Una Piattaforma per lo Sviluppo di Applicazioni Multi-Agente

A. Grosso, A. Gozzi, A. Boccalatte

***Abstract*—Il lavoro presentato in questo articolo si pone l'obiettivo di realizzare un framework per la definizione, l'implementazione e la gestione di agenti software. Innanzitutto viene introdotta l'architettura della piattaforma ospitante il sistema multi-agente e viene presentata una breve descrizione del modello di agente adottato al suo interno. L'agente è modellato attraverso l'utilizzo di comportamenti (behaviour), che costituiscono la sua attività decisionale e computazionale. Tali comportamenti operano in base alle informazioni contenute nella base di conoscenza dell'agente (knowledge), che essi stessi accrescono e condividono. Particolare attenzione è stata prestata alla definizione e persistenza dello stato degli agenti e alla schedulazione delle loro attività. L'implementazione di tali concetti si basa sui servizi offerti dalla tecnologia della Common Language Infrastructure (CLI), ed in particolare sull'utilizzo degli "application domain". L'esecuzione delle attività degli agenti all'interno di application domain differenti garantisce infatti la loro totale autonomia, anche operando all'interno di uno stesso processo in un contesto di multi-threading. E' illustrata infine un'implementazione dei servizi di directory e comunicazione della piattaforma.**

***Index Terms*— agents, agent frameworks, agent platforms, programming environments, software libraries.**

I. Introduzione

L'interesse per la ricerca nel campo degli agenti sta aumentando continuamente. I concetti di "agente autonomo" e "sistema multi-agente" (MAS, Multi-Agent System), introdotti per la prima volta nel campo dell'intelligenza artificiale distribuita (DAI, Distributed Artificial Intelligence), possono essere applicati a numerosi contesti per distribuire il controllo e i processi decisionali tra le differenti componenti di un sistema.

L'interesse per i MAS da parte dei ricercatori è testimoniato dalla grande quantità di studi sull'applicazione delle tecnologie ad agenti nei campi scientifici e industriali più disparati [1] [2].

Articolo ricevuto il 30 Giugno 2003.

A. Grosso, A. Gozzi: PhD students presso il Laboratorio di Informatica DIST Siemens-Orsi, Università di Genova, Genova, GE 16145 Italia, (e-mail: agrosso@dist.unige.it; gozzi@dist.unige.it).

A. Boccalatte, professore associato affiliato al Dipartimento di Informatica, Sistemistica, Telematica, Università di Genova, Genova, GE 16145 Italia (e-mail: nino@dist.unige.it).

Malgrado ciò, allo stato attuale, i sistemi ad agente non sono ancora così diffusi per differenti ragioni. Una di queste è il pesante lavoro di programmazione che ancora è necessario svolgere per ottenere sistemi efficienti, in particolare dal punto di vista dell'integrazione con altre applicazioni.

La ricerca nel campo dell'ingegneria del software orientata agli agenti prende il via dalla possibilità di modellare un sistema software al livello di astrazione dell'agente. Tale livello d'astrazione considera gli agenti come unità atomiche che comunicano tra loro al fine di implementare le funzionalità richieste dal sistema.

Questo lavoro si pone l'obbiettivo di realizzare un framework per lo sviluppo e la gestione di agenti software, che abbia caratteristiche tali da soddisfare il paradigma della programmazione ad agenti [3].

Il framework in oggetto si appoggia alla tecnologia della Common Language Infrastructure (CLI) [4] e alle specifiche architetturali promosse da FIPA per la definizione di sistemi multi–agente intelligenti ed interoperanti [5].

Il cuore della piattaforma è la sua libreria attiva attraverso la quale lo sviluppatore è in grado di generare un affidabile ed efficiente sistema multi-agente. Per ottenere tale risultato, la libreria offre al programmatore i seguenti servizi:

--definizione di agenti autonomi, indipendenti e persistenti;

--esecuzione concorrente dell'attività degli agenti attraverso la schedulazione dei relativi comportamenti;

--gestione di strutture dati persistenti condivise tra i comportamenti del singolo agente;

--comunicazione asincrona e transazionale basata sullo scambio di messaggi;

--accesso ai componenti di servizio dell'architettura FIPA.

II. Common Language Infrastructure

La Common Language Infrastructure (CLI) definisce un contesto nel quale differenti linguaggi di alto livello possono essere eseguiti in differenti contesti. Più precisamente la CLI è un ambiente di runtime con: un formato di file, un sistema di tipi (comune a tutti i linguaggi), un sistema di metadati estendibile, un linguaggio intermedio IL (intermediate language) e una libreria di classi. La CLI è disponibile come standard ECMA con il nome ECMA-335. I programmatori possono realizzare software per questo ambiente utilizzando tool e linguaggi di programmazione in grado di produrre dati

eseguibili ad esso compatibili definiti managed code. In esecuzione una catena di eventi è utilizzata per caricare questi dati e convertirli in codice eseguibile per il dato microprocessore e sistema operativo. Si può quindi riassumere brevemente quello che normalmente avviene quando il codice gestito (managed) viene eseguito. Gli assembly contenenti i tipi e i comportamenti sono caricati e validati; dopo di che le classi richieste sono caricate ed il motore di esecuzione (Runtime) costruisce la struttura degli oggetti in memoria, esegue la verifica dei tipi e compila i metodi invocati. L'uscita di questo progetto è un codice eseguibile specifico per il dato sistema operativo ed il dato microprocessore.

A. *Assembly*

Gli assembly definiscono il modello semantico dei componenti per la CLI, sono i componenti di base per le applicazioni; gli assembly sono l'unità fondamentale per il deployment, il versioning e la riusabilità, lo scope dei riferimenti e le politiche di sicurezza. Un assembly è una collezione logica di file contenente il codice e le risorse di una unità funzionale; sotto certi aspetti si può dire che gli assembly sono per la CLI quello che le DLL o le librerie condivise sono per il sistema operativo. Un tipo non può esistere al di fuori del contesto di un assembly. Tutto il codice che deve essere eseguito dal motore di esecuzione della CLI deve essere contenuto in un assembly.

B. *Application Domain*

Gli application domain, nel seguito app domain, sono gli elementi architetturali responsabili per il caricamento degli assembly nel motore di esecuzione: mentre gli assembly risiedono in memoria, gli app domain forniscono barriere per il loro isolamento. L'app domain è quindi una sorta di contenitore di assembly e l'unità fondamentale per l'isolamento delle applicazioni. Questo isolamento garantisce che un app domain possa essere fermato in maniera indipendente dagli altri e non possa accedere direttamente al codice o alle risorse di un altro app domain.

Rispetto ad un processo un application domain è più leggero. Gli app domain sono appropriati per scenari che richiedono isolamento senza gli alti costi associati all'esecuzione di applicazioni all'interno dei processi. Un processo esegue esattamente una applicazione. Al contrario il motore di esecuzione della CLI consente l'esecuzione multipla di applicazioni all'interno di un unico processo caricandole in app domain separati. Tutti gli application domain condividono lo spazio di indirizzamento dello stesso processo e i servizi di runtime del motore di esecuzione, come ad esempio il garbage collector. È possibile definire le politiche di sicurezza relative agli app domain e quindi applicarle agli assembly in essi contenuti. Inoltre il codice all'interno di un app domain è verificato per essere type safe. Gli assembly operano sempre in un contesto di un app domain. Tutte le comunicazioni verso processi esterni o componenti contenuti in altri domini sono mediate dal Runtime della CLI, che utilizza i meccanismi di remoting e di marshalling rafforzando l'isolamento sotto il controllo degli app domain.

C. *Portabilità*

Sono disponibili differenti implementazioni della Common Language Infrastructure. Il Framework .NET è la soluzione proposta da Microsoft Corp. per sistemi Windows, ma anche una versione shared source delle specifiche della CLI è presentata all'interno del progetto Rotor. Sono poi disponibili implementazioni open source: Portable.NET della Free Software Foundation e Mono patrocinato da Ximian. In particolare Mono comprende un compilatore per il linguaggio C#, un motore d'esecuzione per la CLI ed una ampia libreria di classi.

III. Framework ad Agenti

La creazione di strumenti per lo sviluppo di MAS è iniziata nelle università e nei laboratori di ricerca, ma l'interesse in questo campo si è allargato anche all'industria del software. Ciò è certamente un fatto positivo per l'intera comunità degli sviluppatori di agenti: questa evoluzione prospetta molti vantaggi, soprattutto per quanto riguarda il raggiungimento di livelli di qualità industriale da parte dei MAS attraverso l'introduzione di tecniche avanzate di ingegneria del software.

L'ingegneria del software orientata agli agenti (AOSE) [6], [19], [20] utilizza i concetto di agente autonomo come componente fondamentale per il modello di astrazione di realtà complesse. Secondo l'approccio AOSE, l'ingegneria del software dovrebbe essere in grado di ottenere forti vantaggi utilizzando architetture software e framework ad agenti.

I sistemi multi-agente sono software molto complessi; la costruzione di tali sistemi necessita dunque di un approccio metodico e ingegneristico. In questo campo, molto più che in altri settori dell'ingegneria del software, è essenziale l'adozione di uno standard o di una metodologia rigorosa.

Per quanto riguarda i linguaggi di programmazione, Java sembra decisamente essere il preferito dalla comunità degli sviluppatori di agenti. La maggior parte dei framework disponibili sono scritti in Java: sono molto poche le piattaforme che utilizzano altri linguaggi, e tra i più diffusi troviamo C++, Lisp, Smalltalk/VisualWorks e Prolog. È opportuno citare anche alcuni linguaggi speciali creati appositamente per sviluppare agenti: tra essi ricordiamo COOL, (COOrdination Language) e ADL (Agent Definition Language).

Creato dall'azienda americana Sun Microsystem, Java presenta diverse caratteristiche che lo rendono molto attraente per lo sviluppo di sistemi ad agenti. Si tratta di un linguaggio indipendente dalla piattaforma, la cui architettura è studiata in modo che risultino completamente portabili sia il codice sorgente che il codice intermedio ("bytecode"), e per questo molto adatto ad applicazioni distribuite quali i sistemi multi agente. Altre funzionalità, molto sfruttate dai framework ad agenti, riguardano la gestione semplificata del multi-threading e della comunicazione attraverso la rete (RMI, RPC), il trattamento delle eccezioni, la serializzazione e la reflection.

Le funzionalità di Java sopra esposte si ritrovano anche alla base dell'infrastruttura offerta dalla CLI, in particolare attraverso le funzionalità del linguaggio C# [7]. La CLI e il linguaggio C# sembrano offrire una più che valida alternativa a Java, fornendo strumenti per il multi-threading, la reflection e la comunicazione (remoting), punti di forza di Java.

Prendiamo ora in esame alcune delle piattaforme più diffuse ed utilizzate, facenti parte di progetti ancora attivi e soggette quindi ad una regolare manutenzione (sia correttiva che evolutiva). E' chiaro che se un progetto è stato abbandonato probabilmente non ha riscosso pareri positivi nel mondo dell'IT e sicuramente è stato poco utilizzato dagli sviluppatori.

In base a tutte queste considerazioni si è scelto di fare una breve panoramica sui seguenti framework, di seguito elencati in ordine alfabetico:

1) AgentTool
2) FIPA-OS
3) JADE
4) JiVE
5) OpenCybele
6) Zeus

Particolare attenzione è rivolta a due dei framework elencati: FIPA–OS e JADE. Ciò è dovuto all'evidente correlazione che sussiste tra tali piattaforme ed il prodotto che ci proponiamo di realizzare, che si vuole concentrare sulla definizione di una libreria di sviluppo. Inoltre i framework in questione sono tra i più utilizzati dagli sviluppatori di ambienti multi–agente e aderiscono allo standard FIPA.

AgentTool [8], nato da una collaborazione tra la Kansas State University (KSU) e l'Air Force Institute of Technology (AFIT), è stato sviluppato dal team "Agent Lab" guidato dal professor Scott DeLoach ed è realizzato in Java. AgentTool è principalmente uno strumento di *progetto*. Possiede un ambiente grafico che consente di tracciare diagrammi che descrivono un sistema ad agenti in ogni sua parte. Non è FIPA-compliant, ma segue la metodologia di sviluppo MaSE (Multi-agent System Engineering, anch'essa sviluppata da DeLoach) [9]. Il framework include Spin, un tool per la verifica automatica della coerenza delle conversazioni.

FIPA-OS [13] è stato sviluppato da Nortel Networks/Emorphia, ed è realizzato in Java; OS sta per open source.

Per la fase di sviluppo FIPA-OS fornisce lo strumento TaskGenerator: data una definizione di protocollo, il tool genera automaticamente del codice Java che semplifica l'implementazione del protocollo stesso. Per quanto riguarda il supporto runtime, sono presenti alcuni strumenti per gestire e monitorare lo stato degli agenti, o per l'invio di messaggi. Come dice il nome stesso, questo framework è FIPA-compliant; la gestione degli agenti, quindi, segue gli standard definiti da FIPA. FIPA-OS possiede un meccanismo automatico di "protocol check": un protocollo di interazione tra agenti è definito come un grafo all'interno di una classe particolare (tutti i protocolli standard FIPA sono già definiti). A livello più basso (trasporto) FIPA-OS usa le tecniche RMI e IIOP. La prima privilegia l'efficienza ed è di solito utilizzata tra agenti FIPA-OS, la seconda è FIPA-compliant ma meno efficiente, ed è utilizzata per dialogare tra piattaforme diverse. FIPA-OS supporta il linguaggio FIPA-ACL tramite varie classi di utilità; inoltre possiede classi per codificare-decodificare i vari linguaggi utilizzati come *contenuto* dei messaggi ACL (XML, SL, RDF).

JADE ("Java Agent Developement framework") [14] è stato sviluppato da TILab S.p.A in collaborazione con l'Università di Parma, ed è realizzato in Java. Il supporto runtime è molto buono: dall'interfaccia grafica di Jade (che è essa stessa un agente, chiamato RMA, Remote Management Agent) è possibile gestire e monitorare anche gli agenti che risiedono su altre piattaforme. Uno strumento estremamente utile per il collaudo di un sistema è l'agente Sniffer, che analizza le conversazioni che avvengono all'interno del sistema. Il framework JADE è FIPA-compliant. JADE supporta bene la creazione di più agenti nella stessa Virtual Machine. Un agente può creare un proprio log tramite il metodo write(). Ad ogni agente è assegnato un thread, ai behaviour no: l'esecuzione concorrente dei behaviour è gestita da uno scheduler interno all'agente (nascosto all'utente). JADE include un meccanismo che consente l'implementazione di protocolli di interazione FIPA o di altri protocolli. A livello più basso JADE usa i protocolli di trasporto RMI e IIOP, proprio come FIPA-OS, oltre ad HTTP. JADE supporta il linguaggio FIPA-ACL e ha un supporto per la definizione di ontologie.

JiVE ("JAFMAS integrated Visual Environment") [10], è stato sviluppato presso la Cincinnati University. Si basa sulla libreria JAFMAS ("Java-Based Framework for Multi-Agent Systems"), una piattaforma preesistente e ideata presso la stessa università. Ricalcando la distinzione già vista per AgentTool, possiamo dire che JiVE è uno strumento di progetto mentre JAFMAS è un'infrastruttura di implementazione. L'ambiente di JiVE è basato su un'architettura client-server che consente, tra l'altro, a più sviluppatori di lavorare allo stesso progetto, anche contemporaneamente (funzioni collaborative). JiVE non può essere definito FIPA-compliant, anche se utilizza il linguaggio FIPA-ACL.

OpenCybele [11], è la versione open source di un progetto più esteso chiamato Cybele, sviluppato da IAI, Intelligence Automation Inc, un'impresa privata americana che si occupa di ricerca, ed è realizzato in Java. OpenCybele non possiede alcuno strumento di sviluppo grafico, né per la fase di sviluppo né per quella di deployment. OpenCybele non è FIPA-compliant; è basato su una metodologia chiamata ACP (*Activity-Centric Programming*). ACP non è una metodologia di sviluppo, ma un modello concettuale basato sulle Attività, definite come insiemi di metodi event-driven.

Zeus [12], sviluppato da BT-Lab (British Telecom Laboratories), è anch'esso realizzato in Java ed è FIPA-compliant. Zeus si presenta come uno strumento completo per lo sviluppo ad agenti, in quanto copre esplicitamente tutte le fasi della realizzazione di un MAS dall'analisi e progettazione, all'implementazione e deployment. Per il

progetto, Zeus possiede un ambiente visuale dove è possibile definire l'ontologia, gli agenti, i task che essi sono in grado di eseguire, e molte altre caratteristiche. Per l'implementazione, Zeus genera automaticamente buona parte del codice necessario e fornisce una libreria di classi Java per sviluppare gli agenti. Per la fase di deployment il framework presenta numerosi strumenti per la visualizzazione e il controllo in run-time degli agenti.

IV. L'Architettura della Piattaforma Proposta

L'architettura della piattaforma è un punto chiave per la definizione di un sistema multi–agente, dal momento che la scelta di una specifica architettura influenza fortemente le capacità degli agenti e la loro efficacia, in particolare per ciò che riguarda le reali potenzialità di interoperare con agenti di piattaforme differenti. È stato adottato lo schema logico del modello di gestione degli agenti proposto da FIPA all'interno delle specifiche FIPA2000 [15]. Questa architettura sembra offrire ottime funzionalità nel rispetto dei paradigmi di programmazione ad agenti, garantendone autonomia, proattività e abilità sociali. In particolare l'architettura FIPA è stata adottata da alcuni tra i più utilizzati e diffusi framework ad agenti [16], che possono attestare la sua efficienza e affidabilità; benché tali fattori dipendano fortemente anche dalla modellazione dell'agente e dalla sua specifica implementazione. Infatti, FIPA propone l'architettura come astrazione della piattaforma ad agenti e non fornisce specifiche sull'implementazione interna degli agenti stessi, lasciando totale libertà agli sviluppatori. L'architettura prevede, oltre agli agenti, la presenza di componenti di servizio, definiti obbligatori per il corretto sviluppo dell'attività dei sistemi multi-agente. I componenti di servizio, adottati anche dalla piattaforma ad agenti proposta, possono essere così brevemente riassunti: un controllore e supervisore dei servizi della piattaforma (Agent Management System), un servizio di directory (Directory Facilitator), e un servizio di trasporto (Message Transport Service).

Ad ora il framework qui proposto non rispetta in tutto le specifiche FIPA, dal momento che non viene implementato l'Agent Comunication Language (ACL), previsto dallo standard. L'adozione però della suddetta architettura logica e l'implementazione dei servizi da essa previsti assicura futura compatibilità con sforzi di programmazione limitati.

V. Il modello di Agente

Una breve descrizione del modello di agente, adottato all'interno del framework e fortemente basata sul ben noto paradigma di programmazione ad agenti [3], è qui di seguito presentata. In questo contesto gli agenti sono entità autonome e intelligenti che agiscono senza la necessità di stimoli esterni; gli agenti prendono decisioni in base alla loro conoscenza e alle informazioni acquisite dall'interazione con altri agenti. L'agente è modellato attraverso l'utilizzo di comportamenti, *behaviour*, che caratterizzano la sua attività e ne influenzano lo stato. Oltre ai comportamenti, che costituiscono l'attività decisionale e computazionale dell'agente, viene introdotto il concetto di base di conoscenza, *knowledge*. In sostanza per *knowledge* si intende una raccolta di strutture dati in cui l'agente immagazzina le informazioni che occorrono ai *behaviour* per la loro attività e che essi stessi accrescono e condividono. Si noti che nella *knowledge* non dovrebbero essere inseriti quegli attributi accessori, come indici o buffer, che i *behaviour* utilizzano ma che non costituiscono informazione. Tali parametri non sono dati provenienti dall'esperienza dei comportamenti e quindi non necessitano di particolari attenzioni per la condivisione e la persistenza.

La definizione di un agente, quindi, consiste nello stabilire quali siano i behaviour che lo compongono e quale invece la base di conoscenza che tali comportamenti condividono

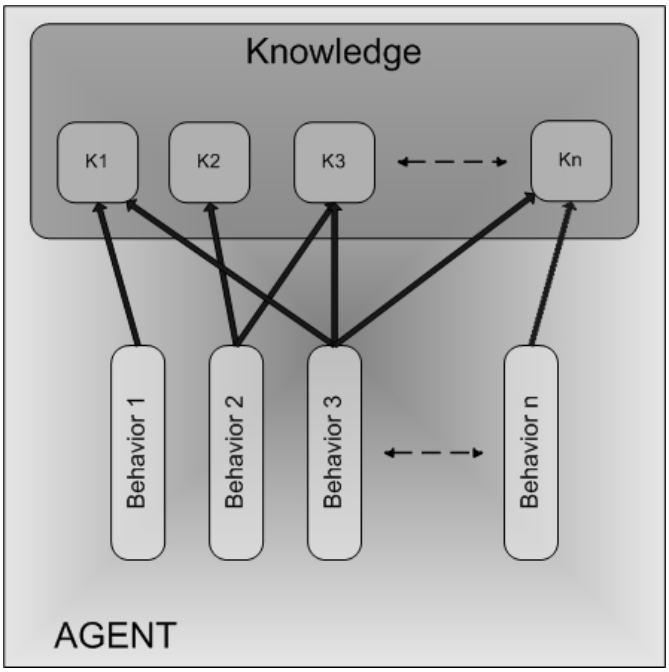

Fig. 1. Il modello di agente. L'agente è multi comportamento e l'accesso alle unità di knowledge avviene in un contesto concorrente.

durante la loro esecuzione. Strutturando in tal modo l'agente e rendendone trasparente l'implementazione pratica, lo sviluppatore che utilizza il framework potrà concentrarsi su una programmazione propriamente orientata agli agenti, e, disponendo di librerie di *behaviour* e *knowledge* già definite, sarà in grado di creare e disporre con poche righe di codice (o direttamente da interfaccia grafica) di un agente completo e già eseguibile.

A seguito del modello proposto, emerge chiaramente il problema dell'accesso concorrente da parte dei behaviour alle risorse condivise, sia alle knowledge che, ad esempio, alle code di messaggi degli agenti. La soluzione proposta da JADE, per gestire l'attività dei behaviour, consiste nel disporre di un unico thread per ogni agente, che implica una schedulazione cooperativa dei comportamenti. Tale soluzione non comporta alcun "overhead" relativo alla sincronizzazione per l'accesso alla risorsa condivisa, ma porta alla necessità di

disporre di una struttura gerarchica dei behaviour, in grado di rappresentare i comportamenti più complessi come aggregazione di comportamenti semplici. La soluzione adottata dal framework proposto consiste nell'utilizzo di un thread per ogni behaviour, in modo analogo a come opera FIPA-OS. Ciò comporta la necessità di gestire l'accesso alle risorse condivise dell'agente, e di prevedere la presenza di un gestore delle conversazioni per la consegna dei messaggi, nonché di un meccanismo per l'accesso alle knowledge (trattato nella sezione seguente).

VI. Problematiche sull'Utilizzo Concorrente della Knowledge

Le informazioni che definiscono lo stato dell'agente, cioè quelle necessarie allo svolgimento della sua attività, devono essere inserite nella *knowledge* per poterne garantire la persistenza e controllarne l'accesso concorrente da parte dei comportamenti. Strutture dati condivise possono infatti essere facilmente oggetto di errori e inconsistenze dovuti al problema delle corse sugli accessi o al verificarsi di un deadlock. Emerge quindi la necessità di garantire la consistenza dell'informazione contenuta nella *knowledge* e far sì che il sistema le preservi dal rischio di deadlock. Pertanto l'accesso dei *behaviour* alle *knowledge* è regolamentato da un semplice algoritmo di prevenzione del deadlock, che prevede i seguenti passi:

1) Definire, all'inizio del blocco di codice in cui si vuole utilizzare la knowledge, tutte le risorse di cui si necessiterà.
2) Bloccare in un ordine stabilito l'accesso alle knowledge dichiarate al punto 1 (Locking).
3) Utilizzare tali risorse all'interno del codice.
4) Rilasciare tutte le risorse sempre seguendo lo stesso ordine (Unlocking).

Si noti che si è parlato di base di conoscenza ma spesso utilizzando *knowledge* al plurale; questo deriva dal fatto che si è stabilito di suddividere la base di conoscenza di un agente in più *knowledge* per garantirne una gestione performante. Infatti occorre trovare il giusto compromesso tra l'avere un'unica *knowledge* o averne molte. Maggiore è il numero delle *knowledge* e minore è la quantità superflua di risorse informative condivise che viene bloccata dai *behaviour*. D'altra parte, è evidente che una eccessiva frammentazione delle risorse aumenterebbe la complessità dell'algoritmo di accesso e rilascio delle stesse. La soluzione del problema è demandata allo sviluppatore dell'agente che, a seconda delle esigenze, potrà, con i servizi esposti dalla libreria di sviluppo del framework, decidere la suddivisione dell'informazione.

Il fatto di delineare una procedura d'accesso alle strutture dati da imporre ai *behaviour* consente inoltre di avere il pieno controllo sulla manipolazione di tali risorse. Ciò rende possibile la definizione di un ambiente transazionale e di conseguenza gestibile in modo consistente da parte di un DBMS. Con tale possibilità si alleggerisce l'onere, da parte dello sviluppatore, di dare persistenza agli agenti, garantendone il pieno recupero anche in caso di crash del sistema.

VII. La Schedulazione dell'Attività degli Agenti

Un fattore decisamente rilevante per la scelta della CLI, come tecnologia di riferimento per l'implementazione della piattaforma, è costituito dalla presenza degli application domain, già discussi nella sezione II.B. L'introduzione di tale strumento all'interno del motore di *runtime* della CLI permette l'esecuzione di più applicazioni all'interno di uno stesso processo. L'application domain ci consente di garantire la totale autonomia dell'agente, anche operando in multi-threading nel contesto di uno stesso processo. Si noti come la gestione di un ambiente multi-agente comporti l'esecuzione concorrente degli agenti stessi e dei relativi comportamenti. La sensazione è che gli strumenti hardware/software disponibili e tradizionalmente utilizzati per l'implementazione di agenti concorrenti, ossia i processi e i thread, non siano del tutto adeguati all'obiettivo proposto. Le tecnologie esistenti propongono a tale riguardo due principali soluzioni: il dualismo agente-processo ed agente-thread. La corrispondenza con un processo rende effettivamente autonoma ed indipendente l'attività dell'agente, ma in questo caso la coordinazione e la gestione degli agenti è possibile solo a livello di sistema operativo. Al contrario, l'utilizzo dei thread per l'esecuzione dei comportamenti degli agenti non sembra garantire pienamente il loro isolamento; niente impedisce al programmatore di definire risorse condivise tra gli agenti. Una soluzione a tale problema sembra esserci offerta dagli application domain, che costituiscono l'unità d'isolamento delle applicazioni nella CLI [ECMA00]. Eseguendo l'attività di ciascun agente all'interno di differenti application domain, si è in grado di disporre dei seguenti servizi:

--l'esecuzione di un agente può essere fermata in maniera indipendente da quella degli altri agenti;

--un agente non può accedere direttamente al codice o alle risorse di un altro agente;

--un errore all'interno di un agente non può coinvolgere altri agenti;

--un applicazione è in grado di controllare la locazione da cui il codice è caricato e la relativa versione.

Gli application domain sono quindi appropriati per scenari che richiedono isolamento senza i pesanti costi associati all'esecuzione di applicazioni all'interno di processi differenti. Un processo esegue esattamente una applicazione. Al contrario il motore di esecuzione della CLI permette a più applicazioni l'esecuzione in un singolo processo caricandole in application domain differenti.

VIII. Creazione di un Istanza dell'Agente

Il framework proposto si basa su un modello orientato agli oggetti per la definizione della piattaforma, dei suoi componenti di servizio, degli agenti, *knowledge* e *behaviour*. Tale modello è progettato per consentire un'elevata riusabilità

dei template di agenti, knowledge e behaviour definiti dallo sviluppatore. Il C# è il linguaggio adottato per l'implementazione di tale modello.

Lo sviluppo degli agenti all'interno della piattaforma avviene, come nel caso di FIPA-OS e JADE, per specializzazione delle classi della libreria di sviluppo che implementano le entità agente e behaviour; inoltre, seguendo il modello prima descritto, dovranno essere definite anche le classi relative alle knowledge di cui l'agente dovrà disporre. Nella libreria di sviluppo del framework, differentemente da JADE e FIPA-OS, sono contenute due classi distinte per modellare l'agente: la classe *Agent*, che implementa l'agente vero e proprio, e la classe *AgentTemplate*, utilizzata dagli sviluppatori per definire tipi di agenti. È all'interno dell'AgentTemplate che il programmatore deve definire tutti gli elementi che caratterizzano l'attività dell'agente e che si riconducono a k*nowledge* e b*ehaviour*.

Vediamo ora come dall'assembly, l'unità base delle applicazioni della CLI, contenente la classe descrivente l'agente, si riesca a creare all'interno del framework l'agente software vero e proprio. Nel momento in cui il dato assembly viene passato alla piattaforma, questa si occuperà di creare il relativo application domain definendone le politiche di sicurezza ad esso associate. La piattaforma provvederà quindi a caricare all'interno dell'application domain l'assembly relativo alla definizione dell'agente, ma anche gli assembly riguardanti le strutture dati che questi necessita di utilizzare. In un secondo tempo verrà creata, sempre all'interno del dato dominio, l'istanza dell'agente vero e proprio. Si noti che l'istanza dell'agente all'interno della piattaforma è del tutto distinta dall'istanza della classe implementata dallo sviluppatore. La classe definita dal programmatore è semplicemente una sorta di contenitore di k*nowledge* e b*ehaviour*. All'interno del framework l'agente corrisponde ad una classe inaccessibile allo sviluppatore e controllata solamente dai componenti di servizio interessati.

In definitiva si crea e si gestisce all'interno della piattaforma un application domain per ciascun agente che si intende schedulare. Ogni agente disporrà quindi di tutti i servizi relativi ai domini della CLI subendone però le restrizioni, che in maniera evidente gli impediranno l'accesso alle attività e alle strutture dati degli altri agenti. Naturalmente sarà garantita la possibilità di comunicazione tra gli agenti, ma limitata a procedure prestabilite e controllate dalla piattaforma. Tali funzionalità si basano sostanzialmente sullo scambio di messaggi, come del resto deve essere per definizione l'interazione tra agenti.

IX. Servizi di Directory e Comunicazione

I servizi di comunicazione e di directory sono definiti come un'astrazione dei relativi componenti di servizio della piattaforma; espongono quindi le funzionalità del dato servizio agli agenti, fornendo al programmatore interfacce per l'implementazione vera e propria.

In questo modo la portabilità della libreria di base della piattaforma è assicurata dalla presenza di differenti implementazioni del motore di esecuzione della CLI, e nel contempo una stretta integrazione con il sistema operativo è possibile attraverso differenti implementazioni dei servizi.

Ad ora la soluzione proposta è basata sull'utilizzo di Microsoft Message Queue (MSMQ). MSMQ sembra essere molto adatto alla comunicazione tra agenti, dal momento che può essere utilizzato per implementare soluzioni per contesti sincroni e asincroni che richiedano alte prestazioni. Esso assicura la consegna dei messaggi, un efficiente meccanismo di routing, un supporto alla sicurezza e alle transazioni, e un sistema di priorità per il recapito dei messaggi. La soluzione prevede l'utilizzo di una coda MSMQ per ogni agente della piattaforma e la presenza di un "conversation manager", che si occupa della consegna dei messaggi ai vari behaviour a seconda della conversazione cui fanno riferimento.

Per ciò che concerne il servizio di directory, viene proposta l'integrazione del framework con Microsoft Active Directory (AD). L'idea è quella di fornire un servizio di "pagine gialle" interno alla piattaforma in grado di interagire con gli agenti, ma anche capace di interoperare con AD per poter esporre tramite esso funzionalità avanzate agli utenti. In questo modo è possibile disporre di un potente strumento distribuito per l'amministrazione del sistema multi-agente.

X. Conclusioni

Il risultato di questo lavoro è un nuovo framework per lo sviluppo di applicazioni ad agenti. Il framework proposto introduce un modello di agente definito nel rispetto del paradigma di programmazione ad agenti, in grado di plasmare l'agente come entità multi comportamento caratterizzata da uno stato di conoscenza reso persistente ed accessibile in un contesto concorrente. La piattaforma è stata implementata con il linguaggio di programmazione C# ed avvalendosi dei servizi offerti dal Framework .NET, un'implementazione delle specifiche della CLI. Ciò ha consentito di ottenere un'infrastruttura software in grado di sviluppare MAS all'interno dei quali le attività degli agenti sono eseguite in modo concorrente e sicuro nel contesto di application domain. Inoltre avere come "target" uno specifico sistema operativo, i sistemi Windows in questo caso, consente l'integrazione del framework a livello di progetto con i componenti del S.O. e di sfruttare i servizi nativi di una piattaforma specifica Resta comunque il fatto che la libreria di sviluppo su cui il progetto si basa è stata implementata basandosi su uno standard, ed è quindi portabile sulle differenti implementazioni della CLI.

Gli aspetti non ancora affrontati dal lavoro presentato, rispetto ai framework brevemente illustrati nella sezione III (in particolare JADE e FIPA-OS), riguardano principalmente l'adozione di un linguaggio standard di comunicazione tra gli agenti, che ne inibisce anche l'interoperabilità con le altre piattaforme. Inoltre la portabilità della tecnologia scelta per la realizzazione del framework è legata di fatto alle

implementazioni della CLI, i cui progetti allo stato attuale sono attivi e non ancora del tutto perfezionati. Ciò se da un lato ne riduce la portabilità, evidenzia dall'altro la grande capacità di integrazione della piattaforma con servizi nativi del sistema operativo "targettizzato". I futuri sviluppi della piattaforma riguarderanno il supporto del linguaggio FIPA-ACL e la realizzazione di strumenti di monitoraggio runtime. Legato solamente alla versione per Windows è prevedibile l'esposizione della piattaforma come servizio del S.O., per poter utilizzare metodi di replica da esso offerti.

RIFERIMENTI

[1] E.Oliveira, K.Fischer, O.Stepankova, "Multi-agent systems: which research for which applications", Robotics and Autonomous Systems 27 (1999) 91-106.

[2] N.R.Jennings, M.Wooldridge, "Applications of Intelligent Agents", In "Agent Technology: Foundations, Applications, and Markets", N.R. Jennings and M.J Wooldridge (Eds.), Springer. (1998).

[3] M.Wooldridge, "Intelligent Agents", in "Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence", G. Weiss (Ed.), MIT Press, Cambridge, MA. (1999).

[4] ECMA, "Common Language Infrastructure", Standard ECMA-335, available at http://www.ecma-international.org, (2002)

[5] "Foundation for Intelligent Physical Agent" http://www.fipa.org

[6] Wooldridge M. J. and Jennings N. R., "Software Engineering with Agents: Pitfalls and Pratfalls", IEEE Internet Computing, 3(3):20–27, May/June 1999.

[7] ECMA: "C# Language Specifications", Standard ECMA-334, available at http://www.ecma-international.org, (2002)

[8] Scott A. DeLoach and Mark Wood, "Developing Multiagent Systems with agentTool", International Workshop, ATAL-2000, Boston, MA, USA, July 7-9, 2000, Proceedings, Lecture Notes in Artificial Intelligence. Springer-Verlag, Berlin, 2001.

[9] Scott A. DeLoach, Mark F. Wood, and Clint H. Sparkman, "Multi-agent Systems Engineering", The International Journal of Software Engineering and Knowledge Engineering, Volume 11 no. 3, June 2001.

[10] A.K.Galan, "Comparison of Existing Design Tools with JiVE", in "JiVE: JAFMAS integrated Visual Environment". http://www.ececs.uc.edu/~abaker/JiVE/Chapter2.html

[11] "OpenCybele Agent Infrastructure - Users Guide". http://www.opencybele.org/docs/Users.pdf

[12] H.S. Nwana, D.T. Ndumu and L.C. Lee, "ZEUS: An advanced Tool-Kit for Engineering Distributed Multi-Agent Systems", In: Proc of PAAM98, pp. 377-391, London, U.K., 1998.

[13] Stefan Poslad, Phil Buckle, Rob Hadingham, "The FIPA-OS agent platform: Open Source for Open Standards", Published at PAAM2000, Machestor, UK, April 2000.

[14] Bellifemine, F., Rimassa, G., Poggi, A., "JADE - A FIPA-compliant Agent Framework", in "Proceedings of the 4th International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agents", London, 1999.

[15] "FIPA Agent Management Specification" http://www.fipa.org/specs/fipa00023/SC00023J.html

[16] "Publicly Available Agent Platform Implementation" http://www.fipa.org/resources/livesystems.html

[17] Mingins, C. & Nicoloudis, N., ".NET: a new Component-oriented Programming Platform, Journal of Object-oriented Programming, October/November, 48-51, 2001.

[18] Y.Demazeau, P.M. Ricordel, "From Analysis to Deployment: a Multi-Agent Platform Survey", LEIBNIZ laboratory, 2000. http://www-lia.deis.unibo.it/confs/ESAW00/pdf/ESAW07.pdf

[19] Michael Wooldridge, Paolo Ciancarini, "Agent-Oriented Software Engineering", Handbook of Software Engineering and Knowledge Engineering Vol. 0, No. 0 (1999)

[20] Nicholas R. Jennings and Michael Wooldridge, "Agent-oriented software engineering", In J. Bradshaw, editor, Handbook of Agent Technology. AAAI/MIT Press, 2000.