Université Ibn Zohr
Faculté Polydisciplinaire de Ouarzazate
Filière Master : Intelligence Artificielle et Applications

Module : IA et Systèmes multi-Agents

Application Multi-Agents de Livraison de Colis électronique

Basée sur JADE

Projet tutoré présenté par :

HANAN BASSOU
MARYAME KHOUYA

Sous la direction de :

Prof. IMANE DBIBIH

Année universitaire : 2024-2025

Table des matières

Remerciement

Tout d'abord, nous remercions Dieu tout-puissant pour les privilèges qu'il nous a accordés, l'opportunité d'étudier, et pour les capacités, compétences et patience qu'il nous a données pour réaliser ce travail. Nous remercions notre superviseur, la Docteure IMANE DBIBIH pour son soutien, son aide et ses conseils. Nous n'oublions pas non plus de remercier et d'apprécier nos chers enseignants à toutes les étapes de notre vie, sans lesquels nous ne serions pas arrivés là où nous en sommes maintenant.

Résumé

Ce travail présente une application développée avec **JADE**, simulant un système logistique de livraison de colis électronique fondé sur une architecture **multi-agents**. L'objectif est d'automatiser l'ensemble du processus — de la demande à la livraison — à travers l'interaction de trois agents autonomes : **AgentClient** initie une demande de livraison ; **AgentTri** reçoit les demandes, sélectionne un livreur et transmet la mission ; **AgentLivreur** prend en charge la livraison du colis et notifie le client une fois la tâche accomplie.

La modélisation UML, comprenant les diagrammes de cas d'utilisation, de classes, d'interactions et d'activités, a facilité la structuration du système en clarifiant les rôles des agents et les flux de communication. L'architecture déployée, répartie en conteneurs distincts, offre modularité, évolutivité et autonomie au système.

Les outils Eclipse, JADE, RMA et Sniffer ont contribué à un développement stable et à un suivi rigoureux des interactions entre agents. Ce prototype constitue une base solide pour de futures extensions, telles que l'intégration du suivi en temps réel des livraisons, la gestion sécurisée des paiements, ou encore l'optimisation intelligente des trajets, dans le but de moderniser les services logistiques.

Mots clés : *JADE, agents intelligents, système multi-agents, livraison de colis électronique, interaction agent, UML, conteneur JADE, automatisation, architecture distribuée, RMA, Sniffer, communication ACL, suivi en temps réel, optimisation logistique.*

Abstract

This work presents an application developed using **JADE**, simulating a logistics system for electronic parcel delivery based on a **multi-agent** architecture. The objective is to automate the entire delivery process — from request to delivery — through the interaction of three autonomous agents : **AgentClient** initiates a delivery request ;
AgentTri receives requests, selects a delivery agent, and forwards the task ;
AgentLivreur handles the parcel delivery and notifies the client upon completion.

UML modeling, including use case, class, interaction, and activity diagrams, facilitated system structuring by clarifying agent roles and communication flows. The deployed architecture, distributed across distinct containers, provides modularity, scalability, and autonomy to the system.

The tools Eclipse, JADE, RMA, and Sniffer contributed to a stable development process and effective monitoring of agent interactions. This prototype provides a solid foundation for future enhancements, such as real-time delivery tracking, secure payment handling, or intelligent route optimization, aimed at modernizing delivery services.

Keywords : *JADE, intelligent agents, multi-agent systems, electronic parcel delivery, agent interaction, UML modeling, JADE containers, automation, distributed architecture, RMA, Sniffer, ACL communication, real-time tracking, logistics optimization.*

Table des abréviations

Abréviation	**Définition**
JADE	Java Agent DEvelopment Framework
IDE	Integrated Development Environment (Environnement de développement intégré)
ACL	Agent Communication Language (Langage de communication entre agents)
FIPA	Foundation for Intelligent Physical Agents
RMA	Remote Monitoring Agent (Agent de gestion à distance de JADE)
DF	Directory Facilitator (Service de Pages Jaunes : registre des services des agents)
AMS	Agent Management System (Service de Pages Blanches : registre des agents)
UML	Unified Modeling Language (Langage de modélisation unifié)
GUI	Graphical User Interface (Interface graphique utilisateur)
Java	Langage de programmation orienté objet multiplateforme
Eclipse	Environnement de développement intégré (IDE) extensible et polyvalent

Introduction

1. Contexte général

Avec le développement rapide du commerce en ligne et la demande croissante de rapidité, les applications de livraison de colis sont devenues des outils indispensables du quotidien. Elles permettent aux particuliers comme aux entreprises de commander, suivre et recevoir leurs colis de manière efficace.

Une application de livraison de colis est un outil numérique qui permet aux utilisateurs de commander et de suivre des livraisons de biens ou de marchandises. Elle peut être utilisée par des particuliers, des entreprises ou des services de livraison tiers.

De plus en plus innovantes, les applications de livraison de colis sont nombreuses et permettent de proposer une large gamme de services de livraison à prix accessibles et rapides. Ces applications sont conçues pour faciliter la vie quotidienne des utilisateurs.

Ce système permet aussi de résoudre plusieurs problèmes logistiques, comme la centralisation des demandes, la distribution intelligente des missions, une communication décentralisée entre entités autonomes et une réduction de l'intervention humaine.Grâce à l'intégration de technologies innovantes, elles améliorent les performances logistiques et optimisent les délais de livraison.

2. Types d'applications de livraison de colis

Applications de livraison de repas
Elles permettent de commander des repas auprès de restaurants et de les faire livrer à domicile, comme Deliveroo ou Uber Eats.

Applications de livraison de courses
Elles permettent aux utilisateurs de faire leurs courses en ligne et de les faire livrer à domicile, parfois avec des options de livraison le jour même.

Applications de livraison de colis entre particuliers
Ces applications mettent en relation des particuliers souhaitant envoyer ou recevoir des colis, souvent via des services de covoiturage ou de transport collaboratif.

Applications de livraison pour entreprises

Elles sont utilisées par les entreprises pour gérer leurs propres livraisons, suivre les envois, gérer les itinéraires et communiquer avec les livreurs.

Applications de suivi de colis

Elles permettent de suivre l'état d'un colis à partir de son numéro de suivi, quel que soit le transporteur.

Applications de messagerie

Certaines plateformes de messagerie intègrent des services de livraison, permettant aux utilisateurs d'envoyer des colis à d'autres utilisateurs ou à des adresses spécifiques.

3. Objectifs

Dans ce projet, nous avons développé une application destinée aux entreprises pour la gestion des livraisons de colis électronique standards, reposant sur une architecture multi-agents basée sur le framework JADE.

Chapitre 1

Les systèmes multi-agents

1.1 Introduction

Les systèmes multi-agents (SMA) permettent de modéliser des comportements intelligents distribués en simulant l'interaction entre agents autonomes.

1.2 Concept d'agent

Plusieurs définitions d'agent ont été proposées. Selon Ferber (1995) : *" Un agent est une entité autonome, réelle ou abstraite, qui est capable d'agir sur elle-même et sur son environnement ; qui, dans un univers multi-agent, peut communiquer avec d'autres agents, et dont le comportement est la conséquence de ses observations, de ses connaissances et des interactions avec les autres agents."*[1]

Les principales propriétés d'un agent sont :

- **Situation** : Capacité à recevoir des entrées sensorielles et à agir sur l'environnement.
- **Autonomie** : Capacité à prendre des décisions sans intervention humaine directe.
- **Réactivité** : Capacité à réagir en temps réel à l'environnement.
- **Proactivité** : Capacité à poursuivre des objectifs de manière autonome.
- **Sociabilité** : Capacité à interagir avec d'autres agents ou humains.
- **Mobilité** : Capacité à se déplacer dans un réseau.
- **Apprentissage** : Capacité à modifier son comportement selon les expériences passées.

1.3 Systèmes multi-agents

Un système multi-agent est composé d'un ensemble d'agents interagissant entre eux selon certaines règles. Il présente généralement les caractéristiques suivantes :

- Les agents sont autonomes et peuvent travailler indépendamment.
- Chaque agent joue un rôle dans le système global.
- Les agents coopèrent pour accomplir des objectifs.
- La coordination, la communication et la coopération sont essentielles.
- Les agents partagent parfois un objectif commun.
- Chaque agent a une vue partielle du système.

1.4 Communication dans un système multi-agent

La communication entre agents peut être :

— **Directe (par message)** : Un agent envoie un message explicite à un ou plusieurs autres agents.

— **Indirecte (par environnement partagé ou boîte aux lettres)** : Les agents déposent ou récupèrent des informations dans un espace commun.

1.5 Conclusion

Les systèmes multi-agents permettent de simuler et de résoudre des problèmes complexes de manière distribuée. Ils offrent flexibilité, autonomie et réactivité, et sont parfaitement adaptés à des domaines comme la logistique, la robotique ou les systèmes intelligents.

Chapitre 2

La plateforme JADE

2.1 Introduction

Ce chapitre présente la plateforme JADE, décrivant ses composants, son architecture, ses fonctionnalités et ses outils intégrés pour la conception et la surveillance de systèmes multi-agents.

2.2 Présentation de la plateforme JADE

a. Définition

JADE (Java Agent DEvelopment Framework) est un intergiciel open source permettant le développement d'applications multi-agents pair-à-pair. Il est conforme aux spécifications FIPA et fournit une API Java pour créer, déployer et gérer des agents intelligents sur différentes plateformes (postes fixes, mobiles, etc.)[2].

FIGURE 2.1 – Framework JADE

b. Caractéristiques principales

- Conformité aux spécifications FIPA (Directory Facilitator, AMS, transport de messages, etc.).
- Fourniture des services internes comme **AMS** (Agent Management System) et **DF** (Directory Facilitator).
- Transport et traitement des messages ACL.
- Extensibilité des protocoles d'interaction via des méthodes `handle`.
- Système de gestion d'événements au sein du noyau de la plateforme.
- Observabilité de la plateforme (messages, agents, communication).
- Outils intégrés comme **RMA**, **Sniffer** et **Introspector**.
- Ontologie de management (`fipa-management-ontology`) et services utilitaires comme **DummyAgent Tool**.

c. Interface RMA (Remote Management Agent)

JADE propose une interface graphique pour l'administration de la plateforme via son agent RMA (Remote Management Agent). Cet agent permet d'afficher l'état de la plateforme d'agents à laquelle il appartient (y compris les agents et les conteneurs d'agents), et met à disposition divers outils pour :

— envoyer des requêtes d'administration à l'agent AMS (Agent Management System),

— déboguer et tester les applications développées avec JADE [3].

La figure 4.3 montre l'interface graphique (GUI) de l'agent RMA.

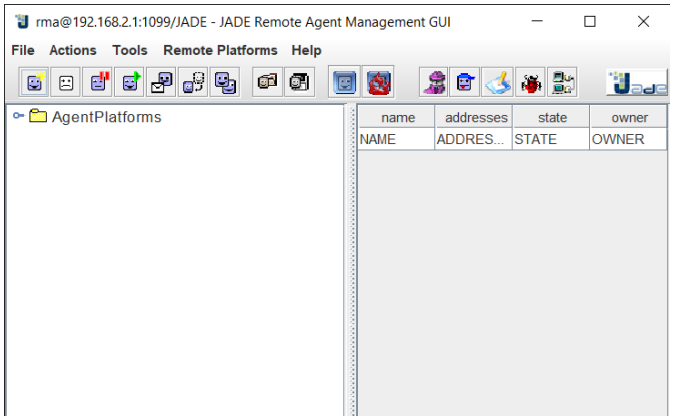


FIGURE 2.2 – Interface graphique de l'agent RMA

d. Sniffer Agent

Le Sniffer est une application Java conçue pour surveiller les messages échangés dans un environnement basé sur JADE. Il est intégré à la plateforme JADE et s'avère particulièrement utile pour le débogage des comportements des agents.

Le Sniffer est fondamentalement un agent conforme aux spécifications FIPA, doté de fonctionnalités de surveillance (sniffing) [4].

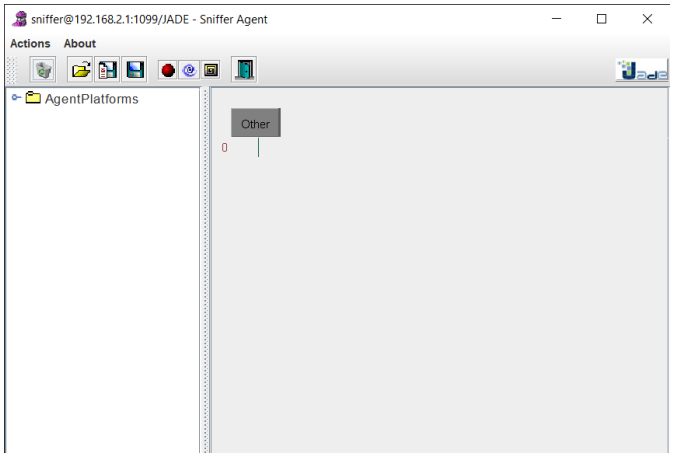


FIGURE 2.3 – Agent Sniffer

2.3 Applications basées sur JADE

a. Définition

Une application JADE repose sur une architecture distribuée orientée agents, dans laquelle chaque composant logiciel (appelé agent) agit de manière autonome, collabore avec d'autres, et prend des décisions en fonction de son environnement et de ses objectifs.

b. comparaison

Type de comparaison	**Applications JADE (multi-agents)**	**Applications traditionnelles**
Communication	Par messages ACL (asynchrone, décentralisée)	Par appels directs ou centralisés
Réactivité	Réaction en temps réel aux événements	Réaction prédéfinie, souvent lente à adapter
Contrôle du système	Distribué entre agents autonomes	Centralisé, contrôlé par un programme principal
Adaptabilité	Dynamique et automatique (auto-organisation)	Requiert des mises à jour manuelles
Architecture	Modulaire, évolutive et extensible	Rigide, difficile à modifier ou étendre

TABLE 2.1 – Comparaison entre JADE et les applications traditionnelles

2.4 Conclusion

La plateforme JADE fournit un cadre robuste pour développer des systèmes multi-agents distribués. Grâce à ses composants (**AMS**, **DF**, **RMA**, **Sniffer**), elle offre autonomie, extensibilité et observabilité, tout en respectant les standards FIPA. Elle constitue un socle fiable pour les applications nécessitant une architecture multi-agents robuste et responsive.

Chapitre 3

Conception

3.1 Introduction

Ce chapitre présente la conception de notre application de livraison de colis basée sur une architecture multi-agents. Il détaille les agents mis en œuvre, leur rôle, ainsi que les diagrammes UML utilisés pour modéliser le système.

3.2 Description de l'application

Dans ce projet, nous travaillons sur une application d'entreprise pour des colis électroniques standards

Cette application simule un système intelligent de livraison de colis en utilisant la plateforme JADE (Java Agent DEvelopment Framework). Le système repose sur une architecture multi-agents avec les entités suivantes :

- **AgentClient :** initie la demande de livraison.
- **AgentTri :** analyse la demande et assigne un agent livreur.
- **AgentLivreur :** exécute la livraison et notifie la fin de mission.

Les agents communiquent par messages ACL, conformément aux normes FIPA. Chaque agent s'enregistre dynamiquement auprès du DF (Directory Facilitator), tandis que l'AMS (Agent Management System) supervise la plateforme.

3.3 Diagramme de cas d'utilisation

Les diagrammes de cas d'utilisation (DCU) sont des diagrammes UML utilisés pour une représentation du comportement fonctionnel d'un système logiciel. Ils sont utiles pour des présentations auprès de la direction ou des acteurs d'un projet, mais pour le développement, les cas d'utilisation sont plus appropriés. En effet, un cas d'utilisation (use cases) représente une unité discrète d'interaction entre un utilisateur (humain ou machine) et un système. Ainsi, dans un diagramme de cas d'utilisation, les utilisateurs sont appelés acteurs (actors), et ils apparaissent dans les cas d'utilisation.[5]

Acteurs

Les acteurs sont des entités externes qui interagissent avec le système, comme une personne humaine ou un robot. Une même personne (ou robot) peut être plusieurs acteurs pour un système, c'est pourquoi les acteurs doivent surtout être décrits par leur rôle. Ce rôle décrit les besoins et les capacités de l'acteur. Un acteur agit sur le système (Figure 2.1).

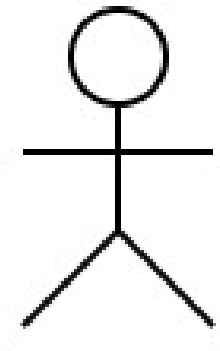


FIGURE 3.1 – Représentation d'un acteur UML

Relations

Trois types de relations sont prises en charge par la norme UML et sont graphiquement représentées par des types particuliers de ces relations. Les relations indiquent que le cas d'utilisation source présente les mêmes conditions d'exécution que le cas issu. Une relation simple entre un acteur et une utilisation est un trait simple.

Inclusions

Dans ce type d'interaction, le premier cas d'utilisation inclut le second et son issue dépend souvent de la résolution du second. Ce type de description est utile pour extraire un ensemble de sous-comportements communs à plusieurs tâches, comme une macro en programmation. Elle est représentée par une flèche en pointillé et le terme include.

Extensions

Les extensions (extend) représentent des prolongements logiques de certaines tâches sous certaines conditions. Autrement dit un cas d'utilisation A étend un cas d'utilisation B lorsque le cas d'utilisation A peut être appelé au cours de l'exécution du cas d'utilisation B. Elle est représentée par une flèche en pointillée avec le terme extend. Ce type de relation peut être utile pour traiter des cas particuliers ou fonctions optionnelles, préciser les objectifs, ou encore pour tenir compte de nouvelles exigences au cours de la maintenance du système et de son évolution.

Généralisations

La troisième relation est la relation de généralisation ou spécialisation. Le cas d'utilisation A est une généralisation de B, si B est un cas particulier de A c'est-à-dire lorsque A peut être substitué par B pour un cas précis. Ces relations sont des traits pleins terminés par une flèche en triangle.

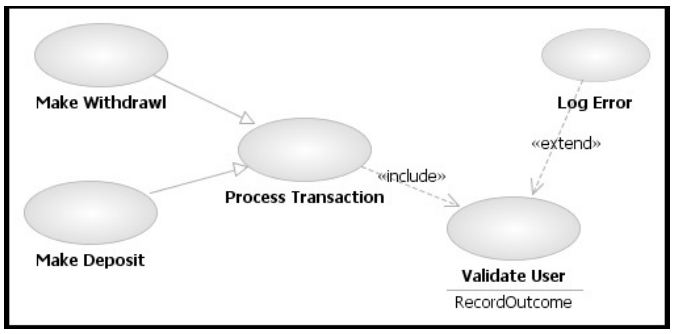


FIGURE 3.2 – Exemple de relations

voici le diagramme de cas d'utilisation de notre application de livraision de colis électronique :

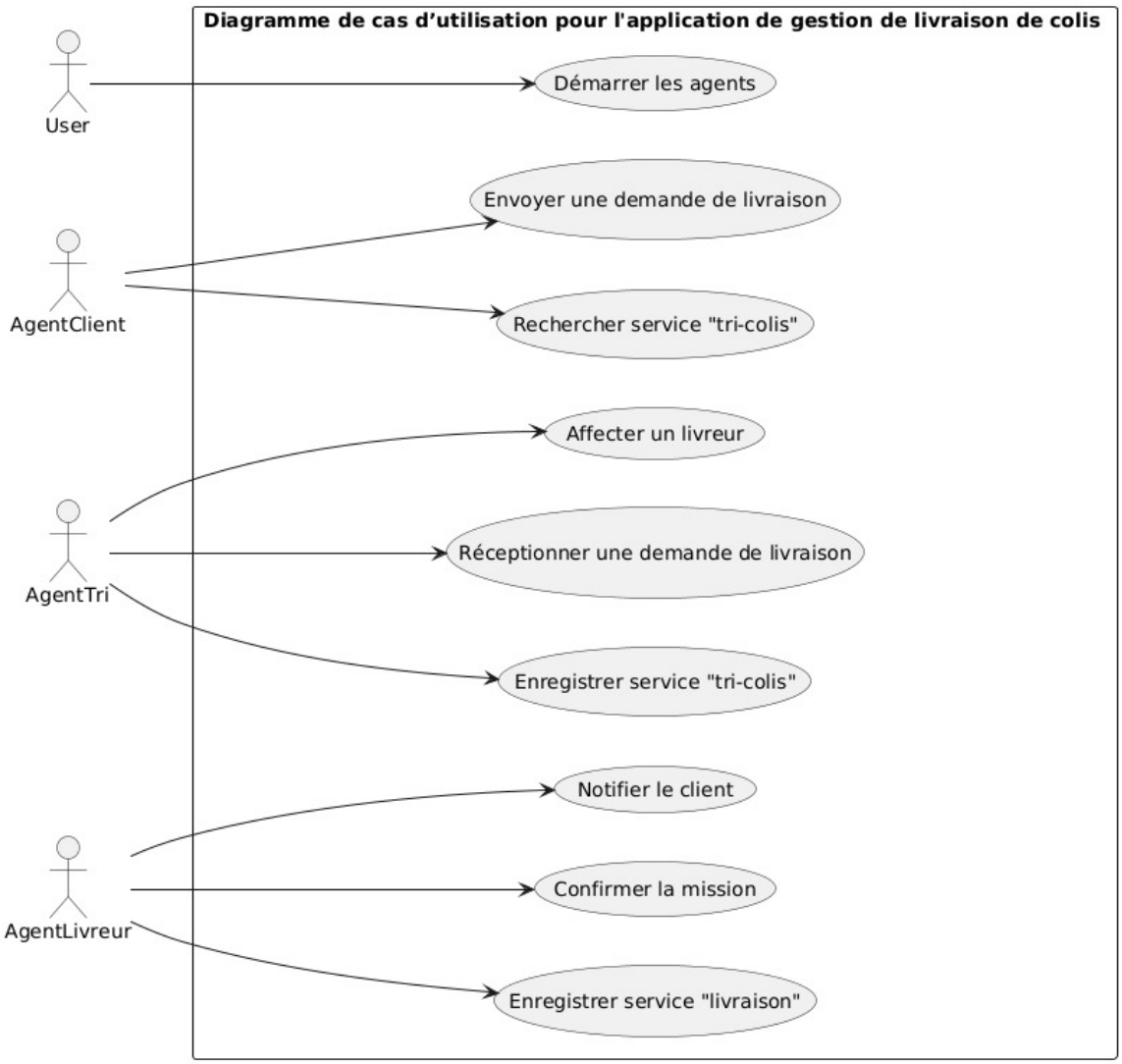


FIGURE 3.3 – Diagramme de cas d'utilisation

3.4 Diagramme de classe

Le diagramme de classes est un schéma utilisé en génie logiciel pour présenter les classes et les interfaces des systèmes ainsi que leurs relations. Ce diagramme fait partie de la partie statique d'UML, ne s'intéressant pas aux aspects temporels et dynamiques.
Une classe décrit les responsabilités, le comportement et le type d'un ensemble d'objets. Les éléments de cet ensemble sont les instances de la classe.

Une classe est un ensemble de fonctions et de données (attributs) qui sont liées ensemble par un champ sémantique. Les classes sont utilisées dans la programmation orientée objet. Elles permettent de modéliser un programme et ainsi de découper une tâche complexe en plusieurs petits travaux simples.[6]

Schéma d'une classe

Une classe est représentée par un rectangle séparé en trois parties :

- la première partie contient le **nom de la classe**,
- la seconde contient les **attributs de la classe**,
- la dernière contient les **méthodes de la classe**.

La seconde et la dernière représentent le comportement de la classe.

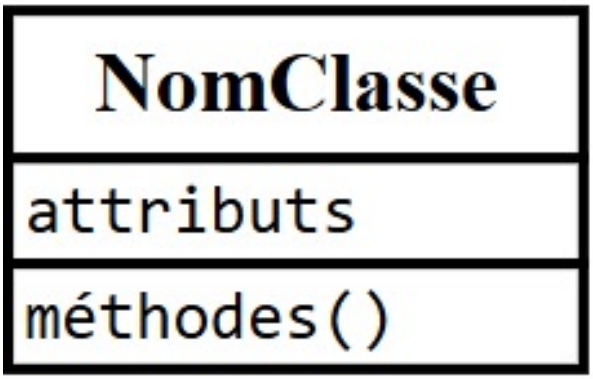


FIGURE 3.4 – Modèle d'une simple classe

voici le diagramme de classe de notre application de livraisionde colis électronique :

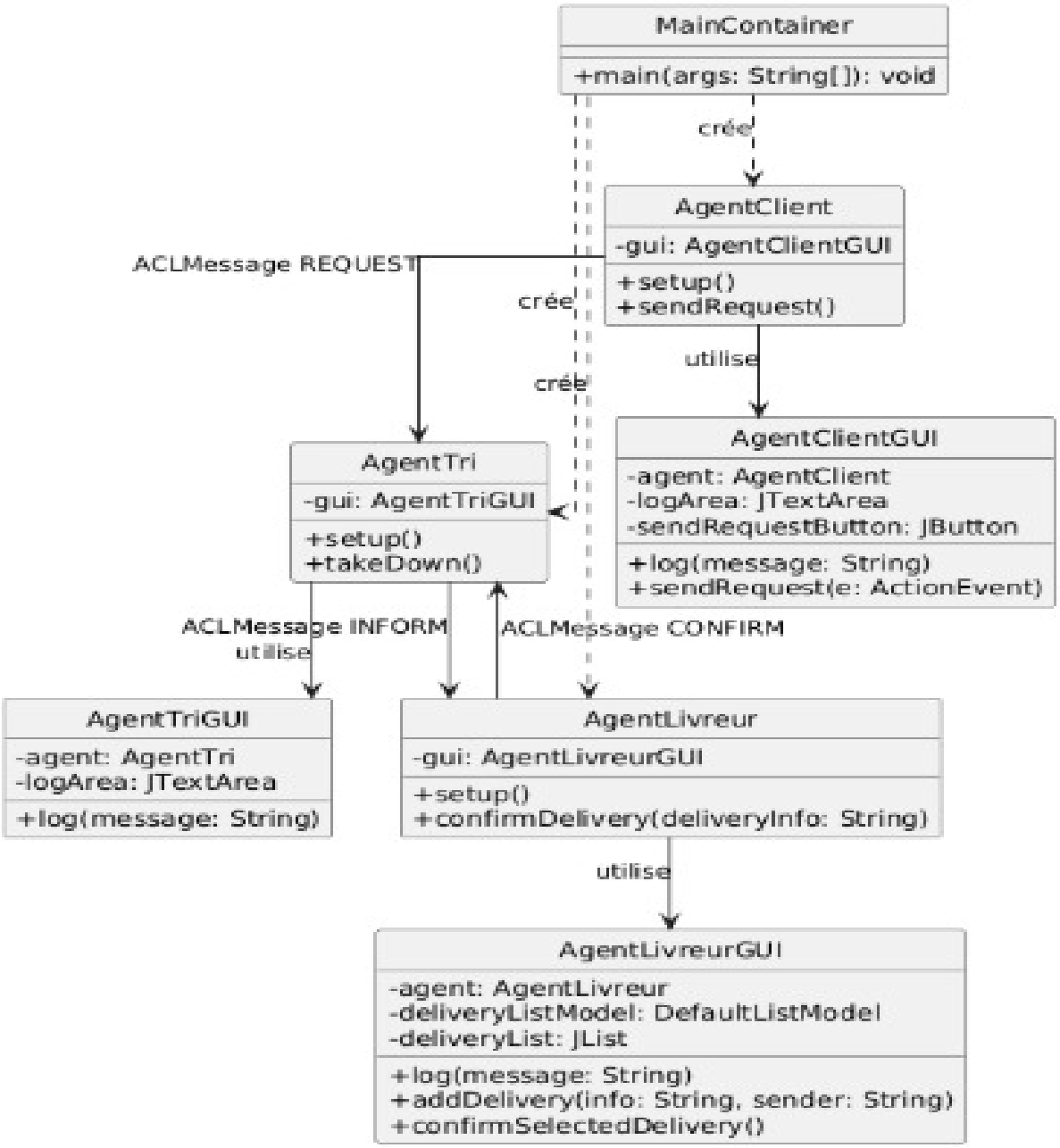


FIGURE 3.5 – Diagramme de classe

3.5 Diagramme d'interaction

Les diagrammes globaux d'interaction définissent des interactions par une variante des diagrammes d'activité, d'une manière qui permet une vue d'ensemble de flux de contrôle.

Ils se concentrent sur la vue d'ensemble de flux de contrôle où les nœuds sont des interactions ou InteractionUses.[7]

voila les diagrammes d'interaction de notre application de livraisionde colis électronique :

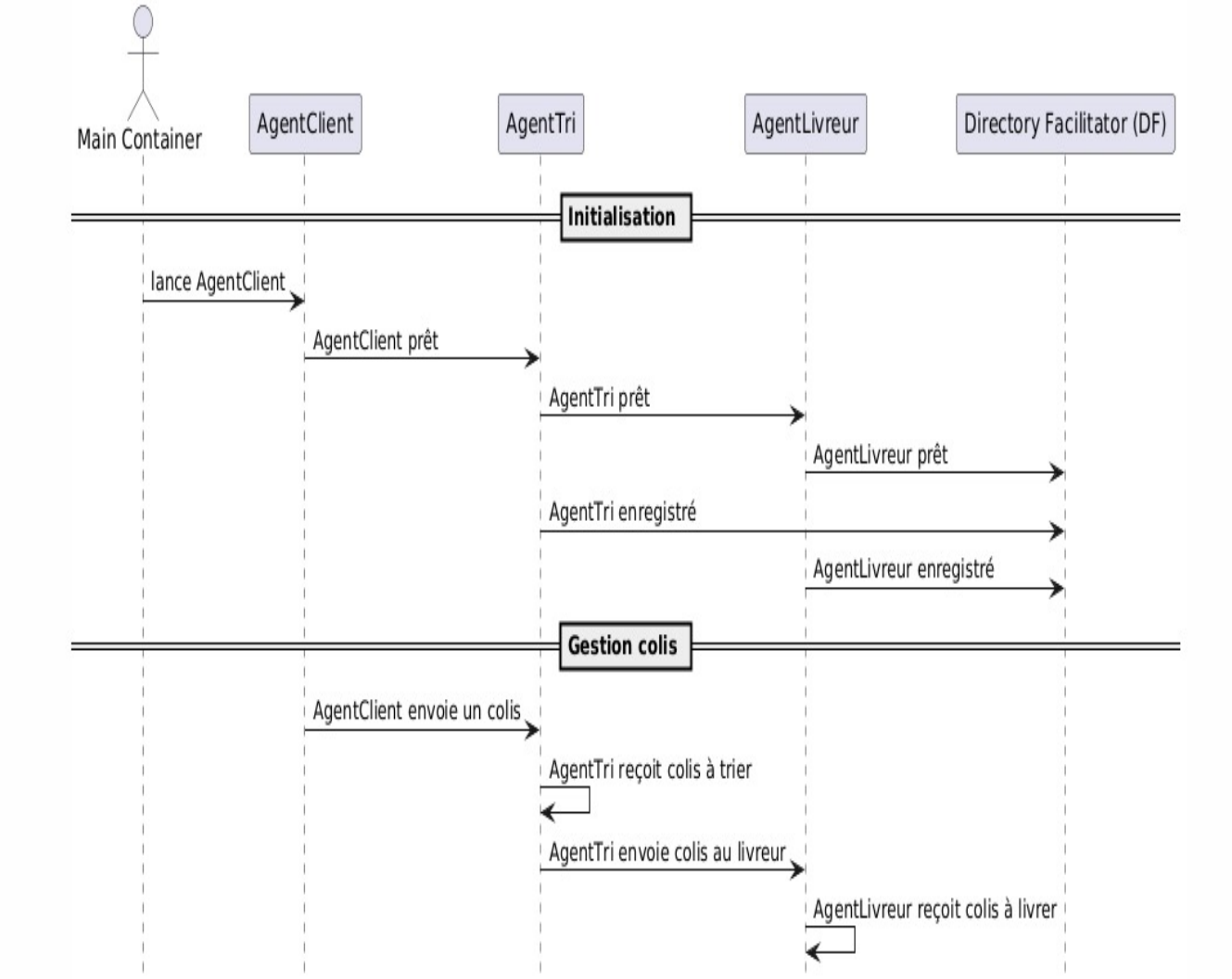


FIGURE 3.6 – Diagramme d'interaction global

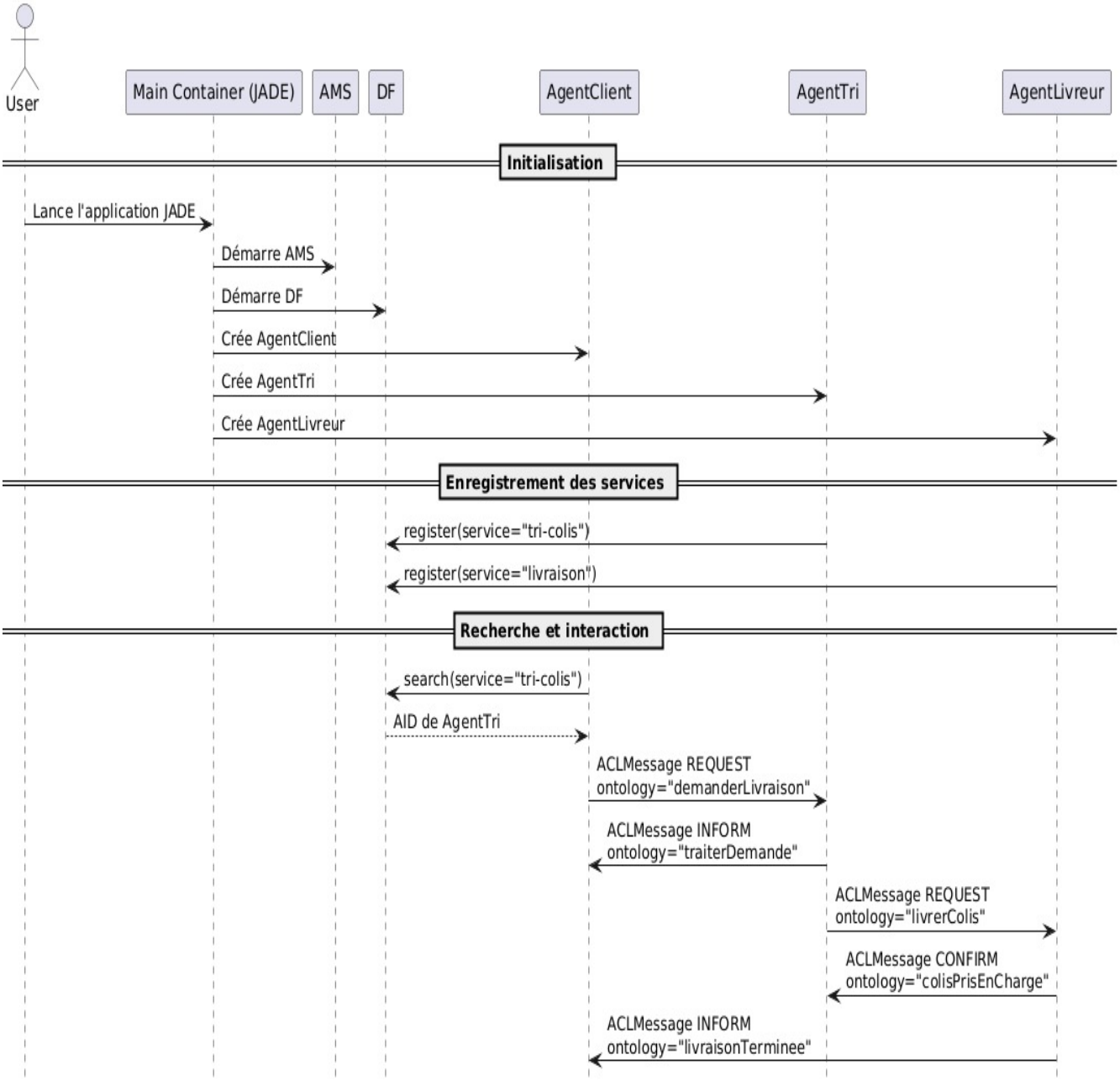


FIGURE 3.7 – Diagramme d'interaction détaillé

3.6 Diagramme d'activités

Le diagramme d'activité est un diagramme comportemental d'UML, permettant de représenter le déclenchement d'événements en fonction des états du système et de modéliser des comportements parallélisables (multi-threads ou multi-processus). Le diagramme d'activité est également utilisé pour décrire un flux de travail (workflow).[8]

voici le diagramme d'activités de notre application de livraision de colis électronique :

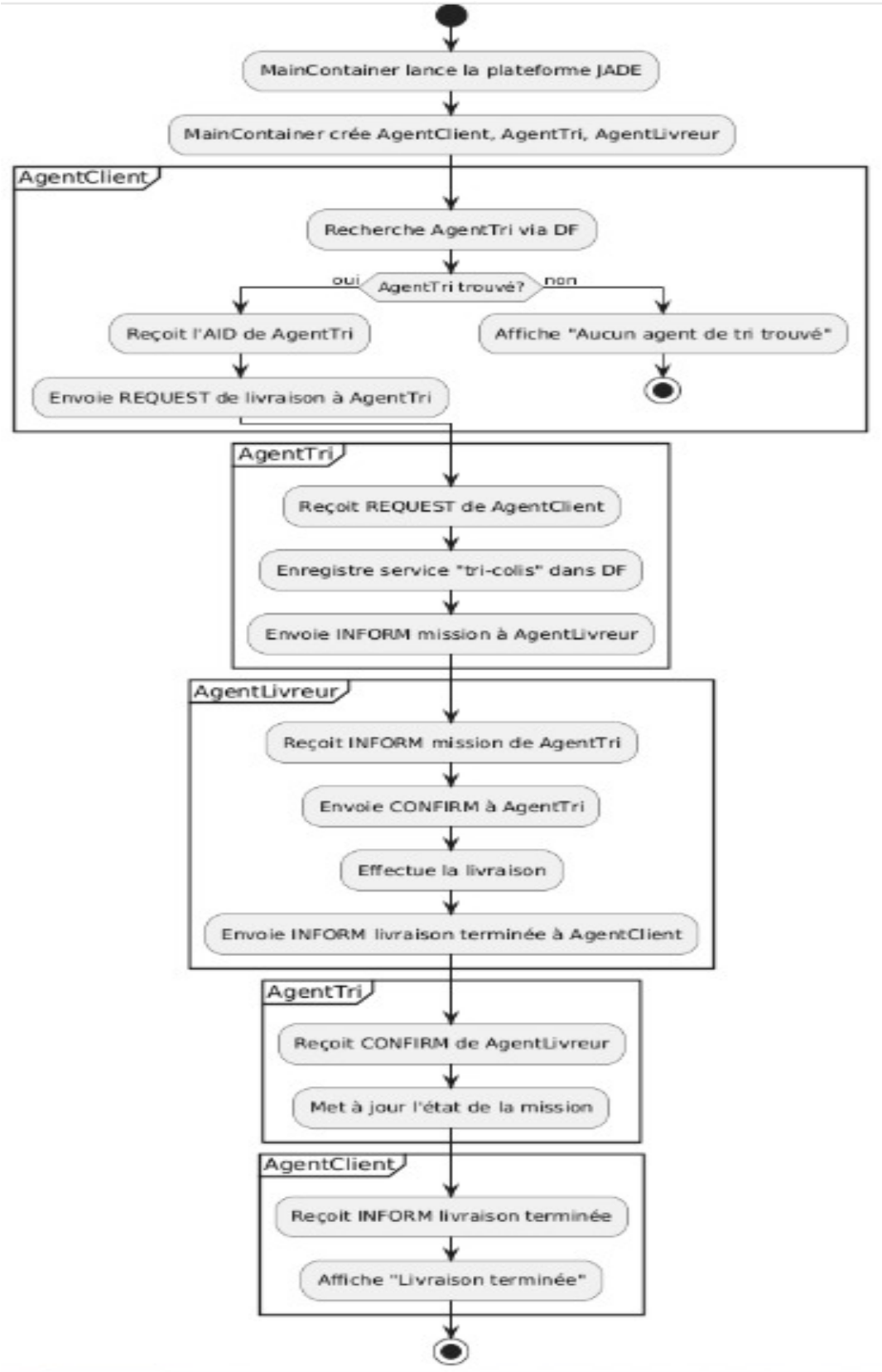


FIGURE 3.8 – Diagramme d'activités

3.7 Conclusion

La modélisation UML a permis de structurer et visualiser clairement notre système multi-agents. Chaque agent a été défini avec précision selon son rôle dans le processus de livraison. Les diagrammes de cas d'utilisation, de classes, d'interaction et d'activités ont joué un rôle clé dans la conception de l'application, en facilitant la compréhension du fonctionnement global du système et des communications inter-agents.

Chapitre 4

Réalisation

4.1 Introduction

Cette partie décrit la mise en œuvre concrète du système multi-agents conçu à l'aide de la plateforme **JADE**. Elle présente les outils logiciels utilisés, l'architecture technique adoptée, les agents développés ainsi que leurs comportements et ontologies. Enfin, les résultats obtenus, notamment via l'interface RMA et l'outil Sniffer, permettent d'illustrer le fonctionnement du système et d'analyser les interactions entre agents.

4.2 Outils utilisés

4.2.1 IDE : Eclipse et Langage Java : JADE

FIGURE 4.1 – logo Java

Langage Java est Java est un langage de programmation orienté objet, multiplateforme et largement utilisé. Il est connu pour sa portabilité, sa sécurité et sa fiabilité, ce qui en fait un choix populaire pour le développement d'applications variées, allant des applications de bureau aux applications web et mobiles, en passant par les logiciels d'entreprise et les jeux[9].

FIGURE 4.2 – IDE Eclipse

Eclipse IDE est un environnement de développement intégré libre (le terme Eclipse désigne également le projet correspondant, lancé par IBM) extensible, universel et polyvalent, permettant potentiellement de créer des projets de développement mettant en œuvre n'importe quel langage de programmation.
Eclipse IDE est principalement écrit en Java (à l'aide de la bibliothèque graphique SWT, d'IBM), et ce langage, grâce à des bibliothèques spécifiques, est également utilisé pour écrire des extensions [10].

4.3 Architecture générale

a. Conteneurs

Un conteneur est un environnement d'exécution JADE :
• Environnement complet d'exécution pour un agent
• Exécution concurrente de plusieurs agents
• Contrôle le cycle de vie des agents
• Assure la communication entre agents
• Un seul conteneur héberge l'AMS, le DF, c'est le conteneur principal (main container)
• AMS Service de Pages Blanches : référence automatiquement les agents suivant leur nom dès leur entrée dans le système.
• DF Service de Pages Jaunes : référence à leur demande les agents suivant leur(s) service(s).
• Le conteneur principal peut être répliqué via des services de réplication L'application est structurée autour de deux conteneurs :

— **MainContainer** : héberge les agents principaux et le Sniffer.

— **AgentContainer** : déploie les agents depuis un second processus.

b. Agents

La classe Agent représente la classe de base commune à tous les agents définis par l'utilisateur jade.core.Agent.

AgentClient.java

— Envoie une requête de livraison à un agent de type `tri-colis`.

— Utilise un message `REQUEST` avec ontologie `demanderLivraison`.

— Attend une réponse de confirmation après la livraison.

AgentTri.java

— Reçoit les demandes de clients.

— Transmet la mission au livreur via un message `INFORM`.

— Reçoit un `CONFIRM` de mission acceptée.

AgentLivreur.java

— Reçoit la mission de tri.

— Envoie un `CONFIRM` à AgentTri.

— Informe AgentClient une fois la livraison effectuée.

4.4 Implementation de Code

1. Comportements et Services des Agents

Agent	Comportement(s)	Service(s)	Définition / Rôle
AgentClient	`setup()` (comportement implicite)	Aucun	Envoie la demande de livraison avec `REQUEST`
AgentTri	`setup()`, `takeDown()`	`tri-colis`	Trie les colis et gère les confirmations de livraison
AgentLivreur	`CyclicBehaviour`	Aucun	Effectue la livraison à l'agent de tri
MainContainer	(Java Class)	Aucun	Conteneur principal avec RMA/Sniffer

2. Ontologies Utilisées

`Ontologie`	**Utilisée par**	**Sens**
`demanderLivraison`	AgentClient → AgentTri	Pour initier une demande de livraison de colis.
`traiterDemande`	AgentTri → AgentLivreur	Pour demander à un livreur de livrer le colis.
`livrerColis`	AgentLivreur → AgentTri	Pour confirmer que le colis a été livré.

TABLE 4.2 – Ontologies utilisées dans le système multi-agents JADE

4.5 Resultats

4.5.1 Interface graphique : RMA

L'agent RMA permet de visualiser et gérer les agents actifs. Il offre une interface graphique pour :

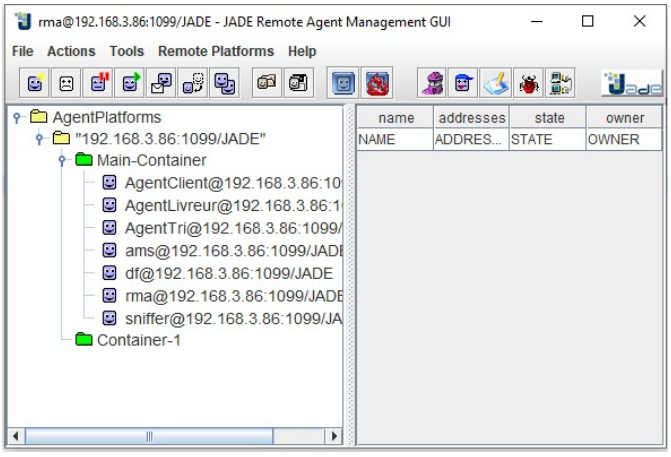


FIGURE 4.3 – Interface graphique de l'agent RMA

4.5.2 Débogage : Sniffer Agent

Le Sniffer est un outil de JADE permettant de surveiller les échanges de messages ACL entre agents. Il est essentiel pour :

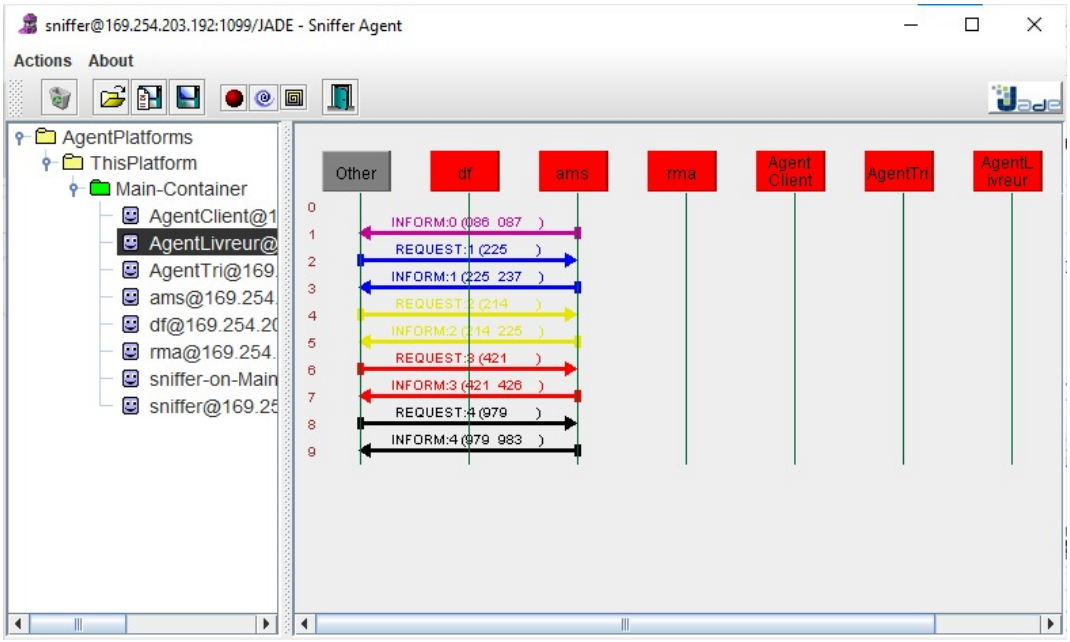


FIGURE 4.4 – Agent Sniffer

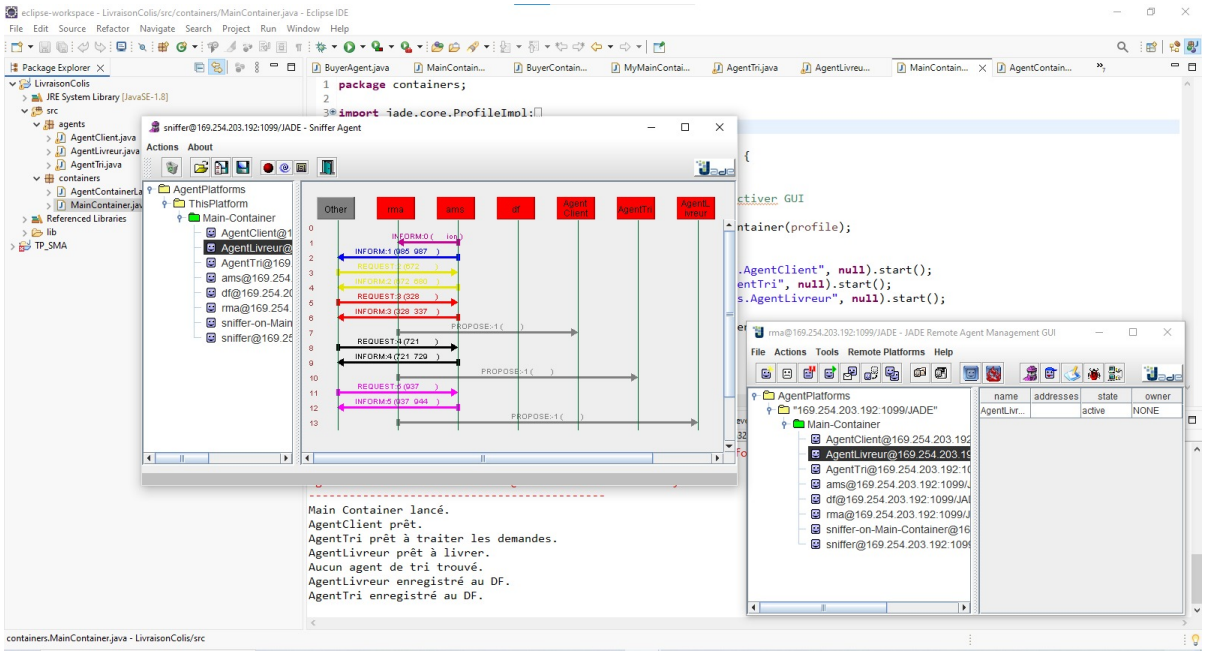


FIGURE 4.5 – Agent Sniffer 1

4.5.3 Exemple de message

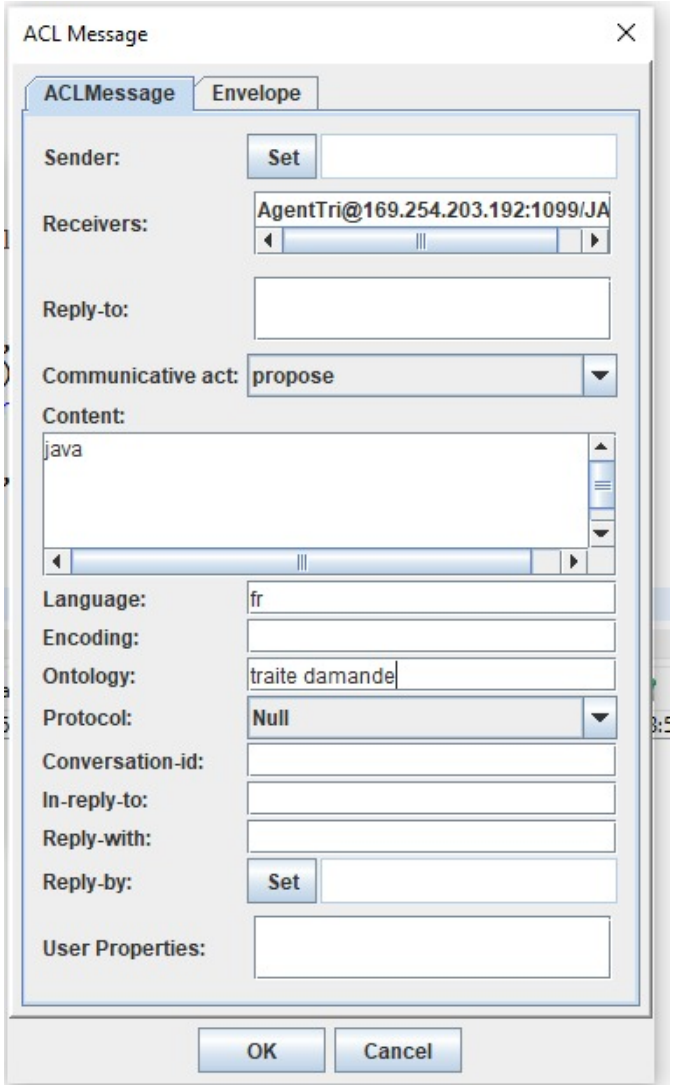


FIGURE 4.6 – message Agent

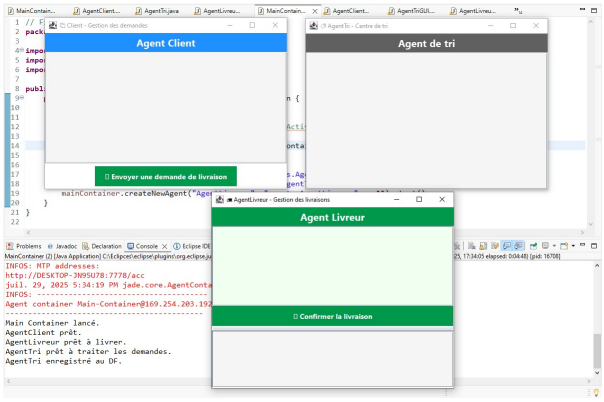
4.5.4 Console

le console avec un exemple de colis électronique (ex. Laptop).

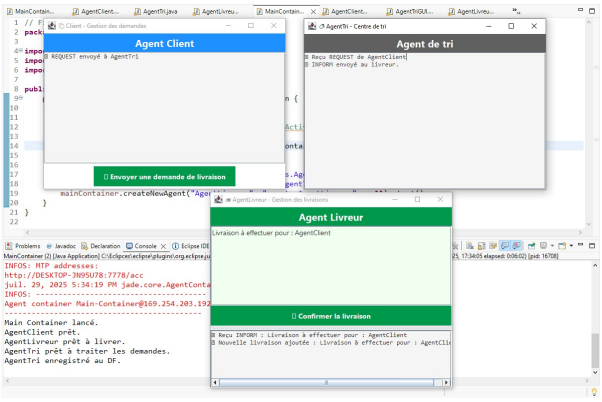
```
INFOS: Service jade.core.mobility.AgentMobility initialized
juil. 28, 2025 1:39:16 PM jade.core.BaseService init
INFOS: Service jade.core.event.Notification initialized
juil. 28, 2025 1:39:16 PM jade.mtp.http.HTTPServer <init>
INFOS: HTTP-MTP Using XML parser com.sun.org.apache.xerces.internal.jaxp.SAXParserImpl$JAXPSAXParser
juil. 28, 2025 1:39:16 PM jade.core.messaging.MessagingService boot
INFOS: MTP addresses:
http://192.168.137.1:7778/acc
juil. 28, 2025 1:39:16 PM jade.core.AgentContainerImpl joinPlatform
INFOS: ----------------------------------------
Agent container Main-Container@192.168.2.1 is ready.
--------------------------------------------
Main Container lancé.
AgentClient prêt.
AgentTri prêt à traiter les demandes.
AgentLivreur prêt à livrer.
AgentTri enregistré au DF.
AgentLivreur enregistré au DF.
AgentTri a reçu un colis à trier : 73bec82e-5dd1-478a-9d86-07fc5738d86f;Jean Dupont;2.5;123 Rue de Paris, 75000 Paris
AgentTri a envoyé le colis au livreur.
AgentClient a envoyé un colis : 73bec82e-5dd1-478a-9d86-07fc5738d86f;Jean Dupont;2.5;123 Rue de Paris, 75000 Paris
AgentLivreur a reçu un colis à livrer :
 - ID : 73bec82e-5dd1-478a-9d86-07fc5738d86f
 - Destinataire : Jean Dupont
 - Poids : 2.5 kg
 - Adresse : 123 Rue de Paris, 75000 Paris
```

FIGURE 4.7 – exemple de colis

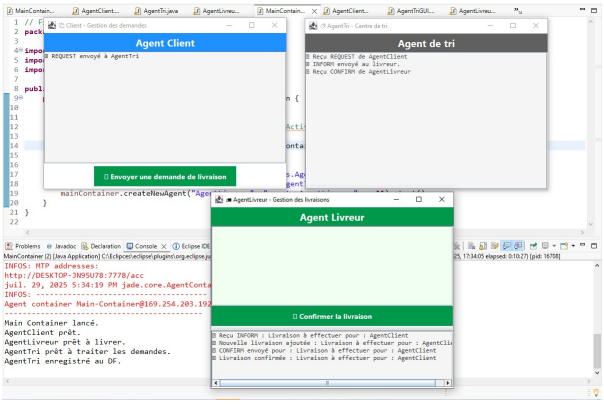
4.5.5 les interfaces graphique (GUI)



(a) Interface GUI - Agent Client et Initialisation des Agents



(b) Interaction entre Agents - Demande de Livraison



(c) Finalisation du Processus - Confirmation par le Livreu

FIGURE 4.8 – les interfaces graphique (GUI)

Figure (a) : Initialisation des Agents

`Contenu :`

La figure montre l'interface graphique d'un Agent Client avec la création d'un nouvel agent via la commande mainContainer.createNewAgent("Agent").

Les logs indiquent que le Main Container est lancé avec les adresses MTP (Message Transport Protocol).

Les agents sont initialisés et prêts :

AgentClient : Prêt à envoyer des demandes.

AgentLivreur : Prêt à effectuer des livraisons.

AgentTri : Prêt à traiter les demandes et enregistré dans le DF (Directory Facilitator).

Contexte : Cette capture d'écran illustre la phase d'initialisation du système où tous les agents sont opérationnels et connectés.

Figure (b) : Demande de Livraison

`Contenu :`

AgentClient envoie une REQUEST (demande de livraison) à AgentTri, contenant une liste de données.

AgentTri reçoit la demande, la traite, et envoie un INFORM à AgentLivreur pour notifier la livraison à effectuer.

AgentLivreur affiche un message indiquant la livraison à effectuer pour AgentClient.

Éléments clés :

Logs confirmant l'envoi/réception des messages ACL (Agent Communication Language).

Bouton "Envoyer une demande de livraison" dans l'interface client.

Contexte : Cette figure montre le flux de communication entre agents lors d'une nouvelle demande de livraison.

Figure (c) : Confirmation de Livraison

`Contenu :`

AgentLivreur envoie un CONFIRM à AgentTri pour valider la prise en charge de la livraison.

AgentTri reçoit la confirmation et met à jour son état.

Les logs affichent :

"Livraison confirmée : Livraison à effectuer pour : AgentClient".

Durée écoulée depuis le démarrage.

Lien de GitHub[11] :

https://github.com/hanan-ui/livraisonColis-lectronique/tree/main/src

4.6 Conclusion

La réalisation du projet a permis de mettre en œuvre les principes de conception multi-agents à l'aide de la plateforme JADE. L'architecture déployée, reposant sur une séparation des agents dans des conteneurs, offre modularité et évolutivité. Le système fonctionne de manière autonome et coordonnée :

- Le client envoie une demande
- L'agent de tri traite la demande et désigne un livreur
- Le livreur effectue la livraison et envoie un retour Les outils Eclipse, JADE, RMA et Sniffer ont permis un développement structuré, une exécution stable, et un bon suivi des interactions entre agents.

Chapitre 5

Conclusion générale

Ce projet a permis de développer une application de livraison de colis électronique fondée sur une architecture **multi-agents**, implémentée avec le framework **JADE**. Trois agents autonomes interagissent pour automatiser le processus : de la demande du client jusqu'à la confirmation de livraison.

La modélisation UML (cas d'utilisation, classes, interactions) a facilité la conception du système, en clarifiant les rôles et les flux de communication. Cette approche décentralisée améliore l'efficacité, la flexibilité et l'autonomie du système logistique.

Ce prototype ouvre la voie à des solutions évolutives intégrant des fonctionnalités avancées telles que le suivi en temps réel, la gestion sécurisée des paiements ou l'optimisation des trajets, contribuant ainsi à moderniser les services de livraison.

Bibliographie

[1] Mohamed Chebout, *Monitoring orienté-aspect des applications multi-agents basées JADE*, ResearchGate, 2021.
Disponible sur : https://www.researchgate.net/profile/Mohamed-Chebout/publication/353014480_Monitoring_oriente-aspect_des_applications_multi-agents_basees_JADE/links/614897bea595d06017dd202f/Monitoring-oriente-aspect-des-applications-multi-agents-basees-JADE.pdf

[2] JADE, *Java Agent DEvelopment Framework*, https://jade.tilab.com/

[3] JADE Documentation, *Introduction to the RMA*, https://jade.tilab.com/documentation/tutorials-guides/jade-rma/introduction-to-the-rma/

[4] JADE Documentation, *Introduction to the Sniffer*, https://jade.tilab.com/documentation/tutorials-guides/sniffer/introduction/

[5] Wikipédia, *Diagramme de cas d'utilisation*, https://fr.wikipedia.org/wiki/Diagramme_de_cas_d%27utilisation

[6] Wikipédia, *Diagramme de classes*, https://fr.wikipedia.org/wiki/Diagramme_de_classes

[7] Wikipédia, *Diagramme global d'interaction*, https://fr.wikipedia.org/wiki/Diagramme_global_d%27interaction

[8] Wikipédia, *Diagramme d'activités*, https://fr.wikipedia.org/wiki/Diagramme_d%27activit%C3%A9

[9] Wikipedia, *Java (langage)* — Définition et explications, consulté en juillet 2025. Disponible sur : https://fr.wikipedia.org/wiki/Java_(langage)

[10] Techno-Science.net, *Eclipse IDE - définition et explications*, https://www.techno-science.net/definition/517.html

[11] code github : https://github.com/hanan-ui/livraisonColis-lectronique/tree/main/src