Implementazione FIPA Dutch Auction Interaction Protocol in JADE

MAS_07:Implementazione di “Auction Protocols” in JADE

Progetto Sistemi Operativi 2

Corso di Laurea Magistrale in Ingegneria Informatica e dell’Automazione

Anno accademico 2015/2016

Università Politecnica delle Marche

Barjam Bardhi matricola 1070591

Michele Di Carlo matricola 1068146

***Abstract*— Il presente documento descrive una possibile implementazione in JADE (Java Agent Development Framework) del protocollo di interazione d'asta Olandese secondo FIPA (Foundation for Intelligent Physical Agents).**

Keywords*— *FIPA Dutch Auction Interaction Protocol; Agent Oriented Programming; JADE; Behaviour.

I. INTRODUZIONE

Il termine "asta olandese" deriva dai mercati dei fiori in Olanda, dove si utilizzava tale metodo al fine di determinare il valore di mercato dei fiori. L'asta olandese proprio per il modo di operare, in cui inizialmente l'iniziatore (anche detto banditore) propone un bene ad un prezzo più elevato rispetto a quello di mercato, abbassandolo progressivamente, viene definita "asta al ribasso". Al contrario dell'asta inglese dove invece il banditore parte da un prezzo più basso rispetto al valore reale e provvede ad aumentarlo, appartenente alla categoria delle "aste al rialzo". Questo articolo si occupa del protocollo di interazione d'asta Olandese secondo FIPA, non ancora entrato a far parte degli standard, ma di tipo "Experimental", il quale prevede la presenza di due tipologie di agenti: iniziatore e partecipante. L'iniziatore, come si può facilmente intuire dal nome è l'agente che da inizio al protocollo tramite l'invio di un messaggio (inform-start-of-auction nella Fig.1) a ciascun partecipante attraverso il quale li porta a conoscenza dell'intenzione di voler iniziare una nuova asta. A cui fa seguito l'invio di un altro messaggio (cfp-1 nella Fig.1) sempre da parte dell'iniziatore e destinato a tutti i partecipanti, in cui avanza la propria proposta d'asta, la quale include il nome e il rispettivo prezzo del bene. Trattandosi della prima proposta, il prezzo viene detto prezzo iniziale. A questo punto i partecipanti che sono intenzionati all'acquisto del bene al prezzo ricevuto risponderanno con un apposito messaggio (propose nella Fig.1) indicando la propria disponibilità ad acquistare. Mentre nel caso in cui non siano interessati in quanto il prezzo risulta essere troppo elevato non inviano messaggi al banditore. Se l'iniziatore ha ricevuto la proposta da un solo partecipante risponderà con un messaggio di conferma (accept-proposal nella Fig.1), attraverso il quale comunica al partecipante che ha vinto l'asta. Mentre se in corrispondenza a un'offerta dell'iniziatore fanno seguito più risposte da parte dei diversi partecipanti, allora il prodotto viene venduto solo al partecipante che ha comunicato per primo all'iniziatore la sua intenzione d'acquisto (cioè utilizza la politica FCFS per stabilire la precedenza), al quale viene inviato un messaggio di conferma (accept-proposal nella Fig.1), mentre ai restanti partecipanti viene comunicata l'impossibilità di procedere all'acquisto tramite un apposito messaggio (reject-proposal nella Fig.1). Nel caso in cui l'iniziatore non ha ricevuto almeno una risposta e non ha raggiunto il prezzo di riserva (cioè il minimo prezzo a cui è disposto a vendere il bene oggetto dell'asta), ha luogo una nuova iterazione, in cui invia un messaggio (cfp-2 nella Fig.1) a ciascun partecipante con il prezzo del prodotto decrementato, di una quantità decisa dall'iniziatore stesso. L'iterazione appena descritta viene eseguita fino a che non si trova nella situazione in cui l'iniziatore non ha ricevuto risposte ed ha raggiunto il prezzo di riserva o è riuscito a vendere, in questi casi il banditore invia agli agenti un apposito messaggio ([no-bids]inform-2 nella

Fig.1) in cui comunica la terminazione dell'asta. Nella Fig.1 viene riportato il protocollo appena descritto.

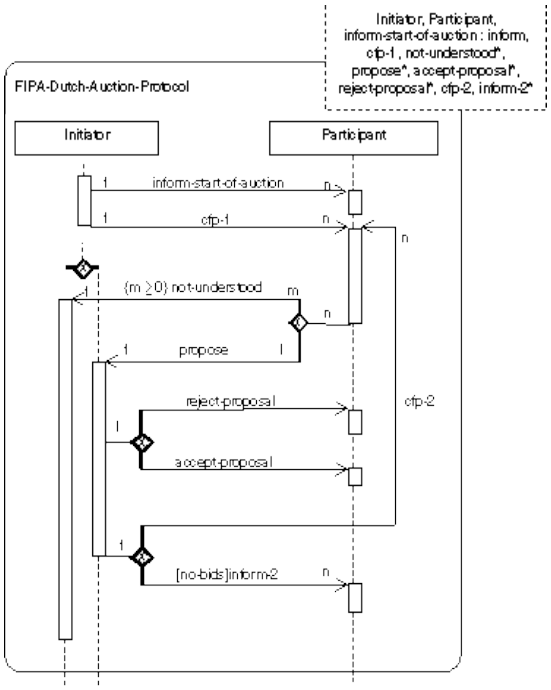

Fig.1 FIPA Dutch Auction Protocol

II. IMPLEMENTAZIONE

A. Soluzioni implementate

Prima di iniziare ad implementare il protocollo così come descritto da FIPA, sono state cercate possibili evoluzioni. A seguito della lettura di diversi articoli e documentazioni, si è giunti alla conclusione che il protocollo non ha subito modifiche significative, per cui è stato deciso di implementare la versione del protocollo "Experimental" FIPA, corrispondente a quello riportato nella Fig.1. Nello specifico è stato scelto di realizzare la versione del protocollo in cui vengono prese in considerazione anche le quantità, cioè in cui i partecipanti possono acquistare anche solo una porzione del bene messo all'asta. È stata effettuata tale scelta in quanto anche il caso di aste relative a beni unitari può essere ricondotto a questo imponendo quantità pari a uno. Per quanto riguarda lo sviluppo sono stati realizzati due progetti distinti, tra cui il primo che prevede la possibilità di avere un solo iniziatore e più partecipanti e il secondo che prevede la possibilità oltre che di avere molti partecipanti di avere più iniziatori, con la capacità di mettere all'asta beni uguali o diversi, con diverse: quantità, decremento del prezzo in caso di assenza di offerte e tempi di attesa.

B. Strumenti utilizzati

Per l'implementazione del protocollo i principali strumenti utilizzati, sono:

- il linguaggio di programmazione **Java**;
- un framework per la programmazione ad agenti **JADE** (Java Agent Development Framework), si tratta di una piattaforma ad agenti sviluppata completamente in Java, che rispetta gli standard definiti da FIPA. In particolare al fine dell'implementazione sono state utilizzate due librerie di JADE: "jade.jar" e "commons-codec-1.3.jar";
- l'ambiente di sviluppo integrato **Eclipse;**
- libreria "json-simple-1.1.1.jar", non indispensabile al fini del funzionamento del protocollo di interazione, ma sfruttata per il salvataggio e il caricamento delle configurazioni delle simulazioni.

C. Organizzazione del progetto

Come accennato in precedenza sono stati realizzati due progetti distinti, che risultano essere organizzati allo stesso modo per quanto riguarda la suddivisione in package e le librerie incluse. I file sorgenti di entrambe i progetti sono suddivisi in **5 package** (o cartelle):

- **agents**: contiene i sorgenti relativi alle due classi degli agenti dell'asta, Initiator e Participant, entrambe estendono la classe Agent, presente all'interno del package "jade.core" della libreria "jade.jar". All'interno delle classi Initiator e Participant sono presenti attributi e metodi utili al fine della simulazione. Particolare importanza riveste il metodo setup(), invocato al momento della creazione degli ogetti della classe Agente, all'interno del quale è stata implementata oltre l'inizializzazione dei valori degli attributi, l'aggiunta dell'apposito behaviour.
- **behaviours**: include i sorgenti relativi ai comportamenti degli agenti, in particolare si tratta di classi che estendono la classe Behaviour, contenuta all'interno del package "jade.core.behaviours" della libreria "jade.jar". In particolare le classi che estendono la classe Behaviour implementano i metodi done() e action(), quest'ultimo metodo viene eseguito continuamente fino a che il metodo done() torna false.
- **gui**: all'interno di questo package sono presenti tutte le classi utilizzate per implementare l'interfaccia grafica. A tale proposito sono state sfruttate principalmente le classi del package Swing (javax.swing) messo a disposizione da JAVA.
- **ontology**: comprende tutte le classi che implementano Concetti e Azioni dell'ontologia oltre che l'ontologia stessa utilizzata nel progetto. Tutte le classi di questo package implementano le interfacce contenute nel package "jade.content".
- **util**: racchiude le classi degli agenti utilizzate al fine di contenere i parametri della configurazione della simulazione, sfruttate in particolare nella GUI.

III. BEHAVIOUR INIZIATORE

La classe "BehaviourInitiator.java" inclusa nel package "behaviour" del progetto, modella il comportamento dell'iniziatore dell'asta, che risulta essere essenzialmente la

stessa in entrambe i progetti. BehaviourInitiator estende la classe Behaviour, della quale sovrascrive i metodi done() e action(). Al fine di rispettare il protocollo dell'asta olandese FIPA, il comportamento dell'iniziatore è stato modellato come una macchina a stati finiti, a tal proposito all'inizio della classe BehaviourInitiator sono state dichiarate e inizializzate le costanti che rappresentano i possibili stati. Il metodo done() è implementato in maniera tale da restituire "true", portando quindi alla terminazione del behaviour, solo se l'agente ha raggiunto lo stato finale END. Mentre nel metodo action() che è quello eseguito ogni volta fino a quando il metodo done() torna false, è modellata la parte del protocollo relativa all'iniziatore dell'asta. Il metodo action() è composta da un costrutto di tipo switch-case che valuta lo stato dell'agente banditore, in base al quale esegue un apposito blocco di istruzioni. È importante notare che lo stato non è un attributo del behaviour, ma dell'agente, che nel caso in esame è un istanza della classe Initiator del package agents. Di seguito sono illustrati i possibili stati:

- **INFORM_START_OF_AUCTION**: è lo stato in cui l'agente di tipo iniziatore si trova appena viene creato. Quando si trova in questo stato e viene eseguito il metodo action(), l'iniziatore crea un messaggio al fine di informare tutti i partecipanti dell'intenzione di iniziare una nuova asta. Viene creato un messaggio con performative INFORM e contenuto "start of auction", aggiungendo come destinatari tutti gli eventuali agenti di tipo partecipante presenti, procedendo poi all'invio del messaggio e aggiornando infine lo stato dell'agente a CALL_FOR_PROPOSAL. Quanto descritto avviene solo nel caso in cui sia presente almeno un partecipante, mentre se non ci sono partecipanti l'asta termina e lo stato dell'iniziatore viene posto a END.
- **CALL_FOR_PROPOSAL_1**: quando l'iniziatore si trova in questo stato ed è presente almeno un partecipante, procede alla creazione del messaggio con performative CFP, nel quale viene anche inserito il bene oggetto della nuova asta sfruttando le ontologie, in particolare l'azione NewGood, che sta a significare appunto una nuova asta relativa a un certo bene. Nell'oggetto di tipo NewGood sono contenuti tutti i parametri dell'asta, quali: il nome del bene, il prezzo e la quantità. Allo stesso messaggio sono aggiunti tutti gli identificativi dei partecipanti (AID), segue l'invio del messaggio e l'aggiornamento dello stato a WAITING_MESSAGE. Mentre se non ci sono agenti disponibili l'asta termina e lo stato dell'agente diventa END.
- **WAITING_MESSAGE**: quando l'agente si trova in questo stato, aspetta un certo numero di millisecondi (tale tempo di attesa è scelto dall'utente) prima di leggere i messaggi e proseguire l'esecuzione. In seguito all'attesa controlla se ha ricevuto risposte o no. Nel caso in cui non ha ricevuto risposte, significa che non ha ricevuto offerte, allora aggiorna il prezzo, abbassandolo della quantità prestabilita (decisa dall'utente) solo se maggiore al prezzo di riserva, aggiornando lo stato a CALL_FOR_PROPOSAL_2. Se invece è stato raggiunto il prezzo di riserva lo stato viene aggiornato a INFORM_2. Altrimenti se ha ricevuto messaggi e quindi offerte, procede leggendo in ordine FCFS i messaggi, che nel caso di offerte contengono oggetti dell'ontologia IBuy. Effettua un controllo sulla quantità richiesta dal partecipante, che non può superare quella disponibile, se ciò è rispettato viene comunicato al corrispondente agente partecipante, mediante un apposito messaggio con performative "ACCEPT_PROPOSAL", che ha vinto l'asta. Fanno seguito l'aggiornamento della quantità del bene disponibile e la rimozione di tale partecipante dalla lista degli agenti partecipanti all'asta. Nel caso in cui la quantità richiesta è maggiore della disponibile, allora al partecipante viene inviato un messaggio con performative "REJECT_PROPOSAL" al fine di comunicargli che non si è aggiudicato il bene. Viene poi aggiornato lo stato in base alla disponibilità del bene: se il bene è stato completamente venduto lo stato diventa INFORM_2, altrimenti lo stato passa a WAITING_MESSAGE. Cioè rimane nello stesso stato poichè nel caso in cui siano stati ricevuti più messaggi, quindi più offerte, tutti i messaggi dei partecipanti sono processati (con politica FCFS) e vengono soddisfatte tutte le richieste fintanto che c'è sufficiente quantità del bene.
- **CALL_FOR_PROPOSAL_2**: se il prezzo attuale è maggiore o uguale al prezzo di riserva e ci sono agenti che partecipano all'asta, crea un messaggio da inviare con l'opportuna performative CFP e contenuto "call for proposal 2", inserendo nel messaggio un oggetto dell'ontologia NewPrice, contenente: nome, prezzo e quantità del bene oggetto dell'asta, tale ontologia sta a significare un ribasso del prezzo dell'asta. Allo stesso messaggio sono aggiunti tutti i destinatari, corrispondenti a tutti i partecipanti, seguono l'invio e l'aggiornamento dello stato a WAITING_MESSAGE. Nel caso in cui non ci sono agenti lo stato viene direttamente posto a END, mentre se è stato raggiunto il prezzo di riserva a INFORM_2.
- **INFORM_2**: mette al corrente gli eventuali partecipanti che l'asta è terminata in quanto è stato raggiunto il prezzo di riserva o il bene è stato completamente venduto. A questo scopo invia un messaggio con performative INFORM e contenuto "inform 2" e aggiorna lo stato a END.
- **END**: rappresenta lo stato finale, cioè quando l'agente termina, il metodo done() ritorna quindi true.

Tutti i messaggi inviati dall'iniziatore ai partecipanti, sono contraddistinti da un "conversationID", si tratta di una stringa che identifica univocamente i messaggi al fine di permettere ai partecipanti di distinguere le diverse aste in corso.

IV. CLASSE INITIATOR

All'interno della classe Initiator del package "agents", vi sono gli attributi relativi all'agente di tipo iniziatore, particolare importanza riveste l'attributo "state", che modella lo stato dell'agente, inizializzato a INFORM_START_OF_AUCTION,

che corrisponde allo stato iniziale, sono inoltre definiti le caratteristiche del bene oggetto all'asta (nome, quantità, prezzo attuale, prezzo di riserva), il ribasso ad ogni step e il tempo di attesa. Oltre a un ArrayList relativo ai partecipanti all'asta, che viene popolato tramite l'invocazione del metodo findParticipants, presente all'interno della classe stessa.

V. BEHAVIOUR PARTECIPANTE

La classe "BehaviourParticipant.java" inclusa nel package "behaviour" del progetto con un solo iniziatore modella il comportamento degli agenti partecipanti all'asta. BehaviourParticipant estende la classe Behaviour, della quale sovrascrive i metodi done() e action(). Il metodo done() torna true (quindi l'esecuzione del behaviour termina), quando la variabile stop all'interno del behaviour assume valore true. Il flusso d'esecuzione all'interno del metodo action() si basa sulle differenti caratteristiche dei messaggi ricevuti. Inizialmente si aspetta un messaggio con performative INFORM e contenuto "start of auction", eventuali messaggi diversi sono ignorati per effetto del matching basato sui MessageTemplate. A seguito del primo messaggio come appena descritto, il MessageTemplate viene modificato in maniera tale da ricevere solo messaggi dall'iniziatore, sfruttando il matching basato sul conversationID. Tra i messaggi che il partecipanti può ricevere in questa fase, vi sono:

- messaggio con contenuto "inform 2" il quale indica che l'asta termina per cui l'attributo stop del behaviour viene posto a true, ciò vale per il caso in cui si ha un solo iniziatore. Mentre nel caso multi iniziatore viene posto a true esclusivamente l'attributo stopBehaviour del behaviour corrispondente all'asta terminata);
- messaggio con performative REJECT_PROPOSAL, singnifica che il partecipante non è riuscito ad acquistare pertanto continua la sua esecuzione;
- se non corrisponde ai precedenti tipi allora viene estratto il contenuto, in particolare l'ontologia ricevuta, se si tratta di NewGood, cioè una nuova asta, o NewPrice, cioè il ribasso del prezzo dell'asta. Il comportamento che ne segue è uguale per entrambe i casi, verifica se il prezzo ricevuto è minore o uguale a quello a cui è disposto ad effettuare l'acquisto, rispondendo all'iniziatore con un apposito messaggio con performative PROPOSE e contenente l'ontologia IBuy, la quale include nome, prezzo che è disposto a pagare e quantità, quest'ultima adattata in base alla disponibilità dell'iniziatore.
- se il messaggio ricevuto contiene un ontologia di tipo YouBuy significa che il partecipante ha vinto l'asta, quindi è riuscito nell'intento dell'acquisto, pertanto alla variabile stop viene assegnato il valore true.

Nel progetto che prevede la presenza di più iniziatori il behaviour del partecipante risulta essere leggermente diverso. In quest'ultimo caso sono presenti due diversi behaviour:

- CyclicBehaviourParticipant: che è l'unico che viene aggiunto quando l'agente viene inizializzato ed avviato, il quale esegue il metodo action() in continuazione. Attraverso il meccanismo di matching dei MessageTemplate, è predisposto a ricevere solo messaggi con performative INFORM e contenuto "start of auction". Ogni volta che viene ricevuto un messaggio come appena definito (inviato quindi da un iniziatore), viene fatto partire un behaviour del tipo BehaviourParticipant al quale come parametri sono passati l'agente partecipante e il conversationID del messaggio ricevuto. Pertanto si avranno per ogni partecipante tanti BehaviourParticipant quanti sono gli iniziatori, ciascun behaviour riceverà i messaggi di un unico iniziatore grazie al matching in base al conversationID basato sull'uso dei MessageTemplate.
- BehaviourParticipant: praticamente uguale a quello descritto per il progetto con singolo iniziatore, eccetto la ricezione del messaggio con performative INFORM e contenuto "start of auction", non è prevista in questo behaviour, in quanto viene gestita dal CyclicBehaviourParticipant.

Da notare è che in questo caso i behaviour di tipo BehaviourParticipant terminano individualmente quando la variabile stopBehaviour interna al behaviour assume il valore true, mentre terminano tutti nel caso in cui cui la variabile stop dell'agente partecipante ha valore true.

V. INTERFACCIA GRAFICA

Al fine di poter garantire una maggiore usabilità è stato scelto di implementare un'apposita GUI. In entrambe i progetti lanciando l'applicazione eseguendo il metodo main presente nella classe FrameMain del package "gui" si apre la finestra principale del simulatore di aste implementato. Nel caso del progetto che prevede un singolo iniziatore, a seguito dell'esecuzione si apre la schermata di Fig.2, mentre nel progetto che prevede più esecutori, si apre la schermata di Fig.3. Tramite la GUI è possibile:

- inserire il numero degli iniziatori (solo nel caso del progetto con più iniziatori)
- inserire il nome del bene dell'iniziatore
- inserire il prezzo di partenza dell'iniziatore
- inserire il prezzo di riserva dell'iniziatore
- inserire la quantità del bene dell'iniziatore
- inserire il ribasso dell'iniziatore
- inserire il tempo di attesa per le offerte dell'iniziatore
- inserire il numero di partecipanti
- inserire il prezzo massimo che è disposto a pagare ciascun partecipante
- inserire la quantità di bene desiderata da ciascun partecipante
- inserire il nome del bene di interesse del partecipante (solo nel caso del progetto con più iniziatori)
- avviare la simulazione
- avviare gli agenti
- scegliere di avviare lo sniffer GUI (Fig.5)
- scegliere di avviare il Remote Agent Management GUI (Fig.5)
- visualizzare lo stato dell'esecuzione della simulazione
- creare una nuova simulazione (Fig.4)
- salvare i parametri di una simulazione (Fig.4)
- caricare i parametri di una simulazione (Fig.4)

- uscire dalla simulazione (Fig.4)

VI. CONCLUSIONI

Tramite questi due progetti si è raggiunto l'obbiettivo di simulare il protocollo d'interazione d'asta olandese secondo FIPA, nel caso di più iniziatori e più partecipanti, con la possibilità di prendere in considerazione le quantità, eliminando quindi il vincolo di beni unitari. Grazie allo sviluppo dell'interfaccia grafica è stata raggiunta un'ottima usabilità. In particolare è offerta la possibilità di salvare le configurazioni della simulazione, per poterla riprodurre in futuro semplicemente, grazie al caricamento, che evita di doverne impostare ogni volta una nuova. Il salvataggio e il caricamento si basano su file di tipo JSON. Risulta essere anche molto pratica la possibilità di avviare l'agente di tipo sniffer (tramite GUI), con già automaticamente assegnati tutti gli agenti di tipo iniziatore e partecipante. Ciò è reso possibile dal passaggio come argomento all'atto della creazione dello sniffer una stringa contenente la concatenazione dei nomi degli agenti. Una limitazione imposta nel caso del progetto che prevede la possibilità di avere più iniziatori è quella di dare la possibilità ad un partecipante di mandare una sola propose alla volta, al fine di evitare acquisti in esubero dovuti all'eventuale risposta con messaggio ACCEPT PROPOSAL da parte di più iniziatori che offrono lo stesso bene. Al fine di evitare questo inconveniente è stata inserita un apposita variabile booleana "propose" nella classe Participant del package "agents", oltre ad appositi controlli sul valore della nuova variabile e la quantità residua richiesta dal partecipante. Un'altra scelta implementativa riguarda il comportamento dei partecipanti, realizzato in modo di acquistare quanto possibile il bene anche se in quantità inferiore a quanto complessivamente il partecipante ne richiede, adattando cioè la quantità richiesta in base all'offerta (chiaramente solo se a prezzo inferiore o pari al massimo previsto dal partecipante). Infine i partecipanti hanno la possibilità di acquistare porzioni rispetto alla quantità complessiva richiesta da diversi banditori.

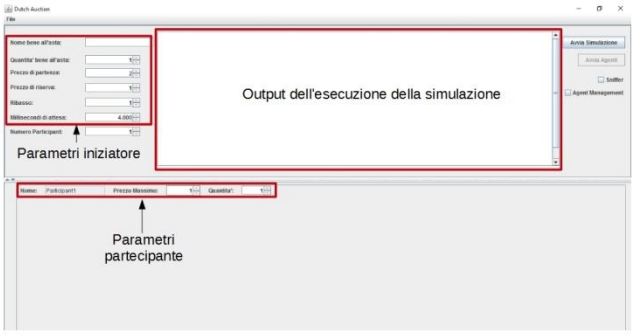

Fig.2 GUI progetto singolo iniziatore.

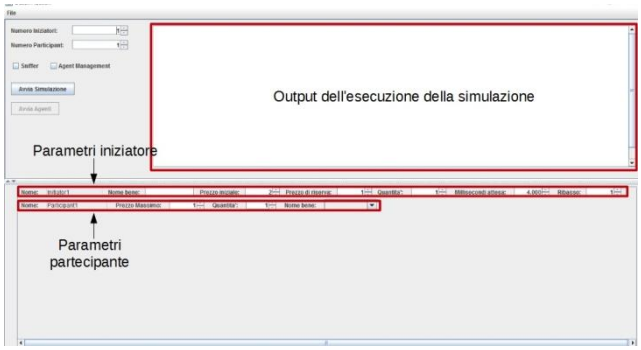

Fig.3 GUI progetto multi iniziatore

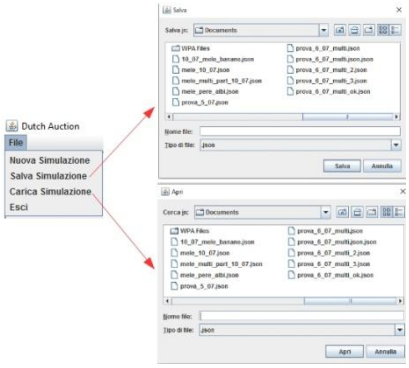

Fig.4 Menù File GUI

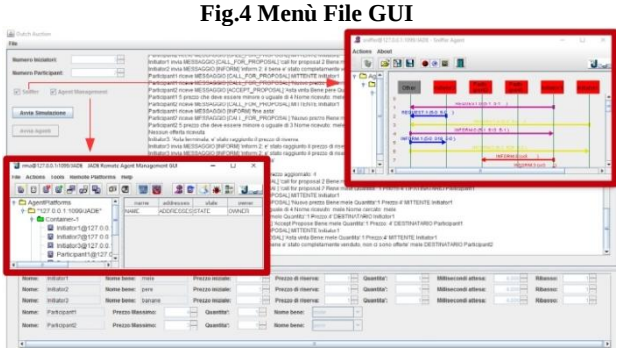

Fig.5 GUI sniffer e agent management

RIFERIMENTI

[1] F. Bellifemine, G. Caire, T. Trucco (TILAB), G. Rimassa (Università di Parma). "JADE PROGRAMMER'S GUIDE", http://jade.tilab.com/doc/programmersguide.pdf

[2] G. Caire (TILAB), "JADE TUTORIAL: JADE PROGRAMMING FOR BEGINNERS". http://jade.tilab.com/doc/tutorials/JADEProgramming-Tutorial-for-beginners.pdf

[3] FIPA,"FIPA Dutch Auction Interaction Protocol Specification", http://www.fipa.org/specs/fipa00032/

[4] JADE, http://jade.tilab.com/

[5] M. D. De Assunção, R. Buyya, "An Evaluation of Communication Demand of Auction Protocols in Grid Environments", 2006

[6] C. Bădică, M. Ganzha, M. Gawinecki, P. Kobzdej, M. Paprzycki, "Utilizing Dutch Auction in an Agent-based Model E-commerce System", proceedings of world academy of science, engineering and technology volume 14 agosto 2006.

[7] C. Bartolini, C. Preist, N. R. Jennings, "A Software Framework for Automated Negotiation", 17 Febbraio 2016

[8] O. Cliffe, M. De Vos, J. Padget, "Embedding Landmarks and Scenes in a Computational Model of Institutions", 2008