Indice

INTRODUZIONE

Nell'ultimo decennio abbiamo assistito ad una diffusione su larga scala delle tecnologie legate all'informatica, in particolare a quelle legate alla trasmissione dell'informazione.Tutto ciò ha reso disponibili una grande quantità di dati accessibili, basti pensare al fenomeno *internet*.
L'abbondanza di informazioni ha però reso problematico, da parte dell'utente, il discernere e l'isolare i dati significativi, fenomeno dell'*information overload* (sovraccarico di informazioni).
A complicare ulteriormente lo scenario, oltre alla quantità di sorgenti dati, c'è anche la varietà di queste sorgenti; i dati cercati , infatti, si trovano spesso in più documenti collocati su diverse sorgenti frequentemente eterogenee.
L'eterogeneità si manifesta sotto diversi aspetti: differenti piattaforme hardware e software sulle quali sono implementate le sorgenti (DBMS, linguaggi di interrogazione, ...), differenti modelli di dati (relazionale, ad oggetti, ...), differenti schemi di rappresentazione della struttura logica.
Per ottenere il dato cercato, quindi , l'utente dovrebbe innanzitutto conoscere le piattaforme su cui può essere memorizzato, la struttura logica di tali sorgenti, i meccanismi di interrogazione da utilizzare ed infine essere in grado di interpretare i risultati ottenuti.
Da qui la necessità di realizzare strumenti in grado di omogeneizzare, in modo automatico o semi-automatico, l'accesso alle risorse informative. Nasce quindi la problematica dell'integrazione delle sorgenti dati.
L'importanza e l'interesse intorno all'argomento non solo dal punto di vista teorico, ma anche applicativo, sono dimostrate dallo sviluppo, a livello commerciale, di sistemi quali *Sistemi di Workflow, Datawarehouse, Dataminer, ecc* che mirano ad ottenere proprio questo risultato.
Le tecniche convenzionali di integrazione sono complicate dal fatto che l'eterogeneità semantica si presenta su grande scala.
Questa eterogeneità coinvolge la terminologia, la struttura ed il dominio delle fonti, riguardo alle funzioni geografiche, organizzative e funzionali dell'uso delle informazioni.
Inoltre , per fare fronte alle richieste dei sistemi d'informazione globali basati su Internet , è importante che gli strumenti sviluppati per il sostegno di queste attività siano quanto più possibile semiautomatici e scalabili.

Per affrontare i problemi riferiti alla scalabilità su grande scala, può essere utile l'utilizzo di agenti mobili intelligenti nella zona di integrazione delle informazioni e, come ci si propone in questa tesi, la loro integrazione nell'infrastruttura di MOMIS (**M**ediator Envir**O**nment for **M**ultiple **I**nformation **S**ources).
MOMIS è un sistema che è stato concepito come un insieme di strumenti capaci di fornire un accesso integrato alle informazioni eterogenee memorizzate in basi di dati tradizionali (per esempio relazionali, e basi di dati orientate ad oggetti) o a file system , come ad esempio le fonti di dati semi-strutturate (XML – file).
MOMIS e' teso al raccoglimento e all'elaborazione degli aspetti semantici di schemi descrittivi delle fonti di informazioni in modo da agevolare l'integrazione e l'ottimizzazione delle interrogazioni.
La particolarità del progetto è il tipo di approccio utilizzato per l'integrazione, cioè un approccio di tipo semantico. Questo consente un'integrazione basata non solo sulla struttura delle sorgenti, ma anche sul significato che il progettista assegna ai singoli campi delle sorgenti stesse.
MOMIS adotta un'architettura a tre livelli:

1 - livello *dati*

2 - livello *mediatore*

3 - livello *utente*

Il mediatore è la parte fondamentale del sistema, cioè il tool semiautomatico che realizza il vero e proprio processo di integrazione. Questi è formato da due macro blocchi:

1 - *Global Schema Builder*

2 - *Query Manager*

Global Schema Builder è il modulo che si occupa dell'integrazione delle sorgenti, il processo avviene grazie all'utilizzo del tool grafico SI DESIGNER, che genera un unico schema globale.

Query Manager è il gestore delle interrogazioni, che, partendo da una singola query formulata dall'utente sullo schema globale, compone ed invia le query alle interfacce con le sorgenti. L'interfacciamento con le sorgenti è garantito dai *Wrapper* che, presidiando le singole sorgenti, hanno il compito di fornire diversi servizi:

1 - fornire una descrizione, nel linguaggio comune ODLi3, della sorgente alla quale e' connesso;

2 - permettere l'esecuzione di query locali e fornire i risultati al mediatore;

3 - esportare i significati, assegnati dal progettista della sorgente, relativi ai singoli campi;

4 - esportare le relazioni intra-schema relative alla struttura;

Questa tesi si occupa del possibile utilizzo di agenti intelligenti all'interno del progetto MOMIS. In primo luogo ci si propone di affrontare lo studio della struttura dell'architettura della piattaforma JADE, analizzando gli oggetti che la compongono, le loro caratteristiche e le loro funzionalità. Inoltre si studierà il concetto di agente (specificando caratteristiche, capacità e possibili utilizzi), proponendo alcuni esempi di agenti JADE , che si occupano di operazioni di gestione delle informazioni all'interno della piattaforma e che sfruttano le proprietà di mobilità tipiche degli agenti JADE. Infine si è affrontato il progetto e lo sviluppo di un agente capace di :

- individuare, tramite la rete Internet, nuove sorgenti informative
- analizzarne il contenuto
- decidere se i dati contenuti sono di particolare interesse
- archiviare le sorgenti informative ritenute interessanti
- creare per ogni sorgente una struttura ad albero gerarchico

1 - Un Sistema Intelligente per l'Integrazione delle Informazioni

La disponibilità di sorgenti di dati, soprattutto sulla rete, ha reso disponibili una grande quantità di informazioni. Allo stesso tempo, però, l'accesso alle informazioni si è notevolmente complicato, questo è dovuto soprattutto alla grande eterogeneità dei dati disponibili, sia per quanto riguarda la natura (testi, immagini, etc.), sia il modo in cui vengono descritti.
Gli standard esistenti (TPC/IP, ODBC, OLE, CORBA, SQL, etc.) risolvono parzialmente i problemi relativi alle diversità hardware e software, dei protocolli di rete e di comunicazione tra i moduli; rimangono però irrisolti quelli relativi alla modellazione delle informazioni.
Infatti i modelli e gli schemi dei dati sono differenti e questo crea una eterogeneità semantica (o logica) non risolvibile da questi standard.
Gli approcci all'integrazione, descritti in letteratura presentano diverse metodologie come ad esempio il *repository independence*, e cioè un approccio che prevede di isolare al di sotto di una vista integrata le applicazioni ed i dati integrati dalle sorgenti, consentendo la massima autonomia e nascondendo al tempo stesso le differenze esistenti; i *datawarehouse* che realizzano presso l'utente finale delle viste, ovvero delle porzioni di sorgenti, replicando fisicamente i dati ed affidandosi ad algoritmi di 'allineamento' per assicurare la loro consistenza in caso di modifiche nelle sorgenti originali.
Verrà ora presentata la proposta dell'ARPA (Advanced Research Projects Agency) per la realizzazione di un'architettura che, al tempo stesso, consenta sia l'autonomia delle sorgenti, sia la flessibilità dell'architettura stessa, senza dover ricorrere alla duplicazione fisica dei dati.

1.1 – L'integrazione intelligente delle informazioni

L'integrazione delle Informazioni (I^2) si distingue da quella dei dati e dei database in quanto non cerca di collegare semplicemente alcune sorgenti ma, piuttosto, di fornire risultati opportunamente selezionati da esse mediante l'uso di tecniche di Intelligenza Artificiale.
Per ottenere i risultati cercati sono quindi necessarie *conoscenza* ed *intelligenza* finalizzate alla scelta delle sorgenti e dei dati, alla loro fusione e alla conseguente sintesi. Si parla allora di Integrazione Intelligente di Informazioni (*Intelligent Integration of Information*, I^3).

1.1.1 – Il programma I^3

Il programma I^3 è un progetto realizzato dall'ARPA, Agenzia facente capo al Dipartimento della Difesa degli Stati Uniti, che mira ad indicare un'architettura di riferimento che realizzi l'integrazione di sorgenti eterogenee in modo automatico.
L'utilizzo di tecniche di intelligenza artificiale accresce il valore informativo dei dati trattati, permette, infatti, di dedurre informazioni direttamente dagli schemi delle sorgenti.
I^3 prevede l'introduzione di architetture sviluppabili seguendo uno standard che definisca i servizi da inserire in qualsiasi integratore, ed abbassi i costi di sviluppo e mantenimento.
Un approccio di questo tipo risolverebbe i problemi di realizzazione, manutenzione ed adattabilità, di eventuali supersistemi, creati ad arte per integrare una notevole quantità di sorgenti non correlate semanticamente.
Riuscendo a riutilizzare la tecnologia già sviluppata, la costruzione di nuovi sistemi risulterebbe più veloce e meno difficoltosa, con conseguente abbassamento dei costi.
Per poter sfruttare un'elevata riusabilità bisogna disporre di interfacce ed architetture standard.
Il paradigma suggerito per la suddivisione dei servizi e delle risorse nei diversi moduli si articola su due dimensioni:

- *orizzontalmente* in tre livelli: livello utente, livello intermedio che fa uso di tecniche di IA, livello delle sorgenti di dati;

- *verticalmente*: diversi domini in cui raggruppare le sorgenti.

I domini nei vari livelli non sono strettamente connessi, ma si scambiano dati ed informazioni la cui combinazione avviene a livello dell'utilizzatore, riducendo la complessità totale del sistema e permettendo lo sviluppo di applicazioni con finalità diverse.
Ad esempio, si supponga di dover integrare informazioni su trasporti marittimi ed aerei: ciò permette all'utente di avere un'idea completa su quale mezzo di trasporto sia più conveniente per le sue esigenze. Aggiungendo a questo altri domini, quali le distanze chilometriche e le velocità dei mezzi, si possono, ad esempio, facilitare le scelte di un rappresentante che deve decidere come e quando iniziare un determinato viaggio.
Il livello intermedio è quello su cui è più importante concentrare l'attenzione per quello che concerne l'uso di tecniche di intelligenza artificiale; esso infatti costituisce il tramite tra i dati posti nelle sorgenti e le applicazioni sviluppate per gli utenti.
Questo livello deve fornire diversi servizi quali:

- servizi dinamici per la selezione delle sorgenti;
- gestione degli accessi;
- gestione delle *interrogazioni*;
- acquisizione e combinazione dei dati;
- analisi e sintesi delle informazioni ottenute.

1.2 – Architettura di riferimento per sistemi I^3

L'obiettivo del programma I^3 è quello di ridurre il tempo necessario per la realizzazione di un integratore di informazioni, fornendo una raccolta e una formalizzazione delle soluzioni prevalenti nel campo della ricerca.
Le problematiche che stanno alla base del progetto I^3 sono sostanzialmente due:

- la difficoltà che si ha nel ricercare delle informazioni all'interno della molteplicità delle sorgenti di informazione individuabili;

- la constatazione del fatto che le fonti di informazione e i sistemi informativi, pur essendo spesso semanticamente correlati tra di loro, non lo sono in una forma semplice ne preordinata.

Proporremo un'architettura di riferimento che rappresenta alcuni dei servizi che un integratore di informazioni deve fornire e le possibili interconnessioni fra essi.
Essa individua cinque famiglie di attività omogenee, illustrate in Figura 1 unitamente ai loro legami. La reciproca iterazione tra queste attività consente di eseguire le operazioni di comunicazione, traduzione ed integrazione dei dati nelle sorgenti.

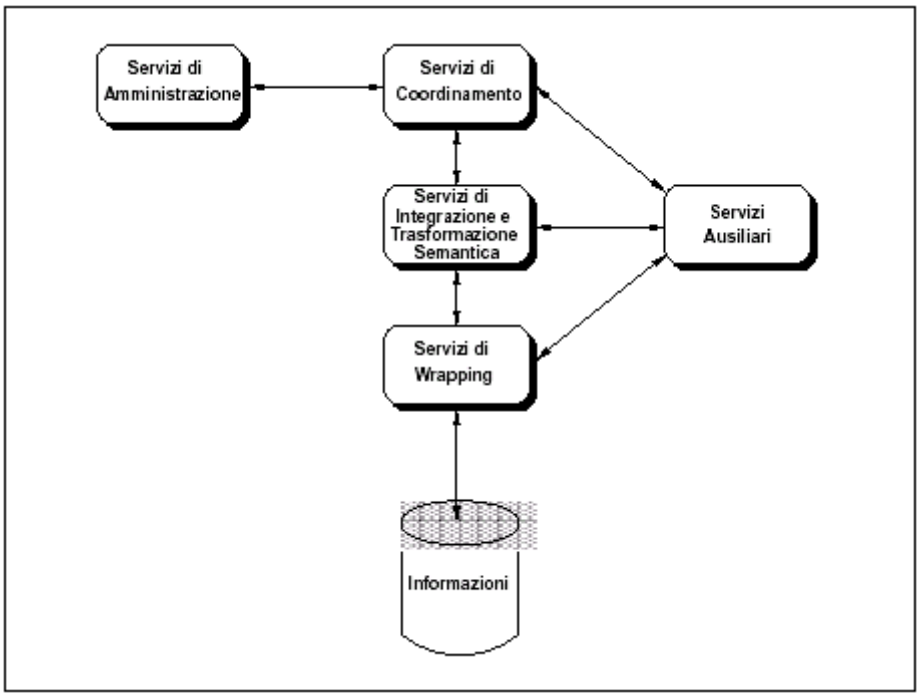

Figura 1

Osservando i servizi individuati nell'architettura proposta è possibile notare che essi sono organizzati, sostanzialmente, su due assi:uno orizzontale e l'altro verticale.
Percorrendo l'asse verticale si può intuire come avviene lo scambio di informazioni nel sistema: in particolare, i servizi di *wrapping* provvedono ad estrarre le informazioni dalle singole sorgenti, tali informazioni vengono poi impacchettate ed integrate dai Servizi di Integrazione e Trasformazione Semantica, per poi essere passate ai servizi di Coordinamento che ne avevano fatto richiesta.
L'osservazione dell'asse orizzontale, invece, mette in risalto il rapporto tra i servizi di Coordinamento e quelli di Amministrazione ai quali spetta , infatti, il compito di mantenere informazioni sulle capacità delle varie sorgenti , ovvero che tipo di dati possono fornire ed in quale modo devono essere interrogate.
Sono presenti inoltre i servizi Ausiliari, responsabili dei servizi di arricchimento semantico delle sorgenti. Vengono ora descritte nel dettaglio le specifiche funzionalità di ogni servizio e le problematiche che devono essere affrontate.

1.2.1 – Servizi di coordinamento

I servizi di Coordinamento sono quei servizi di alto livello che permettono l'individuazione delle sorgenti di dati *interessanti*, ovvero che probabilmente possono dare risposta ad una determinata richiesta dell'utente.
A seconda delle possibilità dell'integratore che si vuole realizzare, possono essere rappresentati da meccanismi che includono dalla selezione dinamica delle sorgenti (o brokering, per Integratori Intelligenti) fino al semplice *Matchmaking*, in cui il mappaggio tra informazioni integrate e locali è realizzato manualmente ed una volta per tutte.

Vediamo alcuni esempi.

1 - **Facilitation e Brokering Services** : l'utente manda una richiesta al sistema e questo usa un deposito di metadati per ritrovare il modulo che può trattare la richiesta direttamente.
I moduli interessati da questa richiesta possono essere o uno solo alla volta (nel qual caso si parla di Brokering) oppure più di uno (e in questo secondo caso si tratta di facilitatori e mediatori, attraverso i quali a partire da una richiesta ne viene generata più di una da inviare singolarmente a differenti moduli che gestiscono sorgenti distinte, e reintegrando poi le risposte in modo da presentarle all'utente come se fossero state ricavate da un'unica fonte).
In quest'ultimo caso, in cui una query può essere decomposta in un insieme di sottoquery, si farà uso di servizi di Query Decomposition e di tecniche di Inferenza (mutuate dall'Intelligenza Artificiale) per una determinazione dinamica delle sorgenti da interrogare, a seconda delle condizioni poste nell'interrogazione.
I vantaggi che questi servizi di Coordinamento portano stanno nel fatto che non è richiesta all'utente del sistema una conoscenza del contenuto delle diverse sorgenti, ma viene fornita un'unica rappresentazione delle fonti e in questo modo un sistema omogeneo che gestisce direttamente la sua richiesta.
All'utente pertanto non è richiesta la conoscenza dei domini con i quali i vari moduli I^3 si trovano ad interagire.E' quindi evidente la considerevole diminuzione di complessità di interazione col sistema che deriva da un'architettura di questo tipo.

2 - **Matchmaking**: il sistema viene configurato manualmente da un operatore al momento dell'inizializzazione. Successivamente tutte le richieste vengono trattate allo stesso modo.
Sono definiti gli anelli di collegamento tra tutti i moduli del sistema, e nessuna ottimizzazione è fatta a tempo di esecuzione.

1.2.2 – Servizi di amministrazione

Sono servizi usati dai Servizi di Coordinamento per localizzare le sorgenti *utili*, per determinare le loro capacità, e per creare ed interpretare TEMPLATE.
I Template sono strutture dati che descrivono i servizi, le fonti ed i moduli da utilizzare per portare a termine un determinato task.
Sono quindi utilizzati dai sistemi meno ”intelligenti”, e consentono all’operatore di predefinire le azioni da eseguire a seguito di una determinata richiesta, limitando al minimo le possibilità di decisione del sistema.
In alternativa all’uso dei Template, possono essere utilizzate le **Yellow Pages**: si tratta di servizi di directory che memorizzano le informazioni sul contenuto delle varie sorgenti e sul loro stato (attiva, inattiva, occupata).
Consultando queste Yellow Pages, il mediatore è in grado di spedire alla giusta sorgente la richiesta di informazioni, ed eventualmente di rimpiazzare questa sorgente con una equivalente nel caso non fosse disponibile.
Fanno parte di questa categoria di servizi il Browsing: si permette all’utente di ”navigare” attraverso le descrizioni degli schemi delle sorgenti, recuperando informazioni su queste.
Il servizio si basa sulla premessa che queste descrizioni siano fornite esplicitamente tramite un linguaggio dichiarativo leggibile e comprensibile all’utente.
Inoltre tale servizio potrebbe contenere dei servizi di trasformazione del vocabolario e dell’ontologia, come pure di integrazione semantica.
Da citare sono pure i servizi di Iterative Query Formulation: si tratta di sistemi che aiutano l’utente a rilassare o a meglio specificare alcuni vincoli della propria interrogazione per ottenere risposte più precise.

1.2.3 - Servizi di integrazione e trasformazione semantica

Questi servizi supportano le manipolazioni semantiche necessarie per l'integrazionee la trasformazione delle informazioni. Il tipico input per questi servizi è rappresentato da una o più sorgenti di dati, e l'output è la "vista" integrata o trasformata di queste informazioni.
Tra questi servizi si distinguono quelli relativi alla trasformazione degli schemi (ovvero di tutto ciò che va sotto il nome di *metadati*) e quelli relativi alla trasformazione dei dati. Sono spesso indicati come servizi di Mediazione, essendo tipici dei moduli mediatori.
In particolare è possibile osservare:

1. Servizi di **integrazione degli schemi**. Supportano la trasformazione e l'integrazione degli schemi e delle conoscenze derivanti da fonti di dati eterogenee. Fanno parte di essi i servizi di trasformazione dei vocaboli e dell'ontologia, usati per arrivare alla definizione di un'ontologia unica che combini gli aspetti comuni alle singole ontologie usate nelle diverse fonti. Queste operazioni sono molto utili quando devono essere scambiate informazioni derivanti da ambienti differenti, dove molto probabilmente non si condivide un'unica ontologia. Fondamentale, per la fase di creazione dell'insieme dei vocaboli condivisi, è la fase di individuazione dei concetti presenti in diverse fonti, e la riconciliazione delle diversità presenti sia nelle strutture,sia nei significati dei dati.

2. Servizi di **integrazione delle informazioni**. Provvedono alla traduzione dei termini da un contesto all'altro, ovvero dall'ontologia di partenza a quella di destinazione. Possono inoltre occuparsi di uniformare la "granularità" dei dati (come possono essere le discrepanze nelle unità di misura, o le discrepanze temporali).

3. Servizi di **supporto al processo di integrazione**. Sono utilizzati nel momento in cui una query è scomposta in molte subquery, da inviare a fonti differenti, e nel momento in cui i risultati provenienti dalle singole subquery devono essere integrati. Comprendono inoltre tecniche di *caching*, per supportare la materializzazione delle viste (problematica molto comune nei sistemi che vanno sotto il nome di datawarehouse).

1.2.4 - Servizi di wrapping

Sono utilizzati per fare si che le fonti di informazioni aderiscano ad uno standard, che può essere interno o proveniente dal mondo esterno con cui il sistema vuole interfacciarsi.
Si comportano come traduttori dai sistemi locali ai servizi di alto livello dell'integratore e viceversa quando si interroga la sorgente di dati.
In particolare, sono due gli obiettivi che si prefiggono:

1. permettere ai servizi di coordinamento e di mediazione di manipolare in modo uniforme il numero maggiore di sorgenti locali, anche se queste non sono state esplicitamente pensate come facenti parte del sistema di integrazione;

2. essere il più riusabili possibile. Per fare ciò, dovrebbero fornire interfacce che seguano gli standard più diffusi (e tra questi, si potrebbe citare il linguaggio SQL come linguaggio di interrogazione di basi di dati, e CORBA come protocollo di scambio di oggetti).
Questo permetterebbe alle sorgenti estratte da questi wrapper "universali" di essere accedute dal numero maggiore possibile di moduli mediatori.

In pratica, compito di un wrapper è modificare l'interfaccia, i dati ed il comportamento di una sorgente per facilitarne la comunicazione con il mondo esterno.
Il vero obiettivo è quindi standardizzare il processo di wrapping delle sorgenti, permettendo la creazione di una libreria di fonti accessibili; inoltre, il processo stesso di realizzazione di un wrapper dovrebbe essere standardizzato, in modo da poter essere riutilizzato da altre fonti.

1.2.5 - Servizi ausiliari

Aumentano le funzionalità degli altri servizi descritti precedentemente: sono prevalentemente utilizzati dai moduli che agiscono direttamente sulle informazioni.
Vanno dai semplici servizi di monitoraggio del sistema (un utente vuole avere un segnale nel momento in cui avviene un determinato evento in un database, e conseguenti azioni devono essere attuate), ai servizi di propagazione degli aggiornamenti e di ottimizzazione.

1.3 - Il mediatore

Secondo la definizione proposta da Wiederhold ”un mediatore è un modulo software che sfrutta la conoscenza su un certo insieme di dati per creare informazioni per una applicazione di livello superiore Dovrebbe essere piccolo e semplice, così da poter essere amministrato da uno, o al più pochi, esperti.”.
Compiti di un mediatore sono quindi:

- assicurare un servizio stabile, anche nel caso di cambiamento delle risorse;
- amministrare e risolvere le eterogeneità delle diverse fonti;
- integrare le informazioni ricavate da più risorse;
- presentare all’utente le informazioni attraverso un modello scelto dall’utente stesso.

Tra gli obiettivi principali del progetto MOMIS c'è quello di realizzare un Mediatore; infatti l'approccio architetturale scelto sviluppa il sistema su tre livelli:

1. utente: attraverso un'interfaccia grafica l'utente pone delle query su uno schema globale e riceve un'unica risposta, come se stesse interrogando un'unica sorgente di informazioni;

2. mediatore: il mediatore gestisce l'interrogazione dell'utente, combinando, integrando ed eventualmente arricchendo i dati ricevuti dai wrapper, ma usando un modello (e quindi un linguaggio di interrogazione) comune a tutte le fonti;

3. wrapper: ogni wrapper gestisce una singola sorgente, ed ha una duplice funzione: da un lato converte le richieste del mediatore in una forma comprensibile dalla sorgente, dall'altro traduce informazioni estratte dalla sorgente nel modello usato dal mediatore.

Facendo riferimento ai servizi descritti in precedenza, l'architettura del mediatore che si è progettato è riportata in Figura 2.

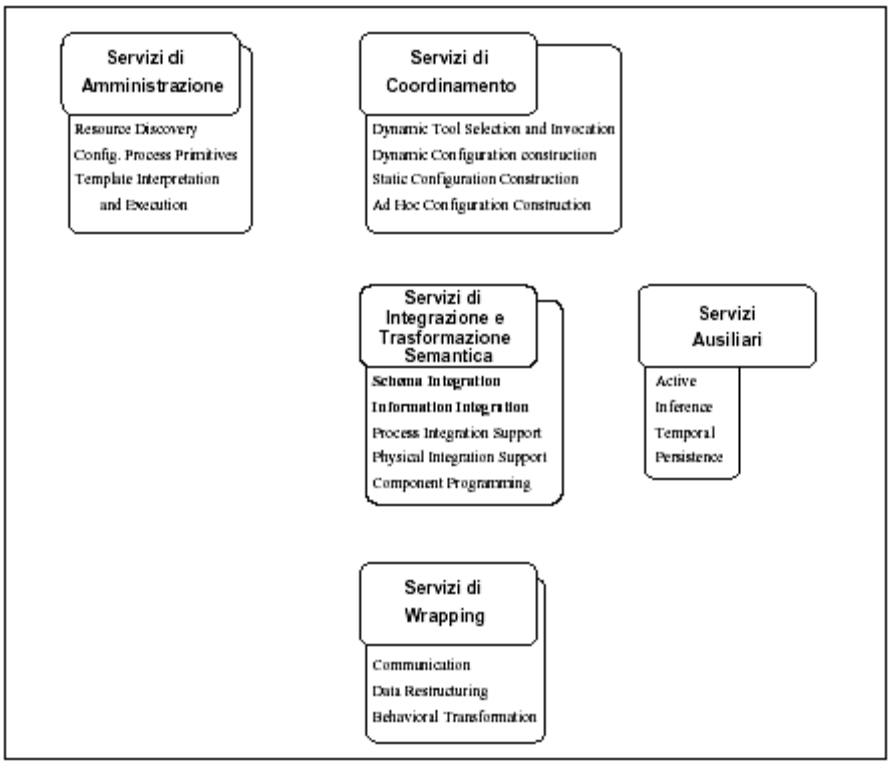

Figura 2

L'impostazione architetturale presentata dimostra che il mediatore in progettazione vuole distaccarsi dall'approccio *strutturale*, cioè sintattico, tuttora dominante tra i sistemi presenti sul mercato. L'approccio strutturale, adottato da sistemi quali TSIMMIS, è caratterizzato dal fatto di usare un self-describing model per rappresentare gli oggetti da integrare, limitando l'uso delle informazioni semantiche alle regole predefinite dall'operatore. In pratica, il sistema non conosce a priori la semantica di un oggetto che va a recuperare da una sorgente (e dunque di questa non possiede alcuno schema descrittivo) bensì è l'oggetto stesso che, attraverso delle etichette, si autodescrive, specificando tutte le volte, per ogni suo singolo campo, il significato che ad esso è associato. Questo approccio porta quindi ad un insieme di vantaggi, tra i quali possiamo identificare:

- la possibilità di integrare in modo completamente trasparente al mediatore basi di dati fortemente eterogenee e magari mutevoli nel tempo: il mediatore non si basa infatti su una descrizione predefinita degli schemi delle sorgenti, bensì sulla descrizione che ogni singolo oggetto fa di se. Oggetti simili provenienti dalla stessa sorgente possono quindi avere strutture differenti, cosa invece non ammessa in un ambiente tradizionale object-oriented;

- per trattare in modo omogeneo dati che descrivono lo stesso concetto, o che hanno concetti in comune, ci si basa sulla definizione manuale di rule, che permettono di identificare i termini (e dunque i concetti) che devono essere condivisi da più oggetti.

Altri progetti seguono invece un approccio definito *semantico*, che è caratterizzato dai seguenti punti:

- il mediatore deve conoscere, per ogni sorgente, lo schema concettuale (metadati);

- informazioni semantiche sono codificate in questi schemi;

- deve essere disponibile un modello comune per descrivere le informazioni da condividere (e dunque per descrivere anche i metadati);

- deve essere possibile un'integrazione (parziale o totale) delle sorgenti di dati.

In questo modo, sfruttando opportunamente le informazioni semantiche che necessariamente ogni schema sottintende, il mediatore può individuare concetti comuni a più sorgenti e relazioni che li legano.

1.4 - Il sistema MOMIS

Il sistema **MOMIS** (**M**ediator Envir**O**nment for **M**ultiple **I**nformation **S**ources), nasce all'interno del progetto **MURST 40%** ***INTERDATA*** (97/98) per rispondere all'esigenza non solo di ambienti software ' *intelligenti* ' in grado di fornire accesso a grosse moli di dati, ma anche capaci mi aumentare la *qualità* delle informazioni ottenibili.
L'obiettivo del progetto è quello di realizzare uno strumento semi-automatico che consenta una reale integrazione di sorgenti distribuite, eterogenee, strutturate e semistrutturate. L'approccio di integrazione adottato è di tipo *semantico* e *virtuale*. Per ' *semantico* ' si intende quanto illustrato nella sezione precedente, mentre con ' *virtuale* ' si intende che la vista integrata delle sorgenti non viene *materializzata* in locale, ma sarà il sistema che, basandosi sull'individuazione delle sorgenti da interrogare, provvederà alla generazione delle subquery da eseguire direttamente sulle sorgenti.
Si opera quindi su di uno schema globale (vista virtuale integrata) che rappresenta le varie sorgenti collegate.
L'adozione di questo approccio è stata dettata da varie motivazioni:

- la presenza di uno schema globale permette all'utente di formulare qualsiasi interrogazione che sia con esso consistente;

- le informazioni semantiche che comprende possono contribuire ad una eventuale ottimizzazione delle interrogazioni;

- l'adozione di una semantica ' *type as a set* ' per gli schemi permette di controllarne la consistenza facendo riferimento alle loro descrizioni;

- la vista virtuale rende il sistema estremamente flessibile, in grado cioè di sopportare frequenti cambiamenti sia nel numero che nel tipo delle sorgenti, ed anche nei loro contenuti (non occorre prevedere onerose politiche di allineamento);

1.4.1 - Il modello dei dati

All'interno del sistema è stato adottato un modello comune dei dati (ODM_I^3).
La base di partenza per la definizione di questo modello è rappresentata dalle raccomandazioni relative alla proposta di standardizzazione per linguaggi di mediazione, risultato del lavoro svolto in ambito I^3. Tali raccomandazioni sottolineano la necessità per un mediatore di poter gestire sorgenti con modelli complessi, come quello ad oggetti, e sorgenti molto più semplici come file di strutture: l'impiego di un formalismo il più possibile completo, e quindi in grado di rappresentare in modo adeguato tutte le possibili situazioni, risulta una possibile soluzione.
Per quanto riguarda il linguaggio di definizione degli schemi si è cercato di cogliere le indicazioni emerse in ambito I^3 discostandosi, nel contempo, il meno possibile dal linguaggio ODL proposto dal gruppo di standardizzazione ODMG-93.
Si è così definito il linguaggio ODL_I^3 come estensione del linguaggio standard ODL in modo da supportare le necessità del nostro mediatore.
Le principali caratteristiche del linguaggio ODL_I^3 sono:

- possibilità di rappresentare sorgenti strutturate (database relazionali, ad oggetti, e file system) e semistrutturate (XML). Ciò significa che tutte le fonti di informazione, indipendentemente dal modello originario, e lo schema globale verranno descritti mediante il modello comune, facendo quindi riferimento al concetto di classe ed aggregazione (sarà poi compito dei Wrapper provvedere alla traduzione in termini del modello originale);

- dichiarazione di regole di integrità (*if then rule*), definite sia sugli schemi locali (e magari da questi ricevute), che riferite allo schema globale, e quindi inserite dal progettista del mediatore;

- per ogni classe, il wrapper può indicare nome e tipo del sorgente di appartenenza;

- per le classi appartenenti ai sorgenti relazionali è possibile definire le chiavi candidate ed eventuali foreign key;

- dichiarazione di regole di mediazione, o *mapping rule*, utilizzate per specificare l'accoppiamento tra i concetti globali e i concetti locali originali;

- utilizzo della *semantica di mondo aperto*, che permette alle classi descritte di cambiare formato (magari aggiungendo attributi agli oggetti) senza necessariamente cambiarne la descrizione (prerogativa, questa, indispensabile per la gestione di sorgenti semistrutturate);

- il linguaggio supporta la definizione di grandezze locali e di grandezze globali;

- traduzione automatica e trasparente all'utente delle descrizioni nella logica descrittiva **OLCD**, con conseguente possibilità di utilizzare comportamenti intelligenti nei controlli di consistenza e nell'ottimizzazione semantica delle interrogazioni;

- introduzione dell'operatore di *unione* (`union`), che permette l'espressione di strutture dati in alternativa nella definizione di una classe;

- introduzione del costruttore *optional* (*), specificato dopo il nome di un attributo per indicare la sua natura opzionale;

- dichiarazione di relazioni terminologiche, che permettono di specificare relazioni di sinonimia (SYN), ipernimia (BT), iponomia (NT) e relazione associativa (RT) tra due tipi.

- dichiarazione di relazioni estensionali: il sistema MOMIS integra gli schemi locali secondo criteri sia intensionali che estensionali. I conflitti intenzionali rappresentano quelle incompatibilità derivanti dall'avere porzioni di schemi sovrapposte, ossia gli stessi aspetti del dominio applicativo rappresentati usando strutture differenti. Ciò che occorre fare è quindi fornire una rappresentazione unificata ed omogenea dei medesimi concetti descritti in sorgenti differenti.
L'integrazione degli schemi non è però l'unico aspetto che occorre gestire per ottenere un'effettiva integrazione di sorgenti eterogenee: è necessario risolvere anche i conflitti derivanti dalla sovrapposizione delle estensioni, cioè dalla presenza, in sorgenti diverse, di informazione relativa alla stessa entità del "mondo reale". Per arrivare ad un'integrazione corretta può non essere sufficiente fare una semplice unione delle estensioni delle classi, in quanto dati relativi alla stessa entità potrebbero essere presenti in più classi.

Questi inconvenienti possono essere risolti soltanto gestendo in modo adeguato le relazioni fra le estensioni. Si introducono, a tale scopo, gli *assiomi estensionali* , i quali descrivono le relazioni insiemistiche esistenti fra le estensioni delle sorgenti. Ogni assioma estensionale definito *vincola* le classi coinvolte ad avere anche un legame intensionale.

Utilizzando questo linguaggio, il wrapper compie la traduzione delle classi da integrare e la fornisce al mediatore: da sottolineare che le descrizioni ricevute rappresentano tutte e sole le classi che una determinata sorgente vuole mettere a disposizione del sistema, e quindi interrogabili. Non è detto che lo schema locale ricevuto dal mediatore rappresenti l'intera sorgente, bensì ne descrive il sottoinsieme di informazioni visibili da un utente del mediatore, esterno quindi alla sorgente stessa. Per quanto riguarda il linguaggio di interrogazione si è adottato OQL_I[3] che adotta la sintassi OQL senza discostarsi dallo standard.
Questo linguaggio richiede un maggiore sforzo dal punto di vista dello sviluppo di moduli per l'interpretazione e la gestione delle interrogazioni (implementando le funzionalità tipiche di un ODBMS), ma risulta altresì estremamente versatile ed espressivo fornendo la possibilità di sfruttare le informazioni rappresentate nello schema globale.
Le maggiori difficoltà implementative sono quindi ampliamente giustificate da una maggiore versatilità e da una migliore facilità d'uso per l'utente finale.

1.4.2 - L'architettura generale di MOMIS

In Figura 3 è illustrata dettagliatamente l'architettura generale di MOMIS. Lo schema evidenzia l'organizzazione a tre livelli utilizzata.

- **Livello Mediatore.** Il nucleo centrale del sistema è costituito dal **Mediatore** (o *Mediator*) che presiede all'esecuzione di diverse operazioni.

Per meglio comprendere i suoi compiti, è opportuno a questo punto illustrare le due fasi ben distinte in cui si articola la sua attività.

La prima funzionalià del Mediatore è quella di generazione dello Schema Globale.

In questa fase il modulo del Mediatore denominato **Global Schema Builder** riceve in input le descrizioni degli schemi locali delle sorgenti espressi in ODL_I^3 forniti ognuno dal relativo wrapper. A questo punto (utilizzando strumenti di ausilio quali ODB-Tools Engine, WordNet, ARTEMIS) il Global Schema Builder è in grado di costruire la vista virtuale integrata (**Global Schema**) utilizzando tecniche di clustering e di Intelligenza Artificiale.

In questa fase è prevista anche l'interazione con il progettista il quale, oltre ad inserire le regole di mapping, interviene nei processi che non possono essere svolti automaticamente dal sistema (come ad es. l'assegnamento dei nomi alle classi globali, la modifica di relazioni lessicali, …).

Oltre allo Schema Globale, altri "prodotti" di questa fase sono le **Mapping Table**, tabelle che descrivono il modo in cui gli attributi globali presenti nello schema generato hanno corrispondenza (*mappano*) nei vari attributi locali presenti negli schemi delle sorgenti.

Un secondo importante modulo che compone la struttura del mediatore è il **Query Manager** che presiede alla fase di query processing.

In questa fase la singola query posta in OQL_I^3 dall'utente sullo Schema Globale (che chiameremo *Global Query*) sarà rielaborata in più *Local Query* (anche esse espresse in OQL_I^3) da inviare alle varie sorgenti, o meglio ai wrapper predisposti alla loro traduzione. Questa traduzione avviene in maniera automatica da parte del Query Manager utilizzando tecniche di logica descrittiva.

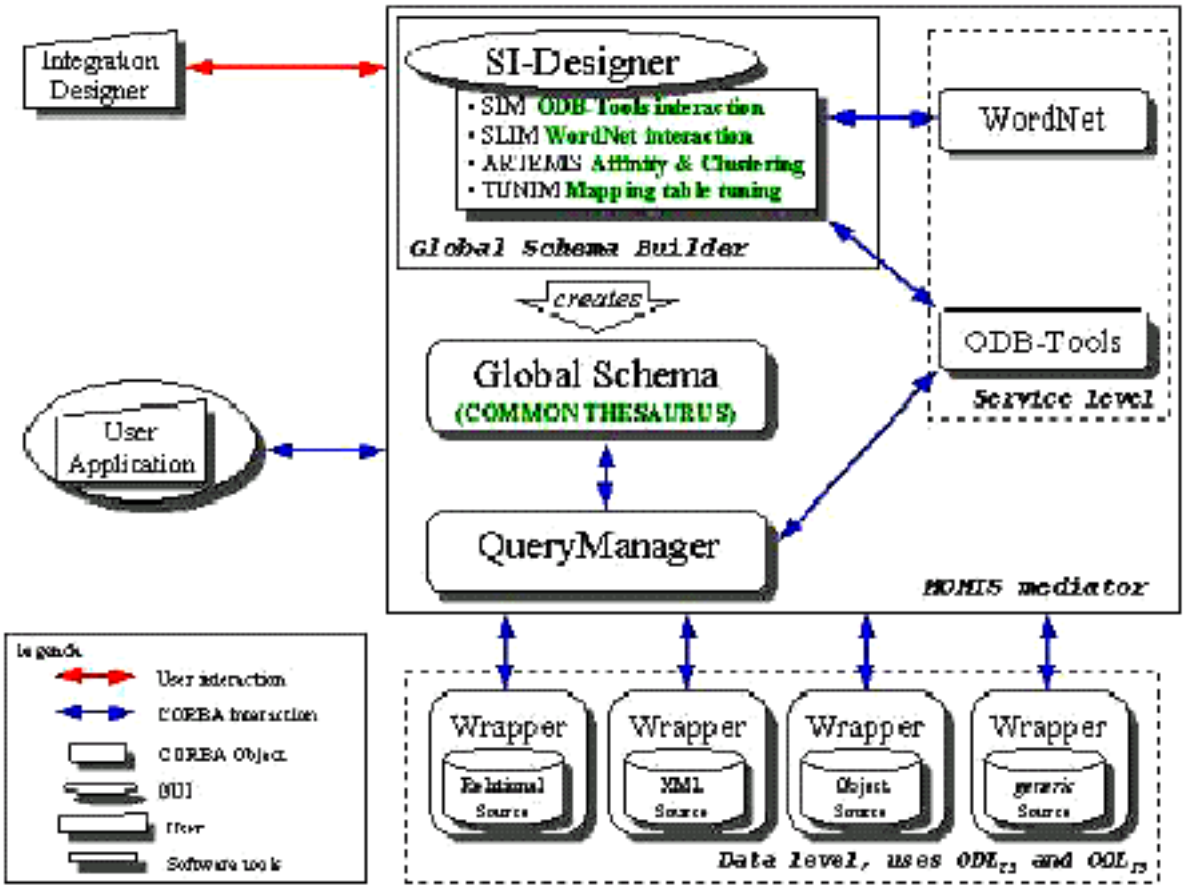

Figura 3

L'ultimo modulo del Mediatore è rappresentato dall'**Extensional Hierarchy Builder** il quale si occupa della generazione della Conoscenza Estensionale (Gerarchie Estensionali e Base Extension) necessaria per ottimizzare le interrogazioni.
Il modulo software utilizzato per l'integrazione si chiama SI-Designer, un tool semiautomatico che realizza le funzionalità sopra descritte.

- **Livello Wrapper.** I **Wrapper** costituiscono l'interfaccia tra il mediatore e le sorgenti; ad ogni sorgente corrisponde un determinato wrapper ed ogni wrapper deve essere disegnato esclusivamente per la sorgente (o la tipologia di sorgenti) che sovrintenderà.

Ogni wrapper ha due compiti ben precisi:

- in fase di integrazione deve fornire al Global Schema Builder la descrizione della sorgente da integrare;

- in fase di query processing deve tradurre la local query (rivolta alla ”sua” sorgente) che gli è stata indirizzata dal Query Manager (e che è espressa in OQL_I^3) nel linguaggio di interrogazione specifico della sorgente per la quale è stato progettato.

Collegate ai wrapper sono quindi le **Sorgenti**, per questo a volte si parla anche di quattro livelli. Esse sono le fonti da integrare, possono essere DataBase (ad oggetti o relazionali) o parti di essi, file system ed anche sorgenti semistrutturate.

- **Livello Utente.** L’utilizzatore del sistema dovrà potere interrogare lo schema globale. L’accesso ai dati si propone del tutto simile a come avvengono le usuali interrogazioni di un DataBase tradizionale: le sorgenti ed il modo in cui i dati vengono recuperati risultano all’utente del tutto trasparenti, in quanto è il sistema ad occuparsi di tutte le operazioni necessarie per reperire le informazioni e combinare le risposte in un’unica risposta corretta, completa e non ridondante.

Vediamo una rapida descrizione dei tool di ausilio al mediatore precedentemente citati:

- *ODB-Tools* è uno strumento software sviluppato presso il dipartimento di Ingegneria dell’Università di Modena e Reggio Emilia. Esso si occupa della validazione di schemi e dell’ottimizzazione semantica di interrogazioni rivolte a Basi di Dati orientate agli Oggetti (OODB).

Facciamo riferimento alla Figura 4 per una breve spiegazione. ODB-Designer si occupa della validazione di schemi: si può inserire la descrizione di uno schema di DataBase (in ODL) ed il sistema realizzerà automaticamente la sua validazione e la sua riclassificazione; vale a dire che si occuperà di verificare che non esistano classi incoerenti (cioè che non possano essere popolate da nessun oggetto) e di determinare eventuali relazioni di specializzazione non esplicitate dallo schema stesso. ODB-Qoptimizer si occupa invece dell'ottimizzazione semantica delle interrogazioni: se si inserisce una query (in OQL) posta su di un determinato schema, questa viene automaticamente riformulata in una equivalente, ma più efficiente, sfruttando l'espansione semantica ed i vincoli di integrità.

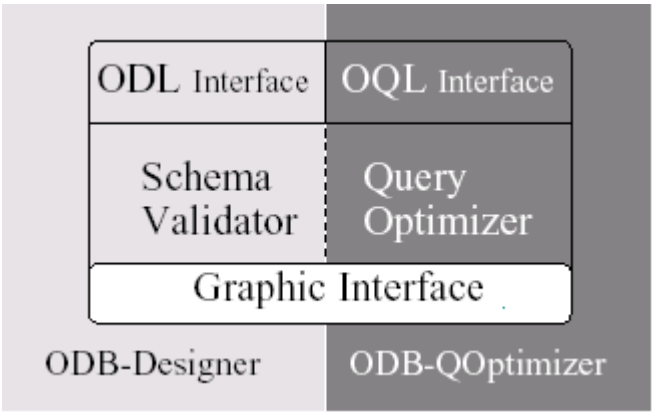

Figura 4

- *WordNet* è un DataBase lessicale in lingua inglese capace di individuare relazioni semantiche fra termini; cioè, dato un insieme di termini, WordNet è in grado di identificare l'insieme di relazioni lessicali che li legano.

- *ARTEMIS* riceve in ingresso il *thesaurus*, cioè l'insieme delle relazioni terminologiche (lessicali e strutturali) precedentemente generate, e sulla base di queste assegna ad ogni classe coinvolta nelle relazioni un coefficiente numerico indicante il suo grado di affinità. Questi coefficienti verranno utilizzati per raggruppare le classi locali in modo tale che ogni gruppo (*cluster*) comprenda solo classi con coefficienti di affinità simili.

1.5 – Il sistema MIKS

Quando l'orizzonte delle applicazioni è particolarmente vasto, la ricerca e l'elaborazione di dati può coinvolgere grandi quantità di informazioni distribuite e spesso eterogenee ed un numero di utenti elevato.
In queste situazioni un sistema basato su architetture client-server può non rappresentare la soluzione più adatta ed efficiente.
Si è pensato quindi di integrare all'interno del sistema MOMIS l'utilizzo di agenti intelligenti per migliorarne l'efficienza.
Il sistema proposto, mostrato in Figura 5 è chiamato MIKS (**M**ediator Agent for **I**ntegration of **K**nowledge **S**ource) ed è basato sull'utilizzo di sistemi multi-agente (MASs).

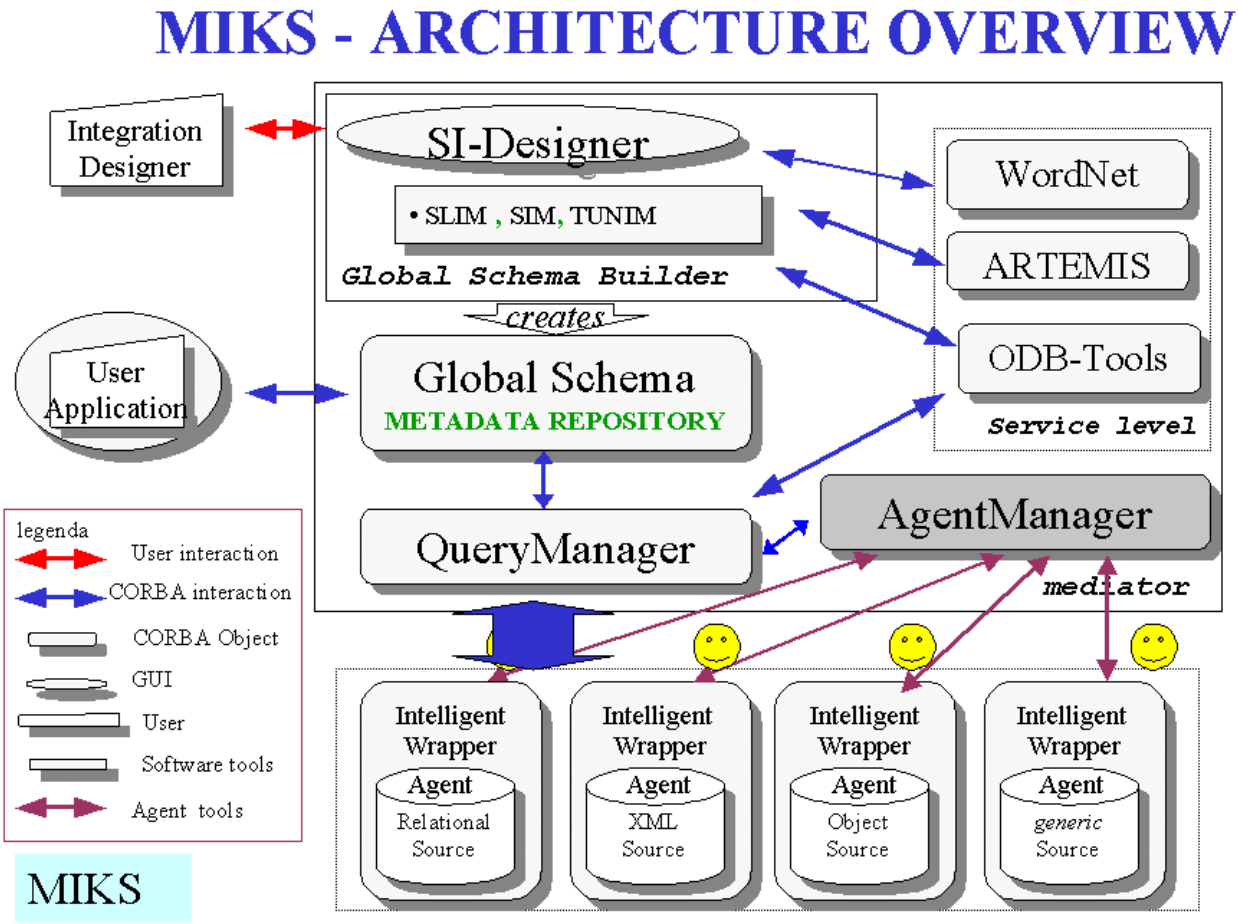

Figura 5

Il modello multi-agente si propone di progettare sistemi in cui gli agenti si comportano in modo autonomo, cooperando ed interagendo fra di loro per svolgere mansioni complesse.
L'utilizzo di agenti all'interno del sistema MIKS , oltre a rendere più efficiente i processi di integrazione e la fase di interrogazione può essere sfruttato per altre funzionalità fra cui la ricerca e l'acquisizione di nuove sorgenti dati.
Spesso si può presentare la necessità di integrare nuove sorgenti di informazioni per migliorare la quantità e la qualità dei dati a disposizione.
In questo caso il sistema dovrà occuparsi di ricercare le nuove potenziali sorgenti dati in un determinato dominio (ad esempio Internet) e gestire la fase di analisi ed immagazzinamento dati.
Per far fronte a queste problematiche può essere utile lo sfruttamento delle caratteristiche degli agenti mobili.
Il sistema infatti, potrebbe delegare ad un certo numero di agenti, il compito di ricercare nuove fonti di informazione specificando alcuni parametri che permettano di individuare dati effettivamente utili alle necessità del sistema.
Altri agenti potrebbero occuparsi dell'acquisizione e dell'archiviazione dei dati ritenuti interessanti.
Grazie alle loro caratteristiche , tutti gli agenti coinvolti anche in diverse mansioni , potrebbero comunicare e cooperare.
In questo modo un agente potrebbe decidere di modificare il proprio comportamento , interrompendo o variando l'attività svolta, in modo da aumentare l'efficienza della ricerca.
Nel capitolo 2 verrà affrontato lo studio della piattaforma di agenti JADE proposta dall'Università di Parma.
L'analisi si propone di trattare le caratteristiche di agenti intelligenti per sfruttarne le proprietà in una futura implementazione all'interno del sistema MIKS dedicata alla ricerca e all'integrazione di nuove sorgenti dati.

2 - JADE - Jade Agent Development Framework

2.1 - Introduzione

Jade e' una struttura di sviluppo software mirata alla gestione di applicazioni e sistemi multi-agente conformi agli standard FIPA.
Jade offre una piattaforma virtuale di agenti che può essere distribuita su macchine remote.
Il modello standard descritto dalle specifiche FIPA e' rappresentato nella Figura 6.

L' **Agent Management System** (**AMS**) e' l'agente che si occupa della gestione della piattaforma .
All'interno di ogni piattaforma può esistere un unico AMS.
L ' AMS gestisce il ' ciclo di vita ' di ogni agente mantenendo una lista dei loro identificatori (**AID**) e degli stati in cui si trovano.
Attraverso la gestione di alcuni oggetti descritti nel paragrafo 2.1, mantiene una versione aggiornata della situazione della piattaforma.
Inoltre viene coinvolto al momento dell'esecuzione di qualsiasi operazione legate alla mobilità.

Il **Director Facilitator** (**DF**) e' un'agente che fornisce , attraverso un'interfaccia grafica , un servizio di pagine gialle per agevolare la gestione di agenti.
Tramite questo servizio aggiuntivo è possibile definire comunità di agenti con determinate caratteristiche attraverso le quali facilitare eventuali operazioni di ricerca e catalogazione.

Il **Message Transport Protocol** (**MTP**) costituisce il software che gestisce lo scambio di messaggi all'interno della piattaforma e di quelli provenienti o diretti da /a piattaforme remote.
Al momento della creazione di una piattaforma, viene immediatamente creato un contenitore di agenti detto Main-Container, all'interno del quale vengono attivati automaticamente l' AMS , un DF di default e il registro RMI. Sempre al momento della creazione della piattaforma viene settato un MTP di default per permettere lo scambio di messaggi.

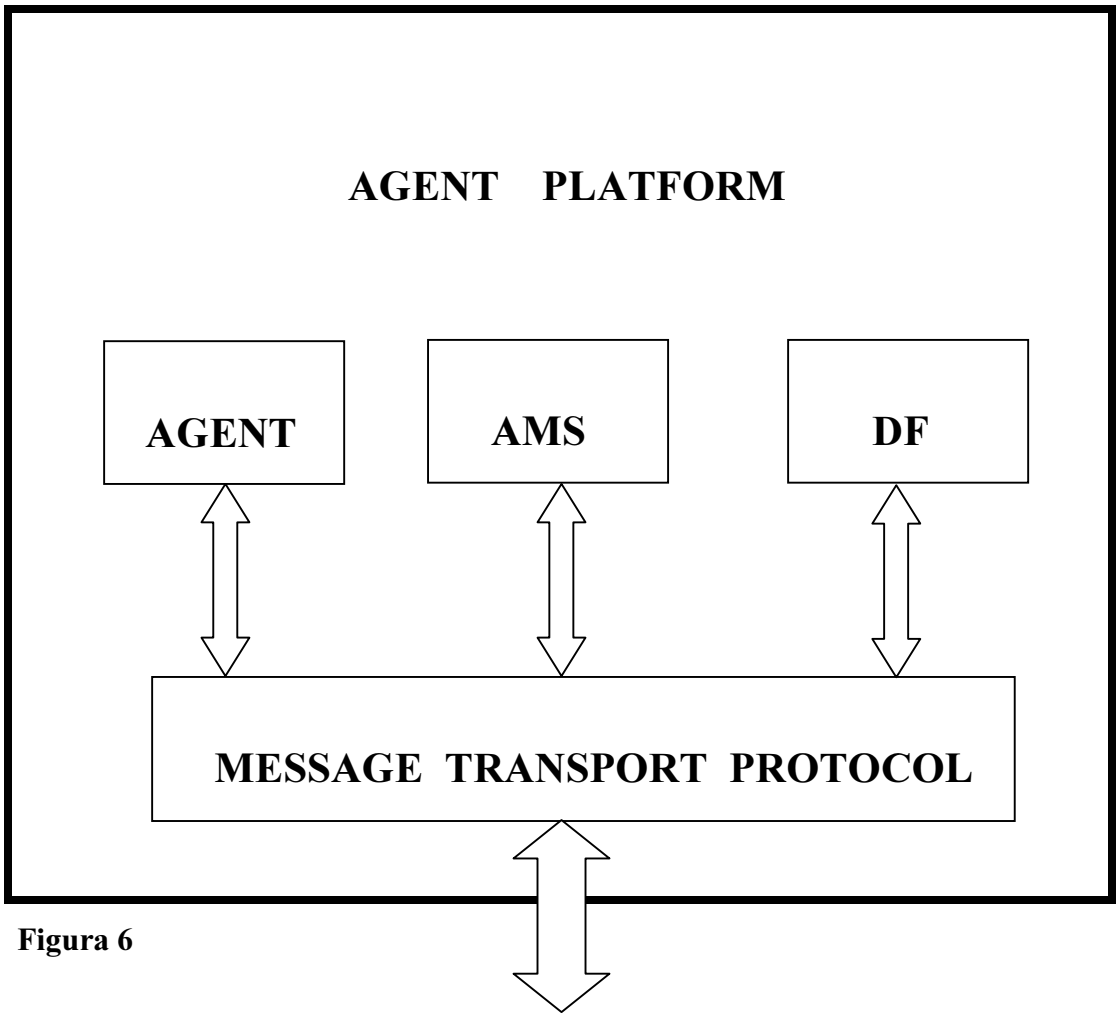

Figura 6

2.2 - Agent Management System (AMS)

La classe ‘ Agent ’ rappresenta la classe base per la definizione di agenti , quindi , dal punto di vista del programmatore , un agente JADE e’ semplicemente un’istanza che estende questa classe.
Questo implica l’eredita’ di caratteristiche necessarie per le interazioni di base con la piattaforma su cui risiede (registrazione , configurazione , gestione remota , ecc…) e di un insieme di metodi base che possono essere utilizzati per implementare le azioni che l’agente dovrà eseguire (spedire / ricevere messaggi , usare protocolli di interazione standard , ecc…).
Il modello di elaborazione di un agente e’ multitask , ed ogni operazione viene eseguita concorrenzialmente.
Uno scheduler , interno alla classe ‘ Agent ’ e nascosto al programmatore , gestisce automaticamente l’esecuzione delle operazioni richieste dall’agente.
Ad ogni agente e’ associato uno stato che può variare istante per istante fra quelli specificati dallo standard FIPA ed e’ rappresentato da alcune costanti definite nella classe ‘ Agent ’.
Il percorso da uno stato all’altro compiuto dal momento della creazione e’ definito ‘ Ciclo di vita ’ ed è rappresentato in Figura 7.
Gli stati definiti dallo standard FIPA sono i seguenti:

- AP_INITIATED : l’agente e’ stato creato ma non e’ ancora stato registrato dall’ AMS. Non ha nome ne indirizzo e non può comunicare con altri agenti.
- AP_ACTIVE : l’agente e’ stato registrato dall’AMS, ha nome e indirizzo regolari e può accedere ai vari servizi.
- AP_SUSPENDED : l’agente e’ bloccato. Il suo thread interno e’ sospeso e nessuna delle operazione richieste può essere eseguita.
- AP_WAITING : l’agente e’ sospeso in attesa di un evento (normalmente l’arrivo di un messaggio).
- AP_DELETED : l’agente e’ definitivamente morto e non e’ più registrato nell’AMS.
- AP_TRANSIT : un agente mobile entra in questo stato mentre sta migrando verso una nuova locazione. Il sistema continua a memorizzare eventuali messaggi che verranno poi spediti alla nuova locazione.

- AP_COPY : stato usato internamente da JADE mentre si sta effettuando la clonazione dell'agente.

- AP_GONE : stato utilizzato internamente da JADE quando un agente mobile ha effettuato la migrazione e il suo stato e' stabile.

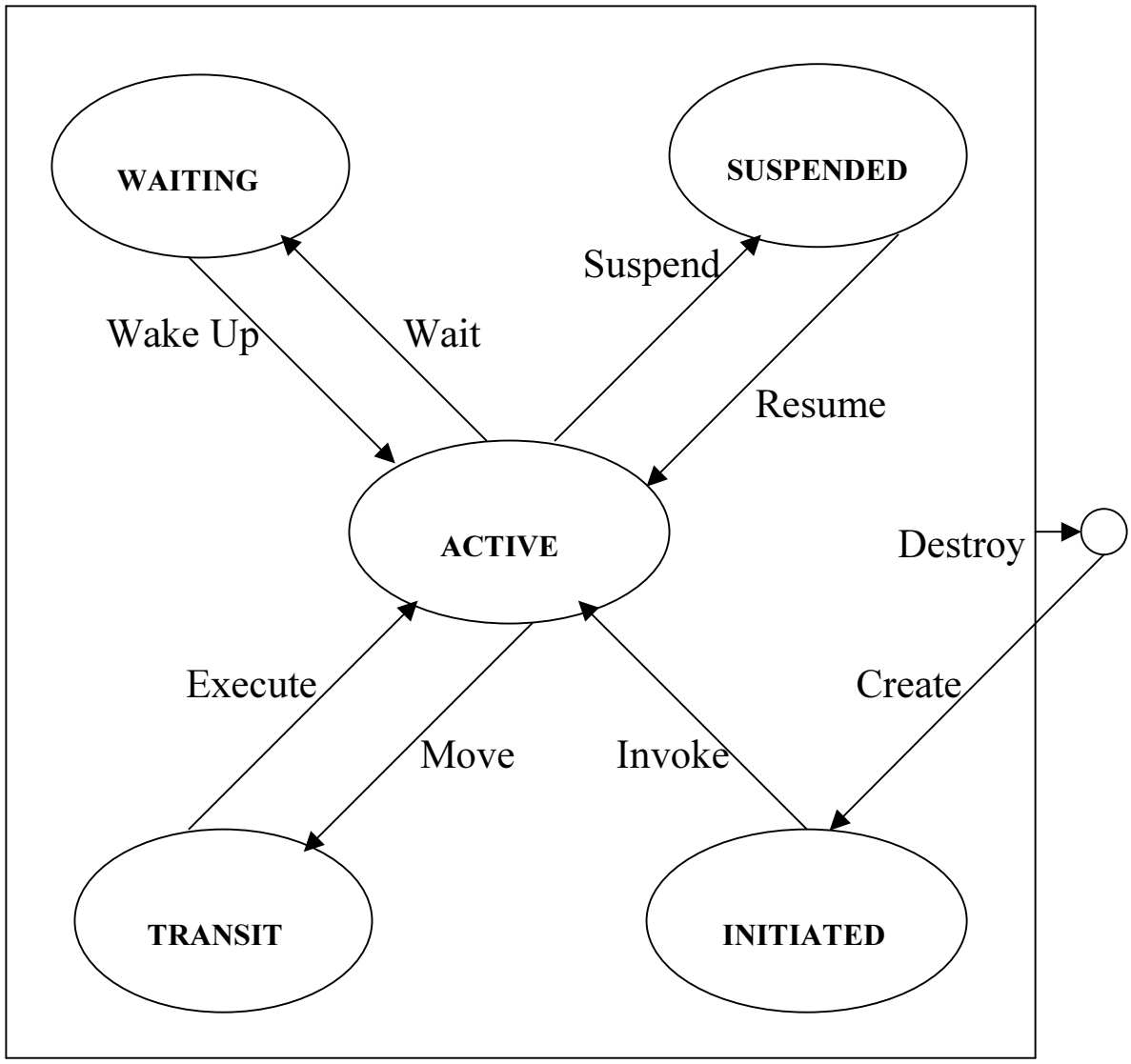

Figura 7

La classe ' Agent ' fornisce metodi pubblici per eseguire transizioni fra i vari stati.

Al momento della creazione di un agente , l'AMS gli associa un identificatore (**AID**) conforme allo standard FIPA.

L’ AID contiene un nome globale che lo identifica univocamente all’interno della piattaforma costituito dalla concatenazione di :

- nome locale attribuito dal programmatore
- simbolo ‘@’
- identificatore della piattaforma su cui e’ stato creato
- simbolo ‘:’ seguito dalla porta occupata dal registro RMI
- ‘/JADE’

Inoltre e’ possibile includere nell’ AID un insieme di indirizzi tra i quali, all’atto della creazione , sono comunque inclusi gli indirizzi già attivati sulla piattaforma su cui risiede.
Dopo essere stato associato all’ AID ,l’agente viene registrato dall’ AMS e posto nello stato AP_ACTIVE. Infine viene eseguito il metodo SETUP( ) , dove vengono definite le azioni che l’agente dovrà compiere.

2.2.1 - AMS-AGENT-DESCRIPTION

L' AMS gestisce un oggetto , chiamato ams-agent-description , in cui sono memorizzate le informazioni relative ad ogni agente.
Quindi , se un agente volesse conoscere la lista degli agenti presenti sulla piattaforma , dovrebbe richiedere all' AMS , tramite un messaggio ACL , l' ams-agent-description utilizzando il seguente codice :

```
registerLanguage(SL0Codec.NAME, new SL0Codec());
registerOntology(FIPAAgentManagementOntology.NAME,FIPAAgentManagementOntology.instance());
ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
msg.setSender(getAID());
msg.clearAllReceiver();
msg.addReceiver(getAMS());
msg.setLanguage(SL0Codec.NAME);
msg.setOntology(FIPAAgentManagementOntology.NAME);
msg.setProtocol("fipa-request");
SearchConstraints constraints = new SearchConstraints();
Search s = new Search();
AMSAgentDescription amsd = new AMSAgentDescription(),
s.set_0(amsd);
s.set_1(constraints);
try {
                Action action = new Action();
                action.setActor(getAMS());
                action.setAction(s);
                List tuple = new ArrayList();
                tuple.add(action);
                fillMsgContent(msg,tuple);
        }
 catch ( FIPAException fe ) {  fe.printStackTrace();  }
send(msg);
```

Il corrispondente messaggio ACL inviato all' AMS e' il seguente :

```
(REQUEST
 :sender  ( agent-identifier  :name AgentX@sparc20:2001/JADE)
 :receiver  (set ( agent-identifier  :name ams@sparc20:2001/JADE ) )
 :content  "((action (agent-identifier
                                        :name ams@sparc20:2001/JADE
                                        :addresses (sequence )
                                        :resolvers (sequence ) )
             (search (ams-agent-description ) (search-constraints ) ) ) )"
 :language  FIPA-SL0
 :ontology  FIPA-Agent-Management
 :protocol  fipa-request
)
```

L' AMS rispondera' con il seguente messaggio :

```
(INFORM
 :sender  ( agent-identifier
                        :name ams@sparc20:2001/JADE
                        :addresses (sequence
                                        http://sparc20:2002/acc
                                        http://sparc20:2003/acc
                                        http://sparc20:2004/acc ))
 :receiver (set(agentidentifier :name AgentX@sparc20:2001/JADE))
 :content  "((result (action (agent-identifier
```

```
                    :name ams@sparc20:2001/JADE
                    :addresses (sequence )
                    :resolvers (sequence ) )
(search(ams-agent-description )(search-constraints)))
(set
    (ams-agent-description
     :name (agent-identifier
                    :name RMA@sparc20:2001/JADE
                    :addresses (sequence
                                http://sparc20:2002/acc
                                http://sparc20:2003/acc
                                http://sparc20:2004/acc)
                    :resolvers (sequence ) )
     :ownership JADE
     :state active )
    (ams-agent-description
     :name (agent-identifier
                    :name df@sparc20:2001/JADE
                    :addresses (sequence
                                http://sparc20:2002/acc
                                http://sparc20:2003/acc
                                http://sparc20:2004/acc)
                    :resolvers (sequence) )
     :ownership JADE
     :state active )
    (ams-agent-description
     :name (agent-identifier
                    :name ams@sparc20:2001/JADE
                    :addresses (sequence
                                http://sparc20:2002/acc
                                http://sparc20:2003/acc
                                http://sparc20:2004/acc)
                    :resolvers (sequence ) )
     :ownership JADE
     :state active)
    (ams-agent-description
     :name (agent-identifier
                    :name AgentX@sparc20:2001/JADE
```

```
                    :addresses (sequence
                                http://sparc20:2002/acc
                                http://sparc20:2003/acc
                                http://sparc20:2004/acc)
                    :resolvers (sequence) )
                :ownership JADE
                :state active )
            (ams-agent-description
                :name (agent-identifier
                    :name due@sparc20:2001/JADE
                    :addresses (sequence
                                http://sparc20:2002/acc
                                http://sparc20:2003/acc
                                http://sparc20:2004/acc)
                    :resolvers (sequence ) )
                :ownership JADE
                :state active )
            (ams-agent-description
                :name (agent-identifier
                    :name uno@sparc20:2001/JADE
                    :addresses (sequence
                                http://sparc20:2002/acc
                                http://sparc20:2003/acc
                                http://sparc20:2004/acc)
                    :resolvers (sequence) )
                :ownership JADE
                :state active )
        ) ) )"
 :reply-with  AgentX@sparc20:2001/JADE1020846308853
 :language  FIPA-SL0
 :ontology  FIPA-Agent-Management
 :protocol  fipa-request
)
```

L’ AMS informa quindi che sulla piattaforma risiedono sei agenti :

RMA@sparc20:2001/JADE
df@sparc20:2001/JADE
ams@sparc20:2001/JADE
AgentX@sparc20:2001/JADE
due@sparc20:2001/JADE
uno@sparc20:2001/JADE

Inoltre per ogni agente vengono visualizzati l’ AID , l’ ownership e lo stato in cui si trova. Se un agente volesse conoscere la lista degli agenti presenti su una piattaforma remota , dovrebbe interrogare con un messaggio analogo l’ AMS remoto , specificando il suo nome globale e almeno uno dei suoi indirizzi attivi.

2.3 - Message Transport Protocol (MTP)

Le specifiche FIPA 2000 propongono vari MTP attraverso i quali messaggi ACL possono essere scambiati.
JADE contiene una struttura su cui possono essere attivati diversi MTP in modo flessibile.
Quindi per ogni contenitore di agenti possono essere attivati diversi MTP tra i quali l'amministratore della piattaforma potrà scegliere il protocollo preferito.
Quando su un contenitore viene attivato un nuovo MTP , alla lista degli indirizzi presenti nel profilo della piattaforma viene aggiunto il nuovo indirizzo.
Contemporaneamente viene aggiornato l' ams-agent-description , modificando l'elenco degli MTP contenuti al suo interno.

2.3.1 - Creazione di un contenitore di agenti e settaggio di un relativo http MTP

Il seguente codice permette , ad un agente , la creazione di un contenitore sulla piattaforma in cui risiede e il settaggio di un HTTP MTP riferito all'indirizzo http://sparc20:2003/acc .

```
String MTPS = new String("MTPS");
Runtime rt = Runtime.instance();
ProfileImpl vai = new ProfileImpl(null,2001,null);
vai.setParameter(Profile.MAIN,"false");
Specifier tra = new Specifier();
tra.setClassName("jamr.jademtp.http.MessageTransportProtocol");
String host_addr1 = " http://sparc20:2003/acc ";
tra.setArgs(new Object[] {host_addr1});
List li = new ArrayList(1);
li.add(tra);
vai.setSpecifiers(Profile.MTPS,li);
AgentContainer cont = rt.createAgentContainer(vai);
```

Automaticamente sullo standard output verrà visualizzato il seguente messaggio :

```
L'agente  :  uno@sparc20:2001/JADE  sta aprendo il container  :
(Profile {main=false, main-host=sparc20, platform-id=sparc20:2001/JADE,
main-proto=rmi, main-port=2001, mtps=jade.util.leap.ArrayList@366573})

    This is JADE 2.4 - 2001/09/25 10:47:18
   downloaded in Open Source, under LGPL restrictions,
   at http://jade.cselt.it/

http://sparc20:2003/acc
Agent container Container-1@JADE-IMTP://sparc20 is ready.
```

Con questo messaggio l' AMS informa che è stato creato , dall'agente

' uno@sparc20:2001/JADE ' , un contenitore il cui nome globale è :

Container-1@JADE-IMTP://sparc20

Inoltre viene mostrato il profilo del contenitore e l'indirizzo http attivato.
Contemporaneamente l' indirizzo viene memorizzato all'interno di un file chiamato :

MTP - < Nome contenitore >.txt

2.3.2 - AP-DESCRIPTION

Se un agente desidera conoscere l'elenco degli MTP attivi sulla piattaforma può richiedere all' AMS, tramite un messaggio ACL , l' ap-description (agent platform description) che contiene le informazioni relative alla piattaforma.
Il messaggio dovrà essere costruito nel seguente modo:

```
registerLanguage(SL0Codec.NAME, new SL0Codec());
registerOntology(FIPAAgentManagementOntology.NAME,FIPAAgentManagementOntology.instance());
ACLMessage gig = new ACLMessage(ACLMessage.REQUEST);
gig.setSender(getAID());
gig.clearAllReceiver();
gig.addReceiver(getAMS());
gig.setLanguage(SL0Codec.NAME);
gig.setOntology(FIPAAgentManagementOntology.NAME);
gig.setProtocol("fipa-request");
try {
      Action kok = new Action();
      kok.setActor(getAMS());
      kok.setAction(new jade.domain.FIPAAgentManagement.GetDescription());
      List pope = new ArrayList();
      pope.add(kok);
      fillMsgContent(gig,pope);
    }  catch (FIPAException fe) {  fe.printStackTrace(); }
send(gig);
```

IL messaggio ACL inviato avrà la seguente forma:

```
(REQUEST
 :sender  ( agent-identifier              :name due@sparc20:2001/JADE )
 :receiver  (set ( agent-identifier       :name ams@sparc20:2001/JADE ) )
 :content  "((action (agent-identifier
                                           :name ams@sparc20:2001/JADE
                                          :addresses (sequence )
                                          :resolvers (sequence ) )
              (get-description ) ) )"
 :language  FIPA-SL0
 :ontology  FIPA-Agent-Management
 :protocol  fipa-request
)
```

L' AMS risponderà con il messaggio seguente:

```
(INFORM
 :sender  ( agent-identifier
                                :name ams@sparc20:2001/JADE
                                :addresses ( sequence
                                                    http://sparc20:2002/acc
                                                    http://sparc20:2003/acc
                                                    http://sparc20:2004/acc ))
 :receiver  (set ( agent-identifier  :name due@sparc20:2001/JADE) )
 :content  "((result (action (agent-identifier
                               :name ams@sparc20:2001/JADE
                               :addresses (sequence )
                               :resolvers (sequence ) )
```

```
(get-description ) )
(set
   ( ap-description
  :name \"\\"sparc20:2001/JADE\\"\"
  :dynamic false :mobility false
  :transport-profile (ap-transport-description
                                  :available mtps
                                   (set
                                (mtp-description
                                 :mtp-name http
                                 :addresses (sequenze
                                                 http://sparc20:2002/acc ))
                                (mtp-description
                                 :mtp-name http
                                 :addresses (sequence
                                                          http://sparc20:2003/acc ))
                                (mtp-description
                                 :mtp-name http
                                 :addresses (sequence
                                                 http://sparc20:2004/acc ))
                           ) ) ) ) ) )"
 :reply-with  due@sparc20:2001/JADE1020845636940
 :language  FIPA-SL0
 :ontology  FIPA-Agent-Management
 :protocol  fipa-request
)
```

Le informazioni contenute , indicano che sulla piattaforma ‘ sparc20:2001/JADE ‘ sono stati attivati tre HTTP MTP con indirizzi :

http://sparc20:2002/acc

http://sparc20:2003/acc

http://sparc20:2004/acc

2.3.3 - Attivazione di un HTTP-MTP da riga di comando

Al momento della creazione di una piattaforma , se non vengono fornite opzioni relative all'attivazione di MTP, l'AMS attiva sul Main-Container un IIOP MTP per permettere comunque un eventuale richiesta di scambio di messaggi e stampa il relativo indirizzo IOR sullo standard output.
Se al contrario si desidera attivare un MTP diverso da quello di default, è necessario specificarne la classe relativa da riga di comando.
Per attivare un HTTP MTP , ad esempio , si procede nel seguente modo :

```
java jade.Boot -port 2001 -mtp
'jamr.jademtp.http.MessageTransportProtocol(http://sparc20:2002/acc)'
```

Il risultato di questo comando sarà la creazione di una piattaforma il cui registro RMI occuperà la porta 2001 , e l'attivazione di un HTTP MTP con indirizzo :

http://sparc20:2002/acc

Inoltre l' ap-description verrà memorizzato all' interno del file :

APDescription.txt

Il contenuto di questo file viene aggiornato ogni volta che sono eseguite operazioni che modificano l'ap-description della piattaforma.
La struttura degli indirizzi HTTP MTP prevede obbligatoriamente il nome della macchina su cui risiede la piattaforma , una porta che deve essere necessariamente libera e deve terminare con la stringa ' acc '.
Sullo standard output verrà visualizzato il seguente messaggio :

```
This is JADE 2.4 - 2001/09/25 10:47:18
   downloaded in Open Source, under LGPL restrictions,
   at http://jade.cselt.it/
http://sparc20:2002/acc
Agent container Main-Container@JADE-IMTP://sparc20 is ready.
```

Oltre ad informazioni relative alla versione di JADE utilizzata e all' indirizzo attivato , viene visualizzato l'indirizzo del Main-Container attivato automaticamente.
Un agente può quindi essere contattato attraverso i vari indirizzi attivati sulla piattaforma , ma nel caso fosse necessario , può attivare un set di propri MTP che non apparterrebbe all'intera piattaforma ma solo a se stesso.
Per supportare questa duplicità la classe ' Agent ' mette a disposizione il metodo **getAID( )** che fornisce l' identificatore dell'agente.
Quando un agente viene creato nel suo AID è inclusa l'intera lista degli indirizzi già disponibili sulla piattaforma.
Ogni agente però , grazie a metodi pubblici , può eliminare o aggiungere MTP e decidere quindi un suo sottoinsieme preferito di indirizzi tramite cui scambiare messaggi.
JADE fornisce direttamente classi che implementano IIOP MTP , ORBacus MTP e HTTP MTP ma lascia anche la possibilità di creare un MTP personalizzato. Tutto ciò che e' richiesto e' l'implementazione di una coppia di interfacce Java definite nel package ' **jade.mtp** '.
L'interfaccia MTP modella un canale bidirezionale attraverso cui possono essere spediti o ricevuti messaggi ACL.

2.4 – Gestione della mobilità di agenti

Nella versione JADE 2.4 , è ammessa e gestita solo mobilità all'interno della stessa piattaforma, quindi un agente è libero di navigare attraverso diversi contenitori , ma è confinato all'interno della singola piattaforma.
Lo spostamento verso nuove locazioni , e la clonazione , sono considerate transizioni di stato nel ciclo di vita dell'agente , quindi , come le altre operazioni che agiscono sullo stato , possono essere eseguite direttamente dall'agente oppure il compito può essere delegato all' AMS tramite un opportuna richiesta contenuta in un messaggio ACL.
JADE fornisce una particolare ontologia , chiamata **JADE-MOBILITY-ONTOLOGY** , tramite la quale vengono resi disponibili concetti e azioni necessarie alla gestione della mobilità. Fra i concetti di particolare importanza, si trovano :

- ***Mobile-agent-description*** : contiene il nome globale dell'agente e la locazione verso cui si muoverà ed altre informazioni opzionali.

Slot Name	**Slot Type**	**Mandatory / Optional**
Name	AID	Mandatory
Destination	Location	Mandatory
Agent-profile	Mobile-agent-profile	Optional
Agent-version	String	Optional
Signature	String	Optional

- ***Location*** : oggetto che descrive la locazione di destinazione.

Slot Name	Slot Type	Mandatory / Optional
Name	String	Mandatory
Protocol	String	Mandatory

Oltre ai concetti appena descritti , **JADE-MOBILITY-ONTOLOGY** mette a disposizione quattro azioni utili a gestire la mobilità di agenti.

- move – agent : azione che permette lo spostamento di un'agente da una locazione all'altra. Utilizza un singolo parametro del tipo Mobile-agent-description in cui sarà specificato il nome dell'agente che si desidera spostare e la locazione di destinazione.

- Clone-agent : esegue una copia dell'agente. Sono necessari due parametri : il primo del tipo Mobile-agent-description che conterrà il nome dell'agente clonato e la locazione di destinazione, ed il secondo di tipo String , che rappresenterà il nome del clone.

- Where-is-agent : richiesta della locazione in cui risiede un agente.
Richiede come singolo parametro l' AID dell'agente di cui si vuole conoscere la locazione.

- **Query-platform-location** : richiesta della lista delle locazioni disponibili all'interno della piattaforma. Non richiede parametri.

Se un agente desidera utilizzare una delle precedenti azioni , dovrà semplicemente spedire un messaggio ACL all' AMS , all'interno del quale specificare l'azione da eseguire seguita dagli opportuni parametri. L'AMS si attiverà nel seguente modo :

- esecuzione dell'azione richiesta

- gestione dello stato dell'agente coinvolto

- eventuale aggiornamento dell' ap-description e dell' ams-agent-description

- risposta all'agente che ha richiesto il servizio tramite un messaggio ACL al cui interno si troveranno le informazioni richieste

2.4.1 – Ricerca della locazione di un agente

Supponendo che un'agente chiamato 'TrovaAgente@sparc20:2001/JADE' volesse conoscere la locazione di un'agente chiamato 'due@sparc20:2001/JADE' dovrebbe spedire un messaggio all' AMS,utilizzando il seguente codice :

```
registerLanguage(SL0Codec.NAME, new SL0Codec());
registerOntology(MobilityOntology.NAME,MobilityOntology.instance());
ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
msg.setSender(getAID());
msg.clearAllReceiver();
msg.addReceiver(getAMS());
msg.setLanguage(SL0Codec.NAME);
msg.setOntology(MobilityOntology.NAME);
msg.setProtocol("fipa-request");
try {
    Action action = new Action();
    action.setActor(getAMS());
    MobilityOntology.WhereIsAgentAction ppp = new MobilityOntology.WhereIsAgentAction();
    AID ics = new AID();
    ics.setName("due@sparc20:2001/JADE");
    ppp.set_0(ics);
    action.setAction(ppp);
    List tuple = new ArrayList();
    tuple.add(action);
    fillMsgContent(msg,tuple);
  }
catch(FIPAException fe) {  fe.printStackTrace(); }
send(msg);
```

Questo codice permette all'agente di utilizzare l'azione 'Where-is-agent' messa a disposizione dalla MOBILITY-ONTOLOGY.
E' necessario che la richiesta venga incapsulata nell'oggetto 'Action' , all'interno del quale deve essere specificato chi dovrà eseguire l'azione e l'identificatore dell'agente di cui si vuole conoscere la locazione.
In questo modo viene inviato all' AMS il seguente messaggio ACL:

```
(REQUEST
 :sender  ( agent-identifier  :name TrovaAgente@sparc20:2001/JADE)
 :receiver  (set ( agent-identifier  :name ams@sparc20:2001/JADE ) )
 :content  "((action (agent-identifier
                                        :name ams@sparc20:2001/JADE
                                        :addresses (sequence )
                                        :resolvers (sequence ) )
 (where-is-agent (agent-identifier
                                       :name due@sparc20:2001/JADE
                                              :addresses (sequence)
                                       :resolvers (sequence ) ) ) ) )"
 :language  FIPA-SL0
 :ontology  jade-mobility-ontology
 :protocol  fipa-request
)
```

L’ AMS risponderà con il seguente messaggio:

```
(INFORM
 :sender  ( agent-identifier
                              :name ams@sparc20:2001/JADE
                              :addresses (sequence
http://sparc20:2002/acc
http://sparc20:2003/acc
http://sparc20:2004/acc ))
 :receiver  (set ( agent-identifier
                              :name TrovaAgente@sparc20:2001/JADE ) )
                         :content  "((result (action (agent-identifier
                          :name ams@sparc20:2001/JADE
                          :addresses (sequence )
                          :resolvers (sequence ) )
              (where-is-agent (agent-identifier
                              :name due@sparc20:2001/JADE
                              :addresses (sequence)
                              :resolvers (sequence ) ) ) )
              (set ( location
                              :name Container-2
                              :protocol JADE-IMTP
                              :address sparc20 ) ) ) )"
 :reply-with  TrovaAgente@sparc20:2001/JADE1020845138756
 :language  FIPA-SL0
 :ontology  jade-mobility-ontology
 :protocol  fipa-request
)
```

2.4.2 - Ricerca delle locazioni disponibili nella piattaforma locale

Se un'agente volesse richiedere le locazioni disponibili sulla propria piattaforma , coincidenti con gli indirizzi dei contenitori attivi , dovrebbe spedire all' AMS , un messaggio ACL utilizzando il seguente codice :

```
registerLanguage(SL0Codec.NAME, new SL0Codec());
registerOntology(MobilityOntology.NAME,MobilityOntology.instance());
ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
msg.setSender(getAID());
msg.clearAllReceiver();
msg.addReceiver(getAMS());
msg.setLanguage(SL0Codec.NAME);
msg.setOntology(MobilityOntology.NAME);
msg.setProtocol("fipa-request");
try {
            Action action = new Action();
            action.setActor(getAMS());
             action.setAction(new MobilityOntology.QueryPlatformLocationsAction());
        List tuple = new ArrayList();
             tuple.add(action);
             fillMsgContent(msg,tuple);
      } catch (FIPAException fe) { fe.printStackTrace(); }
send(msg);
```

Il messaggio ACL corrispondente , inviato all' AMS , sarà il seguente :

```
(REQUEST
:sender  ( agent-identifier                                    :name uno@sparc20:2001/JADE)
:receiver  (set ( agent-identifier
                                :name ams@sparc20:2001/JADE ) )
                                 :content  "((action (agent-identifier
                                 :name ams@sparc20:2001/JADE
                                      :addresses (sequence )
                                      :resolvers (sequence ) )
          (query-platform-locations) ) )"
 :language  FIPA-SL0
 :ontology  jade-mobility-ontology
 :protocol  fipa-request
)
```

L' AMS risponderà con il seguente messaggio :

```
(INFORM
 :sender  ( agent-identifier
                                        :name ams@sparc20:2001/JADE
                                        :addresses (sequence
                                                http://sparc20:2002/acc
                                                http://sparc20:2003/acc
                                                http://sparc20:2004/acc ))
:receiver  (set ( agent-identifier  :name uno@sparc20:2001/JADE ) )
:content  "((result (action (agent-identifier
                                        :name ams@sparc20:2001/JADE
                                        :addresses (sequence )
                                        :resolvers (sequence ) )
                    (query-platform-locations) )
```

```
                (set
                    (location     :name Container-2
                                  :protocol JADE-IMTP
                                  :address sparc20 )
                    (location     :name Container-1
                                  :protocol JADE-IMTP
                                  :address sparc20 )
                    (location     :name Main-Container
                                  :protocol JADE-IMTP
                                  :address sparc20 )
               ) ) )"
 :reply-with  uno@sparc20:2001/JADE1020844591119
 :language  FIPA-SL0
 :ontology  jade-mobility-ontology
 :protocol  fipa-request
)
```

Le informazioni ottenute indicano che sulla piattaforma sono attivi i tre indirizzi :

http://sparc20:2002/acc

http://sparc20:2003/acc

http://sparc20:2004/acc

Inoltre sono presenti tre contenitori con indirizzi globali :

- Container-2@JADE-IMTP://sparc20
- Container-1@JADE-IMTP://sparc20
- Main-Container@JADE-IMTP://sparc20

Se fosse necessario , sarebbe possibile estrarre dal messaggio ACL di risposta , l'elenco dei contenitori disponibili e memorizzarlo in una lista.
Il seguente codice permette di eseguire questa azione , in particolare memorizza l'elenco ottenuto nella lista ' finamia ' .

```
ResultPredicate roar = null;
 try {
            List bubble = myAgent.extractMsgContent(risp);
            roar = (ResultPredicate)bubble.get(0);
      } catch (Exception e) { e.printStackTrace(); }
 Iterator finamia = roar.getAll_1();
```

2.4.3 - Ricerca delle locazioni disponibili in una piattaforma remota

Anche se JADE 2.4 ammette mobilità solo all'interno della piattaforma , le richieste *where-is-agent* e *query-platform-locations* possono essere rivolte all' AMS di piattaforme residenti su macchine remote.
Il metodo da utilizzare e' simile al caso precedente con la variante che il destinatario della richiesta , specificato nel messaggio ACL , sarà l'AMS che gestisce la piattaforma remota.
Supponendo che sia richiesto conoscere le locazioni disponibili su una piattaforma residente sulla macchina 'protos.ing.unimo.it', si dovrà operare nel seguente modo :

```
registerLanguage(SL0Codec.NAME, new SL0Codec());
registerOntology(MobilityOntology.NAME,MobilityOntology.instance());
ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
msg.setSender(getAID());
msg.clearAllReceiver();
AID moz = new AID ("ams@protos.ing.unimo.it:2007/JADE",true);
moz.addAddresses("http://protos.ing.unimo.it:2008/acc");
msg.addReceiver(moz);
msg.setLanguage(SL0Codec.NAME);
msg.setOntology(MobilityOntology.NAME);
msg.setProtocol("fipa-request");
try {
        Action action = new Action();
        action.setActor(getAMS());
        action.setAction(new MobilityOntology.QueryPlatformLocationsAction());
        List tuple = new ArrayList();
        tuple.add(action);
        fillMsgContent(msg,tuple);
  } catch(FIPAException fe) {
          fe.printStackTrace();
         }
send(msg);
```

Il messaggio ACL corrispondente inviato all' AMS remoto sarà il seguente:

```
(REQUEST
:sender  ( agent-identifier  :name LocazioniRemote@sparc20:2001/JADE)
:receiver  (set ( agent-identifier
                    :name ams@protos.ing.unimo.it:2007/JADE
:addresses (sequence http://protos.ing.unimo.it:2008/acc )))
:content  "((action (agent-identifier
                                  :name ams@sparc20:2001/JADE
:addresses (sequence http://sparc20:2002/acc )
                                  :resolvers (sequence) )
            (query-platform-locations ) ) )"
:language  FIPA-SL0
:ontology  jade-mobility-ontology
:protocol  fipa-request
)
```

L' AMS di ' protos.unimo.ing' risponderà con il seguente messaggio:

```
(INFORM
 :sender  ( agent-identifier
                    :name ams@protos.ing.unimo.it:2007/JADE
                    :addresses (sequence        http://protos.ing.unimo.it:2008/acc
                                                http://protos.ing.unimo.it:2002/acc
                                                http://protos.ing.unimo.it:2003/acc ))
 :receiver  (set ( agent-identifier
                    :name LocazioniRemote@sparc20:2001/JADE
                    :addresses (sequence http://sparc20:2002/acc )) )
 :content  "((result (action (agent-identifier
                      :name ams@sparc20:2001/JADE
```

```
                    :addresses (sequence http://sparc20:2002/acc )
                    :resolvers (sequence) )
          (query-platform-locations ) ) (set
                              (location :name Main-Container
                                        :protocol JADE-IMTP
                                        :address protos.ing.unimo.it )
                              (location :name Container-2
                                       :protocol JADE-IMTP
                                       :address protos.ing.unimo.it )
                              (location :name Container-1
                                        :protocol JADE-IMTP
                                        :address protos.ing.unimo.it )
          ) ) )"
 :reply-with  LocazioniRemote@sparc20:2001/JADE1020946151349
 :language  FIPA-SL0
 :ontology  jade-mobility-ontology
 :protocol  fipa-request
)
```

Come si può notare oltre al nome dell' AMS e' necessario specificare uno degli indirizzi attivati sulla macchina remota con la quale si vuole comunicare.
In questo caso si e' ipotizzato che l'agente ' LocazioniRemote ' sapesse che sulla piattaforma remota era stato attivato un HTTP MTP con indirizzo :

http://protos.ing.unimo.it:2008/acc

In realtà , dal messaggio ricevuto in risposta dall' AMS remoto , risulta che sulla piattaforma erano stati attivati anche altri indirizzi.
Nel caso non si conoscessero a priori gli indirizzi attivati sarebbe necessario accedere al file contenente l' ap-description della piattaforma remota ed utilizzare uno degli indirizzi contenuti al suo interno.

2.4.4 - Implementazione dell'azione 'Move – Agent'

L'azione *move-agent* può essere inclusa all'interno di un messaggio ACL per richiedere all' AMS di spostare agenti da un contenitore all'altro.
E' possibile utilizzare questa azione anche per muovere agenti residenti su piattaforme remote. Nel caso , e' necessario richiedere il servizio all' AMS remoto.
Non è permesso però importare ed esportare agenti tra diverse piattaforme.
Quindi l'utilizzo dell'azione *move-agent* e' consentita solo nel caso in cui la locazione di partenza e quella di arrivo si trovino nella stessa piattaforma.
Un agente che volesse muovere un agente chiamato 'uno@sparc20:2001/JADE ' in un contenitore , dovrebbe utilizzare un codice simile al seguente :

```
registerLanguage(SL0Codec.NAME, new SL0Codec());
registerOntology(MobilityOntology.NAME,MobilityOntology.instance());
ACLMessage mess = new ACLMessage(ACLMessage.REQUEST);
mess.setSender(getAID());
mess.clearAllReceiver();
mess.addReceiver(getAMS());
mess.setLanguage(SL0Codec.NAME);
mess.setOntology(MobilityOntology.NAME);
mess.setProtocol("fipa-request");
try {
        MoveAction a = new MoveAction();
        MobileAgentDescription mad = new MobileAgentDescription();
        AID mister = new AID("uno",AID.ISLOCALNAME);
        mad.setName(mister);
        mad.setDestination((Location)finamia.next());
        a.set_0(mad);
        Action action = new Action();
        action.set_0(getAMS());
        action.set_1(a);
        ArrayList blob = new ArrayList(100);
        blob.add(action);
        fillMsgContent(mess,blob);
    } catch(FIPAException fe) {   fe.printStackTrace(); }
send(mess);
```

La locazione in cui l'agente ' uno@sparc20:2001/JADE ' verrà spostato è specificata nella variabile ' finamia ' e dovrà essere una delle locazioni disponibili nella piattaforma.
Il messaggio ACL inviato all' AMS sarà il seguente :

```
(REQUEST
   :sender  ( agent-identifier  :name tre@sparc20:2001/JADE)
   :receiver  (set( agent-identifier  :name ams@sparc20:2001/JADE ) )
   :content  "((action (agent-identifier
:name ams@sparc20:2001/JADE
:addresses (sequence )
                                   :resolvers (sequence ) )
            (move-agent (mobile-agent-description
                              :name (agent-identifier
                                   :name uno@sparc20:2001/JADE
                                   :addresses (sequence )
                                    :resolvers (sequence) )
                             :destination (location
                                          :name Container-5
                                          :protocol JADE-IMTP
                                          :address sparc20 ) ) ) ) )"
 :language  FIPA-SL0
 :ontology  jade-mobility-ontology
 :protocol  fipa-request
)
```

Il messaggio inviato informa che l' agente 'tre@sparc20:2001/JADE ' ha richiesto all'AMS di spostare l'agente ' uno@sparc20:2001/JADE ' nel contenitore il cui indirizzo globale è :

Container-5@JADE-IMTP://sparc20

L’ AMS risponderà con il seguente messaggio :

```
(INFORM
    :sender  ( agent-identifier  :name ams@sparc20:2001/JADE
                                 :addresses
                                (sequence   http://sparc20:2002/acc
                                            http://sparc20:2003/acc
                                            http://sparc20:2004/acc))
    :receiver  (set ( agent-identifier  :name tre@sparc20:2001/JADE ) )
    :content  "FIXME"
    :reply-with  tre@sparc20:2001/JADE1021365224322
    :language  FIPA-SL0
    :ontology  jade-mobility-ontology
    :protocol  fipa-request
)
```

2.5 - DIRECTOR FACILITATOR (DF)

Al momento della creazione di una piattaforma , all'interno del Main-Container viene creato un DF il cui nome è composto dalla stringa ' df@ ' seguita dal nome della piattaforma.
Quando un'agente viene creato , l' AMS lo registra automaticamente.
Il nuovo agente però , non viene registrato dal DF.
Perché ciò accada l'agente deve richiederlo esplicitamente al DF.
Per effettuare la registrazione si utilizza un oggetto , chiamato ' DFAgentDescription ' , in cui sono contenuti i seguenti campi :

- Language
- Ontology
- Protocol
- Name
- Service

Language , *Ontology* e *Protocol* sono di tipo stringa, *Name* rappresenta l' AID dell'agente , mentre *Service* può contenere un elenco di oggetti di tipo ' ServiceDescription '.
Un oggetto ' ServiceDescription ' è a sua volta costituito dai seguenti campi :

- Language
- Ontology
- Protocol
- Name
- Type
- Ownership
- Property

I primi sei sono di tipo stringa , mentre l'ultimo può contenere un elenco di oggetti di tipo ' Property ' .
' Property ' è un oggetto composto da due campi :

- Name
- Value

Il primo di tipo stringa ed il secondo capace di contenere un oggetto Java.

2.5.1 - Registrazione di un agente in un DF residente sulla propria piattaforma

Per effettuare la registrazione in un DF , l'agente dovrà specificare le proprie caratteristiche utilizzando i campi degli oggetti descritti nel paragrafo precedente.
Un agente potrebbe registrarsi nel DF residente sulla propria piattaforma tramite il seguente codice :

```
DFAgentDescription dfd = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType("fipa-df-3");
sd.setName(getName());
sd.setOwnership("JADE");
sd.addProtocols("fipa-request-2");
sd.addOntologies("fipa-agent-management-1");
Property uuu = new Property();
uuu.setName("Servizio-2");
uuu.setValue("1111");
sd.addProperties(uuu);
dfd.setName(getAID());
dfd.addServices(sd);
try {
DFService.register(this,dfd);
} catch (FIPAException e) {
System.err.println(getLocalName()+" registration with DF unsucceeded. Reason: "+e.getMessage());
}
```

2.5.2 - Registrazione di un agente in un DF residente su piattaforma remota

Ad un agente è permessa la registrazione in DF di piattaforme remote , e quindi anche operazioni di ricerca riferite a DF remoti.
Utilizzando il seguente codice , un agente viene registrato nel DF residente nella piattaformaremota:‘protos.ing.unimo.it:2007/JADE‘.
Come nel caso di tutte le operazioni che coinvolgono piattaforme remote è indispensabile specificare almeno uno degli indirizzi attivi su di esse.

```
DFAgentDescription dfd = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType("fipa-df");
sd.setName(getName());
sd.setOwnership("JADE");
sd.addProtocols("fipa-request");
sd.addOntologies("fipa-agent-management");
dfd.setName(getAID());
dfd.addServices(sd);
AID hhh = new AID();
hhh.setName("df@protos.ing.unimo.it:2007/JADE");
hhh.clearAllAddresses();
hhh.addAddresses("http://protos.ing.unimo.it:2008/acc");
try {
        DFService.register(this,hhh,dfd);
        } catch (FIPAException e) {
        System.err.println(getLocalName()+" registration with DF unsucceeded. Reason: "+e.getMessage());
        }
```

2.5.3 - Ricerca di agenti utilizzando le proprietà del DF

Oltre alla gestione delle informazioni che descrivono gli agenti registrati , il DF permette la ricerca di agenti , utilizzando come parametro di ricerca un qualsiasi campo degli oggetti precedentemente descritti o una combinazione di essi.
Questo semplice codice permette di ricercare nel DF tutti gli agenti registrati del tipo ‘ fipa-df-1 ’.
Gli oggetti ‘ DFAgentDescription ‘ degli agenti che soddisfano i parametri di ricerca verranno memorizzati nell’ elenco ‘ agentx ‘ .

```
DFAgentDescription gjg = new DFAgentDescription();
ServiceDescription sds = new ServiceDescription();
sds.setType("fipa-df-1");
gjg.addServices(sds);
DFAgentDescription[] agentx = new DFAgentDescription[900];
try {
      agentx = DFService.search(this,gjg);
} catch (FIPAException e) {System.err.println(getLocalName()+"search with DF unsucceeded.Reason:
"+e.getMessage());}
```

3 – Sviluppo di un agente dedicato alla ricerca e all’archiviazione di sorgenti informative

3.1 – Introduzione

Per mostrare uno dei possibili utilizzi delle proprietà caratteristiche degli agenti mobili, si e' deciso di implementare un agente capace di ricercare in rete una sorgente dati selezionata dall'utente e di immagazzinare ed organizzare le informazioni acquisite.
L'agente realizzato, costituito dalla classe ‘AgentHunter’, è un'estensione della classe ‘GuiAgent’, quindi conserva tutte le caratteristiche e le proprietà tipiche degli agenti.
L'utente accede alle funzioni messe a disposizione dall'agente tramite un'interfaccia realizzata dalla classe ‘InterfacciaAgente’.
Come possibili sorgenti dati da analizzare e' stato deciso di prendere in esame siti della rete Internet.
Una volta che l'utente ha specificato all'agente il sito da ricercare indicandone l'URL, l'agente utilizza la classe ‘SiteSearch’.
Questa classe si occupa dell'analisi e dell'elaborazione delle informazioni contenute nel sito, sfruttando al suo interno la classe ‘HierarchicalTree’ che archivia i dati ottenuti sottoforma di albero gerarchico.

3.1 – Descrizione della classe ‘AgentHunter’

L'agente implementato e' stato realizzato in modo da potersi trovare in cinque diversi stati.
Il primo stato e' relativo all'introduzione dell'URL del sito da analizzare.
Per mostrare la capacita' dell'agente nell'analizzare e decidere l'opportunità dell'acquisizione delle informazioni relative al sito specificato e' stato deciso di indicare una serie di parametri costituiti da semplici stringhe e la modalità di utilizzo di questi parametri.Sono messe a disposizione due modalità :

- Modalità OR : indica all'agente di ritenere interessante il sito preso in esame solo se all'interno delle pagine HTML che lo costituiscono si trova almeno una delle stringhe specificate

- Modalità AND : indica all'agente di ritenere interessante il sito preso in esame solo se all'interno delle pagine HTML che lo costituiscono si trovano tutte le stringhe specificate

Nel secondo e nel terzo stato l'agente e' predisposto ad accettare rispettivamente la serie di stringhe da ricercare e la modalità di ricerca preferita dall'utente.
Una volta inseriti i parametri necessari,l'agente si trova nel quarto stato,in cui e' possibile decidere , se ritornare allo stato iniziale per variare i dati appena inseriti,attivare l'agente per il compito per cui e' stato istruito o uccidere l'agente.
Se si e' deciso di avviare l'esplorazione, l'agente attraverso la classe 'SiteSearch' esegue l'analisi del sito e tramite la classe 'InterfacciaAgente' visualizza le informazioni ottenute , dopo di che l'utente può decidere se ritornarne al primo stato per iniziare una nuova ricerca o uccidere l'agente.
L'uccisione dell'agente comporta la cancellazione di tutti i dati acquisiti durante le ricerche.
L'agente si porta infine in un quinto stato quando l'utente, attraverso la classe 'InterfacciaAgente', accede all'archivio di tutte le ricerche effettuate.

3.2 – Descrizione della classe 'InterfacciaAgente'

L'interfaccia a disposizione dell'utente e' costituita da 3 folder.
Il primo , come mostrato in Figura 8 , consente di inserire in un primo momento l'URL del sito da analizzare attraverso un ComboBox che visualizza i precedenti siti esplorati.
Se l'indirizzo introdotto non è raggiungibile , l'interfaccia mostrerà all'interno del secondo folder un messaggio di errore e permetterà all'utente di inserire nuovi dati.

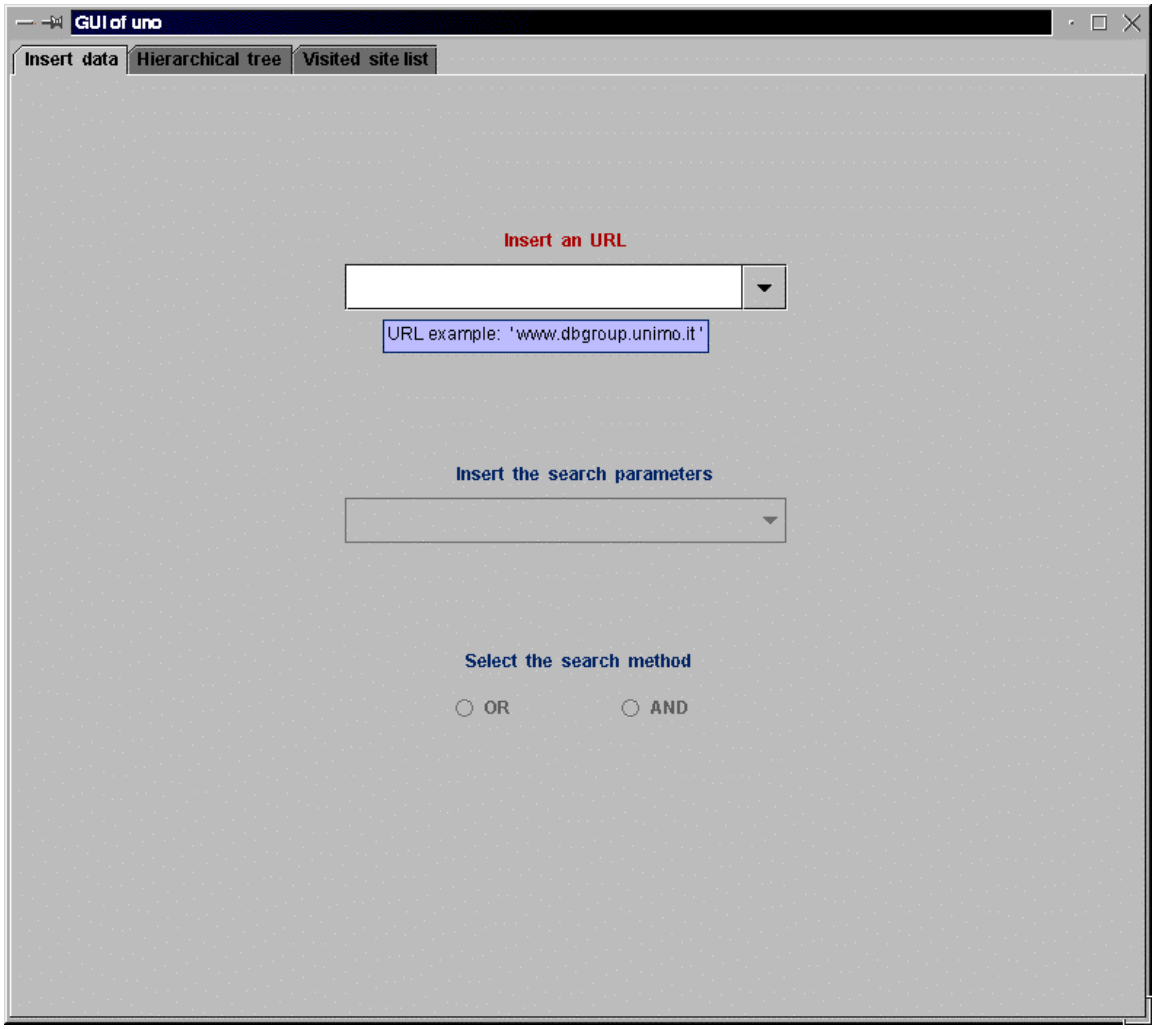

Figura 8

Una volta inserito l'URL, attraverso un secondo Combo box viene richiesta l'introduzione di una o più stringhe separate da uno spazio che rappresenteranno i parametri di ricerca (Figura 9). I parametri relativi a ricerche precedenti sono salvati in una lista a cui è possibile accedere tramite lo stesso Combo box.

Figura 9

Prima di avviare la ricerca è necessario selezionare la modalità desiderata (Figure 10 e 11).


GUI of uno
Insert data
Hierarchical tree
Visited site list
Insert an URL
www.dbgroup.unimo.it
Insert the search parameters
Guerra Vincini Gelati
Select the search method
OR
AND
Search the pages that contain at least one of the parameters

Figura 10

Figura 11

A questo punto si può decidere se modificare i dati inseriti, avviare l'esplorazione o uccidere l'agente (Figura 12).

Figura 12

Una volta terminata la ricerca l'interfaccia mostra in un secondo folder le informazioni ottenute tramite l'utilizzo della classe ‘HierachicalTree’.

La Figura 13 mostra il risultato di una ricerca effettuata sul sito 'http://www.dbgroup.unimo.it'.

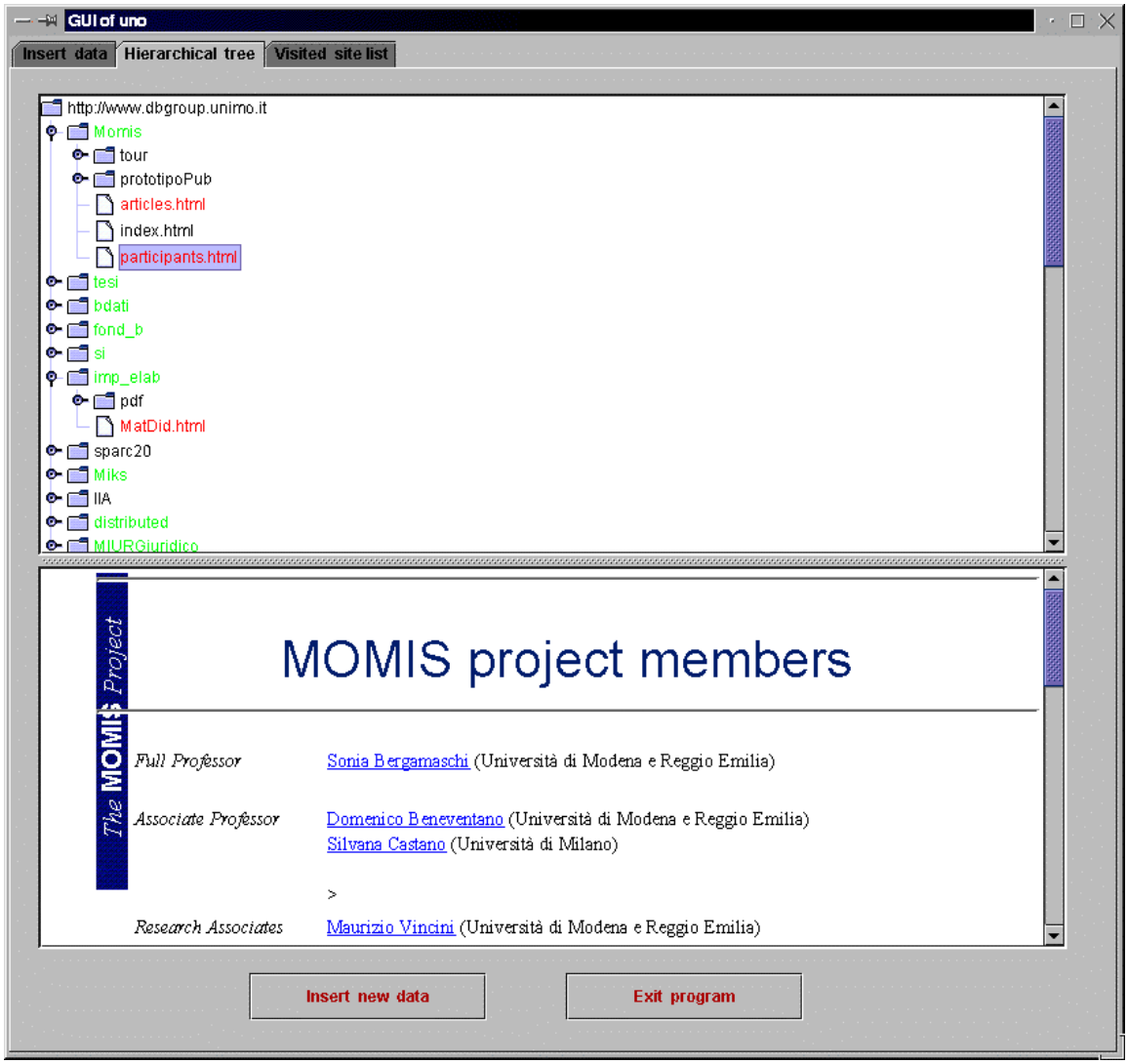

Figura 13

La parte superiore del folder mostra il modello ad albero gerarchico del sito, mentre nella parte inferiore viene visualizzata la pagina HTML del sito selezionata nella parte superiore.
Nel terzo folder viene mostrato l'elenco delle ricerche effettuate , specificando i parametri di ricerca e il risultato ottenuto (Figura 14).

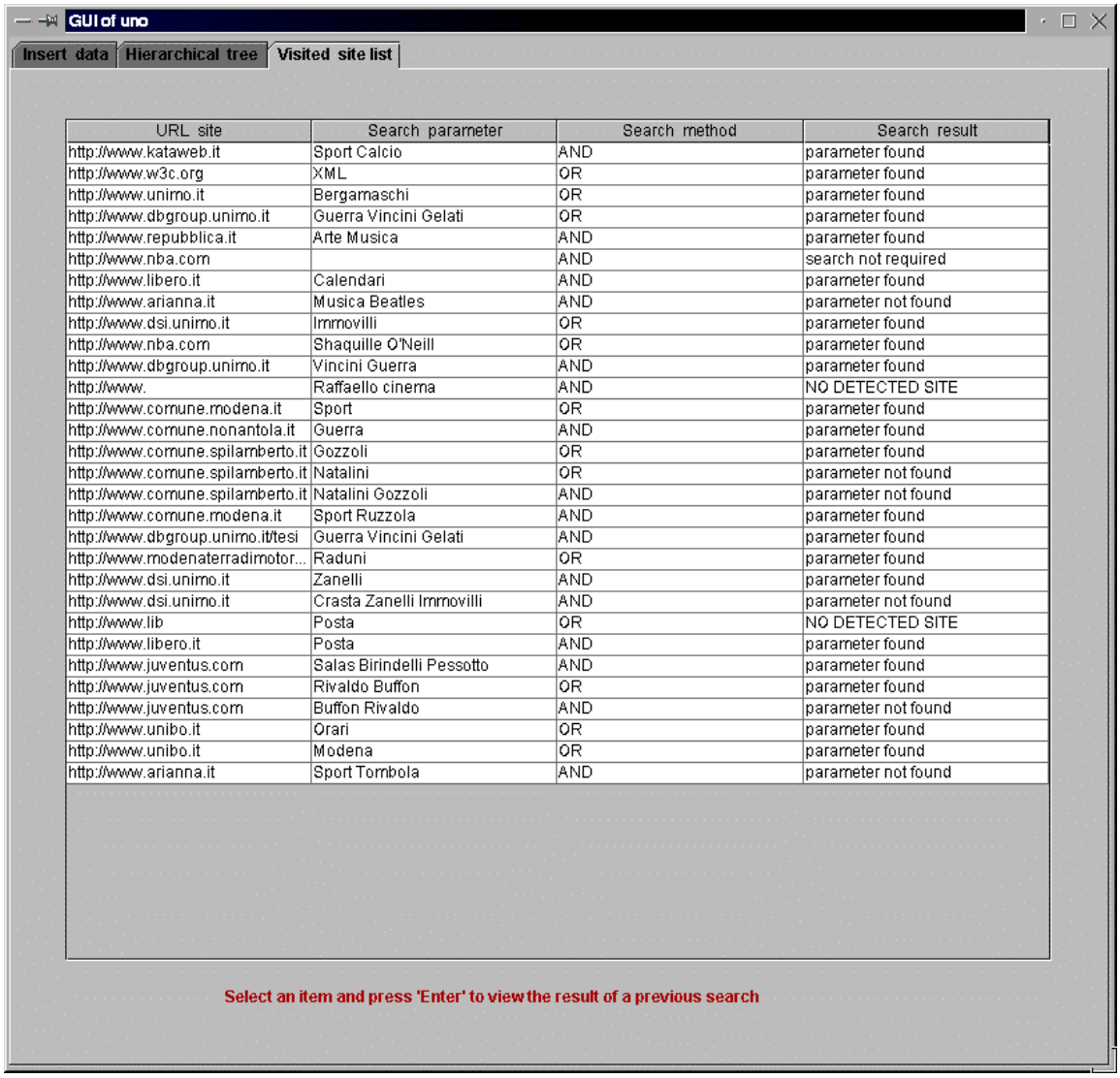

Figura 14

E' possibile selezionare e visualizzare una qualunque delle precedenti ricerche.

3.3 – Descrizione della classe ‘SiteSearch’

La classe ‘SiteSearch’ si occupa dell'analisi e dell'elaborazione delle informazioni contenute nel sito selezionato secondo le indicazioni fornite all'agente.
Dopo aver creato l'URL del sito,viene eseguita una scansione riga per riga della pagina HTML relativa.
L'elaborazione delle pagine HTML si concentra sull'analisi di tutti i riferimenti ad altre pagine HTML e sull'individuazione della presenza delle stringhe inserite come parametri di ricerca.
Ogni riferimento ad altre pagine HTML viene elaborato in modo da ottenere un indirizzo assoluto.
Nel caso non sia stato già individuato in altre pagine del sito preso in esame viene memorizzato in una lista.
Lo scopo prefissato e' quello di ottenere una rappresentazione ad albero gerarchico del sito , quindi i riferimenti ad altre pagine HTML memorizzati saranno solo quelli relativi a pagine riferite al sito stesso.
La scansione del sito quindi, inizierà dall'URL relativo al sito e proseguirà prendendo in esame tutti i riferimenti memorizzati nell'apposita lista.
Contemporaneamente viene ricercata la presenza delle stringhe ricercate.
Al termine della scansione dell'intero sito,vengono creati due file,i cui nomi contengono un numero progressivo per essere distinti dai file relativi a precedenti ricerche.
In uno vengono memorizzati tutti gli indirizzi assoluti relativi a tutte le pagine HTML che compongono il sito,nell'altro l'URL delle pagine che contengono i parametri ricercati e che soddisfano la modalità di ricerca richiesta.
Oltre a queste informazioni vengono memorizzati i parametri di ricerca ed il risultato , in modo da facilitare il reperimento delle passate ricerche.
Una volta creati i file , viene utilizzata la classe ‘HierarchicalTree’.

3.4 – Descrizione della classe ‘HierarchicalTree’

La classe ‘HierarchicalTree’ riceve come parametri i 2 file creati dalla classe ‘SiteSearch’.
Analizzando i parametri viene creata la struttura ad albero gerarchico.
Ogni URL individuata all'interno dei file viene scomposta ed ad ogni oggetto ottenuto viene associata o una directory o una foglia dell'albero.
Se ad esempio di deve elaborare un URL del tipo :

http://www.dbgroup.unimo.it/Momis/prototipoPub/articles.html

nell'albero gerarchico sarà creata la radice 'http://www.dbgroup.unimo.it' che conterrà una directory ‘Momis’ che conterrà a sua volta una directory ‘prototipoPub’ che conterrà infine la foglia ‘articles.html’.
Le pagine contenute nell'albero vengono evidenziate con il colore rosso se contengono le stringhe ricercate e se soddisfano la modalità prescelta.
Le directory contenenti pagine evidenziate in rosso vengono a loro volta evidenziate con il colore verde.
Selezionando tramite il cursore le componenti della struttura gerarchica , viene visualizzata nella parte inferiore della pagina la pagina HTML relativa.
Quando tramite l'interfaccia utente viene richiesto di visualizzare una ricerca eseguita in precedenza, viene richiamata la classe ‘HierachicalTree’ e vengono passati come parametri i file relativi.
In questa situazione nei tre folder dell'interfaccia viene riproposta la situazione relativa alla ricerca selezionata,mostrando i parametri inseriti e l'albero gerarchico risultante.

Conclusioni

MOMIS si colloca nell'ambito dell'integrazione delle sorgenti di informazioni, come ampiamente ribadito nei precedenti capitoli l'obiettivo del progetto è quello di realizzare un'integrazione '*intelligente*' delle informazioni.
Con la parola '*intelligente*' si intende che durante il processo di integrazione vengono utilizzati strumenti di intelligenza artificiale.
Il progetto mira a creare una "vista virtuale" delle fonti coinvolte nel processo, cioè, al contrario per esempio delle "datawarehouse",non vuole materializzare una vista su una macchina locale, infatti nessun dato viene 'copiato', le uniche informazioni che contribuiscono alla creazione della vista sono metadati.
Come e' stato spiegato nel paragrafo 1.5, e' stata proposta una nuova versione di MOMIS che sfrutta l'integrazione con sistemi multi-agente.
Questo sistema, chiamato MIKS, dovrebbe sfruttare le caratteristiche e le proprietà degli agenti intelligenti.
Nella prima parte di questa tesi si e' affrontato lo studio della piattaforma di agenti intelligenti JADE, realizzata dall'Università di Parma, soffermandosi in particolare sulle caratteristiche di mobilità proprie degli agenti.
Successivamente si e' implementato un agente hunter capace di individuare ed archiviare nuove sorgenti informative attraverso la rete Internet per mostrare uno dei possibili utilizzi degli agenti all'interno del sistema MIKS.
Inoltre, per quanto riguarda la ricerca e l'integrazione di nuove sorgenti dati, invece che realizzare un unico agente sarebbe ipotizzabile progettare una comunità di agenti cooperanti fra loro capaci di distribuirsi compiti e risorse.
In questo modo potrebbero essere sfruttate al meglio le caratteristiche degli agenti che riguardano la capacità di dialogare fra di loro, prendere decisioni e modificare i propri compiti a seconda degli eventi che accadono.