ALMA MATER STUDIORUM
UNIVERSITA' DEGLI STUDI DI BOLOGNA
SCUOLA DI INGEGNERIA E ARCHITETTURA SEDE DI CESENA
CORSO DI LAUREA MAGISTRALE
IN INGEGNERIA E SCIENZE INFORMATICHE

“Elasticity” in Multi-Agent Systems

Progetto di Sistemi Autonomi

realizzato da

Casadei Richiard

Indice

Abstract

Nella maggior parte dei domini un agente non ha la completa conoscenza dell'ambiente e completo controllo su di esso. L'ambiente può evolvere dinamicamente indipendentemente dall'agente, quindi in generale, si assume che esso sia non-deterministico e questa caratteristica può causare il fallimento delle azioni dell'agente. Tale scenario si presenta come una sorta di aspetto negativo dell'attuale concetto di sistema multi-agente(MAS) e per ovviare a ciò e rendere il sistema nel suo complesso più affidabile, performante ed adattabile a questi cambiamenti si è pensato fosse utile estenderlo inserendo una caratteristica già nota nel Cloud Computing, l'elasticità.

Dopo aver analizzato gli attuali MAS, cercheremo quindi di dare una definizione esaustiva di elasticità e di vedere come essa possa essere integrata e resa una caratteristica di base anche negli attuali MAS, non dimenticandoci dei MAS che fanno uso di mezzi di coordinazione per l'interazione tra gli agenti e le risorse e nei quali sarà necessario introdurre un concetto di elasticità abbinata alla coordinazione(coordination-aware elasticity).

Capitolo 1

MAS e Agenti

Il paradigma ad agenti utilizzato in sistemi computazionali noti sotto il nome di Multi-Agent System MAS (sistema multi-agente) rappresentano l'oggetto di studio di una nuova area di ricerca che riunisce ed integra alcune nozioni di intelligenza artificiale con aspetti tipici dei sistemi distribuiti, al fine di sviluppare modelli e architetture per agenti intelligenti. A causa della sua diffusione, risulta difficile dare una definizione universale del concetto di agente e riuscire a coglierne le caratteristiche principali. In letteratura alcune tra le più note definizioni presentano un agente come:

> *"An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors." [Russell and Norving, 1995]*
> *(Un agente è tutto ciò che può essere visto come ciò che percepisce il suo ambiente attraverso sensori e che agisce su tale ambiente attraverso degli attuatori.)*

> *"Autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed." [Maes, 1995]*
>
> *(Agenti autonomi sono sistemi computazionali che abitano alcuni ambienti dinamici e complessi, sentono e agiscono autonomamente in questo ambiente, e così facendo realizzano una serie di obiettivi o compiti per i quali sono stati progettati.)*

"An agent is an encapsulated computer system situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives." [Wooldridge, 1997]

(Un agente è un sistema informatico incapsulato situato in un qualche ambiente ed è in grado di adattarsi, eseguire azioni autonome in tale ambiente, al fine di conseguire gli obiettivi di progettazione.)

Cercando di darne una definizione generale in cui siano presenti tutti gli aspetti introdotti in ciascuna definizione, un agente può quindi essere considerato come una particolare entità software che vive in ambienti dinamici e complessi, capace di agire autonomamente, che soddisfa i propri obiettivi eseguendo i compiti per cui è stato progettato ed è in possesso della capacità di interagire con l'ambiente circostante scambiando informazioni con altri agenti o con altri utenti. Da questa definizione si possono evidenziare le principali caratteristiche di un agente[1][2]:

- autonomia: un agente opera senza interventi diretti da parte dell'utente o di altri componenti ed esercita un controllo sia sul proprio stato interno che sul proprio comportamento
- reattività: un agente percepisce l'ambiente circostante mediante sensori e reagisce ai cambiamenti che avvengono in esso
- proattività: un agente presenta comportamenti volti al raggiungimento dei propri obiettivi ed è in grado di prendere decisioni, senza limitarsi ad attendere passivamente le variazioni dell'ambiente circostante;
- abilità sociale: gli agenti sono in grado di interagire tra loro mediante particolari linguaggi di comunicazione, ed il meccanismo normalmente usato è quello a scambio di messaggi

Le caratteristiche di reattività e proattività non devono essere considerate necessariamente in conflitto tra loro in quanto un agente può essere infatti in grado sia di reagire ai cambiamenti dell'ambiente, sia di decidere autonomamente il comportamento da assumere per perseguire i propri obiettivi. Alcuni agenti possono possedere anche caratteristiche di mobilità, ovvero la capacità di spostarsi tra i nodi di una rete e l'apprendimento, ossia la capacità di imparare per migliorare le proprie prestazioni, acquisendo conoscenza mediante l'interazione con l'ambiente circostante. A tal proposito spesso gli agenti sono inoltre modellati per mezzo di concetti cognitivi basati su stati mentali come credenze (beliefs), conoscenze (knowledge), desideri e intenzioni[2].

Ciascuna di queste caratteristiche potrebbe rappresentare una dimensione di classificazione degli agenti software esistenti e proprio per questo motivo, come per quanto già avvenuto con la loro definizione, anche la stesura di una tipologia di agenti risulta abbastanza complessa. Possono essere identificate classi di agenti in base alla loro mobilità, alla caratteristica di reattività o proattività, in base a diversi attributi ideali e primari che dovrebbero esibire oppure secondo i loro ruoli, e queste dimensioni di classificazione sono solo alcune tra quelle che si possono prendere in esame, quindi le classi di agenti che ne derivano non possono essere considerate definitive. In linea di principio combinando tutte queste dimensioni si deduce che esistono agenti in uno spazio veramente multidimensionale, ma per motivi di

chiarezza e comprensione possiamo ridurre questo spazio in un elenco ottenendo così una sorta di classificazione generale delle tipologie di agenti esistenti[2]. Tale elenco prevede:

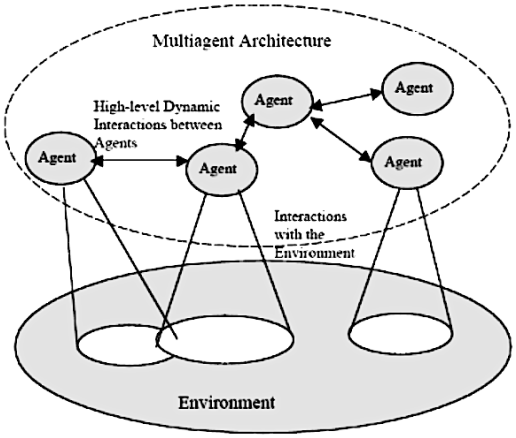
- Agenti collaborativi, agenti autonomi, cooperativi, reattivi e proattivi che possono non essere dotati di capacità di apprendimento. Di conseguenza sono agenti capaci di agire in modo razionale senza guide esterne in sistemi multi-agente aperti e con vincoli temporali
- Agenti interfacce, agenti che supportano l'utente durante la sua attività. Questi agenti intraprendono la loro attività basandosi soprattutto su autonomia e capacità di apprendimento
- Agenti mobili, agenti in grado di spostarsi (migrare) da un computer ad un altro attraverso una rete durante la loro stessa esecuzione
- Agenti informazione/internet, definiti in base all"attività che compiono piuttosto che le loro principali caratteristiche. Il loro scopo è quello di gestire, trattare e raccogliere informazioni provenienti da una varietà di fonti diverse ed eterogenee
- Agenti reattivi, entità semplici ed in grado di comunicare solo in maniera elementare, secondo il modello richiesta/risposta, in quanto non dispongono di una rappresentazione interna dell'ambiente
- Agenti ibridi, agente che combina due o più tra le filosofie descritte e che nascono dalla convinzione che per determinate applicazioni i vantaggi derivanti da un agente ibrido sono maggiori di quelli ottenibili da una singola tipologia
- Agenti intelligenti, agenti dotati di autonomia, capacità di collaborazione e di apprendimento. L'agente esegue i propri compiti e raggiunge i propri obiettivi in modo da ottimizzare un determinato indicatore di performance

Ci sono alcune applicazioni che combinano gli agenti di due o più di queste categorie, e ci riferiamo a questi come i sistemi di agenti eterogenei, il cui obiettivo è l'interoperabilità tra differenti moduli software mediante apposito linguaggio di comunicazione

MAS, environment e società

Un MAS non può essere semplicemente ricondotto alla mera somma di entità interagenti[1], ma è prevista esplicitamente l'analisi della componente ambientale in cui il MAS e gli agenti che lo compongono sono situati e quella della società creata dal gruppo di agenti che interagiscono. L'analisi dell'ambiente implica l'individuazione delle sue caratteristiche di base, delle risorse presenti ed il modo attraverso il quale gli agenti possono interagire con esse, mentre l'analisi della società implica l'individuazione delle regole che disciplinano e fanno evolvere il MAS e dei vari ruoli ricoperti dai vari agenti. Gli agenti possiedono solo una rappresentazione parziale dell'ambiente in cui sono situati e che rende loro possibile effettuare soltanto percezioni ed operazioni di modifica limitate, hanno quindi necessità di interagire tra loro e così facendo costituiscono una vera e propria società e l'interazione al suo interno è guidata da opportune regole sociali che guidano le singole entità verso il

raggiungimento degli obiettivi globali. Una società quindi non può essere costruita come semplice combinazione di agenti progettati separatamente, ma deve essere pensata come astrazione di prima classe per essere utilizzata nell'analisi, nel progetto e nello sviluppo dei sistemi software complessi, mediante l'individuazione delle strutture sociali, delle leggi sociali, e di come tali regole possano essere progettate e fatte rispettare. Allo stesso tempo però, la progettazione di un MAS non può prescindere dalla modellazione dell'ambiente in cui le entità coinvolte si trovano ad operare. In generale gli agenti e la loro società vivono in ambienti che possono essere eterogenei, dinamici, aperti, distribuiti ed imprevedibili e tali proprietà ovviamente condizionano il modo con cui gli agenti stessi rappresentano il mondo circostante, le decisioni ed il modo in cui cercano di raggiungere i propri obiettivi. Inoltre, le caratteristiche dell'ambiente spesso non sono note a priori, ma possono essere parzialmente definite a seconda delle esigenze del sistema.
Di seguito è mostrato in figura la rappresentazione di una architettura MAS[1].



MAS, comunicazione e coordinazione

Nei MAS gli agenti possono interagire scambiandosi informazioni mentre cooperano nel raggiungimento di obiettivi comuni. Il meccanismo di comunicazione normalmente usato è quello dello scambio di messaggi. L'invio o la ricezione di un messaggio può essere visto come un'azione eseguita dall'agente e avviene principalmente come passaggio asincrono. Ogni agente possiede una coda di messaggi entranti e l'effettiva lettura dei messaggi è ad arbitrio dell'agente, esso infatti può leggere il primo messaggio in coda oppure il primo messaggio che soddisfa certi requisiti. A tal proposito sono stati definiti dei Linguaggi di Comunicazione fra Agenti in cui le azioni comunicative sono modellate basandosi sulla teoria filosofica delle azioni.

Gli agenti possono, inoltre, competere per accedere alle medesime risorse, o ancora possono interferire l'un con l'altro perseguendo i propri obiettivi. Al fine di gestire tali interazioni e dipendenze tra le attività messe in gioco all'interno del sistema è stata introdotto il concetto di

coordinazione e l'adozione di un modello di coordinazione. La coordinazione risulta quindi un concetto fondamentale nella programmazione agent-oriented, poiché permette la gestione delle dipendenze tra le attività degli agenti in modo da garantire che tutti si comportano secondo gli obiettivi del sistema nel suo complesso, ed un modello di coordinazione consiste in un framework formale che consente di esprimere le interazioni tra agenti, società ed ambiente in accordo a determinate regole. Ogni modello di coordinazione è costruita su tre principali elementi:

- un mezzo di coordinazione, l'astrazione che abilita le interazioni tra gli agenti
- i coordinati, le entità le cui mutue interazioni sono governate dal modello
- le leggi di coordinazione, hanno lo scopo di consentire sia la delega di gestione delle interazioni di middleware e la separazione del comportamento interattivo da quello computazionale. Possono essere definite attraverso l'ausilio di un linguaggio di comunicazione, cioè una sintassi utilizzata per rappresentare e comunicare le strutture dati e un linguaggio di coordinazione, ovvero l'insieme di primitive messe a disposizione degli agenti dall'infrastruttura.

Capitolo 2

Elasticità

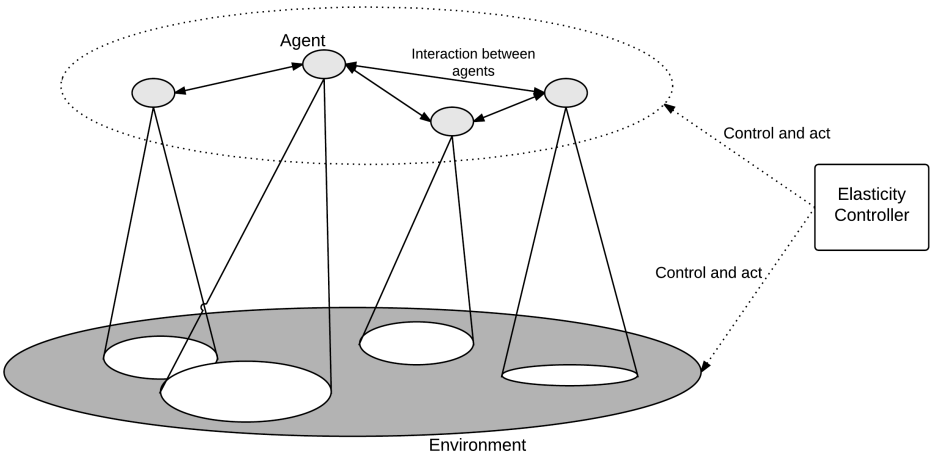
In informatica, il termine elasticità è stato recentemente utilizzato come sinonimo accademico di Cloud Computing. Questa definizione non può però considerarsi veritiera in quanto l'elasticità corrisponde solamente ad una delle tante caratteristiche che il Cloud Computing offre.

Esistono diverse definizioni di elasticità[3], tutte dipendenti dal contesto in cui essa è applicata, ad esempio in fisica abbiamo che "l'elasticità è la proprietà fisica di un materiale di deformarsi sotto stress (per esempio, forze esterne) e di ritornare alla sua forma originale quando lo stress viene rimosso". Quando viene applicata all'informatica, l'elasticità riflette la naturale capacità di un sistema di espandersi o contrarsi in base a determinati fattori ed in base alle necessità. Un altra definizione è fornite in economia, dove la si descrive come "il rapporto tra la variazione percentuale in una variabile e la variazione percentuale in un'altra variabile", ovvero la misura della reattività di una funzione o della sensibilità alle variazioni dei parametri in modo relativo. Questa affermazione ci permette quindi di sostenere che l'elasticità è un modo efficace per misurare la domanda e fornire la risposta a tale domanda. Tali nozioni, in aggiunta a tutte le altre esistenti, fungono da base per l'implementazione di tutti i meccanismi di elasticità e scalabilità applicati alle risorse di un sistema Cloud, che hanno il compito di soddisfare le richieste dinamiche di risorse e fornire un alta qualità di servizio(QoS) all'interno dei servizi offerti. Quest'ultima misura la reattività di risposta ad un cambiamento nell'utilizzo delle risorse e l'algoritmo che risiede al di sotto del servizio dell'applicativo richiede che il miglioramento della qualità del servizio sia proporzionale al consumo della risorsa richiesta. In altre parole, più risorse, tra quelle disponibili, vengono consumate, migliore è la qualità ottenibile. Un sistema o processo che gode della proprietà di elasticità deve decidere come utilizzare le risorse esistenti nel suo ambiente in modo ottimale (in modo da soddisfare tutte le esigenze multidimensionali richieste dal sistema, ma con un vantaggio massimo). L'ambiente quindi risulta dinamico, e costituito da tipi diversi di risorse computazionali, dati e risorse di rete.

I principi di funzionamento di base di un sistema elastico sono la sua capacità di monitorare, gestire e descrivere le sue proprietà in modo dinamico e agire in modo autonomo sulle funzioni di processo e computazione che hanno impatto sulla qualità. L'elasticità quindi fornisce più autonomia alle infrastrutture e ai processi stessi ed ogni consumatore, processo o utente che ne vuole utilizzare i meccanismi deve definirne metriche di monitoraggio, regole da seguire e preferenze sulle azioni correttive da intraprendere. L'inserimento di operazioni autonome di elasticità non devono però intaccare la gestione del sistema poiché gli utenti devono essere ancora in grado di controllare il comportamento del sistema con interfacce semplici ed intuitive[4].

Elasticità nei MAS

L'inserimento della nozione di elasticità all'interno di un sistema MAS porterebbe numerosi vantaggi come una maggiore affidabilità nel sistema, scalabilità delle risorse, degli agenti e dei rispettivi canali di comunicazione, una maggiore efficienza e fault tolerance dovuta all'ottimizzazione dei processi che essa comporta. L'ottimizzazione del sistema porterebbe a quest'ultimo anche la proprietà di load balancing, ed essa dovrà essere supportata da una nuova entità, l'elasticity controller, ovvero un'entità di monitoring di tutte le entità presenti all'interno del sistema che ricopre il ruolo di manager delle risorse e di colui che deve far applicare le azioni suggerite dalle strategie di gestione, precedentemente definite. Questa nuova entità, oltre ad apportare i sopracitati vantaggi permetterà di avere come risultato la separazione del controllo dell'elasticità dalla logica presente nel sistema. All'interno dello stesso sistema, data la natura e la dimensione di quest'ultimo, potrebbe esser necessario predisporre più elasticity controller, magari fornendogli una sorta di ruolo in specifico(manager delle interazioni tra gli agenti, manager delle risorse dell'ambiente). La possibile presenza di più entità elasticity controller con ruoli differenti ci permette di affermare inoltre che, a livello concettuale e di implementazione, sarebbe possibile fornire una ipotetica organizzazione gerarchica di questa entità.



Come accade nel Cloud anche nei MAS le azioni di monitoring e le strategie di gestione dovranno essere opportunamente decise dal progettista del sistema, il quale dovrà sviluppare un sistema di ottimizzazione che a run-time sappia controllare la dinamica delle risorse ed in caso di raggiungimento di determinate soglie di controllo agisca in modo tale da conservare un alto livello di prestazioni del sistema e qualità del servizio.

Le possibili applicazioni dell'elasticità in un MAS riguarderebbero le principali entità coinvolte nel sistema, ovvero gli agenti, le risorse presenti nell'ambiente ed il mezzo di comunicazione degli agenti. Per quanto riguarda gli agenti si è pensato che le attività da monitorare sarebbero quelle che riguardano le interazioni e la comunicazione con gli altri agenti ed il carico di operazioni che essi devono eseguire. E' possibile infatti che durante l'utilizzo del meccanismo di comunicazione per scambio di messaggi, si presenti la situazione

in cui un agente abbia la propria "mailbox" piena e non possa accettare altre richieste di operazioni di cooperazione e messaggi da parte di altri agenti che necessitano di comunicare con lui. Per ovviare a ciò si è pensato ad una forma di elasticità che preveda due possibili soluzioni:

- l'elasticity controller dopo aver monitorato la situazione ed essersi reso conto che una mailbox è piena, può in modo dinamico e a run-time aumentarne la capacità permettendo la ricezione di ulteriori messaggi
- l'elasticity controller può replicare l'agente che possiede la mailbox piena e quindi possiede un notevole carico di operazioni da eseguire, inizializzando il nuovo agente, lasciandolo privo di messaggi e successivamente dirottare i nuovi messaggi al clone. Nel momento in cui uno dei due agenti ritornerà ad avere mailbox vuota, l'elasticity controller provvederà a sospenderlo o ad eliminarlo per evitare di avere un esubero di agenti inattivi nel sistema.

Entrambe le soluzioni ottimizzerebbero il funzionamento dell'architettura multi-agente presente nel sistema diminuendo il numero di messaggi e richieste perse a causa di mailbox piene ed eventuali latenze nell'esecuzione di operazioni da parte degli agenti.

Per quanto riguarda l'elasticità applicata alle risorse presenti nell'ambiente, l'elasticity controller incaricato della loro gestione dovrebbe costantemente monitorare, controllare e ottimizzare le risorse usate, mitigando l'utilizzo delle loro potenzialità sulla base del servizio richiesto per il completamento di operazioni da parte degli agenti(storage, larghezza di banda, CPU, RAM utilizzata, ecc). Così facendo le risorse possono essere monitorate, controllate con una garanzia di trasparenza ed in modo autonomo. Per ottimizzare l'utilizzo delle risorse l'elasticity controller potrebbe ricorrere alla virtualizzazione di quest'ultime, ovvero rendendo disponibili, in caso di necessità, un insieme di hardware e software in maniera virtuale con la capacità di operare contemporaneamente a diverse istanze pur in presenza di una singola macchina o di un solo sistema operativo. L'utilizzo della virtualizzazione riduce la complessità dei sistemi, poiché standardizza la piattaforma hardware e ne migliora la disponibilità così da consentirne un rapido ed elastico rilascio. L'elasticity controller dovrà quindi poter accedere a metodi dedicati alla creazione di istanze virtuali delle infrastrutture presenti nell'ambiente forniti dal provider delle infrastrutture fisiche che ne faranno da host.

Queste soluzioni sono state pensate per sistemi MAS in cui gli agenti cooperano ed interagiscono senza che si presentino interferenze o situazioni di competizione e concorrenza per una stessa risorsa, e che quindi sfruttano come mezzo di comunicazione ed interazione il solo scambio di messaggi. Questa configurazione rispecchia la architettura più semplice per i MAS e per questo si è cercato di fornire alcune soluzioni basilari per quanto riguarda l'inserimento del concetto di elasticità in questo tipo di sistemi.

Per quanto riguarda l'elasticità all'interno di MAS, in cui l'utilizzo di mezzi di coordinazione è imprescindibile o anche in MAS ibridi si rimanda la trattazione al prossimo capitolo, poiché si pensa sia necessario affiancarla alla nozione di coordinazione.

Esempi di elasticità in sistemi/framework MAS

Dopo aver affiancato il concetto di elasticità a quello di MAS e successivamente fornito un semplice modello concettuale di come potrebbero essere rappresentati i nuovi sistemi in cui esso è stato inserito, ora analizzeremo alcuni degli attuali sistemi/framework MAS cercando di individuare dove essi possano ereditare le caratteristiche di elasticità e come ciò possa esser concettualmente sviluppato.

JADE

JADE (Java Agent DEvelopment Framework)[1] è un framework Java per sviluppare applicazioni basate su agenti in conformità con le specifiche FIPA[2] per i sistemi interoperabili, intelligenti e multi-agente. Agisce come un middleware agent-oriented e fornisce una piattaforma agent FIPA e prende in carica tutti quegli aspetti applicativi indipendenti, come la gestione del ciclo di vita dell'agente, le comunicazioni, la trasparenza di distribuzione e altri, necessari per implementare un MAS[5]. Fornisce inoltre agli sviluppatori Java una serie di API per poter implementare le proprie personalizzazioni.

Un singolo sistema (logico) JADE può essere suddiviso tra i diversi host in rete e offre una interfaccia ed un servizio di message passing trasparente e distribuito, un servizio di naming distribuito, una piattaforma interna di mobilità agenti (codice e contesto, in una certa misura) di debug e monitoraggio con strumenti grafici. Ogni host agisce come un contenitore di agenti, cioè, fornisce un ambiente completo di runtime per l'esecuzione, la gestione del ciclo di vita, il passaggio dei messaggi degli agenti JADE. Uno di questi contenitori è il contenitore principale (in realtà, il primo inizializzato ed avviato), responsabile di mantenere un registro di tutti gli altri contenitori della stessa piattaforma attraverso la quale gli agenti possono prendere conoscenza di quali altri agenti sono presenti. Tale architettura, quindi, promuove un'interpretazione P2P di un MAS[5].

Per una data piattaforma JADE esiste un sistema distribuito di message passing, chiamato Agent Communication Channel(canale di comunicazione agente, ACC). Esso controlla tutto lo scambio di messaggi all'interno della piattaforma, sia che sia locale o remoto, implementa tutti i servizi necessari per fornire asincronia nelle comunicazioni, gestisce tutti gli aspetti FIPA ACL (Agent Communication Language) ed il formato dei messaggi, come ad esempio la serializzazione e deserializzazione. Secondo le specifiche FIPA, gli agenti JADE comunicano via passaggio asincrono di messaggi ed ogni agente possiede una coda di messaggi (una specie di casella di posta), dove il JADE ACC fornisce messaggi ACL inviati da altri agenti. Ogni volta che un nuovo messaggio si aggiunge alla mailbox, l'agente ricevente viene notificato senza necessità di blocco o di polling e l'effettiva elaborazione di un messaggio è decisa dall'agente stesso in modo autonomo.

[1] http://jade.tilab.com/

[2] http://www.fipa.org/

JADE presenta un tool di gestione, il Remote Monitoring Agent(RMA), nel quale sono racchiuse le seguenti funzionalità:

- Monitoring e controllo della piattaforma e di tutti i suoi contenitori remoti
- Gestione remota del ciclo di vita degli agenti (creazione, sospensione, il ripristino, l'uccisione, la migrazione e la clonazione)
- Composizione ed invio di messaggi personalizzati ad un agente
- Avvio di altri tool grafici
- Monitoring (solo operazioni di lettura) delle altre piattaforme

Proprio l'utilizzo di queste funzionalità, svolte in maniera automatica, rappresenterebbe una buona implementazione del concetto di elasticità all'interno del framework. Viste le considerazioni fatte nel paragrafo precedente l'applicazione del concetto di elasticità potrebbe esser realizzata inserendo un entità/agente che si occupi di eseguire in maniera automatica le funzionalità di monitoring e di controllo della piattaforma, dei suoi contenitori e degli agenti, affiancandogli la gestione del ciclo di vita di quest'ultimi, in particolare della clonazione e uccisione. Proprio queste due ultime funzionalità potrebbero essere applicate se il monitoring degli agenti fosse incentrato sulla capacità delle relative mailbox. Quando si presenta la situazione in cui una mailbox risulta occupata per un dato periodo, questa nuova entità/agente si occuperà di clonare l'agente con la mailbox piena lasciandolo privo di messaggi. Sarà poi compito dell'ACC dirottare i messaggi al nuovo agente clonato, fornendo così al framework una sorta di proprietà di load balancing. Nel momento in cui uno dei due agenti ritornerà ad avere la mailbox vuota, la nuova entità/agente provvederà all'uccisione di quest'ultimo evitando un eventuale esubero di agenti inattivi. In questo modo forniremo al framework la capacità di adattarsi a situazioni di mailbox piene evitando che richieste di cooperazione e messaggi provenienti da altri agenti vadano persi, migliorando quindi l'efficienza e l'affidabilità del framework. Inoltre tutti i sistemi che utilizzeranno tale framework ne potranno ereditare le proprietà diventando, dal punto di vista della gestione degli agenti, anch'essi elastici. Dal punto di vista della piattaforma e dei suoi contenitori, il monitoring dovrebbe focalizzarsi sulle risorse utilizzate e su quelle metriche che più vanno ad influire sulle performance del sistema nel suo complesso. Da test sulle performance[6] e altri test più recenti svolti dal team di sviluppo di JADE si evince che le maggiori metriche utilizzate per quantificare il "punto di calo delle performance" sono rappresentate dal consumo di memoria, determinata dai metodi `tolalMemory` e `freeMemory` della classe `java.lang.Runtime`, dal numero di thread presenti e dal numero dei container all'interno del container principale. Queste stesse metriche potrebbero essere utilizzate come parametri da controllare costantemente in uno sviluppo del concetto di elasticità esteso anche alla piattaforma e non solo agli agenti. In questo caso l'entità/agente incaricata del controllo della piattaforma dovrà costantemente monitorare tali parametri ed in caso di superamento di determinate soglie, decise in fase di sviluppo o deducibili dai test precedentemente citati, applicherà alcune strategie di elasticità mirate al miglioramento del servizio e delle performance. Si è pensato che una semplice soluzione, applicabile come strategia, possa

essere identificata da una funzione già ampiamente sviluppata nell'attuale framework JADE, ovvero la migrazione degli agenti. Tramite questa funzione infatti l'entità/agente controller potrà distribuire il carico dei vari container, diminuendo così la memoria consumata in quelli in cui è presente un maggior numero di agenti e, rispettando i vincoli dei parametri, potrà conservare un alto livello di performance in quelli in cui verrano migrati gli agenti. Nella situazione in cui però tutti i container siano saturi di agenti e quindi la sola migrazione non possa essere presa in considerazione come unica strategia da applicare, l'entità/agente controller dovrà poter crearne dei nuovi e migrare in modo equilibrato gli agenti presenti nei vari container in quelli nuovi. Non sempre anche la soluzione della creazione di uno o più nuovi container può essere applicata, pertanto una ulteriore strategia applicabile sarebbe quella di sospendere l'esecuzione di alcuni agenti, magari in base ad una scala di priorità delle attività in esecuzione in quel momento.

Il framework JADE con le soluzioni concettuali presentate potrà a tutti gli effetti diventare elastico sia dal punto di vista degli agenti sia da parte di tutte le piattaforme che lo compongono, acquisendo così una proprietà di elasticità totale.

ELDA

Il modello Event-driven Lightweight Distilled StateCharts-based Agent (ELDA) si basa sugli stessi principi fondamentali dei modelli ad agente ma consente inoltre una progettazione più efficace attraverso la specifica basata su diagrammi di stato *(i)* del comportamento dell'agente, *(ii)* più spazi di coordinazione per agenti interni in locale/remoto ed interazioni e altri agente o altri componenti, e *(iii)* una mobilità strong dell'agente[7].

I modelli comportamentali e di mobilità, presenti in ELDA sono basate principalmente sulle caratteristiche derivate dal modello MAO (Mobile Active Object), che consente un approccio multi paradigma alla costruzione di applicazioni distribuite in ambienti distribuiti altamente dinamici. In particolare, la modellazione del comportamento dell'agente si basa sul formalismo Distilled StateCharts (DSC) e la migrazione si basa su un modello di mobilità forte. Il modello di interazione è un'estensione del modello MC (Multi Coordination), che si basa su eventi di alto livello che unificano l'accesso e lo sfruttamento degli spazi coordinazione sottostanti e di eventuali risorse del Server Agent.

L'agente ELDA si basa sul concetto di agente leggero event-driven, un'entità a thread singolo autonomo che interagisce attraverso eventi asincroni, esegue a seguito di reaction ed è in grado di migrare. In particolare, un agente leggero event-driven è rappresentato dalla tupla: <Id, Beh, DS, TC, EQ>, dove, Id è l'identificatore univoco dell'agente, Beh è il comportamento dell'agente, DS è lo spazio dati o conoscenza del mondo dell'agente, TC è il singolo thread di controllo a sostegno dell'esecuzione dell'agente, e EQ è la coda di eventi in entrata rivolti all'agente[7].

Il modello ELDA si basa e divide su modelli comportamentali, di interazione e di mobilità. Il modello comportamentale permette di specificare il comportamento dell'agente attraverso la definizione di stati, transizioni tra gli stati e reazioni degli agenti, ovvero azioni atomiche

conseguenti alle transizioni. In particolare, il comportamento dell'agente è definito in modo tale da reagire ad un insieme specifico di eventi e tale reazione può produrre calcoli/ computazioni, la generazione di uno o più eventi, o una migrazione. Il modello di interazione, che si basa su eventi asincroni, consente una forma di multi-coordinazione tra i vari agenti e tra gli agenti e gli altri componenti del sistema mediante lo sfruttamento di molteplici strutture di coordinamento. Il modello di mobilità è basato su un modello che permette una migrazione forte e trasparente dell'agente(sia autonoma e passiva) ed una semplice programmazione dei punti di migrazione[7].

Il comportamento degli agenti Elda è specificato tramite le Distilled StateChart DSC e forgiato secondo una versione estesa del ciclo di vita del modello FIPA, mentre le interazioni si basano su eventi che formalizzano sia quelli di natura interna sia le richieste o notifiche dal server-agent locale. Esistono quattro principali categorie di eventi:

- Interni, generati dagli agenti per guidare in modo proattivo il loro comportamento che, una volta generati, vengono inseriti nella coda eventi dell'agente generatore
- Di gestione, comprendono le richieste verso il server-agent locale e le notifiche provenienti da quest'ultimo e sono ulteriormente classificati con riferimento alle seguenti funzionalità/ servizi: gestione del ciclo di vita dell'agente, impostazione di timer e accesso alle risorse.
  Gli eventi di gestione del ciclo di vita dell'agente consentono di sfruttare funzionalità adibite alla creazione, alla clonazione, alla migrazione, alla sospensione e alla distruzione dell'agente. Gli eventi di impostazione di timer consentono di temporizzare le attività dell'agente. Gli eventi di accesso alle risorse consentono di accedere alle risorse del server-agent, come file, console, database e sensori/attuatori.
- Di coordinazione, consentono atti di coordinazione tra gli agenti e tra agenti e altri componenti secondo un modello di coordinazione specifico. I modelli di coordinazione inter-agent presi in considerazione sono il diretto(attraverso scambio di messaggi in modo sincrono e asincrono), quello basato su tuple, quelli riguardanti la pubblicazione/ sottoscrizione basati su eventi, mentre quelli considerati per l'interazione dei componenti agente/non-agente sono il modello generale RMI Object ed il Web Service. La coordinazione basata su più modelli può facilitare la progettazione di applicazioni, migliorare l'efficienza e consentire l'adattabilità in ambienti dinamici ed eterogenei, in quanto consente agli agenti di scegliere tra una varietà di diversi spazi di coordinazione e modelli che meglio si adattano alle loro esigenze di comunicazione e di sincronizzazione dinamica.
- Di eccezione, modellati come IN-events che vengono inviati dal server-agent locale per notificare l'impossibilità di eseguire un servizio che è stato richiesto.

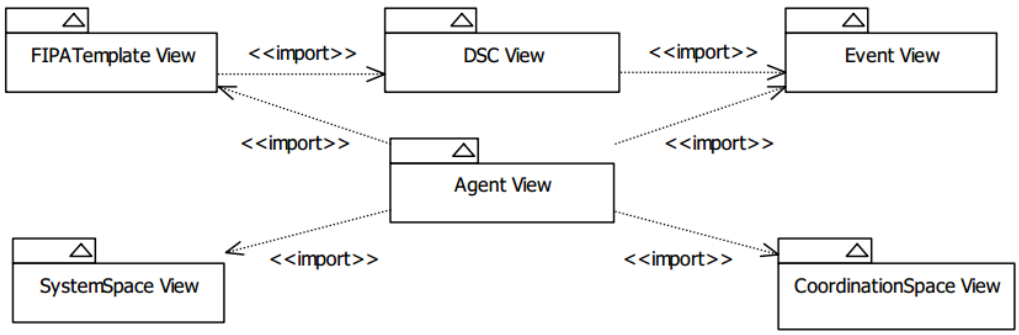
Gli eventi sono inoltre classificati in OUT-events, generati dall'agente e diretti sempre verso il server-agent locale, e gli IN-events che vengono generati dal server-agent locale e consegnati agli agenti di destinazione.

Il modello di mobilità degli agenti ELDA si basa su un modello di mobilità forte che consente di mantenere lo stato di esecuzione dell'agente. La migrazione, definita secondo il modello

FIPA, può essere autonoma (cioè attivato dall'agente stesso) o passiva (cioè imposto dal sistema o indotta da altri agenti). In particolare, i punti di migrazione sono conosciuti dall'agente per la migrazione autonoma poiché sono specificati nel comportamento dell'agente attraverso una definizione appropriata di stati, eventi e transizioni, mentre nel caso di migrazione passiva, i punti di migrazione non sono noti in anticipo poiché esse sono indotti da altri agenti o dal sistema.

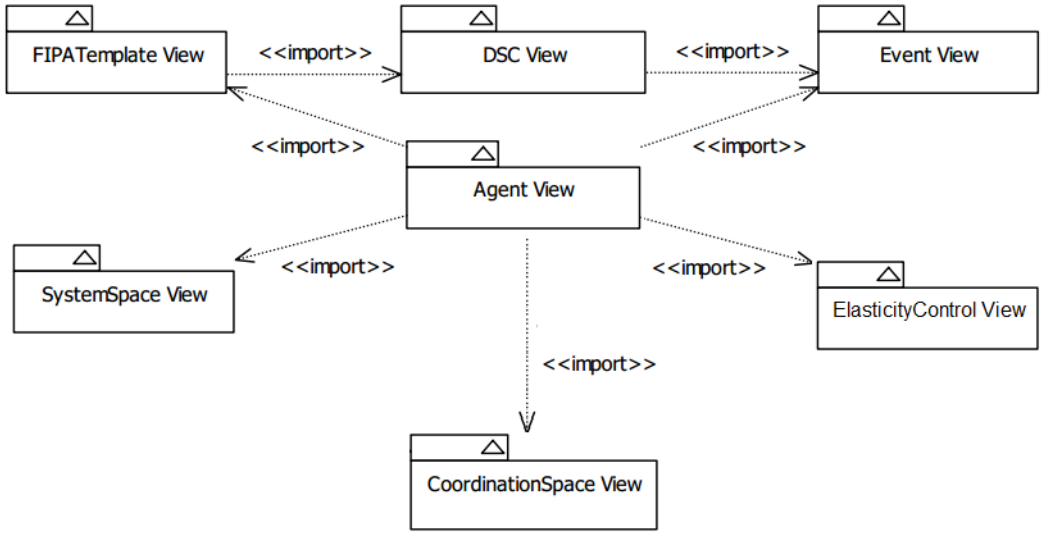
I sistemi multi-agente MAS, sulla base del modello di ELDA, possono essere progettati con il meta-modello di ELDA che fornisce tutti gli elementi di modellazione necessari per la fase di progettazione. Questo meta-modello è organizzato in sei viste correlate come mostrato in figura che corrispondono a[7]:

- Agent View, che rappresenta la struttura di un agente ELDA ed i suoi rapporti con gli spazi di coordinazione e di sistema.
- Event View, che rappresenta la struttura degli eventi.
- SystemSpace View, rappresenta la struttura del spazio di sistema.
- CoordinationSpace View, che rappresenta la gerarchia degli spazi di coordinazione.
- DSC View, che rappresenta la struttura di un DSC.
- FIPATemplate View, che rappresenta la struttura del modello FIPA del comportamento ELDA.



Siccome il meta-modello fornisce tutti gli elementi necessari per la modellazione di un sistema che utilizza il framework ELDA si è deciso di partire proprio da quest'ultimo per inserire il concetto di elasticità ed estendere le funzionalità offerte dal framework stesso. Seguendo questa direzione si è deciso di inserire nel modello una nuova vista correlata alle altre chiamata ElasticControl View, la quale rappresenta la struttura dell'entità/agente che avrà il compito di monitorare il sistema e le strategie di elasticità da attuare. Nel meta-modello "esteso"(figura che segue) l'Agent View mostra che un agente ELDA potrà interagire con lo spazio di sistema, il quale fornisce servizi di sistema, attraverso il ManagementOUT e ManagementIN events, con lo Spazio di coordinazione attraverso gli eventi CoordinationOUT e CoordinationIN e con la ElasticityControl View, dove ricaverà le

strategie di controllo e gestione dell'elasticità e attraverso la quale invierà gli Elasticity events. Tutti questi eventi insieme a quelli interni e a quelli di eccezione, sono inclusi nella Event View.



L'entità/agente avrà il compito di controllare costantemente gli elementi presenti nel sistema, verificare che i parametri e le metriche di controllo rispettino le specifiche ed in caso di generare i relativi eventi di elasticità. Come per JADE, anche per ELDA si crede che gli elementi da monitorare siano gli agenti e la relativa coda, in questo caso di eventi, le risorse che li ospitano e le risorse che gli agenti utilizzeranno. Quando si presenta la situazione in cui una coda eventi di un agente risulta occupata per un dato periodo, l'entità/agente ElasticityController, notificherà tramite un OUT-events il server-agent locale che a sua volta invierà all'agente target un evento di tipo IN-events di gestione implementato appositamente chiamato ad esempio EXAPANDQUEUE. Una volta terminata l'azione di elasticità l'agente dovrà segnalare il completamento dell'operazione, in caso di successo mediante un nuovo evento denominato ELASTICITYDONE, mentre in caso di fallimento come già avviene il server-agent invierà un evento di eccezione. Per quanto riguarda le risorse da monitorare si pensa che anche in questo caso i parametri da monitorare siano quelli riguardanti le componenti di utilizzo della risorsa stessa, ovvero CPU, RAM(o memoria in caso di VM), larghezza di banda ecc. , e che in caso di determinate soglie la nuova entità/agente dovrà inviare al server-agent locale un evento che rispecchia l'azione elastica da svolgere. A tal proprosito si pensa che oltre alle già citate tipologie si debba inserire una nuova categoria di eventi, quella riguardanti gli eventi di elasticità(Elasticity Events). In questa nuova tipologia di eventi potrebbero essere inseriti eventi come: SCALEOUT, SCALEIN, il già citato EXPANDQUEUE ed infine LOADBALANCE. Tali eventi una volta inviati dalla nuova entità/agente al server-agent locale porteranno all'invio degli eventi di gestione, interni e di coordinazione necessari per portare a termine in modo corretto l'operazione di elasticità. Rispettivamente SCALEOUT e SCALEIN potrebbero richiamare eventi di gestione come creazione/clonazione e di eliminazione entrambi associati a quelli di coordinazione,

EXPANDQUEUE eventi interni e LOADBALANCE eventi di coordinazione affiancati da alcuni di gestione come quelli di mobilità e migrazione, sospensione e ripristino.

In questo modo il framework assimilerebbe la proprietà di elasticità e permetterebbe al sistema che ne sfrutta l'architettura di acquisire una maggiore affidabilità ed efficienza, avere scalabilità delle risorse e degli agenti, e sviluppare anche proprietà di fault tolerance e load balancing dovute all'ottimizzazione dei processi ottenute mediante l'inserimento dell'elasticità. Mediante la modifica del meta-modello, il quale risulterebbe come presentato precedentemente, sarà facilitata la progettazione di tutti gli elementi implementativi e le strutture che saranno incaricate di realizzare in concreto il concetto di elasticità e tutte le relative caratteristiche.

Capitolo 3

Coordination-aware elasticity

La coordinazione rappresenta un concetto fondamentale in informatica agent-oriented, poiché permette la gestione delle dipendenze tra le attività degli agenti in modo da garantire che tutti si comportano secondo gli obiettivi del sistema nel suo complesso ed evitando che essi interferiscano tra loro. Il concetto di elasticità, invece, può essere rappresentato da direttive di programmazione che richiedono monitoraggio e controllo della qualità del servizio, del dimensionamento delle risorse del sistema, così come le loro dipendenze attraverso l'applicazione dei meccanismi di elasticità. Entrambi i concetti quindi, all'interno del proprio contesto, vanno ad agire sulle dipendenze delle entità coinvolte e si occupano della loro gestione a runtime: il primo, delle attività degli agenti in un sistema multi-agente, ed il secondo, delle componenti elastiche del sistema[8].

Poiché i modelli di coordinazione forniscono un approccio generale alla gestione di interazione, e l'inserimento del concetto di elasticità in sistemi MAS che ne fanno uso comporta interazioni tra i componenti presenti nell'ambiente e gli agenti e anche solo tra gli agenti stessi, è normale pensare che sia necessario fornire un approccio basato sulla coordinazione abbinata all'elasticità. Esso può essere rappresentato attraverso diverse direttive di programmazione che sostengono la delega e separazione degli interessi, sia in fase di progettazione sia in fase di esecuzione, e tale considerazione apre la strada verso il concetto di coordination-aware elasticity[8].

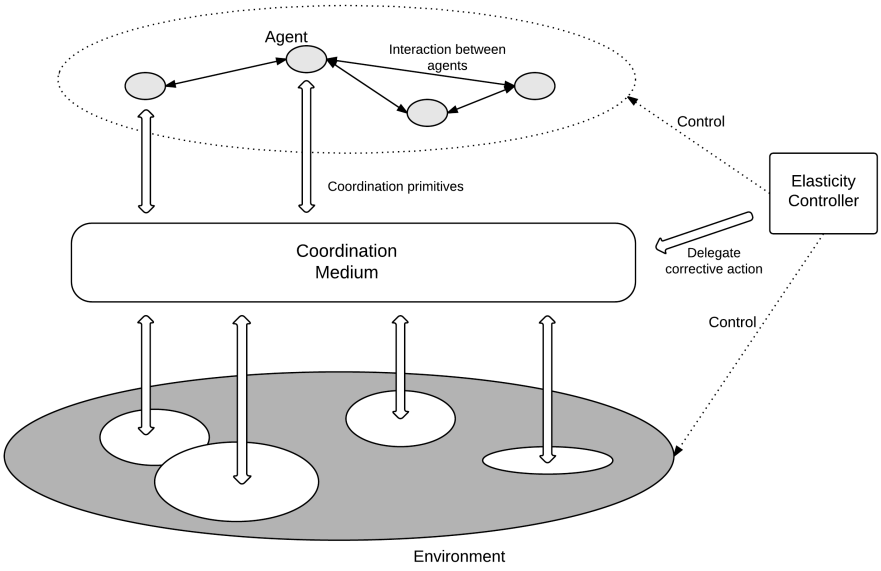
Oltre ai precedentemente citati benefici dell'integrazione dell'elasticità in un sistema MAS, l'inserimento dell'elasticità in un sistema ad agenti che utilizza un mezzo di coordinazione per le interazioni fornirebbe ulteriori vantaggi quali:

- Supporto della delegazione a run-time e della separazione degli interessi, lasciando le componenti infrastrutturali indipendenti da ogni particolare aspetto della computazione (elasticità o coordinazione), e delegando le rimanenti a chi ne è responsabile
- Miglioramento della sicurezza delle interazioni tra gli elasticity controller e le componenti del MAS, realizzando le interazioni mediante primitive e leggi di coordinazione ben definite
- Miglioramento della disponibilità da parte degli elasticity controller, attraverso la distribuzione dei compiti di coordinazione alle componenti dedicate(mezzo di coordinazione), riducendone così il carico computazionale
- Semplificazione del processo di sviluppo, consentendo la separazione dei compiti e delle responsabilità tra "gli addetti all'elasticità" e gli "addetti alla coordinazione"

Anche nel caso di MAS che utilizzano mezzi di coordinazione, l'elasticità dovrà esser supportata e simbolicamente rappresentata dall'entità precedentemente introdotta, ovvero l'elasticity controller. Anche in questa situazione, infatti, essa ricoprirà l'entità di monitoring atta al controllo di tutte le entità presenti all'interno del sistema e gestione delle risorse. Come

per quanto visto nei sistemi MAS "semplici", anche all'interno di questo tipo di sistema, potrebbe esser necessario predisporre più elasticity controller, eventualmente specializzati, che si occupino di controllare costantemente lo stato delle risorse ed in caso di necessità decidere quali azioni correttive applicare e, questa risulta la grande differenza con i sistemi precedentemente analizzati, delegare al mezzo di coordinazione l'attuazione vera e propria dell'azione, nella quale saranno compresi sia meccanismi di coordinazione, sia meccanismi di elasticità mediante l'interrogazione di API dedicate al supporto dell'elasticità.

Si è deciso di parlare di delegazione per rispettare la separazione dei compiti, degli interessi delle singole entità coinvolte nel sistema, ma sopratutto per evitare di complicare il processo di attuazione dell'elasticità e facilitare il compito dello sviluppatore nel progettare il comportamento del componente software(entità) addetto all'elasticità. I vari elasticity controller infatti dovrebbero affrontare anche aspetti legati alla coordinazione per sostenere con successo l'elasticità ed i meccanismi di attuazioni sarebbero macchinosi, richiederebbero parametri difficilmente reperibili in fase di progettazione e sarebbero scarsamente sincronizzabili con gli altri meccanismi in atto nel sistema. Quindi la scelta di far applicare le strategie di elasticità al controller senza che esso interagisca con il mezzo di coordinazione esporrebbe il sistema a guasti o errori e ostacolerebbe la portabilità ed il riutilizzo del codice[8].



L'esecuzione di strategie di elasticità in un sistema in cui è presente un mezzo di coordinazione rappresenta il processo più delicato da attuare e quello nel quale il concetto di coordination-aware elasticity si deve più manifestare. Infatti consiste nella fase dove la maggior parte delle delegazioni, attuate a run-time, di operazioni di elasticità e meccanismi di coordinazione ha effettivamente luogo, in modo da rispettare e sostenere la separazione degli interessi e dei compiti all'interno del sistema. L'esecuzione delle strategie solitamente ha inizio all'interno delle direttive di elasticità, in risposta alla violazione o al raggiungimento della

soglia posta dai vincoli di elasticità, catturati mediante il monitoraggio delle risorse, e talvolta tali direttive potrebbero delegare l'esecuzione di operazioni di coordinazione, oltre alla normale esecuzione di azioni di controllo e gestione dell'elasticità del sistema mediante API dedicate all'interrogazione delle operazioni di elasticità delle risorse. Proprio per questo motivo durante l'applicazione delle strategie di elasticità, al suo interno, possiamo fornire anche una suddivisione dell'esecuzione dei compiti, distinguendo la fase di runtime di elasticità e quella di coordinazione, andando così ad enfatizzare ulteriormente il principio della separazione degli interessi e dei compiti all'interno del sistema.

L'elasticity controller quindi dovrà delegare al mezzo di coordinazione, mediante una richiesta di coordinazione, di gestire gli aspetti riguardanti la coordinazione necessari per il corretto completamento dell'azione di elasticità per poi richiamare le direttive di elasticità e le relative operazioni di attuazione messe a disposizione dall'elasticity controller e ritornando poi al controller il nuovo risultato di completamento. Mezzo di coordinazione ed elasticity controller, quindi dovranno interagire costantemente, mediante opportune direttive di delega e comunicazione, per mantenere i meccanismi di coordinazione ed elasticità sempre sincronizzati.

Mapping tra direttive di elasticità e leggi di coordinazione

Le direttive di elasticità e leggi di coordinazione solitamente sostanzialmente hanno obiettivi ed espressività simili, pertanto è possibile dimostrare che può esistere un mapping tra di esse che ci permetta di fornire i concetti essenziali per la loro integrazione in un sistema MAS con coordination-aware elasticity. Come sappiamo, le direttive di programmazione dell'elasticità sono solitamente composte da primitive di monitoring, vincoli e strategie, che descrivono come i servizi e le risorse dovrebbero comportarsi all'interno del sistema in base al loro stato osservabile e le dinamiche da attuare attraverso l'utilizzo di primitive ben definite. Le leggi di coordinazione, a loro volta, descrivono come i coordinati dovrebbero comportarsi secondo lo stato dello spazio interazione e le loro interazioni osservabili, attraverso l'utilizzo di primitive di coordinazione ben definite[8]. È quindi necessario che le leggi di coordinamento siano in grado di osservare le dinamiche spaziali di interazione e il loro stato e di computare su tale spazio per cambiare il suo stato e le sue dinamiche. Detto ciò è possibile fornire il seguente mapping concettuale:

	Elasticity Directives		**Coordination Laws**
(1)	Directive	$\Longleftrightarrow$	Law
(2)	Runtime Functions	$\Longleftrightarrow$	Primitives
(3)	Monitoring	$\Longleftrightarrow$	Events
(4)	Constraint	$\Longleftrightarrow$	Observation
(5)	Strategy	$\Longleftrightarrow$	Computation

Grazie a questo mapping esistente tra le astrazioni dell'elasticità e della coordinazione, è sempre possibile trovare una corrispondenza significativa tra le direttive di programmazione di elasticità e i costrutti delle leggi di coordinazione, nonostante le possibili differenze nella

semantica di esecuzione dei linguaggi utilizzati. Ciò permette anche di sostenere che, nonostante ogni modello di esecuzione delle leggi di coordinazione varia a seconda del modello di coordinazione adottato e secondo la sua specifica implementazione, le funzionalità e/o le caratteristiche di una delle due possano essere facilmente ed opportunamente integrate con quelle dell'altra per gestire in modo sinergico tutti gli aspetti e le situazioni che si andranno a creare in un MAS che utilizza un mezzo di coordinazione per le interazioni ed in cui vogliamo inserire il concetto di elasticità.

A fronte di ciò si pensa che l'elasticity controller per comunicare al mezzo di coordinazione quale operazione di elasticità ed i rispettivi meccanismi di coordinazione dovrà eseguire, dovrà essere in grado di effettuare la richiesta di delegazione utilizzando una nuova primitiva del linguaggio utilizzato dal mezzo di coordinazione definita opportunamente, mentre per il mezzo di coordinazione, per invocare le funzioni di elasticità, sarà sufficiente richiamare il relativo metodo presente nelle API dedicate al supporto dell'elasticità, utilizzandolo magari all'interno di primitive che istanziano un processo parallelo a quello della coordinazione, e ritornare poi il risultato.

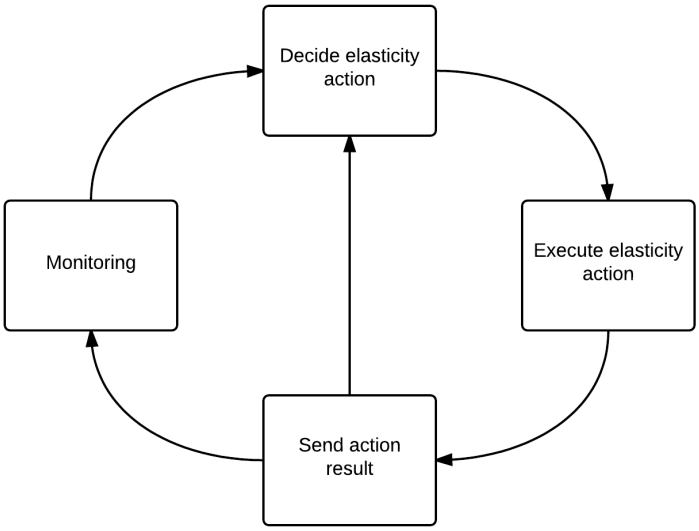
Coordination-aware elasticity loop

L'inserimento del concetto di elasticità nei sistemi MAS, con mezzi di coordinazione e non, ha portato notevoli vantaggi come una maggiore affidabilità, un aumento delle performance in condizioni che prima provocavano rallentamenti o colli di bottiglia, ma sopratutto ha reso il sistema adattabile a scalabile in base alle richieste in ingresso e alle risorse necessarie per portare a termine i task richiesti diminuendo drasticamente le probabilità di fallimento. Ciò permette anche di sostenere che il cambiamento dell'ambiente e del sistema stesso non sarà più non-deterministico, ma sarà solo parzialmente non-deterministico poiché esso potrà cambiare seguendo strategie di elasticità predefinite dal programmatore del sistema. Tali cambiamenti infatti saranno caratterizzati e potranno esser modellati e successivamente implementati seguendo una sorta di ipotetico anello di controllo dell'elasticità che sarà composto dalle principali fasi che compongono un'azione elastica. Tale anello di controllo è stato pensato, nel caso di MAS con mezzi di coordinazione, con il nome di coordination-aware elasticity loop, e prende spunto da una tecnica ampiamente utilizzata per la scalabilità in ambienti Cloud, ovvero la Quality-of-Service.

Concettualmente il sistema in cui sarà presente la coordination-aware elasticity sarà modellato in base alle esigenze e per mantenere sempre elevate le prestazioni e fornire una alta qualità del servizio di coordinazione mediante l'applicazione del coordination-aware elasticity loop, il quale prevede le fasi di:

- Monitoring, fase in cui l'elasticity controller monitora le risorse del sistema, agenti compresi, e ne controlla l'utilizzo in base a misure disponibili ed interrogabili tramite le API dell'infrastruttura e metodi lato agente, come CPU, memoria su disco e larghezza di banda della rete, memoria libera nella mailbox ecc.

- Decide elasticity action, l'elasticity controller in base alle risposte percepite sull'utilizzo delle risorse e a parametri configurati in fase di implementazione dal progettista del sistema, deciderà quale azione correttiva delegare al mezzo di coordinazione tra quelle presenti nelle strategia di elasticità. La procedura di delegazione sarà eseguita dal controller mediante l'invocazione di una primitiva di coordinazione implementata per svolgere attività di elasticità abbinata ad operazioni di coordinazione.
- Execute elasticity action, il mezzo di coordinazione ricevuta la primitiva di elasticità avrà il compito di richiamare tutti i metodi di elasticità e di eseguire tutte le operazioni di coordinazione necessarie per realizzare con successo l'attività di elasticità.
- Send action result, al termine dell'azione di elasticità il mezzo di coordinazione dovrà fornire un feedback all'elasticity controller in modo tale da mantenere sempre sincronizzate le due entità ed i rispettivi meccanismi di funzionamento. In caso di successo dell'azione l'elasticity controller inizierà a monitorare le nuove risorse istanziate ed il procedimento ripartirà, in caso di fallimento invece, l'elasticity controller dovrà decidere quale nuova azione correttiva attuare ed invocare una nuova primitiva.



In questa rappresentazione e nel dettaglio delle rispettive fasi è facile notare come sia stato rispettato il vincolo della separazione dei compiti e degli interessi, imposto per evitare di trovarci di fronte a processi complessi in cui è alta la possibilità di fare errori durante la progettazione e lo sviluppo del sistema.

Conclusioni

Come già più volte visto nella relazione, si è pensato che l'introduzione del concetto di elasticità all'interno dei sistemi MAS porterebbe a quest'ultimi ad ereditare alcune delle caratteristiche base del Cloud Computing, il contesto da cui l'elasticità è stata esportata. Si è pensato che proprietà come la scalabilità delle risorse, degli agenti e dei rispettivi canali di comunicazione, una maggiore efficienza e una maggiore fault tolerance sarebbero concettualmente realizzabili mediante l'introduzione di una nuova entità, l'elasticity controller, la quale attraverso un'operazione di costante monitoring delle risorse deciderà quando e quali azioni elastiche correttive attuare(nel caso di MAS "semplici") o delegare(nel caso di MAS con mezzi di coordinazione). Si è cercato di fornire un semplice modello concettuale, estensione di quello già esistente per gli attuali MAS, per entrambi i tipi di sistema dove si è cercato di mantenere costante l'idea di separazione dei compiti e degli interessi, sapendo però che l'elasticità porterà ad un aumento della complessità di gestione a livello strutturale ed implementativo. A livello di infrastruttura e di ambiente si è pensato però che la virtualizzazione delle risorse ammortizzerebbe questa complessità in quanto essa rende i sistemi più agili e flessibili permettendo di disaccoppiare l'infrastruttura informatica da quella fisica e di avere più istanze della stessa risorsa su un'unica macchina, mentre a livello di agente sono state proposte due principali soluzioni per renderli elastici, ovvero la loro replicazione o l'aumento della capacità della loro mailbox, ed evitare errori o latenze. Entrambe le soluzioni dovranno essere adottate all'interno di strategie di elasticità ben definite in fase di progettazione, ma che potremmo andare a modificare a run-time(magari attraverso un'interfaccia di gestione) visto l'evoluzione del sistema nel tempo, e che prevederanno anche l'estensione di primitive di coordinazione nel caso dei sistemi MAS con mezzi di coordinazione. Tali estensioni porteranno ad un'effettiva integrazione dei concetti, linguaggi e meccanismi a runtime di elasticità e coordinazione all'interno dello stesso sistema.

In questa relazione quindi si è cercato di dare un primo punto di partenza da cui incominciare per uno sviluppo vero e proprio di sistemi MAS e MAS che fanno uso di mezzi di coordinazione con proprietà di elasticità, fornendo anche alcuni esempi di una sua applicazione in sistemi/framework MAS.

Bibliografia

[1] Andrea Omicini, A.Molesini, Enrico Denti - “Metodologie per l'ingegneria del software: Approccio ad agenti”, DEIS Alma Mater Studiorum, Università di Bologna

[2] Agents Introduction - http://agents.umbc.edu/introduction/ao/

[3] Schahram Dustdar, Yike Guo, Benjamin Satzger, Hong-Linh Truong - “Principles of Elastic Processes”, Vienna University of Technology, Imperial College London

[4] Schahram Dustdar, Hong-Linh Truong - “Virtualizing Software and Humans for Elastic Processes in Multiple Clouds– a Service Management Perspective”, Distributed Systems Group, Vienna University of Technology, Austria

[5] Stefano Mariani - “Jade: Java Agent DEvelopment Framework Overview”, DISI Alma Mater Studiorum, Università di Bologna

[6] “Analysis and benchmark of Scalability and Performance of JADE’s”, November 2004

[7] Giancarlo Fortino, Alfredo Garro, Samuele Mascillaro, Wilma Russo - “Using event-driven lightweight DSC-based agents for MAS modelling ”, DEIS, Università della Calabria

[8] Stefano Mariani, Hong-Linh Truong, Georgiana Copil, Andrea Omicini, Schahram Dustdar - “Coordination-aware Elasticity”, DISI Alma Mater Studiorum, Università di Bologna & Distributed Systems Group, Vienna University of Technology