Table des matières

Liste des figures

Liste des tables

Introduction Générale

1. Contexte

Le développement de logiciels, lorsqu'il est mené à terme avec succès, se termine par la livraison d'un logiciel qui devrait satisfaire les exigences initiales du client. Par la suite, après sa mise en service, le logiciel devra évoluer pour répondre aux nouveaux besoins d'un environnement en constante évolution. De plus, c'est pendant l'opération du logiciel que l'organisation découvrira des anomalies, ainsi que de nouveaux besoins d'affaires qui feront surface, et qu'il faudra, après un certain nombre d'années, en modifier la plate-forme technologique. Enfin, il est bon de souligner que le cycle de vie de la maintenance ne devrait pas débuter lors de la mise en service du logiciel, mais, en réalité, bien avant par une participation active des responsables de la maintenance tout au long du processus de développement, quand c'est possible.

La maintenance du logiciel est un des dix grands thèmes de connaissance reconnus comme faisant partie de la discipline de l'ingénierie du logiciel, tel que cela a été décrit dans ISO TR 19759, ainsi que dans le guide au corpus des connaissances en génie logiciel [Abran, 2005] (Software Engineering Body of Knowledge – SWEBOK). Ces documents confirment ce qui avait déjà été identifié par Victor Basili en 1996 : « La maintenance du logiciel est un domaine spécifique du génie logiciel, et conséquemment, il est donc nécessaire de se pencher sur des processus et des méthodologies qui tiennent compte des caractéristiques spécifiques de la maintenance du logiciel » [Basili, 1996].

Les deux activités les plus coûteuses dans la maintenance de logiciels sont la compréhension du problème qui est généralement liée à la compréhension du logiciel maintenu, et la maîtrise de la totalité des effets de propagation des changements proposés [BAR 95]. En effet, un petit changement peut avoir des effets considérables et inattendus sur le reste du système. Le danger encouru lors de la modification, réside dans cette conséquence de l'impact d'un changement obtenu. La modularité en oriente objet, adéquatement utilisée, limite les effets relatifs aux changements. Néanmoins, ils sont subtils et difficiles à localiser. Pour toutes ces raisons, les concepteurs et les mainteneurs ressentent le besoin de mécanismes pour analyser les changements et connaître comment ils sont propagés dans le reste du système. Ce processus s'appelle l'analyse de l'impact du changement.

2. Problématique

La maintenabilité a été déterminée par la manière avec laquelle un système supporte les tâches de maintenance dont l'analyse d'impact et les tests de régression [Briand, 1999]. Avec l'évolution des systèmes logiciels, un flot important de changement doit être pris en considération ainsi que leur propagation sur le reste du système. Réussir à modifier le logiciel de façon disciplinée toute en maintenant son fonctionnement et son intégrité avec un coût raisonnable nécessite une analyse de l'impact du changement avant son implémentation. En effet, l'équipe de maintenance doit être en mesure de fournir des réponses aux questions suivantes :

1. De quel type de changement s'agit-il ?
2. Quelle est l'étendue du changement ?
3. Quelles sont les ressources nécessaires pour faire le changement : temps, personnels, budget et plannings ?
4. Quel est le degré de difficulté à implanter le changement ?
5. Quel est le risque encouru pour accomplir le changement ?

De telles informations aident le responsable de la maintenance dans l'évaluation du coût de la proposition de modification et représentent une source cruciale pour les gestionnaires afin de planifier le changement, et déterminer les ressources nécessaires pour l'implémenter.

Dans la gestion des projets logiciels plusieurs changements sont proposés pour résoudre le même problème et satisfaire le même besoin, l'utilisation de l'analyse d'impact de

changement permet de choisir le changement adéquat en prenant en considération deux aspects : la nature de changement, et son impact sur le reste du système ; vu que plus un changement propage plus le coût de la maintenance augmente.

3. Motivation

L'objectif de ce mémoire est d'améliorer la maintenance des systèmes orientés objet, et d'intervenir plus précisément dans la tâche de l'analyse de l'impact de changement. Nous visons principalement la réduction de l'effort ainsi que le coût de la maintenance. La réduction de cet effort peut être accomplie par la réduction du temps entre la proposition du changement, son implantation et sa réalisation, tout en assurant la qualité du système. A cet effet, nous proposons une approche de validation d'un modèle d'impact de changement pour des systèmes orientés objet en utilisant des systèmes multi-agents.

4. Organisation du mémoire

Ce mémoire présente un travail transversal qui se situe au carrefour de deux domaines à savoir : la maintenance de logiciels et les systèmes multi-agents (SMA). Il est organisé comme suit : le **chapitre1** est une introduction sur la maintenance des logiciels, le **chapitre 2** présente l'état de l'art sur l'analyse d'impact du changement et le modèle d'impact de changement détaillé, suivi par le **chapitre 3** qui est un état de l'art sur les systèmes multi agents. Le **chapitre4** détaille notre approche proposée pour l'étude d'impact de changement et les résultats obtenus.

Enfin, nous clôturons ce mémoire par une conclusion générale dans laquelle nous résumons ce qui a été réalisé dans le cadre de ce travail, puis nous signalons quelques perspectives majeures.

Chapitre I

Introduction sur la maintenance des logiciels

1. Introduction

La maintenance est la dernière phase du cycle de vie d'un logiciel. Elle est définie comme étant le processus de modification d'un logiciel en opération pour lui permettre de toujours satisfaire aux spécifications actuelles et futures [IEEE, 1993] , elle représente la phase la plus importante et la plus coûteuse, 60% à 80% du coût total du cycle de vie du logiciel est dépensé durant la phase de maintenance [Munro, 1998]. Ceci est dû entre autres :

- Au personnel inexpérimenté, non familier avec l'application et parfois peu motivé,
- Aux programmes non structurés, n'obéissant pas aux standards,
- Aux modifications qui génèrent des fautes,
- A la dégradation de la structure du système suite aux modifications non planifiées apportées au système par des gens différents sur des plates-formes différentes sans documentation de leurs tâches,
- A une documentation obsolète, inadéquate, voire inexistante.

Plusieurs techniques, méthodes et outils destinés à cette phase ont vu le jour. On cite comme exemple : la réingénierie, la rétro ingénierie, la compréhension de programme, l'analyse d'impact, les tests de régression, la gestion de configuration de logiciel, la maintenance via le Web, etc.

2. Le cycle de vie d'un logiciel

Le cycle de vie d'un logiciel est l'ensemble des étapes du développement d'un logiciel, de la définition des besoins du client jusqu'à l'achèvement du logiciel en tant que produit commercial. L'objectif d'un tel découpage est de permettre de définir des jalons intermédiaires permettant la validation du développement du logiciel, c'est-à-dire la conformité du logiciel avec les besoins exprimés, et la vérification du processus de développement, où encore l'adéquation des méthodes mises en œuvre.

Le cycle de vie permet de détecter les erreurs au plus tôt et ainsi de maîtriser la qualité du logiciel, les délais de sa réalisation et les coûts associés.

Le cycle de vie du logiciel comprend généralement les activités suivantes :

- ✓ **Phase de spécification des besoins** : déterminer les besoins du logiciel (spécifications générales, spécifications fonctionnelles, spécifications d'interface). Les spécifications des besoins servent à définir ce que doit faire le logiciel et non comment il doit être fait.

- ✓ **Phase de conception du système et du logiciel :** En partant de la spécification des besoins, on décompose le système en deux parties : matériel et logiciel. C'est le processus qui consiste à présenter les diverses fonctions du système d'une manière qui permettra d'obtenir rapidement un ou plusieurs programmes réalisant ces fonctions.

- ✓ **Phase de réalisation et tests unitaires :** Lors de cette étape, on réalise un ensemble d'unités de programme écrites dans un langage de programmation exécutable. Les tests unitaires permettent de vérifier que ces unités répondent à leurs spécifications.

- ✓ **Phase de test du système :** On intègre les unités de programme et on réalise des tests globaux pour être sûre que les besoins logiciels ont été satisfaits**.** Le système est alors livré au client.

- ✓ **Phase de maintenance :** Une fois que le produit est accepté, il passe en phase de maintenance jusqu'à son retrait. Normalement (mais pas nécessairement) ceci est la plus longue étape du cycle de vie, l'activité de maintenance consiste à corriger les erreurs qui n'ont pas été découvertes lors des étapes antérieures du cycle de vie, à améliorer la réalisation des unités du système et à augmenter les fonctionnalités du produit au fur et à mesure que de nouveaux besoins apparaissent.

3. Maintenance des applications

La compréhension d'un programme représente l'une des tâches les plus importantes du processus de maintenance. Les personnes chargées de la maintenance épuisent 40 à 60% de leurs temps à lire le code et à essayer de comprendre sa logique. C'est une activité laborieuse qui augmente le coût de la maintenance [Pigoski, 1997]. En effet, l'équipe qui a développé le système n'est pas toujours celle qui assure sa maintenance, les documents de spécifications sont parfois incomplets et le code source constitue souvent la seule source d'informations fiable. La plupart des problèmes associés à la maintenance du logiciel sont causés par la méthode utilisée pour concevoir et développer le système. De plus, la maintenabilité est un facteur de qualité qui n'est pas incorporé dans le processus de développement. Pour faciliter la

maintenance et augmenter la maintenabilité des systèmes logiciels, il faut, entre autres, définir des standards de codage, de documentation et d'outils de tests dans la phase de développement du logiciel [Swebok, 2004], documenter l'évolution du logiciel et utiliser les techniques destinées à la maintenance.

3.1. Les types de maintenance

Généralement, on considère qu'il existe trois types de maintenance, dépendant de la nature de la modification : corrective, adaptative et perfective. Nous décrivons ces différents types de maintenance dans le paragraphe suivant :

Maintenance corrective : C'est le type de maintenance le plus évident. On s'assure qu'un logiciel en opération continue à satisfaire ses spécifications. En pratique, c'est l'activité de corriger les erreurs résiduelles qui auraient dues être découvertes lors de la phase de test. Comme on ne pourra jamais découvrir toutes les erreurs lors de la phase de test, ce type de maintenance est inévitable. Il ne constitue toutefois qu'environ 17% de l'effort total de maintenance [Lientz, 1978].

Maintenance adaptative : Ce type de maintenance consiste à modifier un logiciel en réponse à un changement intervenu dans son environnement logiciel ou matériel. Le standard IEEE la définit comme suit [IEEE, 1993] :"*modification of a software product performed after delivery to keep a computer program usable in a changed or changing environnement*". Ainsi, la portabilité d'un logiciel d'une plate-forme (système d'exploitation ou processeur) à une autre relève de cette catégorie de maintenance. La maintenance adaptative compte pour environ 18% de l'effort total de maintenance [Lientz, 1978].

Maintenance perfective : La maintenance perfective fait référence aux modifications d'un programme en vue d'augmenter ses performances ou ses fonctionnalités. Ce type de maintenance s'effectue généralement à la demande des utilisateurs qui veulent toujours avoir plus de fonctionnalités. C'est la principale activité de maintenance, elle constitue entre 60 à 70% de l'effort total de maintenance [Lientz, 1978] .

3.2. Activité de maintenance

Carma McClure [McClure, 1992] observe que l'activité d'un programmeur lors de la maintenance peut se ramener à trois tâches principales : compréhension du code, modification, et revalidation.

a. Comprendre le code et les changements à effectuer est une tâche essentielle pour modifier un programme. Le programmeur doit avoir une complète compréhension du programme, sinon le risque d'introduire des erreurs après modification est grand.

b. Le programmeur doit modifier le programme pour que ce dernier soit conforme à ses spécifications actuelles (maintenance corrective) ou nouvelles (maintenance adaptative ou perfective). Il doit s'assurer que ses modifications n'altèrent pas l'intégrité du système ou n'augmentent pas sa complexité. Il doit également faire attention aux effets de bord (introduction d'autres erreurs).

c. Après toute modification, un programme doit être re-testé (revalidé) pour s'assurer que les modifications ont été correctement effectuées et qu'il n'y pas eu d'effets de bord.

3.3. Problèmes de maintenance

La maintenance des logiciels se heurte à un certain nombre de difficultés. Martin et Osborne dans **[**Martin et Osborne, 1985**]** ont relevé quelques facteurs qui font de la maintenance une activité délicate :

- Mauvaise conception des programmes.
- Utilisation de code non-structuré
- Utilisation de plusieurs langages de programmation dans un même programme (logiciel).

Mais une difficulté majeure est celle que pose la compréhension d'un programme (program understanding).

La compréhension de programme comprend la lecture de la documentation, l'analyse du code source et l'identification des changements à effectuer. Selon McClure [McClure, 1992]**,** des trois activités de la maintenance, la compréhension de programme est celle qui consomme le plus de temps (voir figure I.1). Une autre difficulté liée à la compréhension

et la modification d'un système est l'évaluation de l'impact d'un changement et des erreurs qu'il peut introduire dans le code.

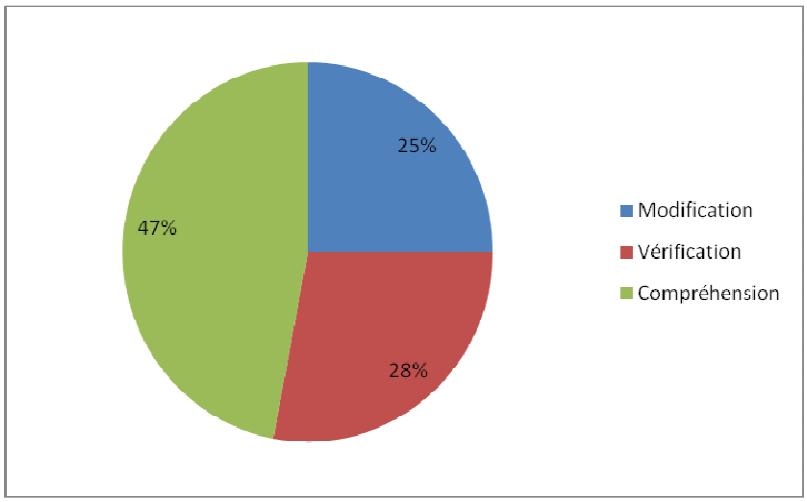


Figure I.1 Répartition du temps entre les activités de maintenance

3.4. Techniques de maintenance

Dans le domaine du logiciel, une plus grande partie de l'effort est accordée à la phase de développement. L'objectif est de livrer un produit qui satisfait les besoins dans les délais et dans les limites du budget sans se préoccuper de la qualité du logiciel. Avec l'augmentation des coûts de la maintenance, de nombreuses techniques ont vu le jour. Ci-dessous, nous présentons brièvement quelques-unes de ces techniques.

Rétro ingénierie : Généralement pour concevoir un logiciel, on passe de la conception au code. La rétro ingénierie prend le chemin inverse. Chikofsky et Cross [Chikofsky Elliot, 1990] ont défini la rétro ingénierie par: *"reverse engineering is the process of analyzing a subject system to identify the system's component and their interrelationships and create representations of the system in another form or at a higher level of abstraction"*. C'est le processus d'analyse d'un logiciel pour identifier ses composants et leurs relations dans le but de générer une représentation du système à un plus haut niveau d'abstraction. La redocumentation et la récupération de conception (Design recovery) représentent deux types importants de rétro ingénierie.

Réingénierie : Selon Chikofsky et Cross, la réingénierie consiste en : "*the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form* "[Chikofsky Elliot, 1990]. C'est un processus composé de l'analyse du système et puis sa reconstitution. Ces deux étapes sont effectuées par les techniques de rétro ingénierie et de"*Forward Engineering*". Cette dernière consiste à transiter d'une représentation de haut niveau obtenue par l'étape précédente pour produire un nouveau système. Certains auteurs ajoutent une troisième étape : la restructuration. Elle transforme la structure interne d'un logiciel sans changer de niveau d'abstraction et sans modifier ou ajouter de nouvelles fonctions. En outre, elle permet une meilleure compréhension du logiciel et facilite sa maintenance [Abran, 1995].

Analyse d'impact : Pratiquement, lorsqu'une proposition de changement parvient à l'équipe de maintenance, elle est transformée en terme logiciel afin de décider si la proposition doit être retenue ou rejetée. Une fois accepté, il faut localiser l'origine de l'anomalie. L'analyse d'impact du changement s'impose à cette étape afin de (i) déterminer tous les composants du système qui doivent changer en conséquence du changement initial, (ii) estimer les ressources nécessaires pour implémenter le changement. Ce résultat permet aux gestionnaires de prendre une décision sur la meilleure solution à mettre en œuvre ou annuler la requête de changement.

4. Conclusion

Dans ce chapitre, nous avons présenté la maintenance, ses différents types ainsi que ses techniques et les problèmes rencontrés durant cette phase, qui est une phase importante dans le cycle de vie d'un logiciel.

Dans le chapitre suivant, nous abordons l'impact de changement qui est l'une des techniques souvent préconisée dans la phase de maintenance de logiciel.

Chapitre II

Analyse d'impact de changement

1. Introduction

Lors de la maintenance, prédire où et comment un changement impacte le reste des composants du système est une tâche difficile. Cette difficulté émane de la manière avec laquelle les modifications sont traitées. Effectuer des changements sans compréhension de leurs effets peut amener à une mauvaise estimation de l'effort, à prolonger les délais de livraison, à dégrader la conception du logiciel, à entamer la fiabilité du produit, voire le retrait prématuré du logiciel [Lee, 1998].

En effet, un changement peut apparaître simple, facile, nécessitant moins de temps, de budget et de personnel. Cependant, lors de son implémentation la personne chargée de la maintenance s'aperçoit que cette tâche est complexe, difficile, voire impossible dans certains cas. Ainsi, l'équipe de maintenance a besoin de moyens et de mécanismes permettant d'analyser le changement, de déterminer son impact sur le reste du système et d'étudier sa faisabilité. L'analyse d'impact du changement est née d'une telle préoccupation. L'analyse d'impact du changement est effectuée principalement lors de la maintenance des programmes, en vue d'évaluer les effets du changement sur le reste du système et de réduire le risque de s'embarquer dans des dépenses coûteuses. Cette évaluation englobe l'estimation des ressources humaines, financières, temps, effort et plannings nécessaires pour accomplir le changement.

2. Impact de changement

2.1. Analyse d'impact

L'impact est l'effet ou l'influence d'une chose sur une autre. Dans notre étude, l'impact est la conséquence d'un changement. L'analyse d'impact est l'étude de la portée de cet impact. L'analyse d'impact du changement est l'estimation des conséquences d'un changement sur le reste du système. Elle a été pratiquée de différentes manières pendant des années. Cependant, il n'y a pas de consensus sur sa définition [Arnold & Bohner, 1993].

Pfleeger et Bohner [Pfleeger, 1990] définissent l'analyse d'impact du changement comme : "*the evaluation of the many risks associated with the change, including effects on resources, effort, and schedule estimates*". Arnold et Bohner [Arnold & Bohner, 1993] définissent l'analyse d'impact par : "*impact analysis is the activity of identifying what to modify to accomplish a change or of identifying the potential consequences of a change*". Turver et Munro [Turver, 1994] la définissent par: "*the assessment of a change, to the source code of a*

module, on the other modules of the system, it determines the scope of a change and provides a measure of complexity". Bohner [Bohner, 1996] la définie par : "impact analysis identifies the consequences or ripple effects of proposed software changes".

Ces définitions adressent l'analyse d'impact selon différents points de vue. Pfleeger et Bohner dans leur définition mettent en relief l'évaluation de l'impact et s'intéressent à l'aspect gestion, tandis que les autres se concentrent sur l'estimation des conséquences que peut avoir un changement sur le reste du système. L'analyse d'impact est donc une tâche prédictive permettant de planifier les changements et d'identifier les conséquences de leurs propagations avant leurs implémentations. Cependant, avant d'implémenter le changement, on n'est pas en mesure de cerner l'ensemble des conséquences. C'est pourquoi la détermination de l'impact total d'un changement se fait en deux phases : lors de l'analyse du code source et lors de l'exécution du système après modification. Deux aspects du changement existent : quantitatif et qualitatif. Le premier consiste à compter le nombre d'éléments touchés et le deuxième détermine la manière avec laquelle ces éléments sont touchés [Cantave, 2001].

L'analyse d'impact peut se faire de différentes manières. Ci-après des exemples cités dans la littérature :

- Utiliser le "cross référence listings" pour voir quelles sont les autres parties du programme qui contiennent les références à la variable ou à la méthode en question;
- Utiliser le "program slicing" pour déterminer le sous-ensemble du programme qui peut affecter la valeur de la variable en question;
- Parcourir le programme par ouverture et fermeture de fichiers;
- Utiliser des relations de traçabilité pour identifier les artéfacts changés;
- Utiliser les systèmes de gestion de configuration pour suivre et trouver les changements;
- Consulter les conceptions et les spécifications pour déterminer la portée du changement.

2.2. Intérêts d'une analyse d'impact de changement

L'identification des impacts possibles avant d'effectuer le changement, nous permettra de réduire les risques d'entreprendre des changements coûteux. En effet, plus les problèmes liés à un changement sont découverts tard, plus leur coût augmente. Généralement, plusieurs changements sont proposés pour résoudre un même problème et satisfaire le même besoin. L'évaluation de l'impact de chaque changement permet de choisir le changement adéquat. Selon Michelle L. Lee [Lee, 1998], l'expérience a montré qu'effectuer des changements dans un code sans comprendre leurs implications peut conduire à une mauvaise estimation de l'effort à produire, à prolonger les délais de livraison, à dégrader la conception d'un logiciel, à entamer la fiabilité du produit et enfin peut amener au retrait prématuré du logiciel. Elle propose dans ses travaux une démarche constituée en 6 étapes :

- Convertir le changement proposé en des spécifications de changement,
- Extraire les informations à partir du code source et les convertir en une représentation interne,
- Calculer l'impact de changement pour le changement proposé (refaire les trois premières étapes pour d'autres changements),
- Donner une estimation des ressources basée sur des considérations telles que la taille et la complexité du système.
- Analyser le coût et les bénéfices des requêtes de changements
- et enfin, le responsable de la maintenance du projet devrait aviser ses superviseurs de l'implication de la requête de changement. Ces derniers décident d'autoriser ou d'annuler le changement.

Pour la gestion de projet logiciel, l'analyse d'impact permet une meilleure estimation des ressources, de l'effort et du temps associés à une activité de maintenance.
En identifiant l'impact potentiel d'une modification, on réduit le risque de s'embarquer dans des changements coûteux et imprévisibles.

Pour un programmeur, l'analyse d'impact peut avoir plusieurs objectifs dont l'évaluation de la complexité d'un code et la conduite des tests de régression. Si chaque modification d'une partie d'un code a un impact sur un grand nombre d'autres parties; cela indique une forte dépendance entre ces parties, un fort couplage et par conséquent une grande complexité.

L'analyse d'impact peut être utilisée pour diriger les tests de régression, i.e., pour déterminer les parties d'un programme qui nécessitent d'être re-testés après un changement. Les tests de régression sont une importante activité de maintenance et d'assurance qualité, visant à assurer que le comportement d'un programme continue, après modification, de satisfaire aux spécifications [Harroldet, 1994]. Pour économiser en effort, ces tests visent les parties d'un code qui ont été affectées ou pourraient être affectées par des changements, au lieu de re-tester le logiciel dans son intégralité. Durant la maintenance, pour un programme modifié, il est question de déterminer les classes qui méritent d'être re-testées. Re-tester toutes les classes d'un système s'avère trop long et trop coûteux, ne pas re-tester suffisamment de classes a une influence sur la qualité du logiciel et sur le degré de confiance qu'on peut lui accorder. L'analyse d'impact nous permet d'évaluer de façon satisfaisante les classes qui devront faire l'objet de nouveaux tests. On oriente le lecteur vers [Cantave, 2001] & [Kung, 1995] pour avoir une idée plus claire sur l'utilisation de l'analyse d'impact pour cet objectif.

L'analyse d'impact peut être utilisée pour étudier la stabilité des logiciels. Pfleeger et Bohner [Pfleeger, 1990] l'utilisent dans ce cadre et proposent un certain nombre de métriques pour mesurer la stabilité d'un code. Leurs travaux se basent sur un graphe de traçabilité (traceability graph), qui montre les interconnexions entre les éléments d'un code, les cas de tests, les documents de conception et les spécifications. Enfin, l'analyse d'impact peut être utilisée aussi pour étudier la changeabilité des systèmes orientés objets, c'est-à-dire voir leur capacité d'absorber des changements **[**Chaumun, 1998**]** & [Kabaili , 2002].

2.3. Travaux de recherche sur l'analyse d'impact

Divers études ont été réalisées sous différentes formes à propos de l'analyse d'impact de changement. Han [Han, 1997] a développé une approche pour calculer l'impact de changement sur les documents de conception et d'implémentation. Son approche ne considère pas les dépendances d'invocation. De plus, les impacts ne sont pas définis d'une façon formelle.

Kazman & al [Kazman, 1996] utilisent des scénarios pour étudier l'impact de changements sur l'architecture du système, mais ils n'ont pas défini de modèle d'impact. Lindvall [Lindvall, 1999] a identifié les changements les plus communs et fréquents de C++ afin que les modèles de changement puissent être conçus pour aider les développeurs à prévoir les futurs besoins.

Dans [Antoniol, 1999], les auteurs ont prédit la taille des systèmes OO évoluant dans le temps en se basant sur l'analyse des classes impactées par une demande de changement. Ils ont prédit la taille de changements en termes de lignes de code ajoutées/modifiées.

Kiran & al [Kiran, 1997] ont comparé l'impact de changement dans le paradigme fonctionnel et le paradigme OO, mais n'ont pas proposé un modèle d'impact. Ils ont employé des programmes développés par des étudiants et ont défini des ensembles de changements qui ont été implémentés par la suite. Les résultats suggèrent que l'effort de maintenance est moins important dans le paradigme OO que dans le paradigme fonctionnel. En particulier, l'impact de l'ensemble de changements considérés est plus localisé dans le paradigme OO que dans le paradigme fonctionnel.

Kung & al [Kung, 1994] & [Kung, 1995] intéressés par les tests de régression, ont développé un modèle d'impact de changements basé sur les trois liens : héritage, association et agrégation. Ils ont aussi défini des algorithmes formels pour calculer toutes les classes impactées incluant des effets de propagation (ripple effects). Li et Offutt [Lee, 1996] pour examiner les effets d'encapsulation, d'héritage et de polymorphisme sur l'impact de changements, ont proposé des algorithmes pour calculer l'impact complet de changements faits dans une classe donnée. Quelques changements, par exemple dans les liens d'héritage, aussi bien que certains liens, comme l'association et l'agrégation, n'ont pas été entièrement couverts dans leurs algorithmes.

Dans [Cantave, 2001], l'analyse d'impact a été faite dans l'objectif de réduire les coûts et la durée des tests de régression. Aucun modèle n'a été défini dans cette étude. L'analyse a été faite à partir du graphe de dépendance extrait de la base de données (repository).

Briand & al dans [Briand, 1999], ont essayé de voir si les mesures de couplage capturant toutes sortes de collaboration d'une classe, peut aider à faire l'analyse d'impact de changement. La stratégie adoptée dans cette étude est différente des autres stratégies dans la mesure où elle est purement empirique**.** Cette étude a montré qu'un certain nombre de métriques de couplage liées à l'agrégation, et au couplage d'invocation sont reliées à la propagation de modification. Cela permet de localiser l'analyse de dépendance et de réduire l'effort d'analyse d'impact.

Dans [Chaumun, 1998] et [Kabaili , 2002] , un modèle de changements et d'impact de changements, a été défini au niveau conceptuel pour étudier la changeabilité des systèmes orientés objet. L'approche adoptée utilise des propriétés caractéristiques de la conception des systèmes orientés objet, mesurées par des métriques afin de prédire la changeabilité.
Le modèle a été raffiné par la suite en C++ afin d'évaluer la changeabilité. Ensuite, plusieurs études empiriques ont été faites pour tester la relation entre les métriques de conception et les impacts évalués par ce modèle. Ces expérimentations ont montré que le couplage de classes est un bon indicateur de changeabilité. En revanche, la cohésion de classes telle que définie par les métriques actuelles, ne peut être utilisée pour prédire la changeabilité.

Dahane [Dahane, 2011] a proposé une approche de vérification empirique d'un modèle d'impact de changement dans un système à objets afin de répondre à la problématique d'analyse d'impact de changement. Des changements ont été effectués sur divers systèmes (7 systèmes) issus de domaines différents. Ensuite une comparaison des résultats d'impacts de changements déduits par le modèle d'impact en question avec ceux obtenus manuellement par expérimentation a été faite.

Dans [Dinedane, 2011], l'auteur a proposé une approche d'aide à la décision pour les gestionnaires de maintenance des logiciels orientés objet dans leur choix de la meilleure solution parmi plusieurs proposées en utilisant la méthode ELECTRE III et en se basant sur certaines métriques de couplage comme indicateurs d'impact de changement.

2.4. Approches et modèles

La synthèse des travaux autour de la problématique d'analyse et prédiction d'impact de changement [Abdi, 2007] a permis de dégager les principales approches adoptées ainsi que les différents modèles (de base) utilisés dans ces travaux.
La Table II-1 résume cette synthèse. De plus, l'auteur distingue les cas où on peut avoir
de l'analyse d'impact puis sa mesure (son calcul) des cas où en plus de l'analyse d'impact, on peut avoir aussi sa prédiction avant sa mesure.

<table>
<tr><td>Approches purement empiriques</td><td colspan="3">Pas de Modèle</td><td>Analyse + Mesure</td></tr>
<tr><td rowspan="2">Approches basées sur modèles</td><td colspan="3">Modèle</td><td rowspan="2">Analyse
+ Prédiction
+ Mesure</td></tr>
<tr><td>G.D</td><td>G.D
+ Formalisme</td><td>Autre que G.D</td></tr>
</table>

G.D : Graphe de Dépendance

Table II-1 Synthèse des différentes approches *[Abdi, 2007]*

2.5. Notion d'impact

Une classe A a un impact sur une classe B si un changement dans A peut entraîner que B ne compile plus ou que le comportement de B soit d'une façon ou d'une autre affecté.

A cette notion d'impact, on associe la notion de dépendance entre deux classes. Lorsqu' une classe A peut avoir un impact sur une classe B, on dit que B dépend de A. Il existe une relation directe (ou dépendance directe) entre deux classes A et B, si ces classes sont liées par au moins une des trois relations suivantes : héritage, agrégation, utilisation.

Les trois relations ci dessus (notées *R*) ont d'un point de vue impact les propriétés suivantes : Réflexivité, Non symétrique et Transitivité. Insistons sur le fait que ceci est valable uniquement du point de vue impact. L'agrégation et l'utilisation, au contraire de l'héritage, ne sont pas des relations transitives.

En effet, si la classe A est une agrégation de B et B une agrégation de C; alors A n'est pas forcément une agrégation de C. Car il se peut qu'aucun attribut de A ne soit une instance de C. Mais par contre, C peut avoir un impact sur A via cette relation d'agrégation. C'est donc l'impact que propage ces relations qui est transitif, mais pas les relations elles mêmes.

2.6. Relations indirectes entre classes

Une classe A a une relation indirecte avec une classe B, si il existe une chaîne B1, B2, …,Bn tel que : A *R* B1, B1 *R* B2, B2 *R* B3, …, Bn *R* B et n > 0

Ceci est noté ainsi : A *R*+ B.

La relation *R*+ est appelée la fermeture de *R*, elle a pour propriété d'être toujours transitive, d'où son appellation habituelle de fermeture transitive. Cette relation comprend toutes les classes qui sont reliées directement ou indirectement par la relation *R*. Si la relation *R* est définie comme suit :

R = {(A, B) tel que : A peut avoir un impact sur B}

et *R*+ = {(A, B) tel que : ∃ B1,…, Bn et A *R* B1, B1 *R* B2, …, Bn *R* B}

Alors pour une classe C, l'ensemble suivant : {X tel que : C *R*+ X} contient toutes les classes reliées à C directement ou indirectement et sur lesquelles C peut donc avoir un impact. Nous désignerons plus loin cet ensemble sous le nom de ***firewall* de C.**

Un problème classique lorsqu'on travaille avec un graphe orienté est de trouver tous les nœuds que l'on peut atteindre à partir d'un noeud donné. Autrement dit, pour un nœud A trouver tous les nœuds X du graphe pour lesquels il existe un chemin de A à X.

Ce problème revient donc à calculer la fermeture transitive de la relation qui correspond au graphe. Le calcul de la fermeture transitive est traditionnellement résolu par le fameux *algorithme de Warshall* **[**Aho**,** 1983**].** Il fournit une matrice (matrice de connectivité) indiquant pour chaque noeud tous les autres qui lui sont reliés. Dans le contexte de l'analyse d'impact pour un code objet, cet algorithme va nous permettre de calculer toutes les classes qui peuvent être affectées par une modification dans une classe donnée.

2.7. Firewall de classe

Nous définissons le firewall d'une classe C comme l'ensemble des classes pouvant être affectées par une modification dans C. Autrement dit, il s'agit de toutes les classes sur lesquelles C peut avoir un impact, et nous le notons *CFW(C)*. Il est important de garder à l'esprit que le firewall comprend les classes possiblement, mais pas nécessairement affectées par un changement. Pour une modification donnée, une classe du firewall peut ne pas être affectée. Mais par contre toute classe affectée se trouve nécessairement dans cet ensemble.

Pour avoir un certain degré de confiance dans un programme modifié, nous devons re-tester toutes les classes du firewall, car il constitue la portée maximale des conséquences d'une modification. Aucun effet de bord ne peut échapper au firewall. Une définition équivalente du firewall, consiste à dire qu'il s'agit de toutes les classes qui doivent être re-testées après qu'une classe ait été modifiée. Pour la modification de plusieurs classes, le firewall est la réunion des firewalls de chaque classe prise individuellement.

3. Modèle d'impact de changement

3.1. Objectifs

La conception d'un système peut être décrite comme un ensemble d'artefacts logiciels (classes) qui sont en interaction. Les liens d'inter-classes sont assumées avoir une plus grande influence sur la maintenabilité que les liens d'intra-classe **[**Rombach , 1990**].**

Ainsi, nous nous concentrons sur comment les divers liens entre des classes influencent en fait l'impact de changement. Quand un changement à un système est considéré, il est nécessaire d'identifier les composants du système qui seront impactés suite à ce changement. Cela permet de s'assurer que le système continue toujours à s'exécuter correctement après que le changement soit mis en œuvre. Notre intérêt est concentré sur comment le système réagit à ce changement et à d'autres changements en général. Notons qu'un système absorbe facilement un changement si le nombre de composants impactés est petit.

3.2. Modèle conceptuel

Un système est considéré comme un ensemble de classes connectées entre eux par différents liens. Une classe est définie comme un groupe de méthodes qui servent comme interface publique ou pour des opérations internes, et une section de variables qui définissent l'état des instances de la classe.

3.2.1. Changements

Un changement à un système est un changement qui peut s'appliquer à un composant. Un composant se réfère à une classe, une méthode, ou bien une variable. Comme exemples de changement, on peut avoir la suppression d'une variable, le changement de la portée d'une méthode, de "public" à "protected", ou le déplacement du lien entre une classe et son parent. Les principaux changements aux systèmes OO sont présentés dans la Table II-2. Ils sont définis puis classifiés selon le composant qu'ils affectent et un total de 13 changements est identifié au niveau conceptuel.

Composant	***Description de Changement***
Variable	Changement de type de variable
	Changement de portée de variable
	Ajout de variable
	Suppression de variable
Méthode	Changement de type de retour de méthode
	Changement d'implémentation de méthode
	Changement de signature de méthode
	Changement de portée de méthode
	Ajout de méthode
	Suppression de méthode
Classe	Changement de structure d'héritage de classe
	Ajout de classe
	Suppression de classe

Table II-2 Principaux changements au niveau conceptuel

3.2.2. Liens

Une fois qu'un composant donné est soumis à un changement, nous sommes intéressés par savoir quelles autres parties (classes) dans le reste du système seront affectées par ce changement. Une partie spécifique peut être affectée, dans le cas où elle est liée au composant changé via quelques liens entre eux. Ces liens sont parmi les quatre types suivants:

- **S** (Association) : Une classe fait référence aux variables d'une autre classe,
- **G** (Agrégation) : La définition d'une classe implique des objets d'une autre classe,
- **H** (Héritage) : Une classe hérite les particularités définies dans une autre classe (parente),
- **I** (Invocation) : Les méthodes d'une classe invoquent des méthodes définies dans une autre classe.

Nous considérons aussi une notation spéciale généralement utilisée dans l'algèbre booléenne :

- L'absence d'un opérateur entre 2 liens signifie une ***intersection***.
- L'opérateur "+" signifie une ***union***.
- L'opérateur "~" avant un lien signifie la ***négation***, par exemple, **~I** signifie l'ensemble des classes qui ne sont pas liées à la classe indiquée par le lien d'invocation.

Les liens sont indépendants les uns des autres et nous pouvons trouver n'importe quel nombre et type de liens entre deux classes. Un changement d'une classe peut aussi avoir un impact dans la même classe. Le pseudo-lien **L** (local) est introduit pour exprimer ceci.

3.2.3. Impact

L'impact d'un changement est l'ensemble de classes qui demandent une correction suite à ce changement. Il dépend de deux facteurs : le type de changement. Par exemple, un changement de type de variable a un impact sur toutes les classes faisant référence à cette variable tandis que l'ajout d'une variable n'a aucun impact sur ces classes. Étant donné un type de changement, l'autre facteur est la nature des liens impliqués. Si, par exemple, la portée d'une méthode est changée de "public" à "protected", les classes qui invoquent la méthode seront impactées, à l'exception des classes dérivées et des classes du même paquetage (package). Nous notons que plus qu'un type de lien entre la classe changée et une classe impactée peut être impliqué dans le calcul de l'impact.

Ainsi, pour un changement donné ch_i dans la classe cl_j, l'ensemble des classes impactées est exprimé par une expression booléenne dans laquelle les variables représentent les liens. Par exemple, la formule d'impact pour un changement hypothétique peut être donnée par : Impact (cl_j, ch_i) = **S~H+G**

Cela signifie que les classes qui sont en association (S) avec la classe changée cl_j et non dérivées (~H) de cette classe, ou les classes qui sont en agrégation (G) avec cl_j : sont impactées. La Table II-3 présente un exemple de changement pour chaque type de composant avec son expression d'impact :

Composant	*Description de changement*	*Expression d'impact (cl_j, ch_i)*
Variable	Changement de type	S+L
Méthode	Changement de portée de « Public » à « Protected »	I~H*
Classe	Suppression	H+G+S+L

Table II-3 *Exemples de Changements*

Ce modèle d'impact permet de prédire quelles classes seraient impactées si un changement est réellement fait. Un changement donné est caractérisé par une transformation du code quelque part dans le système. Si le système est recompilé avec succès, alors il n'y a aucun impact. Sinon, nous sommes face à un impact, c'est-à-dire, des modifications du code qui doivent être faites ailleurs dans le système pour obtenir un code syntaxiquement correct qui se recompilera.

Les considérations sémantiques relatives à la transformation du code ne sont pas considérées dans ce cas car elles ne peuvent pas être inférées du code source seulement. Puisque l'intérêt est centré seulement sur l'impact syntaxique d'un changement, les mesures appropriées que nous devons appliquer sont basées sur l'impact qui dépend seulement de la nature statique du code source. Ainsi, l'impact qui peut survenir pendant le temps d'exécution dû au polymorphisme par exemple est approvisionné.

Enfin, notons que pour certains changements, on sait qu'il y a un impact (impact certain), tandis que dans d'autres cas, il est clair qu'il n'y a aucun impact (impact nul).
Encore, dans quelques cas, on ne sait pas s'il y a un impact ou pas (impact incertain) avant que le morceau du code correspondant ne soit attentivement examiné. Par exemple, le cas de changement dans le type de retour d'une méthode abstraite. Les classes dérivées peuvent ou ne peuvent pas définir la méthode. Si la méthode n'est pas définie dans une classe dérivée, il n'y a aucun impact. Mais, si la méthode est définie dans une classe dérivée, alors il y a un impact dans la définition de cette méthode. En regardant seulement à la définition de la classe dérivée, nous pouvons déterminer s'il y a impact ou non.

* : L'expression obtenue est diminuée du nombre de classes du même package.

Ce type d'impact (impact incertain) a été traité comme un impact certain dans un but de calcul d'impact.

3.2.4 Adaptation du modèle à Java

Le modèle défini au niveau conceptuel (Table II.1) a été déjà adapté en C++. **[**Chaumun, 1998**]** et **[**Schauer, 2001**]**, le travail de [Abdi, 2007] vise les systèmes logiciels codés en Java, une opération d'adaptation de ce modèle à ce langage a été nécessaire. A cet effet [Abdi, 2007] a examiné l'adaptation déjà faite dans **[**Chaumun, 1998**]** et [Kabaili, 2002]. Il a remarqué qu'il y a certains changements qui sont communs aux deux langages, dans le sens où ils peuvent être appliqués aux deux langages. A titre d'exemple, le changement de type de variable, changement de signature de méthode, changement de la structure d'héritage d'une classe, etc. Par contre, il y a d'autres changements qui sont propres au langage C++, principalement les concepts de "virtual" (méthode virtuelle ou classe virtuelle) et " friendship" (classe amie).

La Table II.4 présente quelques exemples de ces changements. Les résultats de l'étude d'adaptation du modèle en question à C++ sont présentés dans [CHA 98]

Concept	***Description de changement***
Virtual (virtuelle)	virtuelle → non virtuelle
	Pure virtuelle → virtuelle
Friendship (Amitié)	Ajout de classe amie
	Suppression de classe amie
….	**….**

Table II-4 Exemples de changements propres à C++

Le concept de "virtual" est introduit en C++ pour gérer les appels dynamiques. Le rôle joué par ce concept en C++ est assuré en Java par le biais de sa machine virtuelle. Le fait de précéder la signature d'une méthode par le mot clé "virtual" en C++ ou "abstract" en Java signifie que cette méthode doit être surchargée "overiding" dans ses classes dérivées. Une erreur lors de la compilation est donnée si une méthode virtuelle pure (en C++) ou abstraite (en Java) n'est pas redéfinie dans les classes dérivées. Nous tenons à signaler que le modèle d'impact a été raffiné en JAVA dans [Abdi, 2007]. La Table II.5 présente un extrait de ce modèle d'impact.

ID du changement	Description du changement	Expression d'impact
v.1.1	Changement de valeur de variable	-
v.1.2	Changement de type de variable	S+L
..		
v.1.4	Suppression d'une variable	S+L
..		
v.1.5	Changement de porté de variable	
v.1.5.1	Public -> Private	S
v.1.5.2	Public -> Protected	S ~H*
v.1.5.3	Protected -> Private	SH
..		
m.2.1	Changement de méthode (Static/Non-static)	
m.2.1.1	Static -> Non-static	I+L
m.2.1.2	Non-static -> Static	L
…		
m.2.6	Changement de porté de méthode	
m.2.6.1	Public -> Private	
m.2.6.1.1	Non-abstract method	I
…		
m.2.6.2	Public -> Protected	
m.2.6.2.1	Non-abstract method	I ~H*
…		
m.2.6.3	Protected -> Private	
m.2.6.3.1	Non-abstract method	HI
c.3.2	Suppression de classe	
c.3.2.1	Non-abstract class	S+G+H+I
…..		
c.3.3	Dérivation d'héritage de classe	
c.3.3.1	Public -> Private	S+I
c.3.3.2	Public -> Protected	(S+I) ~H*
….		

Table II-5 Extrait des Résultats de l'impact des changements (Java)

Le modèle d'impact raffiné en JAVA comporte en tout 52 changements : 12 changements de variable, 25 changements de méthode et 15 changements de classe [Abdi, 2007].

4. Conclusion

Dans ce chapitre, nous avons présenté l'analyse d'impact de changement ainsi que ses intérêts, le modèle d'impact de changement, puis nous avons exposé quelques travaux de recherche réalisés dans le domaine d'analyse d'impact de changement.

Le chapitre suivant comporte une introduction sur les systèmes multi agents qui sont utilisés dans notre étude.

Chapitre III

Les Systèmes Multi Agents (SMA)

1. Introduction

L'intelligence Artificielle (IA) est un reconnue comme étant une discipline informatique qui a pour objectif de modéliser ou de simuler des comportements humains dits intelligents tel que la perception, la prise de décision, la compréhension, l'apprentissage, etc. elle s'attache à la construction de programmes informatiques, capables d'exécuter des tâches complexes, en s'appuyant sur une centralisation et une concentration de l'intelligence au sein d'un système unique. Mais l'IA a vite rencontré un certain nombre de difficultés, dues pour la plupart à la nécessite d'intégrer, au sein d'une même base de connaissances, l'expertise, les compétences et les connaissances d'individus différents qui, dans la réalité, communiquent et collaborent à la réalisation d'un but commun.

L'intelligence Artificielle Distribuée (IAD) [Bond & Gasser, 1988], [Huhns, 1987], [Erceau & Ferber, 1991], [Ferber, 1995] est née, au début des années 80, de la volonté de remédier aux insuffisances et d'enrichir l'approche classique de l'IA en proposant la distribution de l'expertise sur un groupe d'agents, non soumis à un contrôle centralisé, devant être capables de travailler et d'agir dans un environnement commun et de résoudre les conflits éventuels.

En résumé, l'IAD s'intéresse entre autre à la modélisation de comportements intelligents qui sont le produit de l'activité coopérative entre plusieurs agents, d'où la réalisation des systèmes dits « multi-agents ».

2. Définition et propriétés d'un agent

Dans la littérature scientifique, plusieurs définitions existent pour définir un agent. Nous optons pour celle de Mandiau [Mandiau, 2002], qui le définit comme une "entité qui agit dans un environnement". L'agent est capable de percevoir "au moins partiellement" son environnement, et d'agir en fonction de cette perception. Ses actions sont alors le résultat d'un raisonnement rationnel et d'une planification préalable.

D'une manière plus détaillée, Ferber [Ferber, 1995] définit un agent comme une entité physique ou virtuelle:

– qui est capable d'agir dans un environnement,

– qui peut communiquer directement avec d'autres agents,

– qui est mue par un ensemble de tendances (sous la forme d'objectifs individuels ou d'une fonction de satisfaction, voire de survie, qu'il cherche à optimiser),

– qui possède des ressources propres,

– qui est capable de percevoir (mais de manière limitée) son environnement, qui ne dispose que d'une représentation partielle de cet environnement (et éventuellement aucune),
– qui possède des compétences et offre des services,
– qui peut éventuellement se reproduire,
– 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.

Par ailleurs un agent peut aussi être défini par rapport à ses propriétés qui sont essentiellement:
– l'autonomie est définie par Demazeau et Muller [Demazeau et Muller, 1990] comme une existence indépendante : un agent autonome est un agent dont l'existence ne dépend pas de celle des autres. Une autre définition de l'autonomie [Russel, 1995] stipule qu'un agent autonome est un agent qui agit sans l'intervention des humains ou des autres agents, et qui a de ce fait un contrôle sur ses actions et sur ses états internes. Un agent logiciel autonome est une entité logicielle qui, à la différence de l'objet, peut choisir de répondre ou de ne pas répondre à un éventuel appel à ses méthodes ou à ses compétences.
L'agent est ainsi capable de prendre des initiatives sans avoir recours à des interventions externes.
– la réactivité d'un agent est sa capacité de percevoir et de réagir aux modifications survenues dans son environnement. Le choix des actions et de l'ordre dans la quel elles vont être entreprises, se fait en fonction de cette perception.
– la proactivité d'un agent est sa capacité à se fixer ses propres buts et objectifs par ses propres initiatives.
– la continuité d'un agent est sa capacité à être en permanence actif sans avoir a être provoqué par un stimulus externe. Florez définit cette continuité comme la persistance d'une identité et d'un état sur une longue période et ce à la différence des objets logiciels [Florez, 1999]. En effet, ces agents logiciels sont souvent implémentés sur des processus légers dit "threads", qui s'exécutent en parallèles.
– la sociabilité : est la capacité d'un agent de communiquer et d'interagir avec d'autres agents, soit pour échanger des informations nécessaires à son fonctionnement interne, soit pour faire aboutir des mécanismes de coopération et /ou de coordination mutuelles des différentes actions mutuelles.

3. La Typologie des agents

Les agents peuvent être en deux catégories : Des agents cognitifs dits « intelligents » et des agents réactifs dits « moins intelligents ». A ces deux types d'agents, s'ajoute l'architecture hybride et l'architecture BDI « Beliefs Desires and Intentions) conçue pour résoudre les inconvénients des agents réactifs et cognitifs.

3.1 Les Agents réactifs

Ce type d'agents se caractérise par le fait qu'ils n'ont pas de représentation de leur environnement, ni du monde auquel ils appartiennent. Ces agents sont les plus simples à mettre en œuvre du fait qu'ils se comportent selon le stimulus. L'agent sera programmé sous forme de couples « Stimulus/Réponse » [Picard, 2004]. Généralement les agents réactifs sont considérés comme non ou peu intelligents, et nous considérons que l'intelligence émerge de la coopération des différents agents. Ces derniers sont de plus bas niveau et n'ont qu'un protocole et un langage de communication réduit [Briot, 2001].

Figure III.1 Un agent réactif [Boissier, 2001]

Malgré la simplicité apparente et les bons résultats obtenus pour certaines applications [Brooks, 1991], des reproches ont été adressées à cette approche dite "réactive". Parmi lesquelles, il convient de voir que [Wooldridge, 2001] :

- Si les agents ne possèdent pas de modèle de leur environnement, ils doivent posséder suffisamment d'informations locales leur permettant de choisir une action acceptable;

- Il est difficile de voir comment un agent purement réactif peut apprendre de son expérience et améliorer ainsi ses performances;

3.2 Les Agents cognitifs

C'est un type d'agent plus complexe [Erceau, 1993] doté de capacités de raisonnement importantes et disposant d'une représentation de son environnement, des autres agents et d'eux-mêmes. Chaque agent est muni d'une base de connaissances comprenant l'ensemble des informations et des savoir-faire nécessaires à la réalisation de sa tâche ainsi qu'à la gestion des interactions avec les autres agents et avec son environnent. Pour atteindre leurs buts, ces agents sont capables d'échafauder des plans et de coopérer, ils utilisent l'expérience qu'ils ont acquise pour prendre des décisions, ils suivent le plan qu'ils ont établi jusqu'à son terme avant d'entreprendre une nouvelle tâche. Ils font souvent appel à des modes de communication plus complexes qu'une simple perception [Chevrier, 1993].

L'architecture de l'agent cognitif est illustrée par la figure III.2.

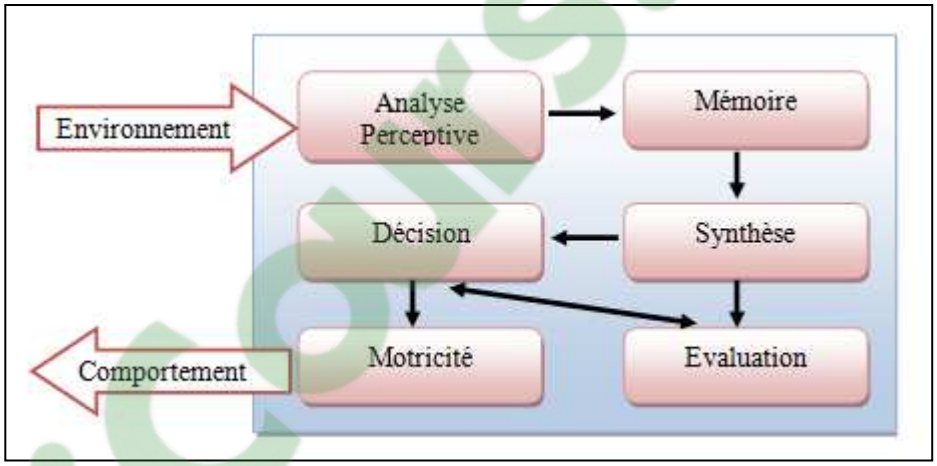


Figure III.2 Architecture d'un agent cognitif

❖ **Différence entre agents cognitifs/réactifs**

Les systèmes multi agents font la distinction entre « agents cognitifs » et « agents réactifs » :
Les agents cognitifs disposent d'une base de connaissances comprenant les diverses informations liées à leurs domaines d'expertise et à la gestion d'interaction avec les autres agents et leur environnement. Les agents sont généralement « «intentionnels » c'est-à-dire qu'ils possèdent des buts et des plans explicites leur permettant d'accomplir leurs buts. Dans ce cadre, comme le précise J.Ferber [Ferber, 1995], les problèmes de coopération ressemblent étonnamment à ceux de petits groupes d'individus, qui doivent coordonner leur activité, et sont parfois amenés à négocier pour résoudre leurs conflits.

Les agents réactifs au contraire ne sont pas « intelligents » pris individuellement. Ils ne peuvent que réagir à des stimulis simples provenant de leur environnement, et leur comportement est alors simplement dicté par la relation avec leur entourage sans que ces agents ne disposent d'une représentation des autres agents ou de leur environnement. Cependant, du fait de leur nombre, ces agents réactifs peuvent résoudre des problèmes qualifiés de complexes (table III. 1 qui résume ces différences).

Systèmes d'agents cognitifs	***Systèmes d'agents réactifs***
Représentation explicite de l'env.	Pas de représentation explicite.
Peut tenir compte de son passé.	Pas de mémoire de son histoire.
Agents complexes.	Fonctionnement Stimulus /action.
Petit nombre d'agents.	Grand nombre d'agents

Table III.1 Comparaison entre agents cognitifs/réactifs.

3.3 Architecture hybride ou mixte

Les agents *hybrides* sont des agents ayant des capacités cognitives et réactives. Ils conjuguent en effet la rapidité de réponse des agents réactifs ainsi que les capacités de raisonnement des agents cognitifs. Cette famille regroupe donc des agents dont le modèle est un compromis autonomie/coopération et efficacité/complexité. Pour illustrer cette famille, nous pouvons citer l'architecture ASIC [Demazeau, 1995] utilisée pour le traitement numérique d'images, l'architecture ARCO [Rodriguez, 1994] créé dans le cadre de la robotique collective et l'architecture ASTRO [Vercouter, 2000] développée pour être utilisée dans les systèmes multi-agents soumis à des contraintes de type temporelles.

La structure d'un agent hybride peut être divisée en trois couches. Une couche réactive qui va s'intéresser au traitement des données provenant de capteurs sensoriels, une couche intermédiaire raisonnant sur les connaissances de l'agent à propos de son environnement et enfin, une couche supérieure prenant en considération les autres agents du système dans la phase de raisonnement (leurs buts, leurs croyances, etc.) [Chaib-draa, 1996]. L'architecture de l'agent hybride est illustrée par la figure III.3.

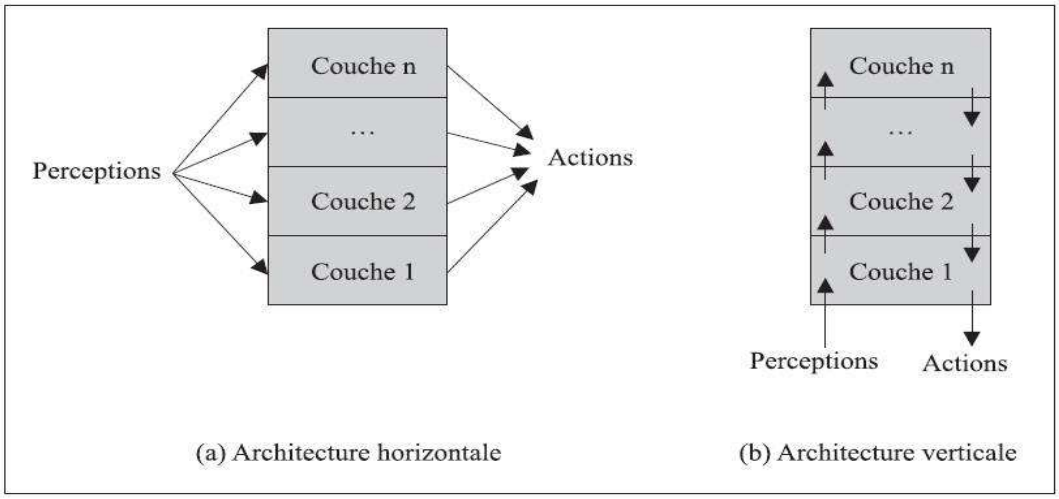


Figure III.3 Architecture d'agent en couche [Jennings, 1998]

3.4 Architecture BDI (Beliefs **D**esires and **I**ntentions)

Les agents BDI [Urbani, 2006] sont inspirés des travaux relatifs aux raisonnements pragmatiques, c'est-à-dire le processus de décision permettant de sélectionner les actions à effectuer pour atteindre ses objectifs. Ces agents sont représentés par un « état mental » ayant les attitudes mentales suivantes [Cohen et al, 1990]:

- **Les croyances** : ce que l'agent connait de son environnement;

- **Les désirs** : les états possibles vers lesquels l'agent pourra s'engager;

- **Les intentions** : les états vers lesquels l'agent s'est engagé, et envers lesquels il a engagé des ressources.

L'architecture BDI est illustrée par la figure (III.4)

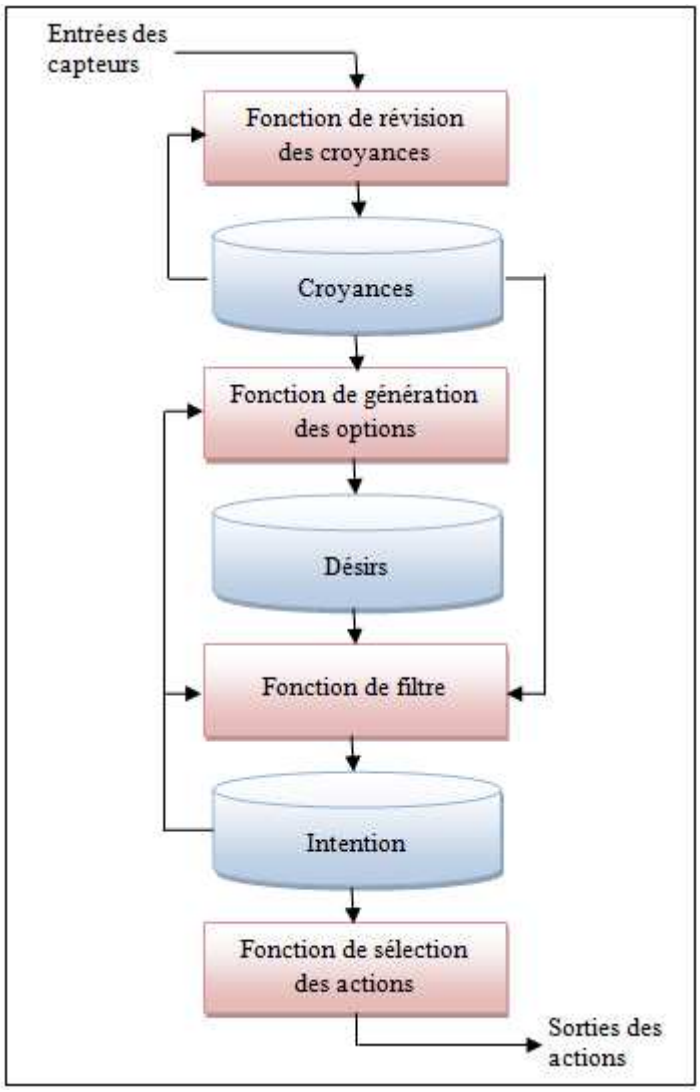


Figure III.4 Principe d'une architecture BDI

Un agent BDI [Jarras et al, 2002] doit donc mettre à jour ses croyances avec les informations qui lui parviennent de son environnement, décider quelles options lui sont offertes, filtrer ses options afin de déterminer de nouvelles intentions et poser ses actions au vu de ses intentions.

4. Les avantages des agents :

En tenant compte du fait que s'agissant d'une nouvelle technologie, on devait solutionner des problèmes qui, avant n'étaient pas automatisables (parce qu'il n'existait pas une technologie capable de donner une solution au problème ou bien parce que la solution existante était trop chère à appliquer), ou améliorer significativement (en ***coût***, ***rapidité*** ou ***facilité***) le développement de systèmes qui pourraient s'implémenter avec les technologies existantes.

Les SMA sont adéquats pour représenter des problèmes qui ont plusieurs méthodes de solution, de multiples perspectives où interviennent de multiples entités.

En plus des avantages comme la ***distribution*** et la ***concurrence*** dans la solution, ils permettent d'utiliser des patrons d'interaction sophistiqués de :

- ***Coopération*** (travail en commun pour atteindre un but commun).
- ***Coordination*** (organisation du processus de solution du problème de manière à éviter des interactions nocives et à exploiter celles bénéfiques).
- ***Négociation*** (formulation d'un accord qui soit acceptable par toutes les parties impliquées).

C'est précisément la ***flexibilité*** et la nature de haut niveau de ces interactions qui distinguent et donnent de l'importance à un SMA par rapport à d'autres paradigmes du génie logiciel.

5. Les Systèmes Multi Agents (SMA)

Un SMA est généralement défini comme étant un ensemble d'agents, en interaction les uns avec les autres, pouvant coopérer, négocier ou collaborer. Ils évoluent dans un environnement qu'ils perçoivent, dans lequel ils peuvent se déplacer et qu'ils peuvent modifier. Gerhard Weiss [Weiss, 2000] définit l'intelligence artificielle distribuée comme étant l'étude, la conception et la réalisation de systèmes multi-agents qu'il présente comme étant des systèmes dans lesquels des agents intelligents interagissent et poursuivent un ensemble de buts ou réalisent un ensemble d'actions.

Ferber [Ferber, 1995] le définit comme un système composé des éléments suivants :

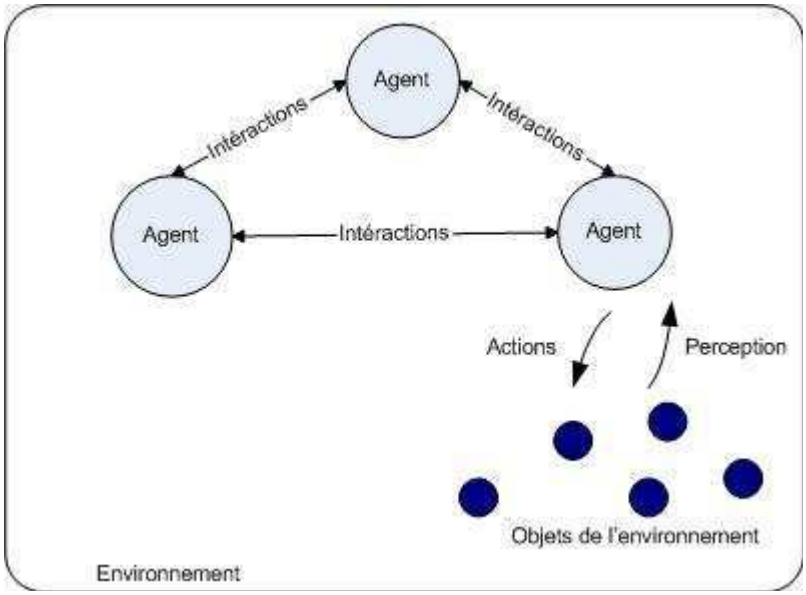
- Un environnement ***E***, dans notre cas, c'est l'espace où peuvent se déplacer les agents ;

- Un ensemble d'objets ***O***. Ces objets sont situés, c'est-à-dire que pour tout objet, il est possible, à un moment donné, d'associer une position dans *E* ;

- Un ensemble ***A*** d'agents qui sont des objets particuliers représentant les entités actives du système ;

- Un ensemble de relations ***R*** qui unissent des objets (et donc des agents) entre eux ;

- Un ensemble d'opérations ***OP*** permettant aux agents de *A*, de percevoir, produire, consommer, transformer, et manipuler des objets de ***O***. Cela correspond à la capacité des agents de percevoir leur environnement, de manager, etc.



Figur III.5 Représentation d'un système multi-agents selon Ferber [Ferber, 1995]

5.1. Les Caractéristiques des environnements SMA

Les principales caractéristiques d'un système multi-agents :

- Chaque agent possède des informations et des capacités incomplètes pour la résolution des problèmes.
- Il n'existe pas un système de contrôle global. Par contre, dans le cas du contrôle centralisé, il existe une entité qui a une vue globale de l'activité du système et qui a la charge de contrôler tout le système.
- Les données sont décentralisées.
- Le calcul est asynchrone.

5.2 Les Différentes formes d'interaction

La force du paradigme agent provient de la flexibilité et de la variété des interactions entre les agents. Ces derniers communiquent entre eux (coopèrent, négocient, se coordonnent) pour pouvoir travailler ensemble et agir dans un environnement commun et atteindre un objectif commun.

Jacques Ferber **[**Ferber, 1997**]** propose la définition suivante de l'interaction :

« *Une interaction est la mise en relation dynamique de deux ou plusieurs agents par le biais d'un ensemble d'actions réciproques. Les interactions sont non seulement la conséquence*

d'actions effectuées par plusieurs agents en même temps, mais aussi l'élément nécessaire à la constitution d'organisations sociales »**.**

[Weiss, 2000] propose le schéma suivant :

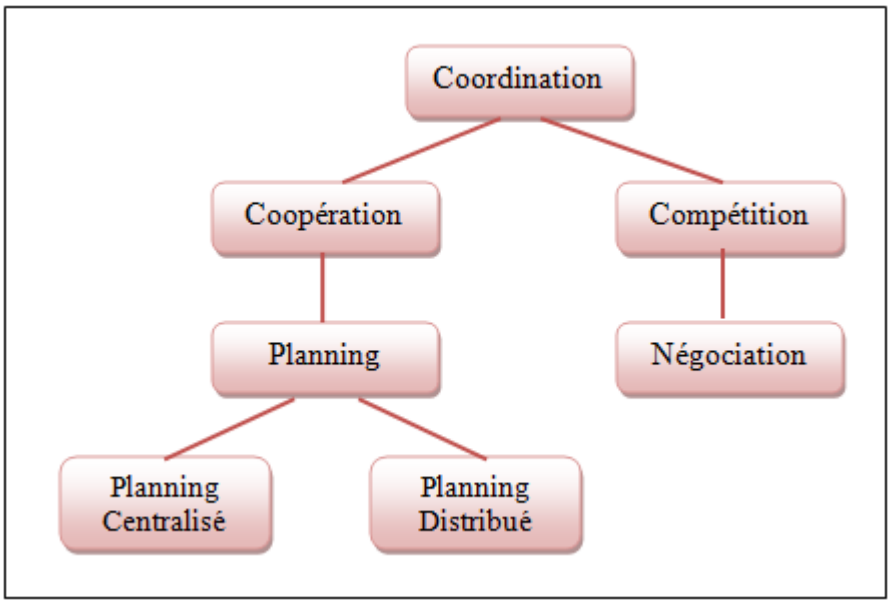


Figure III.6 Formes d'interactions entre agents

5.2.1. Coordination

[Weiss, 2000] définit la coordination comme "la propriété d'un système composé d'au moins deux agents, exécutant des actions dans un environnement partagé". Cette notion de ressources partagées dans l'environnement implique la nécessité de la coordination. Les agents devraient coordonner leurs actions individuelles avec les autres pour aboutir à l'objectif global du groupe. Cette coordination permet alors :

– d'éviter les situations de conflits par la négociation pour les agents antagonistes (ayant des buts et des objectifs contradictoires),

– d'améliorer l'efficacité et l'utilité de chaque agent par la coopération pour les agents non antagonistes.

Il existe deux types de coordination [Boissier et al, 1994] :

a. Coordination centralisée : Dans ce type de coordination, il existe deux types d'entités, un coordinateur responsable de la bonne exécution d'un plan donné et un ensemble d'agents esclaves qui exécute les ordres du coordinateur. Cette approche suppose une hiérarchie de l'organisation. Cependant, la centralisation des messages vers le coordinateur peut provoquer le ralentissement des performances du système.

b. Coordination distribuée : Plusieurs agents ont décidé de coopérer, ils vont donc être responsables chacun d'une partie du plan, chacun d'eux va choisir un protocole d'interaction adapté à la coopération qu'il souhaite mettre en œuvre. Ces protocoles représentent des enchaînements d'interaction possibles sous forme d'un automate à état finis ou d'un réseau de Petri. Cette approche vise à augmenter l'autonomie des agents, ce qui va améliorer la flexibilité du système **[**Engelmore et al, 1988].

5.2.2. Coopération

D'après [Weiss, 2000], la coopération est la coordination parmi des agents non antagonistes, qui cherchent à se satisfaire mutuellement sans se gêner. Cette coopération, initiée par un échange d'information, est souvent associée à la notion de collaboration. La collaboration est une forme d'interaction qui étudie la manière de répartir le travail, et par conséquent l'allocation de tâches, entre plusieurs agents.

La coopération est nécessaire quand un agent ne peut pas atteindre ses buts sans l'aide des autres agents. La coopération entre les différents agents est un problème difficile car les agents ont une connaissance plus au moins précise des autres agents du système, mais ils doivent aussi être informés sur les compétences des autres agents, et les tâches que ces derniers sont entrain de réaliser. Il existe trois types de coopération [Hoc, 1996]:

a. Coopération confortative : Une tâche est exécutée par plusieurs agents exploitant simultanément un même ensemble de données et poursuivant un but en commun, selon des compétences différentes .Une étape de fusion conditionne ensuite la production du résultat.

b. Coopération augmentative : Elle correspond à une décomposition de la tâche à traiter en sous tâches qui s'exécutent d'une manière concurrente. Ce type de coopération est émergeant : les agents remplissent des objectifs locaux, leurs activités modifient localement l'environnement qui va influencer les agents voisins. La solution est l'union des sous résultats locaux.

c. Coopération intégrative : Cette coopération est intentionnelle : elle nécessite la définition d'un plan d'actions concernant la réalisation de la tâche. Les agents concernés vont alors travailler selon un plan séquentiel où chaque agent intègre les résultats des agents précédents.

5.2.3 Négociation

C'est une méthode de coordination qui permet à plusieurs agents, d'atteindre suite à un processus de communication et d'échange d'informations un accord mutuel pour entreprendre

une action donnée d'une certaine manière. Elle induit, par cette communication, des relaxations de buts initiaux, des concessions mutuelles, des mensonges ou des menaces [Fayech, 2003]. Elle est donc considérée comme une méthode de résolution de conflits et de recherche de consensus.

5.3 La Communication entre agents

La communication entre les agents est un point central de la théorie des SMA, les agents doivent disposer d'un protocole de communication pour se coordonner entre eux, négocier, et coopérer. Nous distinguons essentiellement deux modes de communication :

- La communication indirecte se fait en apportant des modifications sur l'environnement. Ainsi, un agent émetteur effectue des modifications sur l'environnement ; l'agent récepteur perçoit la modification et interprète le message. L'agent émetteur peut propager des signaux, ou laisser des traces dans l'environnement.

- La communication directe se fait via l'envoi de messages entre agents, elle est souvent associée à une action.

Les moyens de communication entre les agents considérés les plus importants sont :

5.3.1 Les Tableaux noirs

En intelligence artificielle, la technique du tableau noir (blackboard) est très utilisée pour spécifier une mémoire partagée par divers systèmes [Nii, 1989], [Nii & al, 1989].

Dans un SMA utilisant un tableau noir, les agents peuvent écrire des messages, insérer des résultats partiels de leurs calculs et obtenir de l'information. Le tableau noir est en général partitionné en plusieurs niveaux qui sont spécifiques à l'application. Les agents qui travaillent sur un niveau particulier peuvent accéder aux informations contenire dans le niveau correspondant du tableau noir ainsi que dans des niveaux adjacents. Ainsi, les données peuvent etre synthétisées à n'importe quel niveau et transférées aux niveaux supérieurs alors que les buts de haut niveau peuvent etre filtrés et passés aux niveaux inférieurs pour diriger les agents qui oeuvrent à ces niveaux. Le transfert de messages et les communications basées sur un tableau noir sont souvent combinés dans des systèmes complexes. Dans de tels systèmes, chaque agent est composé de plusieurs sous-systèmes (ou sous-agents) qui communiquent à travers un tableau noir. Les agents communiquent entre eux par échange de messages.

5.3.2 Transfert de plans ou de messages

Dans l'approche de transfert de plans, un agent X communique son plan en totalité à un agent Y qui a son tour lui communique le sien. Cette approche présente plusieurs inconvénients, car le transfert de plans est coûteux en ressources de communication et constitue une approche difficile à mettre en œuvre dans des applications réelles. Dans des situations réelles, il n'est pas possible de formuler à l'avance des plans en totalité. Par conséquent, on a besoin de communiquer des stratégies générales aux agents plutôt que des plans, et il est nécessaire, par la suite, que les agents puissent s'échanger des messages [Smith, 1980].

Les SMA fondés sur la communication par messages se caractérisent par le fait que chaque agent possède une représentation propre et locale de l'environnement qui l'entoure. Chaque agent va alors interroger les autres agents sur cet environnement ou leur envoyer des informations sur sa propre perception des choses ; les agents doivent pouvoir envoyer et recevoir des messages à travers un réseau de communication (qui devra être fiable). Les messages peuvent être de différents types, les deux types minimaux sont la requête et la réponse. N'importe quel agent (quelque soit son rôle) peut accepter ou refuser un message et donc recevoir une assertion. Pour tenir un rôle passif, un agent doit être capable d'accepter une question externe et envoyer la réponse à la source (assertion). Pour tenir un rôle actif, un agent doit pouvoir poser une question et faire des assertions. Il peut ainsi contrôler un autre agent par le biais des questions/réponses [Jarras et al, 2002]. La communication par envoie de message est illustrée par la figure III.7.

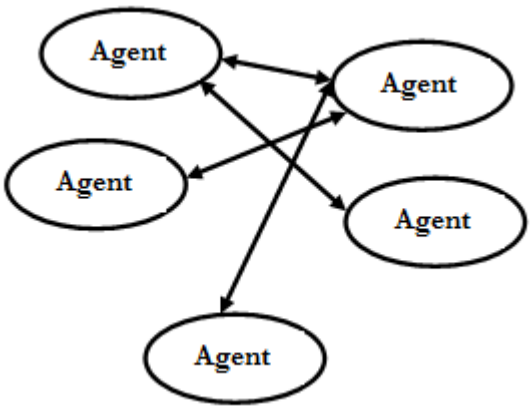


Figure III.7 Communication par envoie de message

5.3.3 Communication via KQML (**K**nowledge **Q**uery and **M**anipulatioin **L**angage)

KQML est un langage issu des travaux de la knowledge sharing effort [Finin, 1993]. Il s'agît d'un langage qui vise à définir un ensemble d'actes de langages qui soient standard et utile .Ces actes de langages appelés aussi performatifs, sont utilisés par les agents pour échanger des informations. La forme de base du protocole est [Fipa, 1999] :

KQML-performative

Sender <word>

Receiver <word>

Language <word>

Ontology <word>

Content <expression> ...)

KQML enveloppe un message dans une structure qui peut être comprise par n'importe quel agent. Les langages qui peuvent être utilisés dans un message KQML sont multiples (PROLOG, LISP, SQL etc.), Ces dernières années, KQML semble perdre du terrain au profit d'un autre langage plus riche sémantiquement ACL (pour Agent Communication Language).

ACL est basé, également, sur la théorie du langage et a bénéficié grandement des résultats de recherche de KQML. Si toutefois, les deux langages se rapprochent au niveau des actes du langage, il n'en est rien au niveau de la sémantique.

6. Domaines d'application des SMA :

Depuis les premières expériences à la fin des années 80 (par exemple, un système de gestion de fabrication [Parunak, 1987] basé sur le protocole contract net [Smith, 1980]), une grande diversité de systèmes ont été construits en utilisant des agents. Celles-ci vont des applications simples, comme les assistants de courrier électronique, aux systèmes complexes, comme le contrôle du trafic aérien [Ljungberg, 1992]. Un aspect intéressant pour l'application de la technologie d'agents est précisément le fait que le concept d'agent permet d'une façon naturelle de modéliser une gamme de systèmes aussi amples et avec des besoins aussi divers [Pavon, 2006].

Ils ont été utilisés avec succès dans la supervision, le commerce électronique, les services Web, etc.

Parallèlement, dans le domaine du logiciel, les composants logiciels font l'objet de propositions variées, qualifiées d'industrielles (EJB, COM, .Net, Fractal) ou plus académiques (ArchJava, ACME, ...). Les applications concernent plus particulièrement le domaine du génie logiciel : la programmation, la vérification ou les infrastructures logicielles. Des applications réussies ont été conduites dans le domaine des lignes de produit multimédia, du commerce électronique, des systèmes d'exploitation, des télécommunications, etc.

7. Conclusion

Dans ce chapitre, nous avons présenté l'univers multi-agents. Nous avons donné quelques définitions essentielles. Pour cela nous avons abordé le concept d'agent considéré comme unité de base d'un SMA, ses différents types, sa structure, son architecture, ses caractéristiques ainsi que ses apports. L'une des caractéristiques principales des SMA tient à l'interaction entre les différents agents du système tout au long de leur exécution. Cette interaction est primordiale, elle permet aux agents de communiquer, d'agir sur leur environnement, de coopérer, collaborer et de s'organiser pour la réalisation d'un but commun.

Dans le chapitre qui suit, nous allons aborder notre approche qui utilise les SMA pour la validation d'un modèle d'impact de changement.

Chapitre IV

Conception et réalisation

1. Introduction et contribution

Ce chapitre est dédié à notre contribution qui consiste à l'étude d'impact de changement pour les systèmes orientés objets à l'aide d'un SMA, en tenant compte de la taille importante des systèmes OO, la complexité de l'opération de l'extraction des constituants du modèle conceptuel du code source (les différents composants du système), la détection de l'impact de changement et sa propagation dans le système. Dans notre étude, nous proposons un modèle à base d'agents, ces derniers interagissent de manière cohérente sous le contrôle d'un agent analyseur (coordinateur).

Tout d'abord, nous présentons l'architecture de notre modèle proposé, ainsi que la structure, et le fonctionnement de chaque composant du système. Ensuite, nous présentons nos différents tests effectués permettant de démontrer les capacités des SMA dans la résolution de notre problématique d'impact de changement et pour finir, nous concluons par une discussion de nos résultats obtenus et nous ouvrons des pistes pour des travaux futurs visant à améliorer ce système et à l'appliquer à d'autres tâches.

2. Approche proposée

La technologie Multi Agents a déjà fait ses preuves dans de nombreux domaines par leur capacité de modélisation, ils permettent de représenter les interactions entre diverses entités pouvant coopérer et communiquer.

Nous proposons dans le cadre de ce travail une approche basée sur une architecture multi-agent qui est divisée en deux parties, la première consiste à l'extraction des différents composants du système en entrée, et la deuxième partie consiste à l'étude de l'impact de changement et la comparaison des résultats obtenus avec ceux du modèle d'impact raffiné étudié dans le chapitre II.

La figure IV.1 montre le schéma global de l'approche, nous avons le système en entrée qui constitue le programme source en entrée de notre système et le module SMA « système multi agent » qui aura pour mission de représenter les différents acteurs qui disposent de leurs propres objectifs, stratégies de décision et préférences. Il est composé de : Agent Analyseur AA (c'est l'agent coordinateur de notre système), Agent Agrégation AG, Agent Héritage AH, Agent Invocation AI, Agent Association AS et Agent Impact AP qui coopérèrent, coordonnent leurs buts et leurs plans d'actions pour résoudre le problème.

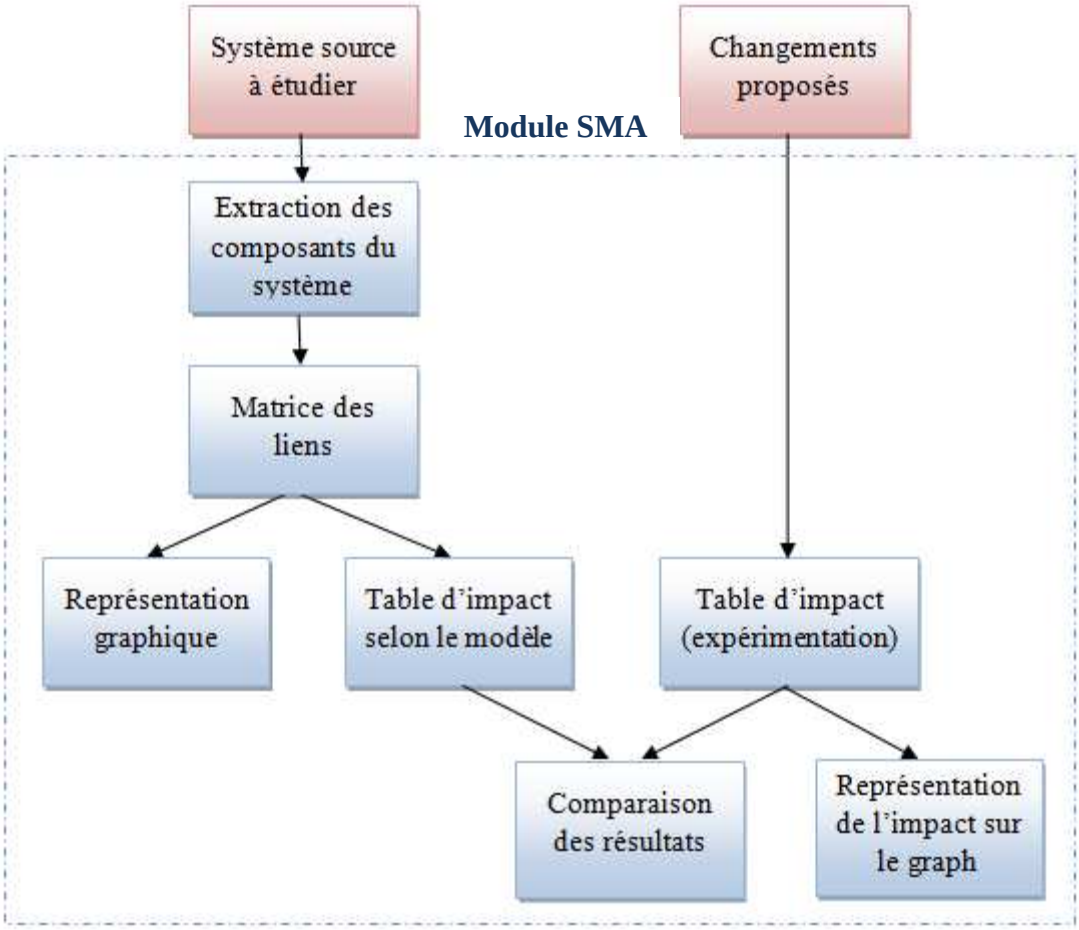


Figure IV.1 Schéma global de l'approche proposé

2.1. Extraction des composants:

L'objectif consiste à extraire les différents composants du système source :

L'agent analyseur AA analyse le code source JAVA du système en Test en entrée, il parcoure l'arborescence du projet et fait ressortir les informations utiles pour notre étude à savoir : les classes, les attributs et les méthodes, par la suite l'agent analyseur demande aux agents « agent héritage AH, agent invocation AI, agent association AS et agent agrégation AG » de commencer la phase de recherche des relations entre classes:

- L'agent agrégation AG examine les classes qui sont en agrégation dans le système.
- L'agent association AS examine les classes qui sont en association dans le système.
- L'agent héritage AH examine les classes qui sont en héritage dans le système.
- L'agent invocation AI examine les classes et définit l'ensemble de méthodes invoquées entre classes dans le système.

2.2. Matrice des liens:

La matrice des liens montre les différentes relations entre classes, elle est alimentée lors de la phase d'extraction par les agents AG, AH, AS, AI qui renseignent chaque lien localisé entre deux classes.

2.3. Présentation graphique:

Dans l'objectif de mettre en évidence les liens entre les différents composants du système cette représentation graphique vient faciliter la compréhension du système et illustrer la dépendance entre les différentes classes en visualisant les liens (Héritage, Association, Agrégation et Invocation) qui existent entre eux. L'agent analyseur à l'aide des résultats obtenus lors des phases précédentes établit la présentation graphique du système.

2.4. Table d'impact selon le modèle:

En se basant sur la matrice des liens calculée et l'expression booléenne générée par le modèle d'impact raffiné (étudié en chapitre 2, table II.5), nous obtenons la table d'impact selon le modèle.

2.5. Changement proposé :

Afin de réaliser l'étude d'impact de changement, des changements sont portés concrètement sur le code source chargé.

2.6. Calcul de la table d'impact après changement:

Après chaque changement effectué sur le code source, l'agent impact AP contact les agents AG, AH, AI, AS pour recalculer la matrice des liens après le changement, il compare par la suite les deux matrices des liens et obtient la table d'impact de changement.

2.7. Représentation du changement :

A partir de la table d'impact de changement obtenue lors de l'étape précédente, l'agent impact interprète le changement sur la représentation graphique, montre le changement et les classes impactées par ce dernier.

2.8. Confrontation des résultats:

Lors de cette phase on compare les résultats d'impact de changements générés par le modèle d'impact et ceux obtenus par expérimentation.

La figure IV.2 ci-après illustre la communication entre les agents et explique le fonctionnement du système.

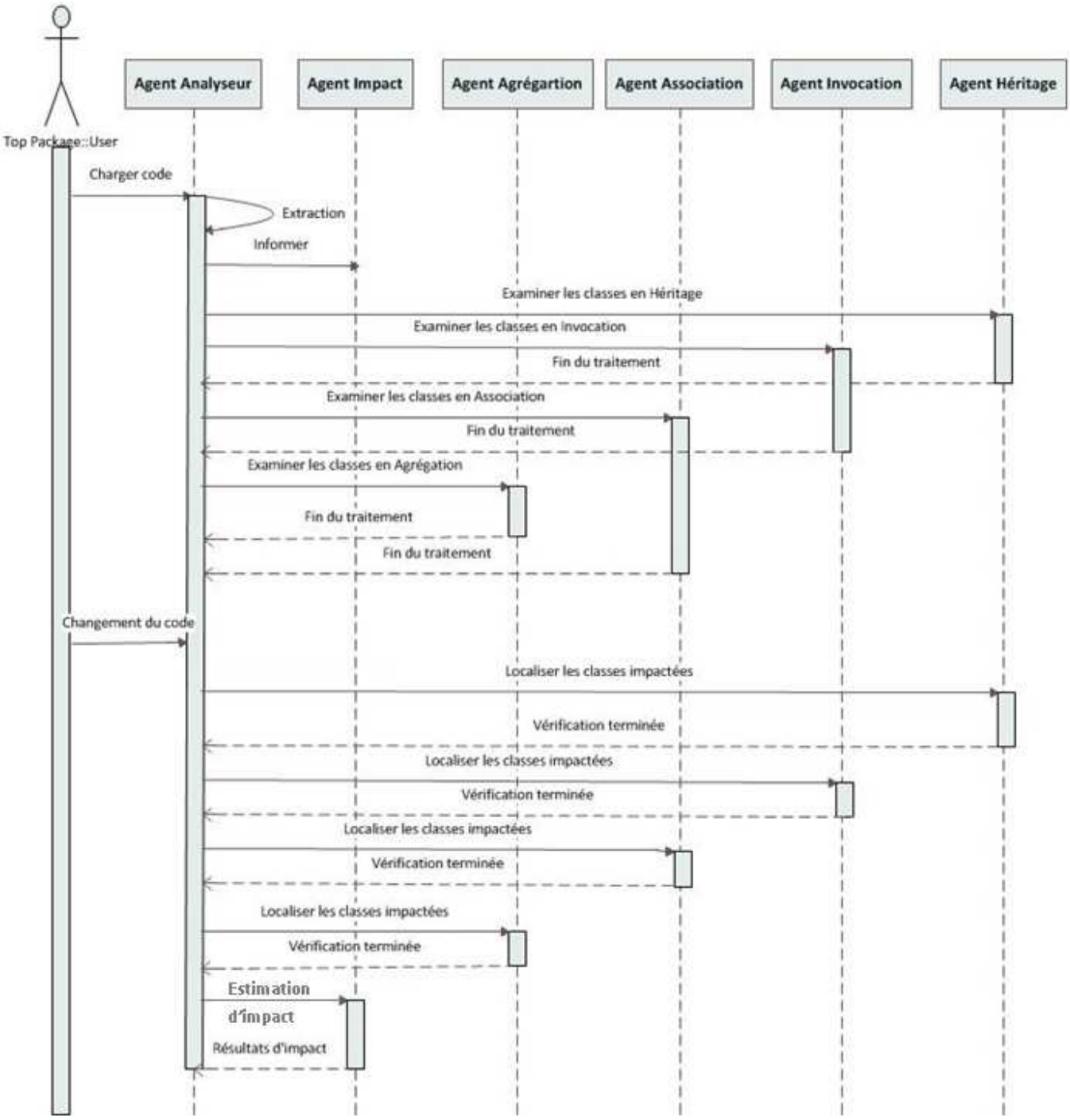


Figure IV.2 Communication entre les agents

3. Structure des agents

3.1. Agent Analyseur :

Cet agent est le noyau central de notre architecture, il est le coordinateur, son rôle est de :

- Distribuer les tâches aux différents agents
- Récolter leurs réponses.
- Garantir la communication entre les agents
- Assurer la bonne exécution
- Coordonner les actions des différents agents pour aboutir à l'objectif du système.

C'est le premier agent qui intervient dans le système. Sa structure est composé de :

- Module de traitement.
- Module d'interface.
- Module de coordination.
- Base de données BDD : Cette base contient les différentes informations concernant l'agent, sa tâche et son historique d'intervention.
- Base de connaissance BC : Cette base est un pool de stockage d'une expertise, elle contient aussi les différentes connaissances que l'agent peut exécuter.

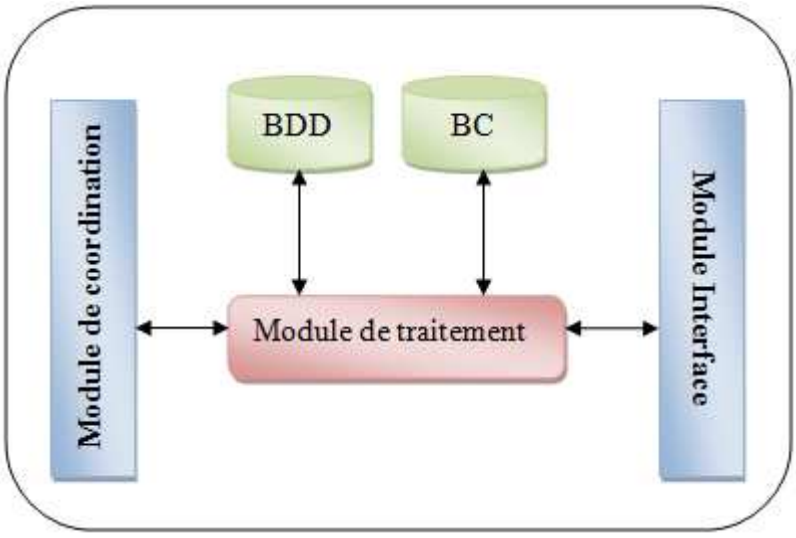


Figure IV.3 Agent Analyseur

3.2. Agent Héritage, Association, Agrégation, Invocation

Ces agents sont responsables de l'extraction des différentes relations entre les classes du système source. En cas d'un changement effectué sur le code, ils sont appelés pour intervenir afin de localiser l'impact du changement.

Les agents AG, AH, AI, AA sont composés de :

- Une base de données : Cette base contient les différentes informations concernant l'agent, sa tâche et son historique d'intervention.
- Un module de traitement.

3.3. Agent Impact

Cet agent est responsable de l'estimation de l'impact de changement en fonction des résultats des autres agents (AI, AG, AS, AH).

4. Communication entre les agents

La communication inter-agents est fondamentale à la réalisation du paradigme agent, pour échanger les informations et les connaissances, les agents utilisent des ACL (Agent Communication Language) pour communiquer.

4.1. Le langage ACL (Agent Communication Language)

Le langage ACL de FIPA (http://www.fipa.org - une organisation qui a pour but de standardiser le langage de communication entre des agents) est fondé également sur la théorie des actes de langage et a bénéficié grandement des résultats de recherche de KQML. Si l'approche du langage ACL est globalement semblable à celle de KQML, un grand soin a été apporté aussi bien à la description formelle de la sémantique des actes de communication qu'à l'introduction de protocoles régissant les règles d'échanges de messages.

4.2. Structure des messages envoyés

Performative	*Description*
sender	le nom de l'agent qui envoie le message
receiver	le nom de l'agent qui reçoit le message
reply-to	Participant à l'acte de communication
content	le contenu du message (l'information transportée par la performative)
language	le langage dans lequel le contenu est représenté
protocol	contrôle la conversation
conversation-id	identificateur de la conversation
reply-with	l'expression qui est utilisée par l'agent répondant pour identifier ce message.
in-reply-to	référence à un message auquel l'agent est entrain de répondre (précisé par l'attribut reply-with de l'émetteur)
reply-by	impose un délai pour la réponse

Table IV.1 Performatives de ACL-FIPA

Les messages echangés entre les agents sont :

- **Request () :** l'agent analyseur envoie un message à tous les agents pour qu'ils commencent leurs traitement.

- **Inform () :** l'agent analyseur envoie un message à l'agent impact pour l'informer que l'extraction des composants a été finalisée;

- **Confirm () :** les 4 agents (AG, AH, AI, AS) envoient un message à l'agent analyseur pour l'informer que l'extraction a été un succès et que les relations entre classes ont été trouvées ;
- **Refuse () :** L'agent participant indique à l'agent analyseur que l'extraction a terminé et aucun lien n'a été localisé.

5. Coopération et coordination entre agents

Dans notre modèle proposé les agents travaillent à la satisfaction d'un but commun, les actions des agents sont exécutées simultanément et cela améliore les performances. Chaque agent utilise ses connaissances et ressources pour résoudre localement un ou plusieurs sous problèmes. Les solutions partielles à tous les sous-problèmes sont par la suite intégrées. Afin de garantir la cohérence globale du SMA, il est nécessaire que les agents coordonnent leurs différentes actions. Dans notre système l'agent Analyseur joue le rôle du coordinateur :

a. L'agent Analyseur parcourt le code source chargé et recense les différent composant du système (classe, attribut, méthode), ensuite il demande aux agents Agrégation, Héritage, Invocation, Association de commencer leurs traitement.

b. Les agents Agrégation, Héritage, Invocation, Association reçoivent une demande de l'agent Analyseur, ils commencent le traitement d'extraction des liens. Une fois la recherche terminée, ils rendent la réponse à l'agent Analyseur et représentent le lien ou les liens trouvés sur la matrice des liens.

c. L'agent Analyseur une fois il rassemble les réponses des 4 agents, il traduit les résultats obtenus en représentation graphique.

e. Dans le cas d'un changement sur le code source, l'agent impact contacte les agents Agrégation, Héritage, Invocation, Association pour .

f. Les agents Agrégation, Héritage, Invocation, Association répètent la recherche des liens et remplissent la nouvelle matrice des liens, puis retournent l'information à l'agent Analyseur.

g. L'agent impact compare les deux matrices des liens, construit la table d'impact de changement et visualise l'impact sur le graphe. Ensuite il passe à la phase de confrontation des résultats obtenus par rapport au modèle d'impact de changement.

6. Réalisation

6.1.Outils de développement

Nous avons implémenté notre outil sous la plateforme Eclipse.

6.1.1. Eclipse

Nous avons implémenté notre outil avec la plateforme Eclipse pour deux raisons principales :

A- C'est un langage Orienté Objet.

B- C'est un environnement de développement intégré (IDE) ayant pour objectif de fournir une plate-forme modulaire pour le développement des applications informatiques. C'est un Open Source qui offre un environnement de développement Java gratuit. La principale caractéristique d'Eclipse est l'extensibilité de l'environnement. Il est construit à base de mécanismes capables de charger, d'intégrer et d'exécuter des modules appelés plug-ins. Un plug-in est une unité de fonction qui peut être développée et délivrée séparément. À titre d'exemple, l'outil de développement Java JDT (Java Development Tools) et l'environnement de développement de plug-in PDE (Plug-in Development Environment). Ce dernier intéresse principalement les développeurs qui veulent étendre Eclipse, puisqu'il permet de développer et d'intégrer de nouveaux outils à son environnement.

6.1.2. AST (Abstract Syntax Tree) Parser

Pour l'analyse syntaxique du code source Java en entrée, nous avons utilisé un plugin Eclipse « ASTParser » qui permet à partir d'un code source d'avoir son arbre syntaxique [Dinedane, 2011].

6.1.3. Plate forme JADE pour l'implémentation du SMA

Le développement d'un SMA est une tâche lourde, complexe qui requiert un investissement très important. A cet effet, pour implémenter le module SMA, nous optons pour l'utilisation

d'une plateforme Multi Agents existante que nous adaptons à nos besoins. Notre choix s'est porté sur la plate forme SMA **jade** (**J**ava **A**gent **DE**velopement framework) **[Fip, 00]** qui est une plateforme gratuite, implémentée en java, son code source et celui de son environnement de développement sont ouverts et modifiables permettant ainsi une utilisation relativement aux besoins exprimés. Elle possède trois modules (normes FIPA).

-DF « Director Facilitor » : Fournit un service de « pages jaunes» à la plate-forme;

-ACC «Agent Communication Channel » : Gère la communication entre les agents;

-AMS « Agent Management System » : Supervise l'enregistrement des agents, leur authentification, leur accès et l'utilisation du système. Ces trois modules sont activés à chaque démarrage de la plate-forme;

- **Outils de la plateforme JADE**

La plate forme jade est dotée d'un certain nombre d'outils tel que :

a. Agent RMA (Remote Management Agent)

Le RMA permet de contrôler le cycle de vie de la plate-forme et tous les agents la composant. Plusieurs RMA peuvent être lancés sur la même plate-forme du moment qu'ils ont des noms distincts. L'interface de l'agent RMA est illustrée par la figure IV.4

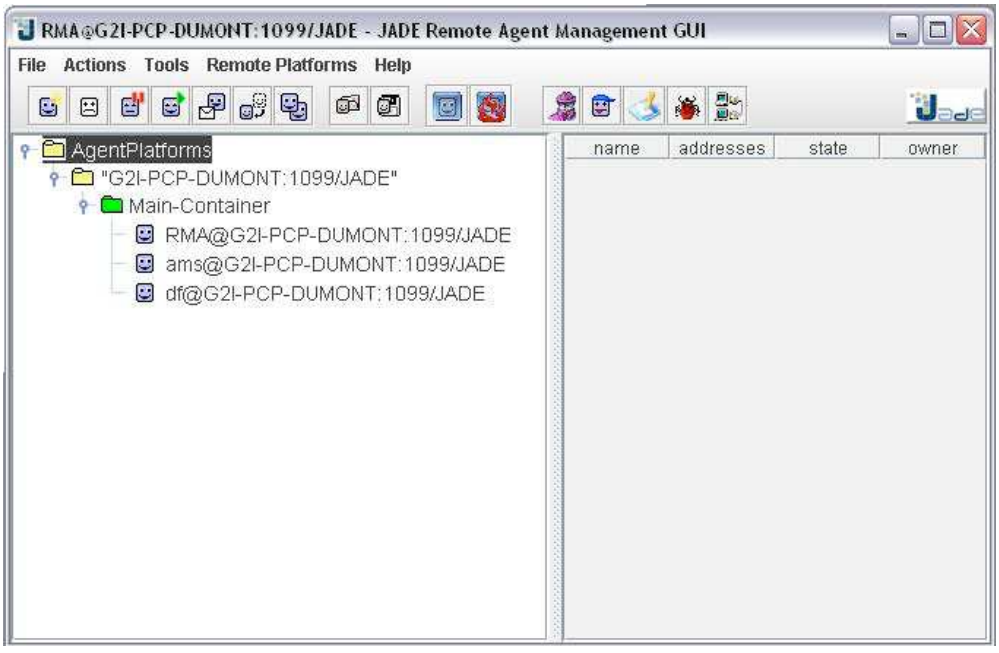


Figure IV.4 Agent Remote Management RMA

b. Agent (Direcory Facilitator)

L'interface du DF peut être lancée à partir du menu du RMA .Cette action est e implantée par l'envoi d'un message ACL au DF lui demandant de charger son interface graphique. L'interface peut être juste vue sur l'hôte où la plate-forme est exécutée. En utilisant cette interface, l'utilisateur peut interagir avec le DF. L'interface de l'agent DF est illustrée par la figure VI.5

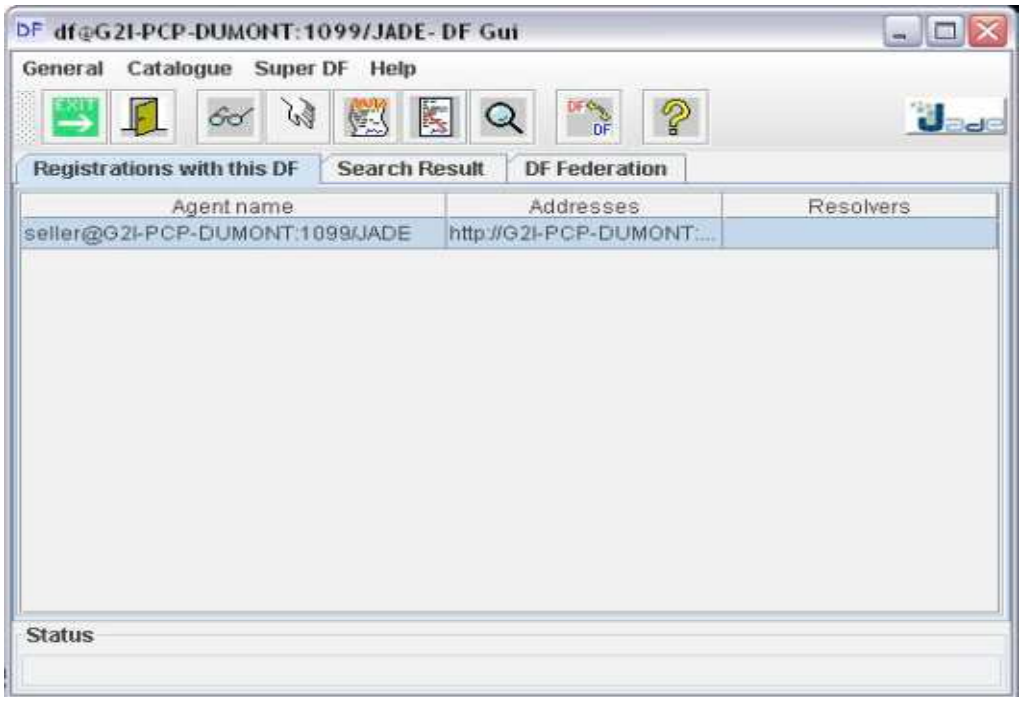


Figure IV.5 Agent Direcory Facilitator DF

c. Agent Sniffer

Quand un utilisateur décide d'épier un agent ou un groupe d'agents, il utilise un agent sniffer. Chaque message partant ou allant vers ce groupe est capté et affiché sur l'interface du sniffer. L'utilisateur peut voir et enregistrer tous les messages, pour éventuellement les analyser plus tard. L'interface de l'agent Sniffer est illustrée par la figure IV.6

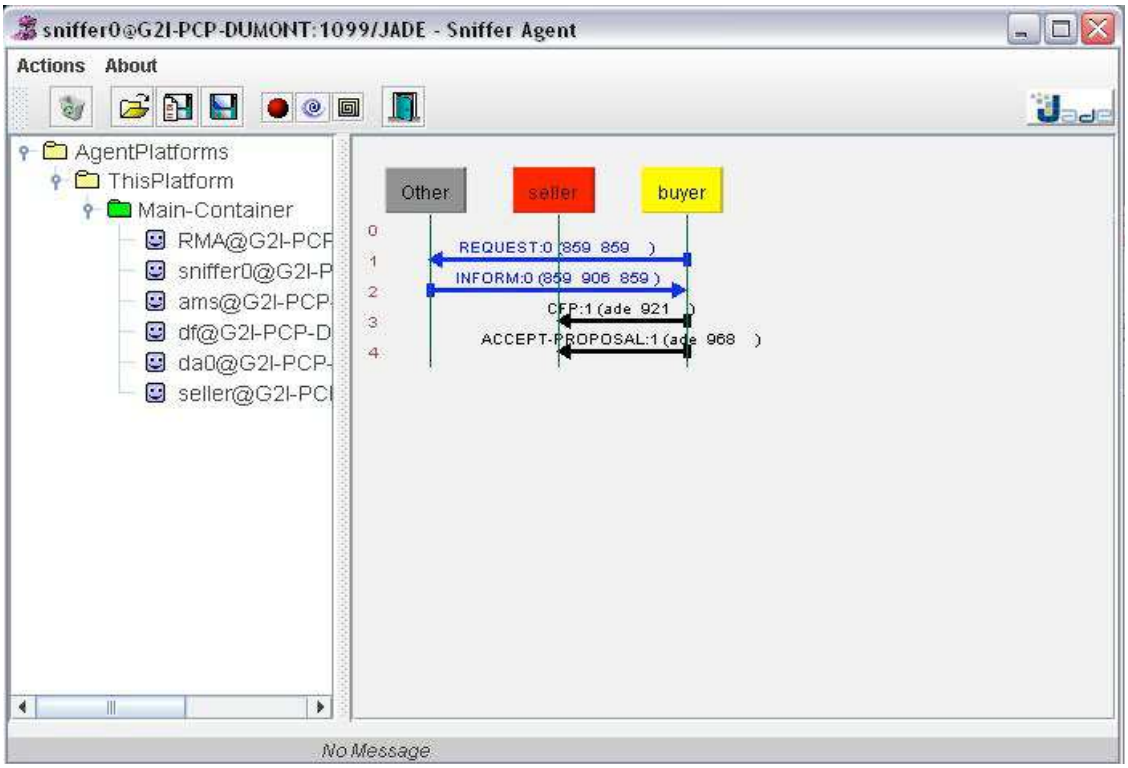


Figure IV.6 Agent Sniffer

d. Agent Dummy

L'outil DummyAgent permet aux utilisateurs d'interagir avec les agents JADE d'une façon particulière. L'interface permet la composition et l'envoi de messages ACL et maintient une liste de messages ACL envoyés et reçus. Cette liste peut être examinée par l'utilisateur et chaque message peut être vu en détail ou même édité. Plus encore, le message peut être sauvegardé sur le disque et renvoyé plus tard. L'interface de l'agent Dummy est illustrée par la figure IV.7.

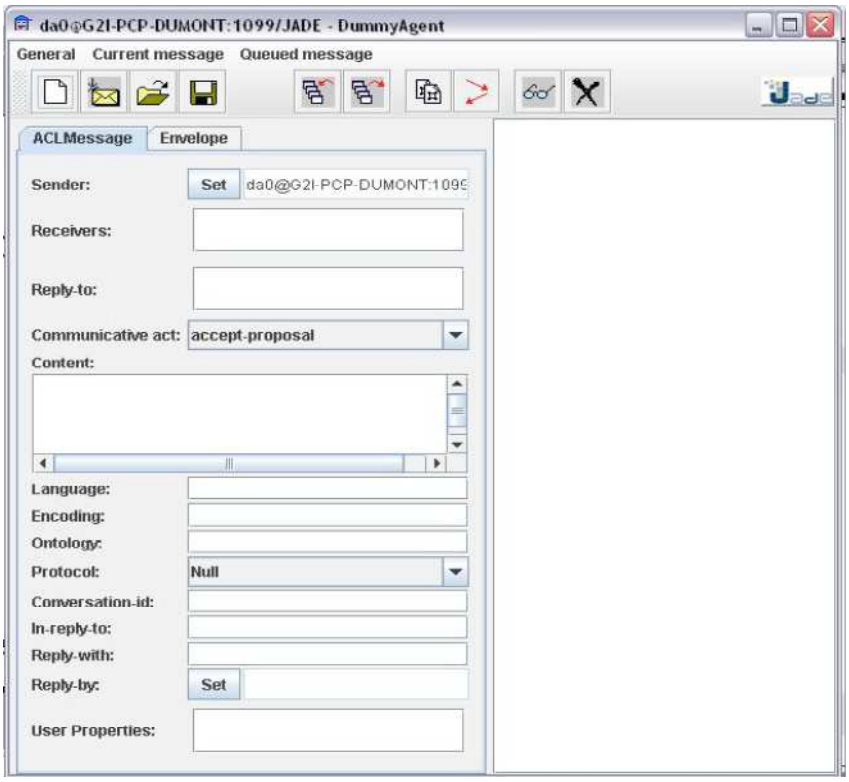


Figure IV.7 Agent Dummy

6.2. Le système test

Pour l'expérimentation de notre approche, on a utilisé plusieurs systèmes de tailles petites et moyennes. Mais pour valider l'expérience on a utilisé le système BOAP : il s'agit d'une application développée avec le langage Java, fournie par le LABO CRIM, vu qu'il a été utilisé dans [Chikhi, 2004] pour estimer l'impact de changement.

BOAP (Boite à Outils pour l'Analyse des Programmes développés au CRIM), est un ensemble d'outils logiciels intégrés qui permet à un expert d'évaluer rapidement le niveau de qualité d'un logiciel.

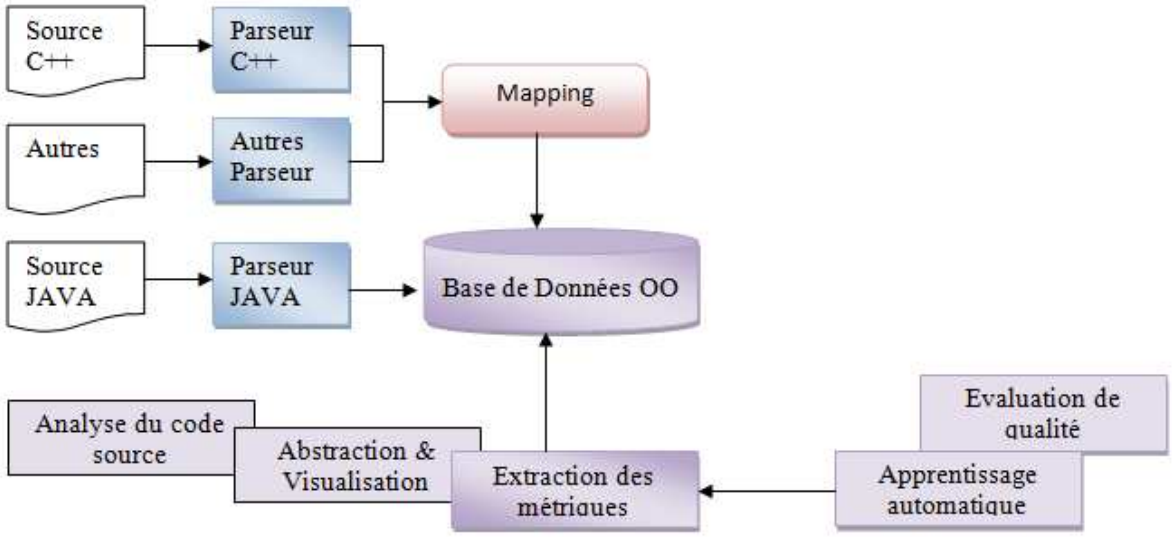


Figure IV.8 Architecture générale de BOAP [Chikhi, 2004].

Les caractéristiques de l'application sont présentées dans le tableau suivant :

Caractéristiques	Système BOAP
Nombre de fichiers	424
Nombre de modules	22
Nombre de classes indépendantes	103
Nombre de classes	430
Nombre de classes de bases	117
Nombre de méthodes	3546
Nombre d'attributs	2247

Table IV.2 Caractéristiques du système étudié

6.3. Mise en ouvre de l'outil

Nous présentons les différentes étapes de mise en œuvre de notre outil. La figure suivante présente le menu principal :

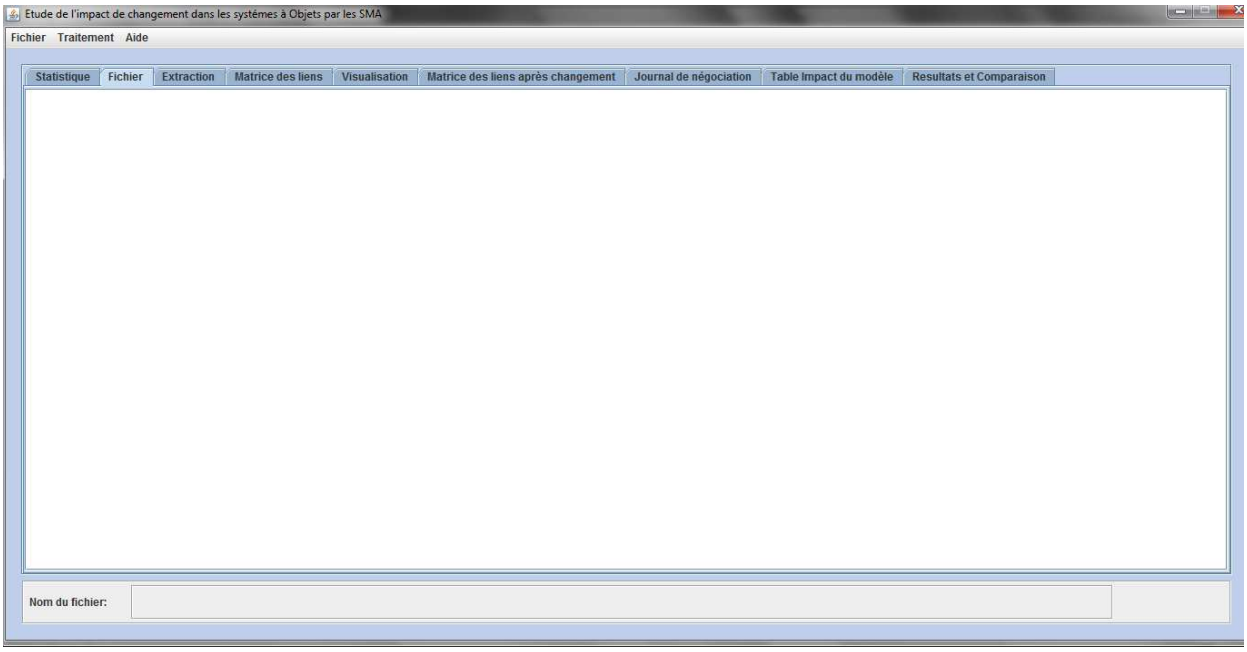


Figure IV.9 Menu principal

Le code source à étudier est chargé en cliquant sur l'icône Charger sous l'icône Fichier dans le Menu (Figure IV.10)

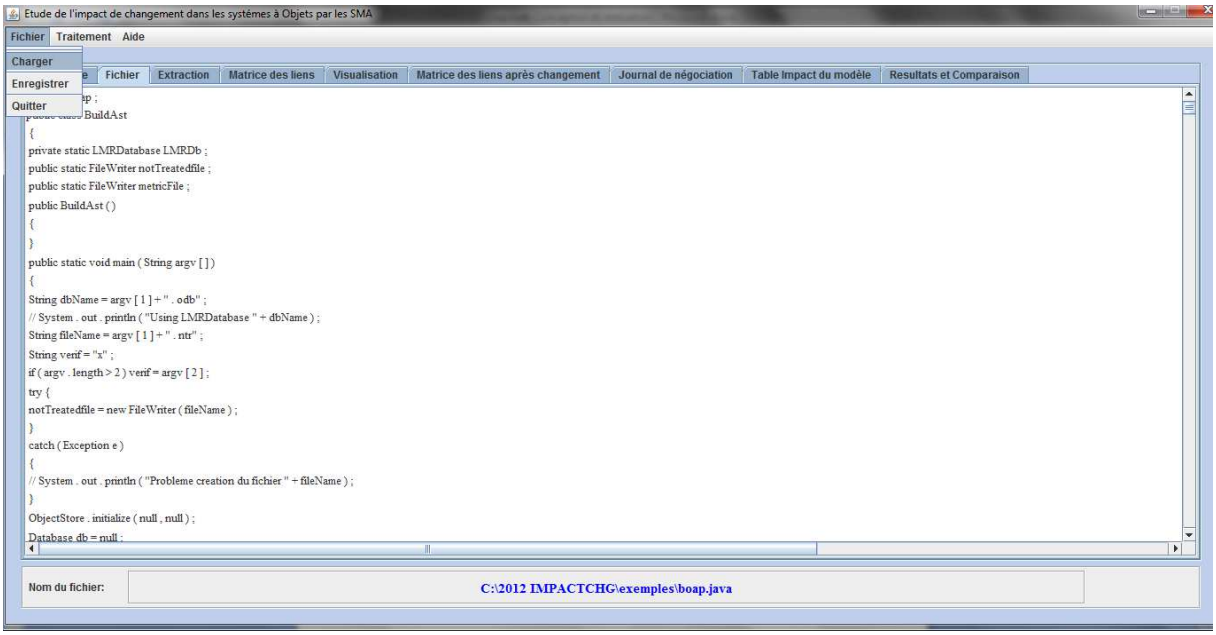


Figure IV.10 Chargement du Code Source

Les agents sont lancés en cliquant sur l'icône Lancer les agents sous l'icône Traitement dans le Menu.

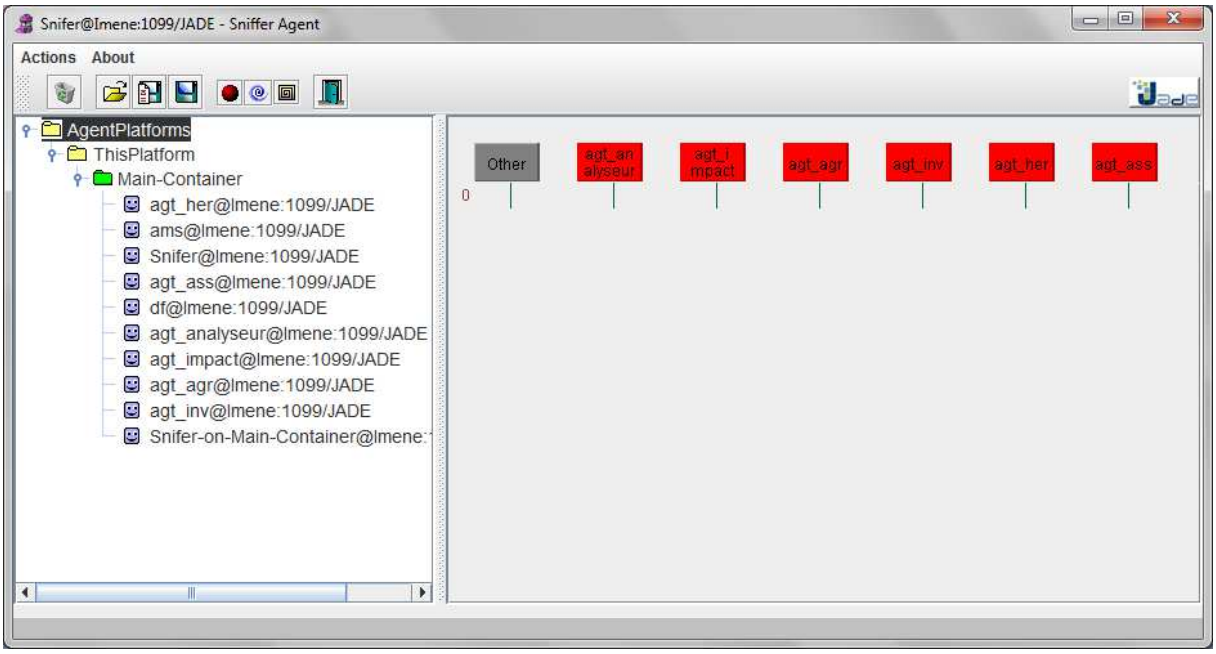


Figure IV.11 Interface de l'agent

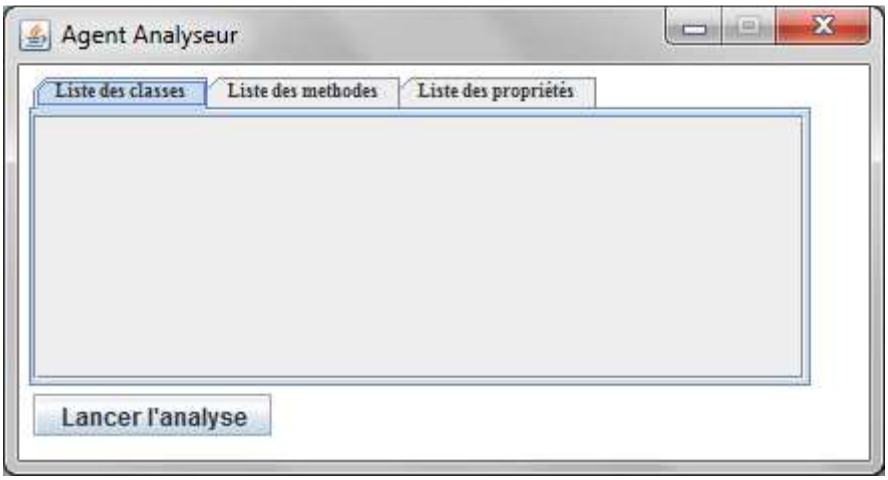


Figure IV.12 Agent Analyseur

Figure IV.13 Agent Impact

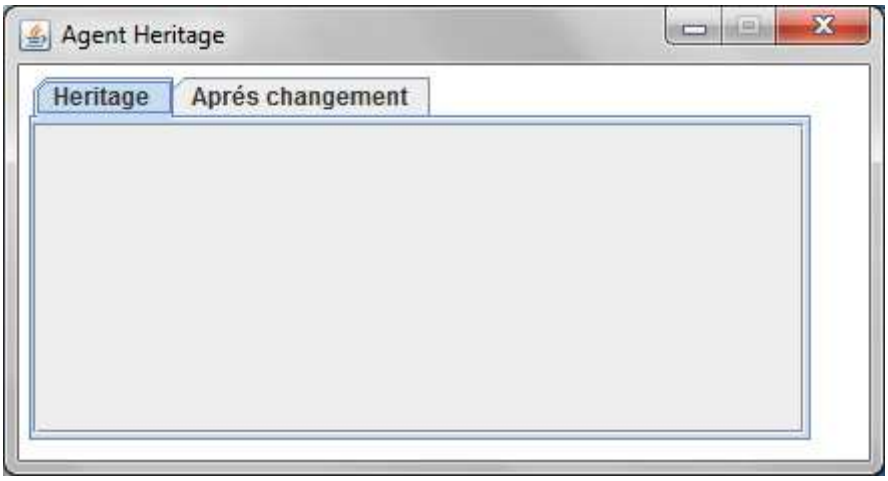


Figure IV.14 Agent Héritage

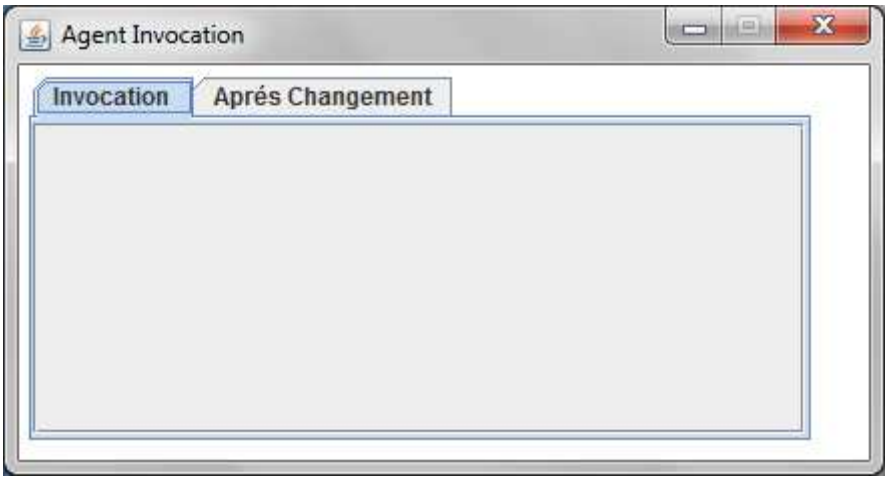


Figure IV.15 Agent Invocation

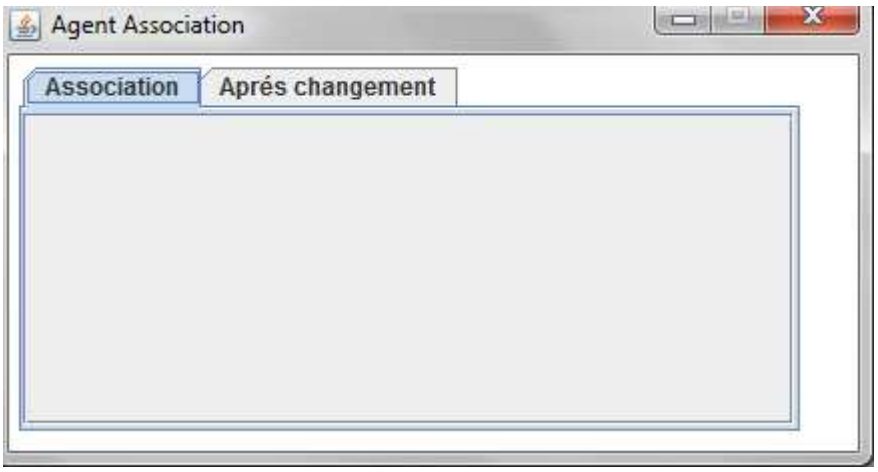


Figure IV.16 Agent Association

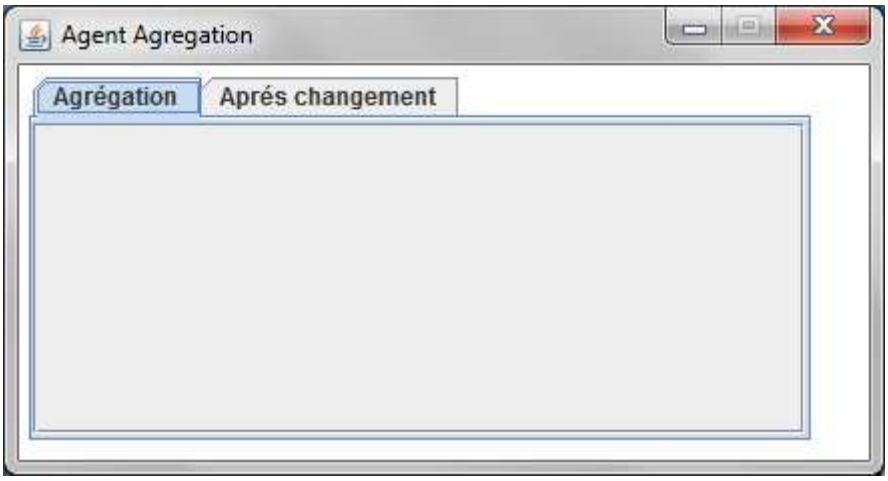


Figure IV.17 Agent Agrégation

La phase d'extraction des composants du système : l'agent analyseur lance le traitement en cliquant sur le bouton lancer l'analyse (figure IV.18)

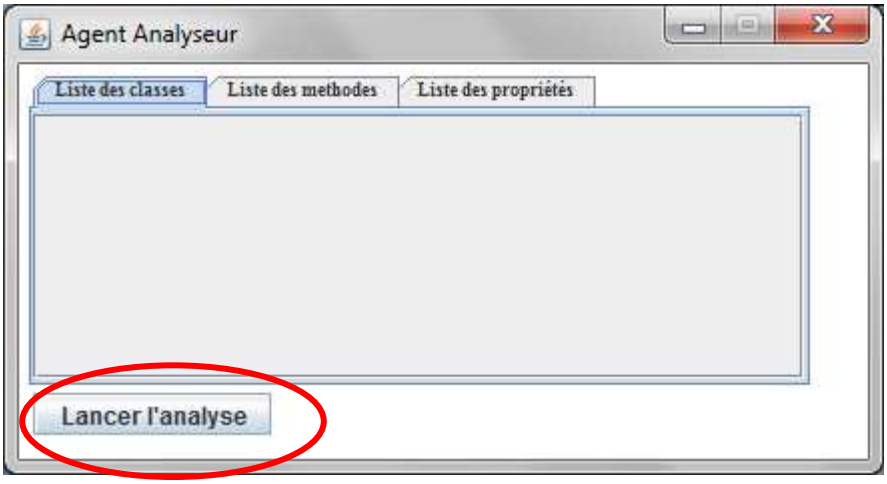


Figure IV.18 Début traitement

Grace à l'agent Sniffer on peut visualiser les différents messages échangés entre les agents durant le traitement (La figure IV.19), l'agent analyseur envoi aux agents «Héritage, Association, Agrégation, Invocation » un message « **Request** » pour le début de traitement d'extraction et informe l'agent Impact du début de traitement en lui envoyant un message «**Inform** ». Le message « **Confirm** » signifie la fin du traitement, les liens sont localisés.

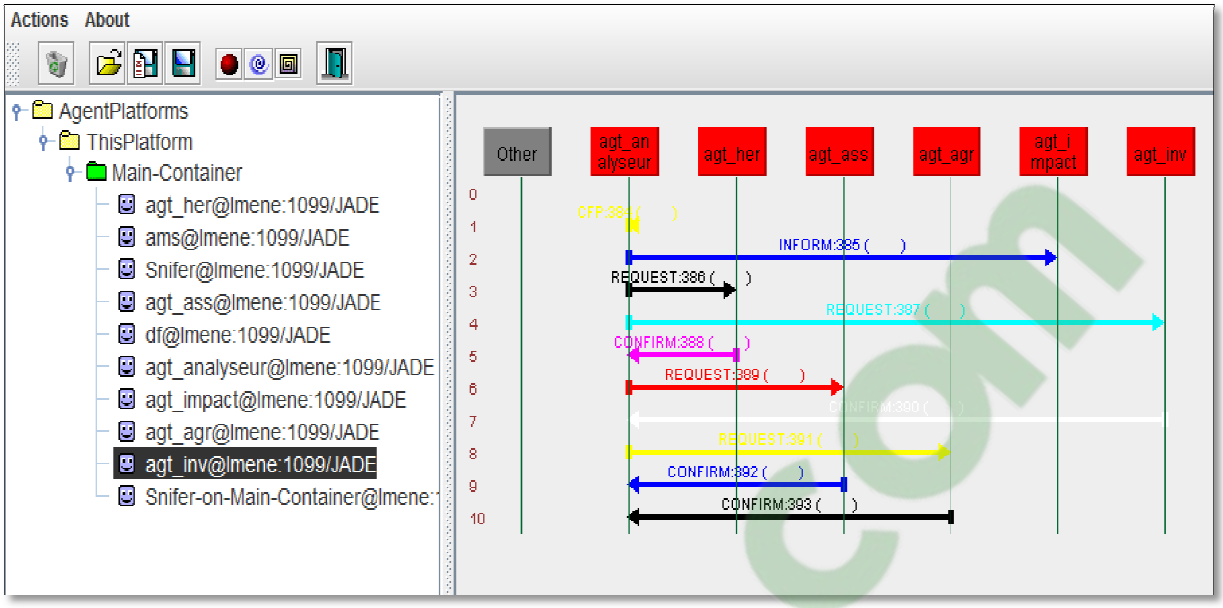
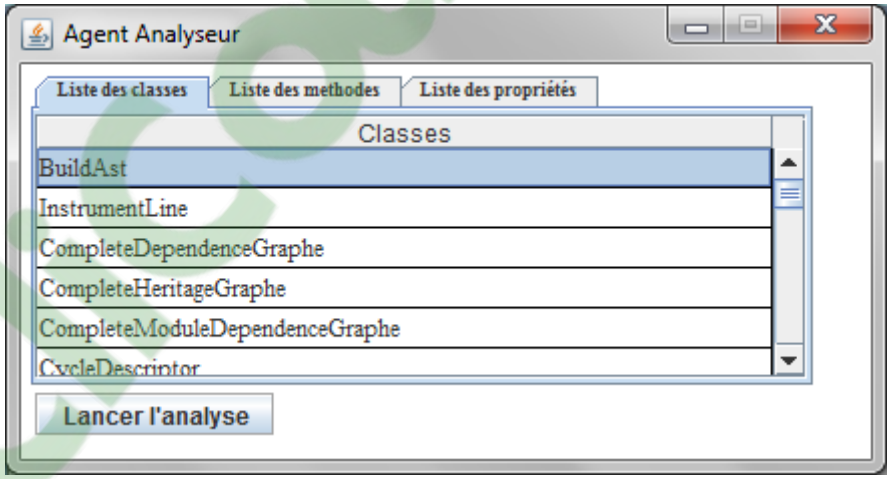
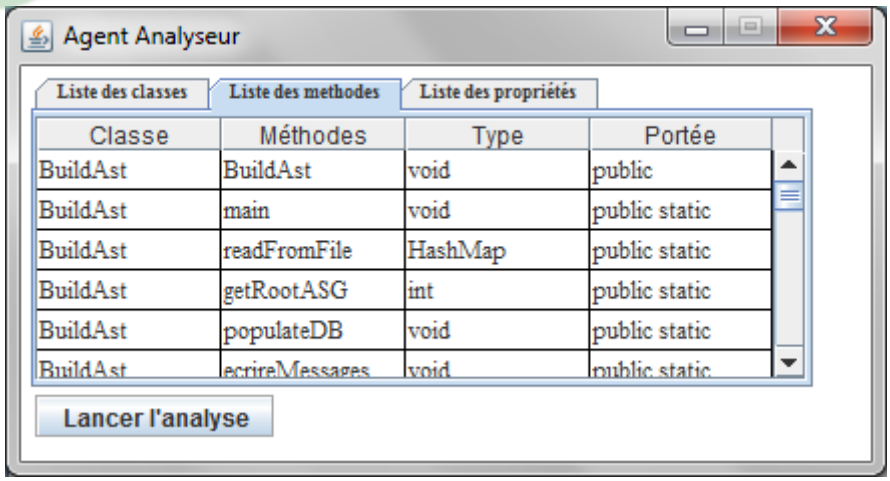


Figure IV.19 Lancement du traitement d'extraction

Les agents exécutent leur tâches, affichent le résultat et informent l'agent analyseur de la finalisation du travail, les figures suivantes montrent les résultats obtenus par chaque Agent :





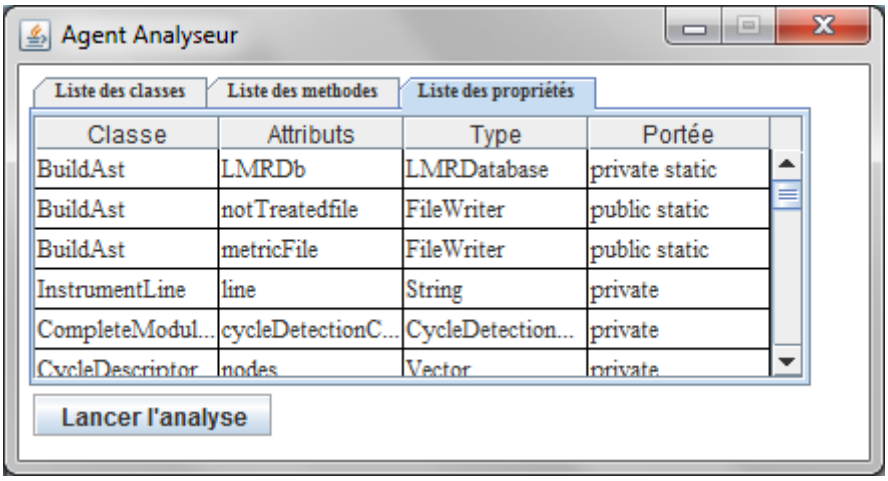


Figure IV.20 Exécution de l'agent analyseur

L'agent héritage extrait toutes les classes en héritage, par exemple on a la classe DependenceGraphe qui hérite de la classe Graphe Abstraction (voir figure IV.21)

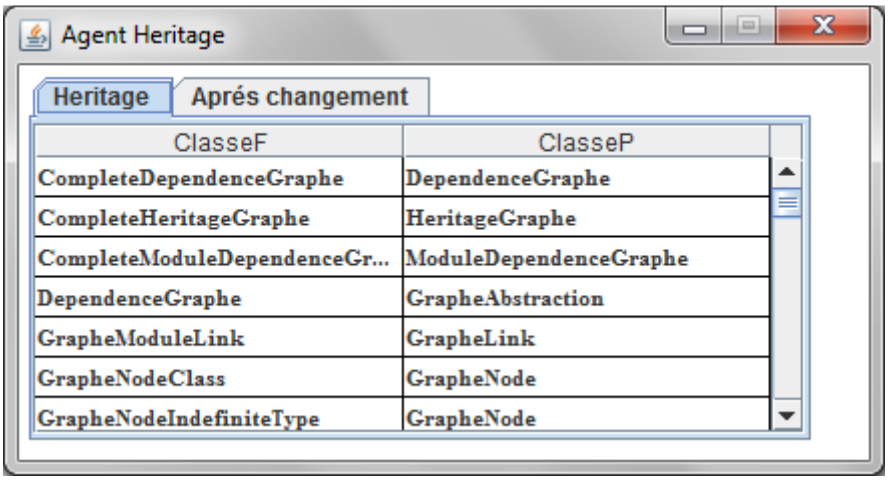


Figure IV.21 Exécution de l'agent héritage

L'agent Agrégation est responsable de la localisation des classes en agrégation, exemple la classe GrapheLink qui est en agrégation avec la classe GrapheNode, comme la définition de Graphe Link implique deux objets de la classe GrapheNode.

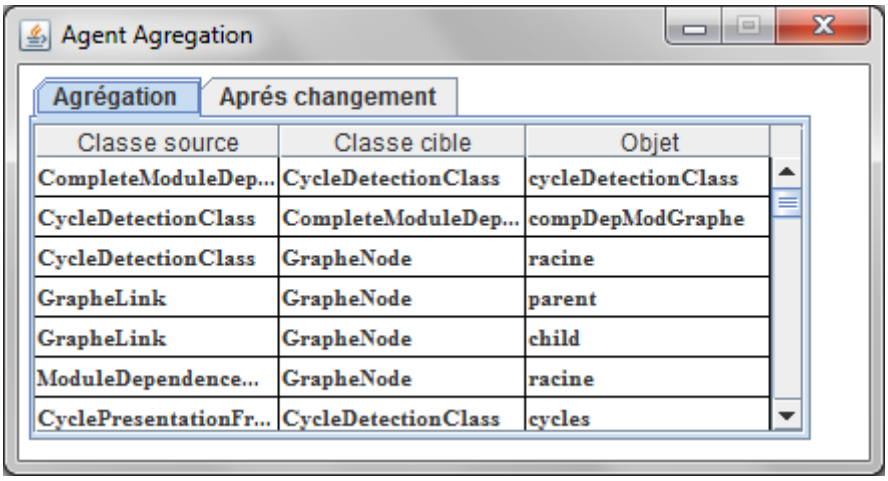


Figure IV.22 Exécution de l'agent Agrégation

L'agent association recherche les classes en association dans le système en entrée, on peut voir sur la figure IV. Plusieurs classes sont en association avec la classe BuildAst sur la propriété LMRDb.

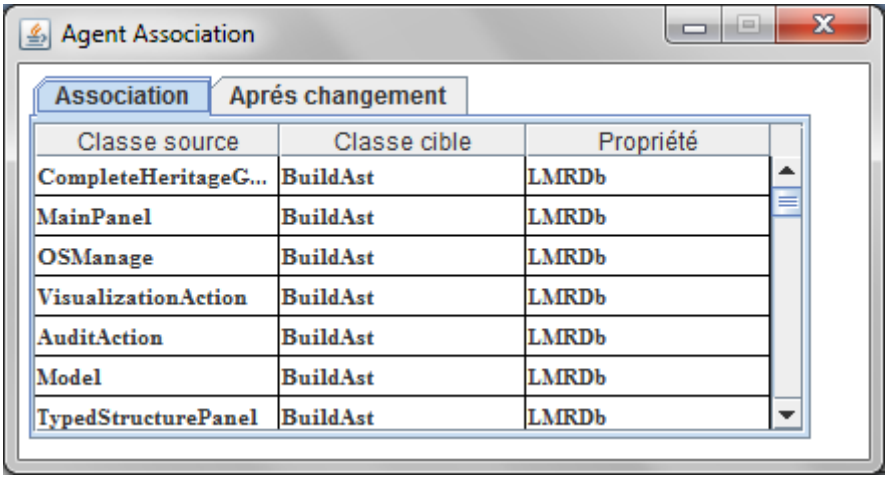


Figure IV.23 Exécution de l'agent Association

L'agent invocation recherche les invocations entre classes qui existent dans le système source, la classe BuidAst invoque 5 méthodes de la classe InstrumentLine (figure IV.24)

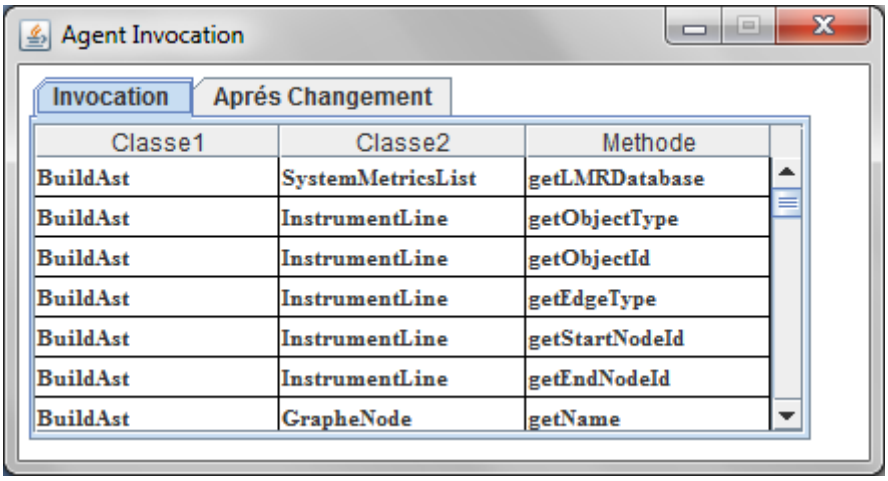
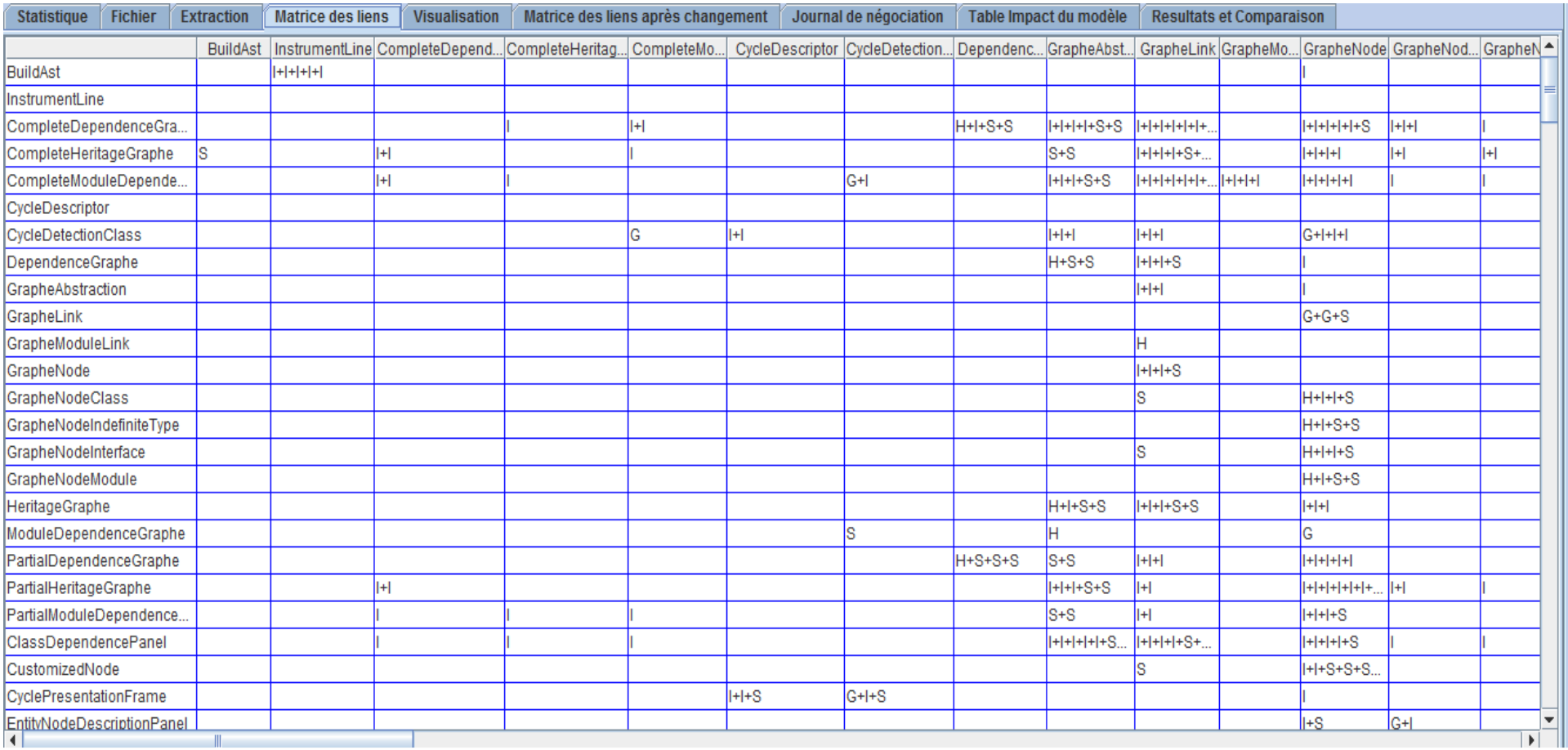


Figure IV.24 Exécution de l'agent Invocation

La matrice des liens est alimenté par les agents « héritage, invocation, agrégation et association » comme montré sur la figure IV.25.

La table d'impact selon le modèle est calculée à partie de la matrice des liens précédente (figure IV.26)



	BuildAst	InstrumentLine	CompleteDepend...	CompleteHeritag...	CompleteMo...	CycleDescriptor	CycleDetection...	Dependenc...	GrapheAbst...	GrapheLink	GrapheMo...	GrapheNode	GrapheNod...	GrapheN
BuildAst		I+I+I+I+I										I		
InstrumentLine														
CompleteDependenceGra...				I	I+I			H+I+S+S	I+I+I+I+S+S	I+I+I+I+I+...		I+I+I+I+I+S	I+I+I	I
CompleteHeritageGraphe	S		I+I		I				S+S	I+I+I+I+S+...		I+I+I+I	I+I	I+I
CompleteModuleDepende...			I+I	I			G+I		I+I+I+S+S	I+I+I+I+I+...	I+I+I+I	I+I+I+I+I	I	I
CycleDescriptor														
CycleDetectionClass					G	I+I			I+I+I	I+I+I		G+I+I+I		
DependenceGraphe									H+S+S	I+I+I+S		I		
GrapheAbstraction										I+I+I		I		
GrapheLink												G+G+S		
GrapheModuleLink										H				
GrapheNode										I+I+I+S				
GrapheNodeClass										S		H+I+I+S		
GrapheNodeIndefiniteType												H+I+S+S		
GrapheNodeInterface										S		H+I+I+S		
GrapheNodeModule												H+I+S+S		
HeritageGraphe									H+I+S+S	I+I+I+S+S		I+I+I		
ModuleDependenceGraphe							S		H			G		
PartialDependenceGraphe								H+S+S+S	S+S	I+I+I		I+I+I+I+I		
PartialHeritageGraphe			I+I						I+I+I+S+S	I+I		I+I+I+I+I+...	I+I	I
PartialModuleDependence...			I	I	I				S+S	I+I		I+I+I+S		
ClassDependencePanel			I	I	I				I+I+I+I+I+S...	I+I+I+I+S+...		I+I+I+I+S	I	I
CustomizedNode										S		I+I+S+S+S...		
CyclePresentationFrame						I+I+S	G+I+S					I		
EntityNodeDescriptionPanel												I+S	G+I	

Figure IV.25 La matrice des liens

	Type de var.	Porté de var.	Porté de méthode	Supp. de classe	Supp. d'une var.
BuildAst	13	12	9	21	13
InstrumentLine	2	1	12	13	2
CompleteDependenceGraphe	0	0	13	13	0
CompleteHeritageGraphe	0	0	8	8	0
CompleteModuleDependenceGraphe	0	0	9	12	0
CycleDescriptor	4	3	7	10	4
CycleDetectionClass	6	5	2	9	6
DependenceGraphe	6	5	2	11	6
GrapheAbstraction	23	22	45	70	23
GrapheLink	63	62	75	138	63
GrapheModuleLink	0	0	6	7	0
GrapheNode	55	54	99	161	55
GrapheNodeClass	0	0	18	19	0
GrapheNodeIndefiniteType	0	0	11	11	0
GrapheNodeInterface	0	0	16	16	0
GrapheNodeModule	0	0	12	13	0
HeritageGraphe	0	0	7	11	0
ModuleDependenceGraphe	3	2	0	4	3
PartialDependenceGraphe	0	0	4	4	0
PartialHeritageGraphe	0	0	7	7	0
PartialModuleDependenceGraphe	0	0	4	4	0
ClassDependencePanel	33	32	10	42	33
CustomizedNode	0	0	11	11	0
CyclePresentationFrame	128	127	31	159	128
EntityNodeDescriptionPanel	128	127	33	160	128
FrameVisualization	66	65	42	110	66
FrameVisualizationCommandPanel	78	77	35	113	78
GrapheDescription	17	16	28	46	17
GrapheDescriptionList	18	17	10	30	18
GraphicDescriptionFrame	75	74	32	107	75
GraphicDescriptionPanel	0	0	3	5	0
HeritagePanel	34	33	10	43	34

Figure IV.26 Table d'impact selon modèle

Une fois la phase d'extraction est terminée, les différents liens entre classes sont localisés, l'agent analyseur en fonction de ces résultats établit la représentation graphique. Pour mieux présenter le graphe l'agent analyseur calcul la matrice d'incidence et la matrice de niveau.

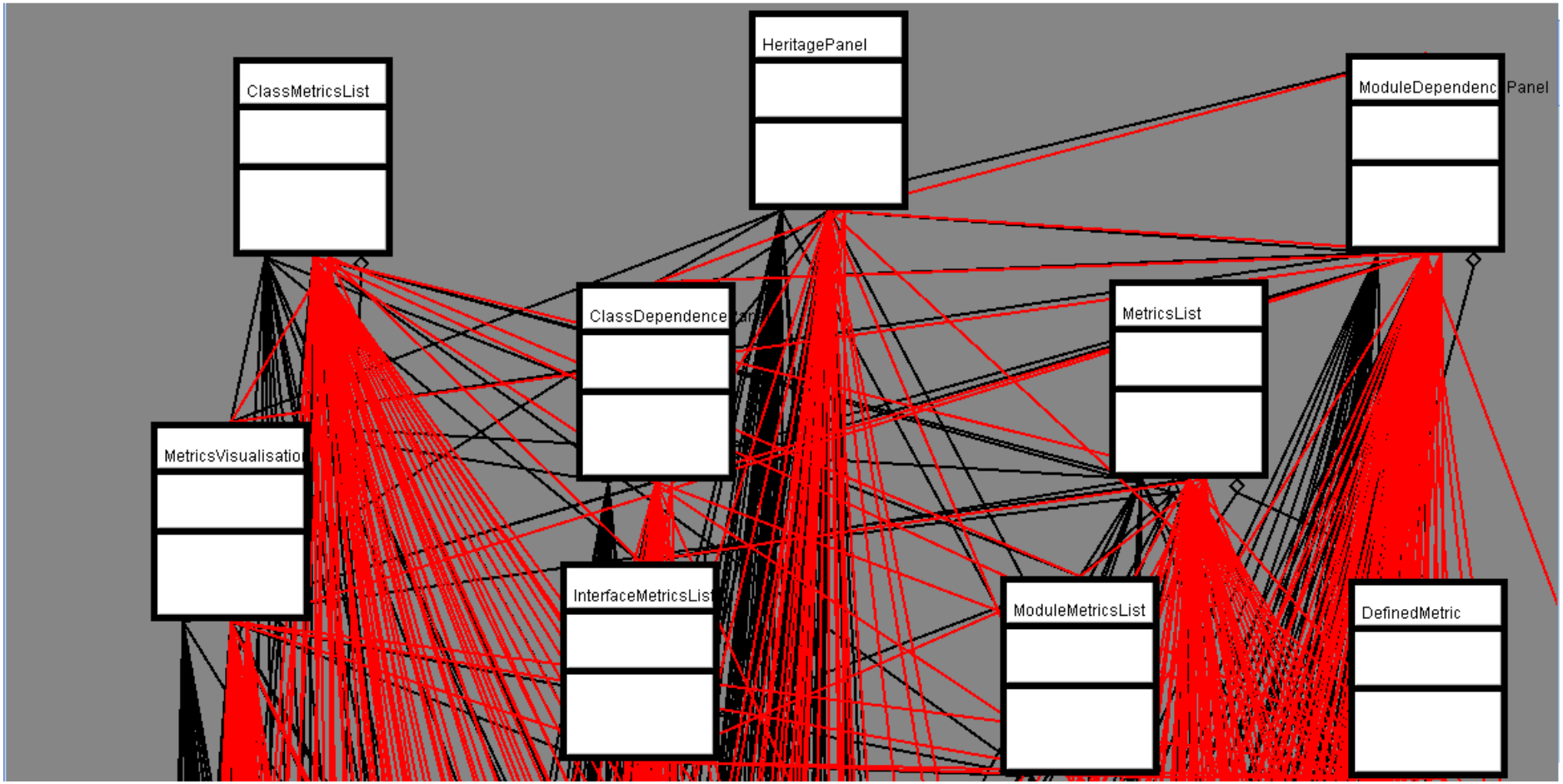


Figure IV.27 Representation graphique (extrait du graphe global)

◇— Signifie Agrégation —— Signifié Association
◁— Signifie Héritage —— Signifié Invocation

Phase de l'étude d'impact :

Dans cette étape nous effectuons des changements sur le code source, relancer les agents pour détecter l'impact du changement. Les changements choisis dans notre étude sont :

a. **Protée de méthode**: changement de la portée de la méthode **getName** de « Public » à « Private » dans la classe **GrapheNode**.
b. **Type de variable:** changement du type de la propriété **CLASS_TYPE** de «int » à « String» dans la classe **GrapheNode.**
c. **Porté de variable :** changement de la portée de la propriété **name** de « Public » à « Private » dans la classe **GrapheNode.**
d. **Suppression de classe :** Suppression de la classe **GrapheNode.**
e. **Suppression d'une variable :** Suppression de la propriété **INTERFACE_TYPE** dans la classe **GrapheNode.**

Les changements sont portés concrètement sur le système (code), l'agent analyseur contact les différents agents et les informe qu'un changement a été effectué sur le code source, l'agent impact avec les agents héritage, association, invocation et agrégation entame la phase d'étude d'impact et une l'impact localisé ils envoient une confirmation à l'agent analyseur.

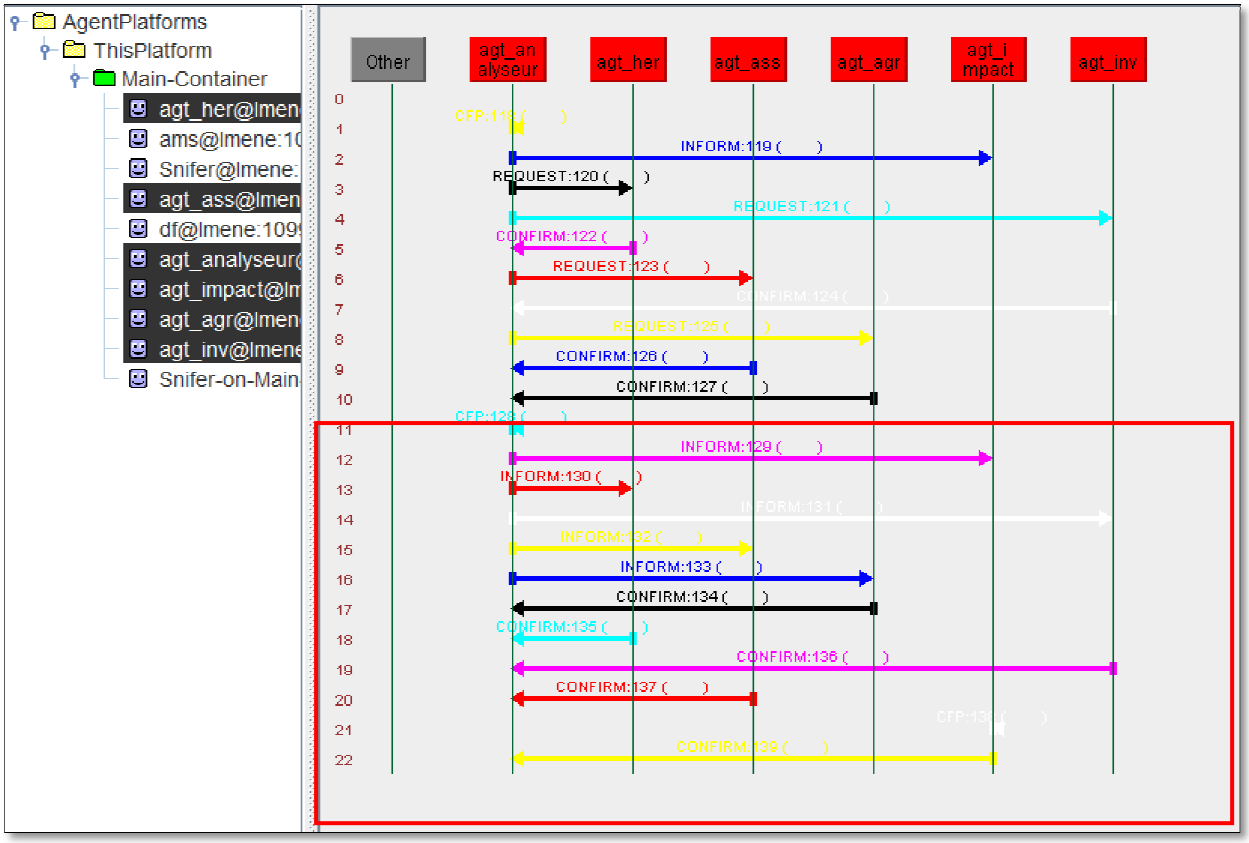
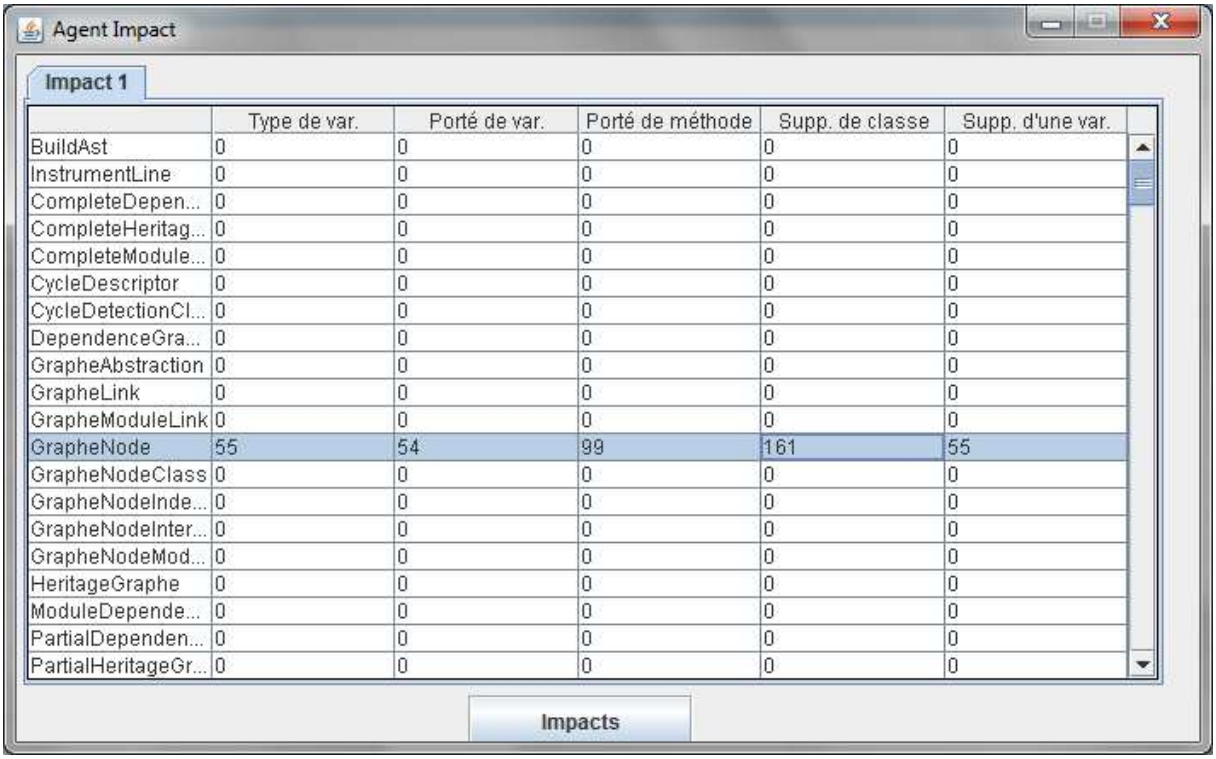


Figure IV.28 Lancement du traitement de recherche après changement

L'agent Impact établit la table d'impact dans la figure IV.29.



	Type de var.	Porté de var.	Porté de méthode	Supp. de classe	Supp. d'une var.
BuildAst	0	0	0	0	0
InstrumentLine	0	0	0	0	0
CompleteDepen...	0	0	0	0	0
CompleteHeritag...	0	0	0	0	0
CompleteModule...	0	0	0	0	0
CycleDescriptor	0	0	0	0	0
CycleDetectionCl...	0	0	0	0	0
DependenceGra...	0	0	0	0	0
GrapheAbstraction	0	0	0	0	0
GrapheLink	0	0	0	0	0
GrapheModuleLink	0	0	0	0	0
GrapheNode	55	54	99	161	55
GrapheNodeClass	0	0	0	0	0
GrapheNodeInde...	0	0	0	0	0
GrapheNodeInter...	0	0	0	0	0
GrapheNodeMod...	0	0	0	0	0
HeritageGraphe	0	0	0	0	0
ModuleDepende...	0	0	0	0	0
PartialDependen...	0	0	0	0	0
PartialHeritageGr...	0	0	0	0	0

Figure IV.29 Table d'impact après changements

La figure IV.30 Représente une comparaison des résultats d'impact obtenus par l'expérimentation à base des SMA et ceux calculé en fonction du modèle raffiné (étudier en chapitre II).

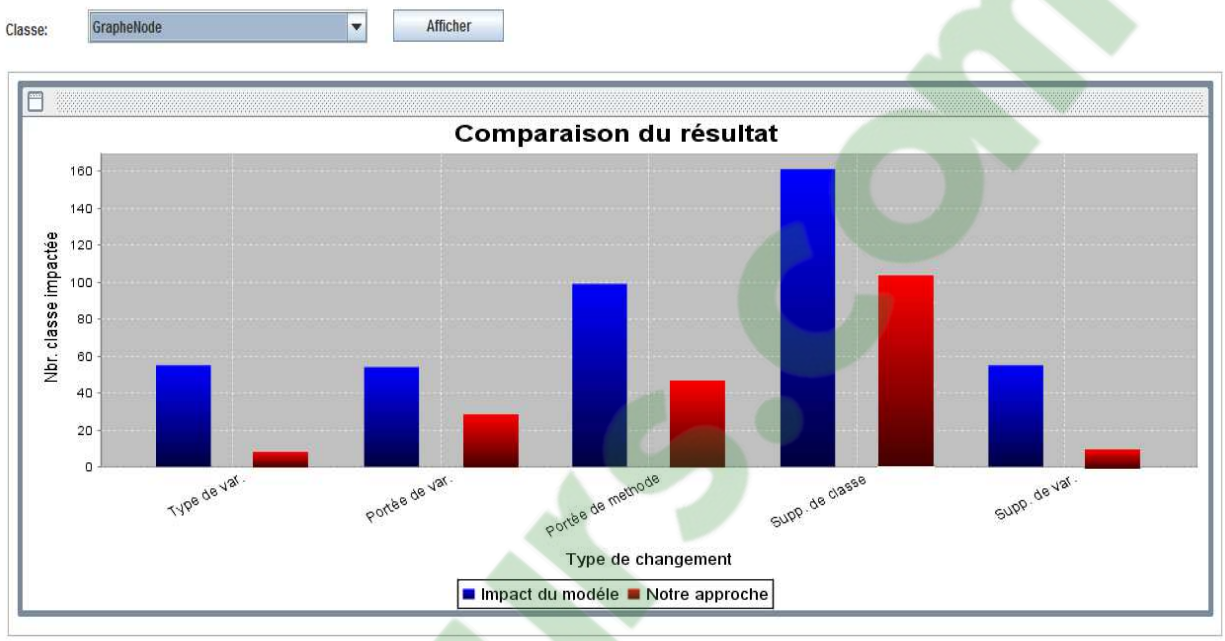


Figure IV.30 Comparaison des résultats

Les résultats obtenus sur le graphe sont traduit dans le tableau comparatif pour les examiner:

Classe Graphe -Node	***Résultats à base du modèle***				
	Type de var.	Portée de var.	Portée de meth.	Supp. de classe	Supp. De var.
	55	54	99	161	55
	Résultats à base de l'expérimentation				
	Type de var.	Portée de var.	Portée de meth.	Supp. de classe	Supp. De var.
	6	27	44	104	7

Table IV.3 Confrontation des résultats

En examinons la table IV.3, on peut remarquer que l'impact de changement calcul » à base du modèle est toujours supérieure à celui obtenu par notre approche, l'impact déduit par le modèle prend toujours la valeur maximale du changement. Si on prend l'exemple du changement du type de variable, selon le modèle l'impact toutes les classes en association

avec la classe changée et la classe elle-même seront impactées tandis que notre système détecte seulement les classes qui font référence à la variable modifiée.

4. Conclusion

Au terme de ce chapitre, nous avons présenté une approche de validation d'un modèle d'impact de changement à base d'un système multi agents, nous avons détaillé l'architecture du système, la structure des agents et leurs rôles. Aussi nous avons présenté l'étude d'impact de changement réelle sur le système BOAP accompagnée d'une discussion des résultats obtenus.

Conclusion Générale et Perspectives

Notre étude est pluridisciplinaire et s'articule autour de plusieurs axes: la maintenance des logiciels, l'étude d'impact de changement et les systèmes multi agents. La maintenance est une phase très couteuse et ceci est dû à:

- La difficulté de la compréhension du problème qui est généralement liée à la compréhension du logiciel maintenu.

- La maîtrise de la totalité des effets de propagation des changements dans le système, particulièrement les systèmes de taille importante.

L'analyse de l'impact du changement est l'une des techniques qui a connu de l'importance dans le domaine de la maintenance, son but est de permettre aux responsables de la maintenance d'évaluer le coût du changement à priori, i.e., avant d'entamer l'implémentation du changement.

L'utilisation d'un système multi agents SMA nous a alors semblé une option innovante et prometteuse pour la validation d'un modèle d'impact de changement vu la taille importante des systèmes étudies dans notre cas. Les systèmes multi agents permet de découper le problème en sous problèmes simple (la notion de modularité), les agents partagent et coopèrent entre eux pour la résolution du problème.

Au début, nous avons procédé à l'extraction des différents composants du système, puis à la représentation graphique du résultat.
Par la suite, nous avons effectué des changements divers sur le système. Ensuite nous avons comparé les résultats d'impacts de changements déduits par le modèle d'impact en question avec ceux obtenus manuellement par expérimentation.

Nous terminons cette conclusion en évoquant quelques perspectives de recherche :

- Enrichir et ajouter de nouveaux modules à notre architecture proposée.
- Prévoir un agent pour la mise à jour de code source après tout changement effectué, en fonction des résultats de l'étude d'impact.

Bibliographie

[Abdi, 2007] Mustapha Kamel ABDI, Analyse et Prédiction d'impact de changementdans système à objets, Thèse de doctorat d'état en Informatique, Université Es-Sénia d'Oran, Avril 2007.

[Abran, 1995] A Abran, P Bourque, R Brisebois, Côté, V, La Ré ingénierie du Logiciel : un bref tour d'horizon. In Le génie Logiciel, reproduit des Actes de la conférence "Le génie logiciel et ses applications", huitièmes journées internationales (GL95), vol. 38, EC2 & Cie, Paris, La Défense, 1995, Pages 41-47.

[Abran, 2005] A Abran, J Moore, , P Bourque, Dupuis, L Tripp (2005) Guide to the Software Engineering Body of Knowledge - 2004 Version - SWEBOK IEEE-Computer Society Press, April 2005, 200 pages, isbn:0-7695-2330-7.

[Aho**,** 1983**]** A. V. Aho, J. E. Hopcroft and J. D. Ullman, "Data Structures and Algorithms", Addison-Wesley Publ. Comp., 1983.

[Antoniol**,** 1999**]** G. Antoniol, G. Canfora and A. De Lucia. Estimating the size of changes for evolving object Oriented Systems: a Case Study. In Proceedings of the 6th International Software Metrics Symposium, pages 250-258, Boca Raton, Florida, Nov 1999.

[Arnold & Bohner, 1993] R. S. Arnold and S. A. Bohner, Impact Analysis - Towards A Framework for Comparison. Proceedings of the Conference on Software Maintenance, September 1993, Pages 292-301.

[Baran, 1995] Baran, A. Bourque, P. Brisebois, R. Côté, "La Ré ingénieurie du logiciel :un Bref tour d'Horizon" , In le génie logiciel, reproduit des Actes de la conférence "Le génie Logiciel et ses applications", huitièmes journées internationales (GL95), Paris, 1995.

[Bohner, 1996] S. A. Bohner, Impact Analysis in Software Change Process : A year 2000 perspectives. In International Conference on Software Maintenance, Pages 42-51, 1996.

[Boissier et al, 1994] O.Boissier,Y.demazeau « Multisensor Fusion and Integration for Intelligent Systems»,.IEEE International Conference on MFI apos,1994.

[Boissier, 2001] Olivier Boissier,Systèmes multi-agents (coordination), ENS Mines Saint-Etienne, 2001.

[Bond et Gasser, 1988] A.H. Bond et L.Gasser. Reading in distributed artificial intelligence, Morgan Kaufmann publishers, Inc, 1988

[Boukerche et Abdi, 2013] Imene BOUKERCHE, Mustapha Kamel ABDI. Evolution Study of Object-oriented Systems by Multi-agent systems. Dans le procedding de ICIST 2013 international conference on information Systems and technologies, mars 22-24, TANGER, MAROC, 2013

[Briand, 1999] L.C. Briand, Jurgen Wust, Hakim Lounis, Using Coupling Mesurement for Impact Analysis in Object Oriented Systems. IEEE International Conference on Software Maintenance (ICSM), 1999.

[Briand, 1999] L. C. Briand, S. Morasca, et V.R. Basili, Defining and Validating Measures for Object-Based High Level Design Metrics. IEEE Transactions on Software Engineering, Vol. 25, No. 5, 1999.

[Brooks, 1991] R.A.Brooks, «Intelligence without reason», Proceedings of the Joint Conference on Artificial Intelligence, Sydney, Australie, 1991.

[Chaib-draa, 1996] B.Chaib-draa, «Interaction between agents in routine, familiar and unfamiliar situations», International Journal of Intelligent and Cooperative Information Systems, 1996.

[Chaumun, 1998**]** M. A. Chaumun, H. Kabaili, R. K. Keller and F. Lustman, “A Change Impact Model for Changeability Assessment in Object-Oriented Software Systems”, in Proceedings of the Third European Working Conference on Software Maintenance and

Reengineering, Pages 130-138, Amesterdam, the Netherlands, March 1998.

[Chevrier, 1993] V.Chevrier, «Etude et mise en œuvre du paradigme Multi Agents l'atome à Gtmas», Thèse de l'Université de Nancy I ,1993.

[Chikhi, 2004] Leila Chikhi, Département d'informatique et de recherché opérationnelle, Faculté des arts et des sciences "estimation d'impact de changement dans les systems à objets".Mémoire présenté à la faculté des études supérieures pour l'obtention du grade maitrise science en informatique, Octobre 2004.

[Chikofsky Elliot, 1990] Chikofsky Elliot J. and Cross, James H, Reverse Engineering and Design Recovery : A Taxonomy. IEEE Software, Vol. 7, No.1, Pages13-17, 1990.

[Cantave, 2001] M. H. Robert Cantave, Abstractions via un Modèle Générique d'Applications Orientées Objets. Maître ès sciences (M.Sc.), Université de Laval, Août 2001.

[Cohen et al, 1990] P.Cohen, H.J.Levesque, « Intention is choice with commitment», Artificial Intelligence, 42(3) 1990.

[Dahane, 2011] M.Dahane « Validation empirique d'un modèle d'impact de changement pour les systèmes à Objets », mémoire Magistère, Université ES-Senia, ORAN, 2011

[Demazeau et Muller, 1990] Y. Demazeau et J. Muller. Decentralised artificial intelligence. Amsterdam, Holland. Elseiver Science Publisher B.V. /North-Holland, 1990.

[Demazeau, 1995] Demazeau, Y. From interactions to collective behavior in agent-based systems. In European Conference on Cognitive Science, Saint-Malo France, 1995.

[Demazeau, 1996] Demazeau, Y. and Costa, A. R. Populations and organisations in open multi-agent systems. In 1st Symposium on Parallel and Distributed AI, Hyderabad, India, 1996

[Dinedane, 2011] Z.Dinedane : « vers une approche d'aide à la décision pour la maintenance des systèmes à objets », mémoire Magistère, Université ES-Senia, ORAN, 2011.

[Drogoul, 1993] A.Drogoul, «De la simulation Multi-Agent à la résolution collective de problèmes», Thèse de l'Université P et M Curie, 1993.

[Durfee et al, 1987] E.H.Durfee, V.R.Lesser, «Using partial Global Plans to coordinate distributed problem solving ». Proceeding of the 10t IJCAI, Italy, 1987.

[Engelmore et al, 1988] R.Engelmore,T.Morgan, «Blackboard system», Addison-Wesley, reading, MA, USA, 1988.

[Erceau, 1993] J.Erceau, «Intelligence Artificielle Distribuée et Systèmes Multi-Agents de la théorie aux applications», 23ème Ecole Internationale d'Informatique de l'AFCET, Neuchâtel, 1993.

[Erceau & Ferber, 1991] J.Erceau & J.Ferber. *L'intelligence artificielle distribuée*, La recherche, no 733, vol.22, juin 1991, pp. 750-758.

[Fayech, 2003] B. Fayech. Régulation des réseaux de transport multimodal : Systèmes multi-agents et algorithmes évolutionnistes. Thèse de doctorat, Ecole Centrale de Lille/ Université des Sciences et Technologies de Lille, Octobre 2003.

[Ferber, 1995] J.Ferber, «Les systèmes multi-agents, vers une intelligence collective», InterEditions, Paris, 1995.

[Ferber, 1997] J.Ferber, «Les systèmes multi-agents: un aperçu général. Technique et Science Informatiques, vol.16 n°8,1997.

[Finin, 1993] T.Finin, «Specification of the KQML Agent-Communication Language» DARPA Knowledge Sharing Initiative, External Interface Working Group, 1993.

[Fipa, 1999] Fipa « Specificaion: Agent Communication Langage Foundation Intelligent Physical Agents», 1999.

[Florez, 1999] R. Florez. Towards astandardization of multi-agent system frameworks. ACM Crosswords Student Magazine, 1999.

[Glass et Noiseux, 1981] Glass, R. L. and Noiseux, Ronald A., Software Maintenance Guide Book, Prentice-Hall, Inc., 1981.

[Han, 1997] J. Han. " Supporting Impact Analysis and Change Propagation in Software Engineering Environments" in Proceedings of the STEP97, London,England, pages 172-182, July 1997.

[Harroldet, 1994] M. J. Harroldet G. Rothermel, “Selecting Regression Testing for Object-Oriented Software”, in proceedings of International Conference on Software Maintenance (ICSM ‘94), Victoria, CA, 1994.

[Hoc, 1996] J.Hoc «supervision et controle de processus la cognition en situation dynamique», PUG, Greboble, 1996.

[Huhns 1987] Michael N. Huhns, ed., Distributed Artificial Intelligence, Pitman Publishing, London, 1987.

[IEEE, 1998] IEEE STD 1219, Standard for Software Maintenance, 1998.

[IEEE, 1993] Computer Society Press, Standards Collection – Software Engineering, The Institute of Electrical and Electronics Engineers, Inc., 1993.

[ISO/IEC, 1995] ISO/IEC 12207, Information Technology, Software Life Cycle Process, 1995.

[Jarras et al, 2002] I.Jarras, B.Chaib-draa, «Aperçu sur les systèmes multi agents», Série Scientifique, Montréal, juillet 2002.

[Jennings, 1998] 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.

[Kabaili , 2002] H.Kabaili, Rouldof K. Keller and Franàoic Lustman, “Predicting the Changeability of Object-Oriented Software with the Design Metrics”, in proceedings of the Submitted to Journal of Software and Systems,

[Kazman, 1996**]** R. Kazman, G. Abowd, L. Bass and P. Clements. Scenario-Based Analysis of Software Architecture. In IEEE Software, Vol. 13, No. 6, pages 47-55, November 1996.

[Kiran, 1997**]** G. A. Kiran, S. Haripriya and P. Jalote. Effect of Object Orientation on Maintainability of Software. In ICSM97, Bari, Italy, pages 114-121,October 1-3, 1997.

[Kung, 1994] D. C. Kung, J Gao, P. Hsia, F. Wen, Y. Toyoshima and C. Chen, “Change impact Identification in Object-Oriented Software Maintenance”, in proceedings of the Conference on Software Maintenance, 1994.

[Kung, 1995] D. C. Kung, J. Gao, P. Hsia, J. Lin and Y. Toyoshima. "Class firewall, test order, and regression testing of object-oriented programs" in Journal of Object-Oriented Programming, Vol. 8, No. 2, pages 51-65, May 1995.

[Lee, 1996] Michelle. L. Lee and A. J. Offutt. "Algorithmic Analysis of the Impact of Changes to Object-Oriented Software" in ICSM96, pages 171-184, 1996.

[Lee, 1998] Michelle L. Lee, “Change Impact Analysis of Object Oriented Software”,these de Doctorat, George Mason University, Virginia, USA,1998.

[Lindvall, 1999**]** M. Lindvall. Measurment of change: Stable and Change-Prone Constructs in a commercial C++ System in Proceedings of the 6th International Software Metrics Symposium, pages 40-49, Boca Raton, Florida, Nov1999.

[Lientz, 1978] B. Lientz, E. Swanson, et G. Tompkins. "Characteristics of application software". Communications of ACM, 21(6):466-471, Juin 1978.

[L.J. Arthur, 1998] L.J. Arthur. Software Evolution: The Software Maintenance Challenge. John Wiley & sons, 1998.

[Ljungberg, 1992] Ljungberg, M. & Lucas, A. 1992. The Oasis Air Traffic Management System. In Proceedings of the Second Pacific Rim International Conference on Artificial Intelligence, PRICAI ‘92,

[Mandiau, 2002] R. Mandiau et E. G. le Strugeon. Systèmes multi-agents. Techniques de l'Ingénieur, S7216 traité Informatique Industrielle, 2002.

[Martin et Osborne, 1985**]** Martin, Roger J. et Osborne, Wilma M., "Guidance on Software Maintenance", Computer Science and technology, U.S. Department of Commerce National Bureau of Standards, NBS Special Publication 500-106, 1985.

[McClure, 1992] Carma McClure, The Three R's of Software Automation : Re-ngineering, Repository, and Reusability, Prentice Hall, 1992.

[Munro, 1998] M. Munro, O. C. Kwon, C. Boldyreff, Survey on a Software Maintenance Support Environment. Technical Report Centre for Software Maintenance, University of Durham, 1998.

[Nii, 1989] P.Nii, «Blackboard systems», Handbook of Artificial Intelligence, Volume IV. Addison-Wesley, reading, MA, USA, 1989.

[Nii & al, 1989] P.Nii, N. Aiello and J. Rice. Expriments on Cage and Poligon: Measuring the performance of parallel blackboard systems. In L. Gasser and M. Huhns, editors, Distributed Artificial Intelligence. Volume II, pages 319-384. Pitman Publishing : London and Morgan Kaifman : San Mateo, CA, 1989.

[Oufella, 2009] Oufella Sarah « Un Système Interactif d'Aide à la Décision de Groupe en Aménagement du Territoire », diplôme magister, 2008/2009.

[Parunak, 1987] Parunak, H. V. D. Manufacturing experience with the contract net. In: M. N. Huhns (Ed.) Distributed AI. Morgan Kaufmann. 1987.

[Pavon, 2006] Juan PAVÓN «INGENIAS : Développement Dirigé par Modèles des Systèmes Multi-Agents» Université Pierre et Marie Curie le 7 décembre 2006. http://grasia.fdi.ucm.es/ingenias/France/hdr-jpavon.v16.pdf

[Pfleeger, 1990] S. L. Pfleeger and Shawn A. Bohner, A Framework for Software Maintenance Metrics. In proceedings of the Conference on Software Engineering, May 1990, Pages 320-327.

[Pigoski, 1997] T. M. Pigoski, Practical Software Maintenance : Best Practices for Managing your Software Investment. John Wiley & Sons, 1997.

[Pruitt, 1981] D. Pruitt, «Negotiation Behavior ». Academic Press, 1981.

[Rodriguez, 1994] Rodriguez M. Modélisation d'un agent autonome: Approche constructiviste de l'architecture de contrôle et de la représentation de connaissances, Thèse de doctorat, Université de Neufchâtel, 1994.

[Rombach , 1990**]** H.D. Rombach, "Design Measurement: Some Lessons Learned", in proceedings of IEEE Software, Vol. 7, No. 2, pages 17-25, 1990.

[Rumbaugh, 1991] J. Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen, Object-Oriented Modeling and Design. Prentice Hall, Englewood Cliffs, New Jersey 07632, 1991.

[Russel, 1995] Russel et Norvig. Artificial Intelligence : A modern approach , Prentice Hall series in Artificial Intelligence,1995.

[Schauer, 2001**]** Reinhard Schauer, Rudolf K. Keller, Bruno Lague, Gregory Knapen,Sebastien Robitaille, and Guy Saint-Denis. The SPOOL DesignRepository: Architecture, Schema, and Mechanisms. In Hakan Erdogmus and Oryal Tanir, editors, Advances in Software Engineering. Topics inEvolution, Comprehension, and Evaluation. Springer-Verlag, 2001.

[Smith, 1980] R.G.Smith, «The contract net protocol », IEEE Transactions, 1980.

[Swebok, 2004] Swebok, Guide to the Software Engineering Body of Knowledge. IEEE Version 2004. http://www.swebok.org/.

[Turver, 1994] R. J. Turver and M. Munro, An Early Impact Analysis Technique for Software Maintenance. Journal of Software Maintenance Research and Practice, Vol. 6, No. 1, 1994.

[Urbani, 2006] D.Urbani, «Elaboration d'une approche hybride SMA_SIG pour la définition d'un systeme d'aide a la decision, application à la gestion de l'eau», these doctorat, Université de Corse, France, 2006.

[Vercouter, 2000] Vercouter, L. Conception et mise en oeuvre de systèmes multi-agents ouverts et distribués, Thèse de doctorat, Université de Jean Monnet et Ecole des Mines de Saint-Etienne, 2000.

[Vernadet et al, 1992] F.Vernadet, P.Azena, « Prototypage de systèmes d'agents communicants », Journée Systèmes Multi- Agents PRC-GRD, Intelligence Artificielle, Nancy, Décembre 1992.

[Weiss, 2000] G. Weiss. Multiagent Systems, A Modern Approach to distributed Artificial Intelligence. The MIT Press, 2000.

[Wooldridge, 2000] Wooldridge M., Jennings N.R., Kinny D. (2000) "The Gaia Methodology for agent-oriented analysis and design" Autonomous Agents and multi-agent systems, Vol. 3.

[Wooldridge, 2001] M.Wooldridge, «An introduction to multi-agent systems», editions John Whiley & Sons, 2001.

Webliographie

1. http://excerpts.numilog.com/books/292118088x.pdf
2. http://www2.ifi.auf.org/rapports/tpe-promo12/tpe-trinh_thanh_hai.pdf
3. http://tel.archives-ouvertes.fr/docs/00/14/23/40/PDF/these_kamoun.pdf
4. http://www.emse.fr/~boissier/enseignement/maop11/courses/jade-prog-4pp.pdf
5. http://liris.cnrs.fr/alain.mille/enseignements/Master_PRO/TIA/SMA/Support_CoursIAD-SMA.pdf
6. http://s3.amazonaws.com/publicationslist.org/data/gelog/ref-315/618.pdf
7. http://opera.inrialpes.fr/people/Tayeb.Lemlouma/Papers/IAD_Presentation.pdf
8. http://www.ibm.com/developerworks/opensource/library/os-ast/

Annexe A

Table A. *Résultats de l'impact des changements (Java)*

ID du changement	Description du changement	Expression d'impact
v.1.1	Changement de valeur de variable	-
v.1.2	Changement de type de variable	S+L
v.1.2	Ajout de variable	-
v.1.4	Suppression d'une variable	S+L
v.1.5	Changement de portée de variable	
v.1.5.1	Public → Private	S
v.1.5.2	Public → Protected	S ~H*
v.1.5.3	Protected → Private	SH
v.1.5.4	Protected → Public	-
v.1.5.5	Private → Public	-
v.1.5.6	Private → Protected	-
v.1.6	Changement de variable (Static/ Non-static)	
v.1.6.1	Static → Non-static	S+L
v.1.6.2	Non-static → Sstatic	-
m.2.1	Changement de méthode (Static/Non-static)	
m.2.1.1	Static → Non-static	I+L
m.2.1.2	Non-static → Static	L
m.2.2	Changement de méthode (Abstract/Non-abstract)	
m.2.2.1	Abstract → Non-abstract	H+ie(3.1.2)+L
m.2.2.2	Non-abstract → Abstract	H+ie(3.1.1)+L
m.2.3	Changement de type de retour de méthode	
m.2.3.1	Non-abstract method	H+ie(3.1.2)+L
m.2.3.2	Abstract method	H+L
m.2.4	Changement d'implémentation de méthode	L
m.2.5	Changement de signature de méthode	
m.2.5.1	Non-abstract method	I+ie(3.1.2)+L
m.2.5.2	Abstract method	H+L
m.2.6	Changement de porté de méthode	
m.2.6.1	Public → Private	
m.2.6.1.1	Non-abstract method	I
m.2.6.1.2	Abstract method	-
m.2.6.2	Public → Protected	
m.2.6.2.1	Non-abstract method	I ~H*
m.2.6.2.2	Abstract method	-
m.2.6.3	Protected → Private	
m.2.6.3.1	Non-abstract method	H I
m.2.6.3.2	Abstract method	-
m.2.6.4	Protected → Public	
m.2.6.4.1	Non-abstract method	-

m.2.6.4.2	Abstract method	-
m.2.6.5	Private → Public	
m.2.6.5.1	Non-abstract method	-
m.2.6.5.2	Abstract method	-
m.2.6.6	Private → Protected	
m.2.6.6.1	Non-abstract method	-
m.2.6.6.2	Abstract method	-
m.2.7	Ajout de méthode	
m.2.7.1	Abstract method	ie (3.1.1)
m.2.7.2	Non-abstract method	I+ie (3.1.2) + L
m.2.8	Suppression de méthode	
m.2.8.1	Abstract method	ie (3.1.2)
m.2.8.2	Non-abstract method	I+ie (3.1.1) + L
c.3.1	Changement de classe (Abstract/ Non-abstract)	
c.3.1.1	Non-abstract → Abstract	G+ H +I+ L
c.3.1.2	Abstract → Non-abstract	H+ L
c.3.2	Suppression de classe	
c.3.2.1	Non-abstract class	S+G+H+I
c.3.2.2	Abstract class	S+ H+ I
c.3.3	Dérivation d'héritage de classe	
c.3.3.1	Public → Private	S+I
c.3.3.2	Public → Protected	(S+I) ~H*
c.3.3.3	Protected → Private	H (S+ ~SG+~SI)
c.3.3.4	Protected → Public	-
c.3.3.5	Private → Public	-
c.3.3.6	Private → Protected	-
c.3.4	Ajout de classe	-
c.3.5	Structure d'héritage de classe	
c.3.5.1	Ajout de classe abstraite	S+G+H+I+ie(3.1.1)+L
c.3.5.2	Ajout de classe non abstraite	H+L
c.3.5.3	Suppression de classe abstraite	H+ie(3.1.2)+L
c.3.5.4	Suppression de classe non abstraite	H+L

* : l'expression obtenue est diminuée du nombre de classes du même package (ces classes gardent l'accès après ce changement).

Notons qu'il y a en tout 52 Changements : 12 changements de variable, 25 changements de méthode et 15 changements de classe.

Nous signalons aussi qu'il se peut qu'un changement déclenche un autre changement (un changement déclenché). Par exemple, le changement identifié "m.2.7.1", qui est l'ajout d'une méthode abstraite dans une classe non abstraite déclenche un autre changement identifié : "c.3.1.1" puisque la classe est devenue maintenant abstraite.

Pour indiquer un changement déclenché, nous ajoutons une note à l'expression finale de la forme : ie (change id) :

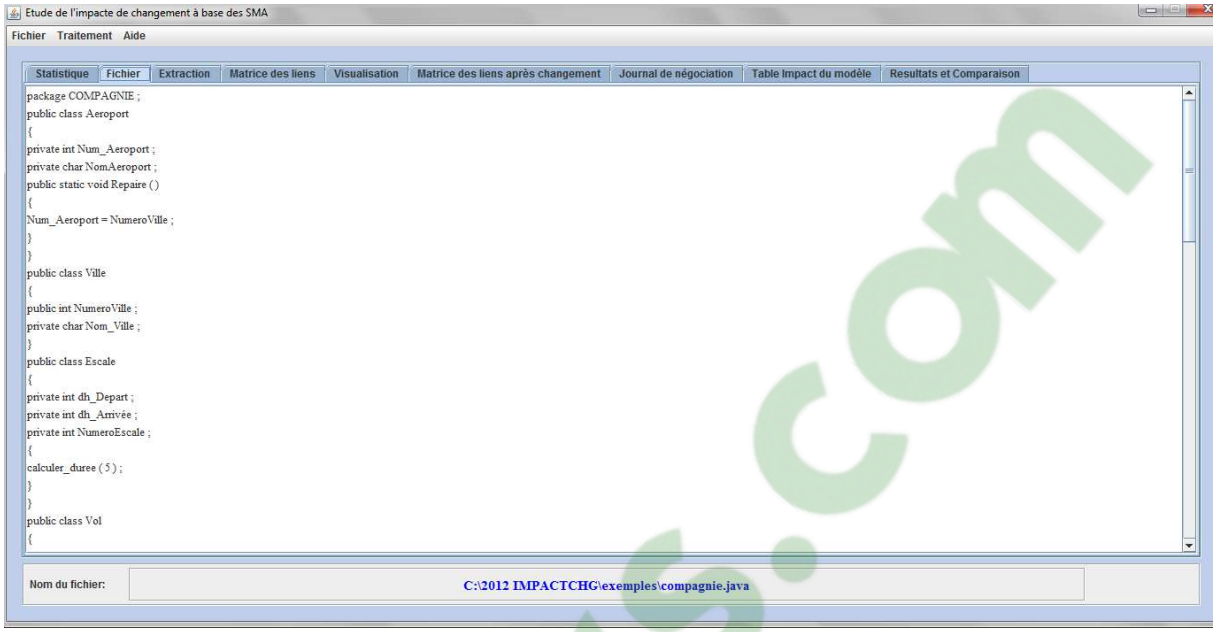
Avec : "ie" : " impact expression " : l'expression d'impact

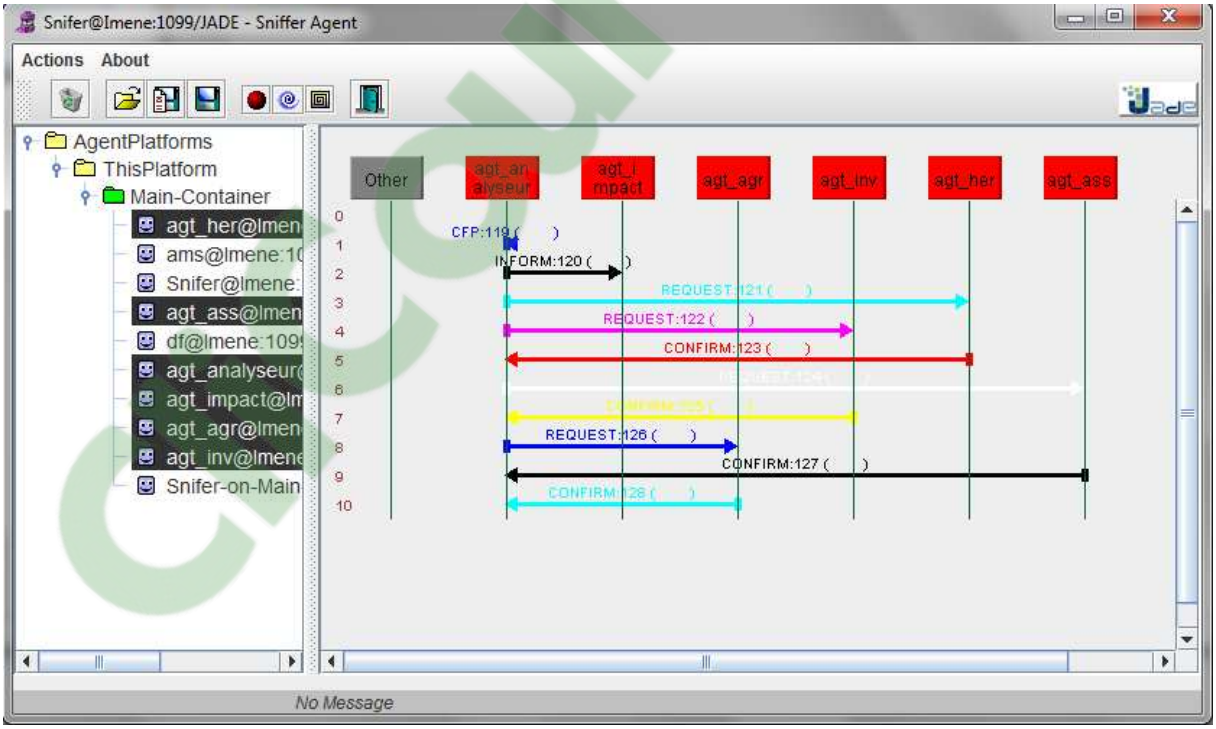
et "change id" : se réfère au changement déclenché.

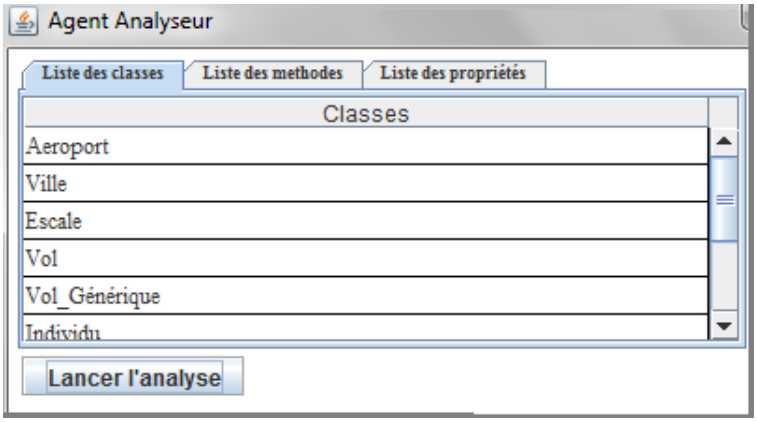
Et c’est effectivement le cas aussi des changements identifiés m.2.2.1, m.2.2.2, m.2.3.1, etc.

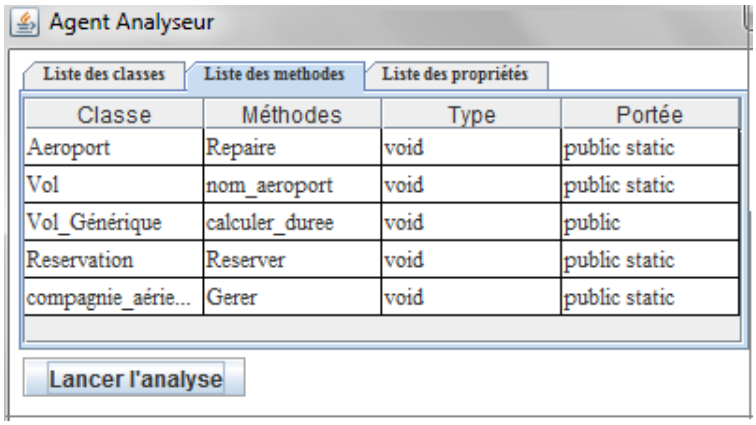
Annexe B

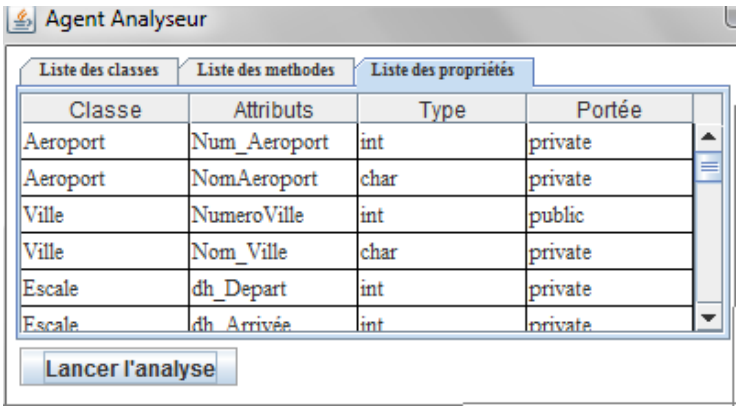
Le système de compagnie aérienne

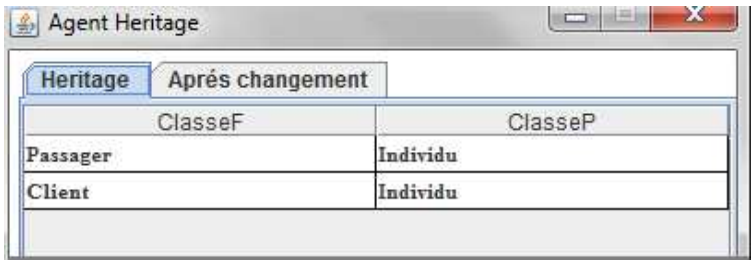





Agent Analyseur
Liste des classes
Liste des methodes
Liste des propriétés
Classes
Aeroport
Ville
Escale
Vol
Vol_Générique
Individu
Lancer l'analyse


Agent Analyseur
Liste des classes
Liste des methodes
Liste des propriétés
Classe
Méthodes
Type
Portée
Aeroport
Repaire
void
public static
Vol
nom_aeroport
void
public static
Vol_Générique
calculer_duree
void
public
Reservation
Reserver
void
public static
compagnie_aérie...
Gerer
void
public static
Lancer l'analyse


Agent Analyseur
Liste des classes
Liste des methodes
Liste des propriétés
Classe
Attributs
Type
Portée
Aeroport
Num_Aeroport
int
private
Aeroport
NomAeroport
char
private
Ville
NumeroVille
int
public
Ville
Nom_Ville
char
private
Escale
dh_Depart
int
private
Escale
dh_Arrivée
int
private
Lancer l'analyse


Agent Heritage
Heritage
Aprés changement
ClasseF
ClasseP
Passager
Individu
Client
Individu

Agent Agregation

Agrégation | Aprés changement

Classe source	Classe cible	Objet
Vol_Générique	Vol	vol

Agent Invocation

Invocation | Aprés Changement

Classe1	Classe2	Methode
Escale	Vol_Générique	calculer_duree

Agent Association

Association | Aprés changement

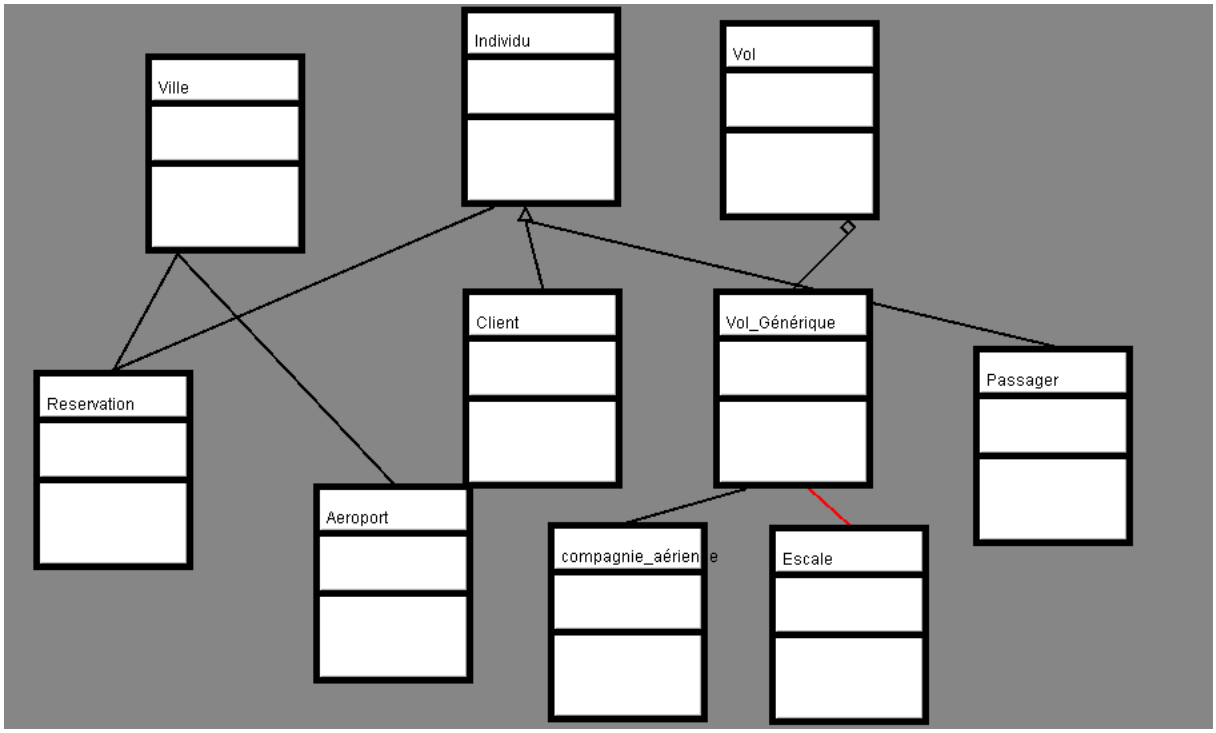
Classe source	Classe cible	Propriété
Aeroport	Ville	NumeroVille
Reservation	Ville	NumeroVille
compagnie_aérienne	Vol_Générique	date_Jour
Reservation	Individu	NumeroIndiv

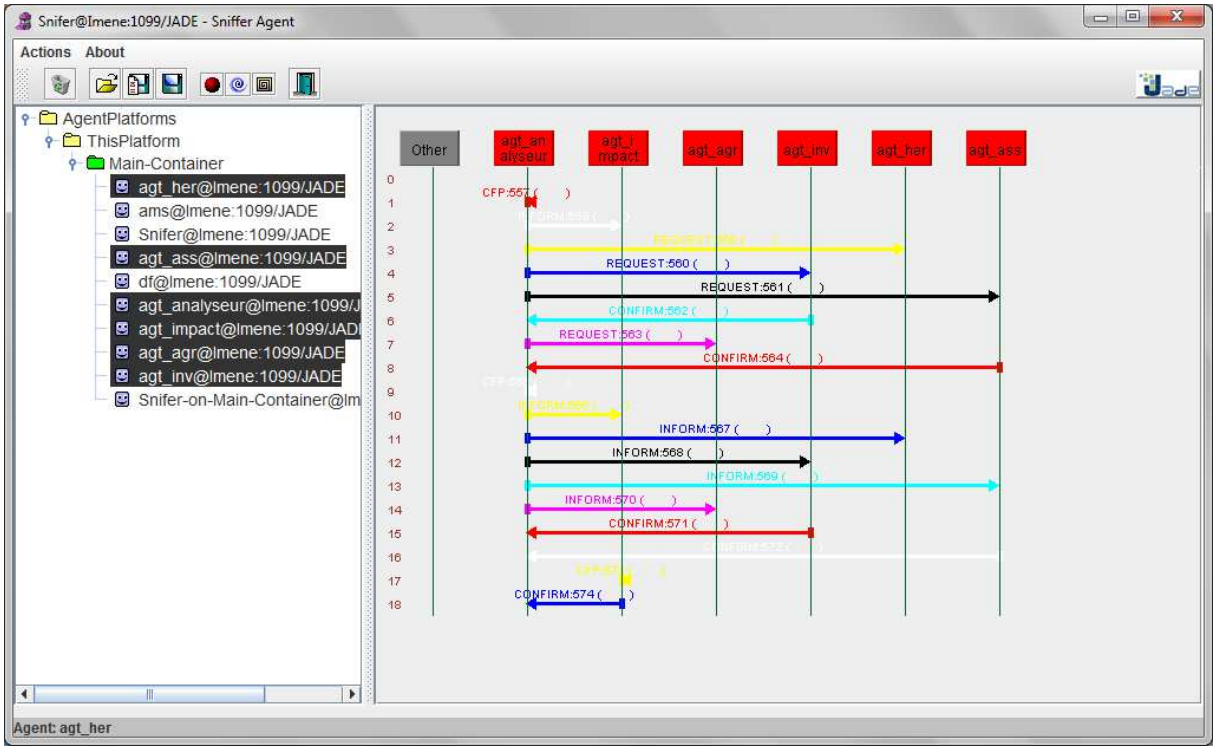
Statistique | Fichier | Extraction | Matrice des liens | Visualisation | Matrice des liens après changement | Journal de négociation | Table Impact du modèle | Resultats et Comparaison

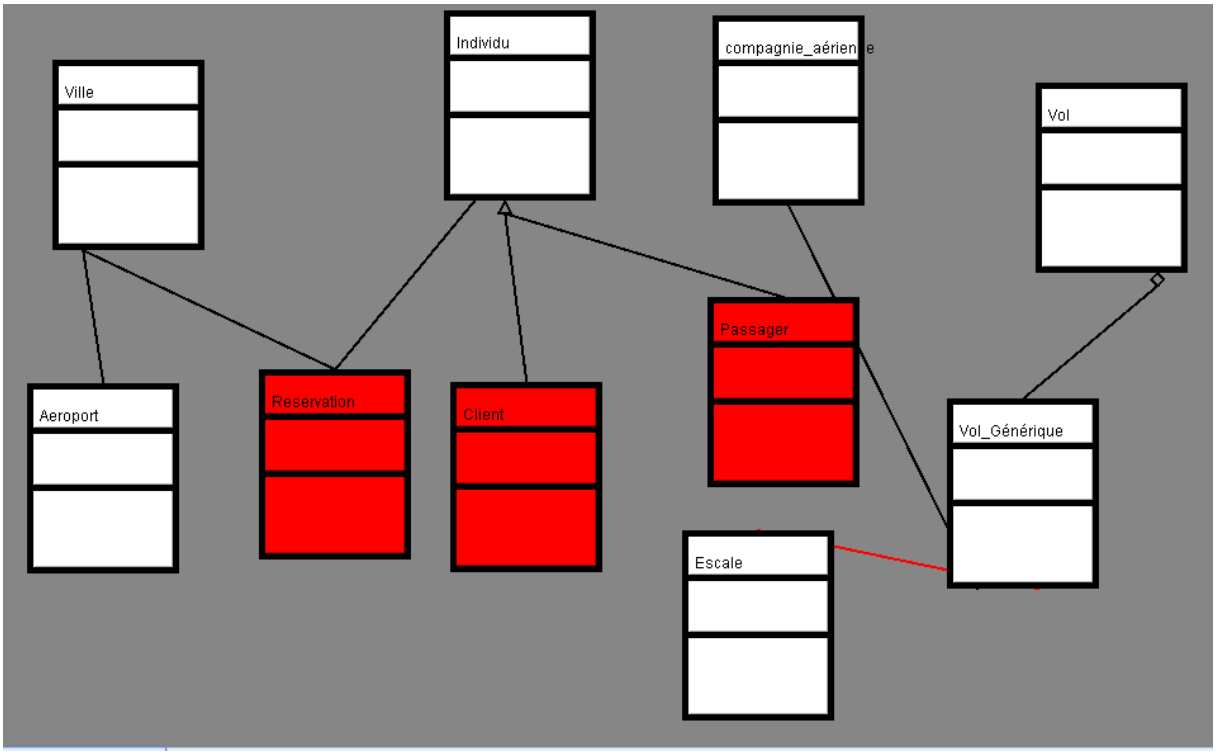
	Aeroport	Ville	Escale	Vol	Vol_Générique	Individu	Reservation	Passager	Client	compagnie_aérienne
Aeroport		S								
Ville										
Escale					I					
Vol										
Vol_Générique				G						
Individu										
Reservation		S				S				
Passager						H				
Client						H				
compagnie_aérienne					S					

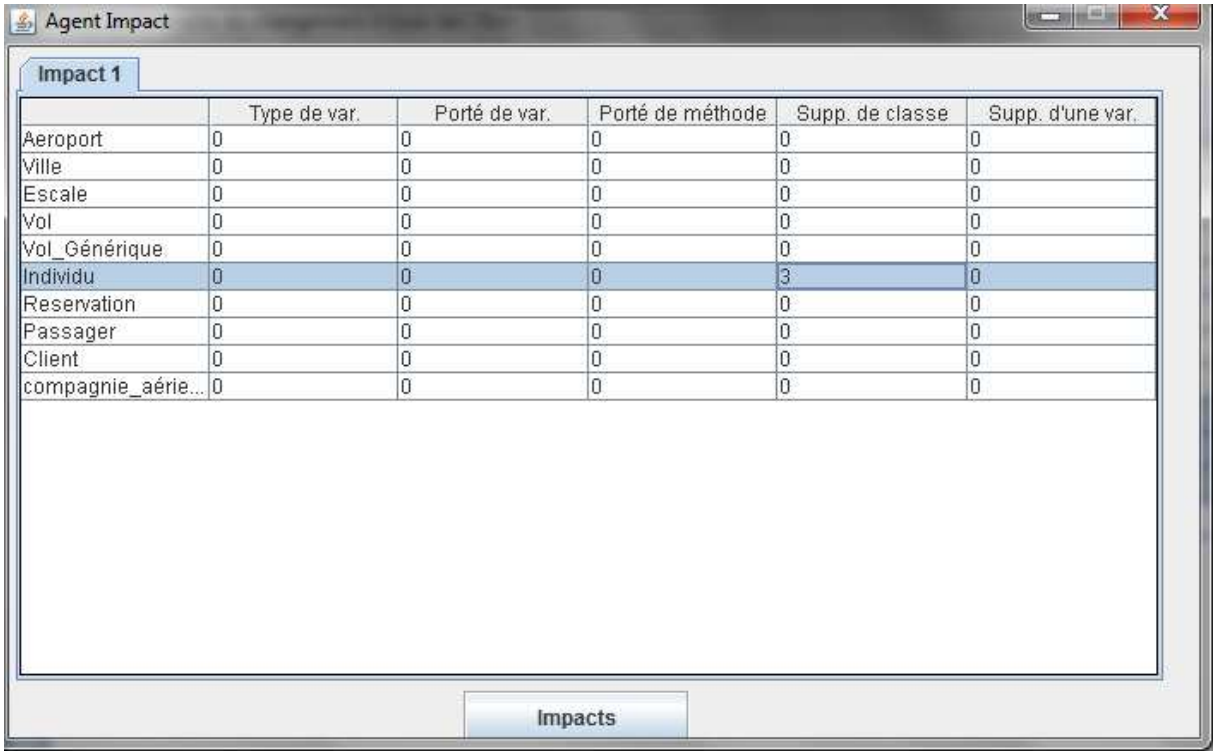
Statistique | Fichier | Extraction | Matrice des liens | Visualisation | Matrice des liens après changement | Journal de négociation | Table Impact du modèle | Resultats et Comparaison

	Type de var.	Porté de var.	Porté de méthode	Supp. de classe	Supp. d'une var.
Aeroport	0	0	0	0	0
Ville	3	2	0	2	3
Escale	0	0	0	0	0
Vol	0	0	0	1	0
Vol_Générique	2	1	1	2	2
Individu	2	1	0	3	2
Reservation	0	0	0	0	0
Passager	0	0	0	0	0
Client	0	0	0	0	0
compagnie_aérienne	0	0	0	0	0

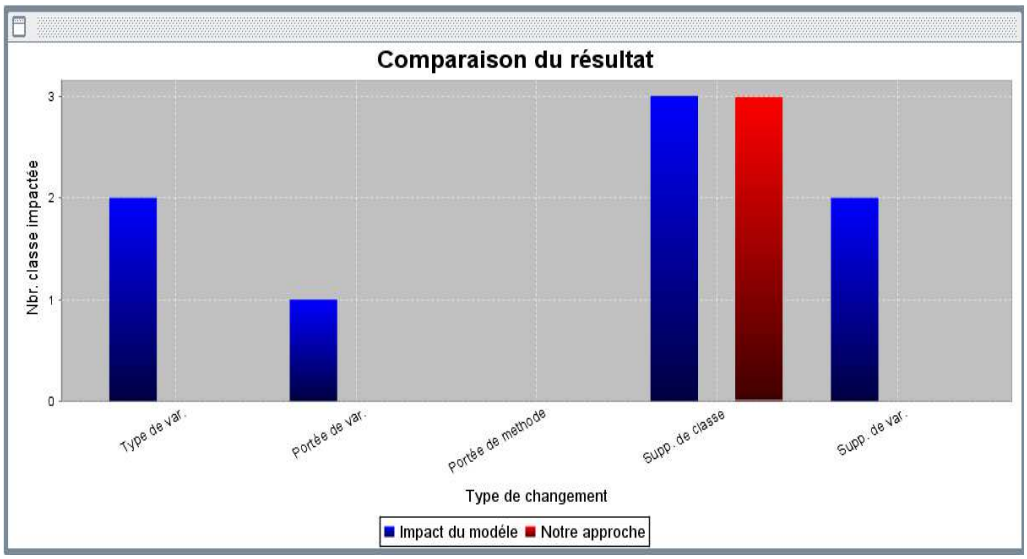

Individu
Vol
Ville
Client
Vol_Générique
Passager
Reservation
Aeroport
compagnie_aérienne
Escale


Snifer@lmene:1099/JADE - Sniffer Agent
Actions About
AgentPlatforms
ThisPlatform
Main-Container
agt_her@lmene:1099/JADE
ams@lmene:1099/JADE
Snifer@lmene:1099/JADE
agt_ass@lmene:1099/JADE
df@lmene:1099/JADE
agt_analyseur@lmene:1099/J
agt_impact@lmene:1099/JAD
agt_agr@lmene:1099/JADE
agt_inv@lmene:1099/JADE
Snifer-on-Main-Container@lm
Other
agt_an alyseur
agt_I mpact
agt_agr
agt_inv
agt_her
agt_ass
REQUEST:560 ( )
REQUEST:561 ( )
CONFIRM:562 ( )
REQUEST:563 ( )
CONFIRM:564 ( )
INFORM:567 ( )
INFORM:568 ( )
INFORM:569 ( )
INFORM:570 ( )
CONFIRM:571 ( )
CONFIRM:574 ( )
Agent: agt_her





	Type de var.	Porté de var.	Porté de méthode	Supp. de classe	Supp. d'une var.
Aeroport	0	0	0	0	0
Ville	0	0	0	0	0
Escale	0	0	0	0	0
Vol	0	0	0	0	0
Vol_Générique	0	0	0	0	0
Individu	0	0	0	3	0
Reservation	0	0	0	0	0
Passager	0	0	0	0	0
Client	0	0	0	0	0
compagnie_aérie...	0	0	0	0	0


Comparaison du résultat
Nbr. classe impactée
0
1
2
3
Type de var.
Portée de var.
Portée de methode
Supp. de classe
Supp. de var.
Type de changement
Impact du modéle
Notre approche

Thème : Etude d'impact de changement des système à objet par les systèmes multi agents SMA

Résumé :

Plusieurs travaux se sont intéressés à l'étude de la qualité de logiciel. Cependant peu d'entre eux qui ont abordé la problématique de l'impact de changement dans un système à Objets. Encore moins sont les travaux qui ont essayé de modéliser l'effet de propagation de changements dans un système à Objets. Notre contribution concerne l'étude de l'impact de changement pour les systèmes Orientés Objet à l'aide des systèmes Multi Agents en tenant compte de la complexité de l'opération de l'extraction des constituants du modèle conceptuel correspondant au code source (les différents composants du système) et la détection de l'impact de changements ainsi que sa propagation dans le reste du système. Dans notre étude, nous proposons un modèle à base d'agents, qui interagissent et communiquent entre eux de manière cohérente pour étudier l'impact de changement.

Mots clés :

Système Multi-Agent; Impact De Changement; Système A Objet; Maintenance Logiciel; Qualité De Logiciel; Agent Analyseur; Extraction; Propagation De Changement; Matrice Des Liens; Table D'impact.