Relazione di Sistemi Distribuiti

A.A.2016-2017

TradeMas

“Trade Multiagent system”

https://bitbucket.org/mgiunchi/trademas

Giunchi Massimiliano

Indice

Introduzione

Questo progetto si pone come obiettivo quello di realizzare un Sistema Multi Agente (sviluppato con il framework JADE) per l'organizzazione del trasporto di merci su strada; da una parte sono presenti gli attori che offrono il servizio di trasporto (spedizioniere, chiamato in seguito shipper), dall'altra avremo coloro che richiedono uno spostamento di merci (contraente, chiamato in seguito buyer). Le due tipologie di soggetti saono in competizione tra di loro e sarà compito del sistema a trovare soluzioni che riescano, tendenzialmente, a massimizzare gli obiettivi ed i vincoli dei partecipanti.

1 Aspetti fondamentali e richiami teorici

1.1 Sistemi distribuiti

Solitamente un sistema distribuito viene definito come un insieme di computer indipendenti che comunica attraverso la rete e che appare all'utente come un singolo sistema coerente; esempi di sistemi distribuiti sono una rete di workstation, un workflow information system, il web. Gli obiettivi che si pone di realizzare sono molteplici tra cui connettere utenti e risorse, offrire trasparenza, realizzare scalabilità ed incrementare la tolleranza ai guasti. Un sistema distribuito svolge computazioni distribuite, ossia calcoli dove almeno due processi hanno contesti spaziali differenti; appare evidente, quindi che il concetto di "distribuito" è fortemente legato a quello di contesto spaziale.

1.2 Gli agenti

Un agente è una qualunque entità in grado di percepire e di interagire con l'ambiente che lo circonda (attraverso sensori e attuatori fisici o astrazioni software). È caratterizzato dal fatto di essere, almeno parzialmente, autonomo e almeno idealmente, "intelligente" in quanto cerca di raggiungere gli obiettivi per i quali è stato programmato; è quindi possibile considerare come agente un programma informatico, un robot, un essere umano.

In informatica, per agente, se utilizziamo la *weak notation*[1], si intende un sistema automatico che porta a termine compiti utili ed è caratterizzato dal fatto di essere:

- autonomo: l'azione è guidata da regole e direttive "interne";
- reattivo: percepisce e interagisce con l'ambiente che lo circonda;
- proattivo: può intraprendere azioni di propria iniziativa in funzione dell'obiettivo;
- sociale: comunica con altri agenti.

La *strong notation*[2] di agente è un'estensione della precedente ed aggiunge proprietà umanistiche e mentali quali *convinzioni, desideri* e *intenzioni*.

Nel campo dell'Intelligenza Artificiale, per Agente Intelligente (o Agente Razionale) si intende un agente in grado di "ragionare" in maniera flessibile di fronte alle diverse situazioni, sfruttando la conoscenza posseduta, dunque esibisce un comportamento intelligente, inteso come la capacità di attuare «la cosa giusta al momento giusto».

[1] Introdotta da M. Wooldridge e N. R. Jennings, "Intelligent agents: theory and practice", *The Knowledge Engineering Review*, vol. 10(2), pp. 115-152, 1995

[2] È stata introdotta da Y. Shoham, "Agent Oriented Programming", *Artificial Intelligence*, vol. 60(1), pp. 51-92, 1993.

1.3 I sistemi multiagente

Un sistema multiagente (MAS) è un insieme di agenti situati in un preciso ambiente ed interagenti tra loro mediante un'opportuna organizzazione; costituiscono una tipologia di modellazione di società ed hanno vasti campi d'applicazione che si estende dallo sviluppo software fino alle scienze umane e sociali (economia, sociologia, psicologia, etc.). Questo tipo di sistema si basa oltre al concetto di agente su quello di contenitore di agenti (*container*) ove l'insieme di tutti i contenitori è detta *piattaforma*; tutto ciò permette di modellare una struttura software complessa costituita da sistemi cooperanti tra loro per il raggiungimento di un obiettivo comune. La piattaforma ad agenti rappresenta una macchina virtuale per l'esecuzione degli agenti, una sorta di contenitore dove essi risiedono e possono operare.

Va sottolineato come ogni agente non è in grado di risolvere un determinato problema in autonomia e per questa ragione la cooperazione per il raggiungimento di un determinato obiettivo deve necessariamente prevedere lo scambio di informazioni tra di essi; da qui nasce la necessità di definire uno standard per la comunicazione che comprenda:

- un protocollo ed un linguaggio di comunicazione che descriva le tecniche di comunicazione tra gli agenti ed un linguaggio che sia indipendente dalla struttura fisica degli stessi;
- un formato per il contenuto dell'informazione che sia univoco e che fornisca la possibilità ad ogni agente di poter riconoscerne il contenuto;
- una o più *ontologie*: fissati il linguaggio ed il formato per il contenuto dell'informazione, si devono definire le ontologie per poter attribuire un significato apprezzabile ai contenuti trasmessi.

Uno dei linguaggi di comunicazione tra Agenti è ACL (Agent Communication Language) che verrà approfondito successivamente.

I MAS sono oggetto di studio da parte di diverse discipline:

- intelligenza artificiale: per gli aspetti decisionali;
- sistemi distribuiti: per le interazioni tra agenti e la delocalizzazione dell'esecuzione;
- ingegneria del software: per l'evoluzione delle componenti sempre più dotate di autonomia e efficienza;

Questa varietà di discipline cui si applicano, fa sì che siano utilizzati per affrontare diverse tematiche:

- Quali azioni possono compiere gli agenti? Come implementarle?
- Quali modelli di comportamento devono seguire?
- Quale sistema di comunicazione devono adottare?
- Come devono reagire di fronte a situazioni non previste?

A tal proposito nel corso degli anni sono nati diversi framework di supporto allo sviluppo dei sistemi multiagente quali JADE, che sarà trattato successivamente.

2 JADE

In questo capitolo verrà descritta la piattaforma ad agenti JADE (Java Agent Development Framework) che è stata impiegata per sviluppare il progetto.

2.1 Framework

JADE è un framework sviluppato in Java che supporta lo sviluppo di applicazioni distribuite e basate sul paradigma di programmazione ad agenti, fornendo un insieme di servizi di base, conformi allo standard FIPA[3], necessari alla creazione e al mantenimento di un sistema multiagente.
Concepito e sviluppato al TILAB (Telecom Italia Lab) di Torino, è un software open source e viene distribuito con licenza LGPL.

2.2 Obiettivi e caratteristiche

L'obiettivo di JADE è quello di semplificare lo sviluppo di MAS garantendo la conformità agli standard attraverso una serie completa di servizi.
JADE può essere considerato un middleware con un'architettura orizzontale e non invasivo in quanto non si occupa degli aspetti interni degli agenti (questa libertà è lasciata agli sviluppatori); esso, altresì, mira a gestire quegli aspetti che sono indipendenti dalle applicazioni, come il trasporto dei messaggi, la loro codifica, l'analisi di essi, o il ciclo di vita dell'agente. Di fatto è facilitato il compito dello sviluppatore che può dedicarsi esclusivamente all'implementazione degli agenti e non alle questioni "di contorno".
JADE è implementato interamente in linguaggio Java e prevede che anche gli agenti siano sviluppati nel medesimo linguaggio.

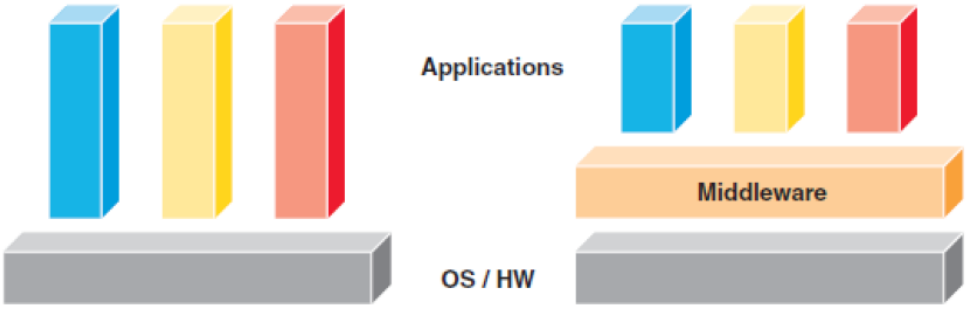

Figura 1 architettura middleware di JADE

[3] La Foundation for Intelligent, Physical Agents è una fondazione no-profit nata nel 1996 il cui scopo è generare modelli di riferimento in grado di essere usati dai progettisti per implementare sistemi complessi, in qualsiasi area e settore, raggiungendo attraverso dei modelli un elevato livello di interoperabilità tra tecnologie differenti.

Alcune delle caratteristiche principali:

- interoperabilità: conformità alle specifiche FIPA per la comunicazione e l'interazione fra gli agenti;
- supporto alla mobilità degli agenti;
- supporto per ontologie e content-language definiti dal progettista;
- open source;
- sistema "leggero" in grado di essere eseguito anche su dispositivi poco performanti;
- fornisce strumenti necessari alla creazione, gestione, debug e monitoraggio del sistema.

2.3 Architettura e componenti principali

L'architettura della piattaforma ad agenti JADE è conforme alle specifiche FIPA e comprende tutti quegli aspetti necessari alla sua gestione e funzionamento; quella software, invece, è basata sulla coesistenza di più Java Virtual Machine e di un sistema di comunicazione sviluppato tramite Java RMI (Remote Method Invocation).

I principali componenti di JADE sono:

- *agent*: corrisponde all'agente software istanziato; ogni agent deve risiedere in un container;
- *container*: un'istanza in esecuzione di JADE; ogni container è un oggetto server RMI che si occupa della gestione degli agenti locali (sulla stessa AP) creandoli, sospendendoli, riattivandoli e terminandoli;
- *platform*: un insieme di container, distribuiti in rete; una platform corrisponde a un MAS, che può risiedere sulla stessa macchina o può essere "splittata" in rete su diversi host.

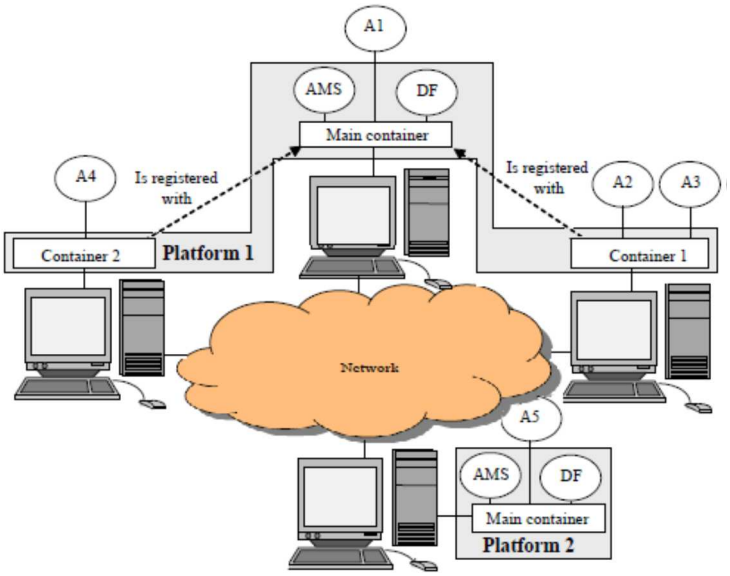

Figura 2: struttura di JADE

Ogni platform contiene più container, ma uno soltanto è il *Main Container*, che deve essere sempre attivo: esso funge da riferimento per l'intera piattaforma ed ogni altro container deve registrarsi presso di esso.

Qualora sia avviato un nuovo Main Container nella rete, esso costituirebbe una nuova piattaforma; inoltre il Main Container, fin dal primo avvio, contiene alcuni agenti speciali quali *Remote Management Agent* (RMA), *Agent Management System* (AMS) e *Directory Facilitator* (DF):

- RMA: è l'agente che permette di monitorare da remoto la piattaforma e le sue componenti; ha una propria GUI e permette di monitorare da remoto anche altre piattaforme; esso viene creato di default nel Main Container ma, a differenza degli agenti speciali quali AMS e DF, può essere spostato in un altro container. Inoltre è possibile avere più agenti RMA per la medesima piattaforma (e di conseguenza più GUI per monitorare il sistema).
- AMS: è l'agente responsabile della piattaforma, che ne supervisiona l'accesso e l'utilizzo da parte degli altri agent; è l'unico agente in grado di eseguire azioni di gestione della piattaforma, come ad esempio "create" o "kill" gli agenti o eseguire lo "shut down" della piattaforma (in genere gli altri agenti per eseguire queste azioni inviano delle richieste proprio all'AMS); per ogni piattforma deve esistere un solo AMS e deve risiedere nel Main Container. Esso inoltre fornisce i servizi di *pagine bianche* e di controllo del *ciclo di vita* degli agenti, ossia mantiene un elenco degli AID (identificatori degli agenti) e tiene traccia dello stato degli agenti: tutti gli agenti presenti nella platform devono registrarsi presso l'AMS per ottenere un AID valido e il loro stato verrà costantemente monitorato da esso.
- DF: si tratta dell'agente che provvede al servizio di *pagine gialle* della piattaforma, ossia è un servizio volto ad informare il sistema (AMS ed altri agenti) dell'esistenza e delle funzionalità degli agenti iscritti; esso comprende:
    - agenti che offrono servizi che si iscrivono al DF specificandone le caratteristiche;
    - agenti cui occorre un determinato servizio (da un altro agente) che lo cercano nel DF;

  All'avvio di una piattaforma viene creato nel Main Container un agente DF; è possibile, tuttavia, che anche altri agenti di tipo DF possano essere attivati per formare, con quello di default, un sistema di pagine gialle distribuito (federazioni di DF).

2.3.1 Implementazione di un agente

Un agente, dal punto di vista del programmatore, è una classe Java che estende la classe `Agent` contenuta nella libreria jade.jar; per inizializzarlo occorre ridefinire il metodo `setup()` all'interno del quale è possibile, ad esempio, allocare le prime risorse e definire una lista di compiti (task, o *behaviour*) che deve compiere.

Per terminare l'agente occorre invocare il metodo `doDelete()` e, opzionalmente, le operazioni di pulizia e di de-allocazione delle risorse, che devono essere inserite nel metodo `takeDown()`.

Ogni agente è caratterizzato da un Agent IDentifier (AID), che corrisponde a:

- un identificatore unico globale (GUID) che identifica univocamente l'agente sulla piattaforma; esso assume la forma: `<name_agent>@<name_platform>:<port_platform>/JADE`;
- un elenco di indirizzi: ogni agente eredita automaticamente gli indirizzi di trasporto della propria piattaforma;
- un elenco di resolver, ossia un insieme di servizi di pagine bianche ai quali l'agente è registrato.

Generalmente per comunicare con un agente, è necessario conoscere solo il suo campo GUID nella forma `<agente-nome>@<piattaforma>`, in quanto gli agenti che comunicano nella medesima piattaforma, sono già in grado di determinare tutti gli indirizzi di trasporto di quest'ultima (comunicazione intra-platform); qualora, invece si necessiti di una comunicazione tra agenti distribuiti su piattaforme differenti (comunicazione inter-platform), gli altri campi si rivelano necessari.

2.3.2 Ciclo di vita di un agente

Gli agenti in JADE hanno un loro ciclo di vita caratterizzato da un insieme di stati che si susseguono da quando "nasce" a quando "termina"; gli stati che assume sono:

- INITIATED: l'agente è stato costruito, ma non è ancora registrato presso l'AMS; non ha nome né indirizzi e non può comunicare con gli altri agenti;
- ACTIVE: l'agente si è registrato con AMS, ha un suo AID e può accedere a tutte le varie funzionalità di JADE; esso può assolvere ai suoi compiti solo nello stato ACTIVE;
- SUSPENDED: l'agente è attualmente bloccato; il suo thread interno è sospeso e di conseguenza non esegue alcuna attività;
- WAITING: l'agente è bloccato in attesa di un evento; il suo thread è in fase di "sleeping" e monitorato dalla JVM per essere "svegliato" al verificarsi di una condizione (tipicamente quando arriva un messaggio);
- DELETED: l'agente è definitivamente terminato; il thread interno non è più eseguito e l'agente non è più registrato presso l'AMS;
- TRANSIT: un agente mobile entra in questo stato quando migra presso una nuova ubicazione (ad esempio in un altro container, o una nuova platform); il sistema continua a registrare i suoi messaggi finché non si è definitivamente stabilito quindi li notifica;
- COPY: questo stato viene utilizzato internamente da JADE per gli agenti che devono essere clonati;
- GONE: impiegato internamente da JADE quando un agente mobile è migrato in una nuova locazione ed ha già uno stato consistente.

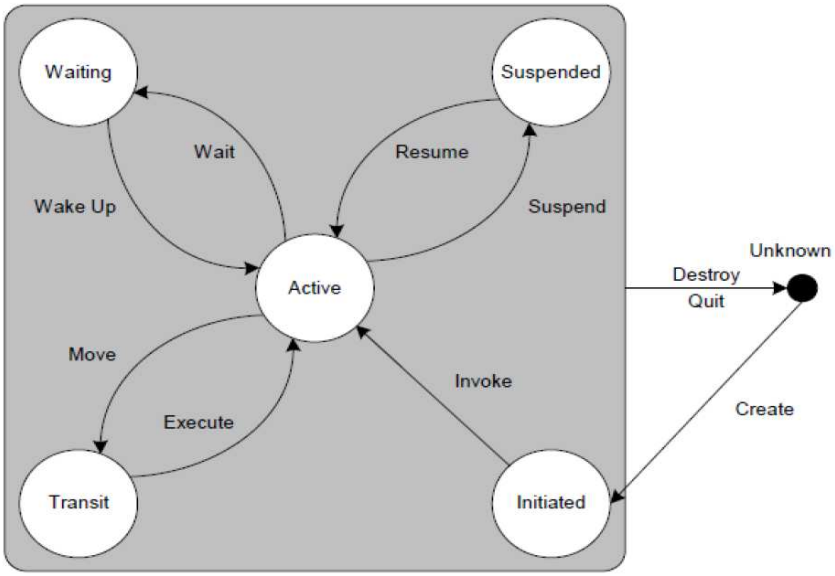

Figura 3: ciclo di vita di un agente

2.3.3 Modello computazionale di un agente

Il modello computazionale di un agente è multitask, dove i task (*behaviours*) sono eseguiti secondo il modello concorrente round-robin non-preemptive; esso è caratterizzato da:

- una politica di scheduling secondo la quale tutti i processi attivi vengono processati dalla CPU secondo un turno, assegnato in maniera circolare;
- non-preemptive, ossia senza diritto di prelazione: lo scheduler della CPU non può togliere il possesso della CPU a un certo processo, per passare il possesso ad un altro, finché il primo processo non termina (scheduling cooperativo).

Ogni funzionalità/servizio fornito da un agente deve essere implementato in uno o più behaviour; lo scheduler, nascosto al programmatore, gestirà automaticamente i behaviour.

Lo scheduling cooperativo comporta alcuni vantaggi quali:

- presenza di un solo thread per agente (questo è importante in ambienti con risorse limitate, come ad esempio gli smartphone);
- migliori performance, in quanto i behaviour sono gestiti più velocemente rispetto i Java Threads;
- eliminazione delle problematiche di sincronizzazione tra behaviour concorrenti che desiderano accedere alla medesima risorsa;
- possibilità di effettuare un'istantanea (snapshot) dell'agente: questo ha introdotto nuove importanti funzionalità, come la persistenza e la mobilità degli agenti.

Tra gli svantaggi va sottolineato come un behaviour bloccato (ad esempio in un ciclo infinito) comporta il blocco dell'agente che lo esegue.

2.3.4 Behaviour

Un agente ha dei compiti (task) da eseguire che ne definiscono il comportamento (behaviour); ogni singola funzionalità/servizio fornito da un agente, deve essere implementato in uno o più behaviour. Si tratta di una classe Java che estende la superclasse astratta `Behaviour` contenuta nella libreria `jade.jar`.

La caratteristica di un behaviour è un'azione ed una condizione di terminazione; per implementare il comportamento occorre ridefinire il metodo `action()`; per specificare invece quando un behaviour è completato, occorre implementare il metodo `done()` che restituisce un booleano. L'aggiunta di un behaviour avviene facendo ricorso al metodo `add()` nel metodo `setup()` o in altri behaviour dell'agente; per rimuoverlo, invece, si impiega `removeBehaviour()`.

Come già citato, JADE adotta il modello cooperativo: i behaviour di un agente, sono aggiunti in una coda circolare (*pool dei behaviour attivi*) ed eseguiti uno alla volta: questo implica che non è possibile implementare un behaviour successivo finché il metodo `action()` in esecuzione non sia stato completato.

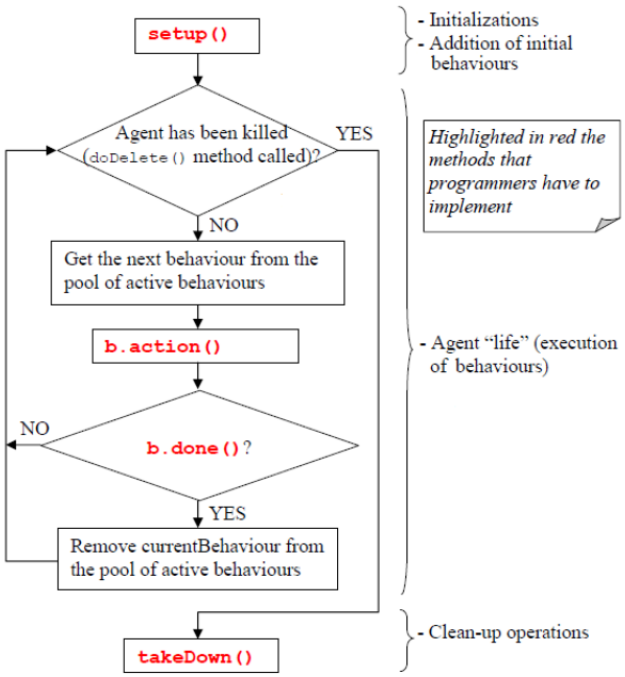

Figura 4: esecuzione dei behaviour di un agente

Oltre ad estendere la classe `Behaviour`, esistono altre tipologie di behaviour che è possibile estendere:

- `OneShotBehaviour`: per i behaviour che devono essere eseguiti una sola volta (il metodo `done()` restituisce sempre `TRUE`);
- `CyclicBehaviour`: per i behaviour che devono essere eseguiti ciclicamente (il metodo `done()` restituisce sempre `FALSE`);
- `TickerBehaviour`: esegue il behaviour ogni *x* millisecondi;
- `WakerBehaviour`: esegue il behaviour una volta sola, dopo *x* millisecondi;

- `FSMBehaviour`: permette di definire un comportamento come una macchina a stati finiti.

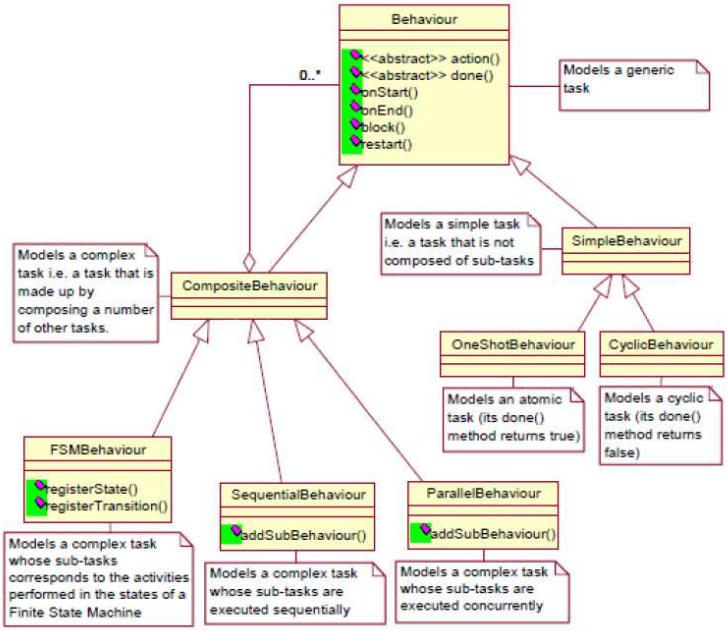

Figura 5 schema UML dei behaviour disponibili che è possibile estendere

2.3.5 Comunicazione

La piattaforma JADE utilizza come struttura fondamentale per la comunicazione i messaggi ACL (Agent Comunication Language). Un Agente che desidera inviare un messaggio deve creare un oggetto ACLMessage, impostare gli attributi con valori appropriati e quindi invocare il metodo `send()` della classe `Agent`; il paradigma di comunicazione utilizzato è il passaggio di messaggi in maniera asincrona.

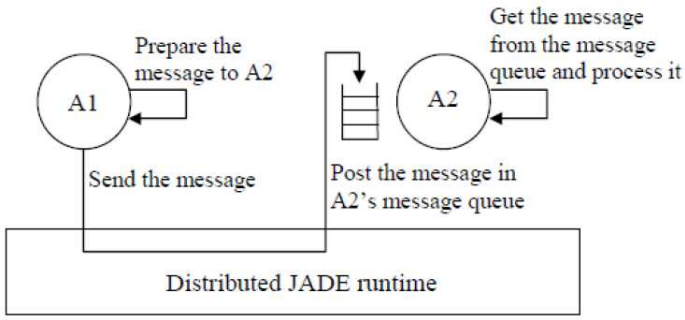

Figura 6

Ogni agente possiede una sorta di mailbox (*coda dei messaggi* dell'agente) dove vengono recapitati i messaggi inviati da altri agenti: non appena un messaggio giunge nella coda dei messaggi di un agente, a quest'ultimo arriva una notifica. È il programmatore, quindi a decidere se e quando estrarre il messaggio dalla coda per processarlo: ciò comporta che l'agente, persino nei casi in cui non è temporaneamente disponibile, non perda alcun messaggio. I messaggi scambiati dagli agenti all'interno della stessa

piattaforma, seguono il formato specificato da Agent Comunication Language (ACL), standard internazionale definito dalla FIPA.

Esso prevede un insieme di campi per ogni messaggio inviato:

Nome field	Descrizione
sender	Il mittente del messaggio
receivers	I destinatari del messaggio
Communicative Act (performative)	L'atto comunicativo, ovverosia l'intento che il mittente vuole esprimere
content	Il contenuto del messaggio: può essere una semplice stringa o un intero testo che segue uno specifico linguaggio
language	Il linguaggio scelto per esprimere il contenuto
encoding	La codifica scelta per il contenuto
ontology	Ontologia di riferimento per comprendere il contenuto
protocol	Il protocollo scelto per trattare il messaggio
reply-to	"rispondere a...", indica l'agente al quale replicare (non necessariamente il mittente)
conversation-id	Identificativo della conversazione
reply-with	"replica con...", per identificare univocamente il messaggio. Utile per gestire conversazioni concorrenti.
in-reply-to	"in replica a...", per indicare il messaggio precedente (identificato dal field reply-with) al quale si sta rispondendo. Utile per gestire conversazioni concorrenti
reply-by	"replica entro...", indica il tempo entro il quale il mittente desidera che il destinatario risponda al messaggio, prima che scada

Figura 7: struttura di un messaggio ACL

Grazie a questi campi un agente può gestire conversazioni multiple con più agenti, senza ambiguità o rischio di "confondere" una conversazione con un'altra.

2.3.6 Protocolli di interazione

I protocolli d'interazione sono modelli prestabiliti di conversazione fra agenti; sono descritti nello standard FIPA, progettati per scopi generali e si basano sugli Atti Comunicativi (performative); grazie ad essi è possibile implementare in maniera rapida conversazioni tra due o più agenti.

Il motivo per il quale può rendersi necessaria una conversazione tra agenti è il seguente: nello standard FIPA ogni messaggio rappresenta un Atto Comunicativo e per ogni atto, è specificato l'effetto/azione che ci si aspetta o la ragione per la quale il messaggio è stato inviato.

Ad esempio: *un agente X invia un messaggio di tipo request A all'agente Y (dove request è la performative ed A è il contenuto).*

Tuttavia, l'agente che ha inviato il messaggio non deve ritenere che il ricevente abbia effettivamente compiuto l'azione descritta nel messaggio, in quanto il destinatario in quel momento può essere occupato,

ignorare deliberatamente il messaggio, o fallire l'azione richiesta; questo introduce alcuni livelli di incertezza e lo scopo delle conversazioni (e cioè dei protocolli) è di eliminare questa ambiguità.

In generale, un IP (Interaction Protocol) prevede due ruoli:

- *Initiator*: l'agente che avvia la conversazione;
- *Responder*: ogni altro agente che partecipa alla conversazione.

Inoltre ogni IP prevede il tipo di messaggio da scambiare e le fasi in cui essi devono essere scambiati; allo scopo di comprendere il funzionamento generale di un IP (e quindi a scopo esplicativo), mostriamo il funzionamento del protocollo *FIPA-Request Protocol*, *FIPA-Propose Protocol* e *FIPA-Subscribe Protocol.*

- FIPA-Request Protocol:

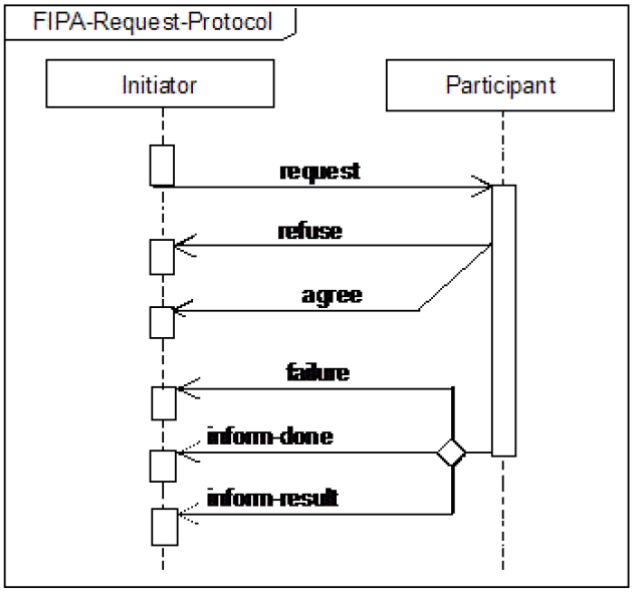

Figura 8: specifica FIPA SC00026 FIPA-Request-Protocol

In questo IP:

- l'agente Initiator richiede al/ai Responder di compiere un'azione (invia *request*);
- ciascun Responder risponde all'Initiator con *agree* o *refuse*;
- ogni Responder che ha precedentemente accettato, non appena completa l'azione o incontra problemi nell'eseguirla, informa l'Initiator (*inform-done*, *inform-result*, *failure*).

Il protocollo FIPA-Propose, permette a un agente Initiator di eseguire una certa azione purchè un agente Responder sia d'accordo.

- FIPA-Propose Protocol:

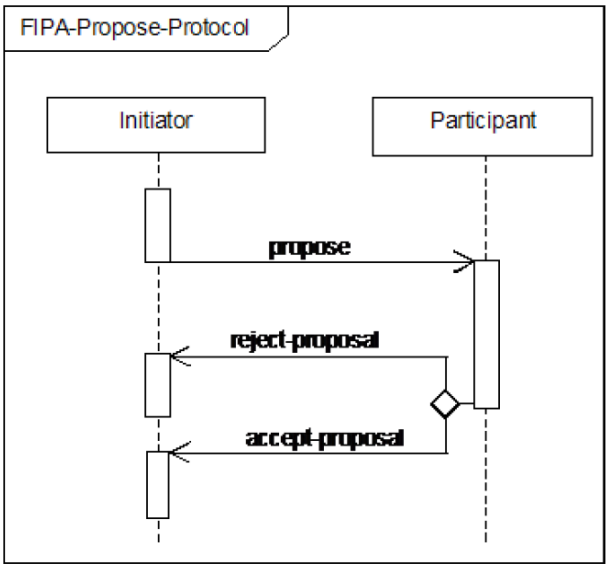

Figura 9: specifica FIPA SC00036 FIPA-Propose-Protocol

In questo IP:

- l'agente Initiator invia una *propose* al Responder;
- il Responder di invia una *refuse-proposal* o *accept-proposal*;
- nel caso di una accept-proposal tipicamente segue un'azione da parte dell'Initiator, che termina con l'invio un responso riguardo al risultato.

In JADE questo è ottenibile implementando la classe `ProposeInitiator` per l'agente Initiator e la classe `ProposeResponder` per gli agenti Responder.

- *FIPA-Subscribe Protocol*

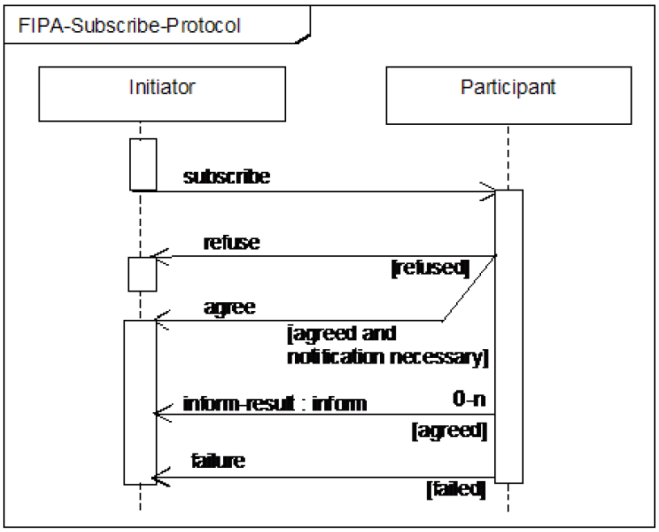

Figura 10: specifica FIPA SC00036 FIPA-Subscribe-Protocol

In questo IP:

- un agente Initiator effettua una richiesta di sottoscrizione (o abbonamento) ad un agente Responder (nelle specifiche FIPA è chiamato Participant);
- il Responder può accettare o rifiutare la sottoscrizione; in caso affermativo invierà a suo piacimento informazioni all'agente Initiator.

In JADE questo è ottenibile implementando le classi:

- `SubscriptionInitiator` per l'agente Initiator;
- `SubscriptionResponder` per l'agente Responder.

Alcuni dei protocolli di interazione descritti nello standard FIPA sono implementati in JADE, anche se con alcune modifiche; oltre a questi, uno sviluppatore è comunque libero di creare propri IP ad hoc.

Nome IP FIPA	Nome IP JADE
FIPA-Request	AchieveRE
FIPA-Request-When	
FIPA-Query	
FIPA-Brokering	
FIPA-Recruiting	
FIPA-Contract-Net	Contract-Net
FIPA-Iterated-Contract-Net	Iterated-Contract-Net
FIPA-Subscribe	Subscription
FIPA-Propose	Propose
FIPA-Auction-English	-
FIPA-Auction-Dutch	-

Figura 11: lista dei protocolli di interazione FIPA e corrispettivi JADE

3 Progettazione

3.1 Analisi del dominio

TradeMas è stato sviluppato al fine di fornire uno strumento per la pianificazione di attività di trasferimento merci tramite veicoli stradali; prima di analizzare il software in tutte le sue componenti, è necessario esaminare il dominio applicativo.

3.1.1 Identificazione degli attori

In questo contesto, esistono due differenti tipi di attori

- *responsabile dei trasporti*: indica l'utente che nel sistema coincide con l'agente *shipper* (spedizioniere), ossia colui che pianifica i trasporti di una azienda, che assegna i veicoli ai conducenti e che consegna loro i piani di viaggio. Sarà suo compito quello di rispondere alle esigenze dei suoi clienti e cercare di impegnare gli autisti evitando tempi morti.
- *responsabile delle merci*: è l'utente che nel sistema coincide con l'agente *buyer* (committente): è l'organizzatore logistico in una azienda, ovvero colui che prepara i carichi di merce che devono essere spediti da un luogo ad un altro.

3.1.2 Trasporto

Con il termine *trasporto* si intende lo spostamento di un mezzo con a bordo passeggeri e/o beni da un punto A ad un punto B di destinazione. Questa è in realtà una semplificazione in quanto tra i due punti possono esserci più punti intermedi che possono essere obbligatori o opzionali, previsti o imprevisti (es. soste per riposo, rifornimento, deviazioni, incidenti, altri carichi ecc.).
In questo progetto per trasporto intenderemo lo spostamento di un mezzo da un punto A ad un punto B.

3.1.3 Veicolo

Per rendere possibile un trasporto occorre servirsi di un mezzo, e quindi di un *veicolo*: un mezzo di trasporto motorizzato.
In questo progetto saranno presi in considerazione esclusivamente veicoli *terrestri* (cioè che si muovono su strada, escludendo quindi navi, aerei…), *civili* (destinati ad un uso non militare), *privati* (quindi escludendo le categorie dei mezzi pubblici quali autobus, metro, treni…).
Ogni veicolo appartiene a una specifica *categoria* che ne definisce le proprietà generali in base alla dimensione, tipo di carrozzeria, la portata, massa a pieno carico e al tipo di allestimenti possibili.
Le categorie trattate sono le seguenti:

- *autovettura*;

- *van*: automobile nella quale la zona dei sedili passeggeri posteriore è adibita vano di carico merci; si tratta di un veicolo commerciale con limite di massa a pieno carico di 3,5 t;
- *furgone* (o veicolo commerciale leggero): viene classificato tra gli autocarri con cabina incorporata nella carrozzeria, cioè i mezzi di trasporto specifici per il trasporto di merci; anch'esso ha il limite della massa a pieno carico di 3,5 t;
- *autocarro* (detto anche *camion* o *truck*): è un veicolo in grado di trasportare merci autonomamente, in quanto fornito di motricità propria. Spesso il termine autocarro/camion è utilizzato impropriamente per indicare altre tipologie di veicoli adibiti al trasporto merci (come gli autotreni o gli autoarticolati). La massa a pieno carico può superare le 3,5t;
- *trattore stradale* (o motrice): è un veicolo adibito esclusivamente al traino di semirimorchi con cui, una volta agganciato, forma un particolare complesso veicolare denominato *autoarticolato*;
- *rimorchio***:** è un veicolo sprovvisto di motore, destinato ad essere trainato da parte di autoveicoli equipaggiati con sistemi di traino opportuni e, all'occorrenza, anche con sistema di frenatura autonoma; qualora la massa a pieno carico del rimorchio sia inferiore o uguale ai 750 kg, si ha un *rimorchio leggero*, altrimenti si definisce *rimorchio non leggero*;
- *semirimorchio*: è un veicolo privo di motore destinato al traino da parte di un trattore stradale con cui forma un complesso veicolare definito *autoarticolato*. Il termine deriva dal fatto che il veicolo scarica il proprio peso in parte sui suoi stessi assi e in parte sulla ralla del trattore stradale che lo traina, per cui, da un punto di vista fisico, è trainato solo per metà.

Figura 12: semirimorchio

- autotreno: è un complesso veicolare composto da un'unità trainante ed una trainata sprovvista di motore; in genere per autotreno si intende un autocarro con il rimorchio. In realtà il termine autotreno indica un convoglio formato da un'unità trainante e una o più unità rimorchiate ma, per la legislazione italiana ed europea, non possono esistere complessi veicolari stradali composti da più di due unità.

Figura 13: autoreno (autocarro e rimorchio)

- *autoarticolato*: è il complesso veicolare formato da un trattore stradale trainante un semirimorchio ed è simile all'autotreno (si tratta di un particolare tipo di autotreno). Con l'autoarticolato il trasporto delle merci è in un comparto separato dall'abitacolo (le merci sono sul semirimorchio, l'abitacolo è nel trattore stradale).

3.2 Casi d'uso

L'utilizzo dei casi d'uso è un modo efficace per catturare potenziali requisiti funzionali di un sistema; ogni caso d'uso presenta uno o più scenari che dimostrano come il sistema può interagire con l'utente finale e/o con altri sistemi al fine di raggiungere un obiettivo.
Di seguito rappresentiamo uno scenario che descrive il sistema:

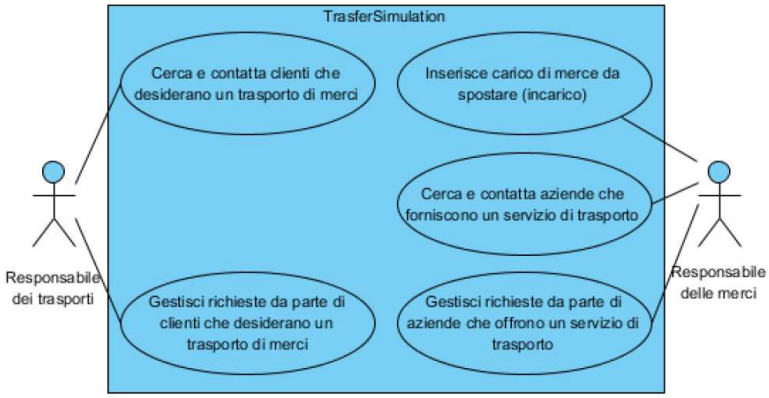

Figura 14: diagramma dei casi d'uso

3.3 Identificazione degli Agenti e relative responsabilità

In questa fase devono essere identificati i tipi di agenti principali; possiamo pensare di agire secondo i seguenti criteri:

- aggiungere un tipo di agente per utente/dispositivo;
- aggiungere un tipo di agente per risorsa.

Prima di associare un insieme di responsabilità per ogni agente identificato, va sottolineato che inizialmente ogni agente non è in grado di determinare quali e quanti altri agenti siano presenti (e disponibili) in rete; d'altra parte un agente di tipo Shipper, deve poter determinare quali agenti di tipo Buyer può contattare e dove risiedono. A tale scopo esiste una terza tipologia di agente, conosciuta a priori, che offre un servizio di Pagine Gialle presso il quale ogni agente può registrarsi; volendo rappresentare le interazioni degli agenti in uno schema, avremo:

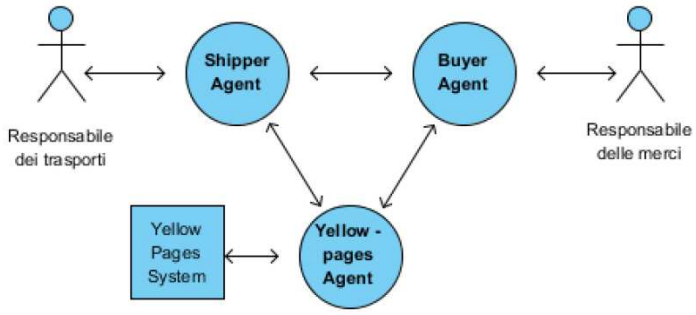

Figura 15: interazioni ti tra agenti

Alla luce di quanto sopra, possiamo determinare una tabella delle responsabilità degli agenti, dove per responsabilità intendiamo le azioni cui competono:

Tipo di agente	Responsabilità
Shipper agent	• Legge da file i dati di configurazione e si connette alla rete. • Legge da file i dati di supporto e definisce la sua flotta (nel caso tale file non esistesse ne crea uno inizialmente vuoto). • Inserisce, modifica, visualizza ed elimina i veicoli, definendo così la sua flotta. • Per ogni veicolo indica se è disponibile ad affrontare una spedizione. PUBBLICAZIONE SULLE PAGINE GIALLE • Contatta l'agente Yellow-pages chiedendo di iscriversi e autodefinendosi uno Shipper. In tal modo potrà essere contattato dagli altri agenti in rete. SCENARIO "INVIA CFP" • Chiede all'agente Yellow-pages la lista di tutti

	gli agenti Buyer disponibili in rete. • Invia dei messaggi (*Call for Proposal*) a tutti i possibili committenti (gli agenti Buyer) offrendosi come spedizioniere. In tali messaggi specifica dei vincoli (ad esempio gli allestimenti dei suoi veicoli disponibili). • Attende le risposte degli agenti buyer per un certo lasso di tempo. • Riceve una o più risposte, che verranno esaminate dall'utente, che sceglierà a sua discrezione. • Invia le risposte ai buyer e aggiorna la lista delle merci prenotate.
Buyer agent	• Legge da file i dati di configurazione e si connette alla rete. • Legge da file i dati di supporto e riempie il set di merci (nel caso tale file non esistesse ne crea uno inizialmente vuoto). • Inserisce, modifica, visualizza ed elimina le merci, definendo così l'insieme delle merci. PUBBLICAZIONE SULLE PAGINE GIALLE • Contatta l'agente Yellow-pages chiedendo di iscriversi e autodefinendosi un Buyer. In tal modo potrà essere contattato dagli altri agenti in rete. SCENARIO "RICEVE CFP" • Riceve una CFP da un agente Shipper, quindi il sistema legge il messaggio e controlla i vincoli specificati. Se il controllo va a buon fine (shipper compatibile con i vincoli della merce), invia la lista delle merci allo shipper.

Tabella 1: tabella delle responsabilità

3.4 Interazioni e message template

Per ogni tipo di agente è necessario definire le interazioni con gli altri agenti ed i relativi protocolli di IP; a tal proposito, qualora nessuno dei protocolli FIPA risulti essere adeguato, è necessario definirne uno ad-hoc (nel progetto corrente non è stato necessario ricorrere alla ridefinizione dei protocolli).

Sulla base della Tabella 1 è possibile redigere un elenco delle interazioni di ogni agente comprendente il tipo di protocollo impiegato, il destinatario dell'interazione e le condizioni di attivazione.

Shipper Agent					
Interazione	**Responsabilità**	**IP**	**Ruolo**	**To**	**Condizione**
1-Recupera la lista di tutti i buyers disponibili	Chiede all'agente Yellow-pages la lista di tutti gli agenti Buyer disponibili in rete.	FIPA Request	Initiator	Yellow pages agent	On request
2-Invita i buyers a presentare le loro proposte	Invia dei messaggi (Call for Proposal) a tutti i possibili committenti (buyers) offrendosi come spedizioniere.	Contract Net	Initiator	Buyer agent	On user request
3-Risponde alla proposta di un buyer	Invia le risposte ai buyer e aggiorna la lista delle merci prenotate	Contract Net	Responder	Buyer agent	Propose received
Buyer Agent					
Interazione	**Responsabilità**	**IP**	**Ruolo**	**To**	**Condizione**
4-Risponde all'invito di uno shipper a presentare una proposta con le merci	Riceve una CFP da un agente Shipper, quindi sono controllati i vincoli specificati. Se il controllo va a buon fine (cioè se lo shipper è compatibile con la merce) invia la lista delle merci allo shipper.	Contract Net	Responder	Buyer agent	Propose received

Tabella 2: tabella interazioni

Come già illustrato, il compito (o task) che un agente deve eseguire, è generalmente svolto all'interno del behaviour dell'agente; note le responsabilità degli agenti (Tabella 1) e quella delle interazioni (Tabella 2), possiamo ora implementare i comportamenti degli agenti:

- per le interazioni 1 e 3 otteniamo la classe `SearchGoodInitiator` che estende la classe JADE `jade.proto.ContractNetInitiator`;
- per l'interazione 4 otteniamo la classe `SearchGoodResponder` che estende la classe JADE `jade.proto.ContractNetResponder`.

Ogni agente in JADE è caratterizzato da una sorta di mailbox dalla quale sono estratti i messaggi ricevuti da parte di altri agenti; sulla base di ciò, un agente estrarrà sempre il primo messaggio ricevuto facendo sì che la mailbox operi come una coda (struttura FIFO). In realtà è possibile ignorare alcuni tipi di messaggi ed estrarre e leggere solo quelli ai quali l'agente è realmente interessato.
Un *Message Template* è un filtro che permette ad un agente di estrarre dalla sua mailbox solo alcuni messaggi, ad esempio specificando il tipo di performative, l'ID della conversazione, il mittente ed altri campi.

4 Architettura software

4.1 Modellazione dei veicoli

Un primo oggetto del dominio applicativo che occorre modellare è l'insieme di classi che rappresentano i veicoli adibiti al trasporto; in virtù dell'analisi del dominio eseguita, la modellazione proposta è la seguente:

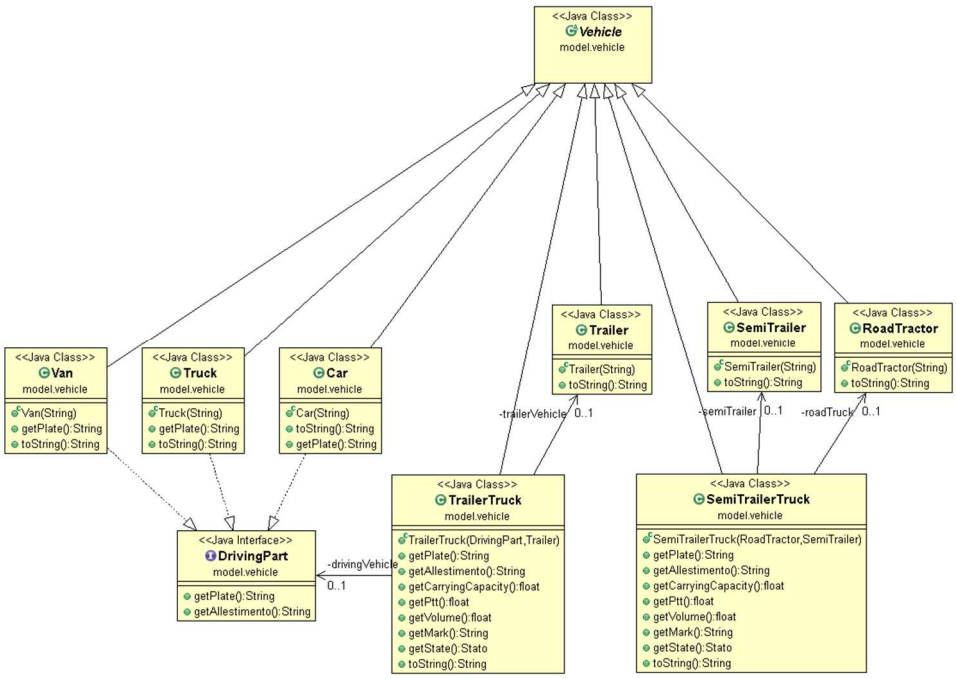

Figura 16: modellazione delle classi del veicolo

Tutte le classi ereditano la classe astratta `Vehicle`, la quale fornisce attributi di base quali: targa, marca, modello, peso, capacità di carico (detto anche portata massima), ptt (massa a pieno carico) misurati in tonnellate, volume (misurato in metri cubi), larghezza, lunghezza e altezza (misurate in metri), stato del veicolo (disponibile, occupato, in riparazione…), allestimenti e locazione attuale.

Dalla classe astratta `Vehicle` (veicolo) sono direttamente estese le classi `Car` (auto), `Van` (per van e furgoni), `Truck` (autocarro), `Trailer` (rimorchio), `RoadTractor` (trattore stradale), `SemiTrailer` (semirimorchio), `TrailerTruck` (autotreno) e `SemiTrailerTruck` (autoarticolato). Le classi `Car`, `Van` e `Truck` implementano l'interfaccia `DrivingPart` (parte trainante): tale interfaccia è necessaria per la composizione degli oggetti della classe `TrailerTruck`: per "realizzarlo" occorre un oggetto `Trailer` e un ulteriore oggetto la cui classe implementi l'interfaccia `DrivingPart` (cioè `Car`, `Van` o `Truck`). Infine, un oggetto di classe `SemiTrailerTruck` deve essere composto da un `SemiTrailer` e da un oggetto `RoadTractor`.

4.2 Modellazione delle merci e dei trasporti

È stata creata la classe `Goods` (merce) per modellare le seguenti informazioni: codice, descrizione, tipo di merce, dimensione, quantità, volume, pericolosità, locazione di partenza e destinazione, data di consegna e, infine, gli allestimenti necessari per essere trasportata da un veicolo. La classe `Transports` (che si compone della classe `Goods`) è stata creata al fine di modellare i luoghi di partenza e destinazione, la date di partenza ed il limite temporale entro cui dovrà essere trasportata.

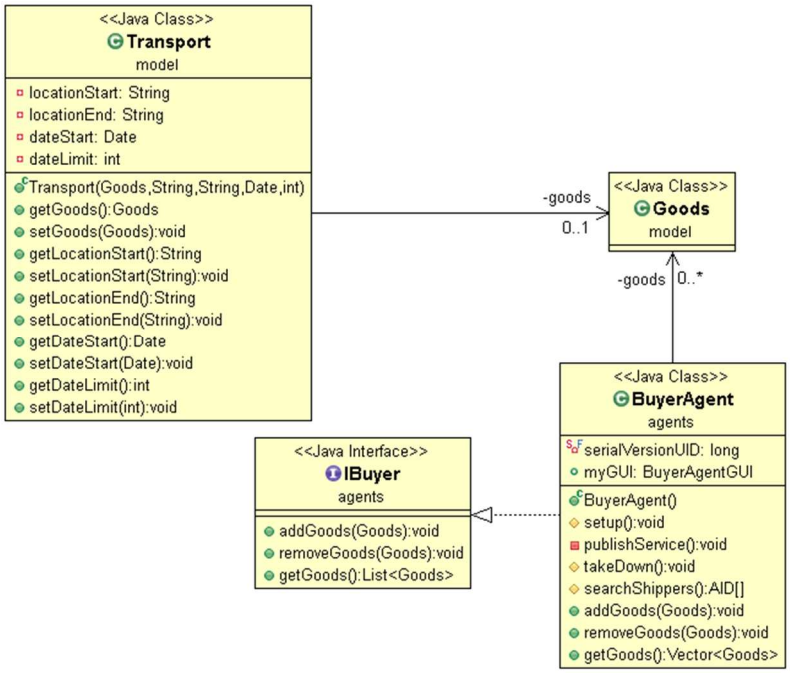

Figura 17: modellazione classe `Goods`, `Transport` e legami con `BuyerAgent`

La classe `BuyerAgent` si compone della classe `Goods` in quanto, quale fruitore del servizio di spedizioni, è colui che possiede determinati "beni" (`Goods`) che devono essere spediti.

4.3 Shipper e Buyer agent

`BuyerAgent`, che si compone della classe `Goods` (è ciò che deve movimentare), offre all'utente (attraverso la GUI implementata con la classe `BuyerAgentGUI`) la lista delle merci da spostare da un luogo ad un altro. L'insieme di attributi per ogni merce è ridotto, ma per questo caso di studio è sufficiente; all'avvio, l'agente si registra presso il servizio di pagine gialle ed avvia il behaviour ciclico realizzato dalla classe `SearchGoodResponder`. Come mostrato nella Figura 17, `BuyerAgent` si compone dei beni da spedire ed implementa l'interfaccia con i metodi associati alle gestione dei medesimi.

Per quel che concerne `ShipperAgent`, all'avvio è creata e visualizzata la relativa GUI che è composta da due schede: in quella "veicoli" sono mostrate all'utente la lista dei mezzi posseduti (o flotta) e tra questi,

quelli che sono disponibili/non disponibili ad effettuare una spedizione. Nella scheda “merci”, invece, sono mostrate all’utente l’elenco delle spedizioni prenotate.

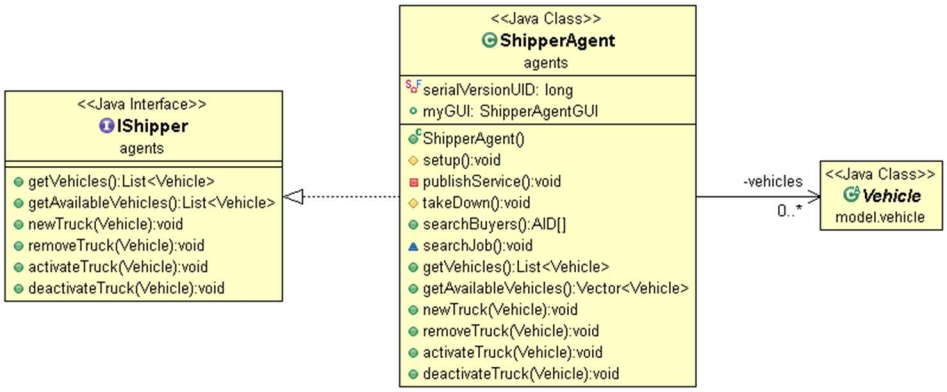

Figura 18: modellazione classe `Shipper`

Anche `ShipperAgent` si registra presso il servizio di pagine gialle; la GUI, inoltre prevede il tasto “Cerca Buyer” che permette di avviare il behaviour `SearchJobInitiator` che di fatto avvia la ricerca di potenziali clienti aventi merci da spedire (la spiegazione del protocollo avverrà in seguito).

4.4 Protocolli di Interazione (IP) impiegati

Gli agenti Shipper e Buyer sono liberi di comunicare e di cominciare la contrattazione in ogni momento; ogni agente, indipendentemente dal tipo, all’avvio si collega automaticamente alla rete e si registra presso il servizio di pagine gialle (`publishService()`). In tal modo ogni altro agente è consapevole della sua esistenza ed è in grado di determinare l’ ubicazione ed i servizi che offre.

In questa versione è stata implementata una versione del Protocollo d’Interazione *FIPA ContractNet*.

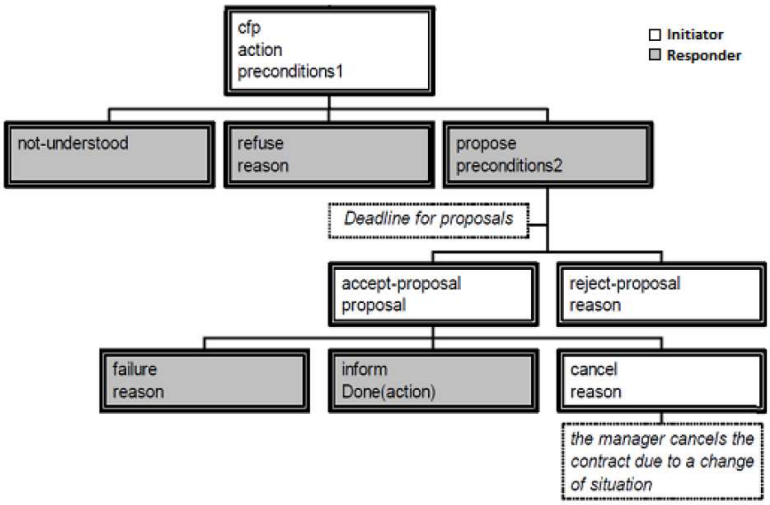

Figura 19: FIPA ContractNet

Il protocollo è stato implementato lato spedizioniere (shipper) dalla classe `SearchGoodInitiator` e lato contractor (buyer) da `SearchGoodResponder`, che estendono rispettivamente le classi JADE `ContractNetInitiator` e `ContractNetResponder`. Di seguito si riporta il funzionamento del protocollo:

1. `BuyerAgent` avvia automaticamente un behaviour attuato dalla classe `SearchGoodResponder` ed attende di ricevere CFP da parte di uno o più `ShipperAgent` (anche contemporaneamente);
2. alla pressione del pulsante "Cerca Buyer" da parte dell'utente, `ShipperAgent` avvierà il behaviour `SearchGoodInitiator`, quindi invierà una Call For Proposal (CFP) a tutti gli agenti di tipo `BuyerAgent` attivi che troverà registrati presso il servizio di Pagine Gialle;
3. `BuyerAgent` che ha ricevuto una CFP si comporta come segue:
   a. se la propria lista delle merci da inviare è vuota, invierà come risposta una REFUSE;
   b. se nella lista risiedono una o più merci, invierà come risposta una PROPOSE.

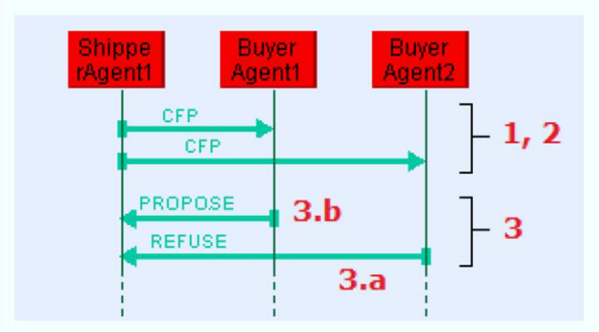

Figura 20: schermata tratta dall'agente Sniffer[4] che mostra le fasi 1,2,3 del protocollo.

4. `ShipperAgent` che ha ricevuto una risposta attua i seguenti comportamenti:
   a. se ottiene REFUSE, termina la comunicazione con quel `BuyerAgent`;
   b. se riceve PROPOSE, ottiene una lista di merci che l'utente visiona al fine di scegliere quelle che vuole assegnare allo spedizioniere.

[4] Si tratta di uno strumento fornito da JADE che permette di analizzare le interazioni di un agente o di un gruppo di agenti attraverso una GUI specifica.

Figura 21: Propose di un agente buyer verso un agente shipper.

5. Dopo la scelta dell'utente, viene inviata la risposta al `BuyerAgent`:
   a. se l'utente preme “Annulla”, oppure preme “Esegui” ma non ha selezionato alcuna merce, viene inviato al `BuyerAgent` una REJECT-PROPOSAL;
   b. se l'utente ha selezionato una o più merci e preme “Esegui”, viene inviato al `BuyerAgent` un ACCEPT-PROPOSAL;

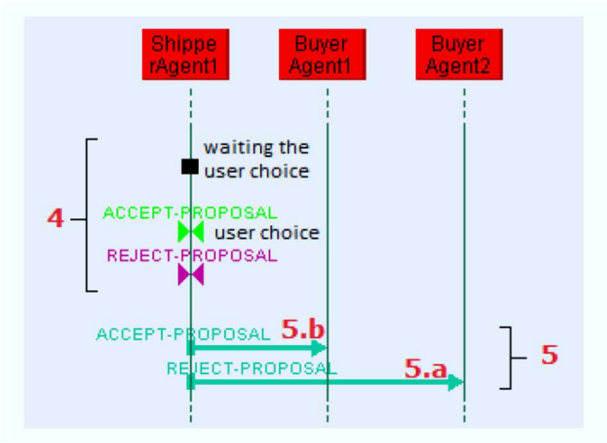

Figura 22: fasi 4 e 5 del protocollo.

6. Il `BuyerAgent` ha ricevuto la risposta:
   a. se riceve REJECT-PROPOSAL, la comunicazione con quello `ShipperAgent` termina;
   b. se riceve ACCEPT-PROPOSAL, il `BuyerAgent` controlla che tali merci siano ancora disponibili (per esempio alcune di esse nel frattempo potrebbero essere state "prenotate" da un altro Shipper).
7. L'algoritmo può fornire i seguenti risultati:
   a. se le merci nella precedente PROPOSE sono ancora tutte disponibili, `BuyerAgent` invia una INFORM allo `ShipperAgent`; quindi la contrattazione ha avuto successo e "prenota" tali merci per lo `ShipperAgent` ed interrompe l'interazione con esso.
   b. se ci sono state variazioni riguardo lo stato delle merci, il `BuyerAgent` invia:
      - una FAILURE nel caso in cui non ci siano merci disponibili;
      - una nuova PROPOSE nel caso ci siano altre merci (o un sottogruppo della propose precedente), allo scopo di informare lo `ShipperAgent` della variazione; in questo caso si riparte dal punto 4.

8. `ShipperAgent` può ottenere le seguenti risposte:
   a. FAILURE: la comunicazione con quel `BuyerAgent` termina;
   b. PROPOSE: si torna al punto 4b;
   c. INFORM: la contrattazione con quel Buyer ha avuto successo e le merci sono state "prenotate" dallo Shipper.

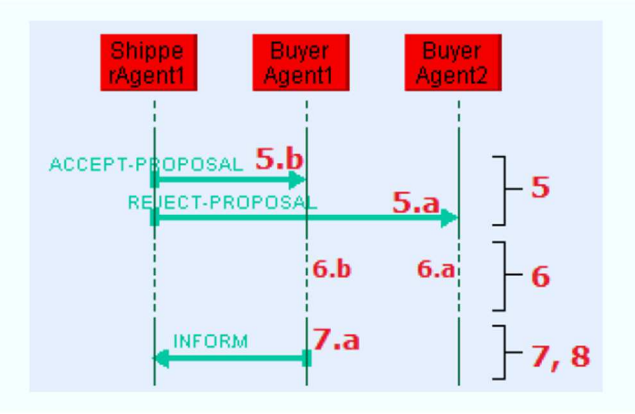

Figura 23: fasi 5,6,7 del protocollo

5 Commenti e sviluppi futuri

Come notato è stato implementato il caso in cui uno spedizioniere ricerca clienti, ossia alcuni buyer di servizi; in futuro, come ampliamento, sarebbe auspicabile aggiungere un protocollo che si occupi del caso contrario, ossia un `BuyerAgent`, che ricerchi servizi di spedizione delle proprie merci (al momento il sistema determina l'elenco degli agenti di tipo shipper attivi).

Altro punto di miglioramento riguarda il protocollo d'Interazione tramite il metodo `matchingGoodsAndVehicles()` appartenente alla classe `SearchGoodResponder`, che alla versione attuale, esegue un match solo sulla base degli allestimenti dei veicoli e degli allestimenti richiesti dalle merci; da questo punto di vista sarebbe, altresì, auspicabile inserire altri attributi utilizzabili per effettuare la selezione: ad esempio si potrebbero ulteriori criteri quali la quantità di merce già presa in carico da un veicolo e quella disponibile, le dimensioni, le tipologie di merce e le date in cui un mezzo è occupato; questo significa, in primo luogo, introdurre lo stato di carico di un veicolo.

Un'altra possibile miglioria riguarda il caricamento dei dati (veicoli per un agente shipper, e merci per un agente Buyer) da file xml; occorrerebbe aggiungere la possibilità di salvare le modifiche eseguite anche mentre il software è in uso.

Ovviamente il progetto è da considerarsi in fase beta e quindi necessita ancora di una fase di refactoring e testing più approfondito.

Appendice A – Guida Utente

Per avviare la simulazione del progetto bisogna posizionarsi nel package `startAgents` e porre in esecuzione la classe `StartSystem`; a questo punto viene avviato il Main Container e si apre una GUI che permette di avviare alcuni agenti quali RMA, Sniffer, Introspector, oltre agli agenti Buyer e Shipper; questi ultimi, all'avvio verificano se esiste un file .xml (avente lo stesso nome dell'agente) contenente l'elenco dei veicoli (nel caso dello shipper) o l'elenco delle merci (nel caso del buyer); in caso affermativo, un parser provvede a caricare la flotta (`ShipperXmlParser`) e l'elenco delle merci (`BuyerXmlParser`) a ciascun agente (elenco visualizzabile nella rispettiva GUI); qualora il file .xml non sia disponibile, è possibile creare manualmente l'elenco dei veicoli e delle merci per ciascun agente creato.

L'avvio della ricerca di possibili buyer avviene attraverso il tasto "Cerca Buyer", mentre la proposta della selezione delle merci da affidare allo shipper, avviene tramite il tasto "Esegui".

Bibiliografia

[1] Slide del corso Sistemi Distribuiti 2016/2017.

[2] Omicini, Andrea, et al. *"Multi-agent infrastructures for objective and subjective coordination." Applied Artificial Intelligence, 2010*

[3] G. Caire (TILAB, formerly CSELT), *JADE Programming For Beginners*, 2009

[4] F. Bellifemine, G. Caire, T. Trucco (TILAB, formerly CSELT), G. Rimassa (University of Parma), *JADE Programmer's Guide (2010)*

[5] F. Bellifemine, G. Caire, D. Greenwood, *Developing Multi-Agent Systems with JADE*, John Wiley & Sons, Ltd, 2007.