

IMPLEMENTAZIONE DI UN'INTERFACCIA

TRA

TU**P**ROLOG

E

JADE

DI

CHRISTIAN TARABUSI

Introduzione

Il fulcro del mio lavoro è stato quello di creare un'implementazione Java che permetta l'utilizzo delle funzionalità di tuProlog all'interno del Framework di JADE 3.0b.

I sistemi

Per prima cosa devo occuparmi dell'introdurre i sistemi da me studiati e utilizzati.

La definizione data dagli stessi autori di tuProlog è:

“tuProlog is a Java-based light-weight Prolog for Internet applications and infrastructures. For this purpose, tuProlog has been designed to be easy deployable light-weight, dynamically configurable, easily interoperable and built upon Java”.

Come si nota un fattore importante è il fatto che è completamente costruito in Java. Grazie a ciò utilizzare degli elementi di programmazione logica all'interno di un nostro qualunque programma Java è estremamente semplice. Si tratta solamente di importare i package relativi a tuProlog ed utilizzare le classi messe a disposizione.

Il secondo sistema chiamato JADE è così definito dai creatori:

“JADE is a software framework to develop agent applications in compliance with the FIPA specifications for interoperable intelligent multi-agent systems. The goal is to simplify development while ensuring standard compliance through a comprehensive set of system services and agents. JADE can then be considered an agent middle-ware that implements an Agent Platform and a development framework”.

Anche questo sistema è Java-complaint poiché interamente scritto in questo linguaggio. Grazie a JADE si possono creare diverse piattaforme su diverse macchine remote. In ogni piattaforma si possono creare dei *contenitori* che vengono considerati indipendenti e all'interno di questi contenitori vengono fatti agire i nostri agenti scritti in Java. Questi agenti possono ovviamente comunicare tra loro attraverso messaggi compatibili con le specifiche FIPA ACL.

JADE fornisce strumenti di alto livello per queste comunicazioni. Inoltre vi sono alcuni tool estremamente utili. Il primo da me utilizzato è il *Dummy Agent*, il quale permette l'invio e la ricezione di messaggi ai vari agenti presenti nelle varie piattaforme.

Il secondo è lo *Sniffer Agent* che permette l'ascolto dei messaggi che viaggiano nella piattaforma e una loro visualizzazione temporale.

Obiettivi

Viste le principali caratteristiche dei due sistemi risulta immediato il pensare ad una loro coesione e questo è stato il mio principale obiettivo.

Nondimeno l'obiettivo di carattere didattico associato all'indispensabile apprendimento del Linguaggio Java da parte mia è stato fondamentale. Non avendo mai affrontato un corso di questo linguaggio ho deciso di impegnarmi in un progetto implementativo di questo tipo per imparare ad affrontare le varie problematiche a cui va incontro uno sviluppatore di software.

Analisi del Lavoro

Per prima cosa ho organizzato un mio piano di lavoro. Ho previsto per prima cosa una buona parte di studio dei due sistemi e del linguaggio Java tramite la creazione di semplici applicativi di carattere didattico.

La prima fase esecutiva è stata prendere confidenza con JADE imparando a creare container e ad utilizzare l'utilissimo tool *Dummy Agent* che permette una semplice gestione dei messaggi.

La seconda fase è stata l'implementazione di alcuni semplici programmi Java che sfruttassero le classi di tuProlog. Ho imparato a gestire le teorie e i goal e successivamente a leggere teorie da file per poi inserirle nelle *Knowledge Base*(KB) da me create all'interno di questi programmi.

Ho iniziato poi a pensare a come effettuare una sorta di interfaccia tra i due sistemi. La prima soluzione da me adottata è stata la più semplice e funzionale a mio parere: creare un agente normalmente *Idle* che, a fronte di messaggi inviati dall'esterno da altri agenti, possa gestire diverse KB e utilizzare le funzionalità di tuProlog. Il nome di questo agente è "*Motore2p*". Ho creato inoltre una sorta di protocollo di comunicazione che definisce la sintassi e la semantica da utilizzare nell'invio di messaggi al mio agente per impostare delle KB.

Lo scenario che ho immaginato è quello di un agente che si muove in un ambiente ostile. Poniamo che questo agente per questioni economiche abbia poca potenza di calcolo e poca memoria disponibile. In questo contesto supponiamo che l'agente, attraverso ad esempio l'esplorazione, riesca a definire un grande numero di fatti. Grazie alla sottostruttura di JADE esso potrà inviare un messaggio contenente questi fatti sotto forma di messaggi ACL al mio motore tuProlog che li immagazzinerà in un'opportuna KB l'informazione.

Un altro scenario immaginabile è quello dei sistemi ad agenti peer-to-peer ne Web. In questo caso messaggi di questo tipo potrebbero andare da un agente all'altro modificando la fiducia che un agente ha negli altri. Questa fiducia potrebbe essere immagazzinata in una KB che viene continuamente aggiornata da fatti del tipo: "*SetXcrede(AgenteA, AgenteB, Fiducia)*". Questo messaggio sta a significare che l'Agente A, a causa di passate interazioni che ha avuto con l'Agente B, indica che quest'ultimo è meritevole di fiducia per il valore Fiducia che sarà un valore compreso tra 0 e 1. E' immediato il pensare che un agente prima di iniziare uno scambio di informazioni con un altro agente controlli nelle proprie KB i valori di fiducia, opportunamente pesati, a proposito di quest'ultimo e decida a priori se effettuare la comunicazione o no.

Dopo la realizzazione di questo agente ho dedicato la mia attenzione alla costruzione di un agente Client chiamato "*Utente*". Ho creato questo agente per poter innanzitutto creare una comunicazione tra due miei agenti senza dover utilizzare il Dummy Agent. *Utente* è un agente particolarmente complesso che ho fornito di un'interfaccia grafica per un più facile utilizzo. Grazie a questo agente si può selezionare un ricevitore all'interno della piattaforma JADE e si può inviare ad esso messaggi o il contenuto di un qualunque file a cui l'agente possa accedere. In particolare se *Utente* viene utilizzato per sfruttare le potenzialità di *Motore2p* esso può, seguendo il protocollo sotto descritto, gestire le KB da remoto e settare teorie contenute in file di testo. Per la creazione di questo agente ho utilizzato molto lo *Sniffer Agent*. Si rimanda al Tutorial per migliori spiegazioni.

Protocollo di comunicazione

Descriverò ora la prassi da seguire nell'utilizzazione dell'agente *Motore2p* da me creato. L'utente a cui mi riferirò può potrebbe essere un utente che invia e riceve messaggi tramite il Dummy Agent (a scopo di verifica di funzionamento) oppure potrebbe essere un qualunque agente implementato che voglia sfruttare il servizio o ancora il mio agente Utente.

Per prima cosa l'utente che vuole utilizzare il servizio deve installare una KB che verrà memorizzata dal fornitore del servizio (ovvero dove risiede *Motore2p*). Per fare questo è necessario inviare un messaggio contenente la parola:

Install

A questo punto il *Motore2p* invierà una risposta contenente un numero. Questo numero sarà identificativo della KB creata.

Tramite questo messaggio un qualunque agente può fare installare un qualsivoglia numero di KB a cui potrà accedere.

NB: Non ho implementato alcuna funzionalità di sicurezza per poter aumentare la poliedricità dello strumento. Infatti altri agenti potrebbero utilizzare KB create da altri, funzionalità che potrebbe rivelarsi utile in alcune applicazioni.

Il passo successivo è quello di *settare* una teoria. Questo passo verrà effettuato tramite l'invio di messaggi contenenti i fatti e le clausole desiderati. La sintassi dei messaggi deve essere:

SetXfatto. o SetXclausola.

La X sta per il numero identificativo della KB a cui si fa riferimento. Si ricorda che ogni fatto o clausola deve essere seguito da un punto.

Per esempio: "Set1p(a,b)." setta il fatto "p(a,b)." nella KB numero 1.

"Set1p(a,b):-p(a,x),p(x,b)." setta la clausola "p(a,b):-p(a,x),p(x,b)." nella KB numero 1.

E' inoltre possibile inserire più fatti o clausole nello stesso messaggio, è sufficiente che essi siano disposti su righe sovrapposte.

Per esempio: "Set1p(a,b).
p(c,d).
p(a,f).

Dopo aver settato la teoria l'utente può richiedere la dimostrazione di un goal. La sintassi dei messaggi di dimostrazione sarà:

DemoXgoal.

Anche in questo caso la X sta per il numero identificativo della KB a cui si fa riferimento.

Per esempio: "Demo1p(A,b)." richiede le soluzioni del goal "p(A,b)." nella KB numero 1.

Il Motore2p risolve il goal usando la KB definita e poi spedisce un messaggio all'utente in cui è contenuta ogni possibile soluzione del goal.

L'utente può anche cancellare il contenuto di una determinata KB con un messaggio:

ClearX.

Per esempio: "Clear1." cancella la KB numerata con 1.

Tutorial

In questo capitolo spiegherò come usare la piattaforma JADE per usare la mia applicazione. Prima di tutto è necessario settare le variabili globali d'ambiente per includere i files: jade.jar, jadeTools.jar, Base64.jar, iiop.jar e 2p.jar. I primi quattro contengono le librerie di JADE e l'ultimo è il file JAR di tuProlog.

Per esempio se si usa un sistema Windows si digiti sul prompt di MS-DOS, supponendo di avere i file in una directory *Temp* su C:

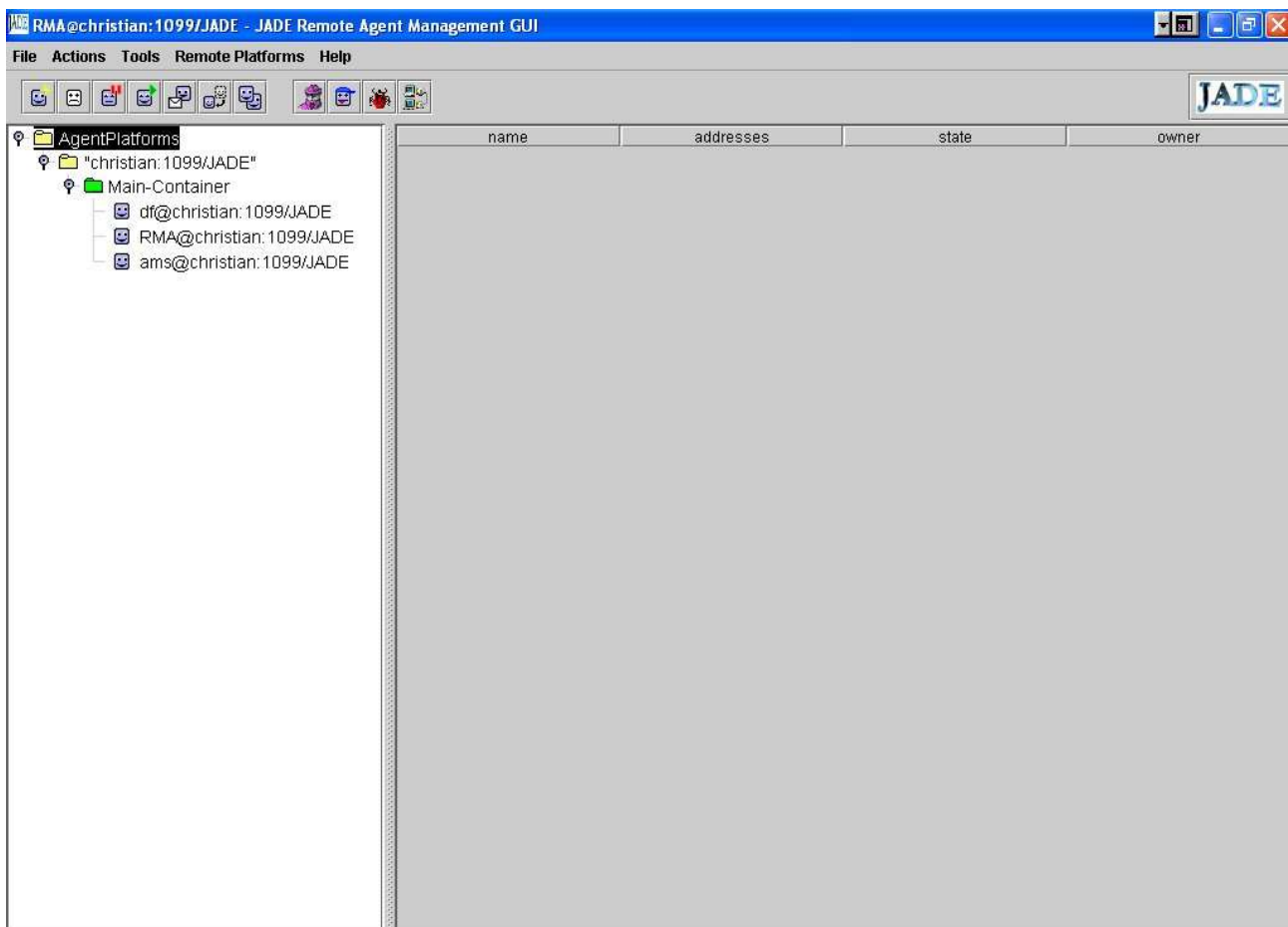
```
set CLASSPATH=.;c:\temp\jade.jar; c:\temp\jadeTools.jar; c:\temp\Base64.jar;  
c:\temp\iiop.jar; c:\temp\2p.jar
```

A questo punto è possibile avviare la piattaforma JADE digitando sul prompt (richiesto JDK 1.2 o più recente):

```
java jade.Boot -gui
```

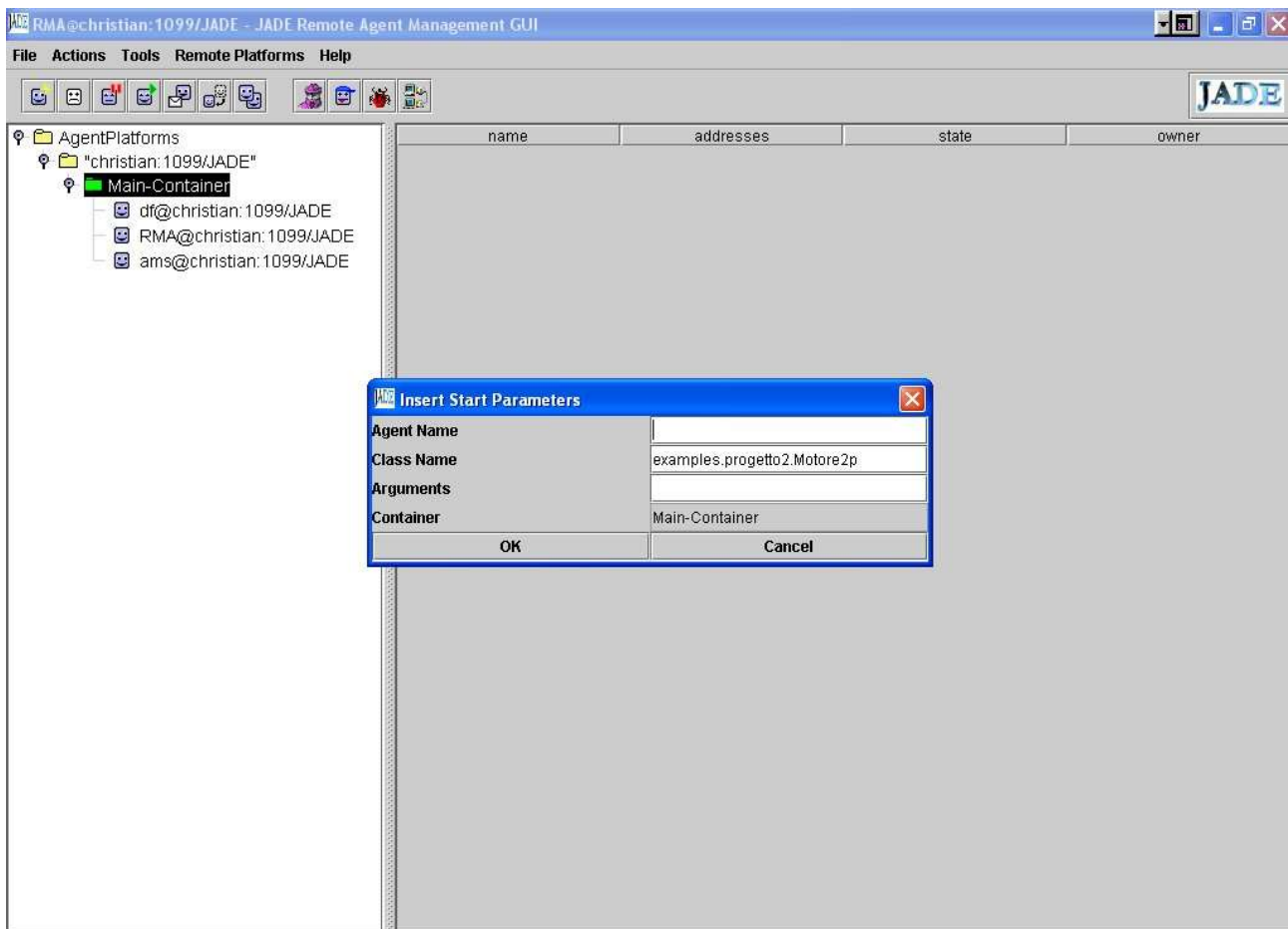
Verrà così avviata, grazie all'opzione `-gui`, l'interfaccia grafica della piattaforma.

Se tutto è stato impostato correttamente apparirà una schermata come questa:



Si può notare all'interno della colonna di sinistra il nome della piattaforma sul mio Pc ovvero Christian:1099/JADE. All'interno della piattaforma vi è un Main-Container che contiene 3 agenti necessari al corretto funzionamento di JADE: df, ams e RMA.

Procediamo ora all'avviamento dell'agente Motore2p per verificarne il funzionamento. Si clicchi con pulsante destro del mouse sul Main-Container e si selezioni Start New Agent. Apparirà questa finestra:

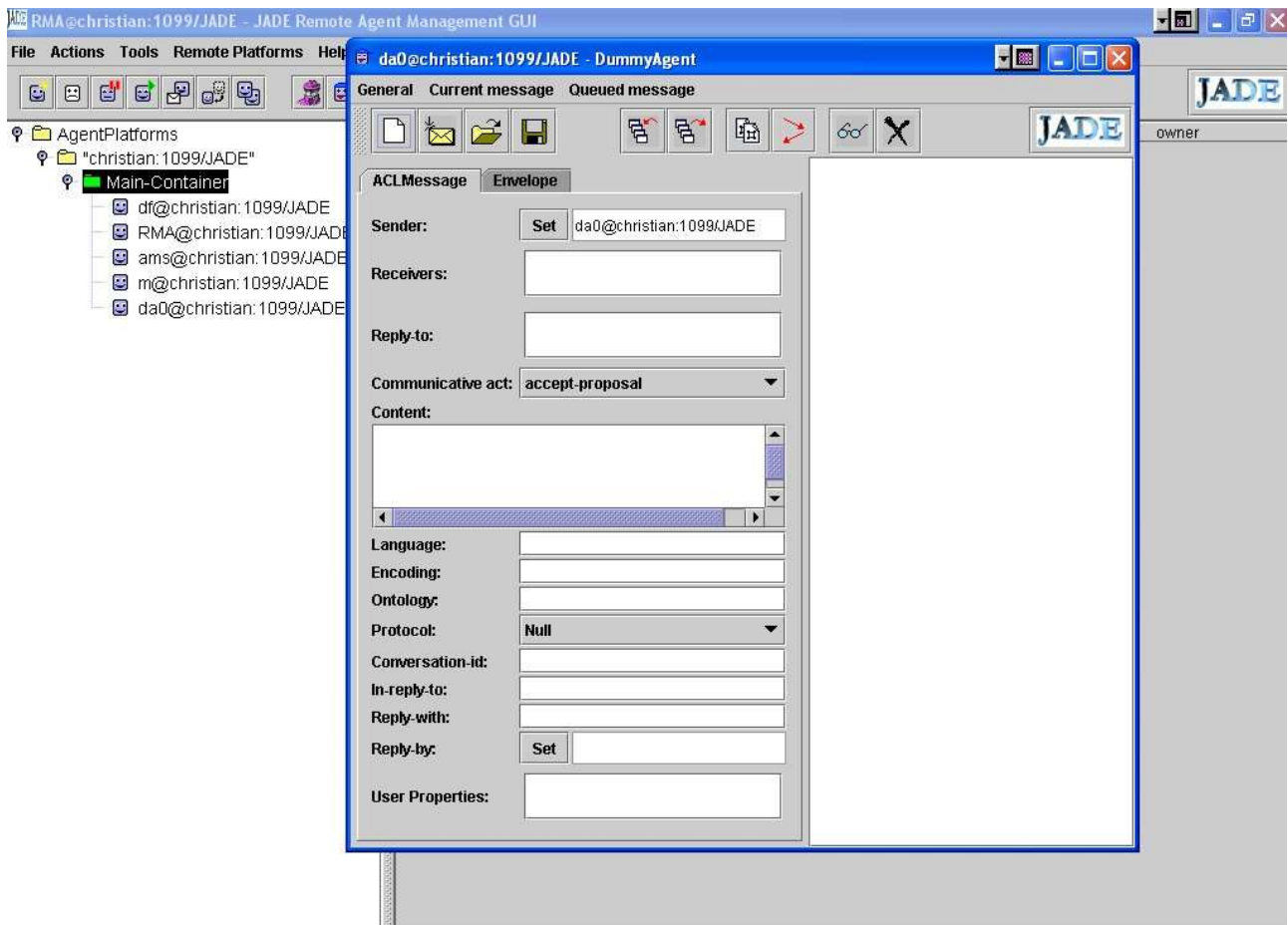


All'interno del campo Class Name ho inserito il nome della classe Motore2p da me costruita preceduta dal percorso che il compilatore Java deve seguire partendo dalla directory in cui si trova Jade.Boot. Consiglio di inserire "m" come nome dell'agente.

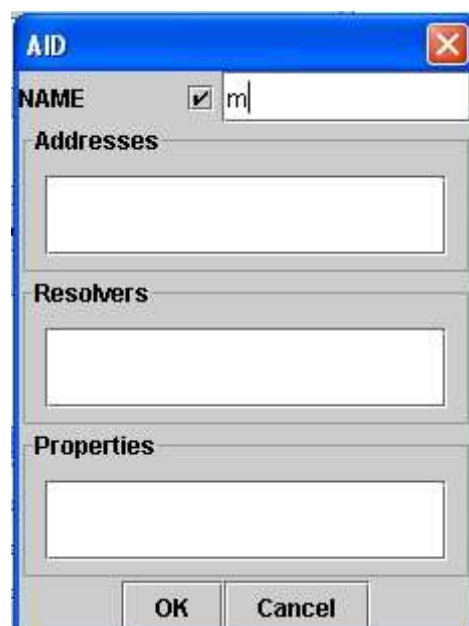
Se tutto è corretto nella colonna di sinistra si aggiungerà un agente con la denominazione:
nomeagente@nomepiattaforma:1099/JADE

A questo punto l'agente m è attivo e in attesa di messaggi provenienti dall'esterno.

Utilizzeremo ora il **Dummy Agent** per verificarne il funzionamento. Selezioniamo in tools il Dummy Agent ricordando di aver selezionato il Main-Container prima. Si aprirà questa finestra:



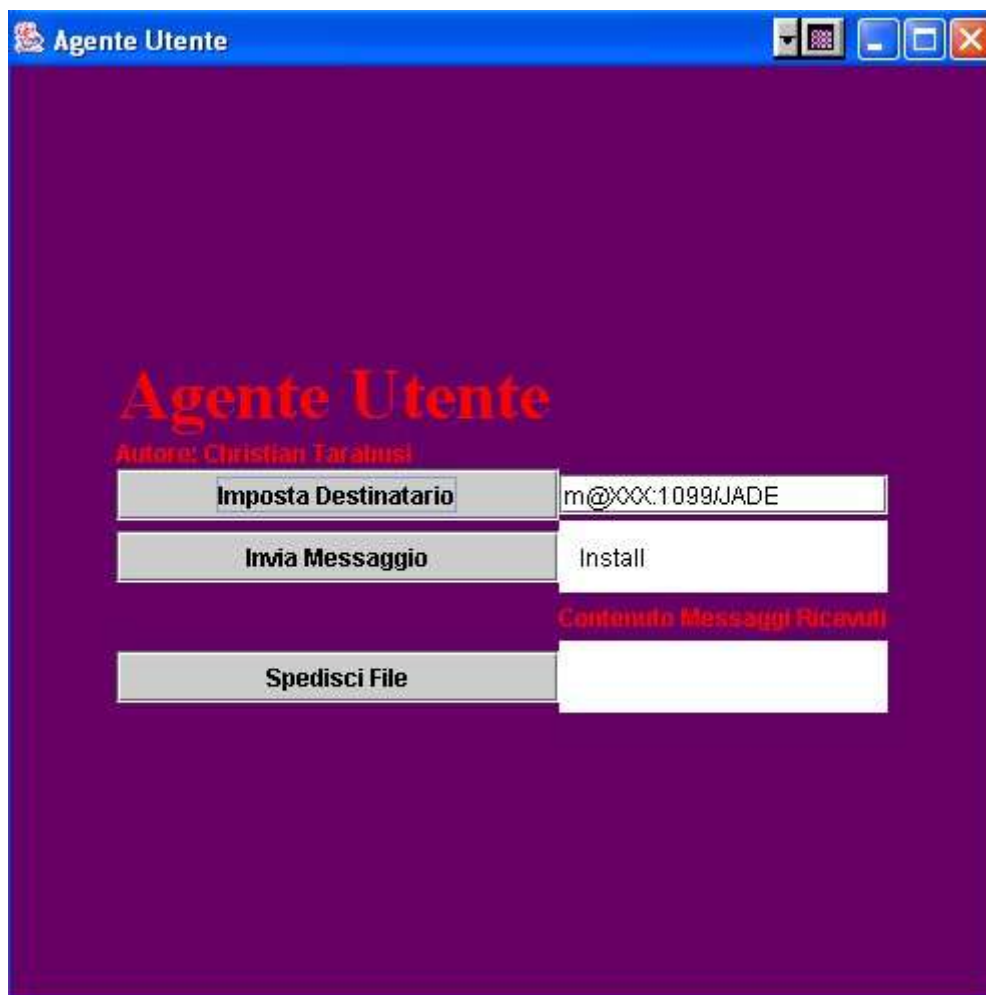
Per prima cosa si deve selezionare il ricevitore a cui il Dummy spedirà i messaggi. Per fare questo si clicca con il destro su Receivers e si seleziona Add. Si aprirà una finestra in cui si potrà scrivere per intero l'identificativo dell'agente a cui i messaggi verranno inviati. Nel caso che l'agente desiderato sia all'interno dello stesso container allora basterà inserire il nome che precede la @ e barrare il Tic a sinistra come viene mostrato nella figura seguente per il nostro agente chiamato "m".



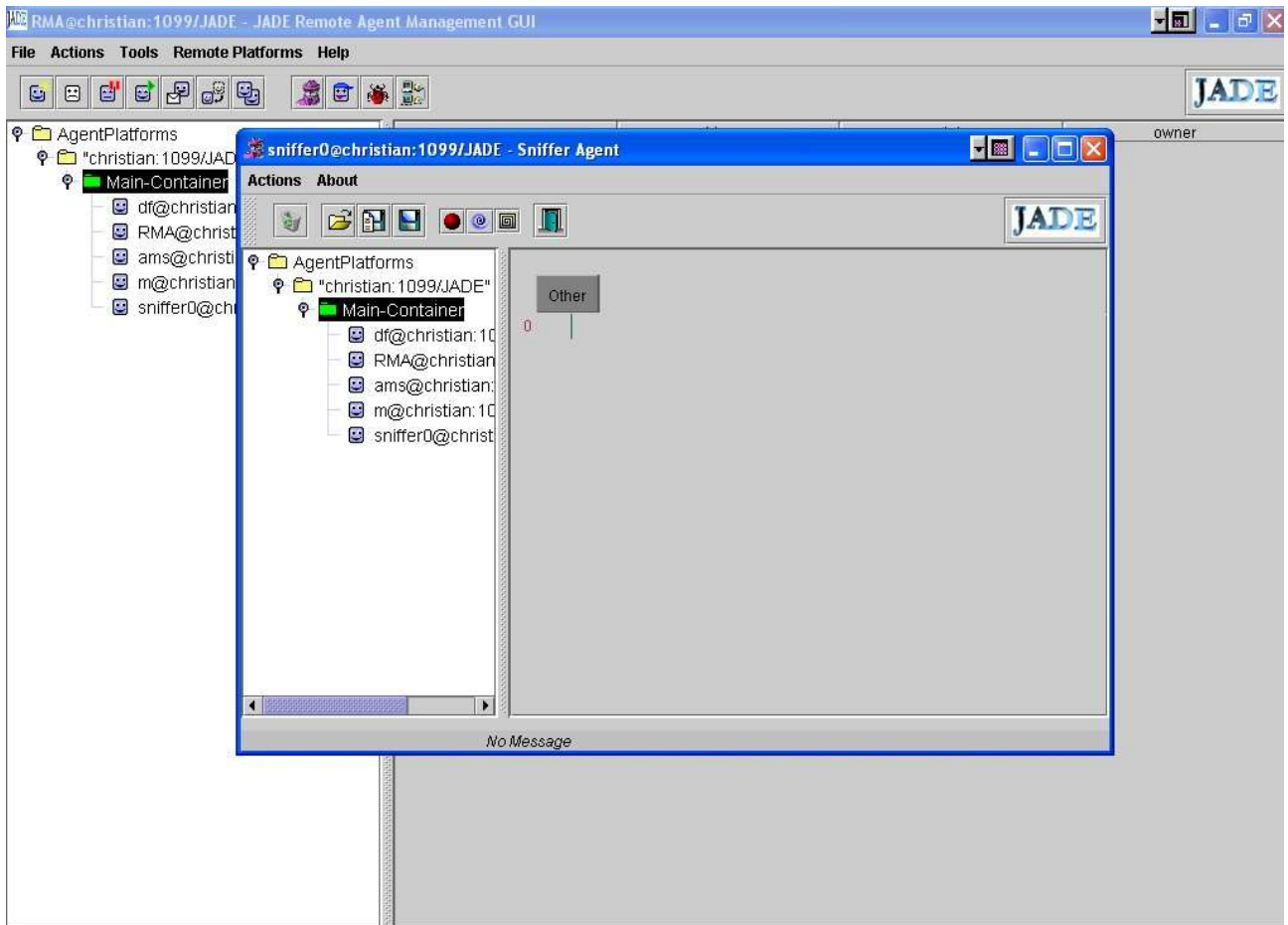
Dopo aver cliccato OK si torna alla finestra principale del Dummy. A questo punto i vari messaggi potranno essere scritti nel campo "Content" ed essere inviati premendo sulla busta gialla in alto a sinistra. L'ampia colonna a destra permette la visione della cronologia dei messaggi inviati e ricevuti dall'agente Dummy. Premendo l'icona degli occhiali è possibile visionare il contenuti dei vari messaggi.

Si vedrà ora il funzionamento del mio agente Utente. Innanzitutto si proceda in modo analogo all'attivazione di Motore2p ma si indichi la classe Utente.

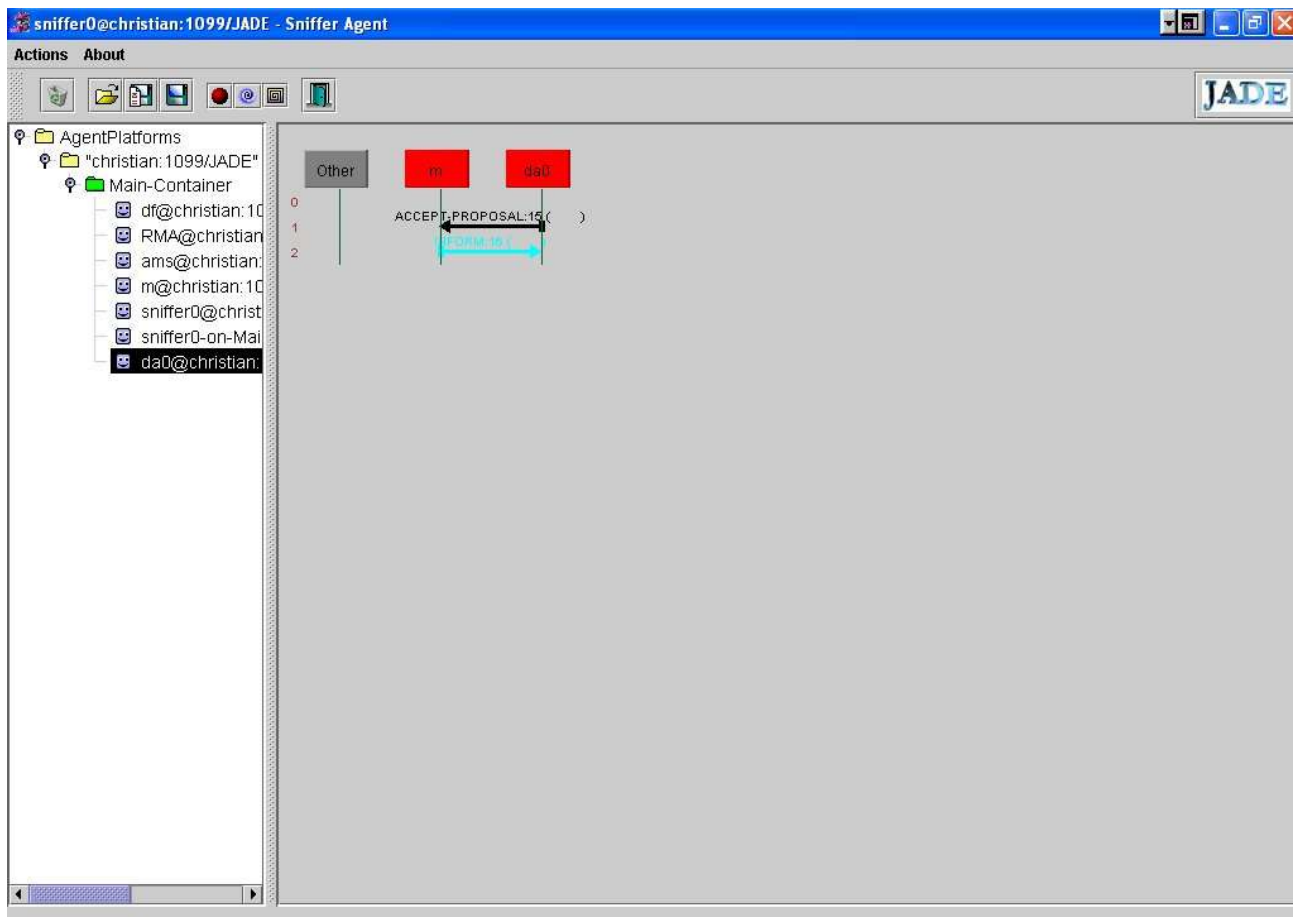
Non appena verrà creato l'agente si aprirà un'interfaccia grafica mostrata in figura. Per inviare un messaggio si deve indicare l'indirizzo completo del ricevente e premere il bottone "Salva Ricevente". Successivamente si scriva il messaggio e si clicchi "Invia Messaggio". E' possibile inviare un file in formato testuale tramite il bottone "Spedisci Oggetto". Se questo file contiene una teoria e viene inviata al Motore2p questo setterà automaticamente questa teoria nella KB numero 1.



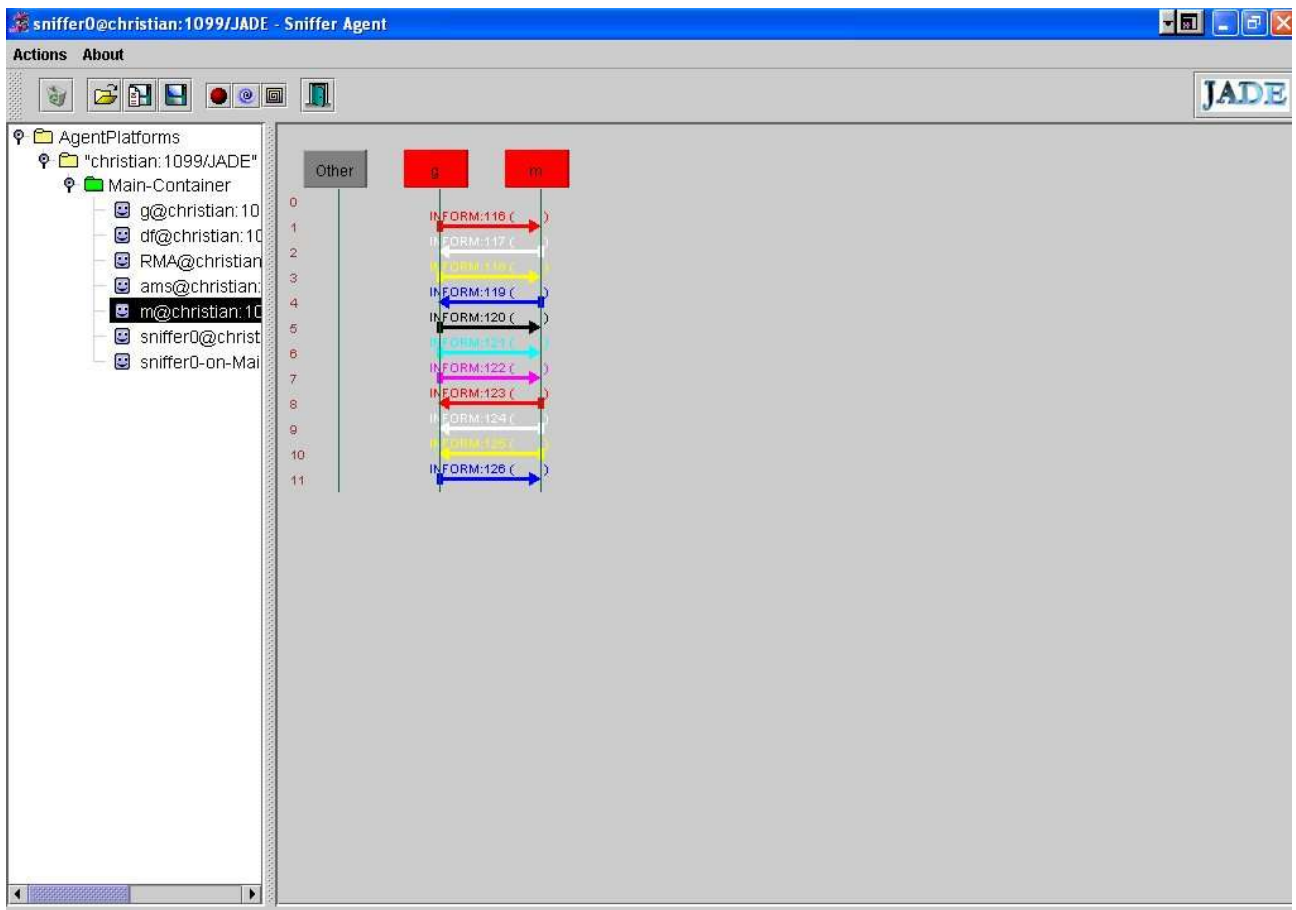
Grazie allo Sniffer Agent è possibile verificare il corretto funzionamento del sistema. Selezioniamo in tools lo Sniffer Agent ricordando di aver selezionato il Main-Container prima. Si aprirà questa finestra:



Ora per iniziare ad osservare i messaggi scambiati tra i vari agenti del sistema è sufficiente cliccare col tasto destro su un agente e impostare “Do Sniff This Agent” e un’icona rappresentante questo agente comparirà nella zona destra della schermata. Dopo l’invio e la ricezione dei messaggi la schermata viene aggiornata con l’aggiunta di alcune frecce. Nella figura seguente si potrà vedere il caso in cui un Dummy Agent (denominato con da0) invia un messaggio di Install al Motore2p “m”(vedi freccia nera) e come questo risponda prontamente inviando un messaggio contenente il numero della KB creata (vedi freccia azzurra).



Verrà ora proposto una schermata esemplificativa del funzionamento della mia interfaccia tra JADE e tuProlog.



Ricordo innanzitutto che “g” è il mio agente Utente ed “m” è l’agente che fornisce le funzionalità di gestione delle KB. Questa è la cronologia di ciò che è stato catturato dallo Sniffer:

- g invia un messaggio di Install e riceve il numero 1.
- g invia un secondo messaggio di Install e riceve il numero 2.
- g invia tre messaggi di Set nella KB 1 contenenti fatti e clausole.
- g richiede la dimostrazione di un goal
- m risponde con tre messaggi distinti, uno per ogni soluzione trovata per il goal.

Si notano un fatto molto importante. A volte lo Sniffer Agent compie delle inversioni dell’ordine temporale dello scambio dei messaggi. Questo malfunzionamento non dipende dalla mia implementazione (che funziona correttamente) ma evidentemente da una discretizzazione temporale non perfetta in certi casi.

A proposito della mia implementazione devo dire che è buon uso creare un numero di KB più grande del necessario poiché nonostante i miei tentativi di assicurare la capacità di memoria delle KB a volte vengono lanciate Eccezioni di tipo “IndexOutOfBoundsException”.

Bibliografia

Per la bibliografia si rimanda alla documentazione fornita gratuitamente dai costruttori dei due sistemi sotto forma di paper. In particolare risultano molto utili le API di JADE, tuProlog e, ovviamente di Java.

Sono ovviamente disponibile ad effettuare eventuali modifiche e perfezionamenti al programmi.