RESEARCH ARTICLE OPEN ACCESS

Joco 0.1: Conteneur D'application Modulaire A Base Des Agents BDI De La Plateforme Jadex Suivant La Méthodologie O-Mase

Jamal Berrich*, Toumi Bouchentouf*, Abdelhamid Benazzi**
*(Department of Computer Science, UMP University/ENSAO, Morocco)
** (Department of Computer Science, UMP University/ESTO, Morocco)

ABSTRACT
As part of the technology and specially the aspects of structuring programming methodologies, the notion of agents and agent oriented programming AOP[3, 6] oriented become an important aspect of programming.
However, the multitude of technologies using agents and diversity of types of agents, the choice of development using agents becomes a tedious task
In this article, we present the container jocov0.1, an implementation of the *O-MaSE* [2] methodology for modeling agents and specially the BDI agents using Jadex framwork.
Keywords **–** AOP, O-MaSE, MAS, Agent, BDI, meta-model, Jadex , JOCO, JOCOLipse, JOCOCore

I. INTRODUCTION

De nos jours, les organismes ont de plus en plus recours à l'informatisation des composantes de leur système d'information du fait qu'elle permet d'offrir des services organisés, fiables et avec un accès rapide. Nous pouvons constater alors que lorsqu'il s'agit de gain de temps et d'argent, les organismes ne lésinent pas sur les moyens peu importe le secteur concerné.

Durant cette évolution, un nouveau mode de programmation a émergé, il s'agit de la POA, c'est une méthode qui exploite les théories de l'intelligence artificielle pour donner aux systèmes une certaine autonomie vis-à-vis aux humains, les applications classiques suivait un modèle séquentiel, les données dont les programmes avaient besoin y étaient stockées en dur. Afin de les dynamiser, on faisait recours aux fichiers qui avaient un certain squelette pour faciliter l'accès aux données. En suite les développeurs ont connu la POO, l'abstraction de l'application sous forme de classes qui représentent en effet des objets de l'univers réel. On spécifie ensuite les relations entre différentes classes à l'aide des diagrammes UML. Dans cette phase, on a connu l'apparition des Design-Patterns (Patrons de conceptions). Parmi ces designs patterns on pourra citer le modèle MVC où on a la séparation entre ce qui est Vue, Modèle, et Contrôleur.

L'ère des SMA est arrivée avec un nouveau concept et des nouvelles notions où le développeur ne fait que modéliser. Néanmoins il cède la main à l'agent pour faire des traitements à l'humaine mais d'une façon informatisée.

II. AGENT LOGICIEL ET ARCHITECTURE BDI

Un agent est un composant logiciel, un module informatique ou bien une entité virtuelle autonome qui est capable d'agir dans un environnement, communiquer directement avec d'autres agents, prendre des décisions, et qui a un comportement qui tend à satisfaire ses objectifs, en tenant compte des ressources et des compétences dont il dispose, et en fonction de sa perception, de ses représentations et des communications qu'il reçoit.

L'architecture BDI est conçue en partant du modèle "Belief-Desire-Intention", de la rationalité d'un agent intelligent, mis au point pour la programmation des agents intelligents. Ce modèle se base sur la mise en œuvre des croyances d'un agent rationnel, cognitif, ses désirs et ses intentions, il utilise ces concepts pour résoudre un problème particulier. En substance, il fournit un mécanisme pour séparer l'activité de sélection d'un plan (suite d'actions) de son exécution. Par conséquent, les agents BDI [1,5] sont en mesure d'équilibrer le temps passé à délibérer sur les plans (en choisissant ce qu'il faut faire) et l'exécution de ces plans (de le faire).

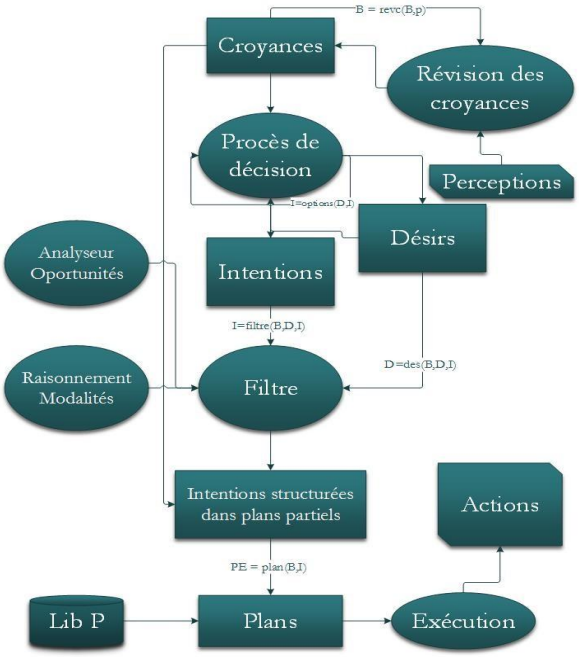

Fig.1 : Architecture BDI d'un agent

III. PRESENTATION DU PROJET JOCO

1. IDEE DERRIERE JOCO

Pour résoudre la problématique liée au développement des SMA, nous avons dans un premier lieu étudié l'architecture interne et le fonctionnement de ces systèmes afin de déterminer les difficultés posées lors de développement orienté agent. Ensuite, nous avons constaté que les agents ont besoin d'un environnement pour être exécutés ; mais ces environnements, d'une part, ne suivent aucune méthodologie de modélisation des agents constituants une organisation [10], et d'autre part, chaque environnement exige à l'agent une structure interne pour le bon fonctionnement du système, ce qui rend le développement des agents très difficile et change selon la plateforme utilisée.

Pour remédier à cela, on nous a proposé de développer une application qui modélisera, tout d'abord, les SMA suivant la méthodologie *O-MaSE* à l'aide d'un éditeur graphique facile à manipuler, et ensuite, de générer les fichiers nécessaires pour le fonctionnement de l'organisation [10] modélisée dans la plateforme Jadex.

Pour englober tout le processus de réalisation d'une application modulaire basée sur les agents, nous avons jugé nécessaires de développer deux applications, une se chargera de la création du cœur du module et l'autre créera les interfaces des modules générés et les injecter dans le cœur de l'application.

2. AVANTAGE DES APPLICATIONS MODULAIRES

Les applications les plus complexes se composent en général de plusieurs sections chacune d'elles accomplit un besoin métier bien spécifique. Maintenant que les applications deviennent de plus en plus immenses un développeur ne peut jamais réaliser une application seul, mais souvent plusieurs équipes contribuent à sa réalisation.

Un module est une application en elle-même qui effectue un traitement métier bien précis, qui sera complémentaire pour les autres parties en vue d'obtenir une application complète.

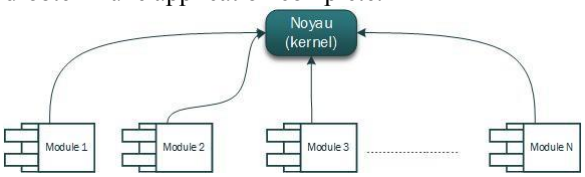

Fig.2 : Architecture d'une Application Modulaire

Les modules doivent donc être semblables et doivent respecter la même structure pour qu'ils puissent fonctionner sur le noyau. Les applications modulaires ont un autre avantage qui est plus important que la distribution des tâches lors du codage, il s'agit de la maintenance et les mises-à-jour en général, si on découvre une anomalie dans une fonctionnalité, il suffit de corriger le module d'où elle parvient, et de recompiler le module en question sans affecter la totalité des parties fonctionnelles.

IV. MODELE OBDI2JADEX

1. PRESENTATION DU FRAMWORK JADEX

Jadex [7] est un Framework qui permet de programmer des composantes actives en combinant les avantages des agents logiciels intelligents en XML et Java et ceux des composantes passives (composants SCA).

La plateforme d'exécution des composants actifs JADEX (AC), permet de faire fonctionner des composants actifs. D'une manière similaire aux environnements d'exécution des composants traditionnels (passif), tels que Java EE, l'infrastructure des composants actifs fournit un conteneur qui gère les composants. Les composants actifs diffèrent des composants traditionnels (passif) en ayant une certaine autonomie à l'égard de leur exécution. Cela signifie qu'ils peuvent décider eux-mêmes activement à exécuter certaines actions. La spécification des composants actifs ne nécessite que l'ajout de quelques propriétés c'est pour cela que des composants actifs très différents peuvent être conçues et mises en œuvre ensemble.

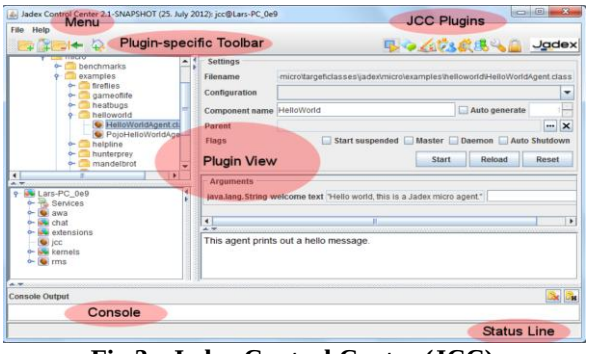

Fig.3 : Jadex Control Center (JCC)

2. AGENT BDI DANS JADEX :

Le noyau Jadex BDI est un moteur de raisonnement basé sur les croyances, les désirs et les intentions, et qu'il peut être utilisé avec différents types d'intermédiaires offrant des services basiques d'agent tels qu'une infrastructure de communication et des facilités de management.

Les concepts du concept BDI initialement proposé par Bratman étaient adaptés pour un modèle plus formel très convoité pour les systèmes multi-agents dans l'architecture des logiciels.

Jadex s'appuie sur l'expérience acquise depuis les principaux systèmes BDI existants et par conséquent améliore les faiblesses BDI précédentes comme la manipulation simultanée des objectifs incompatibles avec délibération des objectifs.

Jadex facilite l'utilisation du modèle BDI dans le contexte de la programmation générale, en introduisant les croyances, les buts et les plans comme des objets de première classe. Ceci peut être créé et manipulé à l'intérieur de l'agent.

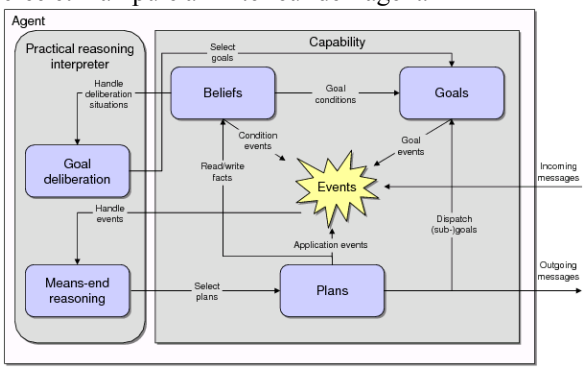

Fig.4 : Modèle BDI dans Jadex

Le raisonnement dans Jadex est un processus de deux intervalles de composants. D'un côté, l'agent réagit aux messages reçus, aux évènements internes et les buts en sélectionnant et exécutant des plans. D'autre part, un agent délivre continuellement à-propos de ses buts, pour décider de ses sous-ensembles consistent qui doivent être poursuis.

Les concepts généraux de Jadex sont les croyances, buts et les plans. Les buts et les plans de l'agent sont définis par le programmeur et décrivent son comportement. Les croyances courantes influent sur la délibération de l'agent, et les plans peuvent changer les croyances courantes pendant leurs temps d'exécution.

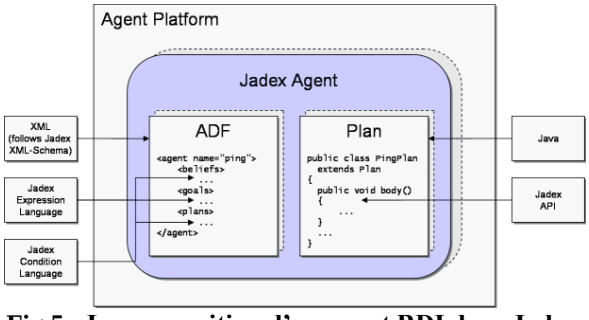

Fig.5 : La composition d'un agent BDI dans Jadex

3. PRESENTATION DU MODELE OBDI2JADEX

L'architecture d'oBDI2Jadex [9] se résume à la création d'un plugin eclipse (nommé par la suite JOCOLipse) qui offre un éditeur graphique, avec une palette contenant les différents éléments de modélisation des SMA suivant *O-MaSE* et une zone de dessin pour représenter les modules. Et un générateur de code d'agents pour le conteneur Jadex.

L'objectif majeur du modèle oBDI2Jadex est d'offrir un outil graphique permettant la modélisation de l'environnement d'exécution, d'interaction et d'échange d'informations des agents BDI d'un cote et de l'autre offrir la possibilité de générer le code relativement au framwork Jadex.

L'avantage majeur du modèle c'est que la maintenance applicatif sera relatif au modèle crée au paravent et non au composant Jadex.

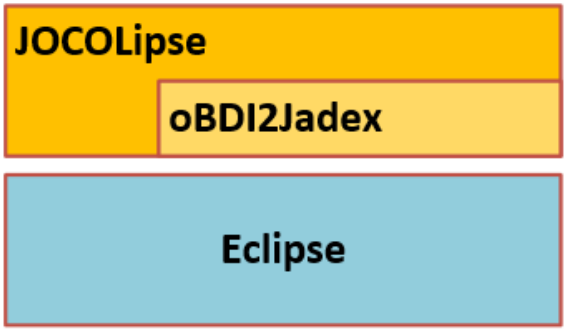

Fig.6 : La couche oBDI2Jadex

Pour faire la liaison entre la méthodologie *O-MaSE* et la structure des agents Jadex, une première étude a été effectué en se basant sur le diagramme d'Agent d'*O-MaSE* et en essayant d'injecter les propriétés relatif à Jadex pour produire un nouveau modèle de diagramme nommé BDIDiagram.

L'instanciation du modèle BDIDiagram produit une structure valide par-rapport à la méthodologie *O-MaSE* et contient toutes les informations nécessaires pour exécuter un agent dans Jadex.

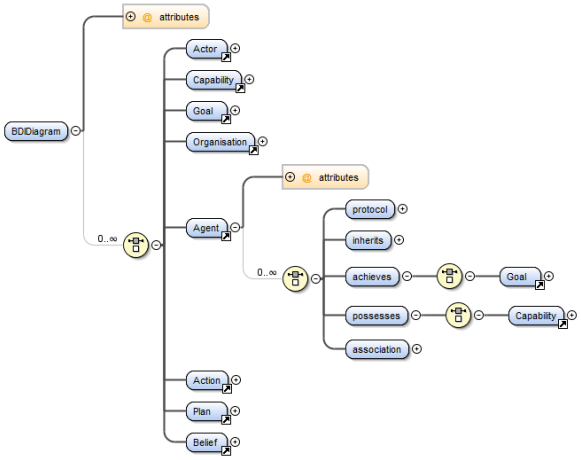

Fig.7 : L'agent BDI suivant le modèle oBDI2Jadex

Ce modèle était la base pour modéliser et créer le plugin JOCOLipse.

V. CONCEPTION ET MODELISATION DE JOCO

1. ARCHITECTURE GENERALE

L'architecture de JOCO se divise en deux parties essentielles, le plugin JOCOLipse qui offre un éditeur graphique, avec une palette contenant les différents éléments de modélisation des SMA suivant *O-MaSE* et une zone de dessin pour représenter le JOCOModule, l'export de ce module résulte la génération de trois plugins fonctionnels dans Eclipse Scout ainsi que les fichiers descriptifs des agents. Et JOCOCore, le cœur de l'application JOCO, c'est une application Client/ Serveur qui se chargera, au niveau du client, d'injecter les interfaces de chaque JOCOModule, et au niveau du Serveur, JOCOCore lance l'application et la plateforme Jadex et il fournit des services permettant la communication entre les différents agents.

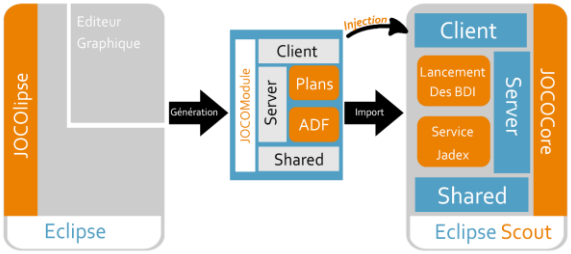

Fig.8 : Architecture de JOCO

2. PROCESSUS D'UTILISATION

Le processus d'utilisation du JOCO est le suivant :

Tache	Description
1	Créer un nouveau projet JOCO dans JOCOlipse, en définissant le nom du projet et du model.
2	Glisse les différents éléments de la méthodologie *O-MaSE* à partir d'une palette vers un espace de dessin afin de modéliser le SMA.
3	Un agent BDI est un agent qui a un plan et un fichier descriptif (ADF), pour cela il est nécessaire de générer ces deux fichiers pour chaque agent modéliser lors de la deuxième tâche, les fichiers générer doivent être adéquates avec la plateforme Jadex.
4	L'export du module modélisé dans JOCOlipse, sous forme de trois plugins fonctionnels dans Eclipse Scout.
5	L'import du projet généré dans JOCOCore qui se base sur l'environnement Eclipse Scout.
6	La création des interfaces et l'injection du module dans JOCOCore.
7	L'export de l'application finale sous trois formats possibles (Desktop, mobile et web).

Tab.1 : Processus d'utilisation de JOCO

On peut expliquer le processus de fonctionnement du système JOCOLipse – JOCOCore par le schéma suivant :

Fig.9 : Processus Métier de JOCOLipse

VI. MISE EN PLACE DU PLUGIN JOCOLIPSE

L'Objectif principale du plug-in JOCOLipse est de modéliser et développer des modules qui seront intégrés dans le noyau JOCOCore d'une application modulaire selon la spécification *O-Mase*. C'est un outil qui rend la tâche de la création d'un nouveau module assez simple car il couvre les différentes étapes de création et génère automatiquement la structure nécessaire du module.
Le plugin permettra au développeur de créer un projet « joco » contenant une palette graphique où il pourra glisser et interconnecter les différents éléments graphiques tout en respectant *O-MaSE*.

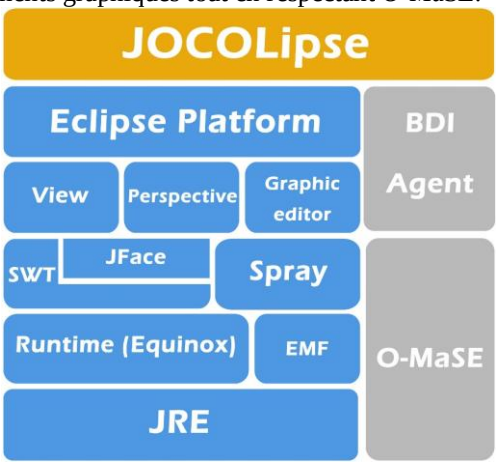

Fig.10 : Structure de JOCOLipse

1. MODELISATION DU PLUGIN

La modélisation du plugin peut être divisée en deux parties majeures, à savoir :

a. Modélisation du méta-model de validation des graphes suivant *O-MaSE*.
b. Modélisation de l'architecture du plugin.

La première étape consiste à créer un projet Encore et d'y modéliser le schéma représentatif du méta-model *O-MaSE* avec ses différents composants graphiques. Dans ce méta model, nous avons pris comme point de départ l'élément organisation d'*O-MaSE* qui joue le même rôle du package dans Jadex. Cette organisation(ou package) est composée de plusieurs agents. Chaque agent joue un rôle. Ainsi un rôle pourra achever des buts et initier des protocoles. L'architecture des agents dans Jadex repose sur les buts, et chaque rôle peut achever un ou plusieurs buts. Dans le cas de buts multiples le choix sera alors conditionné pas des délibérations et des conditions.

Comme dans *Jadex* et *O-MaSE*, un agent possède une capacité qui rassemble un ensemble de plans exécutés à chaque fois que leurs déclencheurs soient captés dans l'environnement. Il vient finalement l'élément *Parameter*, spécifique à *Jadex*,

qui permet d'ajouter d'autres informations et propriétés relatives à un plan, un but ou un protocole. D'autre part, nous avons remarqué que des éléments important dans *Jadex* sont absents dans le métamodel *O-MaSE* tel que les croyances, les faits et les paramètres. Ces éléments sont propres à l'agent et diffèrent d'un agent à un autre. Nous avons donc utilisé l'élément *DomainModel* d'*O-MaSE* pour les classer.

Pour que les associations puissent être traitées comme des éléments dans les étapes suivantes, nous avons été amenés à ajouter de nouveaux éléments représentants les différentes associations entre les classes de notre méta model héritant de la classe *AssociationModel*.

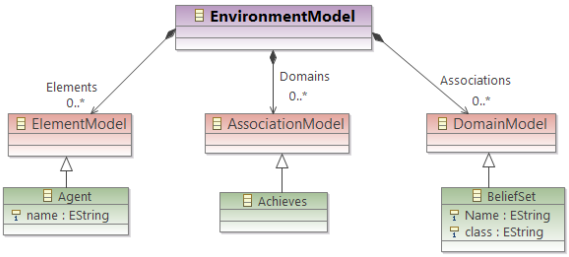

Fig.11 : Relation entre les composants du méta modèle Ecore

La deuxième étape consiste à la création de la structure du module JOCO. Le projet exporté sera constitué de trois plugins suivant la spécification des projets Eclipse Scout, à savoir :

Client	**Plugin de la partie client : Contient les interfaces de l'application.**
Server	Plugin de la partie server : contient les classes java les ADFs, et les différents traitements.
Shared	Client pour la partie shared : cette partie gère les liaisons et les communications entre les deux parties précédentes.

2. REALISATION DU PLUGIN

La structure du module généré est la suivante :

Fig.12 : Architecture de JOCOModule

La figure 13 et 14 représente respectivement, la perspective du plugin eclipse JOCOLipse qui inclut la zone palette des composant O-MaSE pour Jadex, la zone de modélisation et la partie de définition des propriétés de chaque composant du modèle, et la modélisation de l'exemple de traduction en utilisant un agent traducteur.

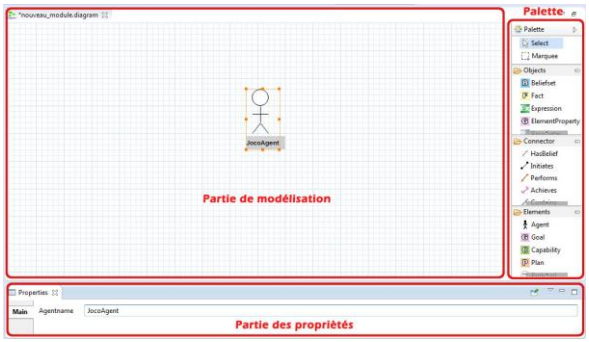

Fig.13 : Perspective JOCOLipse

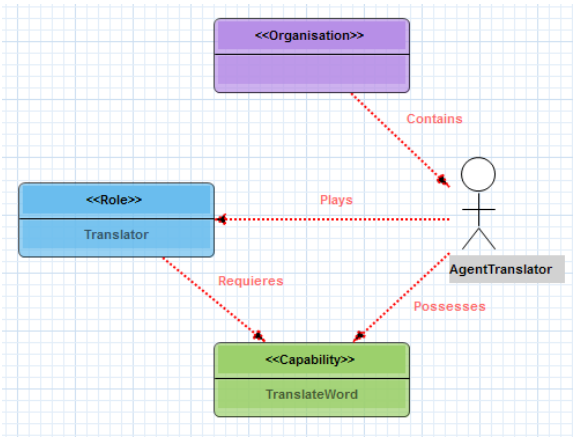

Fig.14 : Modélisation du module de traduction

VII. MISE EN PLACE DU NOYAU JOCOCORE

La réalisation d'une application modulaire exige la mise en place d'un conteneur des modules générés afin d'assurer leur intégration et leur communication.

L'étape précédente consistait à la réalisation d'un plugin JOCOLipse qui automatise et facilite le processus de création d'un JOCO module. Ce dernier est généré à partir d'une description XML Des interconnexions entre les éléments O-MaSE représentés dans la palette Graphique.

Le résultat du noyau était un conteneur des modules générés par la distribution JOCOLipse et qui assurait le lancement de la plateforme Jadex.

1. MODELISATION DU NOYAU

Le noyau JOCOCore est le cœur de notre application, il est basé sur la distribution Eclipse Scout qui a pour avantage le regroupement des deux parties client et serveurs dans une même instance du projet. D'autre part Eclipse Scout sépare les différentes parties du projet et gère leur déploiement sous forme de trois plugins à savoir : Server, Client et shared.

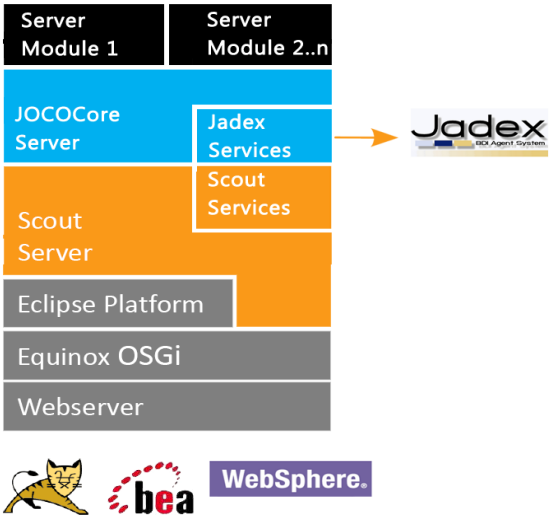

Fig.15 : Architecture du Serveur de JOCOCore

La partie serveur du noyau se base dans son fonctionnement sur les services. En effet tous les traitements réalisés doivent êtres encapsulés dans un service qui gère leur démarrage et leur arrêt. De plus le déploiement de l'application peut être fait sous trois formes : Desktop, Mobile ou web.
Les modules JOCO étant composés d'un plugin serveur, ce dernier utilise les services offerts par le serveur du JOCOCore.

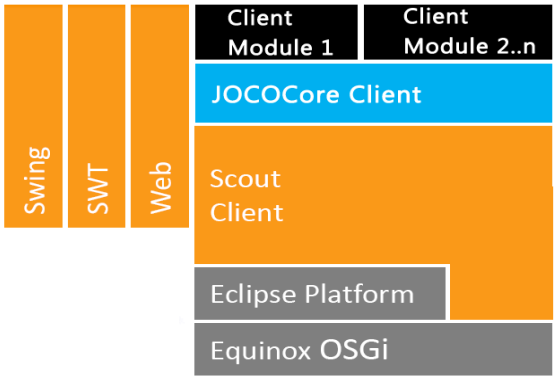

Fig.16 : Architecture du Client de JOCOCore

La partie client du noyau est composée essentiellement des interfaces qui assurent la communication entre l'utilisateur et le traitement associé du coté serveur.
Les modules JOCO étant composés d'un plugin client, ce dernier sera injecté dans la partir client de JOCOCore.

2. REALISATION DU NOYAU

La première phase de la réalisation du noyau JOCOCore consiste à créer un nouveau projet Scout, et de le personnaliser par la suite en créant une interface Desktop principale, au niveau du client. Celle-ci hérite de la classe abstraite AbstractDesktop et implémente la classe IDesktop qui fournit un ensemble de méthode gérant la configuration de l'interface. Le développeur possède par la suite le droit de personnaliser cette interface en ajoutant un système d'onglets, des formes, des menus ou même des wizards.

D'autre part, la partie serveur du noyau JOCO implémente un ensemble de services facilitant la tâche du codage au développeur. Citons à titre d'exemple les services de configuration des différentes bases de données existantes, les services de remplissage des tableaux et des formes, les web services. Le développeur a aussi la possibilité de créer son propre service et de l'adapter à ses besoins. Dans notre cas, nous avons opté pour cette dernière possibilité à savoir : la création d'un service personnalisé. En effet, nous avons procédé à la création de plusieurs services :

Service de lancement de la plate-forme Jadex : Gère le lancement et l'arrêt de l'environnement d'exécution des agents.
Service BDIStarter : Gère le lancement des agents dans la plateforme Jadex.
Service d'envoi des requêtes: Gère l'envoi de la requête depuis l'interface à l'agent responsable de son traitement.
Service de l'ajout d'un Listner à la réponse : Attend la réception du résultat du traitement de la requête et met à jour l'interface client.

La deuxième phase a pour but d'exploiter le module généré par JOCOLipse.

Etape	Description
1	L'ajout des modules dans JOCOCore par l'import des trois plugins généré par JOCOlipse dans Scout.
2	Définition de l'interface IHM (formes, menus, onglets,…) du module JOCO.
3	Injection de l'interface créée dans la vue principale.
4	Utilisation des différents services fournis par JOCOCore à fin d'interagir avec la plateforme Jadex gérant ainsi le cycle de vie des agents.

Tab.2 : Etapes d'exécution d'un module dans JOCO

La figure suivante représente l'interface graphique relative à la modélisation de l'exemple de traduction relatif à la figure Fig.14

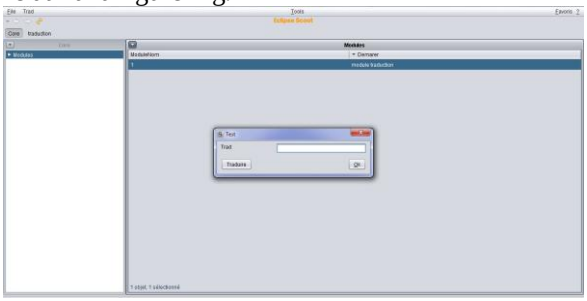

Fig.17 : Vue principale - Module de traduction intégré dans JOCOCore

VIII. CONCLUSION

Nous y avons présenté les différentes phases pour la réalisation de JOCOLipse qui est un plugin de modélisation et de réalisation des SMA, en vue de les intégrer dans JOCOCore qui est un noyau assurant le l'interconnexion de ses derniers.

Afin de faciliter la tâche à toute personne désirant développer des applications orientées agents, destinées à la plateforme Jadex et tout en suivant la méthodologie O-MaSE, le projet JOCO a été donc réalisé.

IX. ACKNOWLEDGEMENTS

Un grand remerciement à Scott Deloach pour son grand lors de l'élaboration de la méthodologie O-MaSE et pour son accompagnement durant la phase de modélisation.

REFERENCES

[1] Anand S. Rao and Michael P. George,BDI Agents : From Theory to Practice, April 1995.

[2] Scott A. DeLoach and Juan Carlos García-Ojeda, O-MaSE: a customisable approach to designing and building complex, adaptive multi-agent systems, Int. J. Agent-Oriented Software Engineering, Vol. 4, No. 3, 2010.

[3] Ingrid Nunes, Carlos J.P. de Lucena, Uira Kulesza, and Camila Nunes, On the Development of Multi-agent Systems Product Lines: A Domain Engineering Process, AOSE 2009

[4] Ingrid Nunes, Simone Barbosa, Michael Luck, and Carlos Lucena, Dynamically Adapting BDI Agent Architectures based, AOSE 2011

[5] Busetta, P., Howden, N., R¨onnquist, R., Hodgson, A.: Structuring BDI agents in functional clusters. In: ATAL '99. pp. 277–289 (2000)

[6] Kiczales, G., Lamping, J., Menhdhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J.: Aspect-Oriented Programming. In: ECOOP 1997. vol. 1241, pp. 220–242. Springer-Verlag,

[7] Pokahr, A., Braubach, L.: Jadex user guide. Tech. Rep. 0.96, University of Hamburg, Hamburg, Alemanha (2007)

[8] Scott A. DeLoach and Mark Wood, Developing Multiagent Systems with agentTool, Intelligent Agents VII. Agent Theories, Architectures, and Languages - 7th.International Workshop, ATAL-2000, Boston, MA, USA, July 7-9, 2000, Proceedings,Lecture Notes in Artificial Intelligence. Springer-Verlag, Berlin, 2001.

[9] J. BERRICH, T.BOUCHENTOUF, 'oBDI2Jadex: An agent model based on O-MaSE methodology to design a BDI agents for Jadex' International Journal of Engineering and Advanced Technology, Volume-2, Issue-6

[10] J.BERRICH, T.BOUCHENTOUF, 'Integration of a Multi-Agent Systems Based on Organization to Validate the Exercises Realized by Learner in an e-Learning Platform' International Journal of Computer Theory and Engineering Vol. 4, No. 2, April 2012