de Pétri.

5.5.5. Langage de Communication Agent (LCA)

Un Langage de Communication Agents (LCA) fournit aux agents le moyen d'échanger des informations et des connaissances.

L'ACL (Agent Communication Language) a été créé par ARPA pour assurer l'interopérabilité entre des agents autonomes et distribués. Le ACL a trois composants : un vocabulaire, un langage de communication entre agent appelé KQML (Knowledge and Query Manipulation Language) et un langage spécifiant le contenu appelé KIF (Knowledge Interchange Format). Un message d'ACL est alors un message KQML contenant une directive de communication et un contenu sémantique dans KIF, exprimé en termes de vocabulaire.

KQML fournit la couche linguistique pour rendre la communication efficace en considérant le contexte des messages. Il a été conçu comme format de message et comme protocole qui permet l'identification, le raccordement et l'échange de l'information entre des programmes. Selon Finin [91], KQML est caractérisé par trois caractéristiques importantes :

1. Les messages KQML sont opaques au contenu qu'ils transportent, ce qui implique que les messages KQML ne communiquent qu'avec des sentences en certain langage, mais aussi une attitude ou à une intention sur le contenu.

2. Des primitifs s'appellent performatives indiquent les actions ou les opérations valides.

3. Un environnement dans lequel les agents communiquent avec KQML peut être enrichi avec un genre spécial d'agents appelés les facilitateurs.

5.5.6. KQML

Le langage KQML [92] a été proposé pour supporter la communication inter-agents. Ce langage définit un ensemble de types de messages (appelés abusivement "performatifs") et des règles qui définissent les comportements suggérés pour les agents qui reçoivent ces messages. Les types de messages de KQML sont de natures diverses: simples requêtes et assertions (ex. "ask", "tell"); instructions de routage de l'information ("forward" et "broadcast"); commandes persistantes ("subscribe", "monitor"); commandes qui permettent aux agents consommateurs de demander à des agents intermédiaires de trouver les agents fournisseurs pertinents ("advertise", "recommend", "recruit" and "broker"). Comme le font remarquer Cohen et Levesque [93], ce langage a été développé de façon ad-hoc pour les besoins des développeurs d'agents logiciels: le terme "performatif" a été utilisé pour nommer diverses commandes qui ont une certaine ressemblance avec des verbes utilisés de façon performative dans le langage naturel; l'interprétation sémantique actuelle de ces commandes n'est pas satisfaisante.

Les caractéristiques principales de KQML sont les suivantes :

- Les messages KQML sont opaques au contenu qu'ils portent. Les messages KQML ne communiquent pas simplement des phrases en un certain langage, mais communiquent plutôt une attitude au sujet du contenu (affirmation, requête, réponse, etc.).
- Les primitifs du langage s'appellent performatives (ces termes vient directement de la théorie d'acte de langage). Les performatifs définissent les actions (opérations) permises utilisées par des agents communiquant entre eux.
- KQML suppose qu'au niveau d'agent, la communication point à point.
- Un environnement dans lequel les agents communiquent avec KQML peut être enrichi avec un genre spécial d'agents appelés les facilitateurs, qui fournissent aux fonctions additionnelles d'agents la gestion de réseau, telle que l'association des adresses physiques en noms symboliques, registration des agents, etc.

KQML utilise un ensemble de types de messages standards. Un exemple est comme suit :

(ask-one

:content (PRICE IBM ?price)

:receiver stock-server

:language LPROLOG

:ontology myOntology)

Dans la terminologie de KQML, "ask-one" est un performatif. Les paramètres de performatif sont présentés par des mots-clés tels que "sender", "language", etc. Le message ci-dessus représente une question au sujet du prix d'un stock partagé d'IBM. Ontology assumé par la question est l'identification par "myOntology", le récepteur du message est "stock-server" et le langage utilisé est "LPROLOG".

Les types de messages KQML sont de diverses natures :

- Simples requêtes et assertions, tel que : ask, tell, ...
- Instructions de routage de l'information, tel que forward, broadcast, ...
- Commandes persistantes, tel que subscribe, monitor, ...
- Commandes permettent aux agents consommateurs de demander à des agents intermédiaires de trouver les agents fournisseurs pertinents, tel que advertise, recommend, ...

6. Domaines d'application des agents

Les systèmes multi-agents ont des applications dans le domaine de l'intelligence artificielle où ils permettent de réduire la complexité de la résolution d'un problème en divisant le savoir nécessaire en sous-ensembles, en associant un agent intelligent indépendant à chacun de ces sous-ensembles et en coordonnant l'activité de ces agents [58]. On parle ainsi d'intelligence artificielle distribuée.

Cette méthode s'applique, par exemple, pour la surveillance d'un processus industriel où elle met en œuvre la solution de bon sens qui consiste à coordonner plusieurs surveillants spécialisés, plutôt qu'à envisager un seul surveillant omniscient.

Les recherches fondamentales concernent la représentation de la décision des agents, ou les protocoles de communication. Elles s'appliquent principalement aux télécommunications, à l'Internet avec le commerce électronique, à des agents physiques tels les robots, à l'optimisation des systèmes de transports et à la gestion de filières. Il existe une communauté de chercheurs qui s'intéressent aux simulations de sociétés d'agents, en écologie ou en sciences sociales [94].

7. Conclusion

Depuis quelques années, les systèmes multi agents ont pris une place de plus en plus importante en informatique, que ce soit dans le domaine de l'intelligence artificielle, dans les systèmes distribués, de la robotique ou même dans la vie artificielle.

Les systèmes multi agents proposent une nouvelle technologie de construction de logiciels à partir des concepts d'agent et d'interaction, en considérant que chaque unité participante au développement peut prendre la forme d'un agent qui dispose de sa propre autonomie et de ses propres objectifs. Le génie logiciel deviendra alors « multi agent » comme il est aujourd'hui « objet ». Dans ce cadre, l'approche multi-agent en génie logiciel et précisément dans le domaine des ateliers de génie logiciel, sera traitée dans le chapitre suivant.

Chapitre 3 :

Conception SMA d'un atelier de génie Logiciel

1. Introduction

Produire et maintenir des logiciels de complexité élevée est le problème qui reste récurrent pour les projets de génie logiciel. L'application des méthodologies de développement de logiciels paraît donc nécessaire pour garantir un minimum de qualité des logiciels produits. De nombreuses méthodologies ont été développées et les méthodologies les plus complètes présentent elles mêmes une complexité certaine due au nombre élevé d'artefacts exigés aux règles de la méthodologie et à leurs processus plutôt compliqués à mettre en œuvre. Face à la complexité grandissante des problèmes à résoudre ou celle des méthodologies de développement utilisées dans les différents projets, il est certain que le développeur a besoin de plus en plus d'assistance pour atteindre des objectifs conformes aux prévisions.

Les outils CASE existant proposent une assistance à la manière de modélisation mais peu d'entre eux s'intéressent au déroulement des activités du processus. Ce déroulement est essentiellement guidé par une kyrielle de documents fournis par les auteurs de la méthodologie [9]. Il est acquis que la qualité des processus logiciels conditionne la qualité du produit à réaliser [95]. D'où l'importance de ***l'ingénierie des processus*** (Software Process Engineering), une discipline du génie logiciel dont l'objectif est de supporter les processus logiciels en fournissant les moyens de modéliser, d'analyser, d'améliorer, de mesurer et d'automatiser les activités de développement.
L'intégration d'une assistance intelligente et automatisée au processus logiciel permet de rendre l'outil CASE un participant actif en anticipant les activités à mener en parallèle avec les activités de l'utilisateur. L'assistance intelligente et active au processus logiciel était le sujet de plusieurs recherches citées précédemment mais très peu ont utilisés l'approche agents que nous croyons qu'ils sont capables de mieux gérer l'assistance d'une manière beaucoup plus efficiente. Dans ce chapitre, nous essayons de proposer une solution multi-agents. Nous commençons par introduire une présentation brève de RUP. Nous donnons ensuite une analyse du problème à résoudre en suivant la méthodologie MASE qui servira à définir une conception de la solution. La conception d'une ontologie de domaine et de tâches est nécessaire pour définir un vocabulaire commun entre les agents.

2. Choix du RUP

RUP reste de loin le processus logiciel le plus utilisé et le plus adapté pour les très grands projets. Il utilise une ***approche itérative*** et il est fondé sur les ***best practices*** (bonnes pratiques).

RUP est piloté par les cas d'utilisation. Ils sont décrits à l'aide d'une représentation spécifique à RUP, plus riche que celle contenue dans UML.

Ses forces sont essentiellement :

- Cadre générique
- Référentiel de bonnes pratiques;
- Gestion des risques dans les projets;
- Cadre propice à la réutilisation;
- Approche basée sur l'architecture;
- Traçabilité à partir des Uses Cases jusqu'au déploiement.

3. Présentation du RUP (Rational Unified Process)

Le processus unifié [34] a trois caractéristiques qui sont les suivantes:

- Conduit par les cas d'utilisation (Use-Case Driven) : Le processus emploie les Use Cases à conduire le processus de développement depuis sa création jusqu'au déploiement.
- Centré Architecture : Le processus vise à comprendre les aspects les plus significatifs statique et dynamique en termes d'architecture logicielle. L'architecture est une fonction des besoins des utilisateurs et est capturé dans le cas d'utilisation de base.
- Itératif et incrémental : Le processus reconnaît qu'il est pratique de diviser de grands projets en petits projets ou des mini-projets. Chaque mini-projet comprend une itération qui se traduit par une augmentation d'échelon. Une itération peut englober tous les workflows dans le processus. Les itérations sont prévues, avec les Use Cases.

3.1. Modèle d'un processus RUP

Un processus décrit qui fait quoi, quand et comment. Le processus unifié de rational est représenté par 5 éléments élémentaires, comme ils sont montrés par la figure 17:

Rôles: le qui

Activités: le comment

Artefacts: le quoi

Workflows: le quand

Disciplines: le conteneur des quatre types d'éléments précédents.

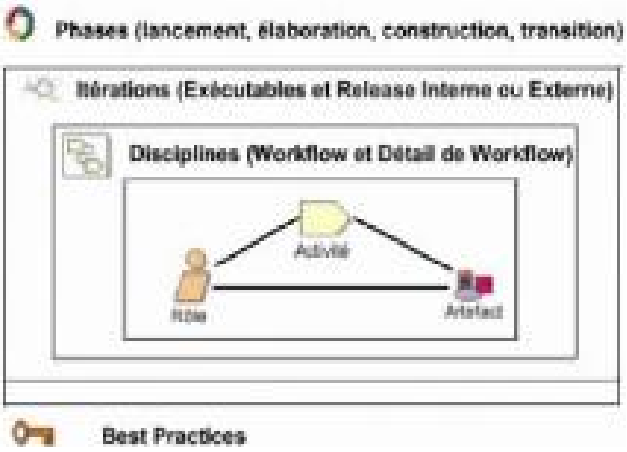

Figure 16 : La Boite Noire du RUP

3.1.1. Les rôles

C'est le concept essentiel du processus. Il définit le comportement et les responsabilités d'un individu ou d'un groupe d'individus travaillants en équipe. Le comportement est exprimé en termes d'activités que le rôle performe. Les responsabilités sont exprimées en relation d'artefacts que le rôle crées, modifies ou contrôle. La figure 17 montre un exemple de rôle dans RUP.

3.1.2. Les activités

Une activité est l'unité de travail qu'un rôle performe et qui produit un résultat significatif dans le contexte du projet. L'activité a un but clair, habituellement exprimé en termes de créer ou mettre à jour des artefacts, tels qu'un modèle, une classe, ou un plan.

3.1.3. Les artefacts (Work products)

Sont utilisés comme entrée par les rôles pour accomplir une activité, et comme résultats pour ces activités.

Les Artefacts sont de plusieurs types :

- Un modèle, comme le modèle des use-case ou le modèle conceptuel.
- Un élément modèle—un élément dans le modèle—comme une classe, un use case, ou un sous système.
- Un document, comme un business case ou software architecture document.
- Un code Source
- Des exécutables

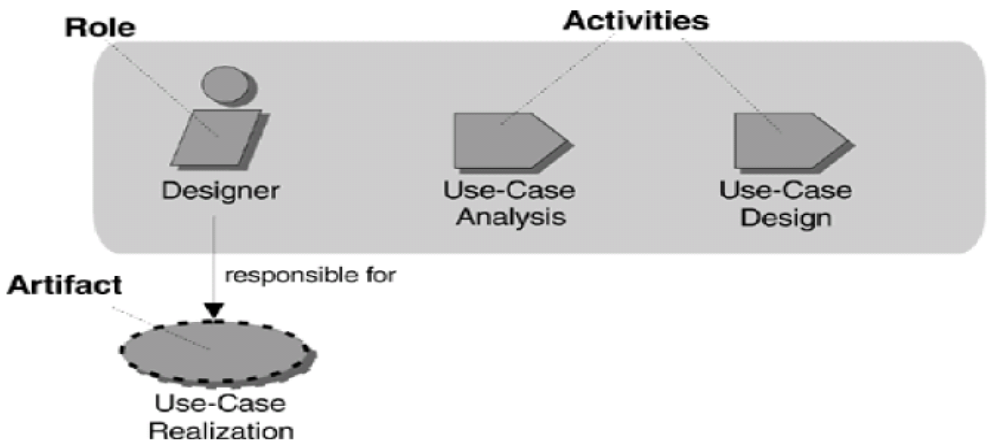

Figure 17 : Exemples Rôle, activités, et artefact

3.1.4. Les workflows

C'est une séquence d'activités qui produit un résultat d'une valeur observable. En termes d'UML, il peut être exprimé comme un diagramme de séquence, de collaboration ou d'activité.

Rational Unified Process utilise trois types de workflow:

- Le Noyau de workflows (Core workflows), associé à chaque discipline. Dans chaque discipline il existe un « core workflow » qui donne globalement le flux des activités.
- Les détails de Workflow (Workflow détails), pour affiner le déroulement du core workflow. Chaque « core workflow » couvre beaucoup d'activités, on utilise « Workflow details » pour exprimer un groupe spécifique d'activités étroitement liées. Par exemple, les activités sont exécutées ensemble ou d'une manière cyclique ; elles sont exécutées par un groupe de personne travaillant ensemble sur un même projet. « Workflow details » présente aussi les flux d'informations – les artefacts qui sont à l'entrée et à la sortie des activités- pour montrer comment les activités interagissent à travers les différents artefacts.
- Les plans d'itération (Iteration plans) qui sont d'autres moyens pour présenter le processus en le décrivant d'avantage dans la perspective de ce qui se produit dans une itération typique. Ils sont réellement les plus proches de ce qu'un moteur de workflow manipulerait. On peut les considérer comme des instanciations du processus pour une itération donnée, choisissant les activités qui seront effectivement gérées pendant l'itération et les répliquées selon les besoins.

3.1.5. Les disciplines

Les disciplines sont des conteneurs utilisés pour organiser des activités du processus. Il y a neufs disciplines dans le processus RUP, et ils représentent une division de tous les rôles et activités dans des groupements logiques par des sujets de préoccupation ou la

spécialité. Les neuf disciplines de noyau sont divisées en six disciplines techniques et trois disciplines de support. Six disciplines techniques "***engineering disciplines***" (Modélisation d'activités « Business Modeling », Expression des besoins, Analyse et conception, Implémentation, Test et Déploiement) et trois Disciplines de support ***« supporting disciplines »*** (Gestion de configuration, Gestion de projet et Environnement).

3.1.6. Des éléments additionnels du processus

Rôles, activités, workflows et artefacts, organisés dans les disciplines, représentent l'épine dorsale (the backbone) de la structure statique du processus RUP. Mais d'autres éléments sont ajoutés à des activités ou des artefacts pour rendre le processus plus facile à comprendre, à utiliser et à fournir des conseils plus complets pour les praticiens. Ces éléments de processus supplémentaires sont : les directives (Guidelines), les modèles (Templates) les experts d'outils (Tool mentors) et les concepts.

3.2. Le cycle de vie du RUP

Un cycle est constitué de quatre phases, comme le montre la figure 18, exécutées dans l'ordre : création, élaboration, construction et transition.

La phase de création (Inception phase) consiste à définir l'étendue du projet en spécifiant les cas d'utilisation (périmètre et objectif du projet), et déterminer si la poursuite du projet est possible. Elle se conclut par le jalon LCO (Life cycle Objectif).

La phase d'élaboration se conclut par le jalon LCA (Life Cycle Architecture). Elle a comme objectif de définir l'architecture du système et l'établir comme référentiel des développements à venir, et aussi d'avoir détaillé 80% des exigences.

La phase de construction est concentrée sur la conception détaillée, l'implémentation et les tests. Elle se termine par le jalon IOC (Initial Operation Capability : 1ère version opérationnelle).

Lorsqu'on finit une itération et que certains éléments sont validés on peut parfois passer à la phase suivante. Rappelons que pour chaque phase on exécute des itérations, et pour chaque itération on exécute l'ensemble des disciplines.

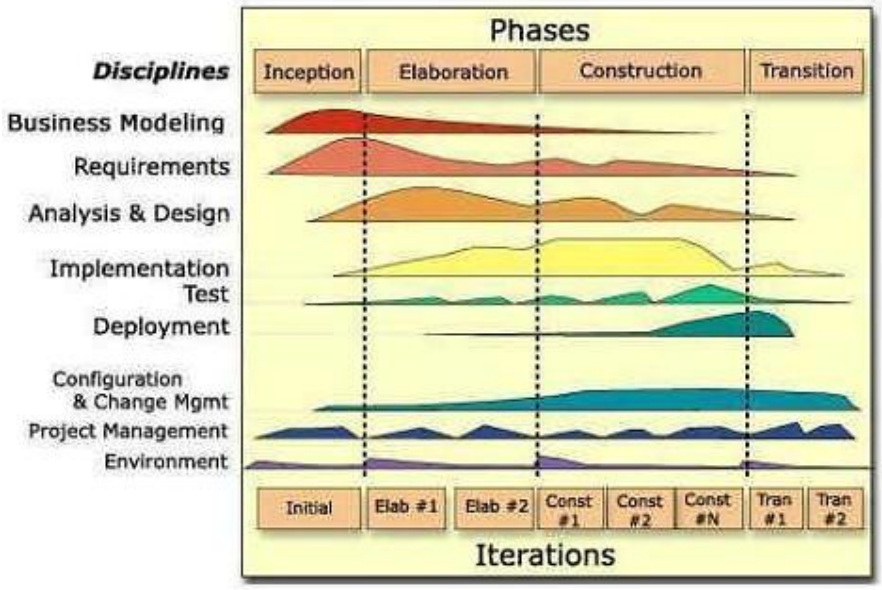

Figure 18 : Le cycle de vie du RUP

4. L'assistance au processus logiciel

Un modèle d'assistance au processus logiciel a été défini par N. Boujlida [96] puis a été amélioré par celui-ci dans le projet ALF. Il propose des fonctionnalités de travail du système qui sont : réaliser une tâche, observer et contrôler (monitoring) et des fonctionnalités d'assistance au développeur qui sont : guider, prendre une initiative, expliquer et apprendre.

Contrôler : L'assistant doit protéger le développeur de ses initiatives qui viennent tôt en respectant l'état du logiciel (l'utilisateur doit respecter le processus de développement du logiciel). D'un autre côté, il doit le protéger d'aller au-delà de ses droits. Par exemple, quand un utilisateur adopte un rôle, il doit respecter ce rôle jusqu'à ce qu'il le change explicitement ; vérifier si une invocation est conforme à l'ordre ; si ses pré-conditions sont vrais ou non…etc.

Guider : Le système doit être capable d'informer le développeur de l'ensemble des opérations pouvant être performées. La fonctionnalité de guidance implique principalement trois facilités: Que faire ensuite ?, comment cela fonctionne ? Et comment le faire ?

Expliquer : Quand une opération ne peut pas être performée ou complétée, il faut donner à l'utilisateur des explications et des conseils.

Prendre des initiatives: Une dimension de l'assistance est l'initiative. Les initiatives à prendre sont essentiellement basées sur les règles ; la partie condition spécifie les circonstances dans lesquelles l'initiative spécifiée dans la partie action est à prendre. D'après Nitscuke [97] une assistance peut être active ou passive. Une autre dimension

concerne le degré d'automatisation ; un système d'assistance peut agir automatiquement et exécuter des tâches pour l'utilisateur, proposer les prochaines actions possibles et fournir des informations. Une autre dimension est liée au fait que le système d'assistance doit être adaptatif ou adaptable, il doit pouvoir s'adapter lui même aux besoins de l'utilisateur et l'utilisateur doit pouvoir l'adapter.

Selon Lonchamp [50], l'assistance au développeur peut être catégorisée en deux façons :

La façon de modéliser (A way of modeling) où on dispose d'un ensemble de concepts de modélisation fondamentaux pour capturer la connaissance sémantique sur un problème et sa solution avec un ensemble de vues et de notations pour présenter cette information.

La façon de travailler à travers le processus (A way of working) où le processus décrit le modèle à construire, quand et comment le construire avec un ensemble de règles pour prendre des décisions de modélisation.

Le premier type d'assistance est assez répandu avec les outils CASE UML, le second est beaucoup moins répandu et fera l'objet de l'assistance proposée par la suite.

5. L'approche proposée

Les outils CASE sont actifs au niveau de la méthode (ils peuvent vérifier la cohérence des modèles de logiciels et vérifier les règles inter modèle), mais ils sont plutôt passifs aux niveaux du processus logiciel et du développeur. Pour fournir une assistance active pour les développeurs, un outil CASE de modélisation doit avoir une connaissance intégrée des méthodes de logiciel, du processus logiciel, du domaine d'application et du comportement du développeur. Un outil doit intégrer et utiliser ces niveaux multiples de connaissance afin qu'il puisse s'adapter au développement de la tâche en question et aux attentes du développeur. Un outil de modélisation actif doit être doté de moyens d'auto-apprentissage et d'auto-amélioration et devrait être en mesure de travailler et raisonner avec une information incomplète [98].

L'objet de ce travail est de proposer une architecture qui va rendre un AGL plus actif et plus flexible. Cette activité est concrétisée par la réalisation d'une assistance intelligente qui va lier les outils avec la connaissance. Pour atteindre cet objectif, on introduit le concept d'agents où le système à proposer va être considéré comme un ensemble d'agents.

Chaque agent va avoir un objectif local dans la modélisation et l'ensemble des agents collaborent pour réaliser les activités du processus logiciel.

Les systèmes multi agents s'adaptent bien à la conception de notre outil à cause des caractéristiques suivantes :

- Les ateliers de génie logiciel sont complexes ;
- La distribution des données et du contrôle s'impose.

Notre système va être développé en suivant la méthodologie MASE. La méthodologie MaSE (Multi-agent Systems Engineering) utilise l'abstraction fournie par les systèmes multi agents pour aider les concepteurs à développer des logiciels intelligents et répartis. MaSE considère les agents comme étant une abstraction du paradigme orienté objet et donc une spécialisation des objets [99]. Cette méthodologie comporte sept phases : trouver les buts, appliquer les cas d'utilisation, raffiner les rôles, créer les classes d'agents, construire les conversations, assembler les classes d'agents ainsi que l'implémentation. MaSE est l'une des méthodologies les plus complètes et les plus simples à déployer qui s'appuie sur l'analyse et le développement. C'est une méthodologie itérative dont l'objectif est de permettre aux concepteurs de se déplacer, en toute liberté, entre les différentes phases pour apporter plus de détails et éventuellement combler les lacunes [100].

La conception d'un SMA commence par une première étape qui permet l'identification de buts et de sous buts pour aboutir à un diagramme d'hiérarchie de buts.

5.1. L'identification de buts et le diagramme d'hiérarchie de buts

L'objectif principal de notre travail est d'assister le développeur pendant la modélisation avec RUP. Assister est synonyme de guider, fournir des explications et contrôler les intervenants dans un projet au déroulement. On peut identifier les autres buts importants comme suit:

Assister le développeur dans :

1. La réalisation d'un projet
    i. La réalisation des différentes phases
    ii. La réalisation des différentes disciplines
    iii. La réalisation des différentes itérations d'un projet
    iv. La réalisation des différents groupes d'activités d'une itération
        1. L'assistance d'un rôle
            a. La réalisation des différentes activités d'une itération
                i. La réalisation des différentes étapes d'une activité

ii. La production, la consultation ou la mise à jour d'un artefact

Et éventuellement, assister le développeur (ou une équipe de développeurs) dans ces différentes tâches et prendre des décisions de modélisation, comme l'anticipation de quelques étapes d'une activité…etc. Le diagramme suivant (Figure 19) montre une hiérarchie possible des buts.

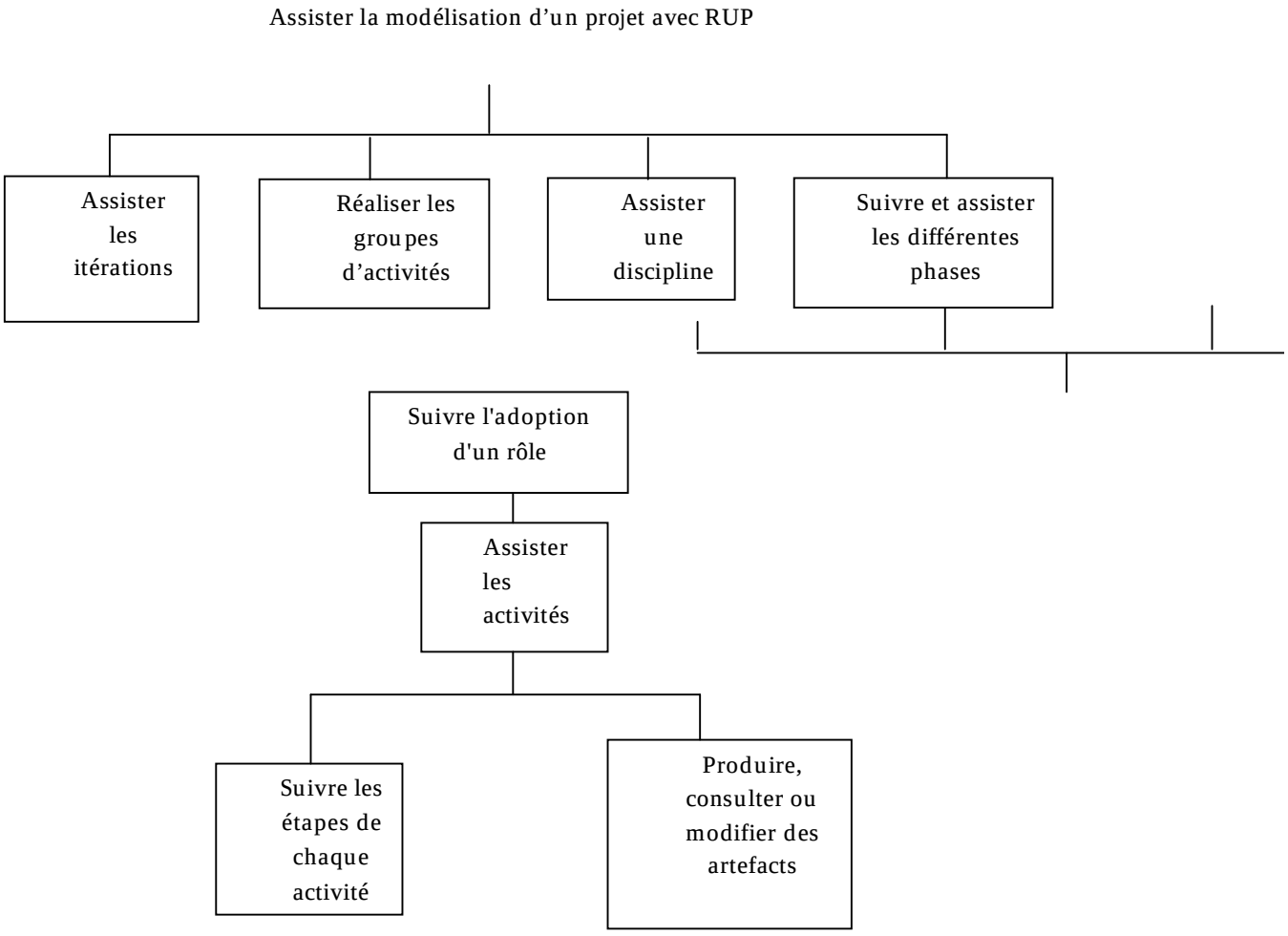

Figure 19 : Diagramme d'hiérarchie de buts

5.2. Les cas d'utilisation

Pour identifier les fonctionnalités du système à concevoir, on utilise le diagramme des cas d'utilisation. Le système n'a qu'un seul acteur externe qui est le développeur. Ce dernier, pour réaliser sa modélisation, il interagit avec le système en utilisant les cas d'utilisation montrés dans la figure 20.

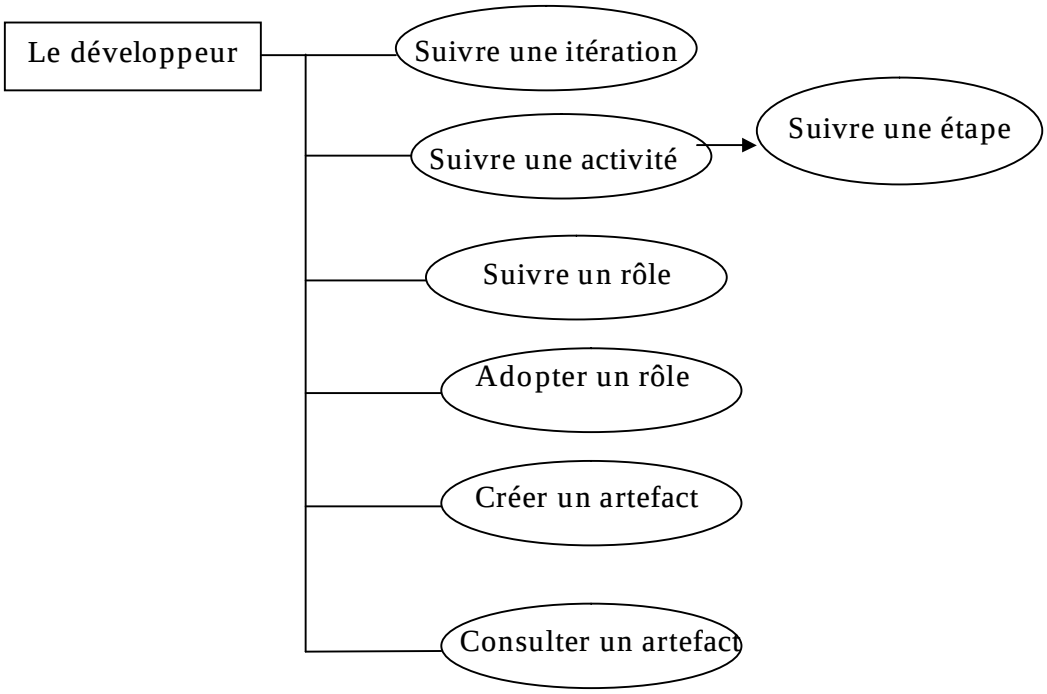

Figure 20 : Diagramme de cas d'utilisation

5.3. Le diagramme de séquence

Les Cas d'utilisation sont structurés dans un diagramme de séquence comme il est montré dans la figure 21, pour aider la détermination des communications demandées dans le système multi agents. Le digramme de séquence définit la communication minimale qui doit avoir lieu entre les rôles du système.

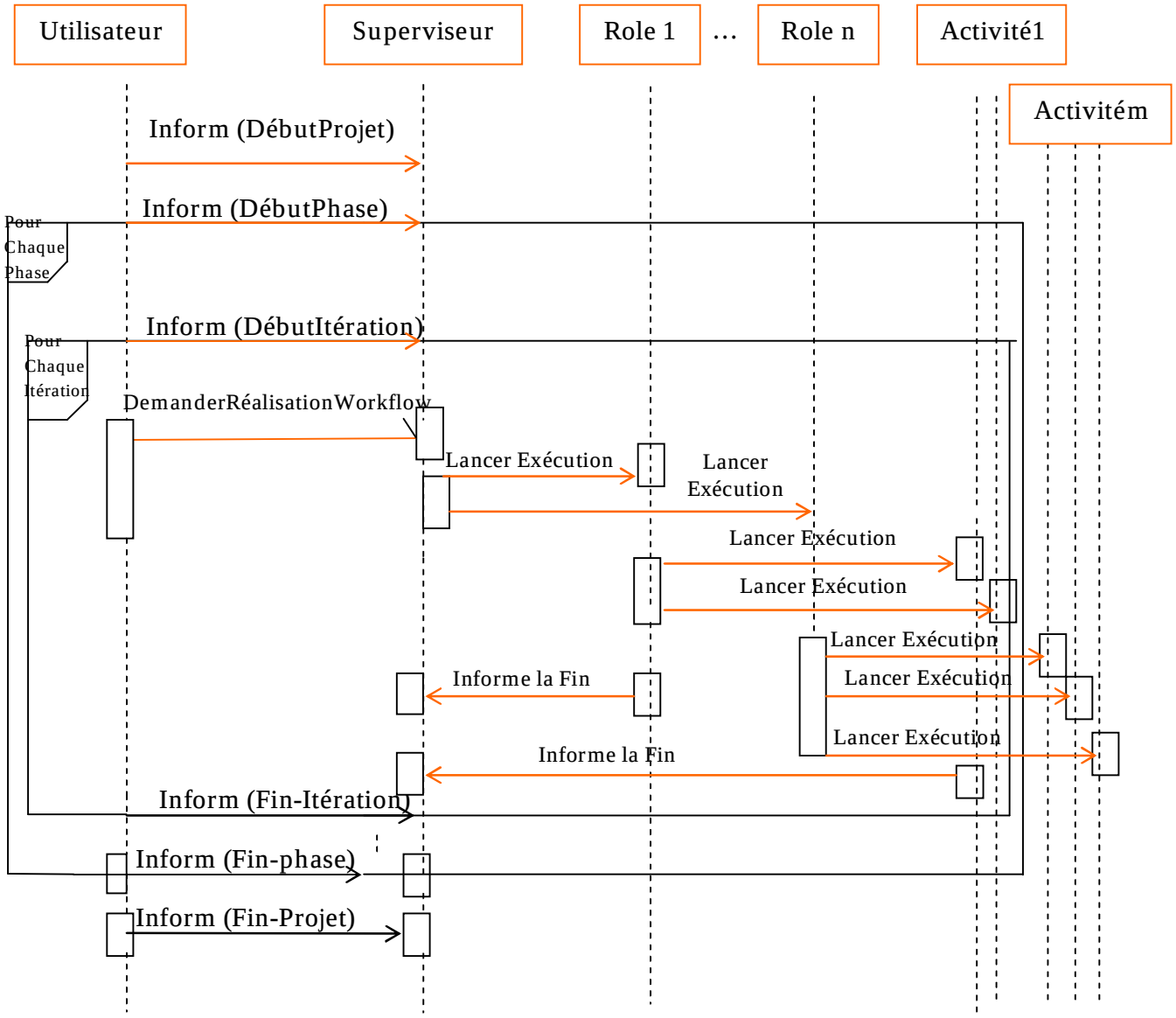

Figure 21 : Le diagramme de séquence

5.4. Les rôles du système

Le but de la phase d'analyse de MaSE est de définir un ensemble de rôles qui peuvent être employés pour réaliser les buts du système. Ces rôles sont définis explicitement par l'intermédiaire d'un ensemble de tâches. MaSE est construite sur la prétention que les buts du système seront satisfaits si chaque but suit un rôle, et chaque rôle est joué par au moins une classe d'agent. Le concepteur peut choisir de permettre à un rôle d'être responsable de plusieurs buts suivant le critère efficacité.

A partir des cas d'utilisation et les buts du système on peut dégager les rôles suivants :

L'Interface du développeur : Gérer les interactions du développeur avec le système.

L'assistant Activité : Réalise les différentes étapes d'une activité, crée, modifie ou consulte un artefact. A la fin il uniformise les résultats.

L'assistant Rôle : Assistant du développeur pendant l'adoption d'un rôle. Le développeur adopte un rôle, responsable de la réalisation d'un ensemble d'activité qui produit chacune un artefact. Il combine à la fin les résultats.

Le superviseur : Contrôle l'exécution des itérations, des workflows, des rôles et des activités et fournit des éléments d'assistance au développeur.

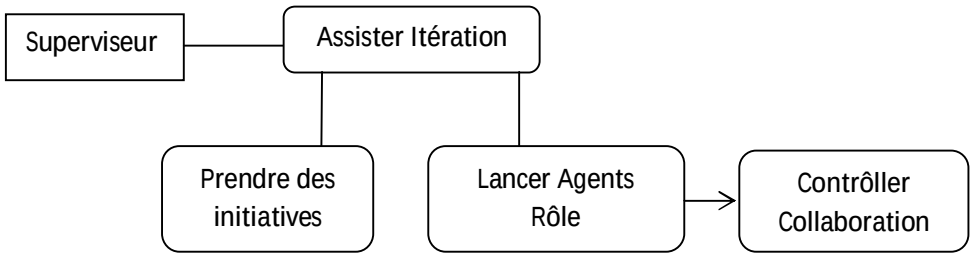

Figure 22 : Le diagramme de rôle de l'agent Superviseur

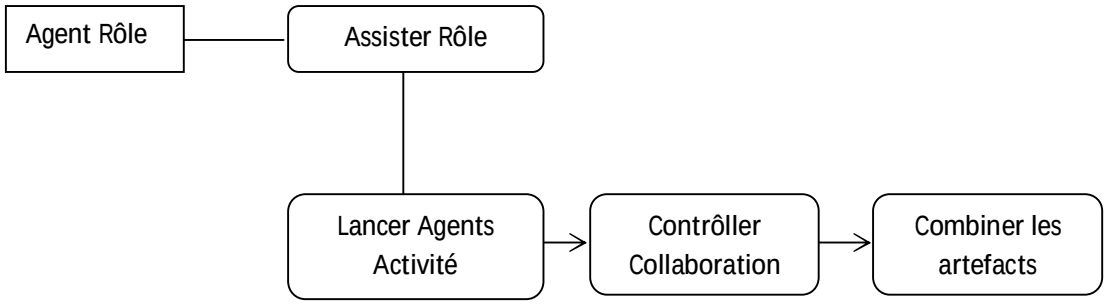

Figure 23 : Le diagramme de rôle de l'agent Rôle

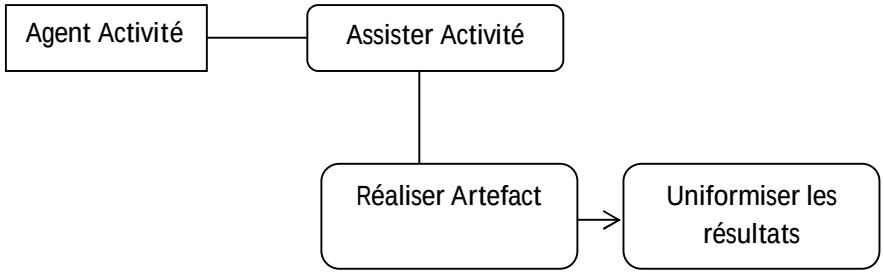

Figure 24 : Le diagramme de rôle de l'agent Superviseur

Ainsi, le diagramme global de relation entre les rôles est le suivant (La figure 25) :

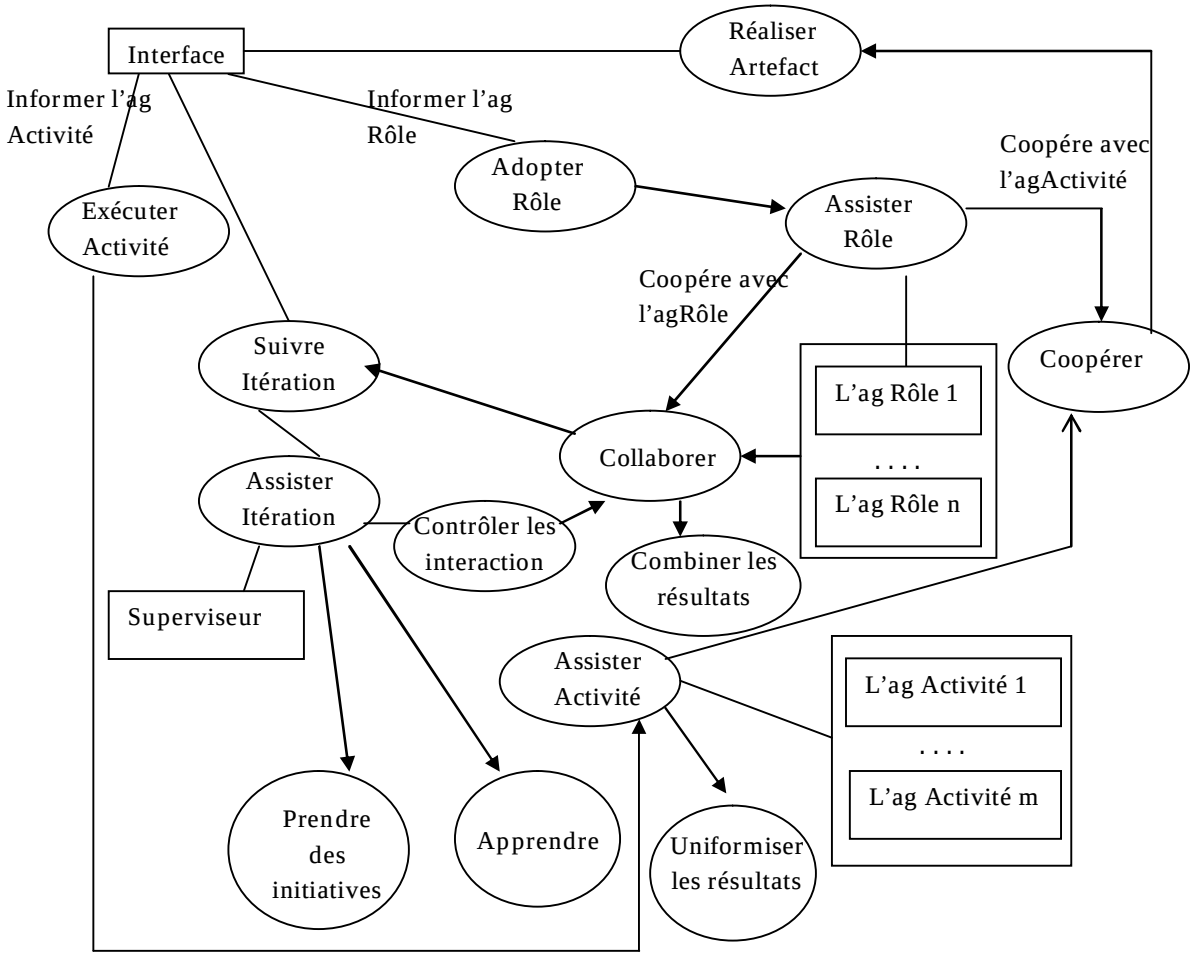

Figure 25 : Le diagramme de rôle

5.5. Le modèle d'ontologie

Gruber [101] a défini l'ontologie par :

«***Une ontologie est une spécification formelle et explicite d'une conceptualisation d'un domaine de connaissance***».

La ***conceptualisation*** renvoie à un modèle abstrait d'un quelconque phénomène après en avoir relevé les concepts significatifs et les relations existantes entre ces concepts. Par ***explicite***, il faut entendre que le type de concepts utilisés, ainsi que leurs contraintes

d'utilisation, sont définies de façon explicite ; quant à l'objectif ***formel***, il exprime le fait que l'ontologie doit être lisible par ordinateur.

C'est une base de formalisation des connaissances. Elle se situe à un certain niveau d'abstraction et dans un contexte particulier. C'est aussi une représentation d'une conceptualisation partagée et consensuelle, dans un domaine particulier et vers un objectif commun. Elle classifie en catégories les relations entre les concepts.

Une ontologie est utile pour :

1. Capturer, modéliser, transformer et intégrer une connaissance dans un contexte particulier.
2. Traiter et automatiser les raisonnements sur cette connaissance.
3. Partager en collaboration une compréhension commune d'une structure d'informations parmi les agents humains et / ou parmi les agents logiciels.
4. Etre capable de réutiliser le domaine d'intérêt par des connaissances opérationnelles ad hoc ou autres standards qui unifient les mécanismes de modélisation des catégories ontologiques de connaissances sur l'existence.

La perception du système demande deux ontologies :

1. Une ontologie de domaine pour représenter les concepts de RUP et les liens entre eux afin de créer un vocabulaire commun entre les agents.
2. Une ontologie de tâches représentant les tâches de RUP utilisées par les agents Superviseur et AgRôle, et les liens entre elles.

Dans ce qui suit, on donne une définition des deux types d'ontologies.

Ontologie de Tâches : Ce type d'ontologies est utilisé pour conceptualiser des tâches spécifiques dans les systèmes, telles que les tâches de diagnostic, de planification, de conception, de configuration ; soit tout ce qui concerne la résolution du problème. Elle régit un ensemble de vocabulaire et de concepts qui décrit une structure de résolution des problèmes inhérente aux tâches et indépendante du domaine.

Ontologie du domaine : Cette ontologie régit un ensemble de vocabulaire et de concepts qui décrit un domaine d'application ou monde cible. Elle permet de créer des modèles d'objets du monde cible. L'ontologie de domaine est une méta-description d'une représentation des connaissances, c'est-à-dire, une sorte de méta-modèle de connaissance dont les concepts et les propriétés sont du type déclaratif.

Dans le cas de notre application, la formalisation des concepts suivra la spécification SPEM (Software Process Engineering Metamodel) de l'Object Management Group OMG. L'OMG propose un méta modèle pour la description des processus d'ingénierie logicielle SPEM. Ce méta modèle est un stéréotype d'UML qui utilise le formalisme UML OCL, d'où son adoption pour la représentation des ontologies. Une brève définition de ***SPEM*** est donnée par la suite.

5.5.1. SPEM

5.5.1.1. Le méta modèle SPEM

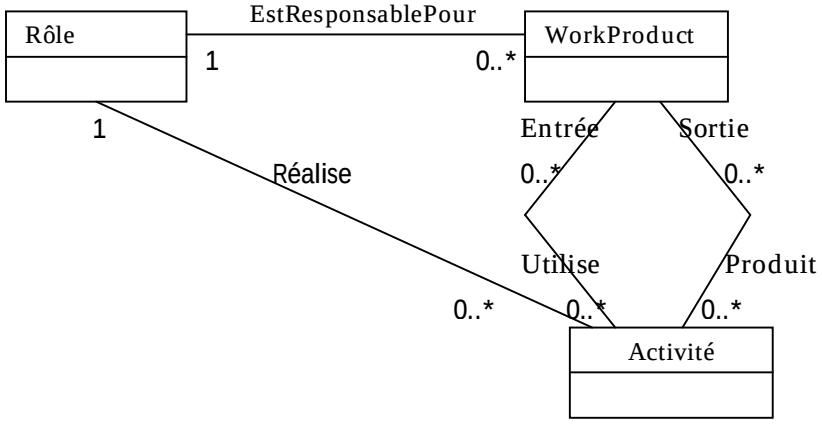

Figure 26 : Les éléments essentiels du SPEM et leurs relations [102]

L'idée principale de SPEM est de considérer un processus de développement logiciel comme une collaboration entre des entités abstraites, les rôles qui réalisent des opérations appelées activités sur des entités tangibles appelées artefacts. La figure 23 présente les liens entre ces trois concepts. Plusieurs rôles peuvent collaborer par l'échange de produits et le déclenchement de l'exécution de certaines activités. Le but global d'un processus est de fournir un ensemble d'artefacts dans un état bien défini.

5.5.1.2. SPEM comme profil UML

Un profil UML est une adaptation d'UML qui utilise les mécanismes d'extension offerts par UML de façon standardisée pour un but ou un domaine particulier. Un profil UML fournit des stéréotypes pour les entités caractérisant le domaine pour lequel il est défini. SPEM est défini à la fois comme un méta modèle de processus et comme un profil UML pour le domaine que constitue l'ingénierie des processus de développement de logiciels. Ce qui permet aux développeurs de tels processus d'utiliser UML comme notation. La figure 24 donne la façon de décrire les éléments d'un processus comme des stéréotypes UML.

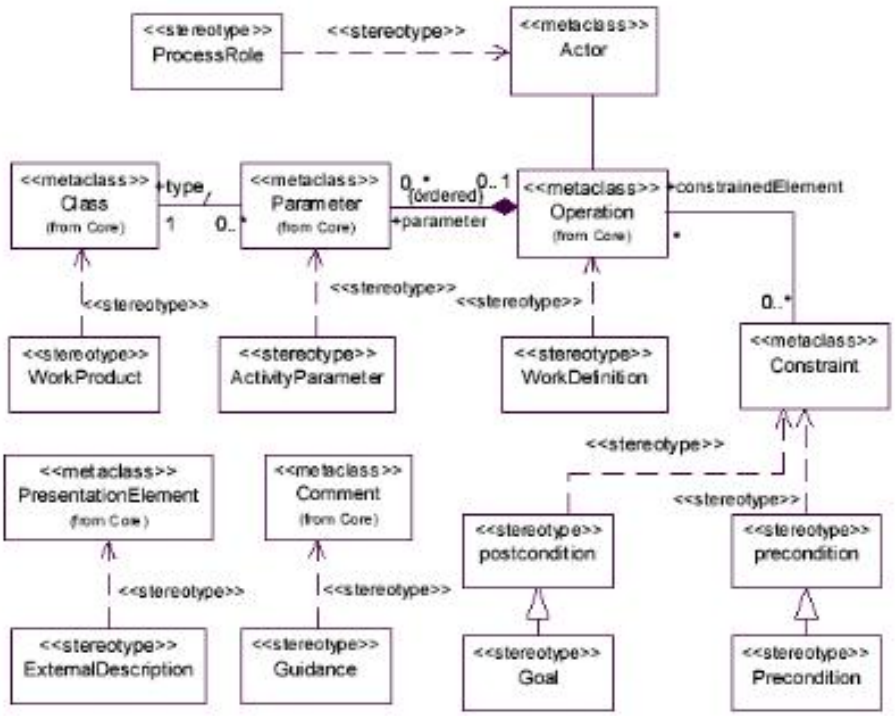

Figure 27 : Correspondance entre SPEM et les éléments du méta modèle d'UML [103]

Une telle correspondance permet de définir un processus de développement logiciel en UML en faisant usage à des stéréotypes propres à SPEM, ceci en représentant les éléments stéréotypés propres au processus par les éléments d'UML proposés par la figure de correspondance, c'est-à-dire les classes et les diagrammes d'activités.

5.5.2. Ontologie de domaine

L'ontologie de domaine représente le vocabulaire commun utilisé par les agents pour exprimer les concepts du processus, les relations entre eux ainsi que leurs formalisations.

5.5.2.1. Les concepts

Activité : Une activité est l'unité de travail qu'un rôle performe et qui produit un résultat significatif dans le contexte du projet.

Etape : il y a trois types d'étape d'une activité :

- Etapes de réflexion où le développeur réalisant le rôle comprend la nature de la tâche, rassemble et examine les artefacts en entrée, et formule les sorties.
- Etapes de réalisation où la personne réalisant le rôle crée ou met à jour plusieurs artefacts
- Etapes de revue où l individu réalisant le rôle inspecte les résultats au moyen d'un certain nombre de critères

Guide de travail : Les activités peuvent être associées à des guides de travail. Ceux ci présentent des techniques et des conseils pratiques qui aident le rôle réalisant l'activité

Guide outil : Les activités, les étapes ainsi que les guides associés fournissent un guidage général. Pour aller un peu plus loin dans les étapes, les Guides d'outils sont des éléments additionnels qui fournissent le guidage en montrant comment réaliser les étapes en utilisant des outils spécifiques.

Artefacts (Work products) : sont utilisés comme entrée par les rôles pour accomplir une activité et comme résultats pour ces activités.

Les Artefacts sont de plusieurs catégories:

Un modèle, comme le modèle des use-case ou le modèle conceptuel.

Un élément de modèle, comme une classe, un use case ou un sous système.

Un document, comme un business case ou un document d'architecture.

Le code source et les exécutables, comme une sorte de composants.

Les exécutables, en tant que produit logiciel.

Dans la catégorie document on distingue trois éléments :

- Guide d'artefact et Point de contrôle : Les artefacts ont des guides et des points de vérification associés qui présentent les informations sur leur développement, leur évaluation et leur utilisation. Les guides d'artefacts capturent l'essence même du travail réalisé, tandis que la description des activités capture l'essence de ce qui doit être fait. Les points de vérification fournissent une référence rapide qui aide à estimer la qualité de l'artefact.

- Plan type : Les plans types sont des modèles ou prototypes d'artefacts. Plusieurs plans types sont associés à la description d'un artefact et peuvent être utilisés pour créer les artefacts correspondants. Les plans types sont liés aux outils à utiliser.

- Rapport : Les modèles et éléments de modèle peuvent avoir des rapports associés. Ces rapports extraient les informations sur le modèle et les éléments de modèle d'un outil.

Rôle : définit le comportement et les responsabilités d'un individu ou d'un groupe d'individus travaillants en équipe. Le comportement est exprimé en termes d'activités que le rôle performe. Les responsabilités sont exprimées en relation d'artefacts que le rôle crées, modifies ou contrôle.

Workflow : C'est une séquence d'activités qui produit un résultat d'une valeur observable. En termes d'UML, il peut être exprimé comme un diagramme de séquence, de collaboration ou d'activité.

Il y a trois types de workflow:

- Core workflows : associé à chaque discipline.
- Détails de Workflow : pour raffiner le déroulement du core workflow. Le Core workflow est représenté généralement sous forme de diagramme d'activités. Ce diagramme d'activité comporte les détails de workflow comme les groupes d'activités et les liens entre ceux-ci.
- Itération plans : autres moyens de présenter le processus, le décrivant davantage dans la perspective de ce qui se produit dans une itération typique. On peut les voir comme des instanciations du processus pour une itération donnée, choisissant les activités qui seront efficacement gérées pendant l'itération et les répliquées selon les besoins.

Discipline : Une division de tous les rôles et activités dans des groupements logiques par des sujets de préoccupation ou de spécialité. Une discipline est une collection d'activités liées par un intérêt majeur pendant tout le projet. Le regroupement des activités dans une discipline permet de mieux appréhender le projet du point de vue de l'approche linéaire.

Dans chaque itération, les tâches sont organisées en neuf disciplines:

- Six disciplines techniques "***engineering disciplines***" qui sont la modélisation d'activités (Business Modeling), l'expression des besoins, l'analyse et la conception, l'implémentation, le test et déploiement)
- Et trois Disciplines de support ***« supporting disciplines »***qui sont la gestion de configuration, la gestion du projet et l'environnement.

Phase : C'est le temps qui sépare deux jalons du projet pendant lequel un groupe d'objectifs bien définis sont réalisés, les artefacts complétés et les décisions d'aller ou non à la prochaine phase sont prises. On distingue quatre phases : Phase d'inception, Phase d'élaboration, Phase de construction et Phase de transition.

Jalon : Point d'arrêt formel d'une phase, il correspond à un point de version.

Jalon mineur : point d'arrêt formel d'une itération, il correspond à un point de version.

5.5.2.2. Relation entre concepts et diagramme des relations binaires

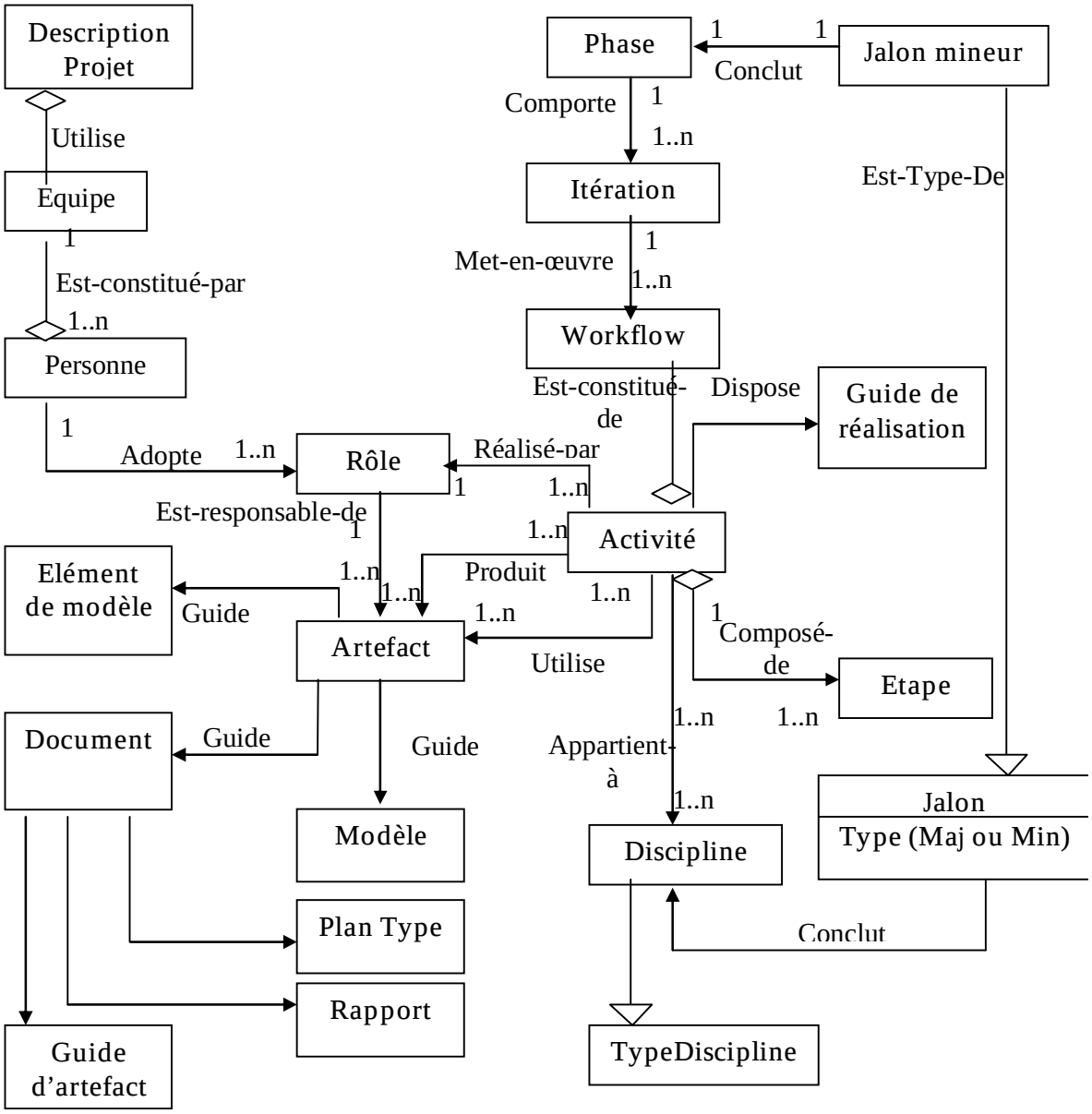

Figure 28 : Diagramme des relations binaires

Le modèle de la figure 25 présente les différents concepts du processus générique RUP. Un projet utilise une équipe qui est constituée de personnes et celles ci adoptent un ou plusieurs rôles pour la réalisation des objectifs du projet. Parallèlement, un projet utilise aussi le processus générique RUP. Ce processus est découpé en phases qui représentent sa structure statique et qui se terminent chacune par un jalon. Chaque phase comporte une ou plusieurs itérations qui sont conclues chacune par un jalon mineur. Chaque itération met en œuvre un workflow pour son achèvement. Ces workflows, parties intégrantes de la dynamique du processus, sont constituées de plusieurs activités appartenant à l'une des neuf disciplines du processus. Et chaque activité est réalisée par un rôle, elle prend en

entrée des artefacts et en produit en sortie. Chaque rôle qu'un développeur adopte est responsable d'un ou de plusieurs artefacts. Chaque activité se décompose atomiquement en étapes et dispose d'un guide de réalisation. Les artefacts peuvent être des guides des modèles, des éléments de modèle ou encore des documents.

5.5.3. Ontologie de tâches

L'ontologie de tâches représente les tâches et les liens entre les tâches pour les différentes disciplines de RUP. Elle représente le vocabulaire commun des agents Superviseur, Interface et les agents Rôle qui seront définis par la suite. Une activité est liée au rôle qui la réalise et aux artefacts produits ou modifiés. Par conséquent, une tâche sera définie par le triplet : rôle, activité et artefact. En RUP, les workflows sont exécutés en détails de workflows (sorte de workflow, workflow detail) ; ce sont des groupes d'activités donnant des résultats observables et représentant chacun un groupe d'activités suivies par des groupes de rôles.

Nous commençons par identifier et définir les groupes d'activités puis les activités en les regroupant par discipline. Ensuite nous proposons un modèle approprié et les contraintes associées. La relation entre les groupes d'activités dans chaque discipline est représentée par un diagramme d'activité UML.

5.5.3.1. Conceptualisation

5.5.3.1.1. L'expression de besoins

a. Groupes d'activités (Workflow details)

Analyser le problème : Ce groupe d'activités consiste à capturer le vocabulaire courant, développer la vision, rechercher les acteurs et les cas d'utilisation et développer un plan de gestion des besoins.

Comprendre les besoins du partenaire : Ce groupe d'activités consiste à capturer le vocabulaire courant, développer la vision, expliciter les requêtes du partenaire, rechercher les acteurs et les cas d'utilisation, gérer les dépendances et réexaminer les changements de besoins.

Gérer les changements de besoins : Ce groupe d'activités consiste à gérer les dépendances, réexaminer les changements de besoins, réexaminer les besoins, structurer le modèle des cas d'utilisation et réexaminer le rapport.

Définir le système : Ce groupe d'activités consiste à développer la vision, capturer le vocabulaire courant, rechercher les acteurs et les cas d'utilisation et gérer les dépendances.

Gérer la portée du système : Consiste à développer la vision, gérer les dépendances, prioriser les cas d'utilisation et réexaminer les changements de besoins.

Affiner la définition du système : Ce groupe d'activités consiste à détailler les cas d'utilisation, détailler les exigences du logiciel, modéliser et prototyper l'interface utilisateur.

b. Activités

Développer le plan de gestion des exigences : Pour développer un plan de documentation des exigences, leurs attributs et les guides pour la traçabilité et la gestion des exigences du produit.

Développer la vision : S'entendre sur les problèmes à résoudre, identifier les partenaires du système, définir les limites du système, décrire les premières caractéristiques du système.

Expliciter les requêtes avec le partenaire : Pour savoir qui sont les partenaires du projet, pour collecter les requêtes sur les exigences à remplir par le système, pour accorder des priorités aux requêtes des partenaires.

Capturer le vocabulaire courant : Pour définir le vocabulaire commun qui peut être utilisé dans toutes les descriptions textuelles du système, spécialement dans la description des cas d'utilisation.

Rechercher les acteurs et les cas d'utilisation : Pour mettre en relief les fonctionnalités du système. Pour définir ce qui sera traité par le système et ce qui sera traité à l'extérieur du système. Pour définir qui et quoi pourra interagir avec le système. Diviser le modèle en packages avec les acteurs et les cas d'utilisation, créer les diagrammes du modèle de cas d'utilisation et développer un aperçu du modèle de cas d'utilisation.

Gérer les dépendances : Pour utiliser les attributs et la traçabilité des exigences du projet pour assister dans la gestion des limites du projet et la gestion des changements des exigences.

Structurer le modèle de cas d'utilisation : Pour extraire le comportement dans le cas d'utilisation qui nécessite d'être considéré comme un cas d'utilisation abstrait ; comme exemple de ce comportement, on cite le comportement courant, le comportement optionnel et le comportement qui sera développé dans les dernières itérations. Pour rechercher de nouveaux acteurs abstraits qui définissent les rôles à partager par plusieurs acteurs.

Accorder des priorités aux cas d'utilisation : Pour définir les entrées à la sélection constituée de l'ensemble des scénarios et les cas d'utilisation qui sont à analyser dans l'itération courante, Pour définir l'ensemble des scénarios et cas d'utilisation qui représentent des fonctionnalités significatives et centrales, Pour définir l'ensemble des scénarios et cas d'utilisation qui couvrent substantiellement l'architecture qui mettent en

jeu plusieurs éléments architecturaux ou qui illustrent un point délicat et spécifique de l'architecture.

Détailler un cas d'utilisation : Pour décrire le flot d'évènements des cas d'utilisation en détail. Pour décrire le flot d'évènements des cas d'utilisation de manière à faciliter la compréhension par les clients et les utilisateurs.

Détailler les exigences logicielles : Pour collecter, détailler et organiser l'ensemble du package d'artefacts qui décrivent complètement les exigences logicielles du système ou du sous système.

Modéliser l'interface utilisateur : Pour construire un modèle de l'interface utilisateur sur lequel on peut discuter, qu'on peut améliorer et qui est aisé à l'utilisation.

Prototyper l'interface utilisateur : Pour créer un prototype d'interface utilisateur.

Réexaminer les exigences : Pour vérifier formellement que les résultats des exigences satisfont le point de vue du client du système.

5.5.3.1.2. L'analyse et la conception

a. Groupes d'activités

Définir une architecture candidate : Ce groupe d'activités consiste à analyser l'architecture et les cas d'utilisation.

Réaliser la synthèse architecturale : Ce groupe d'activités consiste à analyser l'architecture, construire l'architecture preuve de concept et estimer sa viabilité.

Analyser le comportement : Ce groupe d'activités consiste à analyser les cas d'utilisation, identifier les éléments de conception et réexaminer la conception.

Affiner l'architecture : Ce groupe d'activités consiste à prioriser les cas d'utilisation, décrire l'architecture d'exécution, décrire la distribution, identifier les mécanismes de conception, réexaminer l'architecture et structurer le modèle d'implémentation.

Concevoir les composants : Ce groupe d'activités consiste à concevoir les cas d'utilisation, les classes de test et les packages, les sous systèmes, les capsules, réexaminer la conception et implémenter les composants.

Concevoir la base de données : Ce groupe d'activités consiste à concevoir les classes, la base de données, réexaminer la conception et implémenter les composants.

b. Activités

Estimer la viabilité de l'architecture candidate : Pour évaluer l'architecture candidate synthétisée et déterminer si les exigences critiques de l'architecture sont réalisables et peuvent être utilisées par ce système ou une autre solution.

Analyse architecturale : Pour définir une architecture candidate du système basée sur l'expérience des problèmes de systèmes ou domaines similaires. Pour définir les modèles d'architecture, les mécanismes clés et les conventions de modélisation pour le système. Pour définir la stratégie de réutilisabilité. Pour fournir une entrée au planning du processus.

Identifier les mécanismes de conception : Pour transformer les mécanismes d'analyse en mécanismes de conception basés sur les contraintes imposées par l'environnement d'implémentation.

Identifier les éléments de conception : Pour analyser les interactions des classes d'analyse et identifier les éléments du modèle de conception.

Construire l'architecture candidate :

Pour synthétiser au moins une solution qui devra être tout simplement conceptuelle qui répond aux exigences architecturales critiques.

Incorporer les éléments de conception existants : Pour analyser les interactions des classes d'analyse et rechercher les interfaces des classes de conception et des sous systèmes de conception. Pour affiner l'architecture en incorporant la réutilisabilité où cela est possible. Pour identifier les solutions courantes aux problèmes de conception couramment rencontrés. Pour inclure des éléments du modèle de conception significatifs de l'architecture dans la section de la vue logique du document d'architecture logicielle

Décrire l'architecture d'exécution : Pour analyser les exigences concurrentes pour identifier le processus, identifier les mécanismes de communication interprocessus, allouer les ressources de coordination interprocessus, identifier les cycles de vie du processus et distribuer les éléments du modèle à travers le processus.

Décrire la distribution : Pour décrire comment la fonctionnalité du système est distribuée le long des nœuds physiques. Ceci est fait juste pour les systèmes distribués.

Concevoir la capsule : Pour élaborer et affiner la description d'une capsule.

Analyser le cas d'utilisation : Pour identifier les classes qui réalisent un flot d'événement de cas d'utilisation. Pour distribuer le comportement du cas d'utilisation à ces classes en utilisant les réalisations du cas d'utilisation. Pour identifier les responsabilités, les attributs et les associations des classes. Pour noter l'usage des mécanismes architecturaux.

Concevoir le cas d'utilisation : Pour affiner les réalisations du cas d'utilisation en termes d'interactions. Pour affiner les exigences sur les opérations des classes de conception. Pour

affiner les exigences sur les opérations des sous systèmes et ou leurs interfaces. Pour affiner les exigences sur les opérations des capsules.

Concevoir le sous système : Pour définir les comportements spécifiés dans les interfaces du sous système en termes de collaborations des classes contenues dans le sous système. Pour documenter la structure interne du sous système. Pour affiner les réalisations entre les interfaces du sous système et les classes contenues dans le sous système. Pour déterminer les dépendances avec les autres sous systèmes.

Concevoir la classe : Pour s'assurer que la classe fournit le comportement des réalisations du cas d'utilisation requis. Pour s'assurer qu'il y'a assez d'informations pour implémenter la classe sans ambiguïté. Pour manipuler les exigences non fonctionnelles reliées à la classe. Pour incorporer les mécanismes de conception utilisés par la classe.

Concevoir les classes de test et les packages : Pour concevoir les fonctionnalités spécifiques de test.

Concevoir la base de données : Pour s'assurer que la persistance des données est efficacement stockée et est consistante. Pour définir le comportement qui doit être implémenté dans la base de données.

Réexaminer l'architecture : Pour découvrir un quelconque risque ou des risques perçus dans l'agenda ou le budget. Pour détecter un défaut de conception quelconque dans l'architecture. Pour détecter des incohérences potentielles entre les exigences et l'architecture de conception. Pour évaluer une ou plusieurs qualités architecturales spécifiques (la performance, la sûreté, l'évolutivité, la sécurité). Pour identifier les opportunités de réutilisation.

Réexaminer la conception : Pour vérifier que le modèle de conception est conforme aux exigences du système et qu'il sert de bonne base pour son implémentation. Pour s'assurer que le modèle de conception est conforme au guide général de conception. Pour s'assurer que le modèle de conception remplit ses objectifs.

5.5.3.1.3. L'implémentation

a. Groupe d'activités
   - ***Structurer le modèle d'implémentation***
   - ***Planifier l'intégration***
   - ***Implémenter les composants***
   - ***Intégrer chaque sous-système***
   - ***Intégrer le système***

b. Activités

- ***Développer les composants et les artefacts***
- ***Exécuter les tests***
- ***Construire une accumulation***

- ***Définir la structure du modèle d'implémentation (Couches et sous-systèmes)***
- ***Inspecter le code de qualité et de conformité au standard de projet***

5.5.3.2. Formalisation

La formalisation se fait en adoptant la structure du processus SPEM, utilisé comme un profile UML, au processus RUP. La figure suivante présente le modèle adopté en utilisant le package ***Process Lifecycle de*** SPEM.

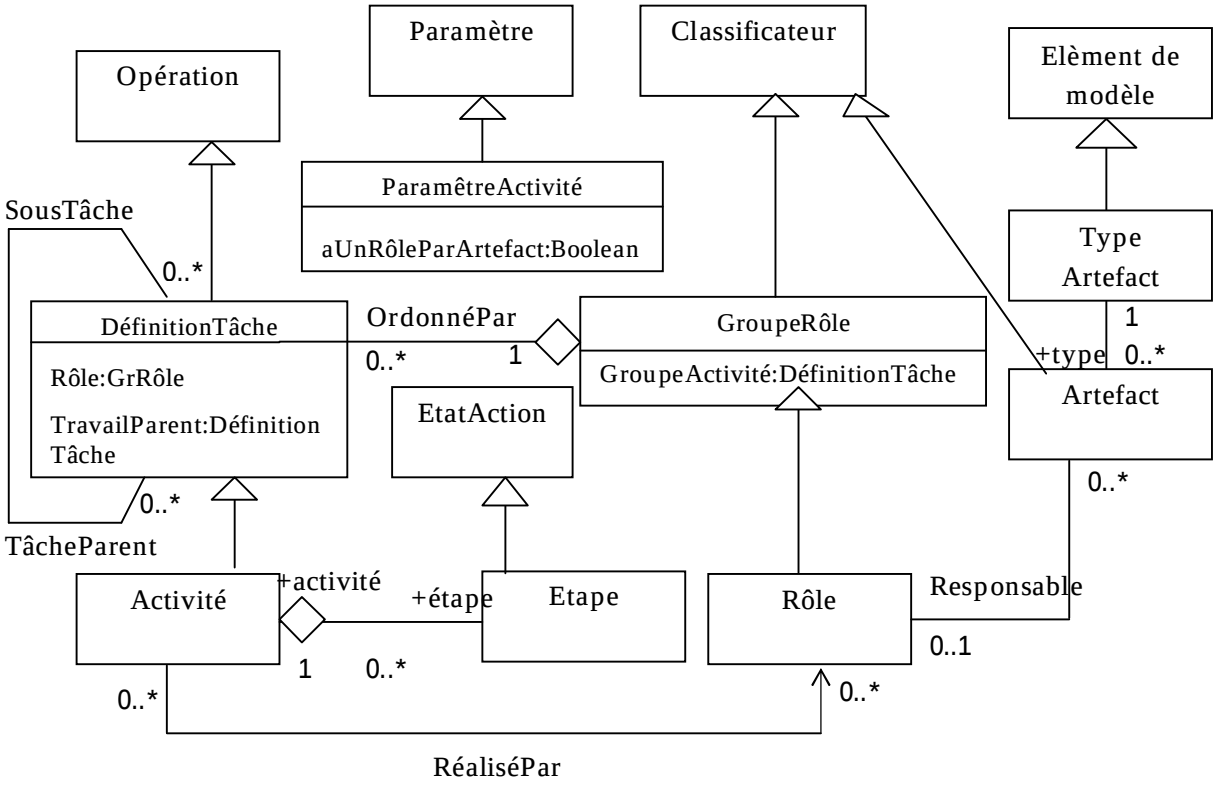

Figure 29 : Modèle de relation entre les tâches dans RUP

Le modèle (adopté de SPEM) de la figure 26, décrit les groupes d'activité comme des tâches (définitionTâche). ***DéfinitionTâche*** est une ***Opération*** qui décrit le travail réalisé dans le processus. Ses sous classes sont ***Activité, Phase, Itération*** et ***CycledeVie*** dans le package ***Process Lifecycle de SPEM.*** Il dispose d'entrées et de sorties explicites référencées par ***ParamêtreActivité***.

Un Artefact est un élément produit, consommé ou modifié par le processus. Un ***Artefact*** décrit une classe de produits de travail résultant d'un processus.

Un ***TypeArtefact*** décrit une catégorie de produits de travail tels que les documents textes, les modèles UML, les exécutables et le code.

Activité est la sous classe principale de ***DéfinitionTâche.*** Elle décrit un travail réalisé par un unique ***Rôle,*** les tâches, les opérations et les actions qui sont réalisées ou assistées par un rôle. Une ***Activité*** peut être décomposée en éléments atomiques appelés Etapes.

Un ***GroupeRôle*** définit un rôle pour un ensemble de ***DéfinitionTâche*** dans le processus &***ProcessPerformer***, dispose d'une sous classe ***GroupeRôle*** qui définit les responsabilités pour des ***Artefacts*** spécifiques et définit les rôles qui réalisent et assistent des activités spécifiques.

5.5.3.3. Représentation des relations entre concepts

Les diagrammes d'UML seront utilisés pour présenter les différents aspects de RUP en respectant la standardisation SPEM. Les figures 27 et 28 représentent les diagrammes d'activité pour les groupes d'activités de la discipline Expression de besoin et de l'implémentation.

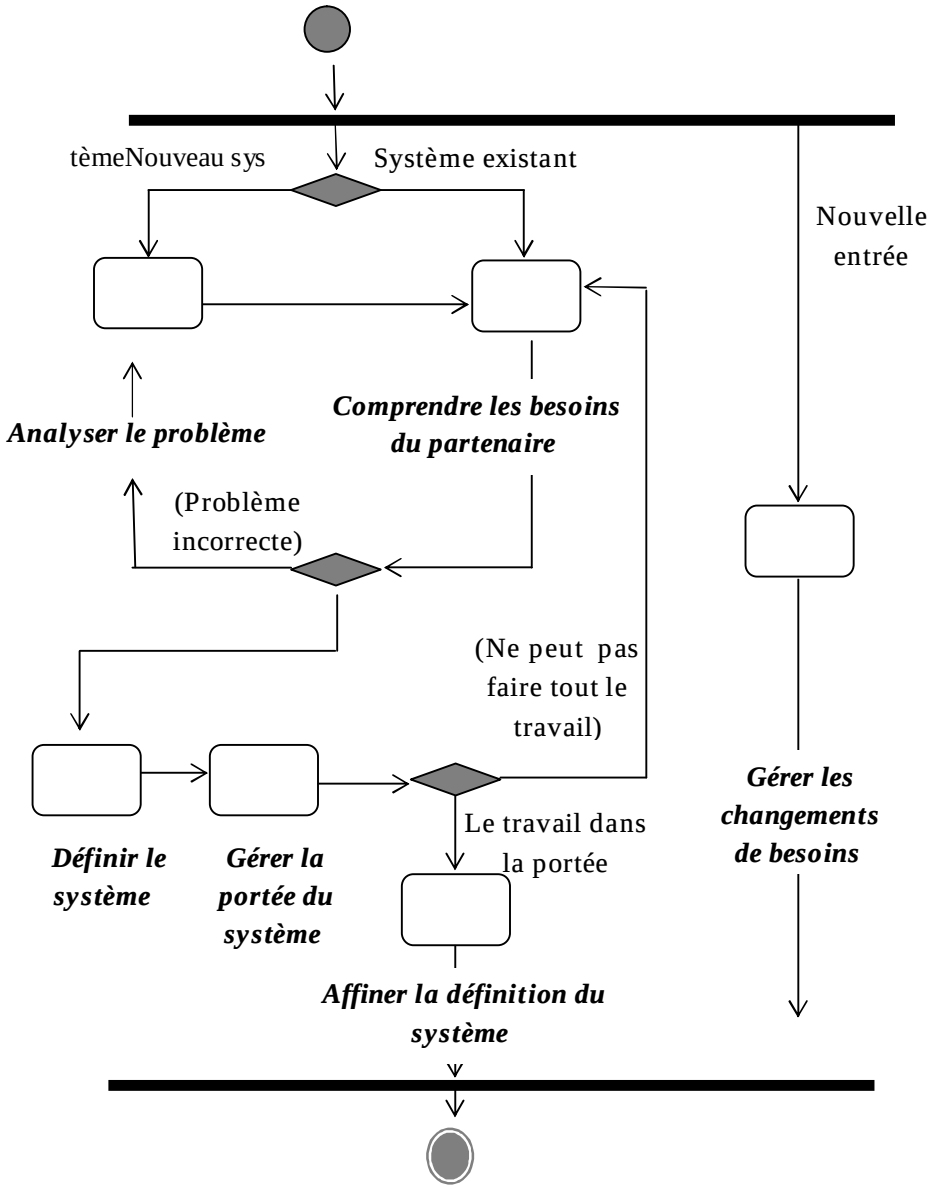

Figure 30 : Diagramme d'activités pour les groupes d'activités de l'expression de besoin

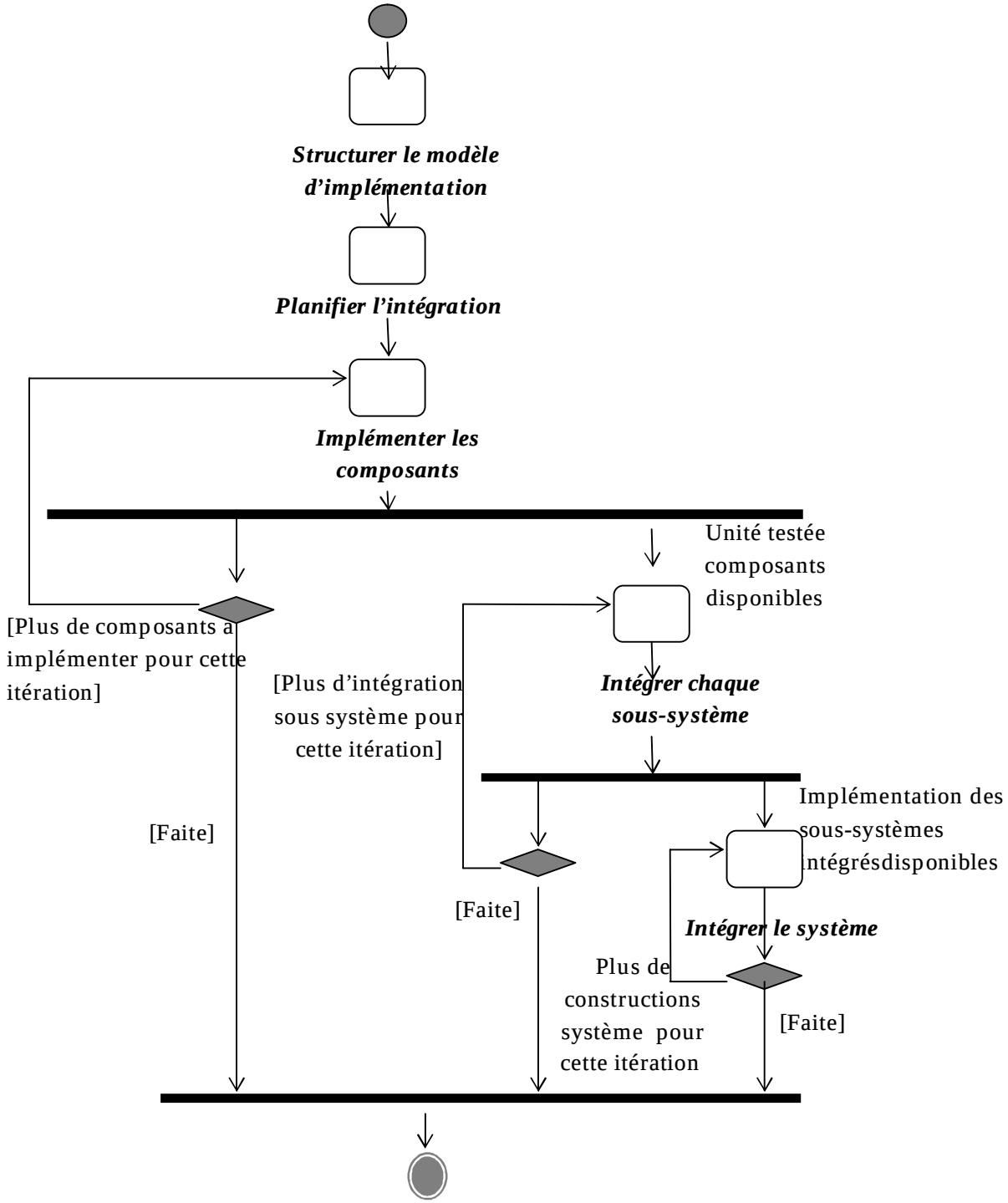

Figure 31 : Diagramme d'activités pour les groupes d'activités de l'implémentation

5.5.3.4. La relation Rôle-Activité-Artefact

L'expression de besoins		
Rôle	Activité	Artefact
Analyste Système	Développer le Plan de Gestion des Besoins Etablir les Requêtes du Partenaire Capturer le Vocabulaire Courant Développer la Vision Structurer le Modèle des Cas d'Utilisation Gérer les Dépendances	Plan de Gestion des Besoins Requêtes du Partenaire Glossaire Vision Modèle des Cas d'Utilisation Spécifications Supplémentaires Attributs des Besoins
Spécificateur des Besoins	Détailler un Cas d'Utilisation Détailler les Besoins Logiciels	Cas d'Utilisation Package des Cas d'Utilisation Spécification des Besoins Logiciels
Architecte Logiciel	Modéliser l'Interface Utilisateur Prototyper l'Interface Utilisateur	Acteur Classe Frontière Historique des Cas d'Utilisation Prototype de l'Interface Utilisateur
Réexaminateur des Besoins	Prioriser les Cas d'Utilisation Réexaminer les Besoins	

Analyse et conception		
Rôle	activité	Artefact
Architecte Logiciel	Estimer la viabilité de L'Architecture Preuve du Concept Analyse Architecturale Identifier les mécanismes de conception Identifier les éléments de conception Construire l'Architecture Preuve du Concept Incorporer les éléments de conception existants Décrire l'Architecture d'Exécution Décrire la Distribution	Modèle de Déploiement Document d'Architecture Logicielle Modèle d'Analyse Modèle de Conception Preuve du Concept Architectural Architecture de Référence Interface Signal Evènement Protocole
Concepteur	Analyse des Cas d'Utilisation Conception des Cas d'Utilisation Conception des Sous systèmes Conception des Classes Conceptions des Classes de Test et des Packages	Réalisation du Cas d'Utilisation Sous système de Conception Package de Conception Classe d'Analyse Classe de Conception
Concepteur de BD	Conception de BD	Modèle de Données
Concepteur de Capsule	Conception de la Capsule	Capsule

Réexaminateur de l'Architecture	Réexamen de l'Architecture	Pas d'artefacts
Réexaminateur de la Conception	Réexamen de la conception	Pas d'artefacts

Implémentation		
Rôle	Activité	Artefact
Implémentateur	Développer les composants et les artefacts Exécuter les tests	Sous-système d'implémentation Eléments d'implémentation Eléments de test Test stub Test du développeur
Intégrateur	construire une accumulation	Plan de construction d'intégration Accumulation
Architecte logiciel	Définir la structure du modèle d'implémentation (Couches et sous-systèmes)	Document d'architecture logiciel Modèle d'implémentation
Réexaminateur de Code	Inspecter le code de qualité et de conformité au standard de projet	Pas d'artefact

Tableau 4 : Relation Rôle-Activité-Artefacts

5.6. Les agents du système

Le résultat du diagramme des rôles conduit aux agents suivants (Figure 23) où l'AGL va se comporter comme un ensemble d'agents qui collaborent ensemble pour réaliser la modélisation.

L'agent Interface : est un agent qui sert d'interface entre l'utilisateur et les agents du système. Il envoie les messages des agents au développeur, et transmet les requêtes de l'utilisateur aux agents. Il communique avec les autres agents via l'agent Superviseur.

L'agent Superviseur : est un agent spécialisé dans le déroulement du workflow, il intègre tous les éléments de déroulement du processus selon des best practices de celui-ci. Il utilise également les éléments de sa base de connaissances et coopère avec les agents

Activité et Rôle pour réaliser un groupe d'activité donnée (rappelons qu'un workflow est exécuté sous forme d'un ensemble de groupes d'activité représentant chacun un détail de workflow (un type de workflow) et donc le groupe d'activités est l'opération qui se déroule dans le processus). Il garde toutes les informations sur le projet en cours de réalisation.

Les agents Rôle : Le système possède un agent pour chaque rôle (définit dans RUP) où un agent Rôle suit l'utilisateur dans la réalisation du rôle, il contrôle l'exécution d'un ensemble d'agents Activité et les artefacts dont il est responsable.

Les agents Activités : On possède un agent pour chaque activité. Chaque agent activité a pour but de créer, consulter ou modifier un artefact. Il se base sur des méthodes et stratégies ou bien des outils à invoquer pour réaliser son travail en coopération avec l'utilisateur.

5.7. Les classes d'agents

Les agents dérivent de la superclasse Agent. Les agents Superviseur, Rôle et Activité appliquent leurs plans lorsque des faits peuvent être vrais. L'agent Interface gère le projet en cours. Tous les agents rôle dérivent de la classe Rôle, et tous les agents Activité dérivent de la classe Activité. La figure 30 montre la hiérarchie des classes.

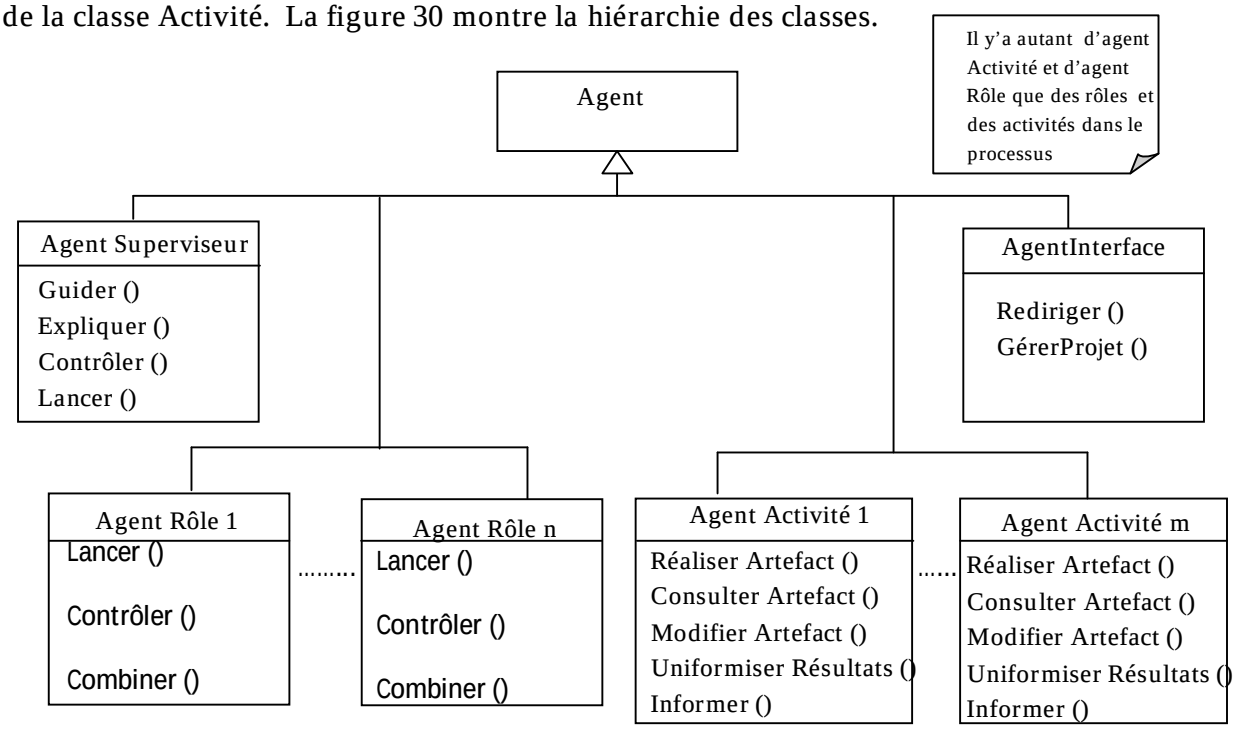

Figure 32 : La hiérarchie des classes d'agents

5.8. Le diagramme de classes

Le diagramme de classes montre les classes d'agents et les relations d'interaction entre elles. Pour notre cas, comme il est montré dans la figure 29, un Projet suit généralement un

processus. Le processus est composé des activités et des rôles réalisant ces activités. Les rôles et les activités peuvent appartenir à plusieurs processus à la fois.

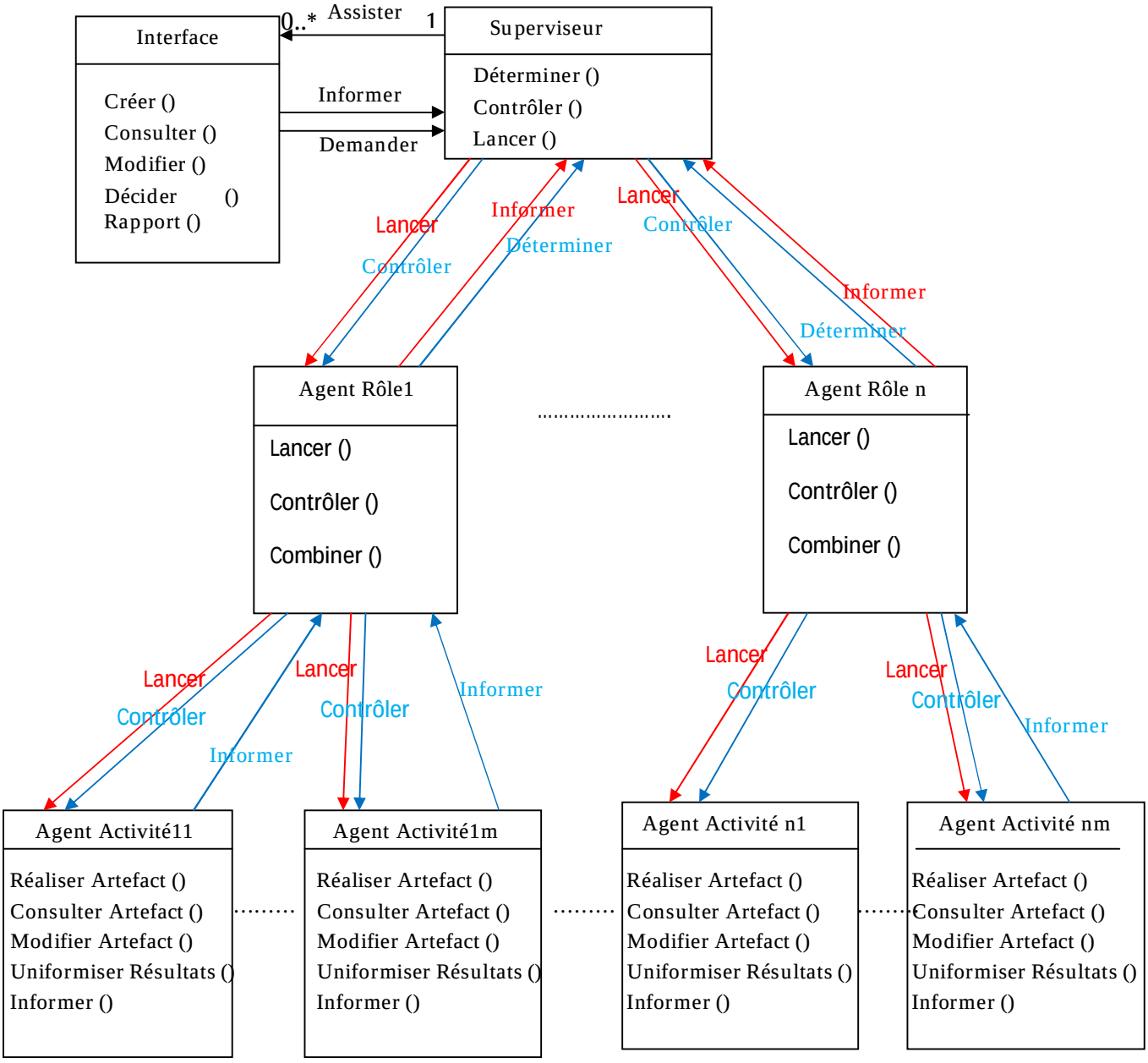

Figure 33 : Diagramme de classes

5.9. Structure Interne des agents

5.9.1. L'agent Superviseur

C'est un agent cognitif BDI, sa structure interne, représentée par la figure 31, est constituée de trois composants. Un module de connaissance représentant les croyances de

l'agent, un module de raisonnement et un module d'action représenté par les plans : Expliquer, Guider et Contrôler. L'objectif de l'agent est toujours d'assister un workflow. Pour ce faire, il va lancer et contrôler l'exécution d'un workflow d'agents rôle. Les croyances, les objectifs et les intentions de l'agent Superviseur sont représentés dans le tableau 4.

a. Le module de connaissance

Ce module représente la connaissance de l'agent sur les phases, les itérations, les disciplines, les rôles, les activités et les groupes d'activités (les détails du workflow). Le séquencement et les dépendances entre eux, les prés et les post conditions de l'application de chaque entité ...etc.

En plus, l'agent Superviseur garde la connaissance sur le projet en cours de réalisation ; les entités qui ont été déjà exécutés, les entités en cours d'exécution et les entités qui ne sont pas encore exécutés.

b. Le module de raisonnement

En se basant sur les connaissances du module de connaissance et les perceptions de l'environnement (à partir des messages des autres agents et de la modification de la base d'artefacts), l'agent peut décider l'action à exécuter en réalisant deux types de raisonnements : ***Raisonnement en chaînage avant*** et ***raisonnement en chaînage arrière***.

Le raisonnement en chaînage-avant consiste à raisonner à partir des conditions courantes jusqu'aux buts. En effectuant le raisonnement du chaînage avant, l'agent peut déterminer si une étape est exécutable ou non sous les conditions actuelles.

Le raisonnement en chaînage arrière est utilisé pour implémenter les comportements proactifs de l'agent, par exemple, la planification. Par le biais du raisonnement de chaînage arrière, l'agent ajoute à son plan un ensemble d'étapes dont les post-conditions répondent à l'exigence de l'objectif et puis planifie les prés conditions de cet ensemble d'étapes comme de nouveaux objectifs.
Pour la décision sur les détails de workflows et les workflows, les contraintes sont évaluées en coopération avec l'utilisateur.
Pour la décision sur les itérations et les phases, elle est déterminée par les jalons et les jalons mineurs, dans la réalité c'est les intervenants au projet (représentés dans notre cas par les rôles) qui les évaluent. Dans notre cas c'est le développeur qui décide.

c. Le module d'action

Le moteur d'action agit comme le cerveau de l'agent superviseur. Il permet de contrôler le comportement de l'agent tel que la prise de décision, la planification, le suivi, la modification…etc. Ce module exécute les décisions prises par le module de raisonnement par l'envoie des messages convenants aux agents concernés ou encore l'exécution des plans Expliquer, Guider et Contrôler à la demande d'assistance sur un workflow par l'utilisateur.

Par exemple, quand le module de raisonnement applique le chaînage en arrière et décide d'anticiper quelques activités, le module d'action envoie un message de recommandation d'exécution à l'utilisateur et envoie des messages aux agents rôles concernés pour commencer l'exécution.

L'agent Superviseur interagit et coopère avec tous les agents du système. A chaque exécution d'une activité ou d'un rôle, l'agent Superviseur reçoit un message et donc il modifie ses croyances et ses intentions.

Les croyances	Objectifs	Plans
Les phases Les itérations Les activités Les groupes d'activités Les dépendances	Assister et suivre une itération	Expliquer Guider Contrôler

Tableau 5 : Croyances, Objectifs et Plans de l'agent Superviseur

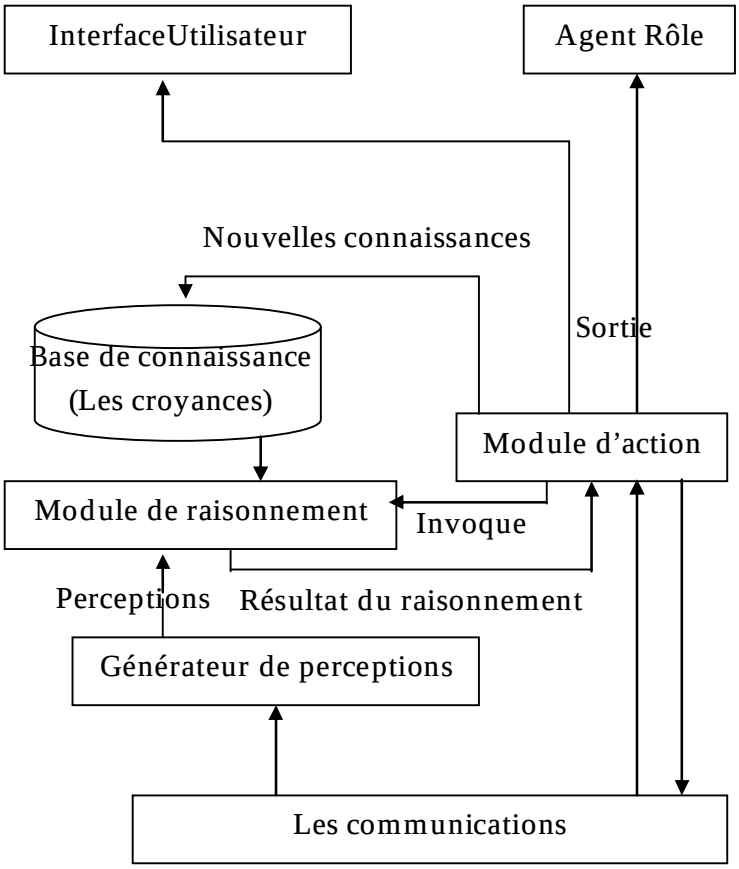

Figure 34 : Structure interne de l'agent Superviseur

5.9.2. Les agents_Rôle

Chaque agent Rôle est un agent BDI (Le tableau 6 donne les croyances, les désirs et les intentions de l'agent) dont le rôle est d'assister l'utilisateur dans l'adoption d'un rôle et l'exécution d'un ensemble d'activités ; dans notre cas il est responsable de l'exécution d'un workflow d'agents Activité. Sa structure est composée des trois modules suivants :

a. Une base de connaissance

Chaque rôle possède de la connaissance sur les activités à exécuter (les contraintes de leur exécution et leur état d'exécution) et les artefacts à produire ou à consulter. Ces connaissances constituent les croyances de l'agent.

b. Un module raisonnement

Utilise les croyances de l'agent qui sont représentés dans la base de connaissances et les règles des dépendances entre les activités pour déterminer les objectifs de l'agent : les actions à exécuter et les décisions à prendre.

c. Un module Action

Exécute les décisions prises par le module de raisonnement et qui sont toujours l'exécution des plans Expliquer, Guider et Contrôler avec à chaque fois des données différentes.

L'agent Rôle communique avec les agents activités dont il est responsable, collabore avec les autres agents rôle, et communique avec l'agent Superviseur.

Les croyances	Objectifs	Plans
Les activités Les artefacts Les dépendances	Assister et suivre un rôle	Expliquer Guider Contrôler

Tableau 6 : Croyances, Objectifs et Plans de chaque agent de type Rôle

5.9.3. Les agents Activité

Chaque agent Activité est un agent cognitif, il se base sur des méthodes, des stratégies et des données (artefacts) en entrée pour réaliser un artefact spécifique en sortie. Il peut même invoquer un outil pour la réalisation de sa tâche, on propose dans ce cas là un agent couverture (un Wrapper) pour chaque outil. L'agent couverture sert à faciliter l'invocation et la communication entre l'agent Activité et l'outil. Ainsi, l'agent activité envoie directement un message à l'agent couverture, ce dernier transmet le message à l'outil qui va réaliser la tâche demandée.

Chaque agent activité dispose de trois modules (Figure 33) :

a. Un module connaissance

L'agent contient la connaissance sur l'artefact utilisé en entrée et celui qui doit être produit en sortie. Il possède également la connaissance sur les outils et les méthodes à invoquer.

b. Un module raisonnement

Il se base sur les croyances déterminés dans le module de connaissance et utilise les règles et les contraintes pour prendre les décisions sur la production des artefacts.

c. Un module Action

Il exécute les décisions prises par le module raisonnement en envoyant des messages aux agents appropriés. L'agent communique à l'agent rôle et collaborent avec les autres agents activité. Il communique aussi avec l'agent Superviseur.

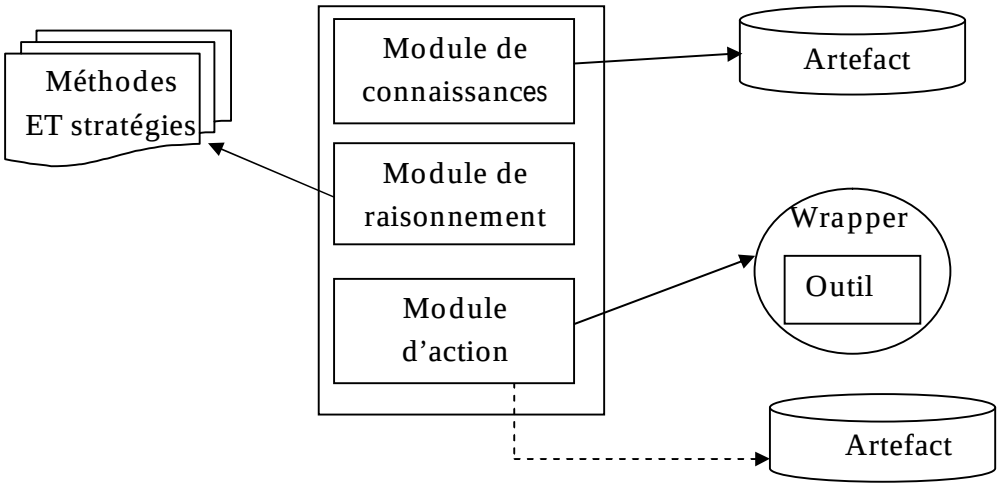

Figure 35 : Structure interne de l'agent Activité

5.10. Interaction entre les agents Rôle

Comme RUP est piloté par les cas d'utilisation (use case driven), donc après l'identification des cas d'utilisation, l'analyse et la conception sont exécutés itérativement pour chaque cas d'utilisation, d'où le schéma suivant d'interaction entre les agents rôle des deux disciplines de l'expression de besoin et de l'analyse et la conception :

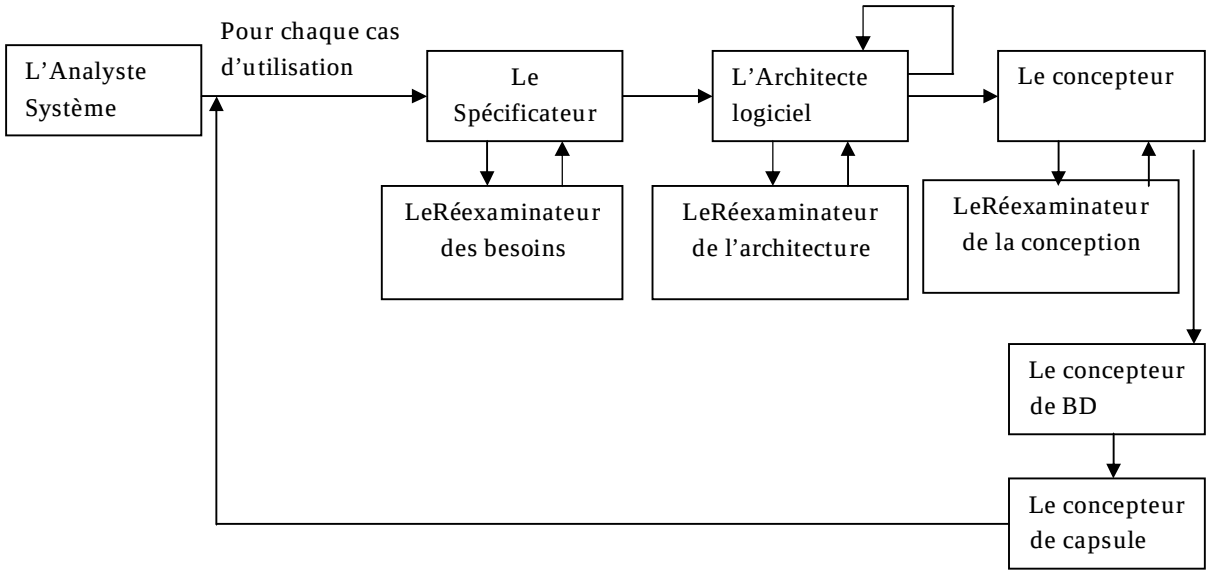

Figure 36 : Interaction entre les agents rôle

Le tableau suivant montre les agents rôle et les agents activités dans la discipline de l'analyse et la conception.

Agent Rôle	Agents Activité	Rôle	Artefact
Architecte Logiciel	L'agent_Estimer la viabilité de L'Architecture	Responsable de l'exécution de la méthode Estimer la viabilité de L'Architecture	Modèle de Déploiement
	L'agent_Preuve du Concept Analyse Architecturale	Responsable de l'exécution de la méthode Preuve du Concept Analyse Architecturale	Modèle d'Analyse
	L'agent_ Identifier les mécanismes de conception	Responsable de l'exécution de la méthode Identifier les mécanismes de conception	Modèle de Conception
	L'agent_ Identifier les éléments de conception	Responsable de l'exécution de la méthode Identifier les éléments de conception	Preuve du Concept Architectural
	L'agent_ Construire l'Architecture Preuve du Concept	Responsable de l'exécution de la méthode Construire l'Architecture Preuve du Concept	Architecture de Référence
	L'agent_ Incorporer les éléments de conception existants	Responsable de l'exécution de la méthode Incorporer les éléments de conception existants	Interface
	L'agent_ Décrire l'Architecture	Responsable de l'exécution de la	Signal Evènement

	d'Exécution	méthode Décrire l'Architecture d'Exécution	
	L'agent_ Décrire la Distribution	Responsable de l'exécution de la méthode Décrire la Distribution	Protocole
Concepteur	L'agent_ Analyse des Cas d'Utilisation	Responsable de l'exécution de la méthode Analyse des Cas d'Utilisation	Réalisation du Cas d'Utilisation
	L'agent_ Conception des Cas d'Utilisation	Responsable de l'exécution de la méthode Conception des Cas d'Utilisation	Sous système de Conception
	L'agent_ Conception des Sous systèmes	Responsable de l'exécution de la méthode Conception des Sous systèmes	Package de Conception
	L'agent_ Conception des Classes	Responsable de l'exécution de la méthode Conception des Classes	Classe d'Analyse
	L'agent_ Conceptions des Classes de Test et des Packages	Responsable de l'exécution de la méthode Conceptions des Classes de Test et des Packages	Classe de Conception
Concepteur de BD	L'agent_ Conception de BD	Responsable de l'exécution de la méthode Conception de BD	Modèle de Données
Concepteur de Capsule	L'agent_ Conception de la Capsule	Responsable de l'exécution de la méthode Conception de la Capsule	Capsule

Réexaminateur de l'Architecture	L'agent_ Réexamen de l'Architecture	Responsable de l'exécution de la méthode Réexamen de l'Architecture	Pas d'artefacts
Réexaminateur de la Conception	L'agent_ Réexamen de la conception	Responsable de l'exécution de la méthode Réexamen de la conception	Pas d'artefacts

Tableau 7 : Les agents rôle et activité de l'Analyse et la conception

5.11. L'architecture du système

La dernière étape dans la construction d'un système multi agents, en utilisant la méthodologie MASE, est la configuration du système.

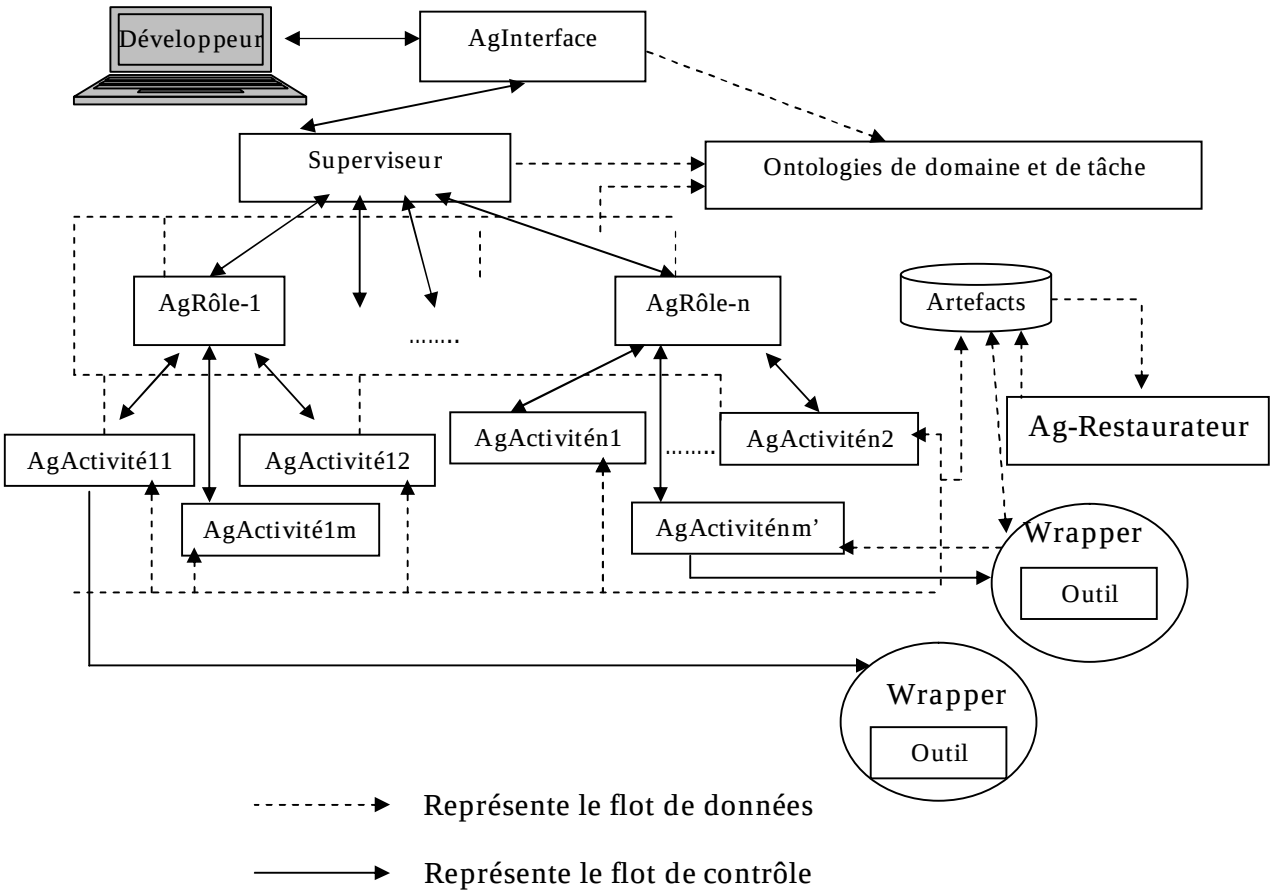

Figure 37 : Architecture proposée

L'architecture de notre système montrée par la figure 34 est constituée de trois composantes; la première représente les ontologies de domaine et de tâche qui représentent le langage commun entre les agents. La deuxième composante représente le système multi agents qui va réaliser l'assistance et la troisième présente l'interface avec le développeur et la base d'objet.

L'ontologie de domaine fournie les concepts de base du processus et l'ontologie de tâche décrit les tâches qui vont être échangés dans le système, formant ainsi un méta langage décrivant la connaissance appropriée au développement du logiciel (à la modélisation du logiciel).

Un AGL qui suit RUP peut être vu comme un ensemble d'agents Rôle responsable chacun d'un ensemble d'agents Activité (d'un workflow d'agents Activités) et manipulant un grand ensemble d'objets appelés artefacts et qui sont tels que : les spécifications, les documents de conception, les protocoles de tests…etc. Les agents collaborent par communication pour atteindre l'objectif commun (à savoir la réalisation du logiciel). Chaque agent a donc son objectif local et l'objectif global. La collaboration se fait selon le processus logiciel RUP.

L'agent Superviseur sert à gérer le workflow des workflows des activités du processus.

La base d'objet

Elle contient tous les objets du logiciel du domaine (y compris les outils). Elle fournit à l'environnement des informations sur les différentes classes d'objets et les relations entre les objets. Ex: Un objet est un composant d'un autre objet, un objet particulier peut être appliqué à un autre objet pour produire un troisième.

Les objets logiciels sont traités en tant qu'entité typées et uniquement nommées dont la structure et les propriétés sont exprimées en termes d'une liste d'attributs. Ces attributs sont référencés aux objets.

Les objets logiciels (les artefacts) sont créés dynamiquement pendant l'évolution du processus de développement de logiciel. La plupart des objets persistent pendant le développement et changent seulement leurs valeurs d'attributs.

Le système utilise un agent Interface pour gérer les interactions du développeur avec le système. Le développeur peut adopter un rôle, exécuter une activité ou suivre un workflow. Si sa demande n'est pas acceptée, des explications et des recommandations vont être données selon le raisonnement des agents.

Les agents Activité ont pour rôle d'exécuter des différentes activités. Le but d'une activité est typiquement de créer ou de développer les artefacts. Il possède de la connaissance sur les artefacts en entrée et en sortie. La base de connaissance d'une activité peut inclure la description des outils qui doivent être invoqué par l'activité, dans ce cas là il se trouve

dans sa base de connaissance les déclarations appropriées qui seront utilisées comme interface dans l'invocation des outils décrits.

L'agent Superviseur joue un rôle principale en contrôlant aussi le déroulement et l'exécution de tout le processus et ceci en gardant toutes les informations du projet en cours d'exécution.

L'agent Superviseur sauvegarde sa base de connaissance avant chaque début d'itération et à chaque fin d'itération, l'agent négocie avec l'utilisateur la satisfaction des résultats. Si non satisfaction, l'agent restaure les connaissances et réexécute l'itération. La restauration de la connaissance sur les artefacts se fait par un agent Restaurateur, qui sauvegarde la base de connaissance à chaque début d'itération et la restaure si l'agent Superviseur lui envoie une demande.

Dans le paragraphe suivant, nous allons donner un exemple sur un outil pour limiter le nombre de rôles et ainsi bien montrer les interactions et les comportements des agents.

6. Exemple d'un outil CASE utilisant la méthode OMCP

Pratiquement, on prend un outil CASE qui utilise une méthode orientée objet connue par la méthode OMCP. On applique la méthode de la spécification des besoins jusqu'à l'analyse et la conception ; la liste finale des artefacts sera la suivante :

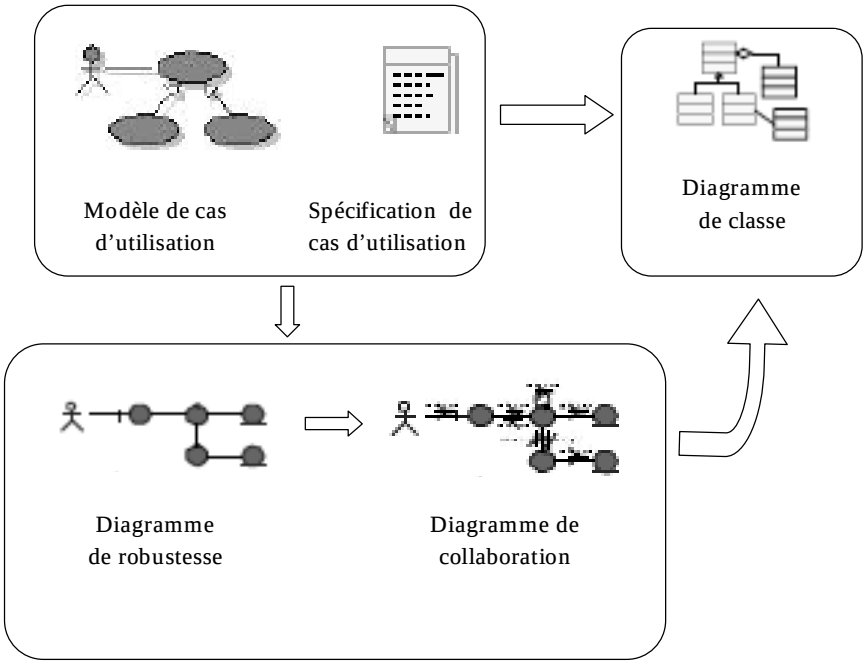

Figure 38 : Liste des artefacts [55]

Et l'architecture du système est donnée par la figure 36 où tous les agents ont besoin d'interagir avec l'utilisateur (via l'agent Superviseur) :

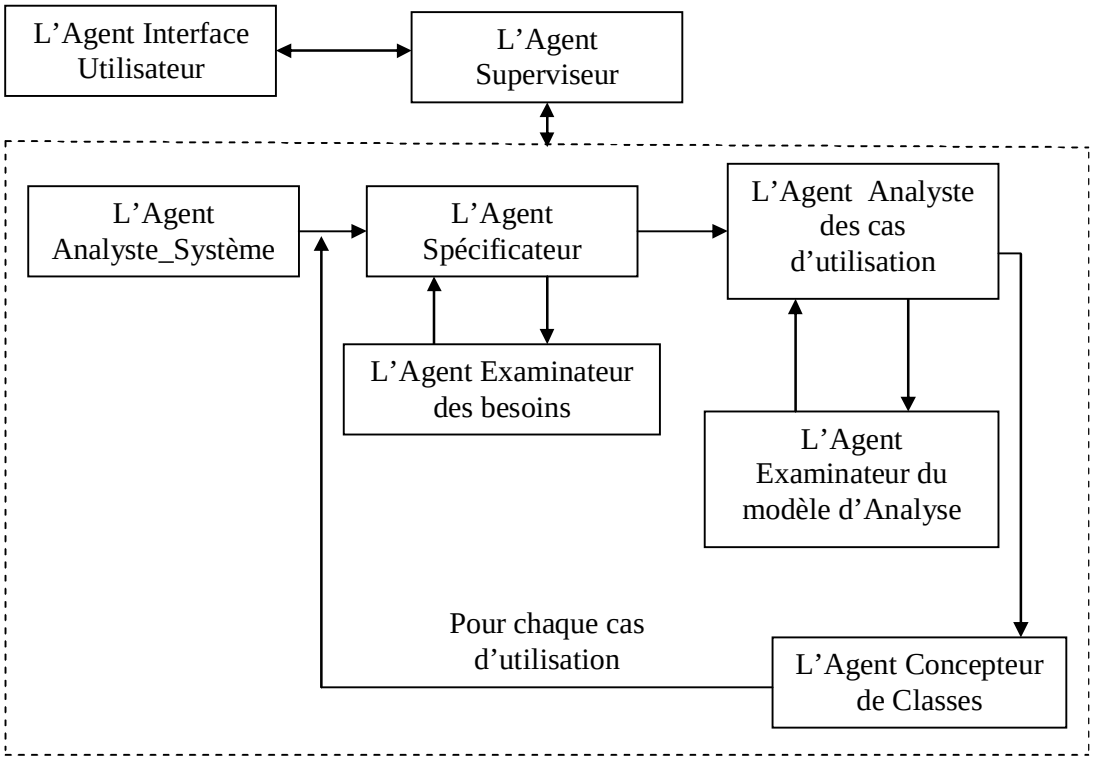

Figure 39 : L'interaction entre les agents Rôle

La collaboration entre les agents est représentée par le tableau suivant :

Agent	Agent-Activité	Rôles	Artefacts	Collaborations
Analyste de système (AS)	Agent responsable de la méthode Trouver les acteurs	Trouver les acteurs	Les acteurs	Envoie message à l'agent Spécificateur (SC)
	Agent responsable de la méthode Trouver les cas d'utilisation	Trouver les cas d'utilisation	Les cas d'utilisation	
	Agent responsable de	structurer les cas	Le diagramme des cas	

	la méthode structurer les cas d'utilisation	d'utilisation	d'utilisation	
Spécificateur des cas d'utilisation (SC)	Agent responsable de la méthode Spécifier les cas d'utilisation	Spécifier les cas d'utilisation	La spécification des cas d'utilisation	Reçoit messages de l'AS Reçoit et Envoie messages de et à partir de l'agent EE Envoie messages à l'AC
Examinateur des Exigences (EE)	L'agent responsable de la méthode Examiner le modèle des cas d'utilisation	Examiner le modèle des cas d'utilisation	Pas d'artefacts	Reçoit et envoie des messages de et à partir de EA
Analyste des cas d'utilisation (AC)	L'agent responsable de la méthode Réaliser les cas d'utilisation	Réaliser les cas d'utilisation	Le modèle d'analyse	Reçoit messages de SC Reçoit et envoie des messages de EA Envoie messages à CC
Examinateur du modèle d'analyse (EA)	L'agent responsable de la méthode Examiner le modèle d'analyse	Examiner le modèle d'analyse	Pas d'artefacts	Reçoit et envoie des messages à partir d'AC
Concepteur de classe (CC)	L'agent responsable de la méthode Concevoir le modèle de classe	Concevoir le modèle de classe	Le modèle de conception	Reçoit des messages à partir d'AC Envoie message à l'utilisateur

Tableau 8 : La collaboration entre les agents rôle

6.1. Le comportement de quelques agents du système

Dans ce qui suit, nous spécifierons le comportement de quelques agents utilisés dans l'architecture précédente :

6.1.1. L'Agent Analyste Système

Rôle : Identifier les acteurs et les cas d'utilisation, et structurer les cas d'utilisation.

Et donc, l'agent Analyste_Système va être responsable d'un ensemble d'agents Activités dont les rôles sont les suivants : Trouver les acteurs, trouver les cas d'utilisation et structurer les cas d'utilisation. Chaque agent (Figure 37) va atteindre son objectif en utilisant le document de vision, le glossaire et les règles de son module de raisonnement représentés par une méthode ou une stratégie d'analyse ; ou bien en utilisant un outil d'analyse. Les deux agents TrouverActeur et TrouverCasd'utilisation peuvent commencer leur exécution, à la fin chaque agent va uniformiser les résultats pour être utiliser par l'agent StructurerCasd'utilisation. L'agent AnalysteSystème (Agent de type Rôle) combine les résultats et informe l'agent superviseur de la terminaison de sa tâche.

Il existe plusieurs types de méthodes d'analyse de besoins :

- Les méthodes formelles
- Les méthodes informelles, comme l'analyse des besoins des phrases en langage naturel.

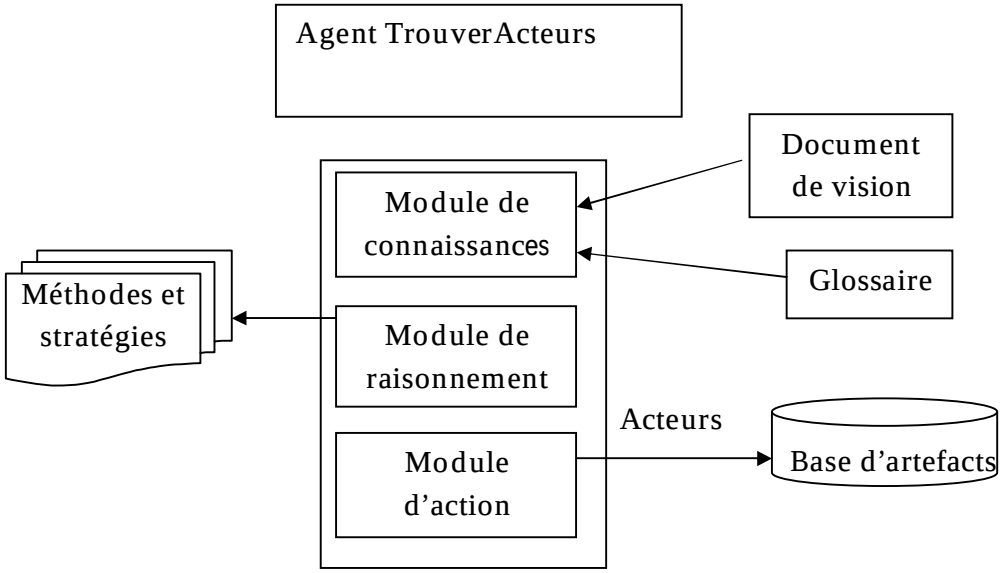

Figure 40 : Structure interne de l'agent responsable de la méthode TrouverActeur dont l'agent Analyste Système est responsable

Le comportement de l'agent Analyste Système peut être représenté par la figure suivante :

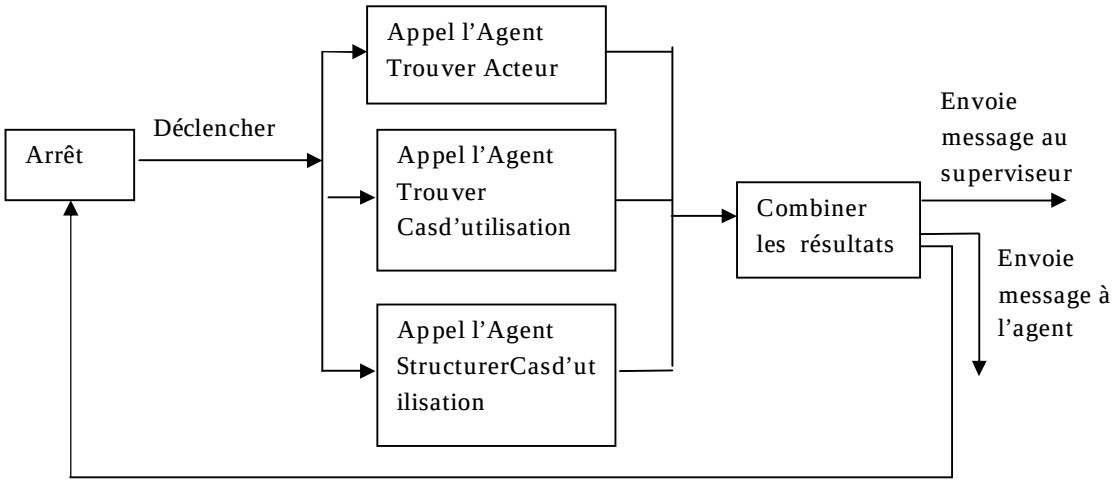

Figure 41 : Le comportement de l'agent Analyste Système

6.1.2. Le comportement de l'agent Spécificateur des cas d'utilisation

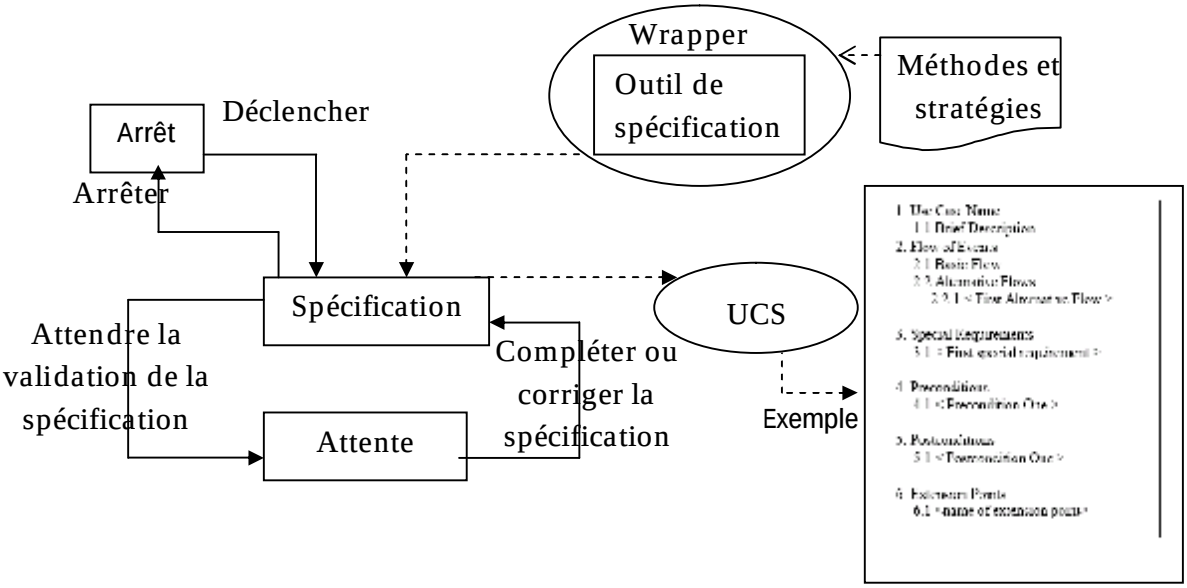

Figure 42 : Le comportement de l'agent Spécificateur pour un cas d'utilisation

En RUP les besoins sont entrées de deux façons, à l'aide d'un document appelé ***Document de vision***, qui contient les besoins en langage naturel et à partir du ***diagramme de cas d'utilisation***. Un ***glossaire*** de termes aussi sert à définir le sens des termes du domaine d'application. La spécification est accomplit suivant des ***méthodes formelles*** ou des ***méthodes informelles***.

6.1.3. Le comportement de l'agent Analyste des cas d'utilisation

Responsable de l'identification des objets, de leurs relations et leurs associations. Donc de construire les digrammes de robustesse et de collaboration. Son comportement peut être résumé par la figure suivante:

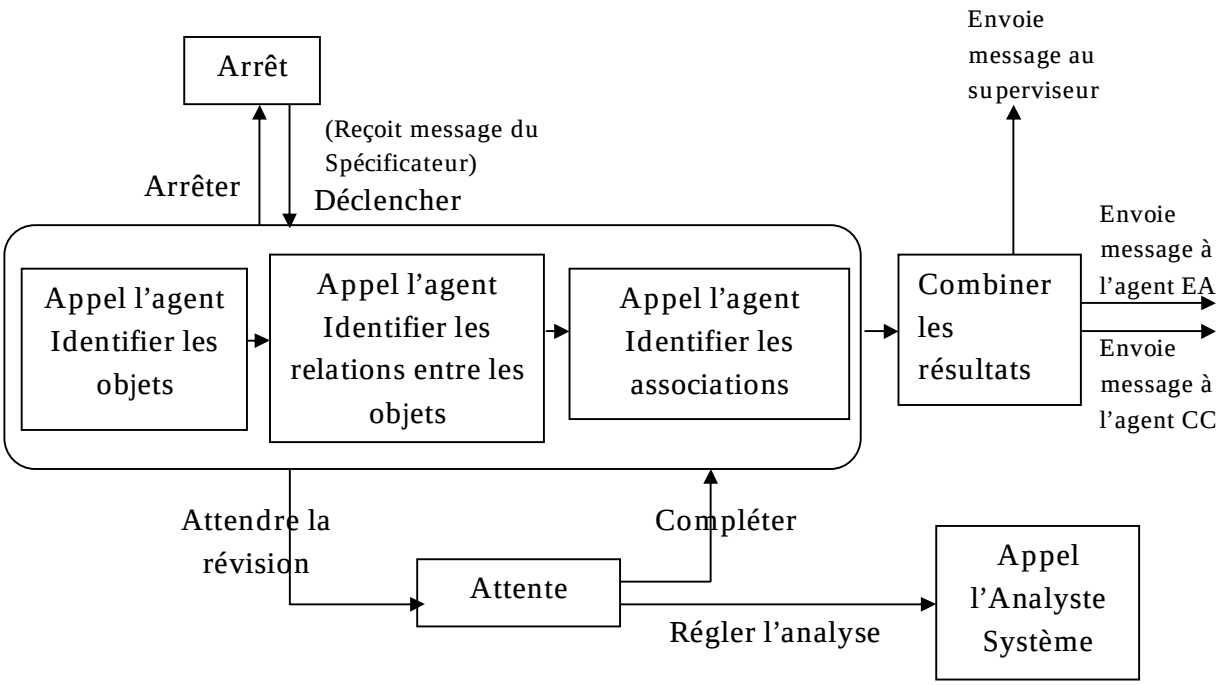

Figure 43 : Le comportement de l'agent Analyste de cas d'utilisation

Les objets et les messages entre eux peuvent être tirés à partir des flous d'événements spécifiés dans la spécification de chaque cas d'utilisation, manuellement par l'utilisateur ou par l'invocation d'un outil.

7. Conclusion

Le but de notre travail est de modéliser un AGL avec un système multi agents afin de participer à l'amélioration de ces AGL et ainsi l'amélioration de la qualité du logiciel produit par ces outils. Pour réaliser cela, on a essayé de fournir à l'utilisateur une assistance intelligente et active pendant la modélisation de son logiciel en suivant le processus RUP caractérisé par ses « best practice » qu'il offre.

Pour atteindre cet objectif, nous avons commencé par analyser notre système à l'aide de la méthodologie MASE avec laquelle on a réalisé les modèles nécessaires à l'analyse et à la conception de l'outil, passant par la réalisation d'une ontologie de domaine et de tâche qui servent comme un langage commun entre les agents, arrivant enfin à une conception finale de notre outil.

Une description de quelques scénarios de fonctionnement est nécessaire pour bien saisir le fonctionnement du système et le comportement des agents, elle fera l'objet du chapitre suivant.

Chapitre 4

Description du fonctionnement de l'outil d'Assistance à la modélisation

1. Introduction

Afin de comprendre plus aisément le fonctionnement de notre système, on va décrire quelques scénarios de son fonctionnement.

Nous commençons le chapitre par une présentation du code OWL des deux ontologies, puis nous décrivons quelques scénarios possibles pour le fonctionnement de l'outil OAM (Outil d'Assistance à la Modélisation).

2. L'ontologie

Pour la construction de l'ontologie du processus de développement générique RUP nous avons utilisé Protégé ; un environnement graphique et interactif de conception d'ontologie de développement de base de connaissances. Les règles sur ces connaissances seront aussi fournies. Elles sont représentées par les dépendances. L'ontologie a été générée avec l'outil Protégé. Une partie du code OWL de l'ontologie est donné dans ce qui suit :

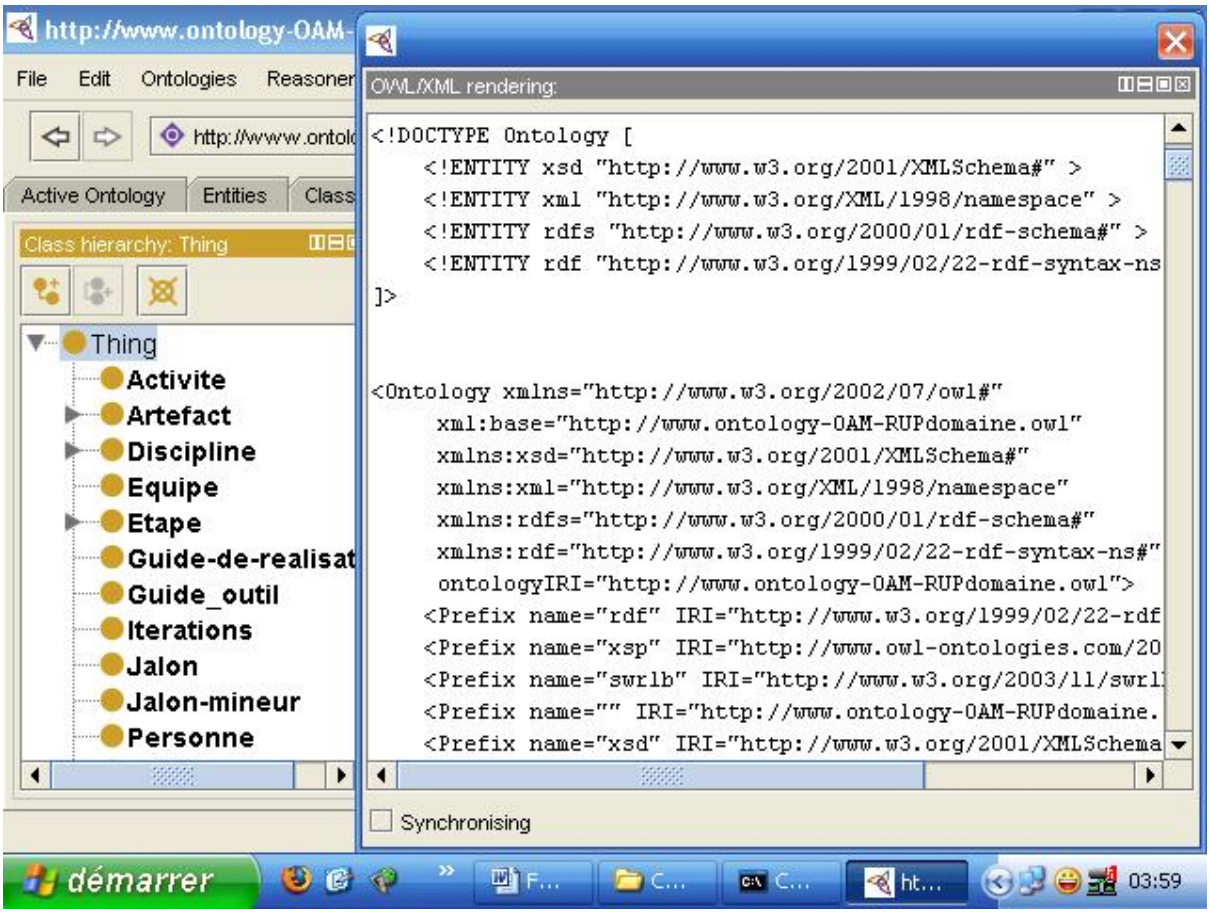

Figure 44 : Une partie du code OWL de l'ontologie de domaine de RUP

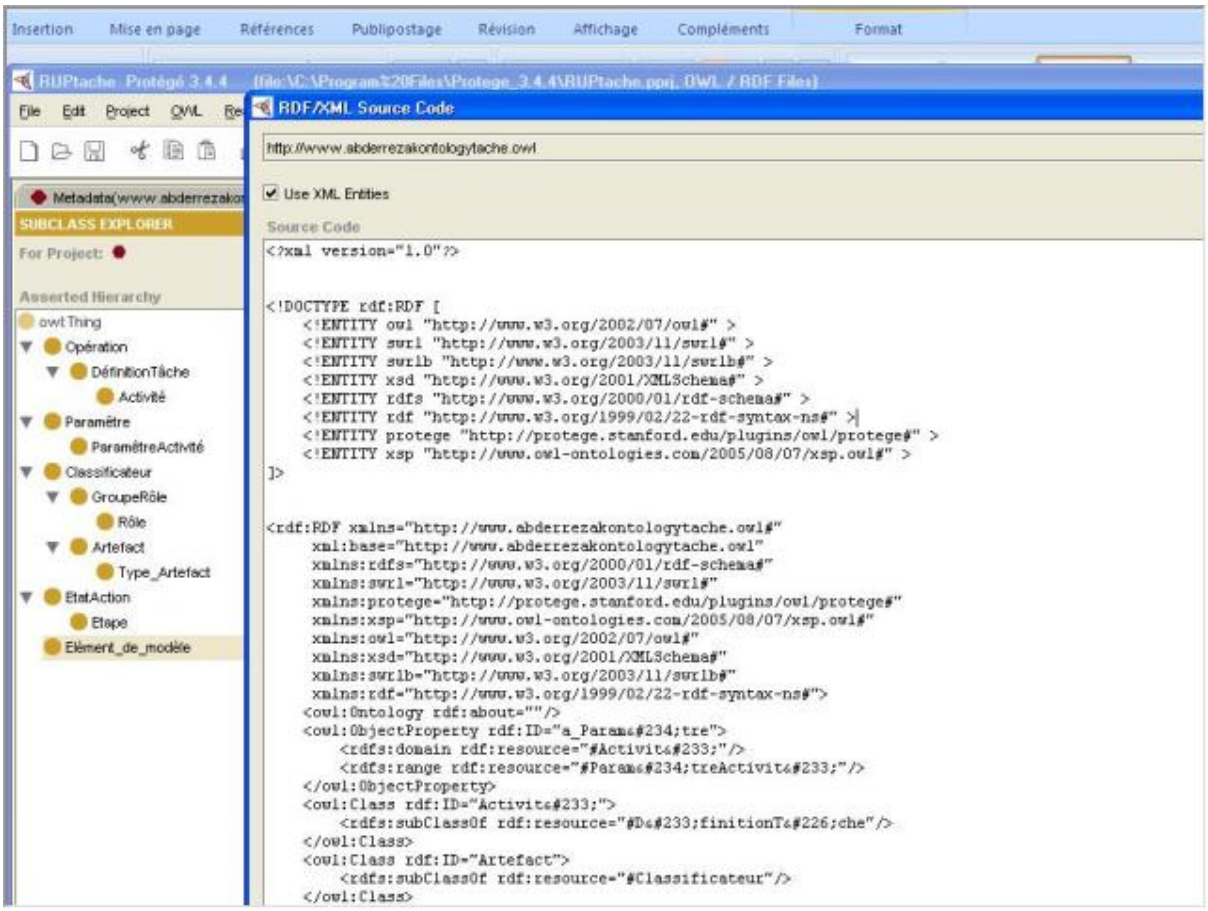

Figure 45 : Une partie du code OWL de l'ontologie de tâche de RUP

3. Quelques scénarios possibles de fonctionnement

a. Scénario de fonctionnement du système

L'outil OAM offre une assistance au développeur pendant le développement de son projet. Cette assistance consiste essentiellement à le guider, lui fournir des explications si nécessaire et contrôler son travail. Le scénario suivant décrit la réalisation d'un projet avec l'outil OAM.

Etape de création du projet : Le développeur crée un projet en précisant les informations relatives au projet : nom, nombre d'itérations, les besoins sont spécifiés dans le document de vision ... L'Agent Interface informe l'Agent Superviseur du début d'un projet. Le superviseur requiert toutes les informations sur le projet. Le développeur spécifie une

itération. L'Agent Interface informe l'Agent Superviseur du début d'une itération. Le superviseur requiert toutes les informations sur l'itération.

- ***Etape de choix du groupe d'activité :*** Le choix du groupe d'activité dépend des contraintes spécifiées, ces contraintes sont discutées à l'utilisateur qui va prendre la décision convenant avec l'expérience de l'agent. L'utilisateur sélectionne le groupe d'activités à réaliser : ***Définir le système.*** Une description du groupe d'activité est fournie au développeur qui sélectionne l'itération concernée. L'Agent Superviseur est aussitôt informé de l'itération en cours. Les Agents de l'itération sont ***Capturer le vocabulaire courant, Développer la vision, Rechercher les acteurs et les cas d'utilisation*** et ***Gérer les besoins*** et ils peuvent être exécutés en parallèle si leurs pré-conditions sont satisfaisantes.
- ***Etape d'adoption d'un rôle :*** Le développeur adopte un rôle celui de l'Analyste Système. L'Agent Superviseur envoie un message à l'agent Analyste Système, ce dernier maintient un ensemble d'agents Activité dont leurs pré-conditions sont vraies. Donc les agents ***Capturer le vocabulaire courant*** et ***Développer la vision*** peuvent s'exécuter.
- ***Etape Réalisation d'une activité :*** Chaque agent utilise en entrée les besoins de l'utilisateur et se base sur une méthode spécifique pour produire à la fin le glossaire et le document de vision. Une autre liste des agents Activité peut être exécutée. L'agent Analyste Système envoie un message aux agents TrouverActeur et TrouverCasd'utilisation. Chaque agent utilise les besoins dans le document de vision et le vocabulaire définit dans le glossaire et invoque une méthode d'analyse ou un outil spécifique pour produire en sortie les acteurs et les cas d'utilisation. A la terminaison, ils envoient des messages à l'agent AnalysteSystème et à l'agent Superviseur qui modifient leurs croyances et leurs objectifs. L'AnalysteSystème peut invoquer maintenant l'agent activité StructurerCasd'utilisation.
- A la fin, l'agent AnalysteSystème envoie un message à l'agent Spécificateur pour commencer son exécution. L'agent superviseur met à jour ses croyances et objectifs et ainsi son plan d'exécution et invoque les autres agents du workflow planifiés dans le plan d'exécution (dont les contraintes d'invocation sont vérifiées) et ainsi de suite.

- ***Etape de demande d'exécution d'une activité :*** Le développeur peut demander l'exécution d'une activité d'un autre groupe qui n'est pas encore disponible, c'est-à-dire, qui demande en entrée des artefacts qui ne sont pas encore disponibles. L'agent

Superviseur fait un raisonnement en arrière et entre touts les groupes d'activités, les rôles, les activités et les artefacts dans ses objectifs et met le plan nécessaire pour leur réalisation.

- Quand des artefacts sont réalisés, le Superviseur change ses croyances et son plan en utilisant le raisonnement en avant et peut anticiper l'exécution des activités dont les pré-conditions sont satisfaites.

Les Agents Superviseur et les agents Rôle interviennent dans la réalisation d'une activité, essentiellement, au niveau de l'assistance en particulier du contrôle.

b. Scénario de fonctionnement par demande de réalisation des artefacts

L'utilisateur spécifie à chaque fois le diagramme (ou tout simplement un artefact) qu'il veut réaliser et le système invoque les agents qui vont collaborer pour sa réalisation.

Supposant que l'état d'exécution du système est la construction du diagramme de cas d'utilisation, c'est-à-dire, l'agent rôle en exécution est AnalysteSystème et l'agent activité en exécution est StructurerCasd'utilisation. L'utilisateur veut exécuter le diagramme de collaboration, il envoie un message à l'agent Superviseur (m1).

L'agent Superviseur vérifie si le diagramme est dans son plan d'exécution, alors il répond directement à l'utilisateur et continue son exécution (m2).

Sinon, le Superviseur diffuse le message (m3) aux agents rôles s'informant sur celui dont un de ses agents activités peut accomplir cette tâche (la réalisation du diagramme de collaboration). Les agents Rôle diffusent de leur part le message aux agents Activité (m4). L'agent Activité concerné répond au message (m5).

Ensuite, le Superviseur met l'agent Activité concerné dans ses objectifs et en utilisant le raisonnement en arrière il maintient dans son plan les agents rôle suivants : L'agent Spécificateur, l'agent ExaminateurCasd'utilisation et l'agent AnalysteCasd'utilisation ainsi que les agents activité et les artefacts dont les agents rôle sont responsables et qui mènent à l'exécution de cet agent. Le superviseur lance ce workflow d'agents, et ils vont tous collaborer pour la réalisation de ce diagramme.

On peut modéliser cette conversation par un automate à états finis (voir figure)

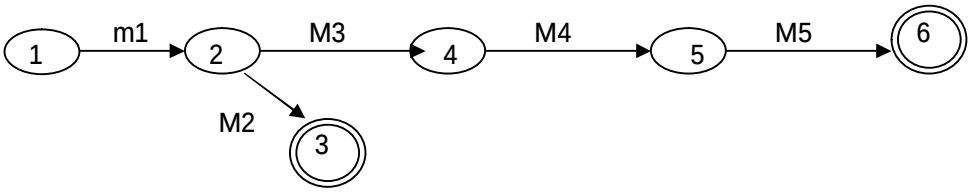

Figure 46 : Automates à état finis pour le scénario de communication b.

c. Scénario de fonctionnement par demande d'adoption d'un rôle

L'utilisateur lance à chaque fois un message pour adopter un rôle et le système invoque les agents appropriés pour son exécution.

Le développeur veut adopter un rôle, par exemple, celui du concepteur. L'Agent Interface informe l'agent Superviseur, ce dernier vérifie si l'agent peut être exécuté :

- S'il est dans son plan d'exécution alors, il lui envoie un message pour commencer l'exécution. L'utilisateur envoie un message à l'agent rôle concerné qui commence l'exécution en lançant le workflow des agents Activités dont il est responsable.

- Sinon, il lui envoie un message de refus et d'explication et utilise le raisonnement de chaînage en arrière pour planifier tous les rôles et les activités qui mènent à son exécution. Il envoie ensuite à l'utilisateur un message lui indiquant les chemins possibles qui mènent à ce rôle.

Conclusion Générale Et Perspectives

Dans ce mémoire, nous nous sommes focalisés sur l'étude de deux domaines différents qui sont les ateliers de génie logiciel et le paradigme agent dans l'objectif final est de proposer un atelier de génie logiciel basé agents et de concevoir le système multi agents permettant de le mettre en œuvre.

L'étude du paradigme agents nous a permis de comprendre des concepts tels que la communication, la collaboration et la coopération entre les agents.

L'étude des ateliers de génie logiciel nous a permis de découvrir le monde des ateliers de génie logiciel, ses différents outils et ses différentes méthodologies.

Nous avons constaté le grand intérêt accordé par la communauté des chercheurs aux ateliers CASE. Le nombre d'outils qui existent et qui ne cesse d'augmenter avec le nombre de recherches qui ont été faites et qui se poursuivent encore et dont chacune aborde un domaine particulier de CASE, montrent cette importance. Ces recherches visent améliorer la performance de ces outils pour les rendre plus flexibles, leur intégration plus facile et leur participation dans le processus du développement du logiciel plus active et tout cela dans le but d'aider les développeurs pour être plus efficace dans la réalisation de leurs projets et produire des logiciels de meilleure qualité.

Une des disciplines nécessaire pour l'outil CASE est le processus logiciel suivi. Le suivi stricte du processus logiciel paralyse la créativité du développeur et rend les outils rigides. L'assistance active et intelligente au processus logiciel rend les outils plus actifs et dynamiques en laissant plus d'espace au développeur pour s'occuper de la créativité et de l'organisation de son travail. Dans ce cadre, nous avons essayé de proposer un système multi agents qui va assister le développeur pendant tout le processus de son développement ou de sa modélisation. Dans la réalisation pratique, nous avons choisit un AGL de modélisation UML qui suit la méthodologie Rational Unified Process (RUP).

Pour atteindre cet objectif, nous avons commencé par décrire la méthodologie RUP, puis nous avons utilisé la méthode MASE pour faire l'analyse et la conception de notre SMA. Durant l'analyse, une base d'ontologies qui contient deux ontologies, de domaines et de

tâches a été conçue. L'ontologie de domaine, représente la description du domaine, c'est à dire les concepts clés de RUP utilisés par les agents du système. Et l'ontologie de tâches décrit quoi réaliser et comment le réaliser. Elle décrit les tâches du processus générique RUP et leur séquencement logique suivant le processus RUP. A la fin de la conception, nous avons déduit une architecture multi agents de l'outil qui permettra d'assister le développeur dans sa modélisation. Chaque agent dispose des règles et des connaissances pour guider le développeur en lui donnant des recommandations ou en prenant des initiatives. Les agents interagissent entre eux en collaborant par communication suivant le processus afin de réaliser la modélisation de logiciel.

Pour finir, nous pouvons dire que ce travail ne représente qu'un squelette d'un AGL qui nécessite, dans les perspectives:

- D'être construit à partir des nœuds. C'est-à-dire, à partir des agents Activités et Rôle.
- une généralisation en le développant en tant qu'une application distribuée pour supporter un groupe de développeurs qui peuvent coopérer sur un même projet et mettre à jour les artefacts concurremment dans un réseau.

- D'enrichir la capacité des agents par d'autres stratégies d'assistance intelligente, comme le raisonnement à partir de cas, dans la réutilisation des artefacts au lieu d'en produire à nouveau …etc.

Glossaire

AGL Atelier de génie logiciel

CASE Computer Aided Software Engineering

ECMA European Computer Manufacturers Association GL Génie logiciel

MASE Multi Agent Software Engineering UML Unified Modelling Langage

OMCP Object Management Creation Process

OMG Object Management Group

OO Orienté Objet

RUP Rational Unified Process

SMA Système Multi Agents

SPEM Software Process Engineering Method

Références

[1] Limayem, M. ; Khalifa, M. & Chin, W. W. "CASE Tools Usage and Impact on System Development Performance". ***Journal of Organizational Computing and Electronic Commerce, 14***(3), (2004), 153-174.

[2] Banker, R. D. & Kauffman, R. J. Reuse and Productivity in Integrated Computer-Aided Software Engineering: An Empirical Study. ***MIS Quarterly, 15***(3), (1991) 375-401.

[3] Church, T., & Matthews, P. ***An Evaluation of Object-Oriented CASE Tools: The Newbridge Experience.*** Paper presented at the The 7th International Workshop on Computer Aided Software Engineering, (1995, July 10-14), Toronto.

[4] Orlikowski, W. J. ***Division among the Ranks: The Social Implications of CASE Tools for System Developers*** Paper presented at the The Tenth International Conference on Information (1989).

[5] Patrick Jaulent : "Génie logiciel : Les méthodes", Editeur Armand Colin, Paris (1994).

[6] I. Sommerville : "Le génie logiciel", Deuxième édition (1991).

[7] C. Solnon : « Atelier de génie logiciel », cours, (1996) « http://www710.univ-lyon1.fr/~csolnon/agl.html »

[8] R. Conradi, C. Fernstrom, A. Fuggetta, "Concepts for evolving software processes", in: Finkelstein et al. [5], Ch. 2, pp. 9–31. (1994).

[9] Claude Albert MOGHOMAYE, "Réalisation d'un système multi-agents d'assistance à la modélisation ». Mémoire d'ingénieur de conception en informatique. Ecole Polytechnique de Yaoundé (2003).

[10] Jean-Marc Jézéquel, Noël Plouzeau, Yves Le Traon « Développement de logiciel à objets avec UML ». IFSIC _ Université de Rennes I, (2005).

[11] Terry Quatrani, "Modélisation UML avec Rational Rose 2000". Edition Eyrolles, (2000), Pp 6-9.

[12] Kimberly Stepien Oakes, Dennis Smith, "Guide to CASE Adoption", Ed Morris, (1992).

[13] Noran. R. J., "Working together to integrate case". ***IEEE Software***, vol. 12, (1999), pp. 12–16.

[14] Waman S. Jawadekar, « Software Engineering, principles and practice ». The Mac Graw-Hill companies (2004). Forth edition (2006).

[15] Portable Common Tool Environment - Abstract Specification. Technical Report ECMA

Standard 149, ECMA, Geneva, December (1990).

[16] John W. Satzinger, Robert B. Jackson, Stephen D. Burd, M. Simond, M. Villeneuve, "Analyse et conception de systèmes d'information" : Ch.3, pp. 94 .Les éditions Reynald Goulet Inc. (2003).

[17] Forte, G. and McCully, K., CASE Outlook: Guide to Products and Services, CASE Consulting Group, Lake Oswego, Oregon, (1991).

[18] David Sharon, "Integrating CASE tools - computer-aided software engineering software integration standards - Forum: Open Files – Column Software Magazine, (August 1993).

[19] Wassermann, A., "Tool integration in software engineering environments". International workshop on Environmrnts, Berlin, Springer- Verlag, (1990).

[20] Thomas, I. & Nejmeh, B. "Tool Integration in a Software Engineering Environment". Technical Report SESD-91-11, Revision 1.1, Hewlett-Packard.

[21] M. R. Cagan. The HP SoftBench Environment: An Architecture for a New Generation of Software Tools. ***Hewlett-Packard Journal,*** 41(3), 36-47, (June 1990).

[22] Reiss, S. P. "Connecting tools using message passing in the FIELD environment",IEEE Software, 7(4), (July 1990), 57-67.

[23] A. W. Brown. "Control integration through message-passing in a software development environment". ***Software Engineering Journal***, 8(3), (May 1993), 121-131.

[24] Belkhatir N., Estublier, J. and Melo, W. "ADELE-TEMPO : An Environment to Support Process Modelling and Enaction", Chapter 8, pp 187-222. Advanced Software Development Series. John Wiley & Sons Inc, (1994).

[25] Canals Gérome, Boudjlida Nacer, Derniam Joan Claudo, Godart Claude, Lonchamp Jacques « Alf : A framework forbuilding Process Centered Software Engineering Environments », (1992).

[26] G. Valetto and G. E. Kaiser "Enveloping sophisticated tools into computer-aided software engineering environments". In ***7th International Workshop on Computer-Aided Software Engineering***, pages 40-48, (1995).

[27] Montangero, C. and Ambriola, V. " OIKOS : Constructing Process-Centred SDEs, Chapter 6, pp 131-151. Advanced Software Development Series. John Wiley & Sons Inc, (1994).

[28] Alfonso Fuggetta "A classification of CASE technology ». IEEE, Computer Society Press Los Alamitos, CA, USA ***Computer,*** 26 (12), (1993), pp 25–38.

[29] http://www.win-design.com/fr/index.htm

[30] http://www.powerdesign.com/

[31] http://www.sybase.com/products/internetappdevtools/powerbuilder

[32] Oracle Designer: ***www.oracle.com/technology/products/designer***

[33] Rational Suite Analyst Studio:

ftp://ftp.software.ibm.com/software/rational/docs/v2002/rs_start_analyst_studio.pdf

[34] http://www-01.ibm.com/software/rational/

[35] Philippe Kruchten, "Rational Unified Process: An Introduction". The Third Edition. Publisher: Addison Wesley, Pub Date: December 19, 2003. ISBN : 0-321-19770-4.

[36] http://www.objecteering.com

[37] Kevin Compton, Yuri Gurevich, James Hugins, Wuwei Shen, "An Automatic Verification Tool for Uml", (Février 2000).

[38] http://www.pcsoft.com/windev

[39] http://www.sybase.com/products/internetappdevtools/powerbuilder

[40] http://www.oracle.com/ip/develop/ids/editions.html

[41] Safe Build: www.safebuild.com

[42] Rational Suite Development Studio:

http://www-01.ibm.com/software/awdtools/suite/dstudio/unix/

[43] http // argouml.tigris.org

[44] Rational Software, "IBM Rational Rose XDE Developer". G507-1041-00

[45] Visible Analyst: http://www.visible.com/Products/Analyst/index.htm

[46] Enterprise Architect v5.0 : http://www.sparxsystems.com.au/products/index.html

[47] IBM Software Group, "Modélisation : Rational Software Modeler et Rational Software Architect".Tech Software. Ibm corporation 2008.

[48] C. Rolland, C. Souveyet, M. Moreno, "***An Approach for Defining Ways-of-Working***", Information System Journal, (1995).

[49] S. Si-Said, C. Rolland, G. Grosz, "***MENTOR: A Computer Aided Requirements Engineering Environment"***, Proceedings of the International Conference on Advanced Information Systems Engineering (CAISE), Lecture Notes in Computer Science 1080, pp 22-43, (1996).

[50] J. Lonchamp, "CPCE: a kernel for building flexible collaborative process-centered environments," pp.95, Software Engineering Environment Conferences, (1995).

[51] Lonchamp Jacques, Seguin François, « The Argo Project », Crin : CNRS, (1997).

[52] Canals Gérome, Boudjlida Nacer, Derniam Joan Claudo, Godart Claude, Lonchamp Jacques, « Alf : A framework forbuilding Process Centered Software Engineering Environments », (1992).

[53] http / / www.jaczone.com

[54] Romi Satria Wahono, "OOExpert: An Agent Based System for Identifying and Refining Objects from Software Requirements Based on Object Based Formal Specification" ***The-12th Indonesian Scientific Meeting", Osaka University, September 6-7, 2003. Indonesian Student Association in Japan,*** Indonesian Institute of Science (LIPI), Department of Information and Computer Sciences, Saitama University.

[55] Dong Liu, "Automating Transition from Use Cases to Class Model", Thèse de mastère de science, Département de génie informatique et éléctrique. Université de CALGARY, Juillet 2003.

[56] Xinpei Zhao, Keith Chan, Mingshu Li, "Applying Agent Technology to Software Process Modeling and Process-Centered Software Engineering Environment".

[57] ***Imed Jarras et Brahim Chaib-draa***, « Aperçu sur les systèmes multi agents », Série Scientifique s-67, Montréal, (2002).

[58] Jacques Ferber, « Les Systèmes Multi Agents: vers une intelligence collective », Inter édition, pp 12-25, (1999).

[59] B. Chaib-draa, "Industrial applications of distributed AI". ***Communications of the ACM***, 38(11), 49-53, (1995).

[60] Briot, J.P. & Demazeau Y. « ***Systèmes Multi-Agents. Principes et architectures »***. Collection IC2. Hermes Science Publications, (2001).

[61] Hunhns, M. & Singh, M. P. "***A Multiagent Treatment of Agenthood"***. Applied Artificial Intelligence: An International Journal, Volume 13, No. 1-2, January-March 1999, 3-10.

[62] Franklin, S., Graesser, A. "***Is it an Agent, or just a Program? A Taxonomy for Autonomous***

Agents". Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages. Springer-Verlag, (1996).

[63] Sycara, K. "***Multi-Agent Systems"***. AI Magazine, Summer 1998, 85-92.

[64] Wooldridge, M. & Jennings, N. "Agent Theories, Architectures, and Languages: a Survey. Intelligent Agents", Wooldridge and Jennings Eds, 1-22. Springer-Verlag, (1995).

[65] RUSSELL S. J., NORVIG P., ***Artificial Intelligence. A Modern Approach***, Prentice-Hall, Englewood Cliffs, (1995).

[66] M. Bratman. "***Intention, plans, and practical reason"***. Harvard University Press, (1987).

[67] WOOLDRIDGE M., « Intelligent Agents », WEISS G., Ed., ***Multiagent Systems : A Modern Approach to Distributed Artificial Intelligence***, Chapitre 1, pp. 27-77, The MIT Press, Cambridge, MA, (1999).

[68] B. Chaib-draa and P. Levesque. "Hierarchical models and communication in multi-agent environments". In ***Proceedings of the Sixth European Workshop on Modelling Autonomous Agents and Multi-Agent Worlds (MAAMAW-94). P***ages 119-134, Odense, Denmark, August 1994.

[69] B. Chaib-draa. Distributed Artificial Intelligence: An overview. In A. Ken, J. G. Williams, C. M. Hall, and R.Kent, editors, ***Encyclopedia Of Computer Science And Technology***, volume 31, pages 215-243. Marcel Dekker, Inc, (1994).

[70] B. Chaib-draa. "Interaction between agents in routine, familiar and unfamiliar situations". ***International Journal of Intelligent and Cooperative Information Systems***, 1(5):7-20, (1996).

[71] B. Chaib-draa and P. Levesque. Hierarchical model and communication by signs, signals and symbols in multi-agent environments. ***Journal of Experimental and Theoretical AI (JETAI),*** 8:7-20, (1996).

[72] I. A. Ferguson. ***TouringMachines: "An Architecture for Dynamic, Rational, Mobile Agents"***. PhD thesis, Clare Hall, University of Cambridge, UK, November 1992. (Also available as Technical Report No. 273, University of Cambridge Computer Laboratory).

[73] K. Fischer, B. Chaib-draa, H. J. Müller, J. P. Müller, and M. Pischel. "A simulation approach based on negotiation and cooperation between agents". ***IEEE Trans. on Systems, Man, and Cybernetics***, 29(4), 1999, pp. 531-545.

[74] M. P. Georgeff and A. L. Lansky. “Reactive reasoning and planning”. In ***The Proceedings of AAAI-87***, pages 677-682, Seattle, 1987.

[75] JENNINGS N., SYCARA K., WOOLDRIDGE M., « A Roadmap of Agent Research and Development », ***Autonomous Agents and Multi-Agent Systems***, vol. 1, n°1, p. 7 - 38, July 1998.

[76] Aloys Mbala Hikolo : « Analyse, conception, spécification et développement d'un système multi-agents pour le soutien des activités en formation à distance », thèse de doctorat de l'Université de Franche-Comté, France, (2003).

[77] B. Moulin and B. Chaib-draa. An overview of distributed artificial intelligence. In G. M. P. O'Hare and N. R. Jennings, editors, ***Foundations of Distributed AI***, pages 3-54. John Wiley & Sons: Chichester, England, (1996).

[78] FRÉDÉRIC SOUCHON : « SaGE, un Système de Gestion d'Exceptions pour la programmation orientée message : Le cas des Systèmes Multi-Agents et des Plates-formes à base de Composants Logiciels », Thèse de doctorat en informatique, Université des Sciences et Techniques du Languedoc, (2005).

[79] HEINZE C., SMITH B., CROSS M., « Thinking Quickly : Agents for Modeling Air Warfare », ***Proceedings of the 9th Australian Joint Conference on AI (Ai’98)***, Australie, 1998.

[80] CHAIB-DRAA B., JARRAS I., MOULIN B., « Systèmes multiagents : Principes généraux et applications », BRIOT J. P., DEMAZEAU Y., Eds., ***Agent et systèmes multiagents***, Hermès, (2001).

[81] J. Ferber. « Les Systèmes Multi-Agents : vers une intelligence collective », pages 68—89, (1995).

[82] A. Drogoul. « De la simulation multi agent à la résolution de problèmes. Une étude de l’émergence de structures d’organisation dans les systèmes multi agents », (1993).

[83] L. Gasser, C. Braganza, and N. Herman. “Implementing Distributed Artificial Intelligence Systems using MACE”. IEEE. Conference on Artificial Intelligence Applications, pages 119—152, (1987).

[84] R. Davis and R. Smith. “Negociation as a metaphor for Distributed Problem Solving”. Artficial Intelligence, pages 63—109, (1983).

[85] N. R. Jennings, “Controlling cooperative problem solving in industrial multiagent systems using joint intentions”. ***Artificial Intelligence***, 74(2), (1995).

[86] Brenner Walter, Zarnekow Rüdiger, Wittig Harmut, “***Intelligent Software Agents: Foundations and Applications”,*** Springer-Verlag, Berlin, (1998).

[87] Bouron T., « ***Structure de communication et d'organisation pour la coopération dans un univers multi-agents »,*** Université Paris VI, (Thèse 3ème cycle, Informatique), Novembre 1992.

[88] Brown, S., Santos, E., et Banks, S., "Utility theory-based user models for intelligent interface agents". In Canadian Conference on AI, (1998).

[89] Penny. Nii. "Blackboard systems : The blackboard Model of problem Solving and the Evolution of the Blackboard Archtectures",AI Magazine, 1986, pp.82-106.

[90] J. Ferber, « Les systèmes multi-agents : un aperçu général », techniques et science informatique, vol. 16-N° 8, 1997, pp.979-1012.

[91] Finin, T., Labrou, Y., & Mayfield, J.. "KQML as an agent communication language". In J. M. Bradshaw (Ed.), 1997, ***Software agents*** (pp. 291-316). CA: AAAI Press.

[92] T. Finin and R. Fritzson. " KQML: a language and protocol for knowledge and information exchange". In ***Proceedings of the Thirteenth International Workshop on Distributed Artificial Intelligence,*** pages 126-136, Lake Quinalt, WA, July 1994.

[93] P. R. Cohen and H. J. Levesque. "Communicative actions for artificial agents ». In ***Proceedings of the First International Conference on Multi-Agent Systems (ICMAS- 1)***, San Francisco, CA, pages 65-72, (1995).

[94] http://cormas.cirad.fr

[95] Humphrey, W.S. and M. Kellner, "Software Process Modeling: Principles of Entity Process Models, ***Proc. 11th. Intern. Conf. Software Engineering***, IEEE Computer Society, Pittsburgh", PA, 331-342, (1989).

[96] N. Boudjlida, F. Charoy, J.C. Derniame, and C. Godart, "Modeling of Software Process Assistance". In ***CASE'89 - 3rd Workshop on Computer-Aided Software Engineering***, London, July 1989.

[97] Julia Nitschke , Hartmut Wandke, " Human Support as a model for Assistive Technology". ***Humboldt Universität Berlin Institute for Psychology. Department for Engineering Psychology.*** Proceedings of the OZCHI, 2000.

[98] Stan Jarzabek and Riri Huang, "The Case for User-Centered CASE Tools", COMMUNICATIONS OF THE ACM, Vol. 41, No. 8, (August 1998).

[99] S.A. Deloach, M.F. Wood and C.H. Sparkman, "Multi Agent Systems Engineering (MASE)", ***International Journal on Software engineering and knowledge engineering,*** 2001, vol. 11 (3), pp. 231-258.

[100] S.A. DeLoach, and W.H. Oyenan, "An Organizational Model and Dynamic Goal Model for Autonomous, Adaptive Systems". ***Technical Report No. MACR-TR-2006-01***, Kansas State University, Mars 2006.

[101] Thomas R. Gruber, "***Towards Principles for the Design of Ontologies Used for Knowledge Sharing*** in Formal Ontology in Conceptual Analysis and Knowledge Representation", Kluwer Academic Publishers, (1993).

[102] Object Management Group. "Software Process Engineering Metamodel Specification". Version 1.1, January 5 2005. formal/05-01-06.

[103] www.omg.org/technology/documents/formal/uml.htm