UNIVERSITA' DEGLI STUDI DI BRESCIA

FACOLTA' DI INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA ELETTRONICA

DIPARTIMENTO DI ELETTRONICA PER L'AUTOMAZIONE

DALL'UTENTE ALL'AGENTE: UN SISTEMA BASATO SU STANDARD FIPA PER L'ACCESSO A SERVIZI IN RETE

Relatore:

Prof. **PIETRO BARONI**

Correlatore:

Ing. **MASSIMILIANO GIACOMIN**

Tesi di laurea di:

EMILIANO VEZZOLI

Matricola 026717

ANNO ACCADEMICO 1999/2000

Alla mia Famiglia
…passata, presente e futura

Ringraziamenti

Un ringraziamento speciale va alla mia Famiglia, per avermi sempre sostenuto in questi anni di studio, soprattutto nei momenti difficili.

A mia madre, con cui mi sono sfogato nei periodi di tensione.

A mio padre, che non ha mai smesso di credere nelle mie capacità.

A mio fratello Marco (si… anche a lui!), che ha sempre preso in giro gli ingegneri ed ora è allievo gestionale…

Un grazie al mio relatore Prof. Pietro Baroni per la continua assistenza e disponibilità, nonché per il costante incoraggiamento nella realizzazione di questo lavoro di Tesi.

Infine, vorrei spendere qualche parola per tutti i miei amici, compagni di viaggio in questa lunga e tormentata avventura.

Grazie anche a loro, per avermi sopportato per così tanto tempo (e ancora mi dovranno sopportare…). Elencarli tutti è impresa assai ardua ma gradirei citare almeno: Marco, Valerio, Tiziano, Stefano A., Elena, Emiliano S., Fabio O., Francesca, Andrea, Stefano Z., Fabio L., Daniela, Angelo, Sergio, Agostino, Francesco… e tutti gli altri che qui non ho menzionato.

Grazie a tutti.

Emiliano

PREFAZIONE

Questo lavoro di tesi ha lo scopo di realizzare un'infrastruttura software per l'interazione tra utente ed un Sistema ad Agenti preposto alla fornitura di servizi via rete. Esso si colloca nell'ambito della realizzazione di un sistema prototipale basato sulla tecnologia degli agenti software per la fornitura di servizi ad una comunità di studenti universitari. Il motivo per cui oggi tale tecnologia assume un ruolo di primo piano nella ricerca sulla strutturazione dei moderni sistemi software va individuato nella necessità di fornire servizi in maniera sempre più completa ed efficiente. Il crescente sviluppo di Internet, dei suoi contenuti ed il recente aumento dei sistemi distribuiti, evidenziano l'esigenza di un nuovo approccio, che permetta la gestione e l'utilizzo di tutte le informazioni acquisibili via rete in modo dinamico e, soprattutto, efficace per l'utente. Gli Agenti Software rappresentano una tecnica innovativa che permette, tra le altre cose, di automatizzare e rendere più efficace la fornitura di un servizio. Questo avviene perché è il sistema ad Agenti che si fa carico di tutte quelle operazioni necessarie allo svolgimento di un determinato compito ed è potenzialmente in grado di fornire servizi più evoluti rispetto a quelli realizzati con tecnologie software più tradizionali.

In questo lavoro di Tesi sono state affrontate diverse problematiche: dopo uno studio approfondito dello standard FIPA e di quanto presente in letteratura relativamente alle tecnologie ad Agenti ed agli strumenti di sviluppo, è stato realizzato un sistema multi-Agente conforme alle specifiche FIPA, servendosi della libreria software JADE. Sono stati programmati diversi Agenti software tra loro comunicanti e cooperanti: un Agente di Interfaccia preposto al dialogo con l'utente, un Agente di

servizio, intermediario con la piattaforma, e tre Agenti di Applicazione dedicati alla ricerca di informazioni in database. Inoltre si è studiato come far comunicare attraverso la rete utenti e sistema ad Agenti, analizzando le caratteristiche delle più diffuse tecniche di interconnessione client/server, allo scopo di permettere l'accesso ai servizi proposti attraverso un comune browser.

INDICE

INDICE DELLE FIGURE

Capitolo 1

LA TECNOLOGIA DEGLI AGENTI SOFTWARE

In questo capitolo vengono presi in esame i concetti chiave alla base di questo lavoro di tesi: la definizione di “Agente Software” e “Sistema ad Agenti”, con particolare riferimento alla loro relazione con l’Intelligenza Artificiale. Si considererà, quindi, l’approccio alla programmazione basata su Agenti software e si farà cenno agli usi applicativi di tale tecnologia.

1.1 LA PROGRAMMAZIONE AD AGENTI

Una delle più significative evoluzioni nel campo dello sviluppo software è stata rappresentata dall’introduzione della programmazione ad oggetti (OOP, *Object Oriented Programming*), un modo per concettualizzare un programma come insieme di *oggetti* (software) che interagiscono tra loro. Ognuno di questi oggetti comprende una struttura dati ed un'insieme di operazioni che possono essere ad essa applicate. Gli oggetti aiutano il programmatore a gestire la complessità dei sistemi software, migliorandone la comprensibilità, la riutilizzabilità e la robustezza.

La *comprensibilità* è migliorata poiché ogni oggetto si riferisce ad un entità che ha un preciso significato nel dominio dei problemi affrontati o delle soluzioni che si cercano; inoltre, gli oggetti interagiscono fra loro mediante meccanismi ben definiti.

La *riutilizzabilità* deriva direttamente dalla comprensibilità: se un oggetto è ben comprensibile, potrà essere utilizzato in applicazioni differenti, anche grazie al meccanismo di estensione denominato ereditarietà.

La *robustezza* riguarda l'architettura del programma e la possibilità di evitare errori in fase di esecuzione. Poiché gli oggetti utilizzano tecniche di incapsulamento per nascondere all'utente alcune loro parti, un programma orientato agli oggetti è di norma meno soggetto ad errori di un programma modulare. Questo perché risulta più facile identificarli ed eliminarli.

Le principali caratteristiche della OOP sono Identità, Classificazione, Ereditarietà, Polimorfismo e Incapsulamento [DEL99].

- ***Identità*:** ogni oggetto è unico, perciò due oggetti sono distinti anche se i loro attributi sono identici.
- ***Classificazione*:** ogni oggetto è istanza di una "classe", che rappresenta un insieme di oggetti aventi le stesse proprietà.
- ***Ereditarietà*:** Mediante questo potente meccanismo, il programmatore può costruire nuove classi estendendo quelle esistenti. Ogni nuova classe può derivare da una sola classe esistente (ereditarietà singola) o da più classi (ereditarietà multipla). Le classi derivate sono anche chiamate "sottoclassi".
- ***Polimorfismo*:** letteralmente significa capacità di assumere diverse forme: nella programmazione ad oggetti questo indica la capacità di una entità di riferirsi ad istanze di classi differenti, purchè discendenti di una classe comune nella gerarchia di eredità. Questo può essere realizzato sia in fase di compilazione, sia in fase di esecuzione. Il polimorfismo consente ad una interfaccia di essere

utilizzata per una classe generale di azioni, indipendentemente dallo specifico tipo di dati su cui sono applicate. Il polimorfismo aiuta quindi a ridurre la complessità del programma, lasciando al compilatore il compito di selezionare il metodo da applicare in una determinata situazione per una specifica sottoclasse.

- ***Incapsulamento*:** ogni oggetto possiede un "interfaccia", costituita da variabili e funzioni a cui si può accedere liberamente dall'esterno, e da una "struttura interna", che contiene dati e funzioni inaccessibili dall'esterno. L'incapsulamento è anche detto "Information Hiding".

Senza entrare nel dettaglio della OOP, che esula dalla scopo di questa trattazione, basti ricordare che questo nuovo approccio ha portato alla nascita di una filosofia di programmazione per **componenti**, tanto che lo sviluppo di classi specializzate nella soluzione di certi problemi o nell'implementazione di certe funzionalità è diventato un mercato vero e proprio. Nuovi linguaggi di programmazione sono nati, alcuni come estensione di linguaggi tradizionali (come il C++, derivato dal C e che permette anche una programmazione «mista», o il CLOS, estensione ad oggetti del Lisp), altri ex-novo (come SmallTalk o Java, impostati totalmente sulla filosofia OOP).

Ora, a qualche anno di distanza, una nuova filosofia di programmazione sta emergendo, grazie alla crescente diffusione di ambienti distribuiti (reti locali, Internet).

Questo scenario, infatti, permette di superare sia la programmazione tradizionale, in cui un programma gira su una macchina, sia la programmazione con approccio client-server, in cui un'applicazione è suddivisa in due o più moduli, che possono risiedere su macchine diverse e dialogano tra loro conservando però una ben definita distinzione tra i

ruoli. Infatti, avendo a disposizione un ambiente di rete (entro i confini di una specifica organizzazione oppure ad un livello più esteso), si possono concepire **applicazioni distribuite**, formate da una comunità di componenti detti **agenti** che non solo comunicano fra loro, ma possono, in alcuni casi, anche essere in grado di spostarsi da un computer ad un'altro insieme con i propri dati.

La programmazione ad agenti, in effetti, può essere considerata l'applicazione pratica di questi concetti.

Come risultato del lavoro di molti ricercatori, nell'ambito delle Università e di grandi aziende, sono già apparse diverse librerie utilizzabili per creare applicazioni multiagente. La programmazione ad agenti è basata strettamente su quella ad oggetti, in quanto un agente può essere definito in modo molto naturale come un oggetto, o meglio come un'istanza di una **classe** che incapsula le funzionalità implementate dall'agente, i cui metodi e proprietà definiscono i **servizi** che esso mette a disposizione degli altri agenti.

Vediamo ora di approfondire cosa si intende per *Agente Software*.

1.2 DEFINIZIONE DI AGENTE

Nonostante il termine "Agente" sia sempre più spesso utilizzato nell'ambito della letteratura tecnica e scientifica, nell'area dell'Intelligenza Artificiale (AI) ed, ancora, in altri settori dell'informatica, una definizione univoca ed universale che lo caratterizzi ancora non esiste. Questo aspetto teorico può apparire secondario alla luce delle centinaia di applicazioni ed implementazioni pratiche che fanno uso degli agenti, ma non è così. Un abuso indiscriminato del termine può generare , infatti, confusione e disorientamento nel mondo

informatico e nuocere allo sviluppo della tecnologia. Tra le numerose definizioni presenti in letteratura quella di Nicholas R. Jennings e Michael Wooldridge sembra essere la piu' adatta nel contesto di questa trattazione [JEN95].

Essi definiscono "[...] un agente come un sistema informatico situato in certo ambiente, capace di azioni **autonome** e **flessibili** al fine di ottenere i suoi obiettivi di progetto". L'agente viene pertanto descritto come un'entità il cui funzionamento è continuo ed autonomo, in grado di operare per conto di altri utenti o altri agenti, ma anche in maniera indipendente da essi. L'agente è " situato in un ambiente " recita la definizione, e con esso deve necessariamente interagire, in particolare mostrando capacità di comunicazione e cooperazione con gli altri agenti eventualmente presenti nel sistema. Michael Wooldridge aggiunge: "un'agente è un'entità che agisce attivamente, modificando l'ambiente piuttosto che subirlo passivamente. Agisce in modo autonomo, senza l'intervento diretto dell'uomo, cercando di raggiungere il suo obiettivo massimizzando una qualche funzione valutativa. E' un sistema di alto livello che sembra soggetto a desideri, giudizi, o a qualche altra funzione di tipo cognitivo. E' un sistema intenzionale. Più correttamente si tratta di un sistema che, data la sua complessità, è conveniente descrivere tramite gli **stati intenzionali**". Anche J.Horberg basa la sua definizione di agente sul concetto di autonomia ma introducendo un nuovo elemento: la mobilità: "L'agente è un'entità mobile, che viaggia attraverso la rete ed è in grado di duplicarsi su altri computer per eseguire il compito assegnatogli. Inoltre, un agente è un'entità autonoma, cioè esso è in grado di eseguire il proprio compito da solo e di prendere decisioni autonomamente, perfino in vece dell'utente"[HOR95].

Si è visto, quindi, come gli agenti vengano visti come **Sistemi Intenzionali**, ma ancora non si è detto cosa significhi questa espressione. Il termine Sistema Intenzionale è stato coniato dal filosofo Daniel

Dennett per descrivere "Entità il cui comportamento può essere previsto attraverso il metodo dell'assegnazione di credenze, desideri e capacità razionali" [DEN87]. Egli ha distinto differenti "gradi" per questi sistemi: "Un sistema intenzionale del primo ordine ha opinioni e desideri, ma non opinioni e desideri che riguardino quest'ultimi,... Un sistema intenzionale del secondo ordine è più sofisticato; ha infatti opinioni e desideri anche verso le proprie opinioni e i propri desideri"; Senza entrare nel dettaglio di questioni che esulano lo scopo di questa trattazione e che rientrano in un ambito piu' filosofico basti sottolineare il concetto che un agente può essere visto come un'entità software responsabile per l'esecuzione di un determinato compito, contenente un certo livello di "intelligenza" e operante per conto di un'utente o di uno/altri agenti.

Jennings sottolinea come sia possibile delineare due diverse caratterizzazioni per il termine agente. Una accezione **Debole** (o Weak) dove il termine agente viene semplicemente usato per indicare un software capace di *Autonomia* (capacità di operare senza l'intervento umano), *Capacità Sociale* (capacità di interagire con altri agenti), *Reattività* (capacità di reagire a stimoli esterni), *Iniziativa* (Pro-Activeness, capacità di prendere l'iniziativa) ed una accezione **Forte** (o Stronger) dove all'agente vengono assegnati attributi tipicamente associati all'attività mentale umana quali *Conoscenza*, *Intenzioni*, *Obblighi*, *Desideri*, ...

L'accezione *Forte* del termine Agente trova maggiori sostenitori tra i ricercatori dell'AI, al contrario della prima (*Debole*), caratterizzata da maggiori consensi in un ambito più generale.

Altri attributi comunemente associati agli agenti sono i seguenti :

- *mobilità*: è la capacità di un agente di spostarsi all'interno dell'ambiente in cui opera, per poter raggiungere le risorse di cui necessita o le informazioni di cui è alla ricerca. Per gli agenti

fisici il movimento avviene all'interno di un ambiente reale, mentre per gli agenti software si intende il trasferimento da un nodo all'altro di una rete di computer. La mobilità rappresenta un ulteriore aspetto di autonomia, in quanto la scelta del percorso da seguire o del sito su cui fermarsi sono lasciate al giudizio dell'agente; si possono distinguere due livelli di mobilità:

- Esecuzione Remota: l'Agente (programma + dati) viene spostato su un altro Host ed eseguito interamente in remoto.
- Migrazione: durante la propria esecuzione un Agente è in grado di spostarsi di propria iniziativa in un altro nodo. Quindi è capace di sospendere la propria esecuzione, migrare (programma + dati + stato di esecuzione) e riattivarsi dal punto di sospensione.

- *cooperazione*: si ha invece quando più agenti sono impegnati nel conseguimento di un obiettivo comune: questo implica sia aspetti di comunicazione che di coordinamento, necessari per organizzare i partecipanti all'operazione seguendo una certa strategia, e per permettere lo scambio dei risultati parziali raggiunti;
- *apprendimento*: è un aspetto molto discusso perché legato al concetto di intelligenza; si tratta della capacità di fare tesoro delle esperienze passate, per servirsene al momento di dover prendere una determinata decisione. La conoscenza conservata da un agente può essere di diverso genere, e spaziare da un singolo bit di stato ad un intero database. Nel prossimo paragrafo si discuterà del rapporto tra Agenti ed Intelligenza Artificiale.

- *adattamento*: indica la proprietà dell'agente di modificare il proprio comportamento in seguito alle mutate condizioni dell'ambiente circostante. L'agente si dice *adattativo* se è in grado di cambiare autonomamente il proprio comportamento per adeguarsi alle situazioni in cui si viene a trovare.
- *Sincerità*: incapacità di comunicare intenzionalmente false informazioni. Si tratta, cioè, di supporre che l'Agente non sia in grado scegliere tra menzogna e verità.
- *Benevolenza*: questo significa che un Agente non può avere obiettivi da raggiungere intrinsecamente in contrasto con quelli di altri Agenti [GEN97].
- *Razionalità*: questo concetto si basa sull'assunzione che le azioni compiute da un Agente siano volte al perseguimento di determinati obiettivi e che, in nessun caso, cerchino di ostacolarne il raggiungimento.

Un Agente non deve necessariamente possedere tutte le proprietà descritte precedentemente ma ve ne sono alcune, come l'adattamento e l'apprendimento, che risultano strettamente correlate tra loro e pertanto può risultare arduo distinguerle. In altri casi si considerano sistemi composti da più agenti, ognuno caratterizzato da certi attributi, e si parla di Sistema multi-Agente. Se è difficile dare una definizione universalmente riconosciuta di Agente, lo è altrettanto dare quella di Sistema ad Agenti. Questa, infatti, si basa sui concetti di Agente e di Piattaforma o Contenitore d'Agenti, per modellizzare una struttura software complessa costituita da processi cooperanti 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 gli agenti possono vivere ed operare.

Le prime ricerche su sistemi composti da più agenti prendevano il nome di Intelligenza Artificiale Distribuita (DAI), e venivano suddivise in due branche: ***Multiple Problem Solving*** (MPS) e ***Distributed Problem Solving*** (DPS). I sistemi **MPS** sono caratterizzati dalla presenza di strutture informative condivise tra gli Agenti che rappresentano lo stato globale del processo di problem-solving.

- La comunicazione tra gli Agenti avviene attraverso una struttura condivisa.
- Sono necessari dei criteri per sapere quando il processo va terminato. Di solito è una delle sorgenti di conoscenza ad indicarlo.
- Il comportamento di problem-solving di un sistema di questo tipo è determinato dalla strategia di applicazione della conoscenza codificata nel modulo di controllo, la quale dipende dalle caratteristiche del problema affrontato.

I sistemi **DPS**, invece, sono costituiti da un gruppo di Agenti *indipendenti* che esibiscono un comportamento *cooperativo* allo scopo di svolgere l'attività di problem-solving. A differenza degli MPS, i DPS non possiedono, al loro interno, strutture informative globali. Questo comporta:

- Assenza di una struttura globale che contenga lo stato della soluzione del problema, la quale esiste in forma distribuita nei vari Agenti dell'architettura.
- Necessità un meccanismo di comunicazione tra gli Agenti, come lo scambio di messaggi, a causa della mancanza di strutture informative globali.

- L'organizzazione degli Agenti può essere non solo gerarchica ma di qualsiasi tipo, garantendo una certa flessibilità.
- Il controllo della procedura di problem-solving può diventare completamente distribuito perché non esistono limitazioni alla effettiva distribuzione del controllo, che vi sono nei moduli centralizzati e nelle strutture informative.

Recentemente l'espressione sistema multi-Agente viene utilizzata per indicare tutti i tipi di sistemi composti da più Agenti Software. Lo scopo è quello di coordinare le azioni di un gruppo di Agenti autonomi, in modo da far emergere un comportamento "intelligente".

Ogni Agente da solo, per quante informazioni o capacità possegga, non è in grado di risolvere un determinato problema. Si dice che l'Agente ha un punto di vista limitato.

Per questa ragione la cooperazione per il raggiungimento di un determinato obiettivo deve necessariamente prevedere lo scambio di informazioni tra gli Agenti stessi. Da qui la necessità di definire uno standard per la comunicazione che comprenda:

1. ***Un protocollo ed un linguaggio di comunicazione***: consistono in uno/più modello/i che descrivono le tecniche di comunicazione tra gli Agenti ed in un linguaggio astratto e completamente svincolato dalla struttura fisica dell'Agente.
2. ***Un formato per il contenuto dell'informazione***. Non è sufficiente stabilire un linguaggio. E' indispensabile anche determinare un formato univoco con cui rappresentare le informazioni, in modo tale che gli Agenti ne riconoscano il contenuto.
3. ***Una o più ontologie.*** Fissati un linguaggio ed un formato per il contenuto dell'informazione, come ultimo passo rimane la

definizione delle Ontologie. Si tratta, cioè, di dare un significato ai contenuti trasmessi.

Fra i tanti linguaggi di comunicazione tra Agenti si ricordano il **KQML** (Knowledge Query and Manipulation Language) e l'**ACL** (Agent Communication Language) che verranno approfonditi nei prossimi capitoli.

1.3 AGENTI E INTELLIGENZA ARTIFICIALE

Come anticipato precedentemente la nozione FORTE di Agente, cioè quella che attribuisce ad esso caratteristiche, termini e concetti tipici della descrizione degli stati mentali umani, viene preferita nell'ambito dell'Intelligenza Artificiale (AI). I ricercatori in questo campo, infatti, presuppongono che ogni Agente sia dotato di una certa "Intelligenza" e "Conoscenza". Molti filosofi sostengono che intelligente è quell'essere cosciente di esistere. Potremmo allora chiederci se può una macchina o un programma per computer essere consapevole della propria esistenza, ma non è questo il nocciolo della questione. J. McCarthy [MCC79] suggerisce , infatti, di tralasciare le complesse implicazioni filosofiche e di soffermarsi, invece, sulla *legittimità* di attribuire qualità mentali alle macchine e sull'*utilità* del farlo.

Il modo di comportarsi di un Agente in un qualsiasi istante della sua "vita" dipende essenzialmente da due fattori: la storia passata e la storia presente, cioè cosa ha fatto negli istanti passati e cosa sta facendo nell'istante considerato. La storia passata rilevante è inclusa nell'Agente sotto forma di credenze o convinzioni (Beliefs) sullo stato del mondo passato. Quest'ultime rappresentano, nella terminologia degli stati

mentali, ciò che viene chiamato “modello del mondo”. Altro non è che una sorta di contesto storico/sociale (Background) in cui è inserito l’agente. L’agente possiede in base a questo contesto credenze riguardo vari aspetti, quali il proprio stato, le proprie capacità e compentenze risolutive, informazioni su altri Agenti. Quindi quando un Agente interagisce con l’ambiente che lo circonda si basa sulle sue convinzioni, distinte in: passato, presente e futuro [MCC79].

1. **Convinzioni sul passato:** rappresentano lo stato del mondo al tempo corrente, costruito sulla base dei risultati delle interazioni che l’Agente ha svolto nel passato ed a cui fa riferimento per le proprie decisioni.
2. **Convinzioni sul presente:** rappresentano lo stato di quella porzione di mondo che sta interagendo con l’Agente. Questo stato emerge dall’interazione in corso e non dipende dalle convinzioni sul passato. Le convinzioni sul passato, al contrario, quando l’interazione presente termina, vengono aggiornate, fornendo una nuova “visione” del mondo in seguito alle ultime iterazioni.
3. **Convinzioni sul futuro:** sono una previsione, fatta dall’Agente, dello stato del mondo nel futuro, alla luce delle convinzioni sul passato e sul presente. Questo stato risulta utile quando le informazioni contenute nelle convinzioni sul passato sono insufficienti per prendere delle decisioni sulle azioni da compiere.

Le *convinzioni* o *credenze* rappresentano una prima categoria mentale di base e le decisioni che l’Agente può o deve compiere sono logicamente vincolate ad esse. Non bastano però a definirne univocamente il comportamento. Va infatti considerato che non è possibile prendere decisioni che contrastino con altre già prese in precedenza. Questo vincolo giustifica l’introduzione di un’altra categoria

mentale di base: la *decisione* [MCC79]. Quest'ultima è la causa delle azioni dell'Agente ma, affinchè possa collaborare con i suoi simili, va introdotta un'altra categoria nel nostro modello mentale: il concetto di *incarico*. E' possibile assimilare la nozione di decisione a quella di incarico, osservando che l'Agente può considerare una decisione come un incarico ricevuto da se stesso. A queste categorie di base possono esserne aggiunte altre, quali i concetti di tempo, azione,… Mediante l'uso di questi concetti, tipici della mente umana, è possibile realizzare un linguaggio in grado di descrivere il comportamento dell'Agente basato sugli stati mentali. Si tratta di un linguaggio astratto e completamente svincolato dalla struttura fisica dell'Agente. L'utilità del modello a stati mentali è rappresentata dal fatto che facilita la comprensione e la progettazione degli agenti, perché basata sul modo di agire di una persona. Anche l'uomo, infatti, sulla base della propria esperienza di vita (convinzioni sul passato) e sugli stimoli che riceve nel presente dall'ambiente in cui è inserito (convinzioni sul presente) prende delle decisioni e compie azioni che, inevitabilmente, si ripercuotono sull'ambiente stesso e sui suoi simili. Quindi può essere modellizzato con il sistema a stati mentali. Essendo un'entità più semplice rispetto ad un essere umano, l'Agente può essere descritto da un insieme limitato e ben definito di stati mentali, tanto che nel campo dell'AI si è sviluppata una disciplina chiamata *Agent-Oriented Programming* (AOP) che si dedica allo studio degli Agenti Intelligenti come entità dotate di stati mentali.

1.4 TIPOLOGIE DI AGENTE

Se arduo è il compito di definire cosa siano gli Agenti, altrettanto lo è cercare di darne una classificazione. Da un certo punto di vista la stessa definizione di Agente data da Jennings e Wooldridge ci viene in aiuto gettando le basi per una prima distinzione tra tipi di Agente: *Weak Agents* e *Strong Agents.*

Oltre a questo primo esempio di classificazione, basato sulle proprietà, è possibile raggruppare gli Agenti in diverse tipologie basandosi su criteri estremamente differenti. Per esempio in base al comportamento oppure al contesto applicativo in cui sono utilizzati [BAL98]. Va poi tenuto presente che , spesso, un agente può possedere aspetti in comune a varie tipologie.

Di seguito viene riportata la classificazione degli Agenti secondo H.Nwana [NWA96]:

1.4.1 ***Agenti Mobili*** *(Mobile Agents)* : sono entità software capaci di muoversi attraverso le reti di calcolatori, interagendo con diversi Host, ricercando e raccogliendo informazioni per conto di altri Agenti o per coloro di cui fanno le veci. Si parla anche di *Agenti di Rete.* Gli Agenti che non godono di questa proprietà sono definiti ***Statici*** (*Static Agents*). Un Agente mobile è in grado di accedere a risorse e servizi remoti grazie alla sua conoscenza dell'infrastruttura di rete e dei servizi disponibili nell'ambiente distribuito. Un agente mobile quando deve spostarsi attraverso la rete deve interrompere la propria l'esecuzione, memorizzare il proprio stato interno, effettuare il trasferimento sul nuovo Host e riprenderla dallo stato interrotto. Queste sospensioni possono essere, in alcuni casi, uno svantaggio per gli Agenti

Mobili. Un Agente Statico, infatti, è in grado di eseguire altre operazioni mentre sta comunicando con altri Agenti Statici.

Nonostante questo inconveniente, l'utilizzo di Agenti Mobili si fa sempre più intenso negli ultimi anni, a fronte della crescita esplosiva di Internet e della necessità di accedere ad una enorme quantità di informazioni e servizi. Molti autori affermano che questo nuovo metodo di programmazione aprirà nuove prospettive per il controllo delle transazioni e della ricerca distribuita in rete , fino ad arrivare al concetto di rete attiva. Altri obbiettano però che gli agenti mobili pongono seri problemi per quel che riguarda la sicurezza, dato che si tratterebbe di avere del codice, i cui effetti non sarebbero noti a priori, in esecuzione sulla propria stazione di lavoro.

1.4.2 ***Agenti di Interfaccia*** *(Interface Agents)* : sono quegli Agenti preposti all'interfacciamento con l'utente ed alla sua assistenza nell'esecuzione di determinati compiti. Numerosi studi sull'argomento sono stati effettuati facendo uso di applicazioni esistenti, connettendole ad agenti di interfaccia in grado di "apprendere". Questo significa, essenzialmente, che l'Agente è in grado di monitorare il comportamento dell'operatore, di imitarlo, di ricevere informazioni da parte sua sia in forma esplicita (sotto forma di comandi o istruzioni) sia in forma implicita (attraverso l'osservazione delle abitudini e delle azioni ripetitive compiute dall'utente stesso). Infine è in grado di contattare ed interagire con altri Agenti per eseguire i suoi compiti.

Gli Agenti di Interfaccia in genere sono Agenti Statici, cioè non sono in grado di spostarsi dalla macchina fisica in cui sono situati, in contrapposizione agli Agenti mobili.

1.4.3 ***Agenti Cooperativi*** *(Co-operative Agents)* : un'agente di questo tipo può cooperare, collaborare con l'ambiente che lo circonda o con altri agenti per il raggiungimento di un obiettivo, sia comune che proprio di un singolo Agente. Questi Agenti necessitano, quindi, di un articolato sistema di comunicazione che permetta loro lo scambio di informazioni. Alla base di questo sistema deve esserci un linguaggio comune. Oggi ne esistono diversi, ma i più noti sono FIPA-ACL e KQML (*Knowledge Query and Manipulation language*).

1.4.4 ***Agenti Ibridi*** *(Hybrid Agents)* : uno dei principali motivi per cui è stata introdotta la programmazione ad agenti è la necessità di accedere alle informazioni ed ai servizi di sistemi distribuiti. E' estremamente difficile che un Agente riesca da solo ad espletare completamente un servizio ad alto livello. Per questa ragione uno degli approcci possibili consiste nel considerare Agenti Ibridi, cioè Agenti che presentano caratteristiche tipiche di diverse tipologie, come la mobilità, la cooperazione, … In questa categoria rientrano, pertanto, tutti quegli Agenti che non è possibile classificare altrimenti.

1.5 APPLICAZIONI DEGLI AGENTI

Il campo di applicazione degli Agenti risulta particolarmente difficile da definire. Questo perchè la programmazione ad Agenti ha fatto la sua comparsa solo agli inizi degli anni novanta e tuttora non esiste uno standard a livello internazionale universalmente riconosciuto. Non è pertanto possibile delineare i confini di un ipotetico dominio applicativo, anzi, i ricercatori sostengono che la tecnologia ad Agenti Intelligenti abbia uno spettro di applicabilità estremamente ampio ed ancora

inesplorato. Tra le principali applicazioni che possono essere realizzate sfruttando il modello ad Agenti vanno ricordate [FIP00]:

1. ***User Assistant*** o Assistenti dell'Utente, che svolgono attività quali, per esempio: *Filtri Email*, *Agenda Elettronica*, *Recupero Informazioni*, *Pianificazione Viaggi* , … Si occupano cioè di tutta una serie di attività volte ad assistere l'utente in base alle sue abitudini, come una sorta di segretaria virtuale.
2. ***Information Retrieval*** (Directory Service, Information Brokerage, …): grazie agli agenti mobili, che migrano direttamente sui nodi contenenti le informazioni, queste possono essere acquisite in maniera più efficiente, soprattutto se le sorgenti di informazione sono molteplici e allocate in diversi punti della rete.
3. ***Intrattenimento*** (Games, Audio/Video Entertainment and Broadcasting,…): Grazie agli agenti si possono realizzare giochi e sistemi di personalizzazione e d'interfaccia audio/video finora impossibili da realizzare. Il sistema ad agenti conosce i gusti dell'utente e ricerca, per esempio, contributi audio e video, che corrispondano alle informazioni contenute nel profilo utente delineato.
4. ***Commercio Elettronico*** (Service Management, Information provision,…): nelle transazioni elettroniche il sistema ad Agenti si rivela molto efficace. Un'Agente può essere inviato in rete dall'utente con l'obiettivo di localizzare offerte convenienti, negoziare e concludere transazioni in propria vece. Inoltre con la tecnica del *Front-End* è possibile per i fornitori di servizi in rete aggirare i problemi legati ai browser internet ed all'installazione di programmi specifici. Quando sopraggiunge la richiesta di un servizio il fornitore invia direttamente al client un agente mobile contenente la logica necessaria per l'interazione con il servizio

stesso. L'eterogeneità di hardware e software dei possibili clienti, in questo modo, non rappresenta più un ostacolo alla creazione di servizi nuovi ed innovativi.

5. ***Network Management*** (Business Process Management, Workflow Management): anche nelle gestione di moderni sistemi di telecomunicazione il modello ad Agenti Intelligenti può risultare utile. A differenza di un sistema tradizionale, basato su una unità di controllo centralizzata, in un sistema ad Agenti è possibile per il gestore del servizio inviare Agenti Mobili contenenti la logica di Management direttamente al nodo della rete dove si trovano le risorse da gestire.
6. ***Mobile Computing***: nei dispositivi mobili, laptops, notebooks, personal communicators ed in tutti quei casi in cui l'utente non è costantemente connesso ad una rete di calcolatori, la tecnologia ad Agenti risulta particolarmente utile. Gli Agenti, infatti, possono operare in vece dell'utente, anche se sconnesso.
7. ***Service Robotics*** (Office Delivery Robots, House Cleaning Robots). Un altro dominio applicativo dove la tecnologia ad Agenti si rivela efficace è nella robotica, dove oggi viene sperimentata su robot ad uso domestico o per portare la posta o fare pulizie negli uffici. Queste macchine possono infatti comunicare tra loro, scambiarsi informazioni, cooperare per assolvere al meglio la loro funzione.

Molte delle applicazioni citate già esistono, anche se solo allo stadio embrionale. Altre sono state sviluppate con tecniche tradizionali che non utilizzano la tecnologia ad Agenti. Il valore aggiunto che tale tecnologia introdurrà in molti settori dipenderà , essenzialmente, dalle seguenti capacità degli Agenti [FIP00]:

1. Operare senza interventi esterni.
2. Interagire con l'uomo o con altri Agenti.
3. Percepire l'ambiente che li circonda e rispondere prontamente ai cambiamenti che si verificano in esso.
4. Prendere l'iniziativa per il raggiungimento di un determinato obiettivo.
5. Muoversi nell'ambiente e attraverso la rete.
6. Adattarsi ai cambiamenti nell'ambiente.

1.6 ESIGENZE DI STANDARDIZZAZIONE

La programmazione ad Agenti rappresenta un campo a differenti stadi di sviluppo. Mentre in alcune aree questa tecnologia si trova ancora ad un livello iniziale, in altre tale approccio ha già raggiunto un considerevole grado di maturità, tanto che numerosi prodotti hanno fatto uso degli Agenti fin dalla loro prima apparizione [FIP00].

In molti casi le cosiddette *Agent-Based Applications* richiedono Agenti capaci di interagire tra loro. A causa della vasta eterogeneità dei prodotti software e delle tecnologie utilizzate nell'implementazione di sistemi ad Agenti, la loro cooperazione ed interazione può riservare degli inconvenienti.

Quindi, per poter sviluppare prodotti, applicazioni o servizi aperti, in grado di interoperare tra loro, è necessario definire uno ***standard*** . Stabilire un modello di riferimento, che sia universalmente riconosciuto, non è facile, soprattutto perché lo stesso termine "standard" può assumere connotazioni distinte:

- ***Formal Standardization*** : è l'approccio tradizionale al concetto di standard, basato sul principio riconosciuto da tutti che quella presa

in esame rappresenta la soluzione migliore. Vi sono, comunque, numerose questioni burocratiche che rallentano questo processo, soprattutto in ambiti relativamente nuovi, come quello degli Agenti Intelligenti.

- ***Industry Standardization*** : si verifica quando una o più aziende rilasciano delle specifiche relative ad un nuovo tipo di prodotto. Approfittando della propria posizione di pioniere in un determinato settore impongono al mercato il proprio standard. Le società concorrenti, temendo di perdere clienti con la creazione di altri formati, rinunciano a nuove iniziative e si adeguano a quello che viene definito uno "*standard di fatto*".
- ***Industry Consortia Standardization*** : un gruppo di società specializzate in un determinato settore tecnologico si riuniscono per la definizione di un formato comune che possa soddisfare al meglio le esigenze di tutti.

Il caso degli Agenti Intelligenti risulta particolarmente complesso perché esso non pretende di delineare delle specifiche tecnologiche per un singolo dominio, ma di stabilire le regole generali per creare sistemi complessi e interoperanti basati su un'enorme varietà di domini, tecnologie e applicazioni.

In particolare si deve definire l'architettura di sistema, stabilendo come gli agenti comunicano tra loro e con l'ambiente circostante e la rete. E' quindi necessario costruire una metodologia sistematica, per specificare e strutturare applicazioni come sistemi multi-Agente. Soltanto nella seconda metà degli anni novanta alcuni team di ricerca ed alcune aziende hanno intrapreso un difficile percorso di studio volto alla creazione di uno standard sulla tecnologia ad Agenti.

La **DARPA** (Defence Advanced Research Project Agency) ha delineato un insieme di elementi considerati indispensabili per definire degli standard nel campo degli Agenti:

1. ***Agent Reference Model*** : va progettato un modello di riferimento per l'Agente ed il sistema che comprenda meccanismi di gestione e controllo, sicurezza, privacy, accesso e veridicità.
2. ***Agent-To-Agent Communication*** : un altro aspetto che non va trascurato concerne la comunicazione tra Agenti. Devono quindi essere definiti un linguaggio e dei protocolli di conversazione e condivisione delle informazioni.
3. ***Agent-Software communication*** : infine, si deve affrontare il problema dell interfacciamento tra software ad Agenti e software che, invece, non ne fanno uso.

Nonostante esistano numerosi consorzi ed associazioni che si occupano in modo diretto o indiretto degli aspetti sopra citati (*W3C*, *International Agents Society*, *IETF*, *Agent Society*,…) [AIW97][MAG98][MAN98], solo due stanno riscuotendo un notevole consenso: **FIPA** e **OMG**. Quest'ultimi rappresentano le due principali tendenze della comunità informatica sull'argomento. OMG (**Object Manager Group**), infatti, basa il suo modello di sistema ad Agenti sui concetti di "mobilità" e "Agente Mobile" focalizzando l'attenzione sulla migrazione di Agenti tra piattaforme caratterizzate dal medesimo profilo (stesso linguaggio, stesso sistema, stesse tecniche di autentificazione e di serializzazione), attraverso interfacce standard CORBA IDL [MAN98].

FIPA (**Foundation for Intelligent Physical Agents**), al contrario, lavora maggiormente sui concetti di "Agente Intelligente" e di cooperazione tra Agenti, cercando di definire formati e protocolli

generici per la comunicazione tra agenti e per la definizione di contenuti e ontologie [FIP00].

OMG è stato concepito per supportare un elevato livello di interoperatibilità tra sistemi eterogenei ad agenti. I suoi obiettivi fondamentali riguardano la standardizzazione di:

1. *Agent Management* : creazione, sospensione, riattivazione, terminazione di Agenti, …
2. *Agent Transfer* : meccanismi per l'invio e la ricezione di Agenti su piattaforme dalle caratteristiche similari,…
3. *Agent System Names* : tecniche di identificazione degli Agenti e dei sistemi ad Agenti,…
4. *Agent System Type and Location Syntax* : sintassi per la definizione della locazione di sistema ed Agenti.
5. *Agent Tracking* : tecniche di registrazione presso il sistema degli Agenti.

Non fanno parte, invece, degli obiettivi di OMG:

1. *Multi-Hop Security* : tecniche di sicurezza.
2. *Code Conversion* : nessun riferimento alla possibilità di spostare Agenti tra piattaforme con caratteristiche differenti.
3. *Agent Communication* : l'aspetto della comunicazione tra Agenti è lasciato agli sviluppatori del sistema. Non viene definito alcun tipo di protocollo o linguaggio.

OMG MASIF (*Mobile Agent System Interoperability Facility*) definisce un modello di riferimento che favorisce l'interoperabilità tra diverse implementazioni di sistemi ad Agenti, grazie ad un insieme di interfacce *object-oriented* basate sul linguaggio IDL (*Interface Definition*

Language), definito dallo stesso OMG. Gli aspetti sviluppati in questo progetto riguardano principalemente:

1. *Agent System* : una piattaforma che può creare, interpretare, eseguire, trasferire e terminare Agenti.
2. *Agent System Type* : descrive il produttore, il linguaggio di programmazione usato, i protocolli ed i meccanismi di serializzazione, …
3. *Place* : locazione e ambiente software in cui può essere allocata la piattaforma ad Agenti.
4. *Region* : set di sistemi ad Agenti caratterizzati dalla stessa "autorità" ma non necessariamente dello stesso tipo.

Altri aspetti che vengono affrontati sono i seguenti:

1. Livelli di interconnessione tra piattaforme basate su Java. Formati e capacità interne degli Agenti.
2. Interfacce IDL per la comunicazione a basso livello tra Agenti.
3. Collaborazione con FIPA. Supporto del linguaggio ACL per la comunicazione ad alto livello tra Agenti.
4. Perfezionamento dei concetti di "Region" e "Place".

Il lavoro di FIPA e OMG MASIF può essere considerato quasi complementare. Mentre nel primo assume maggiore rilevanza l'aspetto della comunicazione ad alto livello (ACL, protocolli di negoziazione, ontologie,…), nel secondo avviene lo stesso per quanto riguarda la mobilità.

Oggi i due gruppi collaborano attivamente per raggiungere la definizione di uno standard universalmente riconosciuto per la programmazione di Sistemi ad Agenti. Nel prossimo capitolo verranno

analizzate nel dettaglio alcune delle caratteristiche di FIPA più significative per questo lavoro di tesi.

Capitolo 2

FIPA: FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS

Riuscire a dare un quadro completo del lavoro di FIPA e del contributo che essa dà allo sviluppo di uno standard a livello internazionale per la creazione di sistemi software ad Agenti Intelligenti, risulta alquanto arduo. Per questo motivo in questo capitolo vengono affrontate essenzialmente quelle tematiche di maggiore interesse per lo sviluppo di questo lavoro di Tesi quali, per esempio, l'architettura del sistema, l'interazione tra uomo e Agente (UDMA), l' Agent Management System (AMS), l' Agent Communication Language (ACL) ed i vari modelli di riferimento.

2.1 FIPA – L'Architettura

La Fondazione per gli Agenti Fisici Intelligenti (FIPA) è una associazione internazionale no-profit nata a Ginevra nel 1996 e composta da varie società ed organizzazioni con l'obiettivo di definire uno standard riguardante le tecnologie ad Agenti. A gennaio 2001 i membri di FIPA risultano essere 65, per 21 paesi nel mondo. Lo scopo finale non è tanto quello di creare una tecnologia adatta a particolari applicazioni ma, principalmente, quello di generare dei modelli di riferimento che possano essere utilizzati dagli sviluppatori per l'implementazione di sistemi complessi, nelle aree e nei settori più disparati. Inoltre, uno degli aspetti

che più preme ai ricercatori di FIPA è il raggiungimento, utilizzando tali modelli, di un elevato livello di interoperabilità tra tecnologie differenti.

La prima documentazione rilasciata, denominata *FIPA97 Specifications,* risale al 1997 e delinea le regole di base per la gestione di una comunità di Agenti, sottolineando gli aspetti relativi alla loro collaborazione. La prima parte di tali specifiche tecniche, denominate *NORMATIVE* , comprende:

- ***Spec. 01 – Agent Management***: viene definita una infrastruttura generica in cui gli Agenti FIPA possono esistere, operare ed interagire. Vengono, inoltre, descritte delle entità di tale infrastruttura, necessarie all'esecuzione, alla comunicazione ed alla gestione degli Agenti stessi.
- ***Spec. 02 – Agent Communication Language***: questa specifica prende in esame le caratteristiche basilari di un linguaggio per Agenti strutturato secondo la *Speech Act Theory* (vedi paragrafo 2.6).
- ***Spec. 03 – Agent Software Integration*** : vengono descritti alcuni servizi utili all'integrazione di programmi applicativi già esistenti con i sistemi ad Agenti.

A queste si aggiungono le cosiddette specifiche *INFORMATIVE* che si occupano di descrivere alcuni test esemplicativi dell'uso di FIPA in applicazioni commerciali.

- ***Spec. 04 – Personal Travel Assistant***: integrazione di servizi eterogenei per provvedere ad un supporto automatizzato e personalizzato all'utente nella pianificazione ed esecuzione di itinerari di viaggio.

- ***Spec. 05 – Personal Assistant***: supporto personale al lavoro d'ufficio tramite la creazione di profili utente che permetta di agevolare e rendere più efficace il raggiungimento di determinati obiettivi.
- ***Spec. 06 – Audio/Visual Entertainment and Broadcasting***: automazione e personalizzazione dei servizi di intrattenimento.
- ***Spec. 07 – Network Management & Provisioning***: automazione dei processi di configurazione e negoziazione nella fornitura di servizi in rete.

In questi documenti viene definito un modello di riferimento per la piattaforma ad Agenti (mostrato in fig. 2.1) specificando il ruolo chiave ed indispensabile di alcuni Agenti nell'amministrazione della stessa.

Le entità fondamentali per il corretto funzionamento della piattaforma (o AP, si tratta di una macchina virtuale per l'esecuzione degli Agenti) sono tre e sono indicate nelle *Spec.01*:

1. ***AMS o Agent Management System***.
2. ***ACC o Agent Communication Channel***.
3. ***DF o Directory Facilitator***.

L'***Agent Management System*** (AMS) è l'Agente che esercita il compito di supervisore, controllando l'accesso e l'uso della piattaforma. E' responsabile per l'identificazione e la registrazione degli Agenti residenti. La Piattaforma (AP) è un sistema software composto da uno o più contenitori di Agenti, i quali devono appartenere, in ogni istante, ad una ed una sola AP, con la facoltà, se necessario, di spostarsi da una all'altra.

L'***Agent Communication Channel*** (ACC) utilizza le informazioni dell'AMS (riguardanti lo stato, l'identità e la locazione degli Agenti) per gestire lo scambio di messaggi tra Agenti all'interno o all'esterno della piattaforma. Per l'interoperabilità tra Agenti situati su piattaforme differenti è indispensabile che l'ACC supporti il protocollo IIOP (Internet Inter-ORB Protocol, vedi capitolo 4, paragrafo 4.5). AMS e ACC sono unici ed indispensabili per ogni AP.

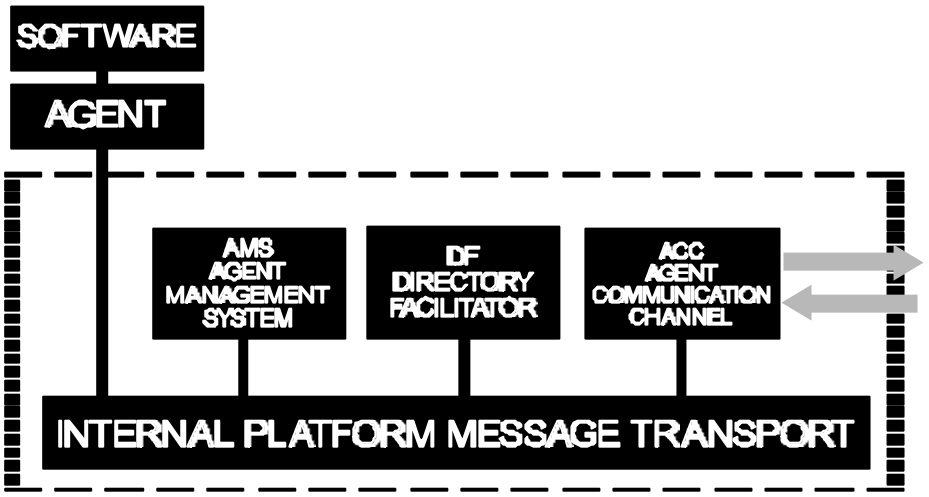

Fig. 2.1 : Modello di riferimento per una piattaforma ad Agenti basata sulle specifiche FIPA97-98.

Infine, il ***Directory Facilitator*** (DF) fornisce un servizio tipo "Pagine Gialle" alla piattaforma. Esso, infatti, tiene traccia della descrizione, delle caratteristiche, delle funzionaltà e della locazione di tutti gli agenti registrati. In ogni AP deve esistere almeno un Agente DF che, tra l'altro, è abilitato al mantenimento di collegamenti (link) ad altri Agenti situati su altre APs.

Un aspetto importante che va evidenziato è che nessuna restrizione è stata posta sulla tecnologia impiegata per realizzare la piattaforma. Esistono esempi di piattaforme basate sullo scambio di e-mail, oppure

CORBA-based, applicazioni Java Multi-Thread e tante altre ancora , che supportano le specifiche FIPA.

Dato che lo scambio di messaggi è uno degli aspetti fondamentali perché alla base della comunicazione tra Agenti, si è resa necessaria la definizione di un linguaggio univoco chiamato **ACL** (***Agent Communication Language***) e di cui si tratterà nel paragrafo 2.6.

Lo standard supporta forme comuni di conversazione tra Agenti attraverso la specifica di protocolli di interazione, rappresentanti i possibili scenari di scambio messaggi/informazioni tra due o più Agenti. Tra questi, per esempio, i protocolli *Query*, *Request* e *Net Negotiation*.

FIPA descrive, inoltre, gli aspetti che riguardano l'*integrazione* tra software ad Agenti e software classico, *mobilità* e *sicurezza*, *ontologie*, *interazione uomo-Agente.* Nell anno 1998, infatti, fanno la loro comparsa le specifiche relative a questi argomenti, che vanno ad integrarsi a quelli della documentazione già definita:

- ***Spec. 08 – Human-Agent Interaction***: servizi per la gestione di dialoghi multi-modali con l'uomo e per la creazione di profili utente adattativi, utili ai processi di personalizzazione dell'interfaccia.
- ***Spec. 10 – Agent Security Management***: gestione della sicurezza della AP attraverso un Agente opportuno, chiamato Agent Platform Security Manager (APSM). Questo si occupa di monitorare il comportamento degli agenti all'interno della piattaforma e quelli mobili nei loro spostamenti, affinchè non vengano violati i protocolli di sicurezza.
- ***Spec. 11 – Agent Management Support for Mobility***: gestione della mobilità degli Agenti tra differenti piattaforme, in base a

standard già esistenti sull'argomento (*OMG Mobile Agent System Interoperability Facility*).

- ***Spec. 12 – Ontology Service***: metodologie che abilitano gli agenti a creare ed amministrare direttamente le ontologie, scoprendone di nuove ed elaborandole con quelle preesistenti riconoscendo relazioni, analogie e differenze , nonché effettuando traduzioni dove necessario.
- ***Spec. 13 – FIPA'97 Developers Guide***: una guida completa ed esauriente alla comprensione dell'argomento "Agenti Intelligenti" e delle specifiche rilasciate da FIPA.

Nel corso del 1999 fanno la loro comparsa bozze di numerose altre caratteristiche per un sistema ad Agenti e concernenti gli argomenti più vari. La mole di documentazione diviene così vasta (più di ottanta specifiche) che FIPA è costretta ad interrompere la pubblicazione di nuovi atti e a provvedere ad un lungo processo di rielaborazione di quanto prodotto. Questa operazione termina con il rilascio, alla fine dell anno 2000, delle nuove specifiche FIPA2000.

2.2 FIPA2000

FIPA2000 è il nome dato al nuovo set di specifiche FIPA approvato nell'Ottobre 2000. Le modifiche apportate al progetto sono rilevanti e modificano molti dei concetti espressi nella documentazione precedente .

In particolare, i maggiori cambiamenti riguardano [FIP00]:

- ***Architettura e Gestione della Piattaforma ad Agenti***.
- ***Codifica e Trasporto dei Messaggi***.
- ***Agent Management Ontology***.

- ***Sintassi ACL***.

Uno degli aspetti più innovativi introdotti vede come protagonista l' **ACC** (*Agent Communication Channel*) che, contrariamente alle versioni precedenti, non è più un Agente. Diviene un'entità integrata nella piattaforma, che supporta il **MTS** (*Message Transport System*), cioè il servizio di comunicazione di default tra agenti su differenti APs (Agent Platforms). Ora l'ACC è in grado di eseguire ***Routing Tasks*** tra piattaforme.

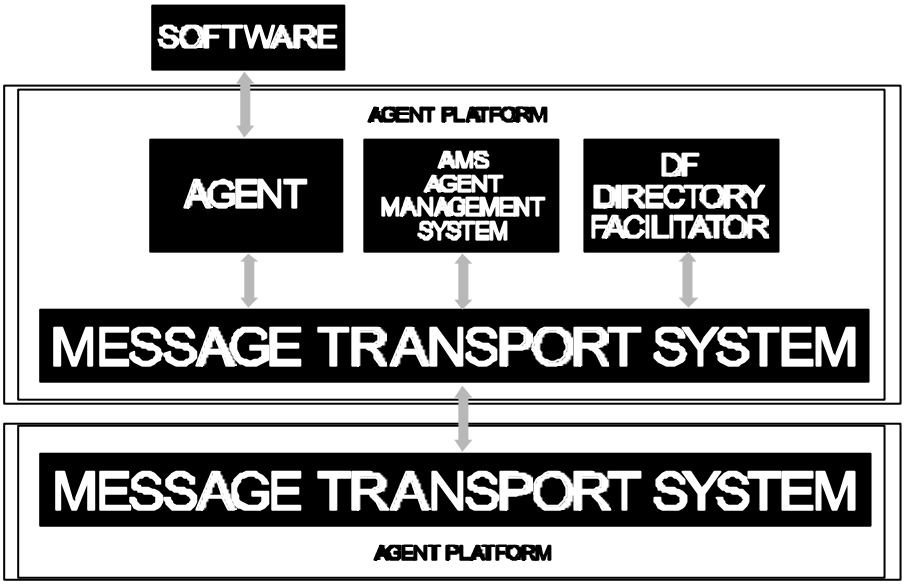

Fig. 2.2 : Modello di riferimento per una piattaforma ad Agenti basata sulle specifiche FIPA2000.

In figura 2.2 è rappresentato il nuovo modello di AP.

Anche la tecnica di *Agent Naming* subisce delle variazioni. Ora FIPA identifica un Agente attraverso una collezione estendibile di *coppie Parametro/Valore*, chiamata ***Agent Identifier*** (**AID**).

Un AID comprende:

- Un nome.
- Altri termini, tra cui: tutti gli indirizzi (*IIOP*, *WAP*, *HTTP*) che indicano dove è situato fisicamente l'Agente, su quale Host, quale Piattaforma e quale contenitore, necessari per le comunicazioni ed la migrazione. Inoltre vengono indicati anche i *Resolvers*, cioè tutti quegli Agenti, come l'AMS, i DFs, ecc… dove l'Agente in questione è registrato. Sono supportati altri parametri, a discrezione del progettista del sistema applicativo.

Solo il nome di un Agente è invariabile mentre gli altri parametri possono essere modificati durante il corso della sua "vita". Inoltre, un Agente supporta differenti tecniche di comunicazione ed è in grado di memorizzare indirizzi multipli (indispensabili per gli spostamenti tra contenitori o piattaforme) nel parametro *:addresses* di un AID.

L'introduzione della rappresentazione tramite AID ha comportato una ridefinizione delle azioni dell'AMS e del DF che, ora, hanno a disposizione un sottoinsieme comune di metodi:

- ***Register*** : realizza le operazioni di registrazione di un'Agente.
- ***Deregister*** : realizza le operazioni di deregistrazione di un'Agente.
- ***Search***: realizza le operazioni di ricerca di un'Agente.
- ***Modify***: realizza le operazioni di modifica dei parametri descrittivi di un'Agente.

Anche la struttura dei messaggi è stata cambiata notevolmente. Nelle nuove specifiche, infatti, sono costituiti da due parti:

- Una parte è composta da una serie di informazioni rilevanti per la consegna ed il trasporto (***Message Envelope***).
- L'altra contiene il corpo del messaggio ACL.

Per quanto riguarda l'interpretazione di un messaggio da parte di un Agente, la semantica ACL viene definita solo per messaggi ACL ed integrata nel corpo del messaggio, mentre nulla stabilisce FIPA relativamente al contenuto informativo ed al suo utilizzo. C'è poi la possibiltà di scelta, a livello ACL, tra tre differenti tecniche di codifica dei messaggi:

- ***String Encoding***.
- ***Bit-Efficient Encoding***.
- ***XML***.

Un'altra novità è che IIOP (*Internet Inter-ORB Protocol*) non costituisce più l'unico protocollo per il trasporto dei messaggi tra piattaforme diverse. Sono stati aggiunti i protocolli:

- **WAP** (*Wireless Access Protocol*).
- **HTTP** (*Hyper-Text Transfer Protocol*) .

Ognuno di questi supporta i tre meccanismi di codifica dei messaggi citati precedentente. Sono stati introdotti, sempre relativamente alla gestione dei messaggi, due nuovi metodi:

- ***Reply-To*** : effettua l'invio di un messaggio di risposta nel caso di conversazione tra Agenti.
- ***Encoding*** : per la scelta della tecnica di codifica.

La documentazione completa di FIPA2000 è consultabile gratuitamente al sito web della Fondazione. Nel prossimo paragrafo verranno analizzati nel dettaglio alcuni aspetti importanti come, per esempio, il funzionamento e le proprietà dell'AMS.

2.3 Agent Management System (AMS)

L'Agent Management System è un componente indispensabile ed unico di una AP ed è responsabile di tutte le operazioni di gestione che la riguardano, come la creazione/cancellazione di Agenti ed il controllo della migrazione di Agenti tra APs.

Dato che esistono APs con svariate capacità, l'AMS è in grado di generare una decrizione della propria piattaforma, accessibile tramite il metodo di *Get-Description*. Inoltre, mantiene in memoria l'intero ciclo di vita di ogni Agente.

L'AMS rappresenta una specie di unità di controllo all'interno di una AP. Può richiedere ad un agente di svolgere uno specifico compito, per esempio, *Quit* (cioè, termina tutte le azioni in corso sulla AP) e, se necessario, è in grado di forzare l'esecuzione di un particolare *Task*.

Mantiene un indice di tutti gli Agenti presenti in ogni istante sulla piattaforma, compresi i loro AID. In accordo con le specifiche FIPA tutti gli Agenti devono registrarsi presso l'AMS della propria **HAP** (***Home Agent Platform***), cioè la AP dove sono stati creati e responsabile dell'identità degli stessi Agenti. La registrazione comporta l'autorizzazione, previa fase di autentificazione, ad accedere al **MTP** (*Message Transport Message*) per svolgere le mansioni di invio e ricezione di messaggi.

Le descrizioni degli Agenti possono essere modificate in qualsiasi istante e per molti motivi, sempre sotto l'occhio vigile dell'AMS. La vita

di un'Agente presso una AP termina con la sua deregistrazione. In questa fase il suo AID diviene disponibile per essere assegnato, eventualmente, ad un altro Agente.

Trattandosi di un'Agente, anche l'AMS possiede un AID, riservato e rappresentato nella forma seguente:

```
(agent-identifier
:name ams@hap
:addresses (sequence hap_transport_address))
```

Esistono due modi con cui un'Agente si registra presso l'AMS:

- L'Agente viene creato sulla AP.
- L'Agente è stato creato su un'altra AP ed è migrato su quella corrente. Vale solo per le piattaforme che supportano gli Agenti Mobili.

2.3.1 Funzioni supportate dall'AMS

In aggiunta alle *Management Functions*, descritte nel paragrafo 2.2, l'AMS esercita il suo ruolo direttivo attraverso questi metodi:

- ***Suspend Agent*** : sospende l'esecuzione di un Agente.
- ***Terminate Agent*** : pone fine alla vita di un Agente.
- ***Create Agent*** : crea un nuovo Agente ma non lo attiva.
- ***Resume Agent Execution*** : riprende l'esecuzione dallo stato in cui è avvenuta l'interruzione, con il metodo *Suspend*.

- ***Invoke Agent*** : chiamata per un agente, non necessariamente sulla stessa AP.
- ***Execute Agent*** : attiva l'Agente precedentemente creato con l'istruzione *Create.*
- ***Resource Management*** : gestisce le risorse della piattaforma in base agli Agenti disponibili, capacità e funzionalità della stessa AP.

2.3.2 Ciclo di Vita di un Agente

Un'Agente FIPA esiste solo in un contenitore d'Agenti presso un'apposita piattaforma in esecuzione. Senza di esso non ha senso parlare di Agente. Dato il modello con cui è stato definito, un Agente viene rappresentato attraverso il suo ciclo di vita. Questo ciclo, mostrato in figura 2.3, è gestito dall'AMS della piattaforma stessa.

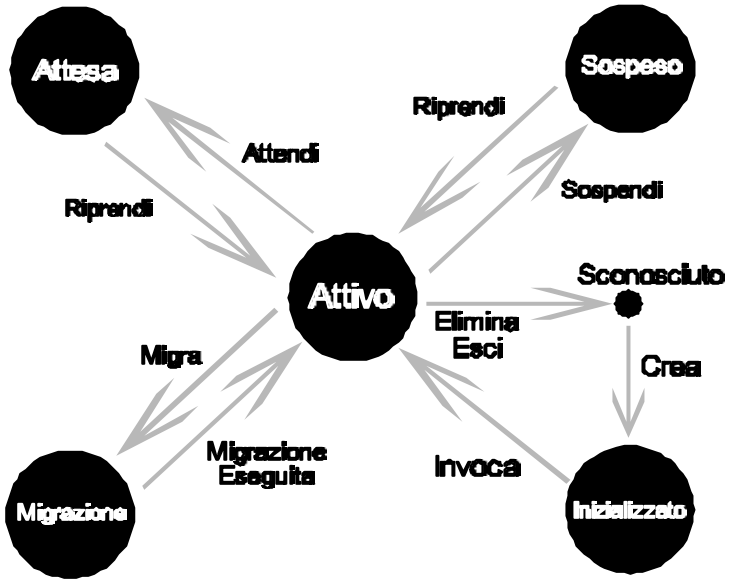

Fig. 2.3 : Ciclo di Vita dell'Agente sulla Piattaforma.

2.4 Directory Facilitator (DF)

Quando un Agente desidera far conoscere le proprie funzionalità ad altri Agenti, si serve del Directory Facilitator o DF.

Il DF mantiene una lista di Agenti dando origine ad un **AD** (*Agent Domain*), cioè un dominio, un insieme di Agenti e servizi. L'utilità del DF consiste nella possibilità di informare il sistema, l'AMS ed altri Agenti (intra ed extra AP), dell'esistenza e delle funzionalità degli Agenti iscritti. Infatti viene definito un metodo di *search* che permette agli Agenti di consultare il DF sull'esistenza e sulle caratteristiche degli Agenti della AP.

2.5 Message Transport Service (MTS)

Il modello di riferimento su cui poggia il Message Transport Service comprende tre livelli distinti ed è raffigurato in fig. 2.4 [specifica FIPA XC00067C]:

- MTP (*Message Transport Protocol*) è usato per effettuare il trasferimento "fisico" di messaggi.
- MTS è un servizio offerto dalla piattaforma e supporta il trasporto di messaggi ACL tra Agenti all'interno ed all esterno della AP.
- Il "contenuto" dei messaggi viene scritto in ACL.

FIPA2000 ha stabilito che ogni MTP può usare una differente rappresentazione interna per definire il ***Message Envelope***, ma deve descrivere gli stessi termini e possedere la stessa semantica.

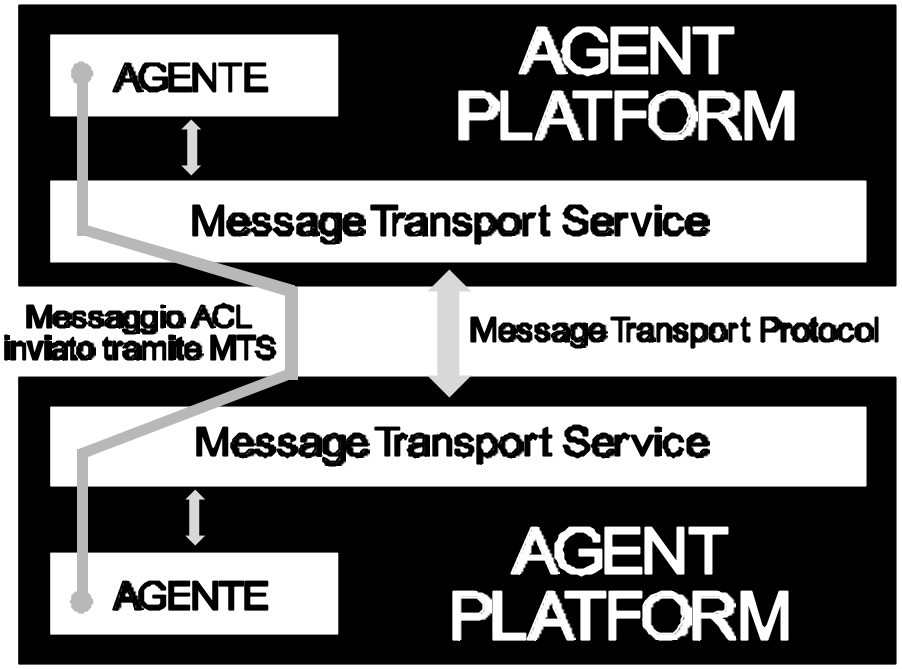

Fig. 2.4 : Modello di riferimento per il Trasporto dei Messaggi.

Gli aspetti generali relativi alla sua forma sono i seguenti:

- Un *Message Envelope* comprende una collezione di parametri.
- Un parametro è una coppia *Nome/Valore*.
- Un *Message Envelope* contiene almeno i campi *:to, :from, :date, :acl-representation.*
- Sono consentiti parametri opzionali.

L'ACC (è integrato nella AP) ha la facoltà, durante l'elaborazione di un messaggio, di aggiungere nuove informazioni al Message Envelope , ma non ha il permesso di sovrascrivere quelle già esistenti.

Le interfacce standard MTP di un ACC vengono utilizzate per lo scambio di messaggi tra piattaforme FIPA-compatibili.

Un Agente che vuole inviare un messaggio ad un suo simile su una AP remota ha a disposizione tre possibilità (vedi figura 2.5):

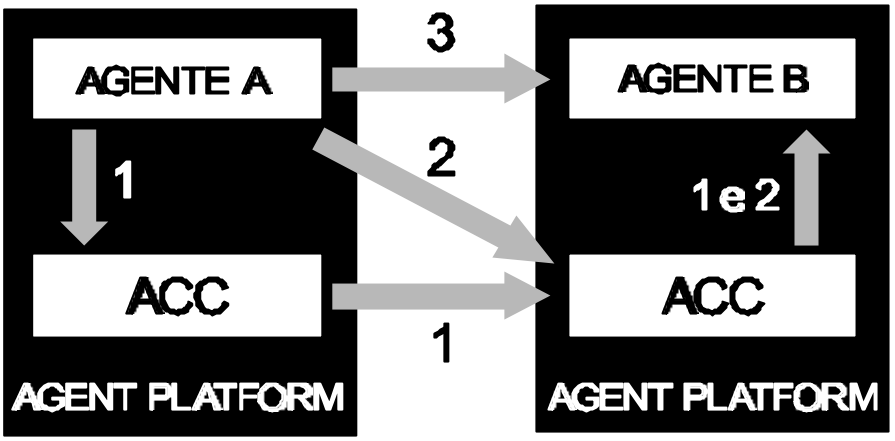

Fig. 2.5 : I tre metodi di comunicazione tra Agenti su piattaforme diverse.

1. L'Agente A invia il messaggio al suo ACC locale utilizzando l'interfaccia standard o quella specifica della AP. L'ACC si incarica di spedirlo al corrispondente ACC remoto, servendosi del MTP più opportuno. L'ACC remoto, infine, consegnerà il messaggio all'Agente destinatario B.
2. L'Agente A spedisce il messaggio direttamente all'ACC della AP remota dove risiede l'Agente destinatario B. Per fare ciò l'Agente A deve avere accesso ad almeno una delle interfacce MTP verso ACC remoti.
3. L'Agente A invia il messaggio direttamente all'Agente B, utilizzando un meccanismo di comunicazione diretta. Di questa tecnica FIPA non si occupa.

2.6 Agent Communication Language (ACL)

L'ACL di FIPA (specifica PC00037F, [FIP00]) è un linguaggio basato sulla "Teoria degli Atti Comunicativi" (***Speech Act Theory***), derivata dall'analisi linguistica della comunicazione umana e incentrata sull'idea che tramite il dialogo non solo si esprima un'affermazione ma, contemporaneamente, si compia una vera e propria azione. Questo significa che, se l'espressione utilizzata è tale da inglobare in sé lo svolgimento di un'azione, si parla di Speech Act. Questo modello di rappresentazione risulta particolarmente intuitivo e permette di definire protocolli d'interazione ad alto livello, quali, per esempio : *Request Protocol, Query Protocol, Contract-Net*, *Auction-English Protocol,…*

Tali protocolli sono l'insieme delle regole alle quali gli interlocutori devono attenersi, affinchè le loro conversazioni assumano un preciso significato.

L'intento di FIPA con la definizione di ACL è quello di gettare le basi per un linguaggio comune fra gli Agenti ed una semantica indipendente dalla struttura interna. FIPA non si occupa dei protocolli di rete e di trasporto a basso livello dell'architettura di comunicazione in un sistema distribuito, ma si assume l'esistenza di tali servizi ed il loro utilizzo per la realizzazione degli "Atti Comunicativi".

2.7 L'Interazione Uomo-Agente (UDMA)

Nonostante in questo lavoro di tesi non siano state affrontate tematiche relative alla personalizzazione dell'interfaccia tra uomo e Agente e, quindi, alla realizzazione di *Personal Assistants,* un aspetto molto importante che è stato studiato riguarda la comunicazione tra uomo e sistema ad Agenti. In particolare, lo scambio reciproco di informazioni.

Secondo le specifiche FIPA (Fipa8a24, [FIP00]) gli Agenti possono:

- Manipolare informazioni sugli utenti, per comunicare con gli stessi o con i propri simili.
- Gestire il dialogo fra l'Agente e l'uomo attraverso l'uso di apposite interfacce (*UI* o *User Interface*).
- Acquisire (direttamente, da altri Agenti oppure tramite apprendimento), mantenere ed estrarre le informazioni sugli utenti che sono richieste per un'interazione personalizzata.

E' indispensabile distinguere, in questo approccio, tra i diversi ruoli corrispondenti a vari tipi di Agente. Si prospetta la necessita di Agenti preposti alla gestione del dialogo utente (*Udma o User Dialog Management Agent*) e Agenti specializzati nella personalizzazione (*Upa o User Personalization Agent*). FIPA, in proposito, definisce due entità generiche:

- ***UDMS*** o ***User Dialog Management Service*** : è il servizio dedicato alla gestione del dialogo con l'utente; incorpora molti tipi di componenti software necessari all'interfacciamento utente-agenti.
- ***UPS*** o ***User Personalization Service***: si occupa della registrazione, mantenimento ed accesso alle informazioni riguardanti l'utente, necessarie per la personalizzazione. Queste informazioni possono essere ricavate osservando i comportamenti dell'utente, costruendo dei profili aggiornabili e memorizzando il tutto in Data-Base.

Il modello di riferimento per l'interazione Uomo-Agenti è mostrato in fig. 2.6. L' UDMS fornisce da un lato l'interfaccia con l'utente (UDMA) e dall altro quella rispetto agli Agenti. L'UDMS funge da traduttore tra il

mondo degli uomini e quello degli Agenti, permettendo a questi ultimi di trattare l'utente come uno qualsiasi degli altri Agenti del sistema.

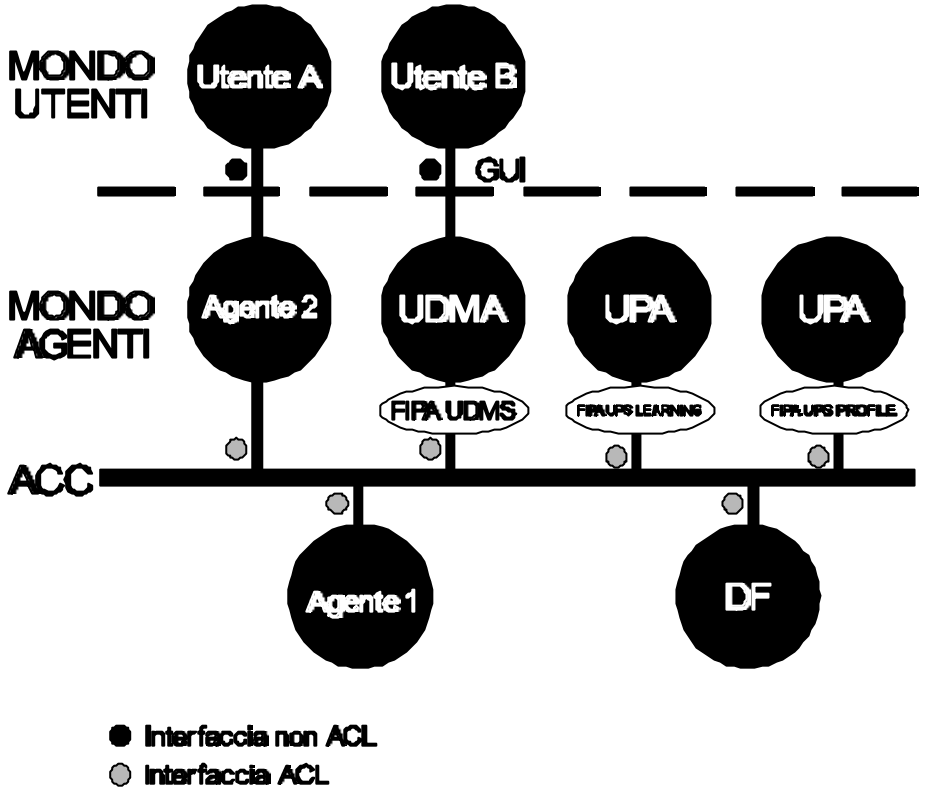

Fig. 2.6 : Modello di riferimento per l'interazione Uomo-Agenti.

In fig. 2.6 si vede come l'Agente UDMA, comunichi da un lato con l'utente e dall altro con gli Agenti (tramite MTS). In particolar modo è in grado di collaborare con l'Agente UPA cosi' da ricavare il modello utente utile alla fornitura di servizi ed interfacce personalizzate. Tra le azioni supportate dall' UDMS vi sono:

- *Present*: mostra le informazioni all'utente.
- *Listen*: accetta l'input dell'utente.
- *Query-User*: fa una richiesta all'utente e riceve la risposta.

- *Query-if-User*: fa una richiesta all'utente e riceve la risposta sia nel caso sia vera che falsa.
- *Start-Conversation*: apre un canale di comunicazione fra l'utente e l'UDMA.
- *Stop-Conversation*: chiude il canale di comunicazione tra utente e UDMA.
- *Identify-User*: identifica l'utente, o gli utenti, che sono collegati, all'istante considerato, con l'UDMA.
- *Detect-User*: Segnala quando un utente, o più utenti, possono interagire.

L'Agente a cui è stato affidato il compito di gestire la comunicazione tra Utente e Sistema ad Agenti è l'UDMA (può non essere unico). Esso risulta pienamente operativo non appena si registra presso il Directory Facilitator (DF) della piattaforma e, in genere, possiede capacità e servizi addizionali. Anche in assenza di un sistema di personalizzazione (UPS) il ruolo dell'UDMA risulta insostituibile. Di seguito vengono riportati alcuni esempi di UDMA, tratti dalle specifiche FIPA.

2.7.1 Scenario 1

In figura 2.7 viene mostrato il modello base di un Agente UDMA. In questo scenario vi è un solo utente ed un solo UDMA. Il dialogo viene iniziato da un agente. Nel capitolo 5 di lavoro di tesi verranno affrontate le motivazioni che hanno portato all'utilizzo della soluzione inversa, cioè con l'utente ad iniziare la conversazione.

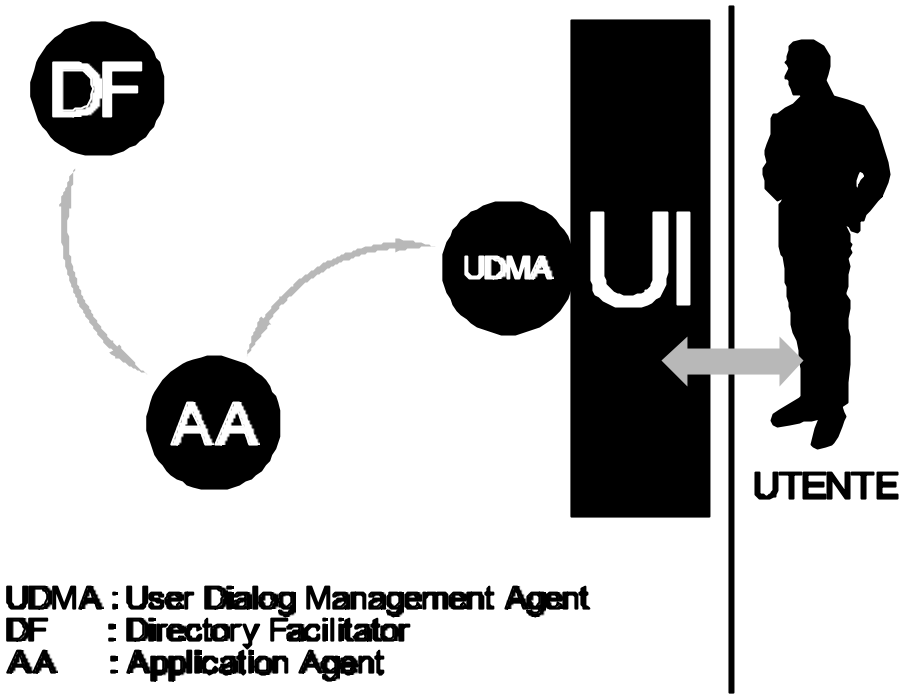

Fig. 2.7: Scenario 1. Un UDMA gestisce il dialogo con un utente.

- L'Agente di Applicazione (AA) richiede al DF, attraverso quale agente deve comunicare con l'utente.
- Il DF ritorna l'UDMA appropriato.
- L'AA contatta l'UDMA e gli spedisce il contenuto informativo da mostrare all'utente.

L'UDMA è in grado di comunicare con l'utente solo se supporta il formato specificato per il contenuto dell'informazione. Tale supporto viene fornito dall'Interfaccia Utente (UI o GUI, in fig. 2.6) e, per esempio, concerne la generazione di pagine HTML, XML, JSP o ASP,...

2.7.2 Scenario 2

Nello scenario mostrato in figura 2.8 si considera il caso in cui vi siano più UDMA e che lavorino tutti per lo stesso utente. Inoltre, si suppone che ogni UI sia diverso (ognuno supporta tecniche diverse, quali voce, testo, video, audio, ecc.) e che ce ne sia almeno uno per UDMA.

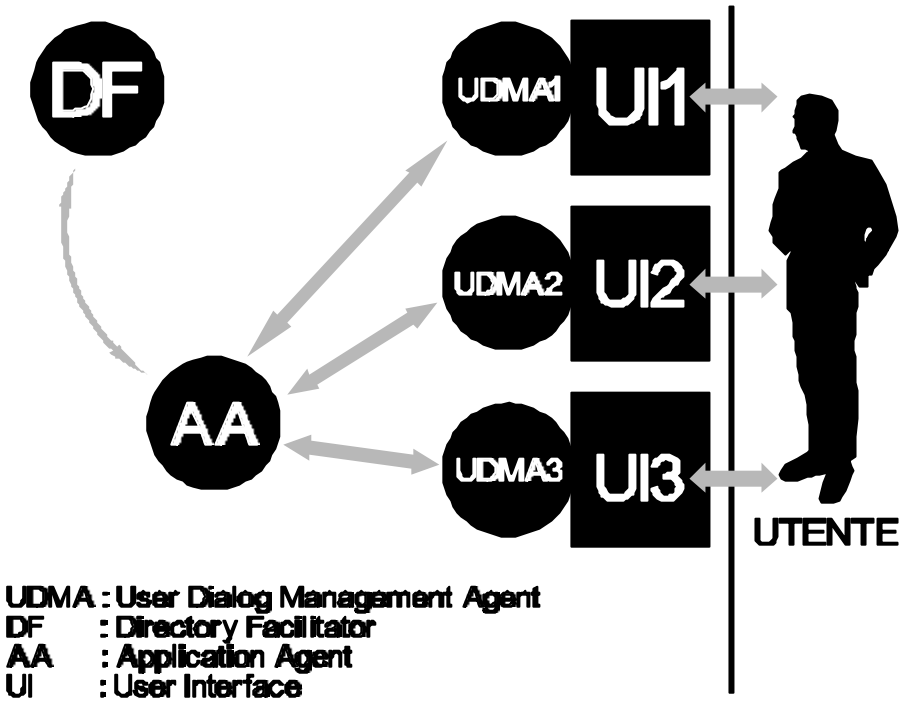

Fig. 7.8 : Scenario 2. UDMA multipli per un utente singolo.

- L'AA richiede al DF un agente che sia in grado di interagire con l'utente con una determinata modalità di output (audio, video,…).
- Il DF ritorna l'UDMA, o gli UDMA, che rispondono a queste caratteristiche.
- Se viene ritornato un singolo UDMA, si torna al caso dello scenario precedente. L'AA usa quell'UDMA.

- In caso vengano ritornati più UDMA, l'AA deve compiere una scelta basata su propri parametri, quali urgenza, gerarchia, costi, ecc.

2.7.3 Scenario 3

Il terzo scenario preso in considerazione prevede una struttura decisamente più complessa, a causa dell'inserimento di un UDMA Broker, più altri UDMA, che gestiscono un singolo utente (vedi fig. 2.9).

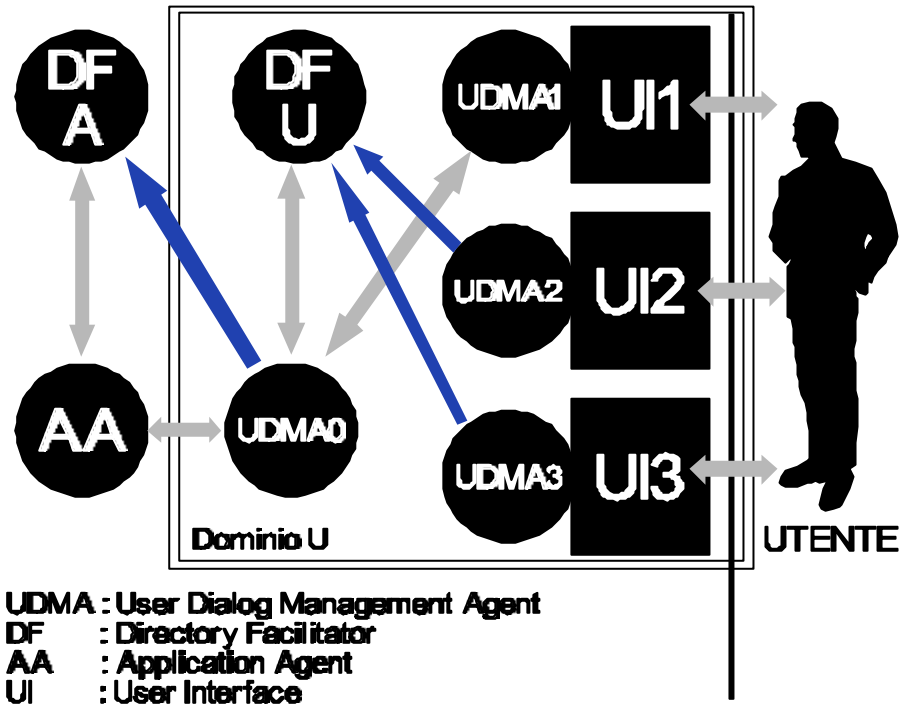

Fig. 2.9 : Scenario 3. UDMA multipli con un Broker.

In questo scenario l'Agente Broker (UDMA-0) coordina gli altri UDMA, che insieme formano un dominio detto Dominio-U. Essi sono registrati presso il DF-U mentre solo l'UDMA-0 viene registrato presso il

DF esterno (DF-A). Per interagire con l'utente, l'Agente di Applicazione (AA) si serve dell'UDMA-0 che, tra l'altro, non supporta interfacce UI.

- L'AA chiede al DF-A un'Agente per interagire con l'utente.
- Il DF-A ritorna l'UDMA-0.
- L'AA richiede l'interazione con l'utente all'UDMA-0.
- L'UDMA-0 chiede al DF-U quali UDMA forniscano questo servizio.
- Il DF-U ritorna l'UDMA appropriato (es. UDMA-1).
- L'UDMA-0 richiede all'UDMA-1 di mostrare il video.

L'UDMA-0 può utilizzare più UDMA in parallelo, o usarli alternativamente, per rendere la comunicazione più efficace.

2.7.4 Scenario 4

In questo ultimo esempio viene analizzato uno scenario composto da più UDMA al servizio di più utenti (fig. 2.10). L'Agente AA si comporta come nel caso dello scenario 2, ma l'UDMA deve definire come registrare i propri utenti presso il DF. Anche l'AA deve controllare chi può instaurare l'interazione in base ad autentificazione dell'utente.

- L'AA richiede al DF un Agente che sia in grado di interagire con l'utente con una determinata modalità di output (audio, video, testo,…).
- Il DF ritorna l'UDMA, o gli UDMA, che rispondono a queste caratteristiche.
- Deve esistere una tecnica che permette l'identificazione dell'utente.

- Ogni utente interagisce con l'Agente AA attraverso uno/più UDMA a seconda dell'interfaccia UI supportata e necessaria alla comunicazione uomo-Agente.

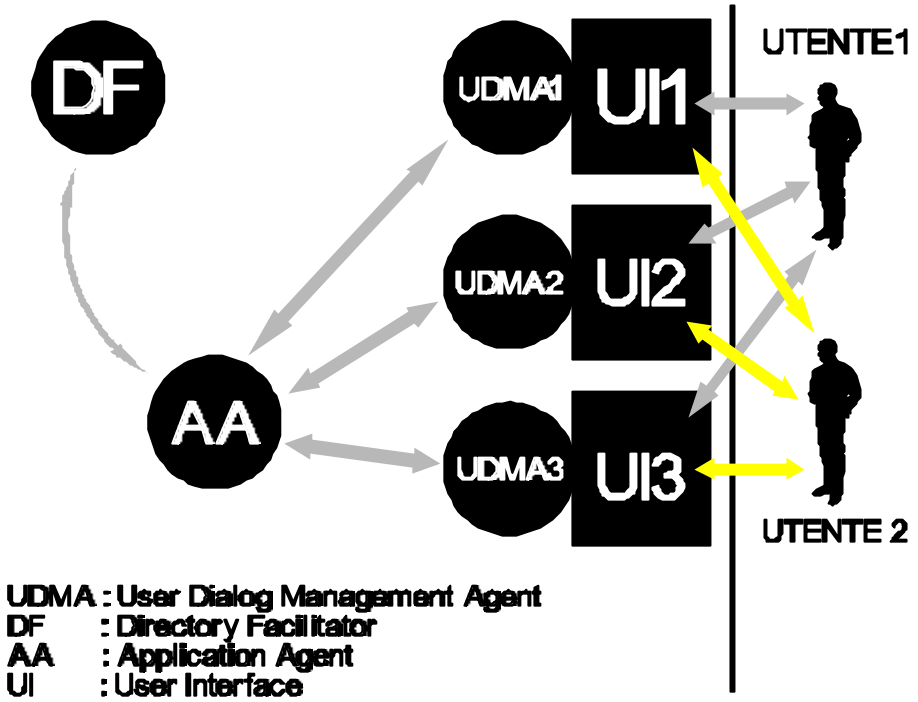

Fig. 2.10 : Scenario 4. UDMA multipli per più utenti.

Alla luce di questi esempi risulta evidente come FIPA non abbia pensato a situazioni in cui è l'utente ad iniziare l'interazione con il sistema ad Agenti. In realtà, la nuova versione (facente parte di FIPA2000) delle specifiche riguardanti la *Human-Agent Interaction* è ancora in lavorazione. In questo lavoro di tesi, infatti, come verrà meglio illustrato negli scenari del capitolo 5, è l'utente che attiva l'agente di interfacciamento con il sistema multi-Agente.

Nel prossimo, invece, si tratterà dei principali ambienti software e librerie per lo sviluppo di sistemi ad Agenti.

Capitolo 3

AMBIENTI SOFTWARE E LIBRERIE PER LO SVILUPPO DI SISTEMI AD AGENTI

Negli ultimi anni sono apparse diverse librerie e tools di sviluppo per creare sistemi multi-agente. Di seguito vengono analizzati alcuni dei più noti tra questi, confrontandoli sulla base delle seguenti caratteristiche: architettura di sistema, mobilità, comunicazione e sicurezza.

3.1 Telescript

Telescript [WHI95] è stato prodotto agli inizi degli anni Novanta dalla General Magic, ed è stato il primo sistema espressamente progettato per applicare la tecnologia ad agenti mobili. Nonostante non abbia avuto successo dal punto di vista commerciale e praticamente non esista più, esso ha una notevole importanza storica, essendo stato il pioniere di tutti i sistemi ad agenti mobili attuali.

- ***Architettura di sistema***: I server, chiamati posti (*places*), offrono servizi installando alcuni agenti statici, pronti ad accogliere e interagire con gli agenti mobili che approdano sulla locazione.
- ***Mobilità***: Per spostarsi gli agenti possono specificare la destinazione *assoluta*, sfruttando la primitiva *go* ed introducendo l'indirizzo del posto in cui si vogliono muovere, oppure indicando la destinazione *relativa* attraverso la primitiva *meet*, in cui cioè indicano il nome di

un altro agente o di una risorsa con i quali vogliono interagire e che quindi vogliono 'incontrare'.

- ***Comunicazione***: i meccanismi di comunicazione sono basati sul trasferimento e l'invocazione locale di metodi di altri Agenti. Non sono previste tecniche di scambio messaggi.
- ***Sicurezza***: Telescript contiene diversi supporti alla sicurezza, tra i quali un controllo degli accessi basato su un meccanismo simile alle capability. Ad ogni agente e ad ogni posto è associata una *'autorità*'; ciascun agente è dotato di un 'permesso', nel quale sono codificati i suoi diritti d'accesso e i limiti sull'uso delle risorse; quando ad esempio un agente giunge in un determinato luogo, la piattaforma software che lo governa può richiedergli il permesso ed eventualmente limitare o rifiutare il suo accesso.

Lo scarso successo commerciale di Telescript è principalmente da imputare al fatto che imponeva ai programmatori l'apprendimento di un linguaggio completamente nuovo. La General Magic lo ha per tali ragioni accantonato, riservandosi lo sfruttamento dell'esperienza accumulata per sviluppare un nuovo sistema basato sul linguaggio Java, chiamato Odyssey.

3.2 Odyssey

Odyssey [ODY97] rappresenta l'evoluzione di Telescript della General Magic ed è realizzato in linguaggio Java. Ne eredita tutte le caratteristiche dal punto di vista della mobilità.

- ***Comunicazione***: A differenza del proprio predecessore presenta numerose peculiarità relative a diversi meccanismi di comunicazione

tra oggetti remoti: dallo standard RMI a DCOM di Microsoft e CORBA IIOP. Gli Agenti dialogano con i propri simili e con le piattaforme attraverso l'invocazione locale dei loro metodi. Manca, però , un sistema avanzato di comunicazione tra gli agenti.

- ***Architettura di sistema***: Un altro limite di questa libreria è legato all'impossibilità di eseguire diverse istanze della JVM sullo stesso calcolatore, impedendo di fatto l'opportunità di avere diversi server Odyssey sulla stessa macchina fisica. Attualmente Odyssey viene distribuito gratuitamente per scopi non commerciali.
- ***Sicurezza***: nessun meccanismo di sicurezza è stato previsto in questo prodotto software.

3.3 Tacoma

Tacoma [JVS95] è un progetto nato nel 1994 dalla collaborazione tra l'Università di TromsØ (Norvegia) e la Cornell University. Gli agenti sono scritti in Tcl, un linguaggio nato appositamente per la programmazione degli agenti mobili.

- ***Architettura di sistema***: Un agente viene creato inserendo il programma in una specifica cartella, chiamata '*code*'; il nome della locazione viene invece memorizzato nella cartella chiamata '*host*'. La cartella rappresenta, essenzialmente, un contenitore d'Agenti. Lo stato di un agente può essere esplicitamente memorizzato in cartelle (dette *folders*), che possono essere aggregate in schedari (*briefcases*).
- ***Mobilità***: La primitiva *meet* permette la migrazione dell'agente sull'host specificato, su cui deve essere presente un altro agente che sia in grado di eseguire il codice incorporato. Lo schedario che

contiene codice, destinazione ed altre cartelle definite dall'applicazione, viene spedito all'agente sulla locazione di destinazione. Perciò il sistema non cattura lo stato dell'agente a livello di thread nel momento in cui esso migra, ma ricomincia da capo l'esecuzione del programma sulla destinazione.

- ***Comunicazione***: Entrambi i modelli di comunicazione sincrona ed asincrona sono supportati. Una forma alternativa di comunicazione si fonda sull'impiego dei 'cabinet', una sorta di deposito comune dove gli agenti possono memorizzare dati specifici dell'applicazione alla quale possono poi accedere anche altri processi.
- ***Sicurezza***: Un grosso limite di Tacoma è costituito dalla mancata implementazione di meccanismi di sicurezza.

3.4 Agent Tcl

Agent Tcl [GRA96] è stato sviluppato dal Dartmouth College nella seconda metà degli anni Novanta;

- ***Architettura di sistema***: anche questa libreria si basa su agenti Tcl, ai quali il sistema consente di migrare tra i vari server che supportano la loro memorizzazione, la loro esecuzione e le loro comunicazioni. Un interprete Tcl modificato è usato per eseguire il programma dell'agente, e permette di catturarne lo stato di esecuzione a livello thread.
- ***Mobilità***: La migrazione consiste nel trasferimento dell'intero sorgente del codice, dei dati e dello stato dell'agente ed avviene in modo assoluto, specificando il nome della locazione di destinazione. Il nome degli agenti dipende dalla locazione in cui si trovano, e

perciò cambia a seguito di ogni spostamento. Oltre alla migrazione di tipo assoluto, Agent Tcl supporta anche il meccanismo che permette ad un agente di creare un proprio clone e di spedirlo poi sulla locazione remota.

- ***Comunicazione*** : è supportato un meccanismo di scambio messaggi tra Agenti basato su metodi chiamati *agent_send* e *agent_receive.* La comunicazione avviene in serie, cioè per un Agente le operazioni di invio/ricezione non possono avvenire contemporaneamente.
- ***Sicurezza*** : Agent Tcl utilizza un ambiente di esecuzione 'Safe Tcl', che fornisce un accesso controllato alle risorse. Quando un agente vuole eseguire un'operazione ritenuta pericolosa per il sistema, esso deve ottenere un'opportuna mediazione; in sostanza non può accedere direttamente alle risorse. Il controllo degli accessi è realizzato legando i diritti che un agente può vantare su una risorsa alla locazione di provenienza dell'agente stesso; in pratica si tratta di una lista di controllo di scarsa granularità. Per eseguire la verifica dell'identità degli agenti viene chiamato un programma esterno (chiamato PGP), che è usato anche per criptare i dati da trasferire quando necessario.

3.5 Aglets

Aglets [AGL98] è un sistema basato su Java sviluppato da IBM tra il 1996 e il 1998. Gli agenti vengono chiamati *aglets* e sono in grado si spostarsi tra i vari server remoti, chiamati contesti (*aglet contexts*).

- ***Architettura di sistema***: Quando un programmatore vuole creare una nuova classe di aglet, essa eredita dal sistema una serie di metodi

(*callback methods*); questi metodi vengono invocati ogni qualvolta occorre un certo evento durante il ciclo di vita dell'agente. Ad esempio esisterà un metodo *OnArrival* che verrà invocato ogni volta che l'agente arriva su una nuova locazione; è compito poi del programmatore sovrascrivere eventualmente l'implementazione di tali metodi per personalizzare il comportamento dei propri aglet. Il progetto degli Aglet è principalmente composto da due componenti:

- *J-AAPI* (Java – Agents Application Program interface) è l'insieme di tutte quelle classi che permettono l'implementazione di agenti utilizzando il linguaggio Java. Il pacchetto comprende un server ad interfaccia grafica chiamato Tahiti, che permette di creare, mandare in esecuzione, registrare, clonare, e compiere altre operarazioni sui propri Agenti.
- *ATP* (Agent Transfer Protocol) è il protocollo che permette il trasferimento di Agenti mobili attraverso Internet.

- ***Mobilità***: La migrazione degli agenti è di tipo assoluto. La mobilità è implementata attraverso la serializzazione degli oggetti fornita dal linguaggio Java, e lo stato a livello thread non viene catturato. Quando un agente viene riattivato su una locazione, il suo metodo *run* viene automaticamente invocato.
- ***Comunicazione***: L'unica forma di comunicazione supportata dal sistema consiste nello scambio di messaggi.

L'ambiente di sviluppo degli Aglets è l'ASDK (Aglets Software Development Kit) ed è distribuito gratuitamente da IBM. Per ora i

meccanismi di sicurezza sono ancora piuttosto limitati, ma IBM ha reso noto che presto verranno notevolmente incrementati [AGL98].

3.6 Voyager

È anch'esso un sistema basato sul linguaggio Java ed è stato sviluppato dalla ObjectSpace [OBJ97] a partire dal 1997.

- ***Architettura di sistema***: Un aspetto nuovo introdotto da *Voyager* è l'utilità chiamata *VCC* (Virtual Code Compiler) che, partendo da una generica classe di Java, ne crea una equivalente accessibile remotamente, chiamata 'classe virtuale' (*virtual class*). L'istanza di una classe virtuale può essere creata su un host remoto e tale istanza è accessibile indipendentemente dalla locazione. Questo meccanismo è utilizzato per implementare gli agenti. Ad un agente vengono assegnati un identificativo globale univoco e, durante la costruzione dell'oggetto, un eventuale nome simbolico. È disponibile un servizio di gestione dei nomi in grado di individuare l'agente a partire dal suo identificativo o dal suo nome.
- ***Mobilità***: La migrazione degli agenti viene realizzata attraverso la primitiva *moveTo*, di cui sono dotate le classi virtuali. Lo spostamento può avvenire sia in maniera 'assoluta', specificando la destinazione, oppure 'relativa' indicando un riferimento virtuale ad un oggetto che si vuole raggiungere. Lo stato a livello thread non è mantenuto; è comunque possibile evitare che l'agente debba ripartire da capo la sua esecuzione, specificando il metodo che deve essere richiamato una volta completato il trasferimento.

- ***Comunicazione***: La comunicazione può avvenire attraverso invocazione dei metodi dei riferimenti virtuali ed è basata su diverse forme di RMI: sincrona, asincrona, one-way, future (asincrona), one-way multicast e publish/subscribe. Gli agenti possono essere raggruppati gerarchicamente in gruppi; in questo modo è realizzabile una sorta di multicasting.
- ***Sicurezza***: Voyager implementa un rudimentale sistema di sicurezza chiamato *Security Manager* , con il semplice scopo di controllare l'accesso alle risorse da parte degli Agenti. Nessun meccanismo di protezione è previsto per salvaguardare le comunicazioni.

3.7 Concordia

Anche *Concordia* [MIT97], sviluppato da Mitsubishi Electric dal 1997, supporta agenti mobili scritti in linguaggio Java.

- ***Architettura di sistema*** : Come molti altri sistemi di questo tipo, sfrutta i meccanismi di serializzazione degli oggetti e di *class loading* forniti da Java, e non cattura lo stato di esecuzione a livello di thread. Ogni oggetto 'agente' è associato ad uno specifico oggetto 'itinerario' (*itinerary*), che indica il cammino che l'agente dovrà seguire per trasferirsi e i metodi che può eseguire su ciascun host. I server possono cautelarsi e proteggere le proprie risorse disponendo di liste di controllo degli accessi statiche basate sull'identità degli utenti a cui gli agenti sono univocamente associati. Il modulo principale è costituito dal *Concordia Server*, che fornisce i servizi di comunicazione e l'ambiente di esecuzione per gli Agenti. Mentre possono esistere più moduli di questo tipo, all'interno della rete

locale vi deve essere un solo *Administrator Manager*. Esistono poi i moduli che si occupano della tolleranza ai guasti (*Persistence Manager*) e della migrazione degli agenti (*Queue Manager*). La combinazione di questi garantisce una tolleranza ai guasti maggiore rispetto ad altri pacchetti per la programmazione ad agenti.

- ***Mobilità*** : La migrazione degli agenti viene gestita dall' Administrator Manager e lo spostamento può avvenire solo in maniera 'assoluta', specificando la destinazione.
- ***Comunicazione*** : La collaborazione tra gli agenti è gestita da un modulo chiamato *Event Manager*, il quale raccoglie i risultati parziali dei vari Agenti e calcola il risultato globale che viene a sua volta comunicato a tutti gli Agenti attraverso l'invocazione di metodi remoti. Tale struttura presenta troppi problemi a livello di comunicazione tra Agenti per l'inadeguatezza nella gestione di attività complesse. Manca infatti un metodo per la comunicazione diretta tra Agenti che non coinvolga l'Event Manager.
- ***Sicurezza*** : questo tool di sviluppo presenta diverse caratteristiche inerenti alla sicurezza: il trasferimento degli Agenti viene reso sicuro attraverso un protocollo chiamato SSL. La verà novità è il meccanismo di protezione degli Agenti stessi dal tentativo di accesso al codice o allo stato da parte di altri Agenti.

3.8 Ajanta

Ajanta [KAR98] è un sistema basato sugli agenti mobili sviluppato da un gruppo di ricerca dell'Università del Minnesota a partire dal 1998. Anche questo sistema è implementato attraverso il linguaggio Java, di cui

sfrutta i meccanismi per quanto riguarda la sicurezza (Security Manager), la serializzazione degli oggetti e l'invocazione remota dei metodi.

- ***Architettura di sistema*** : Quando un agente viene creato, esso eredita automaticamente la struttura della classe *base_agent* fornita da Ajanta; a questo punto il programmatore può ridefinire i metodi *run*, *arrive* e *depart* ereditati .
- ***Mobilità*** : La migrazione degli agenti può avvenire sia in maniera assoluta che 'relativa', attraverso la primitiva *go*; prevede inoltre l'oggetto 'itinerario' che contiene il cammino e i metodi associati ai vari server. La gestione dei nomi è indipendente dalla locazione e realizzata da un servizio globale del sistema, chiamato URN (*Uniform Resource Naming*).
- ***Sicurezza*** : Il principale obiettivo che i progettisti di Ajanta si sono posti è stata la dotazione di un sistema di meccanismi in grado di garantire un'esecuzione il più possibile sicura e robusta degli agenti mobili in ambiente aperto. Per questo motivo sono previste tecniche di crittografia ed autenticazione per la comunicazione sicura e meccanismi di protezione degli agenti da attacchi ostili da parte di altri agenti o delle locazioni stesse. Il controllo d'accesso alle risorse è realizzato attraverso liste di capability e le autorizzazioni sono basate sull'identità dell'utente proprietario dell'agente.

3.9 Jafmas

Jafmas [MAG98] (Java-based Agent Framework for Multiagent Systems) è composto da classi Java che consentono la realizzazione di sistemi multiagente complessi, senza utilizzare moduli di controllo centralizzati per la comunicazione tra Agenti.

- ***Architettura di sistema*** : Questo pacchetto è strutturato in modo tale che un'applicazione multi-agente rappresenti una comunità di Agenti che "conversano" e "collaborano" tra loro per il raggiungimento di un'obiettivo comune. La metodologia per lo sviluppo di applicazioni seguita da Jafmas prevede cinque tappe principali:

    - Identificazione degli Agenti che compongono l'applicazione, suddivisi in categorie a seconda delle funzioni e dei servizi svolti.
    - Definizione delle conversazioni degli Agenti, rappresentate attraverso Automi a Stati Finiti, così da avere una mappa di tutti i possibili dialoghi all'interno dell'applicazione.
    - Definizione delle regole di conversazione : vengono definite delle metodologie per la gestione dei dialoghi tra Agenti.
    - Analisi del modello delle conversazioni attraverso un modello a reti di Petri, così da assicurare la coerenza globale della computazione svolta dal sistema multi-Agente.
    - Implementazione del sistema multi-Agente con Jafmas : Il programmatore estende la classe Agent per creare le classi

di agenti definite dall'applicazione. Al costruttore dovranno essere passati come parametri (tramite il metodo *createAgent()*) il nome dell'agente e l'array delle proprietà.

- ***Mobilità*** : Lo spostamento di un'Agente può avvenire sia in maniera 'assoluta', specificando la destinazione, oppure 'relativa' indicando un riferimento virtuale ad un oggetto che si vuole raggiungere. Lo stato a livello thread non è mantenuto; è comunque possibile evitare che l'agente debba iniziare da capo la sua esecuzione, specificando il metodo che deve essere richiamato una volta completato il trasferimento.
- ***Comunicazione*** : Il meccanismo di comunicazione tra Agenti, strutturato a reti di Petri, supporta il broadcasting. La classe *Message* permette di definire i tipi di messaggi scambiati fra gli agenti. È molto importante il suo attributo performative, che contiene il tipo di azione che l'agente intende compiere. L'insieme di queste azioni è scelto dal programmatore in funzione della particolare applicazione.
- ***Sicurezza*** **:** Non è previsto alcun meccanismo per gestire la sicurezza del sistema o per controllare l'accesso alle risorse.

3.10 Jade

Jade (Java Agent Development Environment) verrà trattato approfonditamente nel capitolo successivo. In questo paragrafo si vuol dare soltanto un'idea generale sui principi di funzionamento che ne hanno determinato l'adozione per la realizzazione del sistema ad agenti oggetto di questo lavoro di tesi. Jade nasce nel 1997 dalla collaborazione del CSELT (Centro Studi e Laboratori Telecomunicazioni) di Torino e

del Dipartimento di Informatica dell'Università di Parma, con lo scopo di implementare uno strumento di sviluppo di sistemi multi-Agente conformi alle specifiche FIPA (Foundation for Intelligent Physical Agent). Rispetto alle librerie sofware già disponibili, Jade riserva una particolare attenzione alla trasmissione e decodifica dei messaggi, lasciando ai programmatori il compito di definire le applicazioni ed i protocolli d'interazione. Vi sono, inoltre, numerosi tools utili in fase di test e debugging , quali un'interfaccia grafica per la gestione ed il controllo della piattaforma (*Remote Management Agent*) ed un agente dedicato al monitoraggio delle conversazioni tra gli Agenti (*Sniffer Agent*). Tra le principali caratteristiche che fanno di Jade un valido supporto allo sviluppo di sistemi ad Agenti vi sono:

- La possibilità di lanciare sullo stesso host più di una piattaforma ad agenti, più DF (*Directory Facilitator*), più AMS (*Agent Management System*),…
- L'utilizzo del protocollo *IIOP* compatibile con le specifiche FIPA per connettere fra loro diverse piattaforme.
- Un meccanismo di trasporto dei messaggi *ACL* estremamente efficiente. Quando mittente e destinatario si trovano all'interno della stessa piattaforma i messaggi sono trattati come oggetti Java. Nel caso contrario, cioè quando mittente e destinatario si trovano su piattaforme diverse il messaggio viene automaticamente convertito nel formato stringa FIPA-compatibile.
- Un meccanismo di registrazione e deregistrazione degli Agenti.
- Un servizio di *Naming* in accordo con le specifiche FIPA.

3.11 FIPA-OS

FIPA-OS, sviluppato dall'inglese Nortel Networks nel 1999 [HAD99] è un'implementazione "***Open-Souce***" di FIPA, disponibile gratuitamente. Questo significa da un lato che chiunque può contribuire al suo miglioramento ma, dall'altro, che nessuno vigila sulle modifiche apportate. Questo prodotto software permette lo sviluppo di Agenti basati sullo standard FIPA ed è realizzato in linguaggio Java [NOR00]. La gerarchia delle classi, dei metodi e l'interfaccia grafica rivelano quanto Jade abbia influenzato i suoi sviluppatori.

- ***Architettura di Sistema***: come il concorrente Jade, a cui si ispira fortemente, FIPA-OS consente la creazione di piattaforme multi-Agenti gestite tramite AMS e DF. Le azioni eseguibili da un Agente sono chiamate *Tasks* (in Jade sono i *Behaviours*) e la loro esecuzione viene controllata e gestita da un'entità nota come *TaskManager* (TM). Il suo scopo è quello di permettere l'esecuzione di più Tasks in parallelo e monitorarne gli effetti sugli altri Agenti della piattaforma.
- ***Mobilità***: al momento l'ultima versione di FIPA-OS, la *1.3.2* (18 Ottobre 2000) non supporta meccanismi di mobilità per gli Agenti.
- ***Comunicazione***: basata sullo scambio di messaggi tra agenti, le comunicazioni vengono gestite attraverso l'*IPMT* (*Internal Platform Message Transport*). Sono supportate le codifiche ASCII e XML per i messaggi ACL.
- ***Sicurezza***: nessun meccanismo di sicurezza è stato implementato.

Essendo un prodotto nuovo, FIPA-OS necessita di integrazioni e miglioramenti, soprattutto relativamente a mobilità e sicurezza. Il fatto di essere "*Open-Source*" permette una rapida evoluzione del sistema ma,

come di recente sottolineato ad un Meeting mondiale sull'Argomento, molte delle modifiche introdotte mostrano problemi di compatibilità e conformità allo standard FIPA.

3.12 Confronto tra le librerie

Si fa a questo punto una breve riepilogo delle principali scelte progettuali che caratterizzano questi sistemi. In particolare si fornirà una schematica rappresentazione delle primitive di programmazione supportate, focalizzando l'attenzione sulle problematiche principali nell'ambito della mobilità degli agenti, della comunicazione e della sicurezza [CIC99][SAT99]. Nella Tabella 1 vengono messi a confronto: la gestione dei nomi (cioè la tecnica di identificazione degli Agenti) ed il supporto della mobilità.

Sistema	Gestione dei nomi	migrazione degli agenti
Telescript	**Dipendente dalla locazione.**	**Assoluta (go) e relativa (meet).**
Odyssey	**Dipendente dalla Locazione**	**Assoluta e relativa.**
Tacoma	**Dipendente dalla locazione.**	**Assoluta e relativa, supportate entrambe dalla sola primitiva meet.**
Agent Tcl	**Dipendente dalla locazione; alias simbolici opzionali.**	**Assoluta, agent_jump;**
Aglets	**Dipendente dalla locazione.**	**Assoluta, dispatch; supporta l'entità astratta Itinerary.**
Voyager	**Globale ed indipendente dalla locazione.**	**Assoluta e relativa, supportate entrambe dalla sola primitiva moveTo.**
Concordia	**Dipendente dalla locazione. Disponibile un servizio di directory.**	**Assoluta, basata sul contenuto dell'Itinerary dell'agente.**

Sistema	Gestione dei nomi	migrazione degli agenti
Ajanta	**Globale ed indipendente dalla locazione.**	**Assoluta e relativa; supporta l'entità astratta Itinerary.**
Jafmas	**Dipendente dalla locazione**	**Assoluta e relativa**
Jade	**Globale ed indipendente dalla locazione**	**Assoluta e relativa**
FIPA-OS	**Dipendente dalla locazione**	**Non supportata**

Tabella 1: Mobilità degli agenti supportati dai vari sistemi.

Per quanto riguarda la *mobilità* degli agenti, si è visto che la distinzione principale è tra migrazione assoluta, in cui viene espressamente richiesto il nome del server su cui ci si vuole trasferire, e migrazione relativa, in cui cioè l'agente indica il nome di un altro agente o di una risorsa sulla cui locazione desidera spostarsi, qualunque essa sia in quel momento. Alcuni sistemi, poi, per dotare le loro primitive di trasferimento di un maggiore livello di astrazione, introducono il concetto di 'itinerary', che contiene una lista dei server da visitare e il corrispondente codice da eseguire su tali locazioni.

Per portare a termine nel migliore dei modi il proprio compito, è di norma necessario che gli agenti possano comunicare tra loro; in molti casi una loro collaborazione è infatti indispensabile al fine di ottenere il risultato desiderato. Perciò devono essere messe a disposizione dal sistema una serie di primitive di *comunicazione* e *sincronizzazione* tra gli agenti che rendano possibile la cooperazione.

Un'applicazione può inoltre aver bisogno di conoscere lo stato di un agente che si trova su un host remoto; ad esempio, se si verifica un errore, si deve poter forzare la terminazione di un agente, oppure si deve essere in grado di bloccarlo in qualche modo. È quindi opportuno prevedere alcune primitive di *controllo*, attraverso le quali le applicazioni possano monitorare i propri agenti.

In Tabella 2 sono confrontate le tecniche di comunicazione, quelle di gestione degli eventi e di controllo delle azioni compiute dagli Agenti.

Sistema	Comunicazione	Eventi	Controllo degli agenti
Telescript	**Trasferimento e invocazione locale dei metodi di altri agenti.**	**Eventi supportati a livello di linguaggio.**	**Non previsto.**
Odyssey	**Scambio messaggi.**	**Non previsti.**	**Non previsto.**
Tacoma	**Meet permette agli agenti sulla stessa locazione di scambiarsi dati (brief-cases).**	**Non previsti.**	**Non previsto.**
Agent Tcl	**Scambio dei messaggi attraverso agent_send e agent_receive; comunicazione in serie.**	**Gli eventi sono come messaggi.**	**Non previsto.**
Aglets	**Scambio di oggetti messaggio; comunicazione sincrona, one-way oppure future-reply.**	**Non previsti.**	**Forza gli agenti a rientrare con la primitiva retract.**
Voyager	**Supporta CORBA, RMI, DCOM; invocazioni sincrone, one-way, future, e multicast.**	**Conforme al modello di evento JavaBeans.**	**Non previsto.**
Concordia	**Trasferimento ed invocazione locale dei metodi degli altri agenti; integra CORBA;**	**Modelli di eventi di tipo publish-subscribe e multicast.**	**Non previsto.**
Ajanta	**Trasferimento ed invocazione locale di metodi di altri agenti.**	**I server possono fare richieste circa lo stato degli agenti.**	**E' dotato di controllo degli accessi**

Sistema	Comunicazione	Eventi	Controllo degli agenti
Jafmas	**Strutturate a reti di Petri.**	**Non previsti.**	**E' dotato di controllo degli accessi**
Jade	**Conformi alle specifiche FIPA**	**Tramite Remote Management Agent**	**E' dotato di controllo degli accessi**
FIPA-OS	**Conformi alle specifiche FIPA**	**TaskManager**	**Non previsto**

Tabella 2: Primitive di comunicazione e controllo

Aspetto certamente non secondario per i sistemi basati sulla tecnologia ad agenti mobili è quello della *sicurezza* e della *robustezza ai guasti.* Finché un agente può spostarsi attraverso host e reti non sicuri, il programmatore necessita di operazioni che gli permettano di proteggere i dati; questi mezzi comprendono strumenti di crittografia delle informazioni riservate e dei messaggi, e meccanismi di controllo delle intromissioni nell'area dati e nel codice. Allo stesso tempo è opportuno dotare gli host stessi di primitive di protezione delle proprie risorse.

Infine, quando un agente incontra dei problemi che lo costringono ad interrompere la propria esecuzione, per il proprietario sarebbe preferibile disporre di meccanismi per recuperare in qualche modo lo stato in cui esso si trovava al momento dell'arresto. Alcuni sistemi prevedono meccanismi che permettono di salvare lo stato di un agente su memoria non volatile in corrispondenza dei *checkpoint* indicati dal programmatore; quando l'agente si arresta, viene richiamata un'opportuna primitiva di restart che carica l'ultimo stato memorizzato.

In Tabella 3 sono riassunti questi aspetti per i vari sistemi analizzati.

Sistema	Comunicazione sicura	Protezione delle risorse	protezione degli agenti
Telescript	**Trasferimento degli agenti autenticato (RSA) e criptato (RC4).**	**Capability list; limite di quota; autorizzazione basata su authority.**	**Non prevista.**
Odyssey	**Non prevista**	**Non prevista.**	**Non prevista.**
Tacoma	**Non prevista.**	**Non prevista.**	**Non prevista.**
Agent Tcl	**Autenticazione e crittografia (PGP)**	**Safe Tcl; non prevede autorizzazione.**	**Non prevista.**
Aglets	**Non prevista.**	**Diritti d'accesso statici a due livelli (trusted, untrusted)**	**Non prevista.**
Voyager	**Non prevista.**	**Security Manager; diritti d'accesso statici a due livelli (native, foreign).**	**Non prevista.**
Concordia	**Trasferimento degli agenti autenticato attraverso il protocollo SSL.**	**ACL statiche basate sull'identità del proprietario.**	**Gli agenti sono protetti dagli altri attraverso il meccanismo d'accesso alle risorse.**
Ajanta	**Trasferimento degli agenti autenticato attraverso i protocolli ElGamal e DSA.**	**Capability list; autorizzazione basata sul proprietario dell'agente.**	**Meccanismo per verificare tentativi di intromettersi nello stato o nel codice dell'agente**
Jafmas	**Non prevista.**	**Non prevista.**	**Non prevista.**
Jade	**Attraverso il protocollo Jade-IPMT**	**Tramite Security Manager**	**Tramite SecurityManager**
FIPA-OS	**Non prevista**	**Non prevista**	**Non prevista**

Tabella 3: Meccanismi di sicurezza e protezione degli agenti

Analizzando a fondo le caratteristiche dei vari tools di sviluppo di sistemi multi-Agente si può osservare come ognuno di loro abbia

peculiarità innovative ed utili sotto certi aspetti, ma anche carenze e lacune in altri.

- *Aglets*, *Voyager*, *Odyssey*, *Jade* e *Jafmas* supportano, al contrario di *Concordia*, il cosiddetto **Meeting**, cioè l'invocazione diretta di metodi fra Agenti nello stesso ambiente di esecuzione.
- *Jade* e *Jafmas* sono gli unici ad essere stati progettati con un efficiente meccanismo di coordinazione e collaborazione tra gli Agenti. *Odyssey* ne è sprovvisto mentre *Voyager* e *Concordia* ne offrono uno piuttosto limitato.
- *Jade* e *Voyager* supportano direttamente l'invocazione di metodi remoti (RMI). *Jade* in modo estremamente flessibile. In *Concordia* la comunicazione è basata su eventi mentre in *Aglets* e *Jafmas* si sfrutta lo scambio di messaggi. In *Odyssey*, addirittura, non esiste alcun meccanismo per chiamare metodi remoti.
- *Jade* utilizza oggetti Java per le comunicazioni tra agenti sullo stesso host mentre *Concordia* e *Voyager* supportano "Eventi Distribuiti".
- *Jade* supporta le specifiche FIPA (in particolare Jade 2.0 supporta completamente FIPA 2000).

Data la versatilità, flessibilità e soprattutto capacità di coordinazione degli Agenti e del modello per la conversazione, Jade è diventato negli ultimi anni un punto di riferimento a livello internazionale per la realizzazione di Tools utili alla creazione di Sistemi multi-Agente. La possibilità inoltre di accedere gratuitamente ai codici sorgente, ad esempi e di utilizzare Agenti dedicati all'assistenza al programmatore nelle fasi di test e debugging, nonché il supporto completo alle specifiche FIPA rendono ideale questo pacchetto software per il progetto e la realizzazione del presente lavoro di tesi.

Capitolo 4

JADE : JAVA AGENT DEVELOPMENT ENVIRONMENT

In questo capitolo viene descritto il pacchetto applicativo per la creazione di sistemi multi-Agente chiamato Jade, sottolineando maggiormente le caratteristiche rilevanti per la realizzazione di questo progetto.

4.1 Jade

Jade (*Java Agent Development Environment*) è un ambiente di sviluppo creato allo scopo di facilitare l'implementazione di sistemi multi-Agente conformi alle specifiche rilasciate dalla *Foundation for Intelligent Physical Agents* (FIPA) [JAD00].

Grazie alla collaborazione tra Fipa, CSELT (*Centro Studi e Laboratori Telecomunicazioni*) di Torino e Dipartimento di Informatica dell'Università di Parma, Jade vede la luce nella sua prima versione 1.0 nel Luglio 1998. Da allora si sono susseguite numerose versioni ed aggiornamenti. Al momento della stesura di questa Tesi si è giunti alla release 2.1.

Jade dimostra fin dalle origini le sue enormi potenzialità, grazie alle caratteristiche che lo contraddistinguono:

- Una piattaforma ad Agenti conforme alle specifiche FIPA. Tale piattaforma include AMS (Agent Management System), DF (Directory Facilitator) e ACC (in accordo con FIPA2000 l'ACC

non è più un'Agente). Questi tre elementi, fondamentali per il funzionamento della piattaforma, vengono attivati automaticamente al lancio della piattaforma stessa, semplificando il lavoro del programmatore.

- Piattaforma ad agenti distribuiti. Una piattaforma ad agenti può essere suddivisa in diversi host. Nonostante sia possibile avere in esecuzione diverse *Java Virtual Machine* (JVM) sullo stesso Host (e quindi diverse APs) il loro utilizzo viene sconsigliato dai progettisti perché:

    - La CPU dell Host va in sovraccarico.
    - La memoria non e` condivisa.
    - Le comunicazioni sono lente.
    - I vari agenti non si possono indirizzare con il solo nickname, ma occorre fornire anche il platform ID ed almeno un indirizzo di trasporto.
    - Se il server ha un crash, muoiono tutti gli agenti perche` comunque girano sulla stessa macchina.

- Grazie alla gestione dei *Threads* di Jade, ogni agente può eseguire diversi *tasks* concorrentemente, garantendo un sistema di comunicazione tra Agenti "leggero" ed efficiente. Si ricorda che un *Thread* è una parte di un programma predisposta per essere eseguita autonomamente concorrentemente ad altri flussi di controllo.
- Possibilità di creare più Directory Facilitator (DF) sulla stessa piattaforma, consentendo la realizzazione di applicazioni basate su domini multipli e strutturati gerarchicamente (nell'ambito della stessa AP).

- Possibilità di registrazione di uno stesso Agente presso domini diversi, così da rendere disponibili i propri servizi ad una comunità di Agenti più vasta di quella di origine.
- Un meccanismo di trasporto ed un'interfaccia per l'invio/ricezione dei messaggi tra Agenti.
- Supporto del protocollo IIOP FIPA per la comunicazione tra differenti piattaforme.
- Una tecnica di comunicazione tra Agenti della stessa piattaforma basata sul trasporto di messaggi ACL codificati come oggetti Java, che permette di alleggerire il lavoro dell'ACC e del programmatore. Quando gli Agenti comunicanti non appartengono allo stesso contenitore (cioè allo stesso dominio software dove sono in esecuzione), Jade provvede automaticamente alla conversione del messaggio nel formato ACL conforme alle specifiche FIPA.
- Una libreria di protocolli basati su FIPA. Per esempio i protocolli per la conversazione tra Agenti o per la migrazione.
- Registrazione automatica degli Agenti all'Agent Management System (AMS) della piattaforma.
- Assegnazione automatica ed univoca ad ogni Agente di un nome. Nelle ultime versioni di Jade, in accordo con FIPA2000, quello che veniva chiamato *GUID* (Globally Unique IDentifier) è stato rimpiazzato con l'*AID* (Agent Identifier) [vedi paragrafo 4.6].
- Un'interfaccia grafica estremante intuitiva ed efficiente in grado di monitorare ogni attività e comunicazione degli Agenti della piattaforma.

4.2 L'architettura della piattaforma ad Agenti.

L'architettura della piattaforma ad Agenti Jade è conforme alle specifiche FIPA e comprende tutti quegli aspetti necessari alla sua gestione e funzionamento, tra cui ACC, AMS e DF. Ogni comunicazione tra Agenti avviene attraverso lo scambio di messaggi, che rispettano le specifiche FIPA ACL. In figura 4.1 viene mostrato uno schema di tale architettura.

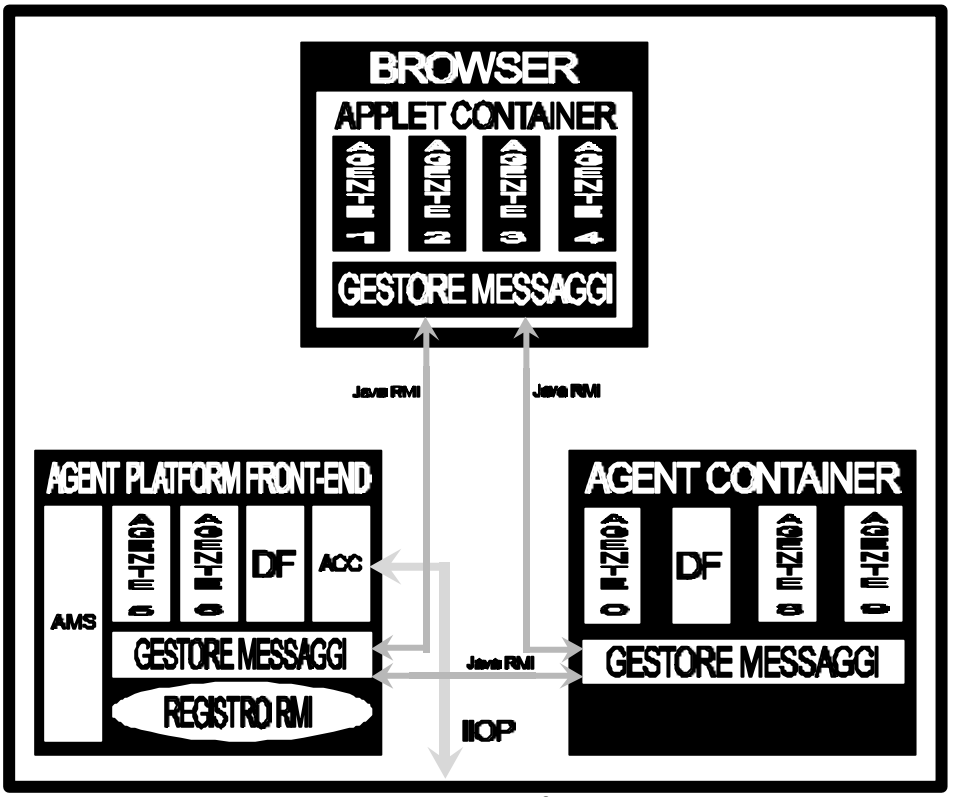

Fig 4.1 : Architettura Software di Jade.

L'architettura software è basata sulla coesistenza di più Java Virtual Machine e di un sistema di comunicazione tra queste sviluppato tramite Java RMI (Remote Method Invocation).

Come si evince dalla figura 4.1 una piattaforma ad Agenti è costituita da un insieme di contenitori d'Agenti. Ognuno di questi contenitori è un

oggetto server RMI che si occupa della gestione degli Agenti locali (sulla stessa AP), creandoli, sospendendoli, riattivandoli e terminandoli. Esso smista i messaggi ACL in arrivo, instradandoli in base al destinatario ed inserendoli nella coda privata di messaggi dell'Agente. Quest'ultima altro non è che un buffer in cui vengono inseriti i messaggi ricevuti dall'Agente, in ordine di arrivo.

Per agevolare il lavoro del programmatore Jade fornisce un'interfaccia grafica (***GUI***: Graphic User Interface) molto intuitiva che permette la gestione, il monitoraggio ed il controllo dello stato di tutti gli Agenti della piattaforma. L'interfaccia implementa, al suo interno, un'Agente, chiamato ***RMA*** (Remote Monitoring Agent), così come i Directory Facilitator (**DF**), permettendo in ogni istante di conoscere il rapporto esistente tra gli Agenti della AP e la piattaforma stessa.

Le classi principali di Jade sono le seguenti:

- ***Jade.core***: rappresenta il cuore dell'intero package. Per esempio, in Jade.core.Behaviours si trovano tutte le classi che descrivono i comportamenti elementari degli Agenti.
- ***Jade.lang***: contiene, tra le altre cose, le classi riguardanti i linguaggi di comunicazione utilizzati, tra cui ACL.
- ***Jade.domain***: implementa le specifiche FIPA riguardanti modelli, linguaggi, ontologie,…
- ***Jade.proto***: contiene le classi che implementano i protocolli FIPA per l'interazione tra Agenti, quali FIPA-REQUEST, FIPA-QUERY, …
- ***Jade.gui***: include un set di classi generiche utili alla realizzazione di interfacce grafiche GUI.
- ***Jade.mtp***: queste classi sono dedicate a tutti i metodi per il trasporto dei messaggi (Message Transport Protocols).

- ***Jade.tools***: contiene una serie di tools utili alla gestione ed al controllo della piattaforma e/o degli Agenti, come:
  - ***RMA*** (Remote Management Agent).
  - ***DA*** (Dummy Agent) per il monitoraggio e il debugging.
  - ***SNIFFER***, un'Agente preposto all'intercettazione dei messaggi durante il loro tragitto. Questi vengono poi mostrati a video tramite un'opportuna intefaccia grafica.
  - ***SOCKET PROXY AGENT***, è un'Agente che si occupa di convertire messaggi ACL in testo ASCII per l'invio su connessione TCP/IP.

4.3 La Comunicazione tra Agenti

Il contenitore principale della piattaforma Jade è denominato *Front-End* e, oltre al DF e all'AMS, al suo interno presenta un registro RMI, usato dagli altri contenitori d'Agenti per condividere la piattaforma. Esso infatti tiene traccia di tutti i contenitori con i loro riferimenti (indirizzi) RMI in aggiunta all' *Agent Global Descriptor Table*, che tiene, a sua volta, traccia di tutti gli Agenti e delle informazioni (AIDs) che li riguardano. Quando un nuovo contenitore viene attivato, esso crea un registro RMI sull host corrente mettendosi in ascolto su una determinata porta TCP. Dopodichè inizializza il sistema ad Agenti FIPA creando DF, AMS e ACC.

Ogni volta che un messaggio perviene ad un Agente, il suo contenitore tiene traccia del mittente, così che per l'invio di eventuali risposte non sia necessario far ricorso all' Agent Global Descriptor Table (***AGDT***), velocizzando in tal modo le operazioni di comunicazione tra Agenti. Ovviamente, in un ambiente dinamico, dove i contenitori appaiono e scompaiono in continuazione, un ricorso all'AGDT risulta inevitabile.

Jade possiede un meccanismo che gli permette la scelta della tecnica più efficiente per la gestione dei messaggi basata sulla locazione dell'Agente ricevente.

Quando un'Agente Jade invia un messaggio si possono verificare le seguenti situazioni:

- Se l'Agente destinatario vive nello stesso contenitore, un oggetto Java rappresentante il messaggio ACL viene inviato al ricevente, senza alcuna conversione (per esempio il messaggio inviato dall'Agente 1 all'Agente 2 in figura 4.2).
- Se l'Agente destinatario vive sulla stessa piattaforma ma in un contenitore differente, un messaggio ACL viene inviato sfruttando la struttura Java Remote Method Invocation per Agenti distribuiti (per esempio il messaggio inviato dall'Agente 3 all'Agente 2 in figura 4.2).
- Se l'Agente si trova su un'altra piattaforma , in accordo con le specifiche FIPA, vengono utilizzati il protocollo standard IIOP e l'interfaccia OMG IDL. Questo comporta la conversione del messaggio ACL in una stringa di caratteri e l'invocazione remota tramite IIOP (*Internet Inter-ORB – Protocol*) [vedi paragrafo 4.5]. Dal lato del ricevitore avviene il processo inverso. Un decodificatore IIOP ricostruisce, a partire dalla stringa, il messaggio originale ACL.

La piattaforma presenta una sola interfaccia verso l'ambiente ad essa esterno ed è incarnata dall' ACC (Agent Communication Channel). Quest'ultimo è un oggetto server CORBA (*Common Object Request Broker Architecture*) IIOP [COR97] che resta in attesa di chiamate remote. Ogni volta che esso riceve un messaggio ACL codificato come

stringa di caratteri, lo analizza e lo converte il un oggetto ***Java ACLMessage***. Esso è, inoltre, in grado di effettuare anche l'operazione inversa quando un Agente Jade deve inviare messaggi al di fuori della piattaforma.

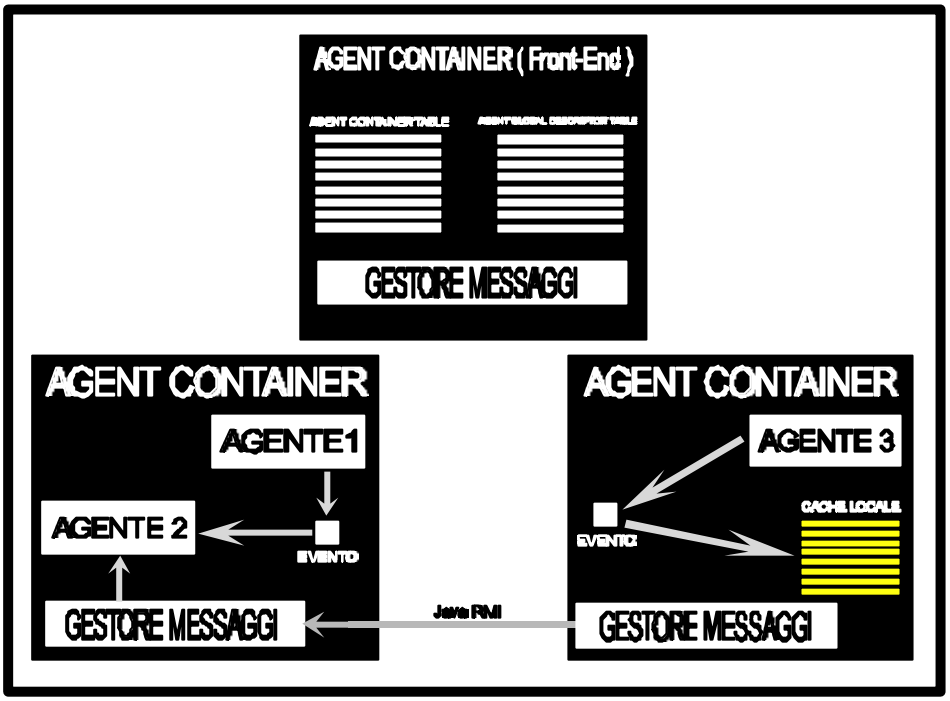

Fig 4.2 : Il modello per la comunicazione di Jade.

Il modello di comunicazione supportato da Jade è stato concepito appositamente per alleggerire le operazioni dell'Agente mittente ma anche di quello destinatario, così da ridurre il rischio di problemi nello scambio dei messaggi. Riepilogando, si possono verificare le seguenti situazioni per ciò che riguarda il passaggio di messaggi tra Agenti.

- *Stesso contenitore d'Agenti*: non vi sono chiamate di metodi remoti.
- *Stessa piattaforma, differenti contenitori, container cache hit*: una singola chiamata RMI crea e gestisce un oggetto ACLMessage.
- *Stessa piattaforma, differenti contenitori, container cache miss*: due chiamate RMI. La prima per aggiornare la cache a seconda del

contenuto del Global Agent Descriptor Table e la seconda per inviare il messaggio.

- *Piattaforme differenti*: viene effettuata una chiamata diretta attraverso l'ACC. Questo comporta una invocazione CORBA remota, una doppia codifica da oggetto Java a stringa di caratteri e da questa a flusso di byte IIOP ed infine il processo inverso al ricevitore.

4.4 L'Architettura interna dell'Agente in Jade

Una delle proprietà fondamentali di un'Agente software è la sua autonomia. Esso non si limita a reagire a degli stimoli esterni ma deve essere in grado di iniziare una conversazione e prendere l'iniziativa per raggiungere un determinato scopo. Questo concetto è ben rappresentato dal modello di Agente fondato su ***Behaviours*** che Jade implementa.

Un'Agente Jade non è altro che un'istanza di una classe Java, estensione della classe base *Agent*, costituita da un set di metodi elementari (concernenti la registrazione presso la piattaforma, configurazione, remote management,…) ed un insieme di tasks sotto forma di comportamenti, definiti dal progettista stesso. Un Agente, cioè, possiede al suo interno una serie di possibili comportamenti associati ai compiti ed alle situazioni che gli si presentano. La gestione multi-thread di Jade permette, inoltre, di eseguire più comportamenti concorrentemente. Ciò significa che mentre un'Agente sta aspettando una risposta da uno o più agenti o è impegnato in una conversazione, può al contempo eseguire senza problemi altre attività, migliorando l'efficienza e la flessibilità dell'intero sistema ed offrendo al programmatore una maggiore libertà nella progettazione dei Behaviours. Non è quindi

indispensabile concepire in modo rigido la sequenza dei comportamenti che, tra l'altro, possono essere sospesi, bloccati, e riattivati alla luce , per esempio, di messaggi giunti all'agente oppure a causa di mutate condizioni dell'ambiente esterno.

Un'Agente può trovarsi in uno dei seguenti stati:

- ***AP_INITIATED***: l'Agente esiste come classe ma non è ancora stato inizializzato. Non esiste ancora l'oggetto Agente. Non ha un nome, non è stato registrato e non può comunicare con altri Agenti.
- ***AP_ACTIVE***: l'Agente è registrato presso l'AMS, ha un nome, una locazione e può accedere alle funzionalità ed ai metodi offerti da Jade.
- ***AP_SUSPENDED***: l'Agente è in stato di sospensione.
- ***AP_WAITING***: l'Agente è in attesa, per esempio di un messaggio da parte da un altro Agente.
- ***AP_DELETED***: l'Agente è stato cancellato. Questo stato comporta la rimozione automatica dall'AMS.
- ***AP_TRANSIT***: un'Agente Mobile entra in questo stato quando è in fase di migrazione verso una nuova locazione.
- ***AP_COPY***: stato sfruttato internamente da Jade nella fase di clonazione di un Agente.
- ***AP_GONE***: anche questo stato viene usato internamente da Jade quando un Agente si è trasferito in una nuova piattaforma o contenitore ed ha assunto uno stato stabile.

La classe Agent è caratterizzata, inoltre, da due metodi principali:

- ***AddBehaviour()***: per aggiungere un particolare task o comportamento alla sequenza dei Behaviours di un'Agente. La sequenza dei Behaviours è un buffer in cui vengono inseriti, per ogni Agente, le azioni ed i comportamenti, in ordine di esecuzione.
- ***RemoveBehaviour()***: per effettuare l'operazione inversa, cioè rimuovere un Behaviour.

Per l'implementazione dei Behaviour vanno definite delle sottoclassi della classe ***Jade.core.Behaviours***. Si possono distinguere comportamenti semplici (***SimpleBehaviour***), cioè composti da un numero ridotto di tasks, o comportamenti complessi (***ComplexBehaviour***) , costituiti, invece, da un numero elevato di tasks e gestito secondo una complessa architettura di Behaviours. Tra le tante, le classi di metodi a cui si fatto maggiormente ricorso in questo lavoro di tesi sono:

- ***OneShotBehaviour***: classe di metodi Behaviour che vengono eseguiti una sola volta. Il suo metodo *Done()* ritorna sempre *True*.
- ***CyclicBehaviour***: questa classe include metodi che vengono ripetuti periodicamente. Il suo metodo *Done()* ritorna sempre *False*.

Done() fa parte del set di metodi elementari associati a tutte le classi Behaviour. E' utile ma, non indispensabile, per segnalare la fine dell'esecuzione di un determinato comportamento.

Come si è accennato precedentemente, Jade si fa carico direttamente della registrazione/deregistrazione degli Agenti presso l'AMS tramite i metodi di ***Setup()*** e ***TakeDown()*** (quest'ultimo fa passare l'Agente allo

stato di AP_DELETED) . Comunque nella classe *Agent* esistono i seguenti metodi per effettuare tali operazioni “manualmente”:

- ***RegisterWithAMS()*** : registrazione dell’Agente presso l’AMS.
- ***DeregisterWithAMS()*** : deregistrazione dell’Agente presso l’AMS.
- ***RegisterWithDF()*** : registrazione dell’Agente presso il DF.
- ***DeregisterWithDF()*** : deregistrazione dell’Agente presso il DF.
- ***ModifyDFDATA()*** : modifica dei dati dell’Agente presso il DF.

Un altro aspetto fondamentale è l’implementazione, in Jade, dei protocolli *FIPA-REQUEST* (vedi fig. 4.3). Una *Request* può essere accettata, ed allora il destinatario risponde al richiedente con un messaggio di *AGREE,* oppure non compresa (*NOT UNDERSTOOD*) ed in questo caso si invita il mittente ad inviare nuovamente la richiesta. Se il messaggio non giunge a destinazione correttamente si ottiene una risposta di tipo *FAILURE* accompagnata dalla causa. Nel caso di accettazione, una volta portata a termine l’azione, il mittente viene informato con un messaggio *INFORM*.

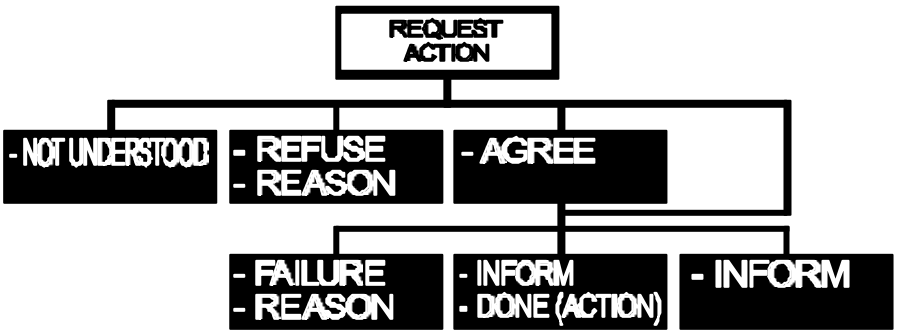

Fig. 4.3 : FIPA-REQUEST INTERACTION PROTOCOL.

I protocolli *FIPA-REQUEST* definiscono un modello per l'interazione tra Agenti definendo i ruoli di ***Initiator*** e di ***Responder***, in una conversazione. Due sono i Behaviours di base associati a questi ruoli:

- ***FipaRequestInitiatorBehaviour*** : al costruttore di un oggetto appartenente a questa classe devono essere passati tre parametri:

    - Un riferimento ad un Agente, necessario per l'invio e la ricezione di messaggi.
    - Il messaggio ACL, rappresentante la richiesta che deve essere inoltrata.
    - Eventuali informazioni addizionali (*Message Template*).

    Tra i parametri opzionali che si possono indicare nel *Message Template* vi sono:

    - ***Message Performative*** → "request". Serve a specificare il tipo di messaggio.
    - ***Protocol*** → "fipa-request". Indica il protocollo da utilizzare.
    - ***Conversation id*** → identifica l'Agente in caso di conversazioni multiple (che coinvolgono, cioè, più Agenti).
    - ***Reply-with*** → è l'AID dell'Agente mittente e serve per l'invio di risposte ai messaggi pervenuti.

- ***FipaRequestResponderBehaviour*** : questa classe implementa il ruolo di Responder del protocollo *FIPA-REQUEST*. La ricezione di un messaggio comporta le seguenti operazioni:

    - Analizza il messaggio spedito ed estrae il nome dell'azione da svolgere servendosi del metodo *getActionName()*.
    - Verifica la propria capacità di eseguire tale azione, cercando una opportuna *Factory*. Una Factory è una interfaccia costituita da un solo metodo dedicato alla creazione di un *ActionHandler*. ActionHandler è una estensione della classe *Behaviour* ed implementa i metodi appropriati all'azione richiesta.
    - Crea un ActionHandler e lo inserisce nella sequenza dei comportamenti dell'Agente.

Naturalmente esistono decine di Behaviours utili ad implementare diverse situazioni modellizzate da FIPA, ma quelle appena descritte sono le più significative.

Per una più accurata trattazione di metodi e classi Jade si rimanda alla guida alla programmazione ed all' HelpOnLine disponibili gratuitamente al sito internet dello CSELT di Torino [JAD00].

4.5 IIOP (Internet Inter-ORB Protocol)

IIOP è un protocollo applicativo client/server, definito da OMG, che permette invocazioni di oggetti remoti attraverso reti di calcolatori. Le sue specifiche definiscono un set di regole per la formattazione di tipologie di dati, chiamato ***CDR*** (*Common Data Representation*), supportati nel *CORBA Interface Definition Language* (***IDL***). L'utilizzo della rappresentazione CDR permette la determinazione di diversi formati per i messaggi che supporti la semantica ORB stabilita nelle specifiche CORBA. Insieme, le regole CDR ed i formati per i messaggi costituiscono un protocollo astratto denominato **GIOP** (***General Inter-ORB Protocol***). I messaggi GIOP possono essere inviati sfruttando un qualsiasi protocollo di trasporto *connection-oriented*, come il TCP/IP. Si può , quindi, affermare che IIOP deriva dalla congiunzione tra GIOP e TCP/IP.

Jade , in accordo con le specifiche FIPA, supporta il protocollo IIOP per la comunicazione tra piattaforme ad Agenti. Questo meccanismo viene sfruttato sia per interagire con altre piattaforme Jade sia con piattaforme non-Jade. La completa trasparenza con cui Jade opera relativamente al passaggio di messaggi tra piattaforme multi-Agente consente al programmatore di non preoccuparsi minimamente di IIOP. E' Jade che si incarica di scegliere le tecnica di comunicazione più efficiente, sia essa un evento Java, un oggetto RMI o CORBA/IIOP, in maniera completamente automatica.

Uno dei compiti lasciati agli sviluppatori dell'applicazione ed agli amministratori della piattaforma è, invece, l'identificazione degli Agenti, nota col termine di *Naming*. Un identificatore d'Agente, infatti, deve includere un set di URL (Universal Resource Locator) riportante gli indirizzi dove può essere contattato. Ogni Agente Jade eredita gli

indirizzi della sua piattaforma di origine mentre il suo ***AID*** (Agent IDentifier), che contiene l'indirizzo locale (senza specificare la AP), viene generato automaticamente dal sistema non appena avviene l'invio di un messaggio.

Poiché la maggior parte delle implementazioni basate su *CORBA ORB* non permettono ancora la scelta di termini significativi come parole chiave, Jade ricorre allo schema di Naming suggerito da FIPA e basato sullo standard OMG che utilizza l'**IOR** per indicare la locazione di un'Agente. L'IOR non è altro che una rappresentazione ASCII di un *reference* ad un oggetto CORBA. Quindi un indirizzo valido potrebbe essere sia un URL come *iiop://fipa.org:50/acc* che un IOR come *IOR:00000001649444c644f4…*

Le rapresentazioni ***URL-Based*** e ***IOR-Based*** sono assolutamente equivalenti, anche se la prima risulta particolarmente più facile e comprensibile per il programmatore. Jade genera indirizzi IOR-Based che possono però essere convertiti nei corrispettivi URL-Based se contenenti solo caratteri riconoscibili dal *FIPA-Compliant Parser*. Al momento dell'attivazione la piattaforma Jade mostra il proprio indirizzo IOR sullo Standard Output (in genere lo schermo) ed, inoltre, lo scrive in caratteri ASCII in un file chiamato *Jade.ior.* Al contrario l'indirizzo URL viene salvato in un file denominato *Jade.url*. Entrambi i file vengono memorizzati nella cartella principale dove Jade viene eseguito. Gli indirizzi di tutti gli Agenti della piattaforma includono questo IOR , così da permettere un'univoca identificazione sia degli Agenti che della piattaforma nei processi di comunicazione con altre piattaforme.

4.6 Le nuove versioni di Jade

Le novità delle ultime versioni di Jade (*2.0, 2.01, 2.1,…*) vanno giustificate e correlate all'intento degli sviluppatori di adeguarsi alle nuove specifiche FIPA rilasciate alla fine dell'anno 2000. I cambiamenti introdotti, purtroppo, hanno reso incompatibile la nuova piattaforma con gli agenti funzionanti sulle versioni precedenti, creando non pochi inconvenienti.

Le nuove caratteristiche sono già state descritte nel corso di questo capitolo, per cui risulta poco significativo ripeterle. E' però, importante sottolineare, per evitare confusione, che nella versione di Jade utilizzata per questa Tesi, la 2.01, sono stati introdotti una nuova struttura gerarchica delle classi, sono stati modificati numerosi metodi e ne sono stati introdotti di nuovi. Per una completa trattazione degli stessi si rimanda al manuale di programmazione [JAD00].

Nel prossimo capitolo verranno descritte le principali tecniche di interfacciamento client-server prese in considerazione per la realizzazione del sistema multi-Agente.

Capitolo 5
TECNICHE DI PROGRAMMAZIONE CLIENT-SERVER

Affinchè l'utente possa accedere a dei servizi attraverso Internet e, non solo, ad esempio, attraverso dei terminali self-service, è indispensabile la perfetta integrazione tra la programmazione web classica (e supportata dalla maggior parte dei browsers in commercio) e il software che implementa l'architettura ad Agenti. Nei prossimi paragrafi vengono descritte brevemente le tecniche prese in considerazione per la programmazione client/server.

5.1 Client-Side e Server-Side

Il funzionamento classico di un'architettura Client-Server prevede che il Client, attraverso un software denominato Browser, effettui una richiesta ad un Server, il quale, tipicamente dopo aver effettuato un accesso al proprio File System, risponde. Esistono due modalità di programmazione per lo sviluppo di applicazioni web nelle quali il servizio fornito non sia ridotto al trasferimento di pagine statiche. Queste due modalità si riferiscono al lato dell'architettura sul quale viene eseguita l'elaborazione e sono denominate *Server-Side* e *Client-Side*.

- ***Server-Side***: in questa modalità il Server, che sulla base della richiesta ricevuta dal Client, cerca un programma nel suo File System, lo esegue (i file eseguibili sono chiamati *Back-End*),

prende l'output del programma e lo invia come risposta al Client. Da notare che nel Server side il programma può essere scritto con qualsiasi linguaggio. E' evidente come eseguendo un'applicazione, ci sia la possibilità di interagire con i file richiesti e come questi acquisiscano dinamicità, con le più svariate possibilità che un programma può offrire, come per esempio l'aggiornabilità immediata, la consultazione di database, ecc.

- ***Client-Side***: ci sono due diverse possibilità per questo tipo di architettura: nella prima il Client invia la richiesta al Server, questi risponde inviando un file contenente un programma che, quando la pagina viene visualizzata dal browser, è capace di attivare un'applicazione che già risiede sul Client, un elaboratore testi, un foglio elettronico o qualsiasi altro. Nella seconda le pagine HTML inviate al Client nascondono del software che viene eseguito ed il cui risultato viene mostrato dal browser. Ovviamente è necessario che il browser stesso sia in grado di eseguire il software accluso.

Nella Tabella 1 riportata di seguito vengono riassunte le caratteristiche tipiche della programmazione *Client-Side* e *Server-Side*.

Java viene spesso descritto come un linguaggio per programmazione client side, con riferimento al suo bytecode “sicuro” e scaricabile dai browsers. Vediamo ora come Java rappresenti anche una buona soluzione (sotto opportune condizioni) per l'altra faccia della medaglia, la programmazione server side.

La Web Programming è nata con la server side programming. Quelli che in origine erano statici documenti HTML, sono divenuti, grazie al server side programming, pagine dinamiche costruite on the fly per processare moduli e visualizzare i risultati di interrogazioni di database. Il primo linguaggio usato per rendere interattive le pagine Web è stato il

Perl, sfruttando la specifica nota come Common Gateway Interface (vedi paragrafo 5.4).

Client-Side	Server-Side
Le pagine appaiono dinamiche direttamente nel browser.	Le pagine sono generate dinamicamente dal server, ma una volta inviate al browser sono statiche.
L'elaborazione del programma richiede risorse del calcolatore su cui il browser viene eseguito.	Il back-end è eseguito sul server e quindi non impegna risorse del calcolatore su cui il browser viene eseguito.
L'architettura complessiva del sistema browser-server rimane invariata	L'architettura complessiva del sistema browser-server deve essere modificata, per introdurre il back-end.
Il browser deve essere abilitato per eseguire il programma associato alla pagina.	Qualsiasi browser può essere utilizzato.
Il programma associato alla pagina deve essere trasferito dal server al browser, impegnando la linea di comunicazione	Il back-end rimane sul server e solo i risultati dell'elaborazione vengono trasferiti lungo la linea.
Soluzione non adatta per sistemi di consultazione di database che non possono essere trasferiti al browser per ragioni di riservatezza, o sicurezza, o dimensioni.	Soluzione adatta per back-end con funzione di sistemi di consultazione di database.

Tabella 1: Programmazione Client-Side e Server-Side.

La CGI è rimasta per anni il principale strumento per la server side web programming. Anche se oggi gli script CGI sono i più diffusi, sono da tempo comparse soluzioni tecnicamente più solide e pratiche per lo sviluppo di applicazioni Web di qualità professionale e respiro industriale. Citiamo ad esempio il *Server JavaScript* di Netscape, le *Active Server Pages* di Microsoft o l'*IntraBuilder* di Borland. Java deve

gran parte del suo iniziale successo alle *applet*, che insieme ai plugin/ActiveX e al JavaScript/VBScript rappresentano la parte client side del web programming.

Tuttavia, sviluppare servizi accessibili via web significa nella maggior parte dei casi dover gestire aspetti di programmazione riguardanti tanto il lato server quanto il lato client. Facciamo un esempio: supponiamo di dover generare un grafico utilizzando dei dati contenuti in un database. Una buona soluzione è generare una pagina HTML contenente i parametri per una applet che visualizza il grafico. La parte server quindi interroga il database e costruisce una pagina HTML contenente la descrizione del grafico, ma la visualizzazione vera e propria viene gestita dalla applet che gira sul client.

Data la necessità di dover considerare entrambe le facce della medaglia, è naturale che molti sviluppatori desiderino utilizzare un solo linguaggio per lo sviluppo di applicazioni sia lato client che lato server. Chiunque abbia tentato di fare server side programming in Java avrà scoperto che sebbene Java sia un ottimo linguaggio per molti usi, di per sé non si presta particolarmente allo sviluppo di applicazioni CGI, perché non fornisce nessuno supporto specifico. Alle esigenze di rendere pratica la server side programming in Java, JavaSoft ha dato una risposta, da molti considerata valida e, per certi versi definitiva: i S*ervlet*.

5.2 Servlet

Un Servlet non è altro che un'applicazione Java che risiede accanto al Web Server, estendendone le funzionalità di base [PUL98]. E' qualcosa di molto simile ad uno script CGI (cioè uno script o un programma eseguito sul server), con la differenza che, essendo scritto in Java, ne eredita tutte le caratteristiche (sicurezza, portabilità,…) e viene gestito da

un apposito *class loader*. Quest ultimo particolare permette una discreta ottimizzazione sia della gestione delle risorse (contrariamente agli script CGI, l'esecuzione di uno o più servlet non richiede la creazione di processi appositi, ma solo il caricamento di una JVM in esecuzione), sia dei tempi di attivazione (il meccanismo tipico di Java di class loading, permette di caricare la classe solo nel momento dell'effettivo bisogno, e di lasciarle in memoria per le richieste successive).

Senza entrare nei dettagli tecnici su come si realizza un Servlet, il motivo fondamentale per cui questa tecnica è stata scartata, nella fase di realizzazione del sistema multi-Agente, è la necessità di installare sul proprio server il *Web Server Jeeves* prodotto da JavaSoft oppure, in alternativa, il *Java Servlet development Kit* (*JSDK*) di Sun. Questo kit, analogamente al JDK, contiene classi, documentazione ed un esecutore di servlet standalone; inoltre include dei componenti per consentire l'utilizzo delle servlet da altri Web Server molto diffusi: in particolare, i Web Server di Netscape e Microsoft, ed il diffusissimo Web Server freeware Apache. Si tratta comunque di un kit molto essenziale, non semplice da utilizzare e che presenta alcuni problemi nell'uso pratico. Analogamente al JDK, è solo una implementazione di riferimento gratuita, pronta ad essere ottimizzata da terze parti per l'uso commerciale.

5.3 JSP (Java Server Pages)

Si tratta di una tecnologia che agli occhi del programmatore viene gestita per mezzo di un linguaggio script, e che è in grado di mescolare codice HTML, componenti riusabili (JavaBeans), applicazioni remote (servlet), codice Java, e script Java-like. Si potrebbe dire che le Java Server Pages rappresentano il collante per unire insieme una serie di oggetti differenti fra loro. La possibilità di utilizzare i componenti come

parte integrante del codice HTML, permette in maniera estremamente semplice di aumentare le potenzialità e la flessibilità, senza la necessità di conoscere complessi linguaggi di programmazione come Java.

E' possibile, inoltre, inserire codice Java all'interno della pagina: questo codice verrà poi gestito dal server che provvederà alla sua compilazione ed esecuzione, per generare pagine dinamiche o effettuare ricerche in database, ad esempio. Questo fatto permette al programmatore di inserire parti di codice Java in modo molto simile a come farebbe con script JavaScript, VBScript o simili.

Come conseguenza si ha un ulteriore impulso alle performance ed alla portabilità dato che il codice prodotto viene eseguito sul server, e non nel browser.

Per eseguire un JSP si deve utilizzare un web server in grado di supportare tale tecnologia, oppure far uso di un cosiddetto application server che estenda le funzionalità del web server.

Ad esempio *Java Server Web Development Kit* (*JSWDK*) della Sun Microsystems o *Jacarta Tomcat 3.1*, sono due prodotti molto diffusi in grado di offrire funzionalità sia per i servlet che per JSP.

Anche questa soluzione comporta , quindi, l'installazione sul proprio server di appositi pacchetti software che spesso generano problemi, sia in fase di configurazione sia in esecuzione, e pertanto è stata abbandonata.

Nonostante ciò è stato molto interessante osservare come il loro impiego si integri in modo innovativo con gli Agenti Software Jade.

Secondo gli studi di Daniel Le Berre dello CSELT [LEB00], infatti, può risultare conveniente implementare un'Agente che resta in attesa di una connessione in una determinata pagina web "sensibile" ed avverte quando se ne verifica una.

Il modello di riferimento si fonda sull utilizzo di tre Agenti, detti *Snooper*, *Buffer* e *Client*. Snooper è l'Agente che "vive" nella pagina web sensibile, Client è, invece, quello che deve mostrare al sistema le

informazioni inerenti le connessioni alla pagina web, mentre Buffer si occupa di tenere traccia di tali informazioni in caso di ritardi e di verificare che il Client le abbia effettivamente ricevute. In figura 5.1 uno schema dei tre Agenti.

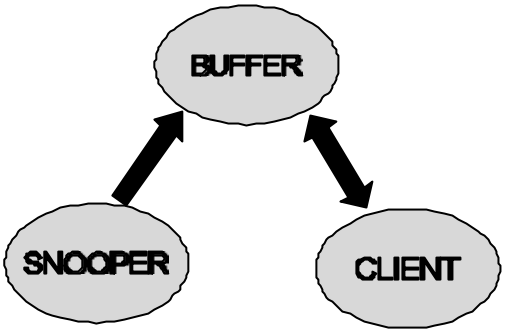

Fig. 5.1 : I tre Agenti Jade Snooper, Buffer e Client

Un esempio di codice Java che implementa i tre Agenti è mostrato nell'appendice A. Per essere in grado di conoscere lo stato del client , viene atteso, per un determinato numero di secondi, un messaggio di conferma (Reply) senza il quale si suppone che il client non sia più connesso al server. In questo caso il messaggio contenuto nell'Agente Buffer non viene più aggiornato. L'architettura del sistema Jade-JSP è quella della figura 5.2.

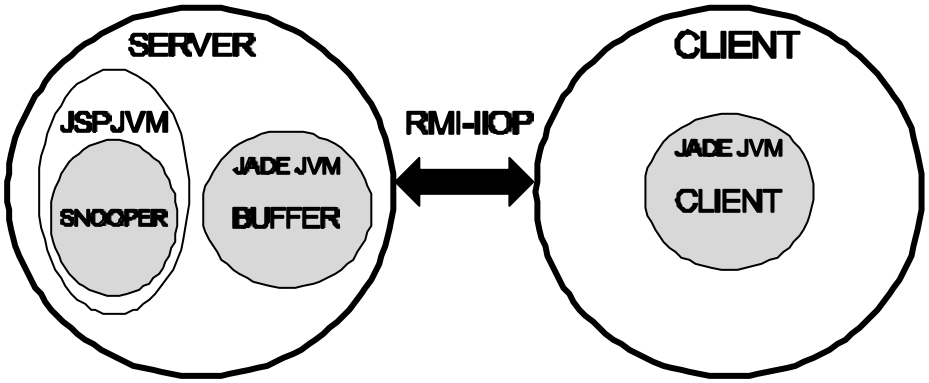

Fig. 5.2 : Modello del sistema Jade-JSP

Il server contiene due differenti JVM: una per l'esecuzione dell'applicazione JSP/Servlet ed un contenitore Jade , un'altra per la Piattaforma Jade.

Il client esegue un'altra istanza della JVM per attivare un secondo contenitore d'Agenti Jade.

Un'interessante aspetto di questo modello è che la AP e l'Agente Buffer sono dissociati dalla pagina JSP. Ciò significa che la cancellazione/perdita della pagina JSP e dell'Agente Snooper non comporta la perdita dell'informazione mantenuta dal Buffer.

Per maggiori informazioni si rimanda al sito web dello CSELT, nella sezione dedicata alle novità [JAD00].

5.4 R.M.I. (Remote Method Invocation)

La chiamata di metodi remoti, o RMI, si utilizza per creare applicazioni Java che possano comunicare nella rete con altre applicazioni dello stesso tipo. Più in dettaglio, l'RMI permette ad un'applicazione di richiamare metodi ed accedere a variabili di un'altra applicazione, eseguita in un ambiente od in un sistema differente, e di trasferire oggetti da una all'altra attraverso una connessione di rete. L'RMI è un mezzo di comunicazione molto più sofisticato rispetto ad una semplice connessione *socket*, perché permette di strutturare programmi Java che non solo eseguono thread in parallelo su di una stessa macchina, ma possono eseguire thread in parallelo su macchine differenti [LEM98].

Gli obiettivi dell'RMI sono di integrare in Java un modello a oggetti distribuiti senza sconvolgere il linguaggio o il modello a oggetti esistente, e di rendere l'interazione con un oggetto remoto semplice come quella con oggetti locali. Ad esempio, si è in grado di utilizzare oggetti remoti esattamente come se fossero oggetti locali (assegnarli a variabili, passarli

ai metodi come argomenti e così via), e la chiamata di metodi remoti dovrebbe essere uguale a quella di metodi locali. Comunque RMI comprende anche meccanismi più sofisticati, per richiamare metodi su oggetti remoti e passare interi oggetti o parti di essi utilizzando, a seconda dei casi, un riferimento o un valore; comprende anche alcune eccezioni aggiuntive per la gestione degli errori di rete che possono sorgere durante le operazioni che coinvolgono sistemi remoti.

L'RMI è strutturato secondo diversi livelli, e la semplice chiamata di un metodo attraversa più di un livello per giungere al risultato (Figura 5.3). Tra questi i più significativi sono:

- I livelli "stub" e "scheletro" rispettivamente sul client e sul server. Questi livelli si comportano come oggetti surrogati, da entrambi i lati, e nascondono il fatto che la chiamata coinvolge metodi remoti. Ad esempio, l'applicazione client può richiamare un metodo remoto esattamente allo stesso modo di un metodo locale, dato che l'oggetto stub è un surrogato locale dell'oggetto remoto.
- Il livello di riferimento remoto (*Remote Reference Layer*), che gestisce il confezionamento della chiamata ad un metodo: riceve la chiamata e i relativi parametri e li restituisce sotto forma di valori trasportabili attraverso la rete. Questo livello permette di individuare la natura di un oggetto, ovvero se risiede su di una macchina oppure e' distribuito su di una rete.
- Il livello di trasporto (Transport Layer), che e' sostanzialmente un traduttore che trasforma il codice RMI in un codice che può essere inviato attraverso la rete (tramite il meccanismo di Java detto *serialization*). Il livello trasporto si appoggia poi sull'effettivo servizio di trasporto reso disponibile dal sistema operativo (tipicamente basato sul protocollo TCP).

Fig. 5.3 : I livelli di Internet per RMI.

L'architettura RMI strutturata su diversi livelli permette di creare applicazioni indipendenti dalle piattaforme o dagli ambienti software utilizzati. I livelli stub e scheletro permettono al client e al server di comportarsi come se gli oggetti di cui si stanno occupando fossero locali, e quindi di utilizzare le stesse risorse del linguaggio Java di cui si fa uso per accedere a tali oggetti.

Infine il livello di trasporto di rete viene utilizzato indipendentemente dagli altri due ed è così possibile l'uso di diversi tipi di connessione socket per l'RMI (TCP, UDP oppure TCP con altri protocolli, come SSL).

Quando un'applicazione client effettua una chiamata a un metodo remoto, la chiamata passa al livello stub e poi al livello di riferimento, che converte gli argomenti, se necessario, e poi passa la chiamata al server per mezzo del livello di trasporto e dei sottostanti; il livello di riferimento del server estrae gli argomenti e li passa al livello scheletro e

successivamente all'implementazione del server. I valori restituiti dal metodo richiamato compiono il tragitto inverso per tornare al client.

Gli oggetti remoti di Java utilizzati come parametri o valori restituiti vengono passati tramite un riferimento, proprio come se fossero locali. Altri oggetti, invece, vengono copiati. Si noti che queste differenze di impiego influenzano il modo con cui vengono scritti i programmi Java contenenti chiamate a metodi remoti: ad esempio, non si può passare un array come argomento a un metodo remoto, farlo elaborare dal metodo e aspettarsi che la copia locale dell'array sia automaticamente modificata. Gli oggetti locali non funzionano così, quando tutti gli oggetti vengono passati come riferimenti.

L’RMI è stato preso in considerazione per questo lavoro di Tesi alla luce della possibiltà di realizzare un’interfaccia utente basata su applets Java che consentisse ad un Agente di essere eseguito sul client. Purtroppo questa alternativa è stata accantonata perché necessita di avere JADE ed il JDK installati sul client.

5.4 CGI (Common Gateway Interface)

Data la natura stessa del progetto e l'obiettivo di realizzare un servizio accessibile tramite un comune browser (Microsoft Internet Explorer, Netscape Navigator,…) la tecnica di interfacciamento client/server risultata vincente si basa sullo standard CGI.

CGI (Common Gateway Interface) è uno standard, un insieme di regole, per l'interfacciamento di applicazioni client/server [GOL98] [CGI96] [SJL97].

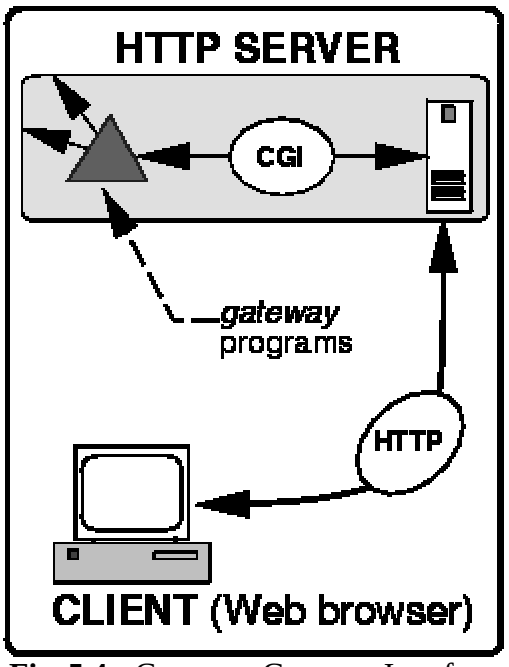

Fig. 5.4 : Common Gateway Interface.

Un semplice documento HTML, che il server restituisce quando sopraggiunge una chiamata dal browser client, è un oggetto statico. Un programma CGI, invece, viene eseguito sul server in risposta alle richieste ricevute, in modo tale da restituire un'informazione dinamica.

L'interfaccia CGI è stata definita allo scopo di standardizzare i metodi

di decodifica dei dati in arrivo al server e di uniformare tutta la produzione software futura..

Ad un programma CGI, per assolvere alle funzioni per le quali è stato progettato, devono avere associati dei permessi tali che chiunque possa richiederne l'esecuzione sul sistema sul quale il Web server è installato. Questo, chiaramente, implica di dover ricorrere ad alcune precauzioni, in termini di configurazione del server e di implementazione del programma, al fine di garantire la sicurezza del sistema stesso.

I programmi CGI devono risiedere in una directory speciale, in modo che il server Web possa decidere di procedere nell'esecuzione del file eseguibile piuttosto che limitarsi a visualizzarne il contenuto sul navigatore (browser). Questa directory è, solitamente, sotto il diretto controllo del webmaster, che inibisce all'utente generico la creazione di programmi CGI in essa. In generale, la directory nella quale risiedono i programmi CGI viene denominata "/cgi-bin". Un altro modo con il quale il server identifica un file come eseguibile, invece che come immagine, testo o codice html, è l'estensione .cgi nel nome del file. Essa non è quindi usata, come potrebbe sembrare, per denotare il tipo di linguaggio con il quale il codice sorgente del programma è stato scritto; un programma CGI può essere infatti scritto in qualsiasi linguaggio, che ne consenta l'esecuzione sul sistema.

La dinamica di funzionamento è la seguente:

- Il client richiede al server l'esecuzione di uno specifico programma CGI.
- Il server riceve i dati e li passa al programma CGI.
- Il software li elabora e li restituisce al browser sotto forma di documento HTML, testo, ecc.

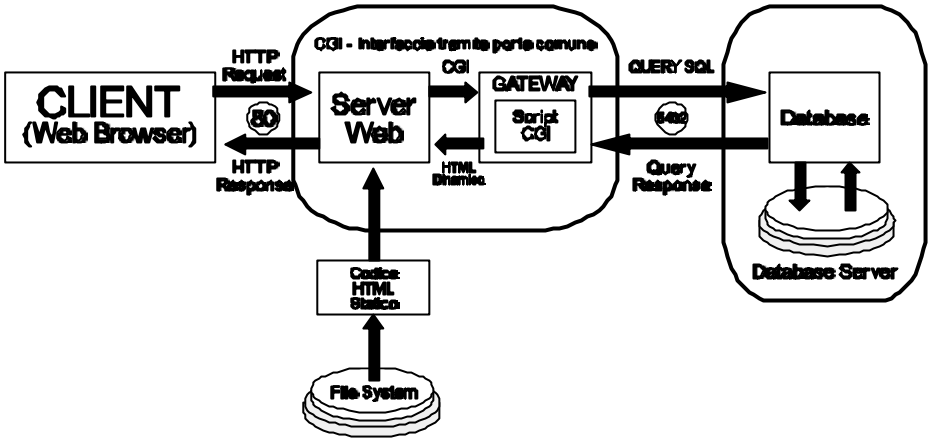

Fig. 5.5: Esempio di uso dell'Interfaccia Common Gateway per l'accesso a database attraverso la rete.

All'interno dei documenti HTML, sono presenti delle strutture attraverso le quali è possibile richiedere l'esecuzione di programmi CGI. Queste sono:

1. i moduli (form),
2. la struttura Isindex,
3. il link diretto al nome del programma CGI.

Per permettere che l'output del programma CGI sia dinamico è necessario consentire al client di specificare, unitamente all'invocazione del programma, dei parametri di ingresso. Ci sono due famiglie di parametri che il programma riceve e può utilizzare:

1. le variabili ambientali: parametri definiti dai sistemi client e server in uso,
2. i parametri specificati dall'utente, i cui valori vengono inseriti in strutture appositamente definite nell'ambito della pagina Web.

Variabili Ambientali (Environment Variables): Il server definisce delle speciali variabili: sono memorizzate nell'ambiente del sistema, e per questo risultano facilmente accessibili. Tali variabili vengono ricavate dalle informazioni inviate dal browser e rese disponibili al programma CGI in esecuzione sul server. Esse danno informazioni sul browser (tipo, files supportati, nome dell'host client, e altri), sul server (nome, versione, porta del socket, e altri) ed anche sullo stesso programma CGI (nome del programma e path). Le più importanti variabili d'ambiente CGI sono:

- SERVER_SOFTWARE: il nome e la versione del server http che risponde alla richiesta cgi;
- SERVER_NAME: il nome del server http che può essere un alias di DNS o un indirizzo IP numerico;
- GATEWAY_INTERFACE: la versione delle specifiche CGI;
- SERVER_PROTOCOL: il nome e la versione del protocollo di comunicazione HTTP;
- SERVER_PORT: il numero di porta a cui è indirizzata la richiesta (porta 80, generalmente);
- REQUEST_METHOD: il metodo usato per la richiesta (GET, POST, ed altri);
- SCRIPT_NAME: il path completo del programma CGI;
- QUERY_STRING: contiene i dati passati dalle richieste GET, ISINDEX, ISMAP;
- REMOTE_HOST: il nome dell'host che invia la richiesta;
- REMOTE_ADDR: l'indirizzo IP dell'host che invia la richiesta;
- AUTH_TYPE: il metodo usato dal protocollo di autenticazione dell'utente. Questa variabile è settata solo se il server supporta tale protocollo e solo se lo script cgi è protetto;

- REMOTE_USER: il nome dell'utente autenticato. Questa variabile è settata solo se il server supporta il protocollo di autenticazione e se lo script cgi è protetto;
- CONTENT_TYPE: per le richieste POST e PUT questa variabile contiene il tipo MIME dei dati passati al cgi;
- CONTENT_LENGTH: la lunghezza dei dati passati con il metodo POST;
- HTTP_ACCEPT: i tipi MIME accettati dal client;
- HTTP_USER_AGENT: il nome e la versione del browser che ha inviato la richiesta.

Parametri Utente: Esistono due metodi che possono essere usati per passare dei parametri in ingresso al programma CGI:

1. il metodo GET,
2. il metodo POST.

Per determinare quale metodo viene usato, il programma CGI controlla la variabile ambientale REQUEST_METHOD, che può essere impostata con valore pari a GET oppure a POST. Se è impostata a POST, allora la lunghezza dell'informazione trasmessa è contenuta nella variabile d'ambiente CONTENT_LENGTH.

***Metodo* GET**: I dati inseriti dall'utente nel modulo (form) , appositamente definito all'interno della pagina HTML che l'utente aveva precedentemente richiamato, vengono inviati dal browser come parte dell'URL richiesto; ovvero vengono "attaccati" (appended) all'URL originale (specificato nel modulo dall'attributo ACTION) e comunicati in questo modo al server.

Il server li memorizza nella variabile d'ambiente QUERY_STRING, che verrà poi messa a disposizione del programma CGI.

Quando, ad esempio, si seleziona "Submit" nella pagina HTML, il contenuto del form viene elaborato dal browser, e viene inviata una query che si presenta nella forma:

```
http://nome.del.server/cgi-bin/    ¿
programma?nome1=valore1&nome2=valore2&nome3=valore3…
```

***Metodo* POST**: Il contenuto del modulo viene codificato esattamente come per il metodo GET; quello che cambia è il modo in cui i dati vengono inviati al server. Anzichè aggiungere il contenuto del modulo all'URL, esso viene spedito in un blocco di dati e-mail: ovvero tutti i campi (e relativi valori presenti nel form) vengono riuniti ed elaborati per formare uno o più messaggi e-mail.

L'attributo ACTION, se specificato, conterrà l'URL al quale il blocco di dati verrà spedito.

Esecuzione: Ogni volta che un client richiede l'URL corrispondente ad un programma CGI, il server lo esegue. L'uscita del programma viene poi inviata, più o meno direttamente, al client. Esistono varie tecniche per eseguire programmi CGI sul server web. Le più note sono:

- ***Chiamata Diretta.*** Tramite una linea di comando virtuale (la riga nel browser dove si specifica l'URL da visitare) è possibile eseguire direttamente un programma CGI. Attraverso gli argomenti resi disponibili con la linea di comando le informazioni vengono decodificate dal server e passate al programma, che le può utilizzare , ad esempio, per la ricerca in una banca-dati.
- ***Isindex.*** Questo è il secondo modo per poter eseguire un programma CGI sul server Web. Esso non ha bisogno di essere inserito all'interno di un modulo, nè di specificare il nome o il

valore. Permette, infatti, il passaggio di un solo parametro, senza nome, ma solo con il suo valore. La forma dell'URL che viene composto all'invio dei dati è: *http://nome.del.server/cgi-bin/programma?valore* . Con l'elemento ISINDEX, l'invio del comando di esecuzione del programma CGI avviene premendo il tasto invio della tastiera, dopo l'immissione del paramentro nella finestra creata da ISINDEX.

- ***Moduli HTML.*** I moduli (form) rappresentano una delle possibili strutture presenti all'interno di un documento HTML. Sono dei raccoglitori nei quali vengono inserite istruzioni che definiscono dei campi per l'inserimento di dati. All'interno dei moduli possono essere inseriti campi di input, tasti e caselle di testo. L'invio del comando di esecuzione del programma CGI avviene "cliccando" il tasto "Submit", dopo aver immesso i vari parametri nelle diverse finestre visualizzate nel form stesso. Il tasto Submit è presente convenzionalmente alla fine del modulo, dentro la finestra del navigatore.

Uscite Prodotte: Il programma CGI trasmette i risultati al client (browser) rendendoli disponibili in formato convenzionale (tipicamente html), sullo standard output (stdout) del programma stesso. Può anche creare delle intestazioni proprie o tipiche del server risparmiando la fatica a quest'ultimo.

I programmi CGI hanno, pertanto, un ruolo di Response. Possono restituire una grande varietà di tipi di documenti (immagini, documenti HTML, semplici files di testo, ...), oppure riferimenti ad altri documenti. Il client deve, in ogni caso, sapere che tipo di documento gli è stato restituito, per poterlo trattare opportunamente. Il programma CGI deve quindi comunicare al server quale tipo di documento sta restituendo: il

protocollo CGI richiede, a questo scopo, la presenza di un'intestazione (header) nell'output.

Come sarà chiaro nel capitolo 7, in questo lavoro di tesi si sfrutta la tecnica CGI per attivare in remoto un agente di interfaccia, scritto in Java, passandogli una serie di parametri, utili alla fornitura di un servizio.

Capitolo 6

IL PROGETTO

Dopo aver affrontato le tematiche relative agli Agenti Software, agli standard ed agli strumenti di sviluppo, in questo capitolo viene esposto il progetto di un'architettura multi-Agente, argomento di questo lavoro di Tesi.

6.1 Introduzione

Il sistema multi-Agente, descritto nelle prossime pagine, fa parte di un complesso progetto del gruppo di ingegneria della conoscenza ed interazione uomo-macchina, volto allo sviluppo di un prototipo di sistema software basato sulla tecnologia ad Agenti, per la fornitura di servizi ad una comunità di studenti/docenti in ambito universitario.

L'idea di base è quella di costruire progressivamente il sistema utilizzando standard aperti e distribuendo i compiti tra numerose entità autonome: gli Agenti.

L'utilizzo di questo tipo di tecnologia dovrebbe permettere l'integrazione di servizi via via più evoluti, basati su Agenti autonomi ed "intelligenti" accanto a quelli più tradizionali, in un quadro unitario.

Nell'approccio proposto nessun Agente possiede capacità sufficienti per gestire autonomamente dei servizi ma, dalla cooperazione tra gli Agenti, si può giungere alla realizzazione di sistemi integrati, anche costruiti dinamicamente, e veramente utili agli utenti.

L'architettura del sistema proposto, denominato ***AteneoOnLine Multi-Agent System*** (*AOL-MAS*), è stata concepita nel rispetto delle specifiche FIPA2000 ed è composta da diverse entità, analizzate nel dettaglio nei paragrafi a seguire.

6.2 Il Dominio considerato

Nonostante il tema principale di questo progetto riguardi l'interfacciamento tra l'utente ed il sistema multi-Agente e le tecniche di comunicazione e scambio informazioni, un aspetto non trascurabile per la realizzazione di test significativi e per l'integrabilità della struttura con gli elementi software già realizzati (vedi tesi [MAS00], [GAR00]), è determinato dal Dominio di applicazione. L'obiettivo finale, infatti, consiste nella creazione di un sistema per la fornitura di servizi via rete ad una vasta comunità di utenti. Si è pertanto posta la necessità di scegliere, per la sperimentazione, un esempio di tale comunità.

I sistemi distribuiti basati sulla tecnologia ad Agenti, inoltre, sono caratterizzati dalla presenza di numerose piattaforme (APs), localizzate in un ambiente eterogeneo e su diversi Host. Inoltre, per garantire un funzionamento continuo e fault-tolerant, è indispensabile che l'architettura sia strutturata secondo una ben specifica gerarchia, facendo in modo tale che una unità malfunzionante non pregiudichi il funzionamento del sistema.

Tale visione comporta la necessità di individuare univocamente non solo il bacino d'utenza ma, soprattutto, i rapporti esistenti tra le diverse piattaforme.

Per tutti questi motivi si è deciso di considerare come Dominio applicativo la comunità universitaria, suddivisa in una serie di entità più piccole quali, per esempio, le diverse facoltà di un Ateneo, i possibili

utenti (studenti, docenti, personale di servizio,…). Ognuna di esse viene rappresentata, nel modello di riferimento (vedi fig. 6.1), da una piattaforma ad Agenti. Le varie APs sono in relazione tra loro secondo una gerarchia ad albero.

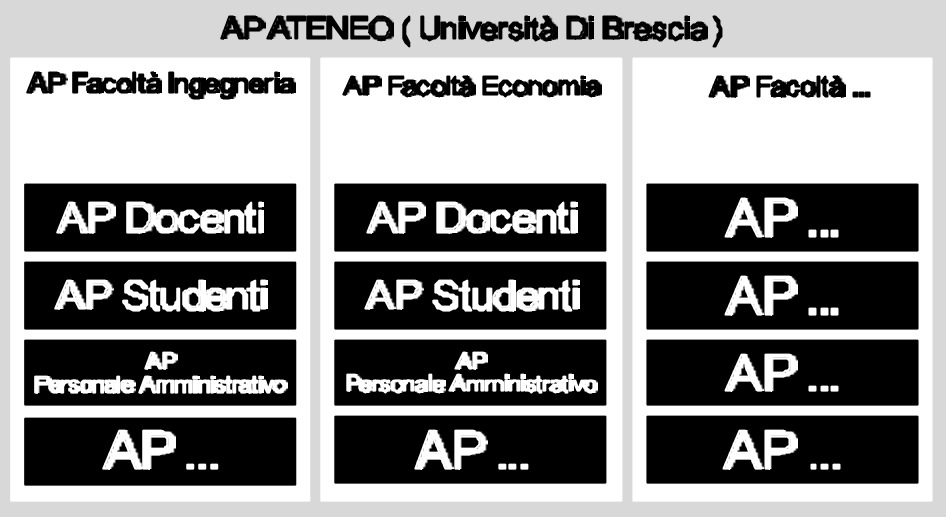

Fig 6.1 : Modello della struttura multi-Piattaforma.

La AP Ateneo è la radice della struttura realizzata anche se, per la scalabilità dell'architettura, vi è la possibilità di aggiungere altri livelli superiori. Nei DFs delle APs figli (Studenti, Docenti,…) si possono registrare i DFs delle Aps genitori (per esempio il DF della Facoltà di Ingegneria) e così via risalendo la gerarchia. Ogni Piattaforma offre servizi differenti a seconda del tipo di utenza. Così gli studenti si servono degli Agenti residenti sulla AP a loro dedicata per compilare piani di studio, acquistare testi, cercare informazioni su docenti ed esami,…

Per i Docenti, invece, ci possono essere servizi per agevolare adempimenti burocratici, per l'accesso a regolamenti univeritari, per l'organizzazione di meeting, …

6.3 Il Modello di riferimento

Lo scopo di questo lavoro non è quello di costruire ed amministrare una piattaforma complessa come quella mostrata in fig. 6.1, pertanto l'architettura effettiva a cui si è fatto riferimento nello sviluppo è quella, più semplice, di figura 6.2 dove viene considerato un sistema ad Agenti composto da una AP genitore (*Ateneo*), una AP figlio (*Facoltà di Ingegneria*) ed una AP nipote (*Servizi Studenti*).

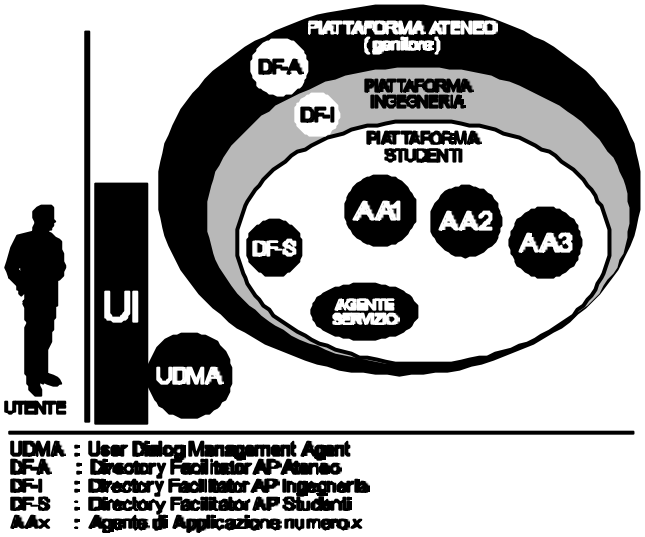

Fig. 6.2 : Modello di Riferimento per il sistema *AOL-MAS*.

Questo approccio, sebbene non indispensabile, si è reso utile per una più completa simulazione del sistema Multi-Agente. Per circoscrivere il dominio di studio si è, inoltre, deciso di fissare l'attenzione solo su alcuni servizi destinati agli studenti.

Il meccanismo di funzionamento è piuttosto articolato:

- Uno studente accede attraverso Internet al sito web dell'Università e, dopo un'eventuale identificazione personale, accede alla pagina relativa ai servizi di cui può usufruire. Utilizzando l'interfaccia

grafico/testuale (UI) fornita dal sito stesso chiede l'attivazione di un servizio specifico e fornisce i parametri necessari alla fornitura tramite compilazione di moduli (fig. 6.3).

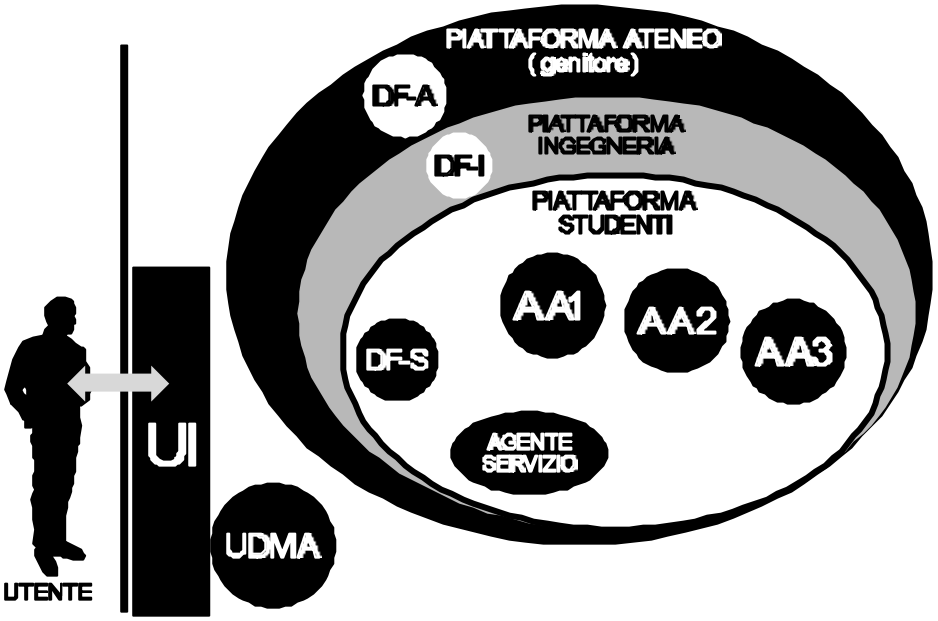

Fig. 6.3 : L'utente scambia informazioni sul servizio con l'interfaccia fornita dal Sistema MAS.

- La richiesta di fornitura servizio attiva l'UDMA (User Dialog Management System) di sistema che, raccoglie i dati inseriti nei form, verifica che si tratti di un servizio destinato agli studenti ed interroga il Directory Facilitator (DF-S) della AP Studenti, al fine di trovare un Agente in grado di risolvere il problema (fig. 6.4).

- Ogni utente è assistito dal proprio UDMA personale. Se il DF-S rivela che la piattaforma Studenti non è in grado di sostenere il servizio, è compito dell'UDMA risalire la gerarchia di APs (formulando delle *Query* ai rispettivi DFs) alla ricerca degli Agenti opportuni (fig. 6.5).

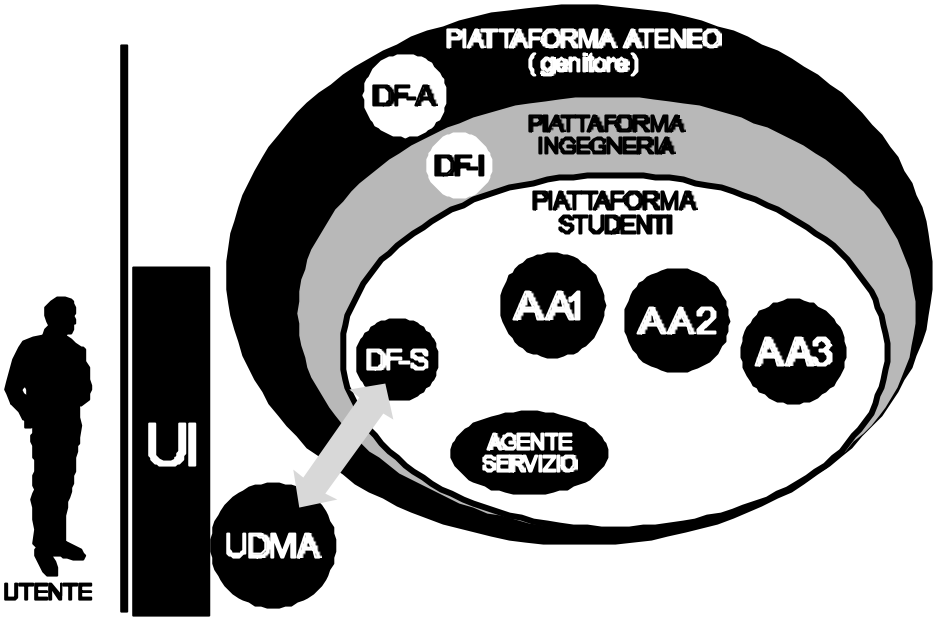

Fig. 6.4 : L'UDMA interroga il DF-S della AP Studenti.

- La piattaforma Studenti ha la priorità sulle altre AP genitori perché l'UDMA riceve fin dall'inizio istruzioni sulla tipologia di servizio. Solo in caso di insuccesso interroga il Directory Facilitator della Facoltà di Ingegneria (DF-I) e poi, quello di Ateneo (DF-A). Se nessuno dei DF ha registrato Agenti addetti al servizio, un messaggio di errore viene inviato allo studente connesso.

- In caso di successo, invece, il DF-S ritorna l'AID (*Agent Identifier*) dell'Agente di Servizio. A questo punto l'UDMA invia a tale Agente tutti i dati necessari all'espletamento della richiesta ed attende i risultati (fig. 6.6). L'Agente di Servizio riveste un ruolo di primaria importanza nel sistema. Esso, infatti, fa da intermediario tra gli Agenti residenti sulla AP e tutti gli altri Agenti che vogliono interagire o comunicare con essi. L'UDMA non sa se esistono uno o più Agenti idonei ai propri scopi, per cui si rivolge all'Agente di Servizio.

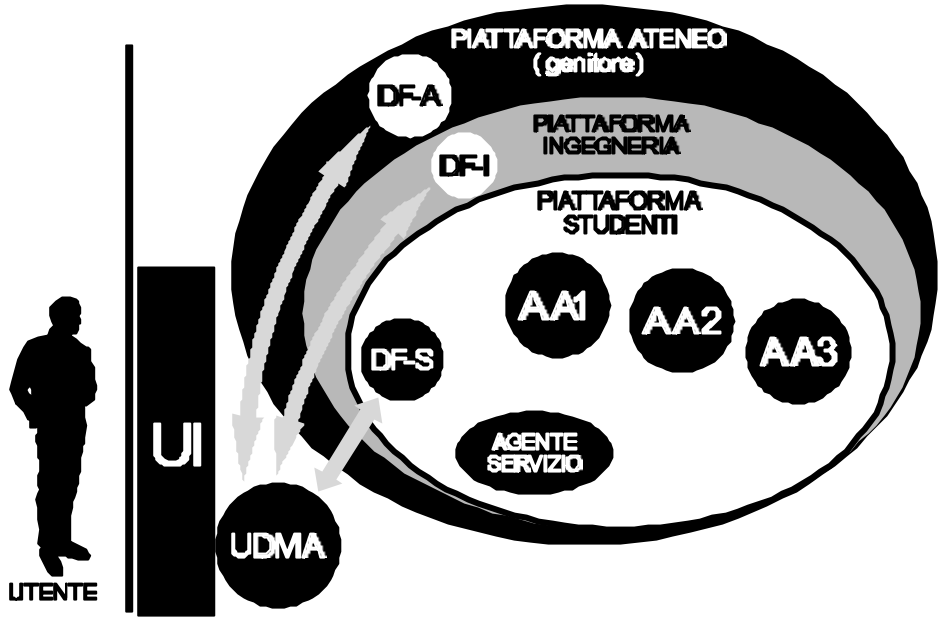

Fig. 6.5 : L'UDMA interroga i DF risalendo la gerarchia della struttura.

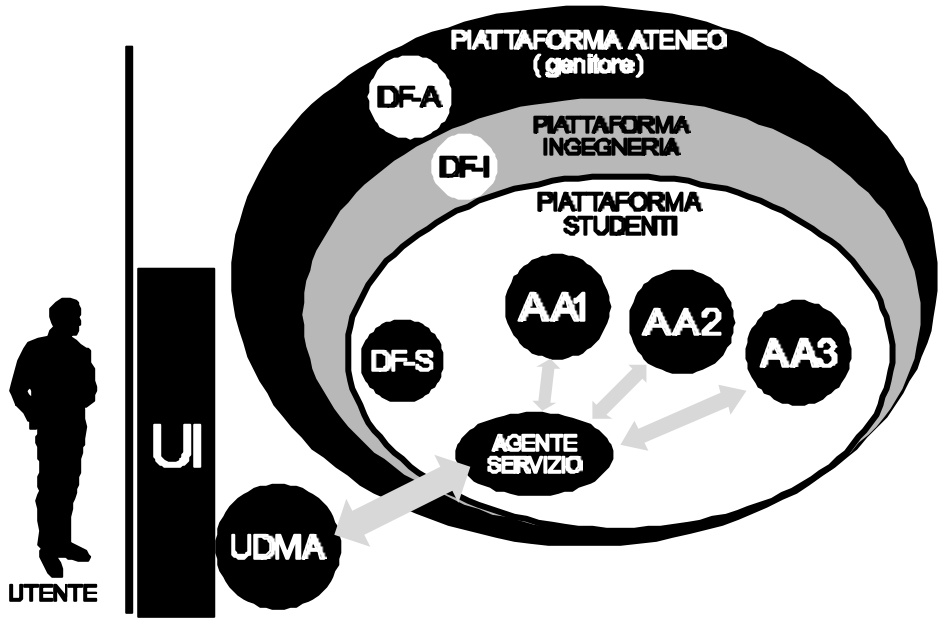

Fig. 6.6 : L'UDMA contatta l'Agente di servizio e fornisce le informazioni necessarie all'espletamento della richiesta.

- L’Agente di Servizio, una volta identificata la richiesta, contatta a sua volta il DF-S allo scopo di verificare l’esistenza di Agenti di Applicazione (AA1, AA2,…) preposti all’operazione desiderata. In caso di insuccesso si ripete la tecnica di Query ai DFs delle varie APs risalendo la gerarchia. Gli Agenti di Applicazione possono interfacciarsi solo con l’Agente di Servizio, restando all’oscuro sulle cause che hanno portato alla richiesta delle loro prestazioni.

- L’Agente di Applicazione chiamato, svolge i compiti affidatigli in completa autonomia, ad esempio interrogando i database disponibili a cui ha accesso. In questa tesi non sono stati progettati Agenti Mobili anche se, la natura distribuita dei sistemi basati su Agenti, prevede che gli Agenti possano migrare alla ricerca di informazioni.

- Quando i dati richiesti sono stati trovati, vengono spediti all’Agente di Servizio che, a sua volta, li invia all’UDMA. Quest ultimo, alla luce dei risultati prodotti decide il formato migliore per mostrarli all’utente (per esempio, generando una pagina HTML) oppure se è il caso di richiedergli ulteriori informazioni.

Gli aspetti concernenti la realizzazione ed i pacchetti software per l’implementazione di questo sistema verranno trattati nel prossimo capitolo ma, comunque, è interessante soffermarsi su alcune caratteristiche concettuali, utili a comprendere meglio il ruolo delle entità descritte precedentemente.

6.4 L'Agente di interfaccia UDMA

Nel capitolo 2 si è analizzato, con l'ausilio di vari scenari, il ruolo di questo Agente preposto alla gestione del dialogo con l'utente. Il suo nome descrive egregiamente il compito che gli è affidato. Si tratta di un ruolo piuttosto delicato perchè prevede l'integrazione tra mondi differenti, come mostrato in figura 6.7. Da un lato l'uomo, dall'altro le tecniche software classiche per l'interazione ed, infine, il mondo degli Agenti.

Fig. 6.7 : Uomo – Software No-Agenti – Software ad Agenti.

E' spontaneo chiedersi in che dominio vada collocato un Agente come l'UDMA. Esso è un ibrido che si colloca tra il mondo degli Agenti ed il software no-Agenti (o tradizionale). Il suo scopo, infatti, è quello di permettere all'uomo di comunicare, di scambiare informazioni e di interagire con gli Agenti.

Per fare ciò è indispensabile che l'Agente di interfacciamento si integri sia con il mondo degli Agenti e, quindi, con le sue tecnologie, i linguaggi, i metodi, sia con il software tradizionale, sfruttandone le capacità espressive per dialogare con l'uomo.

Per garantire un elevato livello di generalità ma, soprattutto, affinchè il sistema studiato funzioni in un ambiente distribuito eterogeneo, è stato necessario definire il linguaggio con cui l'utente effettua le richieste al sistema ad Agenti. Poiché il primo accesso avviene attraverso un sito

web realizzato con tecnologie classiche (quali HTML, applet Java,…) l'approccio migliore ha portato all'adozione di messaggi basati su stringhe in formato ASCII. E' compito, poi, dell'UDMA convertirli in un formato riconosciuto dal sistema ad Agenti. In figura 6.8 il modello di riferimento. L'UDMA può interagire col sistema ad Agenti attraverso l'intermediazione dell'Agente di Servizio.

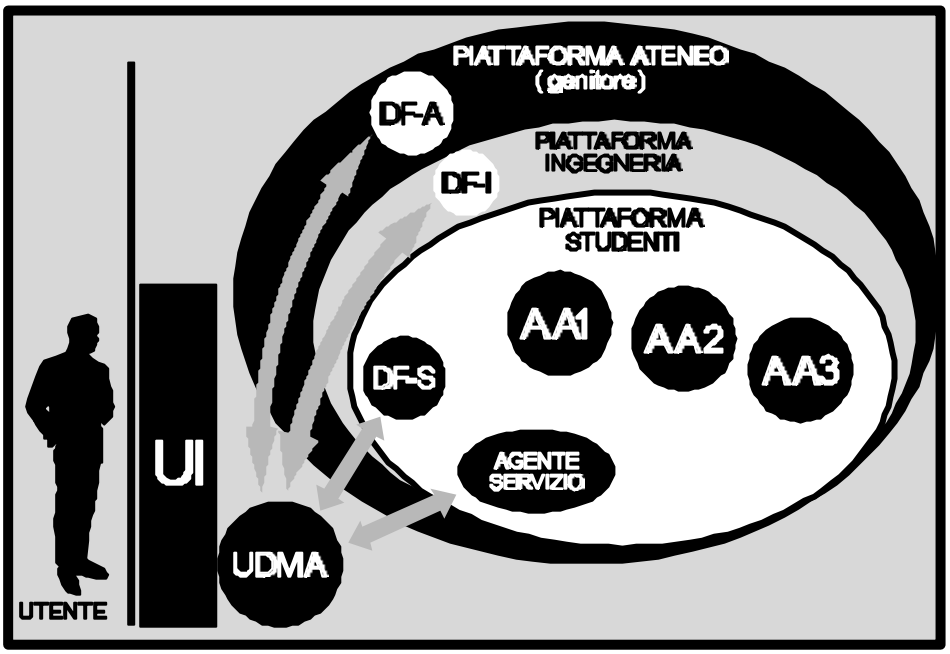

Fig. 6.8 : Modello di riferimento per l'Agente di Interfaccia UDMA.

Nel capitolo successivo risulterà evidente come tale scelta sia accettabile dal punto di vista delle specifiche FIPA.

6.5 L'Agente di Servizio

L'Agente di Servizio è stato progettato come una sorta di "amministratore di condominio" per la piattaforma ad Agenti. E' questa entità, infatti, che si fa carico di gestire tutte le richieste e comunicazioni che coinvolgono gli Agenti di Applicazione presenti sulla AP. L'UDMA contatta, sulla base delle interrogazioni dei DFs , gli Agenti di Servizio locali ed è solo con loro che può interagire.

Il nome che è stato assegnato a questo tipo di Agente non è casuale. Sta ad indicare quale è il suo ruolo nel modello della struttura. E' un Agente addetto alla fornitura di servizi specifici come, per esempio, quelli destinati agli studenti.

Il linguaggio supportato è solo quello degli Agenti (vedremo nel prossimo capitolo che si tratta di FIPA ACL), perché solo con suoi simili deve comunicare. Questa entità non fornisce alcuna risoluzione al problema giunto dall'UDMA ma funge da intermediario.

Alla luce della richiesta effettuata, decide come comportarsi: rifiutarla o accoglierla, analizzarla, verificare l'esistenza di Agenti di Applicazione in grado di assolverla ed, in caso affermativo, passar loro il testimone , restando in attesa di possibili risultati.

Il suo compito finale è quello di informare l'UDMA sulle possibili risposte trovate o se il sistema necessita di ulteriori dati per completare il lavoro che gli è stato affidato.

Anche l'Agente di Servizio, come l'UDMA, nell'eventualità che la propria piattaforma non possieda Agenti capaci di fornire la funzionalità richiesta, deve scalare la gerarchia di APs ed interrogare i rispettivi *Directory Facilitator*. In caso la ricerca si riveli infruttuosa lo comunica all'Agente di Interfaccia, che provvede ad informare l'utente che il servizio richiesto non è disponibile.

Il modello di riferimento è mostrato in figura 6.9.

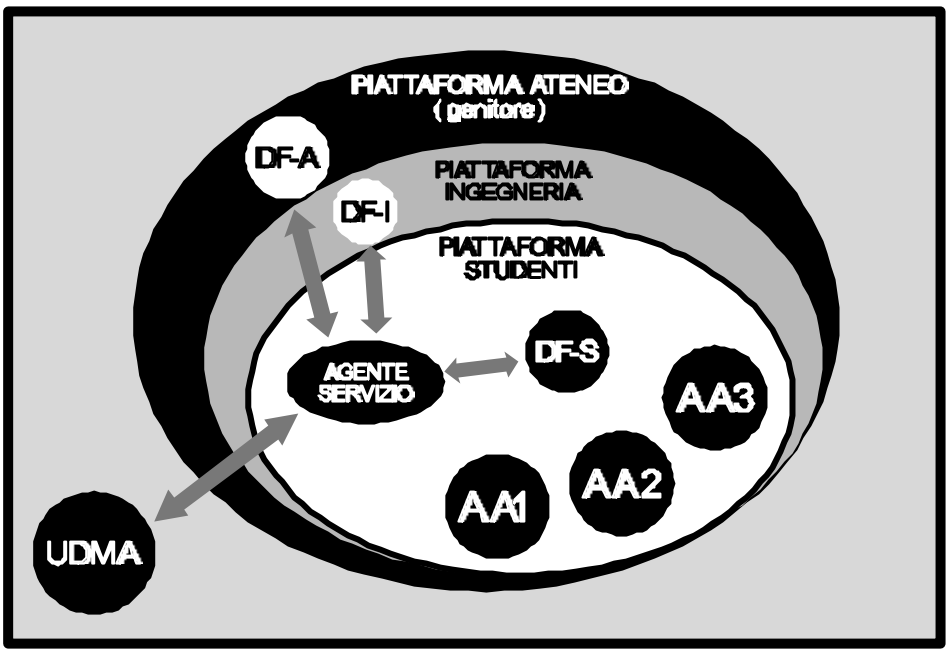

Fig. 6.9 : Modello di riferimento per l'Agente di Servizio.

Alla luce di quanto detto potrebbe sembrare che il ruolo di questo Agente non sia così determinante per il sistema e che le sue funzionalità possano essere integrate negli Agenti di Applicazione. In realtà va considerato un aspetto importante, relativo agli obiettivi finali di questo progetto. Se si considera, infatti, la realizzazione di un sistema ad Agenti completo, collocato in un ambiente dinamico e distribuito, composto da diverse piattaforme e diversi Host, risultano evidenti alcuni aspetti:

- Sulla AP dove si registra l'Agente UDMA possono non esistere Agenti preposti al servizio richiesto. Tali Agenti possono essere registrati presso un'altra AP ma, se l'UDMA non sa dove cercare, il servizio non viene fornito. E' compito dell'Agente di Servizio ricercare uno o più Agenti di Applicazione idonei.
- L'ambiente in cui è collocato il sistema è dinamico. Questo significa che gli Agenti vivono, muoiono, si spostano in continuazione. Gli Agenti di Applicazione destinati a fornire uno

specifico servizio possono essere più d'uno e mobili. L'UDMA non può conoscere a priori quanti e dove sono tali Agenti, né può inviare la sua richiesta a tutti gli Agenti in ascolto.

E', quindi, evidente quanto sia importante il ruolo di intermediario dell'Agente di Servizio. Cercando di generalizzare possiamo dire che ad un problema da risolvere posto dall'UDMA interviene l'Agente di Servizio, cercando di individuare nella rete Agenti preposti alla sua risoluzione.

6.6 Gli Agenti di Applicazione

Gli Agenti di applicazione disegnati per questo lavoro sono di natura estremamente semplice. Dato che la mobilità degli Agenti non costituiva oggetto di studio, essi sono stati concepiti come statici. Uno dei possibili sviluppi futuri potrebbe consistere nel renderli mobili, a caccia di informazioni nella rete.

Il loro obiettivo consiste nell'accedere ad appositi database, tramite autenticazione, alla ricerca delle risposte alle domande formulate dall'Agente di Servizio che, a sua volta, le ha ricevute dall'UDMA. In figura 6.10 è mostrato il modello di riferimento per un Agente di questo tipo.

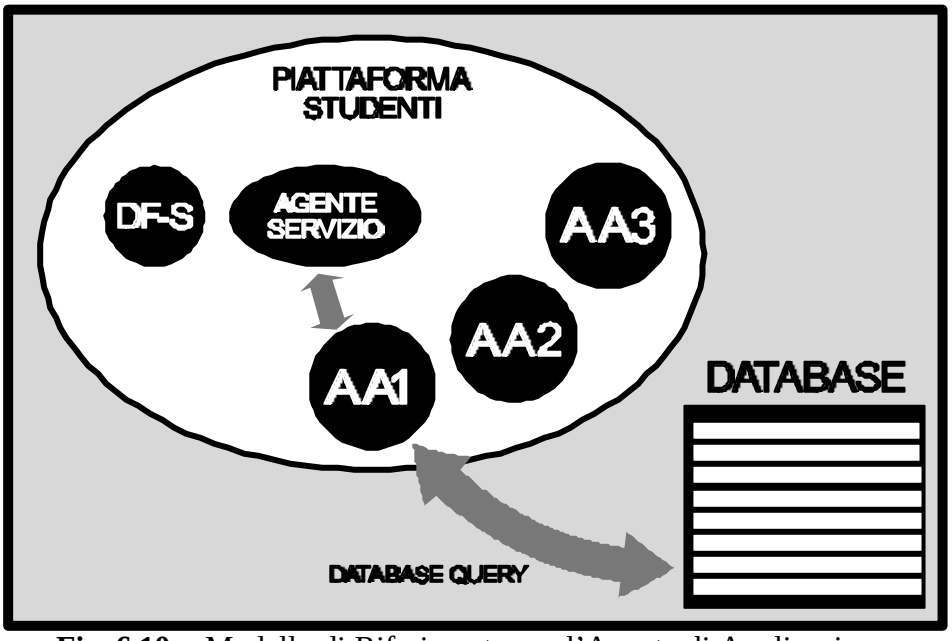

Fig. 6.10 : Modello di Riferimento per l'Agente di Applicazione.

Nel prossimo capitolo verranno analizzati tutti gli aspetti relativi all'implementazione dell'architettura e degli Agenti appena descritti.

Capitolo 7

LA REALIZZAZIONE

Dopo aver affrontato gli aspetti relativi alla progettazione del sistema *AOL-MAS*, in questo capitolo si discuterà della loro implementazione e delle relative problematiche emerse, nonché dei risultati ottenuti.

7.1 Le scelte adottate

Data la natura di questo progetto sono state studiate numerose soluzioni per l'implementazione dell'architettura multi-Agente.

Innanzitutto si è scelto di sfruttare le funzionalità dell'ambiente software JADE, versione 2.01, di cui si è ampiamente parlato nel capitolo 4. Ha permesso, infatti, di realizzare Agenti conformi allo standard FIPA ed ha, al contempo, offerto una ricca serie di strumenti utili alla gestione e manutenzione del sistema creato (quali l'Agente Sniffer, l'Agente Dummy,…).

Jade, come si è già detto, è stato sviluppato utilizzando il linguaggio Java, caratterizzato dal fatto di essere un linguaggio object-oriented di facile portabilità. Java, infatti, è un linguaggio per ambienti eterogenei, il che significa che i programmi Java possono funzionare allo stesso modo su piattaforme hardware/software differenti, includendo i sistemi operativi più diffusi quali Windows di Microsoft, Apple Macintosh, e la maggior parte delle versioni di UNIX, incluso Solaris.

I programmi Java riescono ad ottenere questa indipendenza utilizzando una *macchina virtuale* (JVM), nota anche come *interprete* di Java, che deve essere sviluppata per le diverse piattaforme o sistemi operativi. Gli

Agenti programmati in Java, quindi, possono essere utilizzati su qualunque calcolatore dotato di JVM e, nel caso specifico, delle classi contenute nelle librerie di Jade.

La versione di Java utilizzata è il *Java Development Kit* (JDK) 1.3, rilasciata da Sun Microsystem.

Le classi realizzate sono state testate su diversi sistemi operativi: Windows98, WindowsNT e Linux.

Dato che AOL-MAS ha previsto la programmazione di un'interfaccia utente strutturata come sito web, per consentire agli utenti finali di usufruire del servizio attraverso Internet, è stato indispensabile utilizzare un calcolatore appositamente configurato come server. Il computer in questione, basato sul sistema operativo *Windows NT 4 Server*, è stato equipaggiato del diffusissimo *APACHE HTTP Web Server 2.0* [APA00].

Nei prossimi paragrafi verranno descritti in dettaglio tutti questi aspetti: la creazione dell'interfaccia utente (UI), il sistema ad Agenti e la configurazione del web server.

7.2 L'interfaccia Utente

Uno degli obiettivi primari di questo lavoro di tesi ha riguardato proprio l'interfacciamento tra l'utente ed il sistema multi-Agente attraverso la rete. Data la volontà di creare un'architettura che si integrasse al meglio con le tecnologie web già consolidate è stato costruito un sito web, dal quale gli studenti della facoltà possono accedere a dei servizi sperimentali che sfruttano la tecnologia degli Agenti software. In figura 7.1 viene mostrata la Home Page di tale sito. Per una migliore portabilità, considerando che non tutti posseggono le più aggiornate versioni di Browser, si è optato per un approccio che garantisca un'ampia utilizzabilità, basato su HTML e Java. La pagina

principale è strutturata a 4 *FRAME*, cioè suddivisa in 4 quadri. In uno di essi è inserito il menù dinamico (realizzato come Applet Java) dal quale accedere alle varie pagine e servizi. Cliccando sul tasto *Servizi Studenti* si entra nell'area a loro riservata dove, tramite apposito sottomenù è possibile richiamare uno dei servizi offerti.

Attivato l'apposito *Link*, il sito web mostra una pagina contenente i moduli (*FORM*) o gli Applets, necessari alla fornitura del servizio desiderato (vedi figg. 7.2 e 7.3).

Fig. 7.1 : Home Page dell'Interfaccia Utente AOL-MAS.

L'utente inserisce i parametri richiesti, per esempio nome e cognome di un docente di cui si vuole conoscere l'indirizzo e-mail, oppure i dati

relativi ad una particolare disciplina di cui si vuole contattare un ricercatore. I dati inseriti nei moduli vengono poi spediti ad un programma CGI sul Web Server utilizzando il metodo *GET*, fornito dall'HTML.

Lo script CGI viene richiamato utilizzando gli attributi del tag <FORM>. Uno di questi è *ACTION* e contiene il nome dello script che si occupa dell'elaborazione della scheda. Nel nostro caso:

<FORM ACTION=http://crauto.ing.unibs.it/cgi-bin/udma.bat>

Fig. 7.2 : Form per l'attivazione del servizio Ricerca E-Mail.

Oltre a questo riferimento allo script, ogni campo di input della scheda (un campo di testo, un pulsante di opzioni,…) ha un corrispondente attributo *NAME* che lo identifica univocamente. Quando si inoltra la scheda, allo script CGI indicato in *ACTION* vengono passati i nomi dei campi ed i relativi valori codificati in coppie *nome=valore*. All'interno dello script è possibile determinare il contenuto di ciascun campo (il valore) facendo riferimento al nome.

L'attributo *METHOD* indica, invece, il modo in cui i dati della scheda vengono inviati dal browser al server e, quindi, allo script. *METHOD* accetta due valori: *GET* o *POST*. Per questo progetto è stato scelto il metodo *GET* perché i dati della scheda vengono raccolti ed aggiunti alla fine dell'indirizzo URL specificato nell'attributo *ACTION*. Dunque se tale attributo ha il seguente aspetto:

ACTION=http://crauto.ing.unibs.it/cgi-bin/udma.bat

e la scheda utilizza, per esempio, tre campi di input, tipo di servizio, nome e cognome di un docente, l'indirizzo URL inoltrato è del tipo:

http://crauto.ing.unibs.it/cgi-bin/↵
udma.bat?tiposervizio=ricercaemail&nome=mario&cognome=rossi

Quando il server esegue lo script CGI che si occupa dell'elaborazione dei dati della scheda, assegna alla variabile d'ambiente *QUERY_STRING* tutto ciò che nell'indirizzo URL segue il punto interrogativo. E' compito dello script CGI effettuare il parsing di tale variabile, identificando i vari campi.

Fig. 7.3 : Form per l'attivazione del servizio "A chi rivolgersi".

Nel capitolo 5 si è introdotta la tecnologia CGI e si è detto che uno script di questo tipo può essere scritto in qualsiasi linguaggio di programmazione. Dato che il nostro obiettivo è quello di lanciare un Agente di interfaccia UDMA (scritto in Java) sul server e di passargli i parametri forniti dall'utente, anziché servirsi di uno script scritto in *PERL*

(come usualmente si fa nelle applicazioni commerciali) si è fatto ricorso ad uno file di tipo batch.

Questa scelta è stata dettata da numerosi fattori, tra i quali il sistema operativo su cui è stata effettuata la sperimentazione (WindowsNT) ma, soprattutto, dalla necessità di lanciare da remoto una applicazione java.

Lo script utilizzato è il seguente:

```
@echo off
@echo content type: text/html
@echo.
@"c:\program files\javasoft\jre\1.3\bin\java.exe" ¿
   -cp c:\jade\lib\jade.jar; c:\jade\lib\Tools.jar; c:\jade\lib\Base64.jar; ¿
   c:\ProgettoAteneo06\Agenti Udma "%QUERY_STRING%"
```

La prima riga serve a disabilitare il video come dispositivo standard per l'output, reindirizzandolo così verso la rete. La seconda riga determina il valore dell'intestazione Content-Type. Poiché i risultati che verranno inviati al Client sono pagine HTML, il suo valore è: text/html. In seguito viene richiamato l'interprete Java, a cui si passa il nome dall'applicazione da eseguire ed i parametri, contenuti nella variabile d'ambiente *QUERY_STRING*.

Un altro aspetto va tenuto in considerazione: per poter scrivere ed eseguire uno script CGI è assolutamente necessario avere accesso ad un Web Server. A differenza dei comuni file HTML, infatti, non è possibile eseguire gli scripts sul sistema locale. Come è stato accennato nel primo paragrafo, per questo progetto si è utilizzato l'*HTTP Web Server APACHE*, di cui si parlerà più avanti.

Va però notato che, anche se si ha un Web Server, tale server deve essere configurato in modo particolare per l'esecuzione degli scripts CGI.

Normalmente questo significa che tali scripts devono essere inseriti in una speciale cartella chiamata *cgi-bin*. Nel prossimo paragrafo verranno affrontati gli aspetti relativi alla configurazione del Web Server.

7.3 Il Web Server APACHE

Il Web Server Apache, nato nel 1995, è il frutto della collaborazione di numerosi programmatori, sparsi per il mondo. Il progetto che ha dato il nome al software, infatti, si è sviluppato allo scopo di realizzare un HTTP Web Server rubusto, affidabile e ideale per applicazioni commerciali. Sia il software che i codici sorgente sono disponibili gratuitamente, permettendo così una rapida evoluzione tecnologica del prodotto. In poco tempo, Apache si è affermato come uno dei più efficienti Web Server utilizzati dai fornitori di servizi Internet.

Le caratteristiche che fanno di APACHE un supporto fondamentale per lo sviluppo del progetto AOL-MAS sono [APA00]:

- E' un Web Server conforme al protocollo HTTP/1.1 (RFC2616).
- E' configurabile facilmente ed è strutturato per permettere l'integrazione con moduli aggiuntivi.
- Funziona con sistemi operativi eterogenei: WindowsNT/9x, Netware 5.x, OS/2, Unix,…
- E' in grado di riconoscere oltre 500 tipologie di errori, soprattutto nell'esecuzione degli script CGI.
- Supporta diverse tecniche di URL Aliasing e Virtual Host.

Nel prossimo paragrafo viene descritto come configurare Apache per eseguire script CGI.

7.3.1 La configurazione APACHE per CGI

Affinchè gli scripts CGI vengano eseguiti correttamente, il Web Server deve essere configurato in maniera opportuna. Ci sono diverse possibilità:

ScriptAlias : la direttiva *ScriptAlias* segnala ad *APACHE* che una determinata directory del server è riservata all'esecuzione di programmi CGI.

APACHE considera CGI tutti i programmi residenti in questa cartella e li esegue quando richiamati da un Client. La direttiva *ScriptAlias*, che si trova nel file *httpd.conf*, è del tipo:

ScriptAlias /cgi-bin/ /usr/local/apache/cgi-bin/

Questa direttiva si aggiunge a quella chiamata *Alias*, che definisce un prefisso URL associato ad una determinata directory. *Alias* e *ScriptAlias* sono di solito usate per cartelle esterne a quella di root, denominata *DocumentRoot.*

La differenza tra le due consiste, essenzialmente, nel "valore aggiunto" di *ScriptAlias*, perché indica che tutti i files preceduti dall'URL definito vanno trattati come CGI. Questo significa che una richiesta pervenuta da un client ed indirizzata ad un programma nella cartella *cgi-bin/*, viene reindirizzata alla cartella */usr/local/apache/cgi-bin/.* In questa trattazione la chiamata effettuata dal form HTML col metodo ACTION:

http://crauto.ing.unibs.it/cgi-bin/Udma.bat

diviene:

http://crauto.ing.unibs.it/usr/local/apache/cgi-bin/Udma.bat

CGI esterni alla cartella ScriptAlias : un'aspetto da tenere in considerazione riguarda la possibilità di posizionare i programmi CGI in cartelle differenti, per esempio perché si è deciso di non consentire un accesso indiscriminato alla cartella cgi-bin principale. Per fare ciò si sfrutta un particolare tipo di files, denominati **.htaccess.* Quando viene richiesta l'esecuzione di uno script CGI, Apache cerca, nella cartella a cui appartiene lo script, l'esistenza di un file *.htaccess ed applica le direttive in esso contenute. E' la direttiva *AllowOverride* che configura i permessi e l'accesso alle risorse. Per l'esecuzione di file CGI è, pertanto, indispensabile che i files *.htaccess contengano la direttiva seguente:

AllowOverride Options +ExecCGI

segnalando al Web Server che l'esecuzione di Script CGI è permessa nella directory in questione.

Uso della direttiva Options : nel Main Server Configuration File è possibile esplicitare direttamente la possibilità di eseguire programmi CGI in una determinata cartella, tramite la direttiva *Options:*

```
<Directory /usr/local/apache/htdocs/cartella>
    Options +ExecCGI
</Directory>
```

E' interessante notare che la direttiva *ScriptAlias*, descritta in precedenza, è una combinazione della direttiva *Alias* e di quella *Options.*

In conclusione a questa breve trattazione del Web Server è bene ricordare la necessità di una buona configurazione dei permessi e degli accessi alle risorse, nonché di corretti riferimenti a files e cartelle (*PATH*). Spesso, nel corso della realizzazione del sistema AOL-MAS, oggetto di questa Tesi, i problemi riscontrati sono stati legati ad errori nella configurazione del Web Server.

7.4 Le piattaforme ad Agenti (APs)

Per una sperimentazione del sistema completo sono state attivate quattro piattaforme Jade. Si tratta della AP genitore *Ateneo*, il figlio *Facoltà di Ingegneria* ed i nipoti *Servizi Studenti* e *Servizi Docenti.* Alcuni tests sono stati effettuati su un unico calcolatore, posizionando le APs su porte TCP differenti.

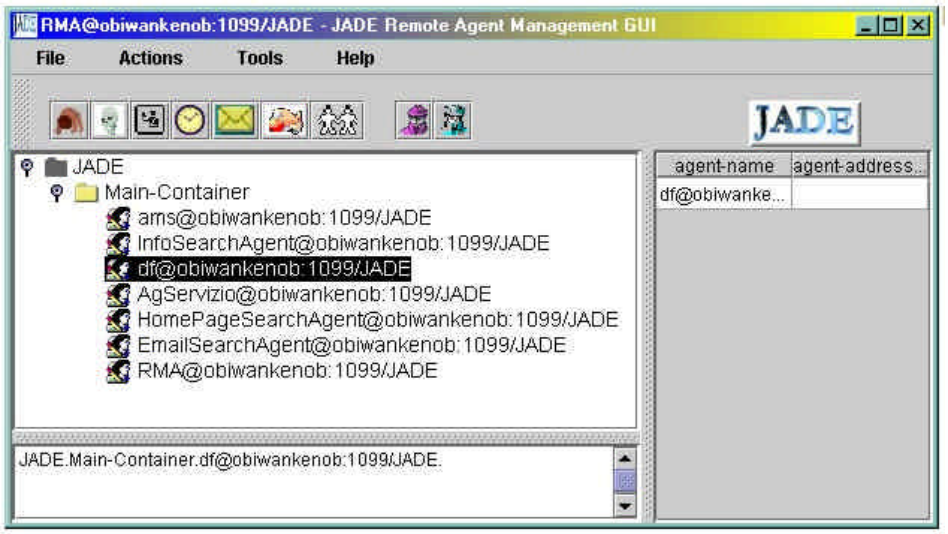

Fig. 7.4 : L'interfaccia grafica Jade dell'Agente RMA.

Tutte le piattaforme possono essere gestite attraverso l'interfaccia grafica fornita dall'Agente Jade chiamato RMA, come raffigurato in fig. 7.4, nel caso della AP_Studenti, dove sono registrati, tra l'altro, l'Agente di Servizio e gli Agenti di Applicazione.

7.4.1 DF Federation

Nonostante l'Agente preposto all'interfacciamento Utente-Agenti contenga già, nel suo codice, informazioni sulla gerarchia delle piattaforme, una tecnica più efficace per l'amministrazione del sistema ad Agenti consiste nella *DF Federation*. Si tratta semplicemente di federare, cioè di associare ad ogni DF di una AP, il DF della AP figlio (come *SubDF*) ed il DF della AP genitore (come SuperDF). Questo significa, per esempio, che il Directory Facilitator della piattaforma Facoltà di Ingegneria ha per figlio il DF_Studenti e per genitore il DF_Ateneo. Questa situazione è mostrata in figura 7.5 dove le APs sono state attivate sullo stesso Host (*Obiwankenob*) ma su porte TCP differenti: 1099 per la AP_Ingegneria, 1999 per la AP_Ateneo e 1299 per la AP_Studenti.

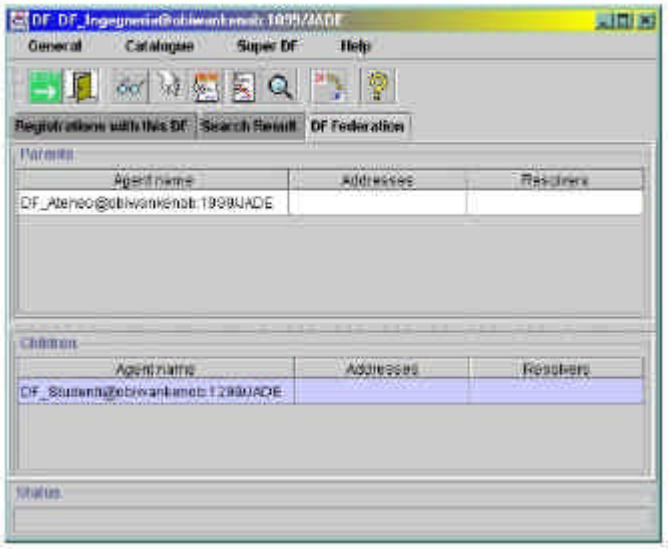

Fig. 7.5 : Interfaccia Jade per la DF Federation.

Questo procedimento risulta particolarmente utile durante le ricerche di un Agente presso il DF di una AP. Infatti, in caso di insuccesso, cioè se l'Agente in questione non si trova registrato sulla piattaforma considerata, la ricerca può essere estesa automaticamente a tutti i DFs

figli e genitori, risparmiando al programmatore riferimenti manuali ad AP che, nella peggiore delle ipotesi, potrebbero non essere attive. Nel caso di una gerarchia a molti livelli la ricerca può proseguire in base ai rapporti di "parentela" tra i DFs.

7.5 Udma e UdmaAgent

L'applicazione CGI attivata dal file batch *udma.bat* (descritto nel paragrafo 7.2) è *Udma.class*. Si tratta di un programma Java che nel suo codice racchiude la classe *UdmaAgent*. L'agente di interfaccia *User Dialog Management Agent*, definito nel capitolo precedente, è un'istanza di tale classe.

L'applicazione crea un contenitore d'agenti presso la AP Studenti sulla porta 1099, dopodichè costruisce un Agente UdmaAgent e lo attiva (tramite il metodo *doStart()* di Jade), passandogli i parametri di servizio provenienti dal Client (in figura 7.6 viene mostrato come cambia il pannello di controllo Jade).

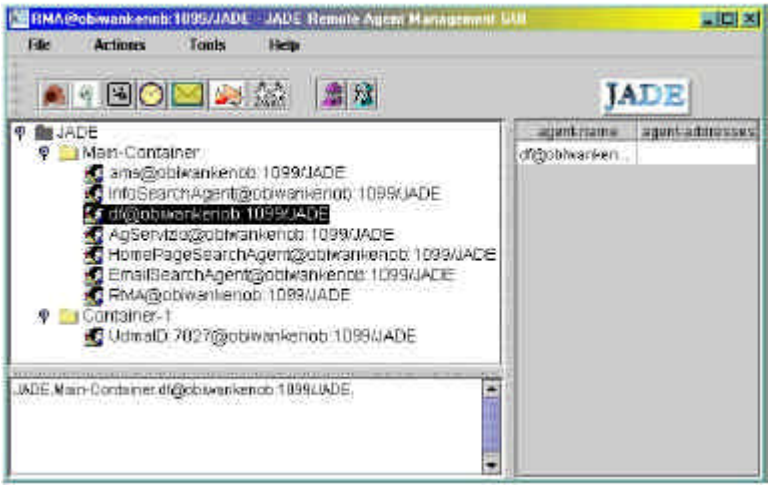

Fig. 7.6 : L'RMA segnala la creazione di un nuovo container e dell'Agente di interfaccia Udma.

La classe UdmaAgent, come tutte quelle che definiscono gli Agenti di questo progetto, sono estensioni della classe *Agent*, fornita da Jade.

E' bene sottolineare che ogni Client viene gestito dal proprio Udma personale, distinguibile dagli altri grazie all'AID.

Di seguito vengono riassunti i Behaviours di tale Agente:

- ***Registrazione***: come prima cosa l'Agente Udma si registra presso il Directory Facilitator della AP Studenti, così da rendere note al sistema la propria esistenza e le proprie funzionalità.
- ***Parsing***: l'Agente analizza il contenuto delle informazioni pervenute dal Client. Si tratta di una stringa di caratteri codificata URL, cioè: *parametro1=valore1&parametro2=valore2&….* Lo scopo di questo parsing è verificare la tipologia di servizio richiesto.
- ***Ricerca AgServizio***: per poter richiedere qualsiasi cosa al sistema ad Agenti, l'Udma deve ricercare presso il DF della AP corrente o presso i DFs federati (figli e genitori) l'esistenza di un'Agente di servizio con cui intefacciarsi.
- ***Richiesta***: questo comportamento implementa il protocollo FipaRequestInitiator per la conversazione tra Agenti. Una volta individuato un Agente di servizio, infatti, l'Udma assume il ruolo di promotore in un dialogo che ha lo scopo di richiedere la fornitura di un servizio specifico. Si possono verificare diversi inconvenienti durante la conversazione: il messaggio può non essere compreso, non pervenire correttamente oppure l'Agente di Servizio è troppo occupato per servire l'Udma in questione. In tutti questi casi il protocollo prevede che il dialogo prosegua, finchè l'Udma è certo che l'Agente di Servizio ha ricevuto correttamente la richiesta.

- ***Attesa***: se l'Agente di servizio accetta la richiesta, l'Udma sospende la propria esecuzione e resta in attesa dei risultati. In questo caso implementa il ruolo di *Receiver*.
- ***Mostra risultati***: questo behaviour consiste nella costruzione di una pagina HTML con i risultati della ricerca, ove disponibili, oppure di moduli opportuni per inserire nuovi parametri, utili alla fornitura del servizio. Tali pagine contengono, inoltre, una serie di collegamenti ipertestuali generati alla luce dei risultati prodotti (*URL*, *Mailto*,…).
- ***Suicidio***: una volta terminato il proprio compito, l'Agente Udma si deregistra dal DF e AMS della AP e si "Suicida" (*doDelete()*), distruggendo il suo contenitore. Quest'ultima operazione è indispensabile al corretto funzionamento del sistema. La JVM in cui il contenitore è stato creato, infatti, è l'applicazione CGI richiamata dal Client. Affinchè i risultati vengano mostrati all'utente, tale JVM deve essere terminata. Solo in questo modo la connessione Client/Server si chiude.

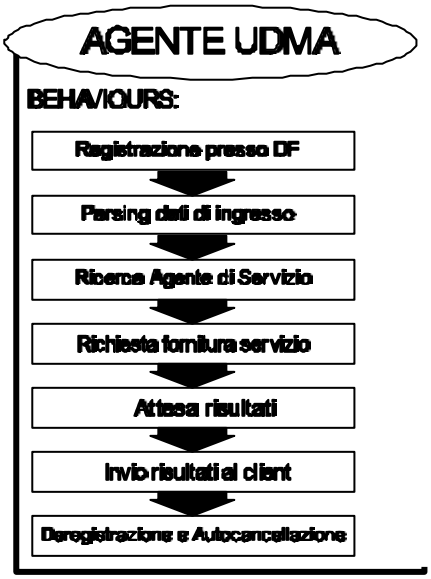

Fig. 7.7 : I Behaviours dell'Agente Udma.

7.6 L'Agente di Servizio

Contrariamente a quanto accade per l'Udma, l'Agente di Servizio, è unico per la AP. Viene creato al lancio del sistema AOL-MAS come istanza della classe *AgServizio* (riportata in appendice A) ed è stato programmato per supportare conversazioni multiple. Può cioè interagire con diversi Agenti concorrentemente, grazie alla creazione di differenti *Thead*, uno per ogni conversazione. Per fare cio' tale Agente incorpora i seguenti Behaviours:

- ***Registrazione***: come per l'Udma, anche l'Agente di Servizio si registra presso il Directory Facilitator della AP Studenti, così da rendere note al sistema la propria esistenza e le proprie funzionalità.
- ***Responder***: questo Behaviour implementa il protocollo FipaRequestResponder, in cui l'Agente risponde alle richieste di altri agenti. Sono previsti il rifiuto, l'accettazione e la richiesta di ripetizione dei messaggi pervenuti, secondo quanto prevede il FipaRequest Protocol descritto nel capitolo 2.
- ***Parsing***: se l'Agente di Servizio accetta la richiesta formulata dall'Udma, analizza la stringa di caratteri ASCII inviatagli, per individuare il tipo di Servizio richiesto.
- ***Ricerca Agente di Applicazione***: dopo l'operazione di parsing, l'Agente ricerca presso il DF della AP corrente o presso i DFs federati (figli e genitori) l'esistenza di un'Agente di Applicazione specifico per il servizio richiesto. In caso negativo l'Agente avverte l'Udma, tramite messaggio, che il servizio non è disponibile. L'identificazione dell'Agente di Applicazione avviene sfruttando il *DFServiceDescriptor* implementato da Jade in base alle specifiche FIPA. Si tratta di un servizio offerto da tutti i DF: ogni Agente,

oltre al proprio AID può essere caratterizzato da una serie di parametri che lo descrivono. Uno di questi è il tipo (*Type*), che qualifica tutti gli Agenti del sistema. Per esempio: *EmailSearchAgent* è il Type dell'Agente di Applicazione addetto alla ricerca di indirizzi e-mail.

- ***Richiesta***: come per l'Udma, questo comportamento prevede l'implementazione del protocollo FipaRequestInitiator. Nel caso particolare in cui l'Agente di Applicazione richiesto si trovi sulla stessa AP Jade, i parametri di servizio vengono inseriti in un record e codificati in *Base64* (tecnica di codifica Jade).
- ***Attesa***: se l'Agente di Applicazione contattato accetta la richiesta, l'Agente di servizio sospende la propria esecuzione e resta in attesa dei risultati. In questo caso implementa il ruolo di *Receiver*.
- ***Invio Risultati***: i risultati provenienti dall'Agente di Applicazione vengono codificati come stringa di caratteri ASCII ed inviati all'Udma. Il ruolo implementato è quello di *Sender*.
- ***Reset***: l'ultima operazione consiste nell'autoreset dell'Agente di servizio, che torna al behaviour *Responder* , in attesa che un altro Udma si connetta.

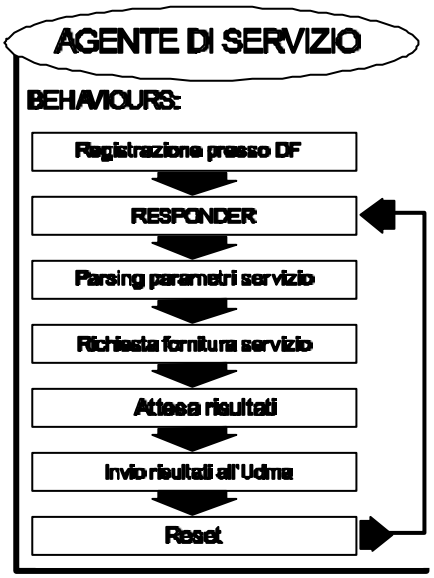

Fig 7.8 : I Behaviours dell'Agente di servizio.

7.7 Gli Agenti di Applicazione

Per la sperimentazione del sistema AOL-MAS sono stati realizzati tre semplici Agenti di Applicazione statici:

- ***EmailSearchAgent***: cerca indirizzi email alla luce di nome e cognome del docente inseriti nel form del sito Internet.
- ***HomePageSearchAgent***: simile al precedente, cerca gli URLs delle HomePages dei docenti universitari.
- ***InfoSearchAgent***: questo Agente ricerca tutte le informzioni disponibili sul personale universitario che si occupa di una particolare disciplina o area tecnica.

Questi tre agenti accedono in via esclusiva (tramite userid e password) ad appositi DataBase dimostrativi creati con Microsoft Access97 per effettuare delle prove. Il meccanismo di ricerca, basato su JDBC e SQL query è descritto nel paragrafo 7.7.1.

I Behaviours implementati sono i seguenti:

- ***Registrazione***: come per l'Udma e l'Agente di Servizio, anche gli Agenti di Applicazione si registrano presso il Directory Facilitator della AP a cui sono assegnati (per esempio AP_Studenti o AP_Docenti), così da rendere note al sistema la propria esistenza e le proprie funzionalità.
- ***Responder***: questo Behaviour implementa il protocollo FipaRequestResponder. L'Agente di Applicazione resta in attesa che l'Agente di Servizio si connetta e formuli la richiesta.
- ***Ricerca DataBase***: in base ai parametri di servizio ricevuti, l'Agente apre un DataBase ODBC (nel caso specifico files

Microsoft Access) e cerca le informazioni richieste sfruttando gli Statements SQL.

- ***Invio Risultati***: i risultati prodotti, ove disponibili, vengono inviati all'Agente di Servizio.
- ***Reset***: l'ultima operazione consiste nell'autoreset dell'Agente di Applicazione, che torna al behaviour *Responder* , in attesa che un altro Agente si connetta.

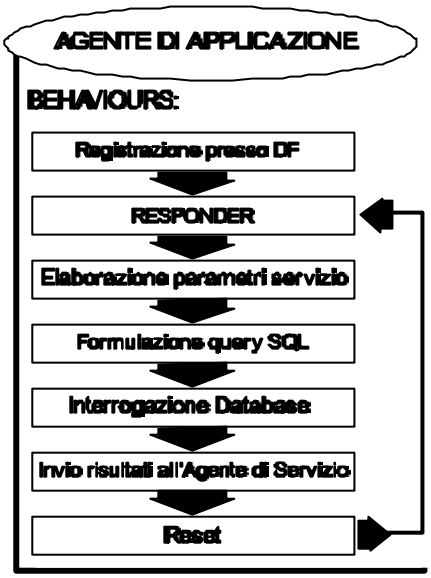

Fig. 7.9 : I Behaviours di un'Agente di Applicazione

7.7.1 JDBC ed SQL

Il supporto incorporato in Java per l'accesso ai database relazionali, chiamato JDBC (*Java Database Connectivity*), è strutturato in modo simile allo standard ODBC (*Open Database Connectivity*) [SCI97].

L'ODBC è un gruppo di API scritte in linguaggio C che permette di scrivere codice in grado di accedere a database di diversi fornitori. E' derivato da uno standard noto come *X-Open SQL*, nato nel mondo Unix, e definito da un consorzio di produttori. A causa dell'uso intensivo di

variabili puntatore, ODBC non è adatto ad essere utilizzato direttamente come API per Java.

Il JDBC definisce un'interfaccia strutturata per accedere ai database SQL (*Structured Query Language*) che è poi l'approccio standard, nel settore, per l'accesso ai database relazionali. Supportando SQL, JDBC consente agli sviluppatori di sfruttare un vasto numero di database esistenti. Ciò significa che le caratteristiche delle piattaforma di origine di un database sono irrilevanti quando questo viene utilizzato in JDBC.

Lo standard SQL, molto diffuso e che prevede un protocollo ben definito per l'accesso ai dati e la loro manipolazione. Dato che SQL non costituisce oggetto di questo lavoro di tesi ci si limita a fornire le indicazioni principali relative alla programmazione JDBC, utilizzate per implementare gli Agenti di Applicazione:

1. ***Caricare il Driver JDBC che si intende utilizzare***. Questa operazione, effettuata tramite la classe Java *DriverManager*, risulta superflua se si è provveduto a specificare nelle proprietà di sistema i driver necessari per l'accesso ai database ODBC.
2. ***Eseguire la connessione al database***. Per eseguire la connessione si usa il metodo statico *getConnection* della classe *DriverManager*.
3. ***Sottoporre le query SQL che si vogliono eseguire***. Il JDBC utilizza istruzioni SQL per accedere ai dati. Si usa la classe *Statement*, i cui metodi accettano qualunque oggetto *String* per indicare i parametri di ricerca e le operazioni da effettuare sul database.
4. ***Reperire le informazioni restituite dalle query SQL***. Si utilizza l'istruzione SQL SELECT utilizzando il metodo *executeQuery*. Si ottiene come risultato un oggetto *ResultSet* contenente le informazioni che soddisfano i criteri specificati.

5. ***Gestire tutti gli eventuali errori conseguenti alle query SQL***. Controllare la presenza di eccezioni eseguendo chiamate a metodi che inviano eccezioni all'interno di blocchi *try/catch*.
6. ***Chiudere la connessione con il database***. Le connessioni a database, una volta create, devono essere terminate.

Nel prossimo paragrafo vengono riepilogati i risultati ottenuti durante il collaudo del sistema AOL-MAS.

7.8 Sperimentazione del Sistema

Il sistema multi-Agente realizzato corrisponde in maniera soddisfacente agli obiettivi prefissati in fase progettuale. Tutti gli elementi che lo compongono, gli Agenti, il sito web ed il relativo Web Server funzionano correttamente, salvo casi sporadici in cui si sono verificate situazioni di stallo nell'esecuzione della Piattaforma Jade.

Durante la sperimentazione, infatti, si sono registrati casi in cui l'Agente di Interfaccia Udma non si suicida perché la AP non glielo consente. Questo aspetto, secondo gli sviluppatori di Jade, verrà risolto nelle prossime releases, dove sono previsti metodi di *AgentKilling* e *ContainerKilling* più affidabili. E' desiderabile, infatti, eliminare del tutto Agenti non più utili. Se l'Udma, una volta terminato il proprio compito, non si autocancella, non solo l'utente non riceve i risultati della ricerca, ma resta attiva una JVM inutile. Le JVM residue appesantiscono il server, rischiando una condizione di stallo.

Perciò AOL-MAS è stato programmato per funzionare continuamente, anche in presenza di tali situazioni, dove provvede l'applicazione Java-CGI a forzare la terminazione della JVM.

Nel paragrafo 7.4 si è parlato dell'utilità dell'interfaccia grafica fornita da Jade per la DF Federation e per la gestione ed il controllo degli Agenti della Piattaforma. Essa rappresenta, infatti, un valido supporto per l'amministrazione e manutenzione del sistema multi-Agente.

Un'ultima caratteristica che si è rivelata interessante per il monitoraggio dell'intero sistema è l'Agente Sniffer, il cui obiettivo è tenere traccia della cronologia di tutte le comunicazioni tra Agenti.

In figura 7.10 viene riportato un esempio di tale Agente, quando il sistema ad Agenti sta espletando la fornitura del servizio Ricerca Indirizzo E-Mail.

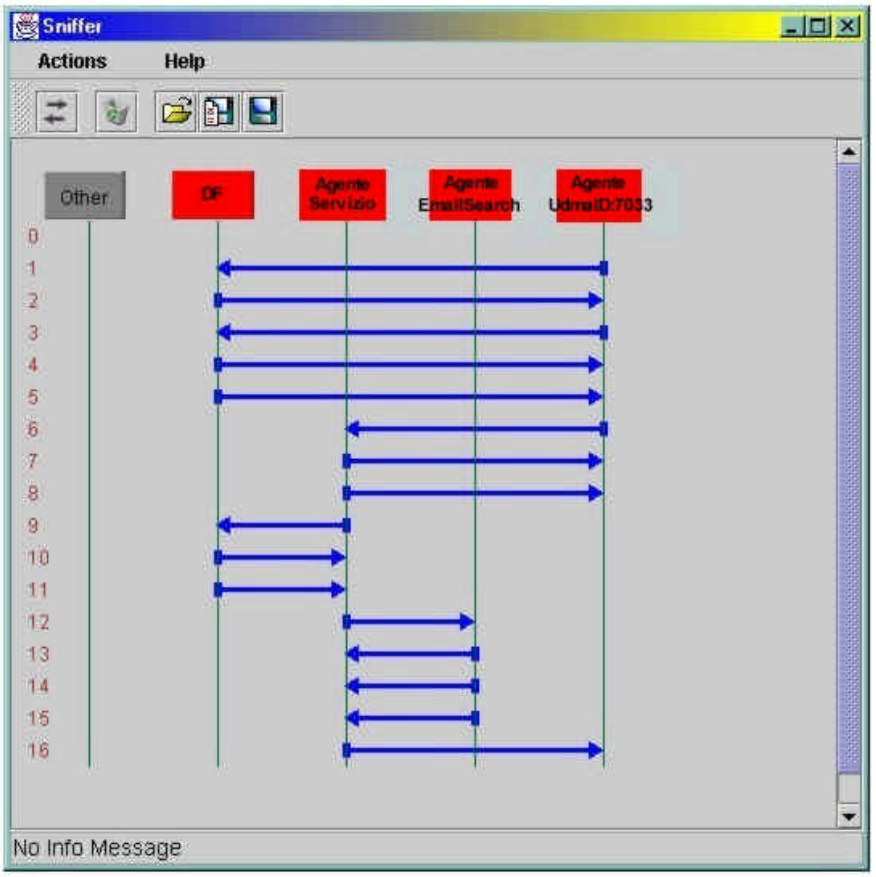

Fig. 7.10 : L'Agente Sniffer tiene traccia di tutte le comunicazioni tra Agenti nel caso della richiesta servizio *Ricerca Indirizzo E-Mail.*

Capitolo 8

CONCLUSIONI

8.1 I risultati ottenuti

Il presente lavoro di Tesi ha portato alla definizione ed alla realizzazione di un sistema multi-Agente completo ed operativo volto alla fornitura di servizi in rete. Tale sistema è basato sulle specifiche FIPA2000 ed è stato implementato servendosi del linguaggio Java e della libreria software Jade.

Sono stati programmati l'agente addetto al dialogo utente, l'agente di servizio della piattaforma ed una serie di semplici agenti di applicazione, destinati alla fornitura di servizi specifici.

Si sono affrontati studi inerenti la comunicazione degli Agenti e le tecniche di interfacciamento client-server, garantendo una integrazione adeguata con le tecnologie classiche per la fornitura di servizi web.

A tale scopo è stato implementato un sito web per l'accesso a tali servizi, programmato in HTML ed applet Java.

I risultati della sperimentazione, nonostante alcuni casi isolati, accennati nel capitolo precedente, si sono rivelati favorevoli al tipo di approccio proposto.

8.2 Sviluppi Futuri

Il sistema AOL-MAS è un prototipo da estendere e migliorare, integrandolo con le architetture ed i servizi già realizzati in altri progetti.

La struttura modulare con cui è stato concepito permettono l'aggiunta di altri servizi, tra i quali potrebbero essere programmati: un Agente

addetto all'identificazione dell'utente, impedendo l'accesso a determinati servizi ai non autorizzati.

Inoltre, un aspetto che andrebbe studiato approfonditamente riguarda la *mobilità* degli Agenti, in particolare quelli destinati ad una applicazione specifica, come la fornitura di un servizio che comporti la ricerca di informazioni. E', infatti, alla luce del modello distribuito dell'informazione che la tecnologia degli Agenti trova migliore applicazione. Appare inutile fare ricorso ad un sistema tanto complesso se l'unico fine è rappresentato dall'accesso ad un database presente sulla stessa macchina dove risiede la piattaforma ad Agenti.

Strettamente legato alla mobilità vi è, poi, l'aspetto *sicurezza*, indispensabile in ambienti distribuiti ed aperti come le piattaforme, dove gli Agenti possono registrarsi ed essere eseguiti, ma allo stesso modo provocare danni. Devono cioè essere realizzati meccanismi per garantire la salvaguardia e regolare l'accesso alle risorse del sistema.

Concludo questo mio lavoro con la speranza di aver dato un piccolo contributo al settore implementativo della tecnologia ad Agenti, consentendomi di sentirmi partecipe di un dibattito internazionale in continua evoluzione. L'aver risolto, con mia grande soddisfazione, tutti i problemi relativi al funzionamento ed alla sperimentazione del sistema ha valorizzato l'impegno e gli sforzi necessari al raggiungimento degli obiettivi prefissati.

APPENDICE A: I CODICI

In questa appendice vengono presentati i codici sorgente Java e Html relativi alle componenti software principali, realizzate per il presente lavoro di tesi:

I listati seguenti si riferiscono alla tecnica Jade-JSP discussa nel capitolo 5, relativamente alla possibilità di creare agenti in applet Java:

A1. User Dialog Management Agent

```
/* Titolo: AteneoOnLine
*  Descrizione: User Dialog Managent Agent ( UDMA )
*  Release: #01 - Gennaio 2001
*  Copyright: Universita' Degli Studi Di Brescia
* @Autore: Emiliano Vezzoli - vezzoli@evweb.it
*/

import java.io.*;
import java.util.*;
import java.util.StringTokenizer;
import java.util.Properties;
import java.lang.*;
import java.lang.Object;

import jade.Boot;
import jade.core.Agent;
import jade.core.AgentManager;
import jade.core.behaviours.OneShotBehaviour;
import jade.core.behaviours.ComplexBehaviour;
import jade.core.*;
import jade.core.behaviours.*;
import jade.core.AID;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.domain.FIPAAgentManagement.*;
import jade.domain.DFServiceCommunicator;
import jade.domain.AMSServiceCommunicator;
import jade.domain.FIPAException;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.JADEAgentManagement.AgentDead;
import jade.proto.FipaRequestInitiatorBehaviour;

/* Questo agente si occupa dell'interfacciamento tra
*  utente e sistema ad agenti sul server, convertendo
*  la richiesta dell'utente pervenuta al server nel
*  linguaggio ACL ed richiedendo il servizio all'agente
*  di servizio. L'UDMA implementa tutti i protocolli FIPA
*  per la conversazione con altri agenti.
*/

class UdmaAgent extends Agent{

  private String data, RisultatoRicerca;

  public UdmaAgent(String InputData){
  data=InputData;
  }

  public int convCounter = 0;                    // identifica il numero di conversazioni
  public AID responder = new AID();              // identifica l'agente di servizio
  public int NextBehaviour=0, NumServizio;       // variabili di controllo
```

```
 public Agent AgenteUdma=new Agent();                    // identifica l'agente corrente

// Questa classe permette di identificare piu' agenti impegnati nella conversazione
private String newConvID(){
  String out = new String(getLocalName() + (new Integer(convCounter).toString()));
  convCounter++;
  return out;
  }

// Questa classe stampa un messaggio di errore verificatori al Behaviour X
// Notare che e' stato utilizzato il dispositivo standard ERR e non quello
// di OUTPUT, perche' quest'ultimo reindirizzato verso il client.
void StampaErrore() {
 System.err.println(getLocalName() + ": Errore!! al Behaviour numero: " + ¿
   NextBehaviour);
}

// Queste classi servono alla costruzione delle pagine HTML di ritorno
void IntestazioneHTML() {

System.out.println("<html><head><meta http-equiv=\"Content-Type\"   ¿
    content=\"text/html; charset=iso-8859-1\"></head>");
System.out.println("<body bgcolor=\"#000000\">");
System.out.println("<div align=\"center\"><center>");
System.out.println("<table border=\"0\" cellpadding=\"0\" cellspacing=\"3\"  ¿
    width=\"600\">");
System.out.println("<tr><td><imgsrc=\"http://crauto.ing.unibs.it/images   ¿
    /lineabianca.jpg\" width=\"600\" height=\"10\" alt=\"lineabianca.jpg (2541  ¿
    byte)\"></td> </tr>");

 if (NumServizio==1) {

 System.out.println("<tr><td bgcolor=\"#FFFFFF\"><p align=\"center\"><img ¿
  src=\"http://crauto.ing.unibs.it/images/titolo1.jpg\" width=\"400\" height=\"70\" ¿
  alt=\"titolo1.jpg (10356 byte)\"></td></tr>");

 } else if (NumServizio==2) {

 System.out.println("<tr><td bgcolor=\"#FFFFFF\"><p align=\"center\"><img  ¿
   src=\"http://crauto.ing.unibs.it/images/titolo2.jpg\" width=\"400\" height=\"70\"  ¿
   alt=\"titolo1.jpg (10356 byte)\"></td></tr>");

 } else if (NumServizio==3) {

 System.out.println("<tr><td bgcolor=\"#FFFFFF\"><p align=\"center\"><img  ¿
   src=\"http://crauto.ing.unibs.it/images/titolo3.jpg\" width=\"400\" height=\"70\"  ¿
   alt=\"titolo1.jpg (10356 byte)\"></td></tr>");

    }

System.out.println("<tr><td><img src=\"http://crauto.ing.unibs.it/images   ¿
    /lineabianca.jpg\"  width=\"600\" height=\"10\" alt=\"lineabianca.jpg (2541 ¿
```

```
   byte)\"></td></tr>");
System.out.println("<tr><td><font color=\"#FF0000\" size=\"4\" ¿
  face=\"Arial\"><strong>   </strong></font></td></tr>");
  }

// Generazione pagina HTML di cortesia, nel caso il sistema ad agente non riesca ad
// espletare il servizio richiesto
void PaginaNessunRisultato() {

System.out.println("<tr><td><p align=\"center\"> <font color=\"#FF0000\" ¿
   size=\"4\" face=\"Arial\"><strong>IL  SISTEMA  MULTI- ¿
   AGENTE  NON HA TROVATO</strong></font></td></tr>");
System.out.println("<tr><td><p align=\"center\"> <font color=\"#FF0000\" ¿
   size=\"4\" face=\"Arial\"><strong>ALCUNA INFORMAZIONE ¿
   RELATIVAMENTE</strong></font></td></tr>");
System.out.println("<tr><td><p align=\"center\"> <font color=\"#FF0000\" ¿
   size=\"4\" face=\"Arial\"><strong>ALLE INFORMAZIONI RICHIESTE ! ¿
   </strong></font></td></tr>");
System.out.println("<tr><td><font color=\"#FF0000\" size=\"4\" ¿
   face=\"Arial\"><strong>    </strong></font></td></tr>");
System.out.println("<tr><td><p align=\"center\">   ¿
             </td></tr>");

if (NumServizio==1) {

System.out.println("<tr><td><p align=\"center\"><a ¿
    href=\"http://crauto.ing.unibs.it/RicercaEmail.htm\" style=\"color: ¿
    rgb(255,255,255)\"><font face=\"Arial\" color=\"#FFFFFF\"><big> ¿
    <strong>RIPROVA</strong></big></font></a></td></tr>");

} else if (NumServizio==2) {

System.out.println("<tr><td><p align=\"center\"><a href=\"http://crauto.ing.unibs.it/ ¿
   RicercaHomePage.htm\" style=\"color: rgb(255,255,255)\"><font face=\"Arial\" ¿
   color=\"#FFFFFF\"><big><strong>RIPROVA ¿
   </strong></big></font></a></td></tr>");

} else if (NumServizio==3) {

System.out.println("<tr><td><p align=\"center\"><a href=\"http://crauto.ing.unibs.it/ ¿
   RicercaInfo.htm\" style=\"color: rgb(255,255,255)\"><font face=\"Arial\" ¿
   color=\"#FFFFFF\"><big><strong>RIPROVA ¿
   </strong></big></font></a></td></tr>");

 }
 System.out.println("<tr><td>     ¿
            </td></tr>");
 }

// Terminazione della creazione pagina HTML
void FineHeaderHTML() {
   System.out.println("<tr><td>     </td></tr>");
```

```
System.out.println("<tr><td><img src=\"http://crauto.ing.unibs.it/ ¿
   images/lineabianca.jpg\" width=\"600\" height=\"10\" ¿
   alt=\"lineabianca.jpg (2541 byte)\"></td></tr>");
System.out.println("<tr><td><p align=\"center\"><a href=\"home1.htm\" ¿
   style=\"color: rgb(0,255,255)\" target=\"principale\"><font ¿
    face=\"Arial\"><strong>HOME</strong></font></a> ¿
   </td></tr></table></center></div></body></html>");
}

// Questo Behaviour si occupa di verificare che i dati arrivati
// al server corrispondano alla effettiva richiesta di un servizio.

class Behaviour0 extends OneShotBehaviour {

  public String DatiRicevuti, limit="&=\n";
  private String[] vect;
  int i=0, NumeroCampi=0;

  public Behaviour0(String data){
  DatiRicevuti=data;
  }

 public String[] TokParse(String input) {
   String limit="&=\n";

   StringTokenizer tok=new StringTokenizer(input, limit);
   NumeroCampi=tok.countTokens();
   for (i=0;i<NumeroCampi;i++) {
     vect[i]=tok.nextToken();
     }
 return(vect);
 }

  public void action() {

  if (NextBehaviour==0) {
  System.err.println("\nLa stringa ricevuta al Behaviour0 e': " + DatiRicevuti);
   TokParse(DatiRicevuti);
   System.err.println("\nL'analisi della stringa ha dato i seguenti risultati: ");
      for (i=0;i<NumeroCampi;i++) {
       System.err.println("Campo N': "+i+" -> " +vect[i]);
       }

    if (vect[0].equals("servizio")) {
       if (vect[1].equals("ricercaemail")) {
       NextBehaviour=1;
       NumServizio=1;
       }
       else if (vect[1].equals("ricercahomepage")) {
       NextBehaviour=1;
       NumServizio=2;
       }
       else if (vect[1].equals("ricercainfo")) {
```

```
      NextBehaviour=1;
      NumServizio=3;
      }
      else {
          System.err.println("\nATTENZIONE! Il servizio richiesto non e' attualmente ¿
             disponibile");
          doDelete();
          }
    } else {
          System.err.println("\nERRORE! Non e' stato specificato alcun servizio");
          doDelete();
          }
   }
  else {
   StampaErrore();
   doWait(10000);
   reset();
   }
  }

}// fine Behaviour0


// Questo comportamento implementa il protocolli FIPA Request Initiator
// per la conversazione con altri agenti.
class Behaviour1 extends FipaRequestInitiatorBehaviour {

    Behaviour1(Agent a,ACLMessage msg, MessageTemplate mt) {
      super(a, msg, mt);
      System.err.println("\nAgente " + getLocalName() + " ha inviato ¿
          il seguente messaggio:\n"+msg.toString());
    }

   protected void handleNotUnderstood(ACLMessage msg){
     System.err.println("\nATTENZIONE! Il destinatario non ha capito la richiesta!");
     System.err.println(msg.toString());
     reset(waitAndGetNewMsg(msg));
   }

   protected void handleRefuse(ACLMessage msg) {
     System.err.println("\nIl destinatario rifiuta di accogliere la richiesta!");
     System.err.println(msg.toString());
     reset(waitAndGetNewMsg(msg));
   }

   protected void handleAgree(ACLMessage msg) {
     System.err.println("\n\tSuccesso! Il destinatario ha accettato la richiesta!");
     System.err.println(msg.toString());
     NextBehaviour=2;
     addBehaviour(new Behaviour2());
   }
```

```
  protected void handleFailure(ACLMessage msg) {
   System.err.println("\n Il procedimento di richiesta e' fallito a causa di:");
   System.err.println(msg.getContent());
   System.err.println(msg.toString());
   reset(waitAndGetNewMsg(msg));
  }

  protected void handleInform(ACLMessage msg) {
      System.err.println("\nIl destinatario avverte che la richiesta inoltrata e' gia' stata
accolta.");
   System.err.println(msg.toString());
  }

  private ACLMessage waitAndGetNewMsg(ACLMessage msg){
   // Attesa casuale tra 0 e 10 secondi.
   int timeout = (new Double(Math.random()*10000.0)).intValue();
   System.err.println("Agente : "+myAgent.getLocalName()+" attendera' "+ ¿
      timeout + " millisecondi prima di riinizializzare il protocollo");
   myAgent.doWait(timeout);
   System.err.println("Agente : "+myAgent.getLocalName()+" e' attivo ");
   ACLMessage reply = msg.createReply();
   reply.setContent(data);
   System.err.println("Agente : "+myAgent.getLocalName()+" ha inviato  ¿
      il seguente messaggio:");
   System.err.println(reply.toString());
   return reply;
   }
}// Fine Behaviour 1

// Questo Behaviour riassume i dati fin'ora pervenuti all'Agente di interfaccia.
class Behaviour2 extends OneShotBehaviour {

      public void action() {

      if (NextBehaviour==2) {
       System.err.println("\nAgente: " + getName() + " Secondo Behaviour "  ¿
           + NextBehaviour);
       System.err.println("\nRichiesta Effettuata con successo!!");
       System.err.println("\nMessaggio: "+ data);
       System.err.println("\nAgente:"+ getName()+" Resta in attesa dei risultati.");
       NextBehaviour=3;
       addBehaviour(new Behaviour3());
       }
      else
       StampaErrore();
       doWait(10000);
       reset();
    }
   }// Fine Behaviour 2
```

```
// L'agente di interfaccia attende un messaggio specifico di risposta
// caratterizzato dall'ontologia "SearchResult".
// Se non giunge, il Behaviour viene sospeso fino all'arrivo del prossimo messaggio.

class Behaviour3 extends OneShotBehaviour {

    // set di caratteristiche della risposta che si aspetta l'Agente di servizio
    MessageTemplate R1 =  MessageTemplate.MatchPerformative ¿
       (ACLMessage.INFORM);
    MessageTemplate R2 = MessageTemplate.MatchOntology("SearchResult");
    MessageTemplate R3 = MessageTemplate.MatchLanguage("PlainText");
    MessageTemplate R4 = MessageTemplate.and(R1,R2);
    MessageTemplate Rtot = MessageTemplate.and(R4,R3);

    private void WaitForAnswer() {

        ACLMessage msg = blockingReceive(Rtot);

      if (msg!= null){
           RisultatoRicerca=msg.getContent().toString();
           System.err.println(getLocalName() + " Ha ricevuto questa risposta alla ¿
              richiesta inoltrata precedentemente: "+ RisultatoRicerca);
           NextBehaviour=4;
           addBehaviour(new Behaviour4(RisultatoRicerca));
       }
       else
       {
       System.err.println(getLocalName()+": Ancora nessuna risposta e'  ¿
          stata ricevuta dal sistema ad Agenti...");
       block();
       }
    }

    public void action() {
      if (NextBehaviour==3) {
        WaitForAnswer();
      }
      else
        StampaErrore();
      }
    }// Fine Behaviour 3

// Questo comportamento crea una pagina html con i risultati della ricerca
class Behaviour4 extends OneShotBehaviour {

  private String RisultatoRicerca;

  public Behaviour4(String stringa){
  RisultatoRicerca=stringa;
  }

    public void action() {
      if (NextBehaviour==4) {
```

```
System.err.println(getLocalName() + " : Pronto per l'invio dei seguenti ¿
    risultati al client : "+RisultatoRicerca);

        if (NumServizio==1) {

            IntestazioneHTML();

            if(RisultatoRicerca.equals("NessunRisultato")) {
              PaginaNessunRisultato();
            }
            else {
              System.out.println("<tr><td><p align=\"center\"> <font ¿
                  color=\"#FF0000\" size=\"4\" face=\"Arial\"><strong> ¿
                   L INDIRIZZO E-MAIL DEL DOCENTE RICHIESTO E' IL ¿
                   SEGUENTE:</strong></font></td></tr>");
              System.out.println("<tr><td><font color=\"#FF0000\" size=\"4\" ¿
                   face=\"Arial\"><strong>    ¿
                   </strong></font></td></tr>");
              System.out.println("<tr><td><div align=\"center\"><center><table ¿
                   border=\"3\" cellpadding=\"4\" bordercolor=\"#FFFFFF\" ¿
                   bordercolorlight=\"#FFFFFF\" bordercolordark=\"#C0C0C0\" ¿
                   bgcolor=\"#FFFFFF\" width=\"600\">");
              System.out.println("<tr><td><p align=\"center\">  ¿
                   <a href=\"mailto:"+ RisultatoRicerca +"\" style=\"color: ¿
                   rgb(0,0,255)\"><font face=\"Arial\" color=\"#0000FF\"><big><big> ¿
                   <strong>"+RisultatoRicerca+"</strong></big></big> ¿
                   </font></a></td></tr></table></center></div></td></tr>");
            }
            FineHeaderHTML();
            }
        else if (NumServizio==2) {

            IntestazioneHTML();

            if(RisultatoRicerca.equals("NessunRisultato")) {
              PaginaNessunRisultato();
            }
            else {
            System.out.println("<tr><td><p align=\"center\"> <font ¿
                color=\"#FF0000\" size=\"4\" face=\"Arial\"><strong> ¿
                LA HOME PAGE DEL DOCENTE RICHIESTO SI TROVA AL ¿
                SEGUENTE URL:</strong></font></td></tr>");
            System.out.println("<tr><td><font color=\"#FF0000\" size=\"4\" ¿
                face=\"Arial\"><strong>    </strong> /font></td></tr>");
            System.out.println("<tr><td><div align=\"center\"><center><table ¿
                border=\"3\" cellpadding=\"4\" bordercolor=\"#FFFFFF\" ¿
                bordercolorlight=\"#FFFFFF\" bordercolordark=\"#C0C0C0\" ¿
                bgcolor=\"#FFFFFF\" width=\"600\">");
            System.out.println("<tr><td><p align=\"center\"> <a href=\""+ ¿
                RisultatoRicerca +"\" style=\"color: rgb(0,0,255)\"><font face=\"Arial\" ¿
                color=\"#0000FF\"><big><big><strong>"+RisultatoRicerca+"</strong> ¿
```

```
</big></big></font></a></td></tr></table></center></div></td></tr>");
            }
          FineHeaderHTML();
          }
       else if (NumServizio==3) {

          IntestazioneHTML();

          if(RisultatoRicerca.equals("NessunRisultato")) {
            PaginaNessunRisultato();
          }
          else {
          System.out.println("<tr><td><p align=\"center\"> <font ¿
             color=\"#FF0000\" size=\"4\" face=\"Arial\"><strong> ¿
              DOCENTI CHE SI OCCUPANO DELLA MATERIA DI ¿
             INTERESSE:</strong></font></td></tr>");
          System.out.println("<tr><td><font color=\"#FF0000\" size=\"4\" ¿
             face=\"Arial\"><strong>    ¿
             </strong></font></td></tr>");

         // I dati ricavati da questo servizio sono numerosi
         // Deve quindi essere effettuati un parsing per distinguere i vari campi e valori

          String limit="&=\n";
          String[][] MatriceDati=new String[20][10];

          StringTokenizer tok=new StringTokenizer(RisultatoRicerca, limit);
          int NumCampi=(tok.countTokens())/10;
          for (int i=0;i<NumCampi;i++) {
             for (int j=0;j<10;j++) {
                MatriceDati[i][j] = tok.nextToken();
             }
          System.out.println("<tr><td><div align=\"center\"><center><table ¿
             border=\"3\" cellpadding=\"4\" bordercolor=\"#FFFFFF\" ¿
             bordercolorlight=\"#FFFFFF\" bordercolordark=\"#C0C0C0\" ¿
             bgcolor=\"#FFFFFF\" width=\"600\">");
          System.out.println("<tr><td><p align=\"left\"><font face=\"Arial\"> ¿
              <strong><small>NOME: <font color=\"#0000FF\">"+MatriceDati[i][1]+" ¿
              </font>; COGNOME: <font color=\"#0000FF\">"+MatriceDati[i][3] ¿
              +"</font></small></strong></font></p><p align=\"left\"><font ¿
              face=\"Arial\"><strong><small>E-MAIL: <font color=\"#0000FF\">" ¿
              +MatriceDati[i][5]+"</font> HOME PAGE: <font ¿
             color=\"#0000FF\">"+MatriceDati[i][7]+"</font> TELEFONO: <font ¿
              color=\"#0000FF\">"+MatriceDati[i][9]+"</font>  ¿
              </small></strong></font></td></tr></table></center></div></td></tr>");
          System.out.println("<tr><td>     </td></tr>");
           }
          }
          FineHeaderHTML();
          }
       else {
          System.out.println("Errore di caricamento pagina..");
```

```
            }
        NextBehaviour=5;
        addBehaviour(new Behaviour5());
        }
        else
        StampaErrore();
      }
}

// Con questo Behaviour l'agente di interfaccia pone fine alla sua vita, dato
// che ha concluso i suoi compiti.
class Behaviour5 extends OneShotBehaviour {

  public void action() {
     if (NextBehaviour==5) {
     System.err.println(getLocalName() + " : Autodisattivazione in corso...");
     try {
        // L'Agente di Interfaccia cancella la propria registrazione presso il DF
        // e l'AMS della piattaforma.
         DFServiceCommunicator.deregister(AgenteUdma);
         AMSServiceCommunicator.deregister(AgenteUdma);
         } catch (FIPAException e) {
         System.out.println(getLocalName()+" : Processo di Deregistrazione ¿
         dal DF fallito a causa di: "+e.getMessage());
         AgenteUdma.doDelete();    // Eliminazione agente di interfaccia UDMA
         System.exit(0);  // Eliminazione del contenitore dell'UDMA
         }
     AgenteUdma.doDelete();    // Eliminazione agente di interfaccia UDMA
     System.exit(0);  // Eliminazione del contenitore dell'UDMA
     }
     else
     StampaErrore();
  }
}
// metodo pricipale della classe UdmaAgent
protected void setup() {

   System.err.println("\nStringa Pervenuta dal client: " + data);

   // Come prima cosa l'agente Udma si registra come tale presso
   // il DF della piattaforma ad agenti.

   DFAgentDescription dfd1 = new DFAgentDescription();
   ServiceDescription sd1 = new ServiceDescription();
   sd1.setType("UdmaAgent");
   sd1.setName(getName());
   sd1.addOntologies("AteneoOnLine");
   dfd1.setName(getAID());
   dfd1.addServices(sd1);
   try {
     DFServiceCommunicator.register(this,dfd1);
   } catch (FIPAException e) {
     System.err.println(getLocalName()+" Registrazione presso il DF fallita! ¿
```

```
      Motivo: "+e.getMessage());
  doDelete();
}
System.err.println(getLocalName()+ " si e' registrato con successo presso il DF");
AgenteUdma=this;  // Necessario per un riferimento corretto nei SubSubBehaviours

// Inizio dei comportamenti dell'agente Udma
   ComplexBehaviour mainBehaviour=new SequentialBehaviour();
// Attivazione controllo dei dati ricevuti dal client.
   mainBehaviour.addSubBehaviour(new Behaviour0(data));

// L'Udma verifica ed attende l'esistenza di un agente di servizio
// in grado di soddisfare la richiesta ricevuta.

DFAgentDescription dfd = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType("ServiceAgent");
dfd.addServices(sd);
SearchConstraints c = new SearchConstraints();
boolean EsitoRicercaAgServizio=false;

try {
  while (true) {
    System.err.println(getLocalName()+ " sta cercando l'esistenza ¿
      di un Agente di Servizio");
    // Il primo DF interrogato alla ricerca di un AgServizio è quello
    // della piattaforma Studenti, dove l'UDMA si aspetta di trovarlo.
        List result = DFServiceCommunicator.search(this,getDefaultDF(),dfd,c);
            if (result.size() > 0) {
              dfd = (DFAgentDescription)result.get(0);
              responder = dfd.getName();
     EsitoRicercaAgServizio = true;
     break;
          }
    else if (EsitoRicercaAgServizio==false) {
    // Inizio a risalire la gerarchia di APs e l'UDMA consulta il
    // il DF della AP Ingegneria, padre della AP studenti
     result = DFServiceCommunicator.search(this, ¿
          Df_Ingegneria@host:port/JADE,dfd,c);
     if (result.size() > 0) {
              dfd = (DFAgentDescription)result.get(0);
              responder = dfd.getName();
     EsitoRicercaAgServizio = true;
     break;
          }
     else if (EsitoRicercaAgServizio==false) {
     // Inizio a risalire la gerarchia di APs e l'UDMA consulta il
     // il DF della AP ATENEO, padre della AP INGEGNERIA
     result = DFServiceCommunicator.search(this, ¿
              Df_Ingegneria@host:port/JADE,dfd,c);
     if (result.size() > 0) {
              dfd = (DFAgentDescription)result.get(0);
              responder = dfd.getName();
```

```
        EsitoRicercaAgServizio = true;
        break;
             }
        }
       }
     Thread.sleep(10000);
     }
   } catch (Exception fe) {
     System.err.println(getLocalName()+" ha ricercato presso il DF senza successo a ¿
         causa di " + fe.getMessage());
     doDelete();
   }

   String convID = newConvID();
   ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
   request.addReceiver(responder);
   request.setLanguage("Plain-Text");
   request.setContent(data);
   request.setConversationId(convID);
   MessageTemplate mt = MessageTemplate.MatchSender(responder);
     mainBehaviour.addSubBehaviour(new Behaviour1(this,request,mt));
     addBehaviour(mainBehaviour);
}
}

// E' la classe principale. Crea un nuovo contenitore d'agenti ed un Agente di
// Interfaccia per il client collegato.
public  class Udma {

   public static void main(String[] dati){

   String input=dati[0];

   // Perchè possa esistere l'agente necessita di un contenitore d'Agenti
   String [] _args = {"-container"};
   jade.Boot.main(_args);

   // Assegno un nome casuale all'agente
   int timeout = (int)(Math.random()*10000.0);
   Integer n = new Integer(timeout);
   String PersonalUdmaID = "UdmaID:"+n.toString();

   // Inizializzo l'Agente di interfaccia Udma
   UdmaAgent myUDMA=new UdmaAgent(input);
   myUDMA.doStart(PersonalUdmaID);

  }
}
```

A2. Agente di Servizio

```
/* Titolo: AteneoOnLine
*  Descrizione: Agente di servizio AgServizio
*  Release: #01 - Gennaio 2001
*  Copyright: Universita' Degli Studi Di Brescia
*  @Autore: Emiliano Vezzoli - vezzoli@evweb.it
*/

import java.io.BufferedWriter;
import java.io.OutputStreamWriter;
import java.util.*;
import java.io.*;
import java.lang.*;

import jade.core.Agent;
import jade.core.AID;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.proto.FipaRequestResponderBehaviour;
import jade.proto.FipaRequestInitiatorBehaviour;
import jade.domain.FIPAAgentManagement.*;
import jade.domain.FIPAException;
import jade.domain.DFServiceCommunicator;
import jade.core.behaviours.*;
import jade.core.behaviours.OneShotBehaviour;
import jade.core.behaviours.ComplexBehaviour;

/* Questo Agente e' incaricato, sulla base della richiesta pervenuta
*  dall'UDMA, di ricercare gli agenti preposti al servizio richiesto
*  tramite il DF. Dopodiche', in base ai risultati, interroga l'agente
*  dedicato al servizio in questione, aspettandone i risultati.
*  Infine invia la o le risposte all'UDMA.
*/

public class AgServizio extends Agent {

 public DFAgentDescription dfd = new DFAgentDescription();
 public int NextBehaviour=1, convCounter = 0;                    // variabili di controllo
 public String data;                                             // stringa ricevuta dall'Udma
 public AID Udma=new AID();                                      // identifica l'Udma
 public AID AgenteApplicazione=new AID();      // identifica l'Agente di applicazione
 public Agent AgenteServizio=new Agent();                        // identifica l'agente corrente

 // Questa classe permette di identificare piu' agenti impegnati nella conversazione
 private String newConvID(){
  String out = new String(getLocalName() + (new Integer(convCounter).toString()));
  convCounter++;
  return out;
 }
```

```
// Questa classe stampa un messaggio di errore verificatori al Behaviour X
void StampaErrore() {
 System.out.println(getLocalName() + ": Errore!! al Behaviour numero: " +
NextBehaviour);
}

// Questa classe si serve del FipaRequestResponderBehaviour per
// implementare il comportamento dell'agente Responder nel protocollo
// Fipa-request

public class Behaviour1 extends FipaRequestResponderBehaviour.ActionHandler ¿
    implements FipaRequestResponderBehaviour.Factory {

  Behaviour1(Agent a, ACLMessage msg){
  super(a,msg);
  }

 // Implementazione del metodo Factory
 public FipaRequestResponderBehaviour.ActionHandler create(ACLMessage msg){
  return new Behaviour1(myAgent, msg);
 }

 public void action(){
  double chance = Math.random();
  System.out.println("\n Chance N: "+chance);
  if (chance <0.1)
   sendReply(ACLMessage.NOT_UNDERSTOOD,"(Chance < 0.2)");
  else if (chance < 0.2)
   sendReply(ACLMessage.REFUSE,"((Chance < 0.2) \"Agente troppo occupato!! ¿
       Riprovare piu' tardi!\")");
  else {
   sendReply(ACLMessage.AGREE,"(Successo, l'Agente di servizio ha accettato ¿
       la richiesta!!)");
   chance = Math.random();
  if (chance<0.1)
   sendReply(ACLMessage.FAILURE,"((Chance < 0.1) \"Si sono verificati ¿
       errori sulla porta\" )");
  else
   sendReply(ACLMessage.INFORM, "(OK! L'Agente di Servizio fornira' ¿
       i dati richiesti il piu' velocemente possibile.) ");

     // Se il messaggio e' arrivato correttamente a destinazione, l'agente di servizio
     // attiva i prossimi comportamenti.
      NextBehaviour=2;
      ComplexBehaviour mainBehaviour=new SequentialBehaviour();
      mainBehaviour.addSubBehaviour(new Behaviour2());
      mainBehaviour.addSubBehaviour(new Behaviour3());
      addBehaviour(mainBehaviour);
  }
 }
```

```
  // Viene eseguito una volta sola, quindi ritorna sempre TRUE
  public boolean done() {
    return true;
  }

  // Questo metodo risulta facoltativo, non essendoci la necessita' di una
  // riinizializzazione del protocollo.
  public void reset() {
  }
 }// Fine Behaviour 1

class myFipaRequestResponderBehaviour extends FipaRequestResponderBehaviour
   {
   myFipaRequestResponderBehaviour(Agent a) {
     super(a);
   }
   protected String getActionName(ACLMessage msg) throws ¿
            NotUnderstoodException, RefuseException {
     data=msg.getContent();                    // Informazioni sul messaggio ricevuto.
     Udma=msg.getSender();                     // Informazioni sull'Udma chiamante.
     System.out.println("L'AgServizio ha ricevuto il seguente messaggio: "+data);
     System.out.println("Da parte dell' Udma chiamato: "+Udma.getName());
     return "ServiceRequest";
   }
 } // Fine classi FipaRequestResponderBehaviour ( Behaviour 1 )

  class Behaviour2 extends OneShotBehaviour {

     public void action() {
       if (NextBehaviour==2) {
         System.out.println("\nAgente: " + getName() + " Secondo ¿
             Behaviour " + NextBehaviour);
         System.out.println("\nRichiesta Ricevuta con successo!!");
         System.out.println("\nMessaggio: "+ data);
         NextBehaviour=3;
         }
       else
         StampaErrore();
         doWait(10000);
         reset();
     }
   }

// Questo comportamento si occupa nuovamente del parsing della stringa, creando
// un oggetto java di tipo QueryX con i parametri del servizio richiesto. Cosi'
// L AgServizio passera' agli agenti di applicazione direttamente questo oggetto,
// opportunamente codificato da Jade in Base64.

class Behaviour3 extends OneShotBehaviour {

  public String limit="&=\n";
  private String[] vect;
   int i=0, NumeroCampi=0;
```

```
public String[] TokParse(String input) {
 String limit="&=\n";                    // marcatori che delimitano i tokens nella stringa

 StringTokenizer tok=new StringTokenizer(input, limit);
 NumeroCampi=tok.countTokens();
 for (i=0;i<NumeroCampi;i++) {
  vect[i]=tok.nextToken();
  }
return(vect);
}

public void action() {

if (NextBehaviour==3) {
  TokParse(data);
  if (vect[0].equals("servizio")) {
     System.out.println("\nRiepilogo Informazioni Richiesta Servizio:");
     if (vect[1].equals("ricercaemail")) {
     Query1 richiesta=new Query1(vect[1],vect[3],vect[5]);
     System.out.println("\nServizio= " + richiesta.servizio);
     System.out.println("\nNome= " + richiesta.nome);
     System.out.println("\nCognome= " + richiesta.cognome);
     NextBehaviour=4;
     addBehaviour(new Behaviour4(richiesta));
     }
     else if (vect[1].equals("ricercahomepage")) {
     Query1 richiesta=new Query1(vect[1],vect[3],vect[5]);
     System.out.println("\nServizio= " + richiesta.servizio);
     System.out.println("\nNome= " + richiesta.nome);
     System.out.println("\nCognome= " + richiesta.cognome);
     NextBehaviour=4;
     addBehaviour(new Behaviour4(richiesta));
     }
     else if (vect[1].equals("ricercainfo")) {
     Query1 richiesta=new Query1(vect[1],vect[3],vect[5]);
     System.out.println("\nServizio= " + richiesta.servizio);
     System.out.println("\nSettore= " + richiesta.settore);
     System.out.println("\nMateria= " + richiesta.materia);
     NextBehaviour=4;
     addBehaviour(new Behaviour4(richiesta));
     }
     else {
          System.out.println("\nATTENZIONE! Il servizio richiesto non ¿
              e' attualmente disponibile");
          doDelete();
          }
  } else {
          System.out.println("\nERRORE! Non e' stato specificato alcun servizio");
          doDelete();
          }
 }
else {
```

```
   StampaErrore();
   doWait(10000);
   reset();
   }
  }
}   // fine Behaviour3

// L'AgServizio ricerca presso il DF della piattaforma un agente di applicazione
// dedicato alla fornitura di un servizio specifico, cercando l'esistenza di agenti
// caratterizzati dal TYPE "NomeServizio".
class Behaviour4 extends OneShotBehaviour {

   private String NomeServizio;
   private Query1 Richiesta;

       public Behaviour4(Query1 NS){
       NomeServizio=NS.servizio;
       Richiesta=NS;
       }


   public void action() {

   if (NextBehaviour==4) {

   DFAgentDescription dfd = new DFAgentDescription();
   ServiceDescription sd = new ServiceDescription();
   sd.setType(NomeServizio);
   dfd.addServices(sd);
   SearchConstraints c = new SearchConstraints();
   try {
     while (true) {
           System.out.println(getLocalName()+ " e' in attesa che un Agente ¿
               addetto alla fornitura del Servizio si registri presso il DF");
           List result = DFServiceCommunicator.search ¿
               (AgenteServizio,getDefaultDF(),dfd,c);
           if (result.size() > 0) {
             dfd = (DFAgentDescription)result.get(0);
             AgenteApplicazione = dfd.getName();
             break;
           }
           Thread.sleep(10000);
     }
   } catch (Exception fe) {
     System.err.println(getLocalName()+" ha ricercato presso il DF senza ¿
         successo a causa di " + fe.getMessage());
     doDelete();
   }

   String convID = newConvID();
```

```
   // Una volta individuato l'agente di applicazione desiderato, viene costruito
   // un opportuno RECORD con i parametri per l'espletamento del servizio.
   ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
   request.addReceiver(AgenteApplicazione);
   try {
     request.setContentObject(Richiesta);
     } catch(IOException e){
     System.err.println(getLocalName()+" ATTENZIONE! Errore di impostazione ¿
        parametri nella richiesta di fornitura del servizio.");
     }
   request.setConversationId(convID);
   NextBehaviour=5;
   MessageTemplate mt = MessageTemplate.MatchSender(AgenteApplicazione);
   addBehaviour(new Behaviour5(AgenteServizio,request,mt));
   }
   else {
   StampaErrore();
   doWait(10000);
   reset();
   }
   }
} // Fine Behaviour 4

// Questo comportamento implementa il protocolli FIPA Request Initiator
// per la conversazione con altri agenti. L'AgServizio invia il RECORD.
class Behaviour5 extends FipaRequestInitiatorBehaviour {

   Behaviour5(Agent a,ACLMessage msg, MessageTemplate mt) {
     super(a, msg, mt);
     System.out.println("\nAgente " + getLocalName() + " ha inviato il ¿
        seguente messaggio:\n"+msg.toString());
   }

  protected void handleNotUnderstood(ACLMessage msg){
    System.out.println("\nATTENZIONE! Il destinatario non ha capito la richiesta!");
    System.out.println(msg.toString());
    reset(waitAndGetNewMsg(msg));
  }

  protected void handleRefuse(ACLMessage msg) {
    System.out.println("\nIl destinatario rifiuta di accogliere la richiesta!");
    System.out.println(msg.toString());
    reset(waitAndGetNewMsg(msg));
  }

  protected void handleAgree(ACLMessage msg) {
    System.out.println("\n\tSuccesso! Il destinatario ha accettato la richiesta!");
    System.out.println(msg.toString());
    NextBehaviour=6;
    addBehaviour(new Behaviour6());
  }
```

```
  protected void handleFailure(ACLMessage msg) {
   System.out.println("\n Il procedimento di richiesta e' fallito a causa di:");
   System.out.println(msg.getContent());
   System.out.println(msg.toString());
   reset(waitAndGetNewMsg(msg));
  }

  protected void handleInform(ACLMessage msg) {
   System.out.println("\nIl destinatario avverte che la richiesta inoltrata ¿
      e' gia' stata accolta.");
   System.out.println(msg.toString());
  }

  private ACLMessage waitAndGetNewMsg(ACLMessage msg){
   // Attesa casuale tra 0 e 10 secondi.
   int timeout = (new Double(Math.random()*10000.0)).intValue();
   System.out.println("Agente : "+myAgent.getLocalName()+" attendera' "+ ¿
      timeout + "millisecondi prima di riinizializzare il protocollo");
   myAgent.doWait(timeout);
   System.out.println("Agente : "+myAgent.getLocalName()+" e' attivo ");
   ACLMessage reply = msg.createReply();
   reply.setContent(data);
   System.out.println("Agente : "+myAgent.getLocalName()+" ha inviato ¿
      il seguente messaggio:");
   System.out.println(reply.toString());
   return reply;
   }
}// Fine Behaviour 5

// L'agente di servizio attende la risposta da un'agente di applicazione.
// Se non giunge, il Behaviour viene sospeso fino all'arrivo del prossimo messaggio.

class Behaviour6 extends OneShotBehaviour {

    private String RisultatoRicerca;

    // set di caratteristiche della risposta che si aspetta l'Agente di servizio
    MessageTemplate R1 = MessageTemplate.MatchPerformative ¿
        (ACLMessage.INFORM);
    MessageTemplate R2 = MessageTemplate.MatchOntology("SearchResult");
    MessageTemplate R3 = MessageTemplate.MatchLanguage("PlainText");
    MessageTemplate R4 = MessageTemplate.and(R1,R2);
    MessageTemplate Rtot = MessageTemplate.and(R4,R3);

    private void WaitForAnswer() {

        ACLMessage msg = blockingReceive(Rtot);

           if (msg!= null){
           RisultatoRicerca=msg.getContent().toString();
           System.out.println(getLocalName() + " Ha ricevuto questa risposta ¿
              alla richiesta inoltrata precedentemente: "+ RisultatoRicerca);
           NextBehaviour=7;
```

```
            addBehaviour(new Behaviour7(RisultatoRicerca));
                }
              else
              {
              System.out.println(getLocalName()+": Ancora nessuna risposta ¿
                 e' stata ricevuta dal sistema ad Agenti...");
              block();
             }
    }

    public void action() {
      if (NextBehaviour==6) {
        WaitForAnswer();
      }
      else
        StampaErrore();
      }
    }// Fine Behaviour 6

// In questo Behaviour l'AgServizio completa il suo lavoro, inviando i risultati
// all'agente di interfaccia ed effettuando il reset dei suoi comportamenti.
class Behaviour7 extends OneShotBehaviour {

  private String RisultatoRicerca;

  public Behaviour7(String Ris){
  RisultatoRicerca=Ris;
  }

  public void action() {
      if (NextBehaviour==7) {

          ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
          msg.setSender(AgenteServizio.getAID());
          msg.addReceiver(Udma);
          msg.setOntology("SearchResult");
          msg.setLanguage("PlainText");
          msg.setContent(RisultatoRicerca);
          send(msg);
          System.out.println(getLocalName() + " Ha inviato questi dati    ¿
              all'UDMA : "+RisultatoRicerca);

          // Reset dei Behaviour.
          // L'AgServizio si mette in attesa di nuove richieste.
          FipaRequestResponderBehaviour UdmaRequester2 = new    ¿
                myFipaRequestResponderBehaviour(AgenteServizio);
          UdmaRequester2.registerFactory("ServiceRequest",new    ¿
                Behaviour1(AgenteServizio,null));
          addBehaviour(UdmaRequester2);
          System.out.println(getLocalName()+": STDBY");
          NextBehaviour=8;
      }
      else StampaErrore();
```

```
    }
}// Fine Behaviour 7

 protected void setup(){

  // L'agente di servizio per prima cosa si registra presso il DF
  ServiceDescription sd = new ServiceDescription();
  sd.setType("ServiceAgent");
  sd.setName(getName());
  sd.addOntologies("AteneoOnLine");
  dfd.setName(getAID());
  dfd.addServices(sd);
  try {
    DFServiceCommunicator.register(this,dfd);
  } catch (FIPAException e) {
    System.err.println(getLocalName()+" Registrazione presso il DF fallita! ¿
       Motivo: "+e.getMessage());
    doDelete();
  }
  // Fine Registrazione presso il DF
  System.out.println(getLocalName()+ " si e' registrato con successo presso il DF");
  AgenteServizio=this;    // Necessario per un riferimento corretto nei subBehaviours
  FipaRequestResponderBehaviour UdmaRequester = new  ¿
      myFipaRequestResponderBehaviour(this);
  UdmaRequester.registerFactory("ServiceRequest",new Behaviour1(this,null));
  addBehaviour(UdmaRequester);

 }

} // Fine Classe Agente di Servizio
```

A3. Email Search Agent

```
/* Titolo: AteneoOnLine
*  Descrizione: Agente di Applicazione dedicato alla Ricerca Email
*  Release #01 - Gennaio 2001
*  Copyright: Universita' Degli Studi Di Brescia
*  @Autore: Emiliano Vezzoli - vezzoli@evweb.it
*/

import java.io.BufferedWriter;
import java.io.OutputStreamWriter;
import java.util.*;
import java.io.*;
import java.lang.*;
import java.sql.*;

import jade.core.Agent;
import jade.core.AID;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.lang.acl.UnreadableException;
import jade.proto.FipaRequestResponderBehaviour;
import jade.proto.FipaRequestInitiatorBehaviour;
import jade.domain.FIPAAgentManagement.*;
import jade.domain.FIPAException;
import jade.domain.DFServiceCommunicator;
import jade.core.behaviours.*;
import jade.core.behaviours.OneShotBehaviour;
import jade.core.behaviours.ComplexBehaviour;

/* Questo Agente e' incaricato, sulla base dei parametri ( nome e cognome docente )
*  ricevuti dall' AgServizio, di attivare la ricerca in un database dell'indirizzo
*  email del docente specificato e di spedirgli la risposta.
*/

public class AgApplicazione1_RicercaEmail extends Agent {

 public DFAgentDescription dfd = new DFAgentDescription();
 public int NextBehaviour=1;                          // Variabili di controllo
 public Query1 parametri;                            // Parametri della ricerca
 public AID AgServizio=new AID();                     // identifica l'AgServizio
 public Agent AgenteApplicazione=new Agent();         // identifica l'agente corrente

 // Questa classe stampa un messaggio di errore verificatori al Behaviour X
 void StampaErrore() {
  System.out.println(getLocalName() + ": Errore!! al Behaviour numero: "
    + NextBehaviour);
 }
```

```
 // Questa classe apre un database ODBC e fornisce l'indirizzo email del docente
 // in base ai campi di ricerca NOME e COGNOME, sfruttando gli statements SQL.

public String CercaEmail(String NomeDocente, String CognomeDocente) throws ¿
       SQLException, ClassNotFoundException { {

    String dbUrl = "jdbc:odbc:Email";
    String user = "EmailSearchAgent";
    String password = "********";
    String IndirizzoEmailDocente="NessunRisultato";

    // Caricamento driver
    new sun.jdbc.odbc.JdbcOdbcDriver();

    Connection c = DriverManager.getConnection(dbUrl,user,password);
    Statement s = c.createStatement();
    // CODICE SQL
    ResultSet rs = s.executeQuery("SELECT Nome, Cognome, Email ¿
        FROM Indirizzi");
    while(rs.next()) {
      String Nome = rs.getString(1).trim();
      String Cognome = rs.getString(2).trim();
      String Email = rs.getString(3).trim();
      if ((Nome.equals(NomeDocente))&&(Cognome.equals(CognomeDocente))) {
        IndirizzoEmailDocente=Email;
        break;
        }
      }
    s.close();  // Chiude il collegamento con il DataBase;
    return(IndirizzoEmailDocente);
    }
}

 // Questa classe si serve del FipaRequestResponderBehaviour per
 // mettere l'Agente di Applicazione in attesa di essere contattato da un
 // Agente di Servizio.

 public class Behaviour1 extends FipaRequestResponderBehaviour.ActionHandler ¿
   implements FipaRequestResponderBehaviour.Factory {

   Behaviour1(Agent a, ACLMessage msg){
    super(a,msg);
   }

  // Implementazione del metodo Factory
  public FipaRequestResponderBehaviour.ActionHandler create(ACLMessage msg){
    return new Behaviour1(myAgent, msg);
  }

  public void action(){
    double chance = Math.random();
    System.out.println("\n Chance N: "+chance);
    if (chance <0.1)
```

```
      sendReply(ACLMessage.NOT_UNDERSTOOD,"(Chance < 0.2)");
     else if (chance < 0.2)
      sendReply(ACLMessage.REFUSE,"((Chance < 0.2) \"Agente ¿
         troppo occupato!! Riprovare piu' tardi!\")");
     else {
      sendReply(ACLMessage.AGREE,"(Successo, l'Agente addetto ¿
         alla Ricerca E-Mail ha accettato la richiesta!!)");
      chance = Math.random();
      if (chance<0.1)
       sendReply(ACLMessage.FAILURE,"((Chance < 0.1) \"Si sono ¿
          verificati errori sulla porta\" )");
      else
       sendReply(ACLMessage.INFORM, "(OK! L'Agente addetto alla ¿
          Ricerca E-Mail fornira' i dati richiesti il piu' velocemente possibile.) ");

       // Se il messaggio e' arrivato correttamente a destinazione, l'agente
       // di applicazione attiva il prossimo comportamento.
        NextBehaviour=2;
        addBehaviour(new Behaviour2(parametri.nome,parametri.cognome));

     }
   }
   // Viene eseguito una volta sola, quindi ritorna sempre TRUE
   public boolean done() {
    return true;
   }
   public void reset() {
   }
  }// Fine Behaviour 1

class myFipaRequestResponderBehaviour extends FipaRequestResponderBehaviour
{
    myFipaRequestResponderBehaviour(Agent a) {
     super(a);
    }
    protected String getActionName(ACLMessage msg) throws
NotUnderstoodException, RefuseException {
    try {
     parametri=(Query1)msg.getContentObject();   // Parametri ricevuti relativi al
servizio.
     } catch(UnreadableException e){
     System.err.println(getLocalName()+" ATTENZIONE! Errore di riconoscimento ¿
        dei parametri per la ricerca, a causa di: "+e.getMessage());
     }
     AgServizio=msg.getSender();    // Informazioni sull'AgServizio chiamante.
     System.out.println("L'Agente addetto alla Ricerca E-Mail ha ricevuto ¿
        l'incarico di ricercare ");
     System.out.println("l'indirizzo di posta elettronica del docente: ¿
        "+parametri.nome+" "+parametri.cognome);
     System.out.println("da parte dell' Agente chiamato: "+AgServizio.getName());
     return "ServiceRequest";
    }
  } // Fine classi FipaRequestResponderBehaviour ( Behaviour 1 )
```

```
// Procedura per la ricerca dei risultati
class Behaviour2 extends OneShotBehaviour {

    private String nome, cognome;
    private String risultato;

    public Behaviour2(String name, String surname){
     nome=name;
     cognome=surname;
    }

     public void action() {
       if (NextBehaviour==2) {
         try {
          risultato=CercaEmail(nome,cognome);
          } catch ( Exception e ) {
          System.err.println(getLocalName()+" ERRORE: "+e);
          }
          if (risultato.equals("NessunRisultato")) {
          System.out.println(getLocalName()+": l'indirizzo di posta elettronica ¿
             richiesto non e' disponibile nel DataBase.");
          } else {
          System.out.println(getLocalName()+": l'indirizzo di posta elettronica ¿
             richiesto e': "+risultato);
          }
          NextBehaviour=3;
          addBehaviour(new Behaviour3(risultato));
          }
        else
          StampaErrore();
          doWait(10000);
          reset();
     }
}


// In questo Behaviour l'Agente completa il suo lavoro, inviando i risultati
// all'agente di servizio ed effettuando il reset dei suoi comportamenti.
class Behaviour3 extends SimpleBehaviour {

  private String IndirizzoEmail;
  private boolean finished=false;

  public Behaviour3(String Ris){
  IndirizzoEmail=Ris;
  }

  public void action() {
      if (NextBehaviour==3) {

          ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
          msg.setSender(AgenteApplicazione.getAID());
          msg.addReceiver(AgServizio);
```

```
          msg.setLanguage("PlainText");
          msg.setOntology("SearchResult");
          msg.setContent(IndirizzoEmail);
          send(msg);
          finished=true;
          // Reset dei Behaviour.
          // L'Agente di Applicazione si mette in attesa di nuove richieste.
          FipaRequestResponderBehaviour AgAppRequester2 = new ¿
               myFipaRequestResponderBehaviour(AgenteApplicazione);
          AgAppRequester2.registerFactory("ServiceRequest",new ¿
               Behaviour1(AgenteApplicazione,null));
          addBehaviour(AgAppRequester2);
          System.out.println(getLocalName()+": STDBY");

       }
       else StampaErrore();
      }
  public boolean done(){
   return finished;
  }
}// Fine Behaviour 7


 public void setup() {

  // L'agente di Applicazione per prima cosa si registra presso il DF
  ServiceDescription sd = new ServiceDescription();
  sd.setType("ricercaemail");
  sd.setName(getName());
  sd.addOntologies("AteneoOnLine");
  dfd.setName(getAID());
  dfd.addServices(sd);
  try {
    DFServiceCommunicator.register(this,dfd);
  } catch (FIPAException e) {
    System.err.println(getLocalName()+" Registrazione presso il DF fallita! ¿
       Motivo: "+e.getMessage());
    doDelete();
  }
  // Fine Registrazione presso il DF
  System.out.println(getLocalName()+ " si e' registrato con successo presso il DF");
  AgenteApplicazione=this;
  FipaRequestResponderBehaviour AgAppRequester = new ¿
       myFipaRequestResponderBehaviour(this);
  AgAppRequester.registerFactory("ServiceRequest",new Behaviour1(this,null));
  addBehaviour(AgAppRequester);
 }
} // Fine AgApplicazione1_RicercaEmail
```

A4. HomePage Search Agent

```
/* Titolo: AteneoOnLine
*  Descrizione: Agente di Applicazione dedicato alla Ricerca Home Page
*  Release #01 - Gennaio 2001
*  Copyright: Universita' Degli Studi Di Brescia
*  @Autore: Emiliano Vezzoli - vezzoli@evweb.it
*/

import java.io.BufferedWriter;
import java.io.OutputStreamWriter;
import java.util.*;
import java.io.*;
import java.lang.*;
import java.sql.*;

import jade.core.Agent;
import jade.core.AID;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.lang.acl.UnreadableException;
import jade.proto.FipaRequestResponderBehaviour;
import jade.proto.FipaRequestInitiatorBehaviour;
import jade.domain.FIPAAgentManagement.*;
import jade.domain.FIPAException;
import jade.domain.DFServiceCommunicator;
import jade.core.behaviours.*;
import jade.core.behaviours.OneShotBehaviour;
import jade.core.behaviours.ComplexBehaviour;

/* Questo Agente e' incaricato, sulla base dei parametri ( nome e cognome docente )
*  ricevuti dall' AgServizio, di attivare la ricerca in un database dell'indirizzo
*  URL, se esiste,  del docente specificato e di spedirgli la risposta.
*/

public class AgApplicazione2_RicercaHomePage extends Agent {

 public DFAgentDescription dfd = new DFAgentDescription();
 public int NextBehaviour=1;                                    // Variabili di controllo
 public Query1 parametri;                                       // Parametri della ricerca
 public AID AgServizio=new AID();                               // identifica l'AgServizio
 public Agent AgenteApplicazione=new Agent();                   // identifica l'agente corrente

 // Questa classe stampa un messaggio di errore verificatori al Behaviour X
 void StampaErrore() {
  System.out.println(getLocalName() + ": Errore!! al Behaviour numero:  ¿
     " + NextBehaviour);
 }
```

```
 // Questa classe apre un database ODBC e fornisce l' Home Page del docente
 // in base ai campi di ricerca NOME e COGNOME, sfruttando gli statements SQL.

public String CercaHomePage(String NomeDocente, String CognomeDocente) ¿
    throws SQLException, ClassNotFoundException { {

    String dbUrl = "jdbc:odbc:HomePage";
    String user = "HomePageSearchAgent";
    String password = "*******";
    String HomePageDocente="NessunRisultato";

    // Caricamento driver
    new sun.jdbc.odbc.JdbcOdbcDriver();

    Connection c = DriverManager.getConnection(dbUrl,user,password);
    Statement s = c.createStatement();
    // CODICE SQL
    ResultSet rs = s.executeQuery("SELECT Nome, Cognome, HomePage ¿
        FROM Indirizzi");
    while(rs.next()) {
      String Nome = rs.getString(1).trim();
      String Cognome = rs.getString(2).trim();
      String HomePage = rs.getString(3).trim();
      if ((Nome.equals(NomeDocente))&&(Cognome.equals(CognomeDocente))) {
        HomePageDocente=HomePage;
        break;
        }
      }
    s.close(); // Chiude il collegamento con il DataBase;
    return(HomePageDocente);
    }
}

 // Questa classe si serve del FipaRequestResponderBehaviour per
 // mettere l'Agente di Applicazione in attesa di essere contattato da un
 // Agente di Servizio.

 public class Behaviour1 extends FipaRequestResponderBehaviour.ActionHandler
implements FipaRequestResponderBehaviour.Factory {

   Behaviour1(Agent a, ACLMessage msg){
    super(a,msg);
   }

  // Implementazione del metodo Factory
  public FipaRequestResponderBehaviour.ActionHandler create(ACLMessage msg){
    return new Behaviour1(myAgent, msg);
  }

  public void action(){
    double chance = Math.random();
    System.out.println("\n Chance N: "+chance);
    if (chance <0.1)
```

```
      sendReply(ACLMessage.NOT_UNDERSTOOD,"(Chance < 0.2)");
     else if (chance < 0.2)
      sendReply(ACLMessage.REFUSE,"((Chance < 0.2) \"Agente troppo ¿
         occupato!! Riprovare piu' tardi!\")");
     else {
      sendReply(ACLMessage.AGREE,"(Successo, l'Agente addetto ¿
         alla Ricerca HomePage ha accettato la richiesta!!)");
      chance = Math.random();
       if (chance<0.1)
        sendReply(ACLMessage.FAILURE,"((Chance < 0.1) \"Si sono verificati ¿
           errori sulla porta\" )");
       else
        sendReply(ACLMessage.INFORM, "(OK! L'Agente addetto alla ¿
           Ricerca HomePage fornira' i dati richiesti il piu' velocemente possibile.) ");

      // Se il messaggio e' arrivato correttamente a destinazione, l'agente di
      // applicazione attiva il prossimo comportamento.
      NextBehaviour=2;
      addBehaviour(new Behaviour2(parametri.nome,parametri.cognome));
     }
    }
   // Viene eseguito una volta sola, quindi ritorna sempre TRUE
   public boolean done() {
    return true;
   }
   public void reset() {
   }
  }// Fine Behaviour 1

class myFipaRequestResponderBehaviour extends FipaRequestResponderBehaviour
{
    myFipaRequestResponderBehaviour(Agent a) {
     super(a);
    }
    protected String getActionName(ACLMessage msg) throws ¿
       NotUnderstoodException, RefuseException {

    try {
     parametri=(Query1)msg.getContentObject();                    // Parametri ricevuti
     } catch(UnreadableException e){
     System.err.println(getLocalName()+" ATTENZIONE! Errore di riconoscimento
        dei parametri per la ricerca, a causa di: "+e.getMessage());
     }
     AgServizio=msg.getSender();              // Informazioni sull'AgServizio chiamante.
     System.out.println("L'Agente addetto alla Ricerca HomePage ha ricevuto ¿
        l'incarico di ricercare ");
     System.out.println("l'URL del docente: "+parametri.nome+" "+parametri.cognome);
     System.out.println("da parte dell' Agente chiamato: "+AgServizio.getName());
     return "ServiceRequest";
    }
  } // Fine classi FipaRequestResponderBehaviour ( Behaviour 1 )
```

```
// Procedura per la ricerca dei risultati
class Behaviour2 extends OneShotBehaviour {

    private String nome, cognome;
    private String risultato;

    public Behaviour2(String name, String surname){
     nome=name;
     cognome=surname;
    }

     public void action() {
       if (NextBehaviour==2) {
        try {
        risultato=CercaHomePage(nome,cognome);
        } catch ( Exception e ) {
        System.err.println(getLocalName()+" ERRORE: "+e);
        }
        if (risultato.equals("NessunRisultato")) {
        System.out.println(getLocalName()+": l'URL dell'HomePage richiesta ¿
           non e' disponibile nel DataBase.");
        } else {
        System.out.println(getLocalName()+": l'URL dell'HomePage richiesta ¿
           e': "+risultato);
        }
        NextBehaviour=3;
        addBehaviour(new Behaviour3(risultato));
        }
       else
        StampaErrore();
        doWait(10000);
        reset();
     }
}

// In questo Behaviour l'Agente completa il suo lavoro, inviando i risultati
// all'agente di servizio ed effettuando il reset dei suoi comportamenti.
class Behaviour3 extends SimpleBehaviour {

  private String HomePage;
  private boolean finished=false;

  public Behaviour3(String Ris){
  HomePage=Ris;
  }

  public void action() {
      if (NextBehaviour==3) {

          ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
          msg.setSender(AgenteApplicazione.getAID());
          msg.addReceiver(AgServizio);
          msg.setLanguage("PlainText");
```

```
          msg.setOntology("SearchResult");
          msg.setContent(HomePage);
          send(msg);
          finished=true;
          // Reset dei Behaviour.
          // L'Agente di Applicazione si mette in attesa di nuove richieste.
          FipaRequestResponderBehaviour AgAppRequester2 = new ¿
              myFipaRequestResponderBehaviour(AgenteApplicazione);
          AgAppRequester2.registerFactory("ServiceRequest",new ¿
              Behaviour1(AgenteApplicazione,null));
          addBehaviour(AgAppRequester2);
          System.out.println(getLocalName()+": STDBY");

        }
        else StampaErrore();
       }
  public boolean done(){
   return finished;
  }
} // Fine Behaviour 7


 public void setup() {

  // L'agente di Applicazione per prima cosa si registra presso il DF
  ServiceDescription sd = new ServiceDescription();
  sd.setType("ricercahomepage");
  sd.setName(getName());
  sd.addOntologies("AteneoOnLine");
  dfd.setName(getAID());
  dfd.addServices(sd);
  try {
    DFServiceCommunicator.register(this,dfd);
  } catch (FIPAException e) {
    System.err.println(getLocalName()+" Registrazione presso il DF fallita! ¿
       Motivo: "+e.getMessage());
    doDelete();
  }
  // Fine Registrazione presso il DF
  System.out.println(getLocalName()+ " si e' registrato con successo presso il DF");
  AgenteApplicazione=this;
  FipaRequestResponderBehaviour AgAppRequester = new ¿
      myFipaRequestResponderBehaviour(this);
  AgAppRequester.registerFactory("ServiceRequest",new Behaviour1(this,null));
  addBehaviour(AgAppRequester);
 }
} // Fine AgApplicazione3_RicercaHomePage
```

A5. Info Search Agent

```
/* Titolo: AteneoOnLine
*  Descrizione: Agente di Applicazione dedicato alla Ricerca Informazioni
*  Release #01 - Gennaio 2001
*  Copyright: Universita' Degli Studi Di Brescia
*  @Autore: Emiliano Vezzoli - vezzoli@evweb.it
*/

import java.io.BufferedWriter;
import java.io.OutputStreamWriter;
import java.util.*;
import java.io.*;
import java.lang.*;
import java.sql.*;

import jade.core.Agent;
import jade.core.AID;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.lang.acl.UnreadableException;
import jade.proto.FipaRequestResponderBehaviour;
import jade.proto.FipaRequestInitiatorBehaviour;
import jade.domain.FIPAAgentManagement.*;
import jade.domain.FIPAException;
import jade.domain.DFServiceCommunicator;
import jade.core.behaviours.*;
import jade.core.behaviours.OneShotBehaviour;
import jade.core.behaviours.ComplexBehaviour;

/* Questo Agente e' incaricato, sulla base dei parametri ( settore e materia )
*  ricevuti dall' AgServizio, di attivare la ricerca in un database del personale
*  universitario che si occupa di tale disciplina, fornendo informazioni su come
*  contattarlo, quali E-mail, Home Page, Telefono ufficio,...
*/

public class AgApplicazione3_RicercaInfo extends Agent {

 public DFAgentDescription dfd = new DFAgentDescription();
 public int NextBehaviour=1;                                    // Variabili di controllo
 public Query1 parametri;                                       // Parametri della ricerca
 public AID AgServizio=new AID();                               // identifica l'AgServizio
 public Agent AgenteApplicazione=new Agent();                   // identifica l'agente corrente

 // Questa classe stampa un messaggio di errore verificatori al Behaviour X
 void StampaErrore() {
  System.out.println(getLocalName() + ": Errore!! al Behaviour numero: "  ¿
    +  NextBehaviour);
 }

 // Questa classe apre un database ODBC e fornisce le informazioni richieste,
 // in base ai parametri SETTORE e MATERIA, sfruttando gli statements SQL.
```

```
public String CercaInfo(String set, String mat) throws SQLException, ¿
   ClassNotFoundException { {

   String dbUrl = "jdbc:odbc:Info";
   String user = "InfoSearchAgent";
   String password = "**********";
   String Informazioni = "";

   // Caricamento driver
   new sun.jdbc.odbc.JdbcOdbcDriver();

   Connection c = DriverManager.getConnection(dbUrl,user,password);
   Statement s = c.createStatement();
   // CODICE SQL
   ResultSet rs = s.executeQuery("SELECT Nome, Cognome, Email, HomePage, ¿
      Telefono, Settore, Materia FROM Indirizzi");
    System.out.println(getLocalName()+"Nel DATABASE sono stati trovati ¿
      i seguenti dati:");
   while(rs.next()) {
     String Nome = rs.getString(1).trim();
     String Cognome = rs.getString(2).trim();
     String Email = rs.getString(3).trim();
     String HomePage = rs.getString(4).trim();
     String Telefono = rs.getString(5).trim();
     String Settore = rs.getString(6).trim();
     String Materia = rs.getString(7).trim();
     if ((Settore.equals(set))&&(Materia.equals(mat))) {
       System.out.println("NOME: "+Nome+", COGNOME: "+Cognome+", EMAIL: ¿
          "+Email+", URL: "+HomePage+", TELEFONO: "+Telefono);
      // Viene costruita una stringa unica con tutti i risultati ottenuti, il
      // cui parsing verrà effettuato prima della rappresentazione all'utente.

Informazioni=Informazioni+"nome="+Nome+"&cognome="+Cognome+"&email="+E
mail+"&homepage="+HomePage+"&telefono="+Telefono+"&";
       }
     }
   s.close();  // Chiude il collegamento con il DataBase;
   return(Informazioni);
   }
}

 // Questa classe si serve del FipaRequestResponderBehaviour per
 // mettere l'Agente di Applicazione in attesa di essere contattato da un
 // Agente di Servizio.

public class Behaviour1 extends FipaRequestResponderBehaviour.ActionHandler ¿
  implements FipaRequestResponderBehaviour.Factory {

  Behaviour1(Agent a, ACLMessage msg){
   super(a,msg);
  }

 // Implementazione del metodo Factory
```

```
  public FipaRequestResponderBehaviour.ActionHandler create(ACLMessage msg){
    return new Behaviour1(myAgent, msg);
  }

  public void action(){
    double chance = Math.random();
    System.out.println("\n Chance N: "+chance);
    if (chance <0.1)
      sendReply(ACLMessage.NOT_UNDERSTOOD,"(Chance < 0.2)");
    else if (chance < 0.2)
      sendReply(ACLMessage.REFUSE,"((Chance < 0.2) \"Agente troppo ¿
        occupato!! Riprovare piu' tardi!\")");
    else {
      sendReply(ACLMessage.AGREE,"(Successo, l'Agente addetto alla Ricerca ¿
        Informazioni ha accettato la richiesta!!)");
      chance = Math.random();
       if (chance<0.1)
                sendReply(ACLMessage.FAILURE,"((Chance < 0.1) \"Si sono ¿
        verificati errori sulla porta\" )");
       else
        sendReply(ACLMessage.INFORM, "(OK! L'Agente addetto alla Ricerca ¿
          Informazioni fornira' i dati richiesti il piu' velocemente possibile.) ");

       // Se il messaggio e' arrivato correttamente a destinazione, l'agente di
       //  applicazione attiva il prossimo comportamento.
        NextBehaviour=2;
        addBehaviour(new Behaviour2(parametri.settore,parametri.materia));
    }
  }
  // Viene eseguito una volta sola, quindi ritorna sempre TRUE
  public boolean done() {
    return true;
  }
  public void reset() {
  }
 }// Fine Behaviour 1

class myFipaRequestResponderBehaviour extends FipaRequestResponderBehaviour
{
    myFipaRequestResponderBehaviour(Agent a) {
      super(a);
    }
    protected String getActionName(ACLMessage msg) throws ¿
       NotUnderstoodException, RefuseException {
    try {
      parametri=(Query1)msg.getContentObject();   // Parametri ricevuti
    } catch(UnreadableException e){
      System.err.println(getLocalName()+" ATTENZIONE! Errore di riconoscimento ¿
        dei parametri per la ricerca, a causa di: "+e.getMessage());
      }
      AgServizio=msg.getSender();    // Informazioni sull'AgServizio chiamante.
      System.out.println("L'Agente addetto alla Ricerca Informazioni ha ricevuto ¿
         l'incarico di ricercare ");
```

```
     System.out.println("Informazioni sul seguente argomento: "+ parametri.settore +" ¿
        "+ parametri.materia);
     System.out.println("da parte dell' Agente chiamato: "+AgServizio.getName());
     return "ServiceRequest";
    }
  } // Fine classi FipaRequestResponderBehaviour ( Behaviour 1 )

// Procedura per la ricerca dei risultati
class Behaviour2 extends OneShotBehaviour {

    private String Settore, Materia, Risultato;

    public Behaviour2(String set, String mat){
     Settore = set;
     Materia = mat;
    }

     public void action() {
       if (NextBehaviour==2) {
         try {
         Risultato=CercaInfo(Settore,Materia);
         } catch ( Exception e ) {
         System.err.println(getLocalName()+" ERRORE: "+e);
         }
         if (Risultato.equals("")) {
         System.out.println(getLocalName()+": Nessuna informazione relativa ¿
            all'argomento richiesto e' disponibile nel DataBase.");
         Risultato="NessunRisultato";
         } else {
         System.out.println(getLocalName()+": Informazioni Richieste: ");
         System.out.println(getLocalName()+": " + Risultato);
         }
         NextBehaviour=3;
         addBehaviour(new Behaviour3(Risultato));
         }
       else
         StampaErrore();
         doWait(10000);
         reset();
     }
}


// In questo Behaviour l'Agente completa il suo lavoro, inviando i risultati
// all'agente di servizio ed effettuando il reset dei suoi comportamenti.
class Behaviour3 extends SimpleBehaviour {

  private String StringaInformazioni;
  private boolean finished=false;

  public Behaviour3(String Ris){
  StringaInformazioni=Ris;
  }
```

```
  public void action() {
      if (NextBehaviour==3) {

          ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
          msg.setSender(AgenteApplicazione.getAID());
          msg.addReceiver(AgServizio);
          msg.setLanguage("PlainText");
          msg.setOntology("SearchResult");
          msg.setContent(StringaInformazioni);
          send(msg);
          finished=true;
          // Reset dei Behaviour.
          // L'Agente di Applicazione si mette in attesa di nuove richieste.
          FipaRequestResponderBehaviour AgAppRequester2 = new ¿
             myFipaRequestResponderBehaviour(AgenteApplicazione);
          AgAppRequester2.registerFactory("ServiceRequest",new  ¿
             Behaviour1(AgenteApplicazione,null));
          addBehaviour(AgAppRequester2);
          System.out.println(getLocalName()+": STDBY");
      }
      else StampaErrore();
      }
  public boolean done(){
   return finished;
  }
} // Fine Behaviour 7

 public void setup() {

   // L'agente di Applicazione per prima cosa si registra presso il DF
   ServiceDescription sd = new ServiceDescription();
   sd.setType("ricercainfo");
   sd.setName(getName());
   sd.addOntologies("AteneoOnLine");
   dfd.setName(getAID());
   dfd.addServices(sd);
   try {
     DFServiceCommunicator.register(this,dfd);
   } catch (FIPAException e) {
     System.err.println(getLocalName()+" Registrazione presso il DF fallita! ¿
        Motivo: "+e.getMessage());
     doDelete();
   }
   // Fine Registrazione presso il DF
   System.out.println(getLocalName()+ " si e' registrato con successo presso il DF");
   AgenteApplicazione=this;
   FipaRequestResponderBehaviour AgAppRequester = new  ¿
      myFipaRequestResponderBehaviour(this);
   AgAppRequester.registerFactory("ServiceRequest",new Behaviour1(this,null));
   addBehaviour(AgAppRequester);
  }
} // Fine AgApplicazione3_RicercaInfo
```

A6. Attivazione Sistema AOL-MAS

A6.a RunApStudenti

```
/* Titolo: AteneoOnLine
*  Descrizione: Applicazione che costruisce la piattaforma Studenti
*  Release #01 - Gennaio 2001
*  Copyright: Universita' Degli Studi Di Brescia
*  @Autore: Emiliano Vezzoli - vezzoli@evweb.it
*/

import java.io.*;
import java.util.*;
import java.lang.Object;
import java.lang.Runtime;

import jade.Boot;
import jade.core.Agent;

public  class RunAPstudenti {

   public static void main(String[] dati){

   // Impostazione parametri piattaforma ad Agenti JADE
    String[] IdStudenti={"-name","Studenti","-port","1099","-gui"};

   // Attivazione Piattaforma:
    Boot.main(IdStudenti);

   // Attivazione Agente di Servizio della Piattaforma
    AgServizio Service=new AgServizio();
    Service.doStart("AgServizio");

   // Attivazione Agente di Applicazione N'1: Ricerca Indirizzi E-Mail
    AgApplicazione1_RicercaEmail Re1=new AgApplicazione1_RicercaEmail();
    Re1.doStart("EmailSearchAgent");

   // Attivazione Agente di Applicazione N'2: Ricerca Home Page Docenti
    AgApplicazione2_RicercaHomePage Re2=new ¿
       AgApplicazione2_RicercaHomePage();
    Re2.doStart("HomePageSearchAgent");

   // Attivazione Agente di Applicazione N'3: Ricerca Info su materie
    AgApplicazione3_RicercaInfo Re3=new AgApplicazione3_RicercaInfo();
    Re3.doStart("InfoSearchAgent");
    }
 }
```

A6.b RunApIngegneria

```
/* Titolo: AteneoOnLine
*  Descrizione: Applicazione che costruisce la piattaforma di Ingegneria
*  Release #01 - Gennaio 2001
*  Copyright: Universita' Degli Studi Di Brescia
*  @Autore: Emiliano Vezzoli - vezzoli@evweb.it
*/

import java.io.*;
import java.util.*;
import java.lang.Object;
import java.lang.Runtime;

import jade.Boot;
import jade.core.Agent;

public  class RunAPingegneria {

   public static void main(String[] dati){

   // Impostazione parametri piattaforma ad Agenti JADE
    String[] IdStudenti={"-name","Ingegneria","-port","1199"};

   // Attivazione Piattaforma:
    Boot.main(IdStudenti);

   // Creo i subDFs rappresentanti il DF_Studenti e il DF_Docenti, figlio di
   // DF_Ingegneria.
    SubDF Studenti = new SubDF();
    Studenti.doStart("DF_Studenti");
     SubDF Docenti = new SubDF();
    Docenti.doStart("DF_Docenti");
  }
}
```

A6.c RunApAteneo

```
/* Titolo: AteneoOnLine
*  Descrizione: Applicazione che costruisce la piattaforma Ateneo
*  Release:  #01 - Gennaio 2001
*  Copyright: Universita' Degli Studi Di Brescia
*  @Autore: Emiliano Vezzoli - vezzoli@evweb.it
*/

import java.io.*;
import java.util.*;
import java.lang.Object;
import java.lang.Runtime;

import jade.Boot;
import jade.core.Agent;

public  class RunAPateneo {

   public static void main(String[] dati){

   // Impostazione parametri piattaforma ad Agenti JADE
    String[] IdStudenti={"-name","Ateneo","-port","1399"};

   // Attivazione Piattaforma:
    Boot.main(IdStudenti);

   // Creo un subDF rappresentante il DF_Ingegneria figlio di DF_Ateneo
    SubDF Ingegneria = new SubDF();
    Ingegneria.doStart("DF_Ingegneria");
    }
 }
```

A6.d Creazione SubDF

```
/* Titolo: AteneoOnLine
*  Descrizione: costruttore di un subDF
*  Release #01 - Gennaio 2001
*  Copyright: Universita' Degli Studi Di Brescia
*  @Autore: Emiliano Vezzoli - vezzoli@evweb.it
*/

import java.io.IOException;
import java.io.InterruptedIOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.InetAddress;

import jade.core.*;
import jade.core.behaviours.*;
import jade.domain.FIPAAgentManagement.*;
import jade.domain.FIPAException;
import jade.domain.DFServiceCommunicator;
import jade.lang.acl.ACLMessage;

public class SubDF extends jade.domain.df {

  public void setup() {

   int len = 0;   // Nome del DF di Input
  byte[] buffer = new byte[1024];

  try {
    AID parentName = getDefaultDF();
    super.setup();
    setDescriptionOfThisDF(getDescription());
    DFServiceCommunicator.register(this,parentName,getDescription());
    addParent(parentName,getDescription());
    System.out.println("Agente: " + getName() + " è stato federato col DF corrente.");
    }catch(FIPAException fe){fe.printStackTrace();}
 }
 private DFAgentDescription getDescription()
 {
    DFAgentDescription dfd = new DFAgentDescription();
    dfd.setName(getAID());
    ServiceDescription sd = new ServiceDescription();
    sd.setName(getLocalName() + "-sub-df");
    sd.setType("fipa-df");
    sd.addProtocols("fipa-request");
    sd.addOntologies("fipa-agent-management");
    sd.setOwnership("JADE");
    dfd.addServices(sd);
    return dfd;
 }
}
```

A7. Interfaccia Grafica Utente HTML

A7.a Frame principale

```
<html>

<head>
<title>AteneoOnLine-MAS</title>
<meta name="Author" content="Emiliano Vezzoli">
<meta content="Ita,Eng" name="DC.Language" SCHEME="RFC1766">
<meta name="ROBOTS" content="INDEX, FOLLOW">
<meta name="description"
content="Sistema multi-agente per l'accesso a servizi i rete">
<meta name="distribution" content="Global">
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
</head>

<frameset framespacing="0" border="false" frameborder="0" cols="230,*">
 <frame name="sommario" target="principale" src="menu.htm" scrolling="auto">
 <frame name="principale2" src="home.htm" scrolling="auto">
 <noframes>
 <body>
 </body>
 </noframes>
</frameset>
</html>
```

A7.b menù laterale

```
<html>

<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<title>Menu</title>
<meta name="Author" content="Emiliano Vezzoli">
<base target="principale">
</head>

<body bgproperties="fixed" bgcolor="#FFFFFF">
<div align="left">

<table border="0" cellpadding="0" cellspacing="0" bordercolor="#000000">
 <tr>
   <td></td>
 </tr>
 <tr>
   <td><div align="center"><center><table border="2" cellpadding="0"
cellspacing="3"
   bordercolor="#000000">
```

```
    <tr>
     <td bgcolor="#000000"><div align="center"><center><table border="0"
cellpadding="0"
     cellspacing="3">
      <tr>
       <td><a href="paginabianca.htm" target="principale"><img
src="images/TastoItaliano.jpg"
        width="75" height="70" alt="TastoItaliano.jpg (2775 byte)"
border="0"></a></td>
        <td><p align="center"><a href="index_e.htm" target="_top"><img
        src="images/TastoEnglish.jpg" width="75" height="70" alt="TastoEnglish.jpg
(3725 byte)"
        border="0"></a></td>
      </tr>
     </table>
     </center></div></td>
    </tr>
    <tr>
     <td bgcolor="#000000">
     <applet width="180" height="30" code="DropDownMenu.class"
codebase="Class"
     style="border: 0px solid rgb(64,0,0)">
      <param name="ROOT"
value="HOME;URL=home1.htm;TARGET=principale">
      <param name="STYLE" value="3DINVERSE">
      <param name="FONTFACE" value="Arial">
      <param name="FONTSIZE" value="20">
      <param name="FONTSTYLE" value="bold">
      <param name="BACKGROUND_COLOR" value="000000">
      <param name="FONT_COLOR" value="FF0000">
      <param name="HIGHLIGHT_COLOR" value="FF0000">
      <param name="TRIANGLE_UP_COLOR" value="FFFF00">
      <param name="TRIANGLE_DOWN_COLOR" value="0000FF">
      <param name="TRIANGLE_OFFSET" value="5">
      <param name="TRIANGLE_WIDTH" value="10">
      <param name="BGIMAGE_ALIGN" value="center">
      <param name="BGIMAGE_VALIGN" value="center">
      <param name="defaulttarget" value="principale">
      <param name="tile" value="true">
      <param name="textIndent" value="5">
      <param name="triangle_offset" value="225">
     </applet>
     </td>
    </tr>
    <tr>
     <td bgcolor="#000000">
     <applet width="180" height="500" code="DropDownMenu.class"
codebase="Class"
     style="border: 0px solid rgb(64,0,0)">
      <param name="ROOT"
value="MENU;STATE=EXPANDED;TARGET=main">
      <param name="SITEMAP"
      value="
```

```
SERVIZI STUDENTI ;{
Ricerca E-Mail Docenti;URL=RicercaEmail.htm;TARGET=principale|
Ricerca HomePage Docenti;URL=RicercaHomePage.htm;TARGET=principale|
A Chi Rivolgersi;URL=RicercaInfo.htm;TARGET=principale|
Piani di Studi;URL=piani.html;TARGET=principale|
Prestito libri;URL=libri.htm;TARGET=principale|
Certificati;URL=certificati.htm;TARGET=principale|
Pagine Degli Studenti;URL=studenti.htm;TARGET=principale;
    }

SERVIZI DOCENTI   ;{
Non disponibili;
    }

SEGRETERIA    ;{
Non disponibili;
}
ORARIO LEZIONI: {
Non disponibili;
}
CORSI DI LAUREA;{
Non disponibili;
}
CORSI DI DIPLOMA; {
Non disponibili;
}
ATTIVITA' CULTURALI; {
Non disponibili;
}

">
      <param name="STYLE" value="3DINVERSE">
      <param name="FONTFACE" value="Arial">
      <param name="FONTSIZE" value="14">
      <param name="FONTSTYLE" value="bold">
      <param name="SUBFONTFACE" value="Arial">
      <param name="SUBFONTSIZE" value="12">
      <param name="SUBFONTSTYLE" value="plain">
      <param name="BACKGROUND_COLOR" value="000000">
      <param name="FONT_COLOR" value="F1F1F1">
      <param name="HIGHLIGHT_COLOR" value="FF0000">
      <param name="TRIANGLE_UP_COLOR" value="FFFF00">
      <param name="TRIANGLE_DOWN_COLOR" value="0000FF">
      <param name="TRIANGLE_OFFSET" value="5">
      <param name="TRIANGLE_WIDTH" value=" 10">
      <param name="BGIMAGE_ALIGN" value="center">
      <param name="BGIMAGE_VALIGN" value="center">
      <param name="defaulttarget" value="principale">
      <param name="tile" value="true">
      <param name="textIndent" value="5">
      <param name="triangle_offset" value="225">
    </applet>
    </td>
```

```
    </tr>
</table></center></div></td></tr><tr><td>
</td></tr><tr><td></td></tr><tr><td></td></tr>
<tr><td></td></tr></table></div></body></html>
```

A7.c Esempio di Form per l'invio dati (RicercaEmail)

```
<html>

<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<meta name="Author" content="Emiliano Vezzoli">
<title>RicercaEmail</title>
</head>

<body bgcolor="#000000">
<div align="center"><center>

<table border="0" cellpadding="0" cellspacing="3" width="600">
 <tr>
  <td><img src="images/lineabianca.jpg" width="600" height="10"
  alt="lineabianca.jpg (2541 byte)"></td>
 </tr>
 <tr>
  <td bgcolor="#FFFFFF"><p align="center"><img src="images/titolo1.jpg"
width="400"
  height="70" alt="titolo1.jpg (10356 byte)"></td>
 </tr>
 <tr>
  <td><img src="images/lineabianca.jpg" width="600" height="10"
  alt="lineabianca.jpg (2541 byte)"></td>
 </tr>
 <tr>
  <td><font color="#FF0000" size="4" face="Arial"><strong>  
</strong></font></td>
 </tr>
 <tr>
  <td><p align="center"> <font color="#FF0000" size="4"
face="Arial"><strong>INSERISCI
   NEL FORM SOTTOSTANTE I DATI RELATIVI AL DOCENTE CHE VUOI
CONTATTARE TRAMITE EMAIL:</strong></font></td>
 </tr>
 <tr>
  <td><font color="#FF0000" size="4" face="Arial"><strong>   
</strong></font></td>
 </tr>
 <tr>
  <td><div align="center"><center><table border="3" cellpadding="4"
bordercolor="#FFFFFF"
  bordercolorlight="#FFFFFF" bordercolordark="#C0C0C0" bgcolor="#FFFFFF"
width="600">
```

```
    <tr>
     <td><p align="center">  </p>
     <form NAME="myfrom" METHOD="GET"
ACTION="http://crauto.ing.unibs.it/cgi-bin/Udma.bat">
      <p><font face="Arial" color="#000000"><strong><input SIZE="30"
NAME="servizio"
      VALUE="ricercaemail" style="color: rgb(0,0,255)"> Servizio Richiesto
</strong></font></p>
      <p><font face="Arial" color="#000000"><strong><input SIZE="30"
NAME="nome"
      style="color: rgb(255,0,0)"> Inserisci qui il nome del Docente
</strong></font></p>
      <p><font face="Arial" color="#000000"><strong><input SIZE="30"
NAME="cognome"
      style="color: rgb(255,0,0)"> Inserisci qui il cognome del Docente
</strong></font></p>
      <p><input TYPE="submit" VALUE="ATTIVA IL SISTEMA AD AGENTI"
      style="color: rgb(255,0,0); font-weight: bold"> </p>
     </form>
     </td>
    </tr>
   </table>
   </center></div></td>
 </tr>
 <tr>
   <td>     </td>
 </tr>
 <tr>
   <td><img src="images/lineabianca.jpg" width="600" height="10"
   alt="lineabianca.jpg (2541 byte)"></td>
 </tr>
 <tr>
   <td><p align="center"><a href="home1.htm" style="color: rgb(0,255,255)"
   target="principale"><font face="Arial"><strong>HOME</strong></font></a></td>
 </tr>
</table>
</center></div>
</body>
</html>
```

A7.d Esempio di Form di selezione (RicercaInfo)

```
<html>

<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<meta name="Author" content="Emiliano Vezzoli">
<title>RicercaInfo</title>
</head>

<body bgcolor="#000000">
<div align="center"><center>
```

```
<table border="0" cellpadding="0" cellspacing="3" width="600">
 <tr>
   <td><img src="lineabianca.jpg" width="600" height="10" alt="lineabianca.jpg (2541
byte)"></td>
 </tr>
 <tr>
   <td bgcolor="#FFFFFF"><p align="center"><img src="images/titolo3.jpg"
  alt="titolo1.jpg (10356 byte)" width="400" height="70"></td>
 </tr>
 <tr>
   <td><img src="lineabianca.jpg" width="600" height="10" alt="lineabianca.jpg (2541
byte)"></td>
 </tr>
 <tr>
   <td><font color="#FF0000" size="4" face="Arial"><strong>  
</strong></font></td>
 </tr>
 <tr>
   <td><p align="center"> <font color="#FF0000" size="4"
face="Arial"><strong>INSERISCI
   NEL FORM SOTTOSTANTE I DATI RELATIVI AL SETTORE  ED ALLA
MATERIA A CUI SEI
   INTERESSATO.</strong></font></td>
 </tr>
 <tr>
   <td><font color="#FF0000" size="4" face="Arial"><strong>   
</strong></font></td>
 </tr>
 <tr>
   <td><div align="center"><center><table border="3" cellpadding="4"
bordercolor="#FFFFFF"
  bordercolorlight="#FFFFFF" bordercolordark="#C0C0C0" bgcolor="#FFFFFF"
width="600">
    <tr>
     <td><p align="center">  </p>
     <form NAME="myform" METHOD="GET"
ACTION="http://crauto.ing.unibs.it/cgi-bin/Udma.bat">
       <p><font face="Arial" color="#000000"><strong><input SIZE="30"
NAME="servizio"
       VALUE="ricercainfo" style="color: rgb(0,0,255)"> Servizio Richiesto
</strong></font></p>
       <p><font face="Arial" color="#FF0000"><big><strong>SETTORE E
MATERIA DI INTERESSE: </strong></big></font></p>
       <p><font face="Arial"><strong><input TYPE="RADIO" NAME="settore"
VALUE="civile">
       Ingegneria Civile

       </strong></font></p>
       <p><font face="Arial"><strong><input TYPE="RADIO" NAME="settore"
VALUE="gestionale">
       Ingegneria Gestionale
</strong></font></p>
```

```
        <p><font face="Arial"><strong><input TYPE="RADIO" NAME="settore"
VALUE="elettronica">
        Ingegneria Elettronica
          
</strong></font></p>
        <p><font face="Arial"><strong><input TYPE="RADIO" NAME="settore"
VALUE="meccanica">
        Ingegneria Meccanica </strong></font>     </p>
        <p>      </p>
        <p><font face="Arial"><strong><select NAME="materia" size="3">
          <option value="algebra">001 - Algebra ed Elementi di geometria</option>
          <option value="analisi1">002 - Analisi Matematica I</option>
          <option value="analisi2">003 - Analisi Matematica II</option>
          <option value="architettura">004 - Automazione e Organizzazione
Sanitaria</option>
          <option value="calcolatori">005 - Calcolatori Elettronici</option>
          <option value="calcolonumerico">006 - Calcolo Numerico</option>
          <option value="chimica">007 - Chimica</option>
          <option value="fisica1">008 - Fisica Generale I</option>
          <option value="fisica2">009 - Fisica Generale II</option>
          <option value="informatica">010 - Fondamenti di Informatica I</option>
          <option value="informatica2">011 - Fondamenti di Informatica II</option>
        </select></strong></font></p>
        <p> </p>
        <p><input TYPE="submit" VALUE="ATTIVA IL SISTEMA AD AGENTI"
        style="color: rgb(255,0,0); font-weight: bold"> </p>
       </form>
       </td>
     </tr>
   </table>
   </center></div></td>
 </tr>
 <tr>
   <td>     </td>
 </tr>
 <tr>
   <td><img src="lineabianca.jpg" width="600" height="10" alt="lineabianca.jpg (2541
byte)"></td>
 </tr>
 <tr>
   <td><p align="center"><a href="home1.htm" style="color: rgb(0,255,255)"
   target="principale"><font face="Arial"><strong>HOME</strong></font></a></td>
 </tr>
</table>
</center></div>
</body>
</html>
```

A8. Agente Snooper

```
import jade.core.*;
import jade.core.behaviours.*;
import jade.lang.acl.*;

/**
 * This agent is to be used in a JSP page. It just sends
 * messages to a buffer agent.
 */
public class Snooper extends Agent {
   private ACLMessage msg;

   public Snooper() {
      // Create the message to send to the client
      msg = new ACLMessage(ACLMessage.INFORM);
   }

   /**
    * The method that will be invoked in the JSP page.
    * @param str the message to send to the client
    */
   public void snoop(String str) {
      // JADE 2.0:
     // getHap() cannot be moved in the constructor because it would not work!
     // each time the previous entry must be removed.
      msg.clearAllReceiver();
      msg.addReceiver(new AID("buffer@"+getHap()));
      // JADE 1.4:
      // msg.removeAllDests();
      //msg.addDest("buffer");
      msg.setContent(str);
      send(msg);
   }

  /* just for testing
protected void setup() {
  snoop("CIAO");
}*/
}
```

A9. Agente Buffer

```
import java.util.Vector;
import jade.core.*;
import jade.core.behaviours.*;
import jade.lang.acl.*;

/**
 * This agent manage all the messages received from the JSP page.
 * If there is an agent listening somewhere, the received messages
 * are forwarded to this agent, either they are buffered.
 */
public class Buffer extends Agent {
   private Vector buffer;
   private boolean online;
   //  JADE 2.0
   private AID clientName;
   // JADE 1.4
  //private String clientName;

   public Buffer() {
          buffer = new Vector();
          online = false;
   }

   /**
    * When an inform message is received (from the JSP page),
    * then the message is forwarded to an online listener,
    * or either cached. A reception confirmation is needed before
    * deleting the forwarded message.
    */
   class ReceiveBehaviour extends CyclicBehaviour {
          private MessageTemplate m1;
          private ACLMessage msg;
          public ReceiveBehaviour(Agent a) {
             super(a);
             m1 = MessageTemplate.MatchPerformative(ACLMessage.INFORM);
          }

          public void action() {
             // Wait for an inform message from the snooper and store it
             msg = receive(m1);
             if (msg!= null){
                    buffer.add(msg.getContent()+"\n");
                    if (online) {
                       addBehaviour(new SendAndWaitConfirmBehaviour(myAgent));
                    }
             } else {
                    // block if there is no message for this behaviour
                    block();
             }
          }
```

```
}

/**
 * Wait during 10 seconds a receiving confirmation of the client
 * agent. If a confirmation is received, the buffer is cleared.
 */
class SendAndWaitConfirmBehaviour extends SimpleBehaviour {
        private boolean finished;
        private static final long TIMEOUT = 10000; //10 seconds
        private MessageTemplate m1;
    private long maximumTime;

        public SendAndWaitConfirmBehaviour(Agent a) {
           super(a);
           ACLMessage msg2 = new ACLMessage(ACLMessage.INFORM);
           // JADE 2.0:
           msg2.addReceiver(clientName);
           // JADE 1.4:
           //msg2.addDest(clientName);
           StringBuffer stb = new StringBuffer();
           for (int i=0;i<buffer.size();i++)
                   stb.append(buffer.get(i));
           msg2.setContent(stb.toString());
           send(msg2);
           finished=false;
           m1 = MessageTemplate.MatchPerformative(ACLMessage.CONFIRM);
           maximumTime = System.currentTimeMillis() + TIMEOUT;
        }

        /**
         * This task is finished only when a confirmation is received
         * or the timeout is reached.
         */
        public boolean done() {
           return finished;
        }

        public void action() {
           ACLMessage msg = myAgent.receive(m1);
           if (msg != null) {
                   // A confirmation is received
                   buffer.clear();
                   finished=true; // the behaviour is finished
           } else {
             if (System.currentTimeMillis() >= maximumTime)
                   // timeout is elapsed
                   // the behaviour is finished without clearing the buffer
                   finished=true;
             else // timout not yet elapsed
                   block(maximumTime - System.currentTimeMillis());
           }
        }
}
```

```
/**
 * Wait a connection from the Client.
 */
class WaitRequestBehaviour extends CyclicBehaviour {
        MessageTemplate m1;
        ACLMessage msg;

        public WaitRequestBehaviour(Agent a) {
           super(a);
           m1 = MessageTemplate.MatchPerformative(ACLMessage.REQUEST);
        }

        public void action() {
           // wait for a request message from the client and send it
           // the content of the buffer. Then the buffer is cleaned.
           msg = receive(m1);
           if (msg!= null){
                   online = true;
                   // JADE 2.0
                   clientName = msg.getSender();
                   // JADE 1.4
                   //clientName = msg.getSource();
                   addBehaviour(new SendAndWaitConfirmBehaviour(myAgent));
           } else
                   // block if there is no message for this behaviour
                   block();
        }
}

/**
 * Wait a disconnection from the client.
 */
class NotOnlineBehaviour extends CyclicBehaviour {
        MessageTemplate m1;
        ACLMessage msg;

        public NotOnlineBehaviour(Agent a) {
           super(a);
           m1 = MessageTemplate.MatchPerformative(ACLMessage.CANCEL);
        }

        public void action() {
           msg = receive(m1);
           if (msg!= null){
                   online = false;
           } else
                   // block if there is no message for this behaviour
                   block();
        }
}

protected void setup() {
        addBehaviour(new ReceiveBehaviour(this));
```

```
        addBehaviour(new WaitRequestBehaviour(this));
        addBehaviour(new NotOnlineBehaviour(this));
    }
}
```

A10. Agente Client

```
import java.io.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

import jade.core.*;
import jade.core.behaviours.*;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

public class Client extends Agent {
    // Used to display the messages
    private JTextArea result;

    public Client() {
        result = new JTextArea();
    }

    /**
     * Wait for a message from the buffer. Confirm the reception.
     */
    class MyBehaviour extends CyclicBehaviour {
        private MessageTemplate m1;

        public MyBehaviour(Agent a) {
            super(a);
            m1 = MessageTemplate.MatchPerformative(ACLMessage.INFORM);
        }

        public void action() {
            // Wait for a message from the Buffer
            ACLMessage msg = receive(m1);
            if (msg!= null){
                // display the message
                result.append(msg.getContent());
                // confirm that the message has been received.
                ACLMessage reply = msg.createReply();
                reply.setPerformative(ACLMessage.CONFIRM);
                send(reply);
            } else {
                block();
            }
        }
    }
```

```
  protected void unsubscribe() {
        ACLMessage msg = new ACLMessage(ACLMessage.CANCEL);
        // JADE 2.0:
        msg.addReceiver(new AID("buffer@"+getHap()));
        // JADE 1.4:
        //msg.addDest("buffer");
        send(msg);
  }

  protected void setup() {
        addBehaviour(new MyBehaviour(this));
        // create a GUI that display the messages
        JFrame frame = new JFrame("Snooper client");
        // When the window is closed, unsubscribe from the buffer.
        frame.addWindowListener(new WindowAdapter() {
              public void windowClosing(WindowEvent we) {
                 unsubscribe();
                 doDelete();
                 System.exit(0);
                 // kill all the container because there is also
                 // 1 client running.
                 // doDelete();
              }
           });
        JScrollPane scp = new JScrollPane();
        scp.getViewport().add(result);
        frame.getContentPane().add(scp);
        frame.setSize(300,200);
        frame.setVisible(true);

        // ask to the buffer stored messages.
        ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
        // JADE 2.0:
        msg.addReceiver(new AID("buffer@"+getHap()));
        // JADE 1.4:
        //msg.addDest("buffer");
        send(msg);
  }
}
```

BIBLIOGRAFIA

- **[AGL98]** - IBM, Inc. IBM AGLETS Documentation Web Page , 1998, *http://aglets.trl.ibm.co.jp/documentation.html*.
- **[AIW97]** - "Design Worshop on Open Intelligent Agent Platforms and Protocols", First Meeting of the Agent Interop Working Group, 1997.
- **[APA00]** – The Apache Software Foundation, "Apache HTTP Web Server Project", 2000, *www.apache.org*
- **[BAL98]** – Baldinelli M. , "La programmazione Multiagente" , Mokabyte numero 16, Febbraio 1998, *www.mokabyte.it*.
- **[BAM96]** - Mark Balmer, "The Java Framework for CGI", version 1.01, 1996, *www.msys.ch*.
- **[BET98]** - Bettini Lorenzo, "Agenti Mobili in Java" , Mokabyte numero 20, Giugno 1998, *www.mokabyte.it*.
- **[CGI96]** - CGI : Common Gataway Interface, Specifications and Documentation, 1996, *www.w3.org/CGI/Overview.html*, *http://hoohoo.ncsa.uiuc.edu/cgi/intro.html*.
- **[CIC99]** - Ciceri Alberto e De Maria Marcello, " Un file system distribuito ad Agenti Mobili: gestione dei diritti", Università degli Studi di Como, 1999.
- **[COR97]** - CORBA Facilities: Mobile Agents System Interoperability Facilities Standard, 1997.
- **[DEL99]** – Delaurenti Marco, " Programmazione orientata agli oggetti ", 1999, *www.vlsilab.polito.it/thesis/node84.html.*
- **[DEN87]** - Dennet, "The Intentional Stance", The MIT Press, Cambridge, Ma, 1987.

- **[ECK00]** - Eckel Bruce, " Thinking in Java ", 2nd Edition, Revision 11, 2000, *www.mindview.net*.
- **[FAR99]** - Farr George e Couthard Phil, "JDBC: come accedere ai dati con Java", IBM, 1999, *www.news400.net*.
- **[FIP00]** - FIPA, Foundation For Intelligent Physical Agents, "Specifications and Documentation", 1997/2000, *www.fipa.org*.
- **[GAR00]** – Garibotti Luigi, "Progetto e realizzazione di un'architettura multiagente basata sullo standard FIPA per la fornitura di beni e servizi in una comunità di utenti di grandi dimensioni", Università degli Studi di Brescia, Anno Accademico 1999/2000.
- **[GEN97]** - Genesereth M., "Agent-Based Framework for Interoperability", Sofware Agents, 1997.
- **[GOL98]** – Golia Riccardo, " Lo standard CGI ", Dipartimento di Elettronica e Informatica, Università di Padova, 1998.
- **[GRA96]** - Gray R.S., "Agent TCL: A Flexible and Secure Mobile-Agents System", Proceedings of the fourth annual TCL/TK workshop (TCL '96), 1996.
- **[HAD99]** – Hadingham Rob, Posland Stefan, Buckle Phil, "The FIPA-OS Agent Platform: Open Source for Open Standards", Nortel Networks, London Road, UK, 1999.
- **[HAL97]** – Hall Marty, " Core Web Programming, Html, Java, CGI ", Prentice Hall, 1997.
- **[HOR95]** - Horberg J., "Talk to my Agents: Software Agents in Virtual Reality", Computer Mediated Communication Magazine, vol.2, numero 2, Febbraio 1995.
- **[JAD00]** - JADE, Java Agents Development Environment, documentazione tecnica e guida alla programmazione, 2000, *http://sharon.cselt.it/projects/jade*.

- **[JEN95]** - Jennings N.R. and M.Wooldridge , "Intelligent Agents: Theory and Practice", Knowloledge Enginnering Review, 1994/95.
- **[JVS95]** - D.Johansen, R.Van Renesse, F.B.Schneider, " An Introduction to the TACOMA distribuited System ", Tecnical Report 95-23, Department of Computer Science, University of Troms0, 1995.
- **[KAR98]** – Karnik N., Tripathi A., "Agent Server Architecture for the AJANTA Mobile-Agent System". In proceedings of the 1998 International Conference on Parallel & Distribuited Processing Tecniques and Applications, 1998.
- **[LEB00]** - Le Berre Daniel, " Using Jade with Java Server Pages (JSP) ", 2000, *http://cafe.newcastle.edu.au/daniel*.
- **[LEM98]** - Lemay Laura, Rogers Cadenhead, "JAVA 1.2, Guida Completa", Apogeo, 1998.
- **[MAG98]** - Magedanz T., "Agent Cluster Baseline Document", 1998.
- **[MAN98]** – Manola Frank, "Agent Standards Overview", OBJS Technical Note, 1998.
- **[MAS00]** - Mastella Stefano, "Personalizzazione dell'intefaccia Uomo-Macchina in un sistema multi-Agente per la fornitura di servizi via rete", Università Degli Studi di Brescia, Anno Accademico 1999/2000.
- **[MCC79]** - McCarthy J., "Ascribing Mental Qualities to machines", Technical Report Memo, Stanford University AI LAB, Stanford, CA, 1979.
- **[MIT97]** - Mitsubishi Electric, "Concordia: An Infrastructure for Collaborating Mobile Agents". In proceedings of the 1st International Workshop on Mobile Agents, 1997.

- **[NEG95]** – Negroponte Nicholas, "Being Digital", Alfred A.Knopf, 1995.
- **[NOR00]** – Nortel Networks, "FIPA-OS Developers Guide", 2000, *www.nortelnetworks.com/fipa-os*.
- **[NWA96]** – Nwana H. , "Software Agents Overview and Classification", 1996.
- **[OBJ97]** - ObjectSpace Inc. " ObjectSpace Voyager Core Package Technical Overview", Technical Report, 1997.
- **[ODY97]** - Odyssey FAQ, General Magic inc., 1997, *www.genmagic.com/agents/odissey-faq.html*.
- **[OMG96]** – Object Management Group-common Facilities RFP3 final Draft, 1996, *www.omg.org*.
- **[PUL98]** - Puliti, "Servlet, JDBC e JavaServer", Mokabyte numero 20, Giugno 1998, *www.mokabyte.it*.
- **[RAM00]** - Raùl Ramos-Pollàn, "Lezione 7: come Java comunica su Internet", 2000, *http://www.ba.infn.it/~zito/jsem/settimana.html*.
- **[SAT99]** - Software Agents Theory, "A General Guide to Agent-Oriented Project Development", 1999, *www.spiralnebula.demon.co.uk*.
- **[SCI97]** – Sciabarra Michele, " Java: il JDBC ", 1997, *www.sciabarra.com*.
- **[SJL97]** - Danijel Skocaj, Ales Jaklic, Ales Leonardis, Franc Solina, "Building Java – Client / CGI - Server Application", 1997, *http://razor.fri.unilj.si*.
- **[SSS96]** - Aurthor Van Hoff, Sami Shaio, Orca Starbuck, and Sun Microsystems Inc., " Hooked on Java ", Addison Wesley Publishing Co., 1996.

- **[TRA98]** – Trainotti Michele, "L'interrogazione di DataBase distribuiti con Agenti Mobili", Università Degli Studi di Trento, Mokabyte numero 17, Marzo 1998, *www.mokabyte.it*.
- **[WHI95]** – White J.E. , " Mobile Agents", Technical Report, General Magic inc., 1995.