Un'Infrastruttura per la Mobilità in AgentService

A. Grosso, A. Boccalatte, C. Vecchiola

***Abstract*—L'articolo presenta la soluzione alle problematiche legate alla mobilità degli agenti adottata nella piattaforma AgentService. L'infrastruttura per il trasferimento degli agenti sfrutta il modello di agente della piattaforma che prevede la separazione tra lo stato e le attività dell'agente. L'implementazione dell'infrastruttura per la mobilità si avvantaggia della struttura modulare della piattaforma e si integra in modo del tutto trasparente per gli agenti e gli altri componenti. AgentService offre un servizio di mobilità debole, nonostante ciò garantisce il mantenimento dello stato degli agenti tramite il trasferimento delle strutture dati, della coda dei messaggi, delle conversazioni e dello stato dei comportamenti. Il servizio di mobilità degli agenti è inoltre sfruttato per l'applicazione di politiche di bilanciamento di carico tra piattaforme federate**

***Index Terms*—Agent Mobility, Load Balancing Policy, Agent Framework**

I. Introduzione

Le metodologie, le architetture e le tecnologie che vengono utilizzate per lo sviluppo di applicazioni distribuite manifestano i propri limiti quando sono applicate a sistemi distribuiti di notevoli dimensioni, potenzialmente illimitati, come si verifica per applicazioni Internet. In particolare ciò è ancor più evidente quando si ha a che fare con sistemi che debbano offrire un elevato grado di configurabilità, alta scalabilità e facilità di personalizzazione. Un metodo efficiente per la soluzione di problemi di questo tipo ci viene offerto dalle tecnologie basate sulla mobilità del codice, in altre parole la possibilità di spostare codice attraverso i nodi di una rete [1].

La mobilità del codice non è un concetto nuovo, gli *Applet Java* ne sono un lampante esempio, ma il lavoro di ricerca in questo campo è sempre attivo, sia per quanto riguarda le disquisizioni concettuali sia per ciò che riguarda gli aspetti puramente tecnologici, si veda ad esempio [2], [3].

Un ambito a cui i concetti di mobilità ben si adattano è quello degli agenti software. Gli agenti ci forniscono, in effetti, un'astrazione tale da rendere semplice l'applicazione del concetto di mobilità. Le caratteristiche intrinseche degli agenti quali l'autonomia, l'inserimento in un ambiente, la pro-attività e la cooperazione, consentono di introdurre con naturalezza il concetto di agente mobile.

L'idea di agente mobile, vale a dire la possibilità di trasferirne il codice e lo stato tra nodi di una rete, porta a diverse considerazioni: la prima è che deve esistere una struttura che consenta all'agente di spostarsi; la seconda è che l'agente dovrebbe avere una certa intelligenza, tale da renderlo particolarmente autonomo nel decidere sugli spostamenti. Una volta realizzata un'infrastruttura adatta e una volta che all'agente saranno forniti tutti gli strumenti perché se ne avvalga, si potranno ottenere svariati vantaggi. Grazie agli agenti mobili, gli amministratori delle piattaforme possono disporre di uno strumento più efficiente per il bilanciamento delle risorse. Dall'altro lato la mobilità sembra rafforzare l'autonomia dell'agente stesso, che ad esempio potrebbe decidere su quale computer migrare in base alla disponibilità delle risorse.

In questo articolo viene presentata l'infrastruttura di supporto alla mobilità per la piattaforma AgentService. La soluzione proposta è risultata particolarmente efficace grazie al modello di agente adottato dalla piattaforma; tale modello consente di gestire con facilità lo stato degli agenti e quindi di mantenerlo persistente e/o trasferirlo altrove. Inoltre lo sviluppo degli strumenti per la mobilità risulta facilitato dall'architettura modulare di AgentService, la quale permette di arricchire i servizi offerti agli agenti attraverso la realizzazione di moduli aggiuntivi; il tutto può avvenire in maniera trasparente e senza la necessità di sostanziali modifiche all'infrastruttura software. Infine la tecnologia offerta dalla Common Language Infrastructure (CLI) [4, 5], su cui AgentService si basa, fornisce gli strumenti necessari ad un'implementazione efficiente del servizio grazie ad alcune particolarità quali la serializzazione, la presenza di primitive di comunicazione remota, l'utilizzo di domini applicativi e la gestione delle unità di distribuzione del codice (Assembly).

Nelle sezioni successive descriveremo le caratteristiche principali della piattaforma AgentService e dei componenti che influiscono maggiormente sullo sviluppo di un'infrastruttura per la mobilità, quali il modello di agente ed alcuni dei moduli che implementano le funzionalità di base (Sezione II). Dopo una breve introduzione sui problemi relativi alle problematiche associate agli agenti mobili (Sezione III), descriveremo in dettaglio le caratteristiche architetturali dell'infrastruttura per la mobilità e la sua

Manuscript received October 4, 2005.

A. Grosso, A. Boccalatte, and C. Vecchiola are with Department of Communication, Computer and Systems Sciences, University of Genova, 16145 Genova Italy (phone: +39-010-353-2812; e-mail: {agrosso, nino, christian}@ dist.unige.it).

interazione con gli altri componenti della piattaforma AgentService (Sezione IV). Valuteremo perciò gli aspetti riguardanti le possibili applicazioni dell'infrastruttura quali l'attuazione di politiche di bilanciamento di carico tra più piattaforme (Sezione V). Alcune osservazioni finali sulle caratteristiche dell'architettura, i suoi punti di forza ed i suoi punti deboli, seguiranno nelle conclusioni.

II. Architettura di AgentService

A. Caratteristiche Principali

AgentService [6] è un framework per lo sviluppo di applicazioni orientate agli agenti basato sulla Common Language Infrastructure, di cui un'implementazione è il .NET framework. AgentService offre un particolare modello di agente ed un ambiente di run-time per l'esecuzione degli agenti conforme alle specifiche FIPA [7]. In letteratura sono presenti numerosi lavori riguardanti piattaforme ad agenti, tra i più diffusi si veda Zeus [8], FIPA-OS [9] e JADE [10]. L'architettura della piattaforma è estremamente modulare: sia le funzionalità di base sia quelle aggiuntive sono implementate utilizzando l'astrazione del modulo. Questa soluzione architetturale rende la piattaforma ad agenti un ambiente di run-time molto flessibile e facilmente personalizzabile a particolari esigenze, nonché facilmente estendile nelle sue funzionalità. I moduli si dividono in due classi: i moduli fondamentali e quelli addizionali. I moduli fondamentali implementano quei servizi necessari alla piattaforma per attivarsi; questi comprendono la gestione degli assembly in cui sono contenuti i tipi degli agenti (S*torage*), la gestione della messaggistica, la gestione della persistenza e le funzionalità di logging. Con i moduli addizionali vengono implementate tutte quelle funzionalità che arricchiscono di servizi la piattaforma, ma la cui assenza non è vincolante per l'attivazione della piattaforma stessa.

Allo scopo di rendere più fruibile la trattazione dell'implementazione della mobilità in AgentService descriveremo brevemente alcuni elementi fondamentali dell'architettura di AgentService: il modello di agente adottato ed i moduli per la gestione dello storage e del servizio di messaggistica.

B. Il Modello di Agente

AgentService modella un agente come un'entità software la cui base di conoscenza è definita da un insieme di dati chiamati *Knowledge* e le cui attività sono descritte da task concorrenti che prendono il nome di *Behavior*. L'insieme delle *knowledge* utilizzate da un agente ne definisce lo stato, mentre il suo comportamento è descritto dall'insieme dei *behavior* che sono in esecuzione. Una *knowledge* è di fatto simile ad un *record* del linguaggio Pascal od ad una *struct* del linguaggio C, sebbene possa essere caratterizzata anche da metodi è principalmente pensata per esporre delle proprietà (queste possono fare riferimento ai tipi base della CLI od a classi anche definibili dal programmatore). I *behavior* sono a tutti gli effetti delle classi generiche definibili dall'utente e presentano un metodo particolare che costituisce l'*entry point* della loro esecuzione. I *behavior* di un agente possono condividere le *knowledge* che ne definiscono lo stato e l'ambiente di run-time garantisce l'accesso esclusivo alle *knowledge*, in modo relativamente trasparente.

Dal punto di vista implementativo ad ogni istanza di agente viene associato un differente *Application Domain* che garantisce l'esecuzione in maniera autonoma del codice in esso contenuto. L'Application Domain è una struttura nuova introdotta con la CLI ed è assimilabile ad un "processo leggero" in quanto ha un proprio spazio di memoria, è possibile creare in esso più thread di esecuzione ed associarvi differenti permessi di esecuzione, ma il suo *setup* è più leggero di quello di un processo. L'esecuzione di un agente all'interno di un Application Domain ne garantisce l'autonomia e l'isolamento dagli altri agenti: l'unico modo per mantenere dei riferimenti ad oggetti in Application Domain differenti è l'impiego di tecniche di comunicazione esplicite quali il *Remoting*.

C. Gestione dello Storage

Lo *Storage* della piattaforma costituisce un repository virtuale di tutte le classi necessarie al ciclo di vita degli agenti che vengono creati ed ospitati all'interno della piattaforma. In particolare nello *Storage* sono memorizzate le classi che definiscono i tipi di agente, le *knowledge* ed i *behavior* usati da tali agenti nonché i tipi logicamente dipendenti se non già compresi nella class library della CLI. I tipi eseguibili ed istanziabili dalla CLI sono fisicamente memorizzati in un file od in un insieme di file logicamente collegati che prende il nome di assembly. In ultima analisi lo *Storage* si occupa della gestione degli assembly che contengono i tipi di agenti e tutto ciò che serve per crearne delle istanze. La CLI per identificare in modo univoco gli assembly utilizza una tecnica di crittografia asimmetrica basato su chiave pubblica e privata normalmente opzionale ma richiesto da AgentService.

Il modulo dello *Storage* viene interrogato ogni volta che la piattaforma deve creare l'istanza di un particolare tipo di agente e restituisce tutti gli assembly necessari a creare l'istanza. Per poter caricare nello *Storage* un assembly occorre firmarlo, l'apposizione della firma ad un assembly permette di identificarlo in maniera univoca indipendentemente dal nome dei file fisici che lo costituiscono. Questo è un elemento che diventerà di fondamentale importanza quando verrà illustrata l'infrastruttura che permette la mobilità.

D. Gestione del Servizio di Messaggistica

Il servizio di messaggistica è gestito da un opportuno modulo che rende di fatto possibile la comunicazione tra agenti attraverso scambio di messaggi (*Messaging*). Il modulo di messaggistica mantiene una coda di messaggi per ogni agente che è ospitato nella piattaforma in cui è installato il modulo. All'atto della creazione di un agente il modulo fornisce un opportuno client per il servizio di messaggistica. Le specifiche di AgentService per l'implementazione del

modulo di messaggistica prevedono che sia garantito il semplice servizio di scambio messaggi, ma è prevista come funzionalità addizionale la possibilità da parte degli agenti di usufruire delle conversazioni che forniscono un servizio di comunicazione connesso tra due agenti. Il modulo fornito con l'installazione di default della piattaforma offre nativamente questo servizio. I messaggi che possono essere scambiati tra due agenti oltre ad aderire alle specifiche fornite da FIPA in tale ambito devono contenere oggetti serializzabili, requisito fondamentale dal momento che due agenti risiedono in Application Domain diversi.

III. Nozioni di Mobilità

La mobilità è sicuramente una proprietà importante per gli agenti così come per gli oggetti; ciò è dovuto al fatto che la mobilità del software è in grado di portare al sistema maggiore robustezza, prestazioni, scalabilità ed espressività [11]. Gli agenti mobili sono quindi divenuti un paradigma per strutturare applicazioni distribuite.

In letteratura è possibile trovare una definizione che caratterizza in maniera sufficientemente esaustiva l'essenza di un agente mobile [12, 13]: un'agente mobile è un'entità software che esiste in un dato ambiente e possiede parte delle caratteristiche dell'agente. Un agente mobile deve contenere tutti i seguenti modelli: un modello di agente, un ciclo di vita, un modello computazionale, un modello di sicurezza, un modello di comunicazione ed infine un modello di navigazione.

In particolare per quanto riguarda il modello di navigazione, FIPA ha definito un adeguato ciclo di vita dell'agente. Tale specifica estende il consueto ciclo di vita prevedendo in più lo stato *transit* e due azioni aggiuntive che consentono rispettivamente di entrare e di lasciare tale stato (*move* ed *execute*). Questo consente di rappresentare ogni stato assumibile dall'agente nel contesto dell'AMS (*Agent Management System*) [7]. L'agente stesso è in grado di richiedere l'azione *move*, mentre è la piattaforma, attraverso l'AMS, ad essere responsabile di completare la migrazione eseguendo l'operazione di *execute*.

Gli agenti mobili richiedono naturalmente un opportuno ambiente di run-time in grado di fornire un servizio di trasferimento per essere spostati da un nodo ad un altro: l'ambiente è costruito sopra ad un sistema host. Il compito primario è fornire un ambiente in cui gli agenti mobili possano operare. Essi devono poter comunicare tra loro sia localmente sia in remoto in modo trasparente.

In letteratura c'è una distinzione tra due differenti tipi di mobilità basata sul fatto che lo stato dell'esecuzione sia o no trasferito assieme all'unità di computazione [14]. I sistemi in grado di fornire il trasferimento dello stato dell'esecuzione si dice supportino la mobilità forte (*strong mobility*), al contrario i sistemi che perdono lo stato dell'esecuzione durante il trasferimento si dice forniscano una mobilità debole (*weak mobility*). Nei sistemi in cui la mobilità è forte, la migrazione risulta completamente trasparente al programma trasferito, mentre con la *weak mobility* è richiesta un ulteriore sforzo di programmazione per salvare manualmente parte dello stato dell'esecuzione.

Ad esempio un semplice agente scritto come un *Applet Java* fornisce mobilità del codice attraverso lo spostamento dei file delle classi da un server web ad un browser, ma naturalmente non vi sono informazioni associate allo stato. Al contrario in Aglets [15], piattaforma basata su Java e sviluppata da IBM, vengono trasferiti anche I valori delle variabili istanziate, senza però tenere conto dello stack e del program counter. Infine un esempio di mobilità forte ci viene fornito da Sumatra [16], sviluppato presso l'Università del Maryland, che consente il trasferimento del contesto di run-time dei thread di Java assieme al codice durante la migrazione.

IV. Mobilità in AgentService

A. Introduzione

Il modello di agente su cui AgentService si basa, sembra possedere tutte le caratteristiche richieste dalla definizione di agente mobile, a partire dalla gestione del ciclo di vita che è quella definita da FIPA.

AgentService implementa un modello di mobilità debole, ma fornisce alcuni servizi aggiuntivi che possono consentire la sua attuazione in maniera trasparente ai programmatori. L'idea base è quella di sfruttare il modello di agente adottato da AgentService trasferendo dell'agente solamente lo stato, vale a dire le strutture dati contenenti le *knowledge* e lo stato dei *behavior*. All'interno della piattaforma di destinazione le attività dell'agente possono essere riavviate in conformità a ciò che è indicato nello stato persistito dell'agente. Inoltre, il framework fornisce agli sviluppatori un punto d'accesso per controllare lo stato e le attività dell'agente prima che riprenda la sua esecuzione. I dettagli di questo processo sono analizzati nella sezione seguente.

B. Implementazione

Grazie al sottostante modello di agente, la mobilità in AgentService può essere ottenuta con relativa facilità: la separazione tra lo stato dell'agente e le sue attività consente una semplice implementazione del processo di migrazione. Per poter schedulare le attività di un agente, l'ambiente di run-time di AgentService necessita dei dati che definiscono lo stato dell'agente e degli assembly contenenti la definizione del tipo di agente. Quindi, spostare un agente tra due installazioni di AgentService implica, innanzi tutto, la presenza sulla piattaforma di destinazione del dato tipo di agente (AgentTemplate) all'interno dello *Storage* e richiede il trasferimento dello stato dell'agente.

Una volta che l'agente è stato spostato è possibile riavviare le sue attività istanziando un nuovo agente del tipo specifico e ripristinandone lo stato in maniera simile a ciò che avviene per un qualsiasi agente di AgentService dopo un crash o un riavvio del sistema. Il ripristino dello stato implica quindi il

caricamento degli oggetti *knowledge* trasferiti, la ricostruzione delle conversazioni in corso e dei messaggi presenti nella coda e l'attivazione di tutti i *behavior* in esecuzione quando l'agente è stato fermato. Le informazioni sugli oggetti *knowledge* e lo stato di ciascun *behavior* (ready, active, suspended) sono tutto ciò che veramente occorre per trasferire un agente. Discorso a parte merita la gestione delle comunicazioni e della reperibilità dell'agente mobile che è trattata nel paragrafo successivo.

Il servizio di mobilità è implementato all'interno di un modulo addizionale della piattaforma che si occupa di seguire la migrazione dello stato dell'agente e, quando necessario, del trasferimento dei relativi assembly. Quando un agente richiede un'azione *move*, gli agenti AMS delle piattaforme coinvolte contrattano la possibilità di uno spostamento e successivamente delegano al modulo di mobilità il trasferimento.

La fase di contrattazione può essere controllata dall'amministratore della piattaforma in due differenti modi, entrambi portano ad influenzare il comportamento dell'AMS. Il primo, quello più semplice, consiste nella modifica, al momento dell'installazione di AgentService, del file di configurazione della piattaforma; attraverso questo è possibile indicare se consentire o no l'hosting di agenti provenienti da altre piattaforme e in maniera duale se permettere l'invio di agenti verso altre piattaforme. Di default sono entrambi negati. Come si vede questo meccanismo di controllo è molto semplice, ma estremamente limitativo, è quindi necessario un approccio che fornisca maggiore flessibilità e potere decisionale.

Il secondo modo attraverso il quale è possibile controllare, da parte dell'amministratore, la mobilità degli agenti consiste nell'implementazione di due metodi specifici. Tali metodi vengono invocati dall'AMS nel momento in cui si verifica la necessità di prendere decisioni sul trasferimento di un agente. L'implementazione di default delle procedure si basa appunto sulle informazioni contenute nel file di configurazione della piattaforma, è, infatti, attraverso l'implementazione di default che viene applicato il controllo descritto nel paragrafo precedente. Ridefinendo invece tali metodi, è possibile modificare da codice il comportamento dell'AMS riguardo alla gestione della mobilità degli agenti nel contesto della data piattaforma di appartenenza. Tale meccanismo consente di personalizzare al meglio le decisioni, ma richiede, rispetto al primo, un maggior sforzo per l'amministratore.

Discorso differente merita l'applicazione di politiche di *load balancing* che sono analizzate nella sezione successiva.

Il processo di trasferimento di un agente può quindi essere attivato direttamente dall'agente stesso, attraverso l'invio di una richiesta all'AMS, o in alternativa utilizzando l'interfaccia di programmazione della piattaforma (*IPlatformController*): gli amministratori della piattaforma possono decidere di spostare agenti tra differenti installazioni di AgentService. Attraverso l'*IPlatformController* anche altre applicazioni software sono in grado di controllare la mobilità degli agenti ed applicare ad esempio algoritmi di distribuzione del carico.

C. Comunicazione e Reperibilità degli Agenti Mobili

Nel processo di trasferimento di un agente è importante considerare come le comunicazioni, intrattenute dall'agente mobile, possano proseguire in maniera trasparente al programmatore anche una volta avvenuto il trasferimento. I problemi di comunicazione e reperibilità sono legati al fatto che gli agenti che hanno comunicazioni in corso con l'agente mobile probabilmente saranno in possesso, relativamente ad esso, di un *agent identifier* (AID) non aggiornato.

In AgentService la comunicazione tra agenti può avvenire in due differenti modi: attraverso l'invio o la ricezione di messaggi semplici (one-shot) oppure tramite lo scambio di messaggi nel contesto di una conversazione. Per garantire la trasparenza, dal punto di vista dei programmatori, nelle comunicazioni con agenti mobili si sono definite delle specifiche aggiuntive per il modulo di messaggistica della piattaforma. L'implementazione del modulo di AgentService che decide di seguire tali specifiche deve garantire la corretta gestione delle conversazioni anche in presenza di agenti mobili. Tale problematica è risolta attraverso l'aggiornamento delle strutture dati che sono alla base delle conversazioni ogni qual volta che un agente coinvolto in una conversazione viene trasferito. Così facendo entrambi gli agenti sono in grado di continuare a scambiarsi messaggi nel contesto di una conversazione in modo indipendente rispetto alla mobilità.

Per quello che concerne invece la comunicazione basata sullo scambio di messaggi semplici, la reperibilità è garantita attraverso l'utilizzo dei cosiddetti *resolvers* presenti negli AID degli agenti. All'interno dell'AID di ciascun agente mobile dovranno essere indicati anche gli AID degli AMS delle piattaforme federate, vedi Sezione V, in modo che possano essere contattati per avere il nuovo indirizzo dell'agente trasferito.

D. Le Fasi del Processo di Mobilità

Le fasi del processo di trasferimento possono essere riassunte come segue:

- contrattazione con la piattaforma di destinazione per la mobilità dell'agente prendendo informazioni sullo *Storage* di destinazione (la contrattazione avviene tra gli agenti AMS delle rispettive piattaforme);
- blocco delle attività dell'agente, persistenza del suo stato e passaggio allo stato *transit* attraverso l'azione *move* (di queste operazione si fa carico l'AMS della piattaforma di partenza);
- se necessario, trasferimento degli assembly richiesti dal dato A*gent Template*, tramite servizio ftp implementato nel modulo di mobilità;
- trasferimento dello stato persistito (lo stato dell'agente contenente tutti gli elementi della *knowledge*, i messaggi, le conversazioni, l'AID e lo stato di ciascun *behavior*), tramite servizio ftp implementato nel modulo di mobilità;
- creazione di un'istanza dell'agente sulla piattaforma di destinazione, ripristino della *knowledge*, creazione degli oggetti *behavior* posti nello stato in cui erano prima della migrazione (di questa operazione si fa carico l'AMS della

piattaforma di destinazione);

- invocazione del metodo *Resume(...)* per consentire al programmatore di personalizzare la riattivazione dell'agente (*Resume* è un metodo dell'Agent Template implementato dal programmatore dell'agente che in questa fase viene invocato dall'AMS);

- rilascio dell'agente e passaggio allo stato *active* attraverso l'azione *execute* (l'AMS attiva l'agente e lo *scheduler* della piattaforma manda in esecuzione i suoi *behaviour*).

E. Aspetti di Sicurezza

Nelle precedenti sezioni abbiamo discusso dei requisiti strutturali necessari ad implementare un'architettura per la mobilità in AgentService. La sicurezza è un ulteriore requisito che si aggiunge a tale architettura al fine di garantire che gli agenti mobili portino avanti le loro attività senza costituire un pericolo per l'ambiente che li ospita. In particolare ciò che si vuole evitare è l'esecuzione arbitraria di codice trasferito attraverso l'infrastruttura che permette la mobilità.

Un primo livello di sicurezza è fornito dallo *Storage* che permette l'esecuzione di solo codice verificato: gli assembly caricati nello *Storage* e quindi anche quelli trasferiti per attuare la mobilità devono essere firmati. La firma di un assembly ci fornisce una prima garanzia sul codice in esso contenuto, in quanto ci permette di riconoscere qualcosa identificato in precedenza. Tale livello di sicurezza può anche non bastare e per tal motivo l'architettura di AgentService permette di personalizzare attraverso la gestione utenti l'insieme dei permessi che sono associati ad un agente, fornendo ad essi solo quei permessi che l'amministratore della piattaforma ritiene necessario conferirgli. La gestione dei profili utente all'interno di AgentService è implementata in un opportuno modulo la cui installazione non è obbligatoria. Le funzionalità avanzate di gestione della sicurezza sono perciò possibili solo in presenza di questo modulo. Questa è una scelta che garantisce la massima flessibilità anche in tale ambito: in contesti in cui non è richiesta una particolare attenzione agli aspetti di sicurezza, l'amministratore della piattaforma può decidere se accettare la mobilità oppure no. Nel caso in cui si voglia abilitare la mobilità, gli agenti ospitati e trasferiti nella piattaforma verranno eseguiti con l'utente predefinito, che disporrà di un set di permessi associati al corrispondente utente del sistema operativo su cui è installata la piattaforma. In presenza del modulo di gestione utenti, l'amministratore potrà decidere se conferire agli agenti trasferiti un predefinito profilo di sicurezza oppure se richiedere che questi siano associati ad un particolare profilo utente presente nel proprio insieme di utenti.

Osserviamo che la gestione della sicurezza attraverso i profili utente è qualcosa che si sovrappone in modo del tutto trasparente al normale ciclo di vita della piattaforma ad agenti: il modello di agente adottato basato sugli Application Domain permette, infatti, di conferire in maniera molto semplice un particolare profilo utente con cui eseguire il codice in esso contenuto. In mancanza di un'esplicita specifica del profilo utente questo viene ereditato dall'Application Domain che lo ha creato.

V. POLITICHE DI BILANCIAMENTO DI CARICO

A. Introduzione

La mobilità degli agenti può essere vantaggiosamente sfruttata per risolvere il problema della distribuzione del carico in un a rete di entità computazionali: i sistemi multi-agente sono in grado di decentralizzare la distribuzione del carico computazionale. Infatti, un'applicazione complessa può essere suddivisa in parti autonome, ognuna delle quali delegata ad un agente mobile. Ogni agente mobile ha il compito di cercare il nodo/piattaforma nella rete a lui più conveniente. Durante l'esecuzione, gli agenti possono spostarsi verso altri nodi dove vi sono risorse computazionali disponibili per poter quindi meglio distribuire il carico.

In alternativa la gestione della distribuzione del carico può essere centralizzata per consentire l'applicazione di politiche/algoritmi che siano in grado di forzare, quando possibile, lo spostamento degli agenti. La scelta di centralizzare le politiche di bilanciamento è volta ad ottimizzare la distribuzione di carico dell'intero sistema piuttosto che ad avvantaggiare il singolo agente.

B. Load Balancing Policy Module

Il bilanciamento del carico in AgentService è gestito dal relativo modulo Load Balancing Policy Module (LBPM). Il modulo fornisce un servizio di federazione di piattaforme per creare un ambiente unico in cui gli agenti sono in grado di muoversi. Per il trasferimento degli agenti, il LBPM sfrutta naturalmente il servizio di mobilità offerto dal relativo modulo. Per implementare correttamente il bilanciamento di carico il modulo ha la necessità di accedere alle informazioni che definiscono il profilo della piattaforma. In particolare deve essere in grado di monitorare il numero degli agenti in esecuzione ed ottenere informazioni pertinenti alle risorse fisiche dell'host. Inoltre, potranno rivelarsi fondamentali al modulo LBPM, ed in particolare alle politiche da esso applicate, le informazioni relative al comportamento a run-time del sistema multi-agente, come ad esempio il numero di messaggi scambiati. La piattaforma è in grado di garantire ad ogni modulo un contesto all'interno del quale è possibile sia reperire informazioni riguardanti il suo profilo, sia registrarsi agli eventi che scandiscono il ciclo di vita del sistema, ad esempio l'attivazione di un agente.

Viste le considerazioni di cui sopra, il modulo LBPM è quindi posto nella condizione di operare sulla mobilità sfruttando tutte le informazioni messe a disposizione dalla piattaforma. Inoltre, collaborando con i moduli LBPM presenti su altre installazioni di AgentService, è in grado di creare un quadro completo della situazione a run-time delle piattaforme coinvolte nel processo di bilanciamento. Queste piattaforme costituiscono, di fatto, una federazione che determina i confini all'interno dei quali sono applicate le politiche di bilanciamento.

Di default, il LBPM fornisce due semplici politiche di bilanciamento orientate alla distribuzione del carico: una politica bilancia il numero di agenti tra le piattaforme, mentre l'altra ha l'obiettivo di spostare nella stessa installazione di AgentService gli agenti che interagiscono più frequentemente. In aggiunta il modulo è progettato per poter applicare nuove politiche di bilanciamento definite dall'utente, caricabile all'interno del modulo attraverso un'architettura a *plug-in*.

C. Processo di federazione delle piattaforme e applicazione delle politiche di bilanciamento

L'applicazione di politiche di balancing richiede, come detto, la formazione di federazioni di piattaforme AgentService. Occorre in pratica che le piattaforme coinvolte siano in grado di conoscere i profili delle piattaforme federate e vi sia un meccanismo centralizzato per lo spostamento degli agenti in funzione della data politica che si vuole applicare. Il sistema di federazione utilizza un modello client/server.

In fase di installazione l'amministratore di sistema deve indicare, attraverso uno specifico file di configurazione del modulo LBPM, la piattaforma AgentService che svolgerà le funzioni di server. A questo punto le altre eventuali installazioni di AgentService possono essere configurate come nodi secondari e fare riferimento alla piattaforma server come nodo primario, comunicando ad essa l'adesione alla federazione, tali comunicazioni avvengo attraverso il servizio di messaggistica normalmente utilizzato dagli agenti. Una volta creata una federazione è possibile per il modulo LBPM del nodo primario (server) richiedere agli altri moduli le informazioni sullo stato delle rispettive piattaforme. Avendo a disposizione i profili delle piattaforme, il LBPM (server) è in grado di applicare correttamente l'algoritmo di balancing prescelto. Tale politica applicata, sulla base dei profili e d'altre eventuali informazioni, determina una differente distribuzione degli agenti tra le piattaforme.

La realizzazione della distribuzione degli agenti è controllata dal modulo ed avviene ciclicamente attraverso le seguenti fasi:

- LBPM server interroga la politica richiedendo il successivo agente da spostare (invoca il metodo GetNextAgentToMove(...) sull'interfaccia ILBP);
- la politica, in base all'algoritmo di bilanciamento, fornisce l'AID dell'agente da trasferire e l'identificativo (PlatformDescription) della piattaforma di destinazione;
- LBPM server comunica al LBPM della piattaforma che ospita l'agente di operare il trasferimento richiesto (la comunicazione, come già detto, avviene attraverso il modulo di messaggistica standard di AgentService);
- LBPM client chiede quindi al locale modulo responsabile del servizio di mobilità l'esecuzione fisica del trasferimento (la comunicazione tra moduli è gestita dalla coda di comandi della piattaforma);
- a trasferimento avvenuto il modulo client notifica al LBPM server il successo dell'operazione;
- LBPM server aggiorna il *profile* delle piattaforme coinvolte ed esegue un nuovo ciclo.

Si noti che il modulo LBPM prevarica il controllo dell'AMS sul ciclo di vita degli agenti. Il modulo, infatti, opera sulla mobilità degli agenti senza effettuare alcuna richiesta all'AMS, elude la fase di contrattazione tra gli AMS. Sebbene questo violi in parte i principi della teoria degli agenti e le direttive FIPA a riguardo, sembra accettabile che per esigenze non strettamente legate alla comunità di agenti ma, ad esempio, alle disponibilità delle risorse hardware, si possa operare sulle infrastrutture degli agenti stessi in maniera quasi trasparente all'AMS. In alternativa il coinvolgimento dell'AMS di ciascuna piattaforma per contrattare il trasferimento degli agenti appesantirebbe eccessivamente il protocollo di distribuzione degli stessi e rallenterebbe le normali attività degli AMS coinvolti.

Per un corretto funzionamento della politica di bilanciamento è opportuno che la configurazione delle piattaforme federate sia tale da non permettere ai singoli agenti di spostarsi autonomamente. Tale vincolo evita eventuali conflitti ed è facilmente imponibile attraverso il file di configurazione dell'installazione di AgentService.

D. Definizione di nuove politiche di bilanciamento del carico

AgentService permette al programmatore di definire nuove politiche di balancing per la distribuzione degli agenti sulle piattaforme. La definizione di una nuova politica risulta un'operazione relativamente semplice: allo sviluppatore non viene richiesto di modificare il modulo, ma solamente di implementare una specifica interfaccia (ILBP) che deve caratterizzare ogni politica di bilanciamento. Tale interfaccia espone i metodi che le consentono di fornire al modulo i trasferimenti da effettuare, di ricevere dallo stesso le informazioni sui profili delle piattaforme federate e di essere notificata degli eventi originatisi nella piattaforma host. Di seguito viene presentata la struttura dell'interfaccia:

```
interface ILBP
{
PolicyDescription GetDescription();
void ConsumeEvent(PlatformEvent evt);
void GetNextAgentToMove(
  out AID aid,
  out PlatformDescription dest);
void AddProfile(PlatformProfile p);
void UpdateProfile(PlatformProfile p);
void RemoveProfile(PlatformProfile p);
}
```

Di particolare interesse è il metodo *GetNextAgentToMove(...)* nel quale viene implementato l'algoritmo di bilanciamento. Eventuali informazioni sui comportamenti a run-time del sistema, l'invio di un messaggio o la creazione di un agente, sono fornite dal modulo di bilanciamento (LBPM) attraverso il metodo *ConsumeEvent(...)*.

La classe che implementa tale interfaccia dovrà essere inserita in un assembly della CLI e quindi aggiunta alle politiche della piattaforma attraverso il relativo file di

configurazione in fase d'installazione della stessa.

Una volta resa disponibile la nuova politica, il relativo algoritmo di balancing potrà essere applicato dall'amministratore della piattaforma della federazione attraverso le procedure già descritte: file d'installazione o interfaccia di programmazione della piattaforma del nodo primario.

E. Test Case

L'infrastruttura per la gestione della mobilità è stata testata con buoni risultati su una federazione di 10 piattaforme AgentService.

Sono stati creati in maniera casuale sulle piattaforme della federazione 100 agenti, la cui attività principale è data dal semplice scambio di messaggi con i *peer* della comunità. Al nodo primario è stata applicata la politica basata sulla limitazione del numero di messaggi interpiattaforma scambiati. Una volta definita la nuova distribuzione ed effettuati i trasferimenti applicati dalla politica di bilanciamento, il numero di messaggi interpiattaforma è diminuito sensibilmente, ed è andato stabilizzandosi anche in funzione di quegli agenti che modificavano il destinatario dei loro messaggi, vedi Tabella I.

L'esito positivo del test relativamente alla politica applicata è naturalmente dipeso dal tipo di attività svolta dagli agenti coinvolti, ma quello che si è voluto attestare è la bontà dell'infrastruttura di mobilità e dei meccanismi di applicazione delle politiche di balancing, non tanto l'efficacia degli algoritmi di balancing stessi. Questi ultimi dovranno essere, in effetti, valutati ed eventualmente ridefiniti in funzione del contesto applicativo della comunità di agenti su cui andranno ad operare.

TABELLA I
ANDAMENTO DEI MESSAGGI INTERPIATTAFORMA SCAMBIATI DURANTE IL TEST

Tempo (secondi)	# Agenti Trasferiti	# Messaggi Interpiattaforma
0-60	0	2412
60-240	32	2157
240-420	9	1123

VI. CONCLUSIONI

L'architettura modulare di AgentService consente la progettazione e l'implementazione di molte funzioni aggiuntive ai normali servizi della piattaforma e l'integrazione con essi. È possibile per tal motivo arricchire la piattaforma con un'infrastruttura che garantisce la mobilita degli agenti implementando tale funzionalità in un modulo. Il modulo della mobilità implementa un servizio di mobilità debole con persistenza dello stato che avviene in maniera del tutto trasparente al programmatore dell'agente. La realizzazione di un'infrastruttura ad agenti mobili più robusta e sicura richiede invece la collaborazione di tale modulo con altri componenti della piattaforma che devono soddisfare alcuni requisiti; in particolare è stata sottolineata l'interazione con il modulo di messaggistica che deve disporre di alcune funzionalità aggiuntive rispetto alle specifiche richieste per tale modulo dalla piattaforma AgentService. Allo scopo di fornire un servizio più sofisticato, il modulo di messaggistica dovrebbe essere in grado di tenere traccia degli agenti che si sono trasferiti e notificare agli agenti residenti nella piattaforma lo spostamento di tale agente. Un modulo di messaggistica che soddisfa i requisiti richiesti dalla mobilità permette inoltre il mantenimento delle conversazioni tra due agenti, anche se uno di essi si trasferisce in un'altra piattaforma; tale operazione viene effettuata in maniera del tutto trasparente agli agenti stessi. Osserviamo che i servizi aggiuntivi richiesti dal modulo che implementa la mobilità sono, di fatto, una violazione della struttura modulare della piattaforma in quanto comportano un debole accoppiamento con il servizio di messaggistica. Occorre però tenere conto del fatto che tali servizi sono richiesti per implementare un'infrastruttura per la mobilità più sofisticata e che le funzionalità di base della mobilità sono garantite indipendentemente dalla presenza o meno di tali servizi. Poiché la scelta di dotare una piattaforma dell'infrastruttura per la mobilità avviene molto spesso in fase di installazione della piattaforma, l'accoppiamento debole con il modulo di messaggistica non costituisce di fatto un problema reale.

La presenza di un'infrastruttura per la mobilità permette di arricchire la piattaforma con servizi sofisticati come ad esempio la gestione del bilanciamento di carico tra piattaforme federate. Tale funzionalità è nuovamente implementata sfruttando i vantaggi dell'architettura modulare di AgentService: un opportuno modulo (LBPM) appoggiandosi al servizio di mobilità, applica algoritmi di bilanciamento del carico per la gestione delle risorse hardware. Le politiche di bilanciamento fornite di default si basano sul numero dei messaggi scambiati e sul numero degli agenti presenti in una piattaforma ma è possibile estendere tali politiche definendo regole personalizzate.

Il processo di trasferimento degli agenti è stato testato con esiti positivi assieme alla funzionalità di bilanciamento del carico. Si è osservato che l'aspetto più oneroso per l'architettura di AgentService è dato dallo spostamento dagli assembly contenenti la definizione dei tipi di agenti (AgentTemplate). Tale trasferimento non avviene sempre, esso dipende dal tipo applicazione, ma in ogni caso dovrebbe essere abbastanza limitato, in quanto normalmente si tende ad avere applicazioni con più agenti dello stesso tipo o perlomeno a riusare i loro elementi base, *knowledge* e *behavior*, vista la modularità del modello di agente adottato da AgentService.

Una limitazione di cui soffre l'infrastruttura per la mobilità qui descritta è la mancanza di interoperabilità con altre piattaforme che non siano installazioni di AgentService, Jade [10] in particolare. La realizzazione di un'infrastruttura per la mobilità tra piattaforme di diversa natura è, di fatto, un aspetto molto difficile da concretizzare in quanto occorre superare i problemi dovuti alla differenza delle tecnologie utilizzate, delle architetture implementate e del modello di agente adottato nelle diverse piattaforme.

RIFERIMENTI

[1] A. Fuggetta, G. P. Picco, G. Vigna. *Understanding Code Mobility*, IEEE Trans. on Software Engineering, Maggio 1998.

[2] Muhammad Kamran Naseem, Sohail Iqbal, Khalid Rashid, *Implementing Strong Code Mobility*, Information Technology Journal 3(2): pp. 188-191, 2004.

[3] R. R Brooks, N. Orr. *A Model for Mobile Code Using Interacting Automata*, IEEE Trans. Mobile Computing 1(4): pp. 313-326, 2002.

[4] Standard ECMA-335: *Common Language Infrastructure (CLI)*, 2nd Edition, Dicembre 2002, ECMA, disponibile presso: http://www.ecma-international.org/publicati-ons/standards/Ecma-335.htm.

[5] Standard ISO/IEC 23271:2003: *Common Language Infrastructure*, 28 Marzo 2003, ISO.

[6] A. Boccalatte, A. Gozzi, A. Grosso, C. Vecchiola. *AgentService*, The Sixteenth International Conference on Software Engineering and Knowledge Engeneering (SEKE'04), Banff Centre, Banff, Alberta, Canada 20-24 Giugno 2004.

[7] FIPA Abstract Architecture Specification, http://www.fipa.org/specs/fipa00001/

[8] H.S. Nwana, D.T. Ndumu, L.C. Lee, *ZEUS: An advanced Tool-Kit for Engineering Distributed Multi-Agent Systems*, in Proceedings of PAAM98, pp. 377-391, London, U.K., 1998.

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

[10] F. Bellifemine, G. Rimassa, A. Poggi, *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.

[11] N. M. Karnik, A. R. Tripathi. *Design Issues in Mobile-Agent Programming Systems*, IEEE Concurrency 6(3): pp. 52-61, Luglio-Settembre 1998.

[12] D. Chess, C. Harrison, A. Kershenbaum. *Mobile Agents: Are they a good idea?*, Technical Report, IBM T.J. Watson Research Center, NY, Marzo 1995.

[13] H. Nwana. *Software agents: An Overview*, Knowledge and Engineering Review, 11(3), Novembre 1996.

[14] G. Cabri, L. Leonardi, F. Zambonelli. *Weak and Strong Mobility in Mobile Agent Applications*, Proceedings of the 2nd International Conference and Exhibition on The Practical Application of Java (PA JAVA 2000), Manchester (UK), Aprile 2000.

[15] D. Lange, M. Oshima. *Programming and Deploying Java Mobile Agents with Aglets*, Addison-Wesley, 1998.

[16] A. Acharya, M. Ranganathan, J. Salz. *Sumatra: A Language for Resourceaware obile Programs*, Mobile Object Systems: Towards the Programmable Internet , J. Vitek and C. Tschudin (Eds.), Springer-Verlag, Lecture Notes in Computer Science No. 1222: pp. 111-130, Aprile 1997.