Un approccio basato ad agenti per l'Autonomic Computing

Davide Fabbri

30 novembre 2009

Indice

1 Introduzione

La crescente complessità computazionale delle applicazioni informatiche, dovuta dalle richieste dell'industrie di business e resa possibile dal continuo progresso tecnologico, sta portando il mondo dell'Information Technology ad affrontare una "crisi della complessità".

Tale complessità dei sistemi informatici non è solamente frutto dell'utilizzo della capacità computazionale messa a disposizione delle singole macchine, ma soprattutto deriva dalla realizzazione di sistemi informatici distribuiti nella rete, i quali sono legati da una fitta rete di relazioni e interazioni; si genera così un traffico di lavoro per tali sistemi impredicibile.

Nelle industrie i costi che intervengono nel ciclo di vita di un determinato sistema informatico rappresentano il "costo totale di proprietà" (TCO: Total Cost of Ownership, definito da Gartner), il quale può essere suddiviso in 2 componenti principali:

Costi legati all'Hardware : fra questi troviamo ad esempio il costo fisico dell'hardware, alimentazione dei sistemi, installazione fisica dei componenti, smantellamento del sistema.

Costi legati al Software : fra questi identifichiamo ad esempio i costi di configurazione software, gestione e mantenimento del sistema, costi di backup e recovery e costi di licenze Software.

Il suddetto progresso tecnologico ha di fatto ridotto considerevolmente i costi legati all'Hardware: sono stati infatti resi disponibili progressivamente sul mercato dispostivi da un rapporto prestazioni/consumi crescente a fronte di un costo pressochè stabile; inoltre l'utilizzo massiccio di sistemi wireless ha ridotto sensibilmente i costi dovuti allo smantellamento di cavi quali quelli utilizzati nelle reti LAN Ethernet.

Le applicazioni e i servizi software utilizzati dalle aziende sono viceversa divenuti più complessi, offrendo nuove funzionalità utili al business delle aziende: ciò ha portato ad un incremento dei costi di configurazione, gestione e mantenimento di tali sistemi informatici, senza contare i costi legati alle interruzioni di tali servizi.

In questo articolo si studierà un possibile approccio utile a ridurre quest'ultimi costi, utilizzando la crescente capacità computazionale offerta dall'attuale tecnologia per governare la complessità dei sistemi informatici odierni.

Nella Sezione 2 si introdurrà il concetto di Autonomic Computing, come definito da IBM (vedi [3]) e in che modo questo possa produrre vantaggi per le società IT nello sviluppo di sistemi informatici. In particolare si delineerà un'architettura di base, le caratteristiche dei principali componenti individuati e i liv-

elli in cui questo movimento può entrare nell'ambito del sistema informatico di ciascuna azienda.

Nella Sezione 3 si effettuerà uno studio sulla possibilità di utilizzare un approccio basato sul meta-modello Agenti & Artefatti per l'Autonomic Computing: si cercherà di collegare i compiti svolti dai componenti definiti dall'architettura dell'Autonomic Computing in funzionalità assunte dalle astrazioni che caraterizzano tale meta-modello; dato che non si ancora affermato un modello di agente nel mainstream, consideriamo in questo caso agenti con architettura BDI (Beliefs, Desires, Intentions), ritenuti più adatti allo scopo prefissato.

Nella Sezione 4 si discuteranno i possibili sviluppi futuri di tale approccio: in particolare si discuterà l'introduzione di funzioni autonomiche in sistemi informatici che gestiscono macchine virtuali, l'utilizzo di tale funzioni in ambito WebServices e applicazioni Service Oriented ed infine si delineerà la possibilità che la coordinazione di componenti autonomici possa far emergere funzioni autonomiche ad un livello superiore (e.g.: a livello globale del sistema informatico); infine si concluderà delineando i risultati ottenuti attraverso questo studio.

2 Autonomic Computing

2.1 Introduzione

Autonomic Computing è il termine coniato dal gruppo di ricerca di IBM per quanto riguarda l'introduzione in sistemi informatici di capacità di autogestione delle risorse che compongono il sistema informatico stesso.[3]

Il termine "Autonomic" deriva dal comportamento del corpo umano, il quale svolge una molteplicità di funzioni senza il nostro diretto controllo; ad esempio le capacità autonomiche del sistema nervoso permettono di controllare il battito del cuore, il livello di glicemia e la temperatura corporea senza il nostro diretto controllo su esse.

Alcune funzioni del corpo umano sono invece parzialmente autonomiche: il più classico esempio è relativo alla respirazione la quale può essere governata in maniera autonoma dal nostro sistema nervoso, ma è possibile riprenderne il completo controllo in qualsiasi momento.

L'obiettivo attuale dell'introduzione di attività autonomiche nel sistemi informatici si avvicina più forse a quest'ultimo esempio relativo alla respirazione umana: si vuole fare in modo che gran parte delle attività di gestione di routine dei sistemi venga svolta non più dai professionisti IT dell'azienda, ma possa essere delegata in maniera affidabile ai sistemi computazionali stessi.

Perseguendo questo scopo, i professionisti IT possono concentrare il proprio lavoro sulla definizione di corrette politiche di business da assegnare ai sistemi informatici, i quali potranno gestirsi in maniera autonoma in accordo alle politiche definite.

Una gestione autonoma consiste nell'incapsulare all'interno dei sistemi capacità di rilevare determinati eventi, valutare le condizioni generali ed eventualmente effettuare talune operazioni per mantenere il sistema produttivo ed efficiente in termini di prestazioni, consumi, sicurezza e affidabilità.

Vantaggi ipotizzati I vantaggi che si presumono ottenibili attraverso l'introduzione di questo tipo di capacità intervengono principalmente sull'efficacia e l'efficienza dei processi IT che intercorrono nell'ambito produttivo di un'azienda: tali misure vengono valutate misurando parametri quali tempo necessario, costi economici e percentuale di successo del processo.

I miglioramenti a cui si mira in questo ambito sono relativi a 3 macro-punti.

Automazione nella selezione dei processi il personale IT nell'ambito dello svolgimento dei processi devono effettuare una serie di operazioni che spesso viene attualmente svolta in maniera poco efficiente: si pensi ad esempio

alla raccolta manuale di informazioni dai log dei servizi, al recupero di informazioni da documentazione scarsa e spesso ricca di lacune. Un'introduzione efficace di funzioni autonomiche permette di automatizzare gran parte dei compiti svolto da questo personale, evitando in maniera significativa errori di tipo umano quali ad esempio un'errata valutazione di un file di log, inserimento di parametri non corretti, etc..

Riduzione di tempi molte delle operazioni svolte nell'ambito di un processo IT richiedono una ricerca di informazioni all'interno del sistema che spesso viene svolta ancora da personale umano, la cui velocità di esecuzione è sicuramente inferiore a quella di un sistema informatico correttamente configurato; inoltre anche la scelta delle possibilità delle operazioni da svolgere (in risoluzione a un particolare problema, in determinate circostanze) può comportare un ingente quantità di tempo perfino per un esperto professionista IT.

Riduzione delle capacità tecniche necessarie i sistemi informatici di grandi aziende sono composti da un insieme di componenti hardware/software spesso molto eterogenei e lo svolgere di attività di basso livello per la configurazione e la ottimizzazione degli stessi comporta la necessità di conoscere una molteplicità di caratteristiche tecniche: se tali operazioni fossero delegate ai sistemi informatici stessi, tali conoscenze teniche non sarebbero necessarie e il personale IT avrebbe la possibilità di eseguire operazioni di più alto livello come la definizione di politiche ottimali per il sistema.

Proprietà Desiderate Le caratteristiche desiderate dei sistemi informatici aventi funzionalità autonomiche possono essere rappresentate dalle seguenti proprietà [2]:

Self-Awareness : un sistema autonomico deve essere consapevole di se stesso, in particolare delle risorse coinvolte, del loro stato e del loro comportamento.

Self-Protecting : un sistema autonomico deve essere in grado di percepire lo stato di pericolo, dovuto ad una minaccia esterna, di una propria risorsa e preservare lo stato di sicurezza e integrità all'intero sistema.

Self-Optimizing : un sistema autonomico deve essere in grado di ottimizzare l'utilizzo delle risorse del sistema in termini sia di prestazioni che di consumi; il risultato ottimale sarebbe riuscire a rispettare politiche che in determinate circostanze prevedono la massimizzazione delle prestazioni, mentre in altre mirino alla minimizzazione dei consumi.

Self-Healing : un sistema autonomico deve poter recuperare a errori occorsi durante lo svolgimento dei processi e ripristinare lo stato di produttività del sistema.

Self-Configuring : un sistema autonomico deve poter automaticamente configurare una nuova risorsa che entra a far parte del sistema; ciò avviene recuperando le informazioni necessarie per tale configurazione e rendere possibile le interazioni con le altre risorse presenti.

Contextually Aware : un sistema autonomico deve essere in grado di rilevare le condizioni dell'ambiente in cui lavora e essere in grado di reagire ad eventuali cambiamenti.

Open : un sistema autonomico deve essere portabile e supportare l'eterogeneità dei componenti hardware e software che compongono i moderni sistemi informatici; a tale proposito è utile basare l'interazione con le varie risorse e l'ambiente su standard e protocolli aperti.

Un sistema autonomico non deve necessariamente mirare a tutte le proprietà *Self-**, ma può concentrarsi esclusivamente su alcune di esse; è opportuno ad ogni modo rendere possibile una futura espansione dello stesso in modo da includere altre proprietà.

2.2 Architettura

Nell'ambito della letteratura relativa all'argomento Autonomic Computing, il punto di riferimento è sicuramente il blueprint prodotto dal gruppo di ricerca IBM [3] nel quale è stata definita una generale architettura per sistemi autonomici.

E' opportuno ricordare che per sistema autonomico si intende l'insieme di componenti (software) che vengono introdotte in un sistema informatico per fornire quelle proprietà *Self-** precedentemente descritte.

L'architettura definita da IBM può essere considerata completa, ma in altri lavori, quale [7] per esempio, vengono introdotti ulteriori elementi che forniscono un supporto infrastrutturale alle funzionalità del sistema.

La struttura generale di tale architettura è osservabile nella Figura 1, mentre qui di seguito verranno descritti i singoli componenti e le relative funzioni all'interno del sistema.

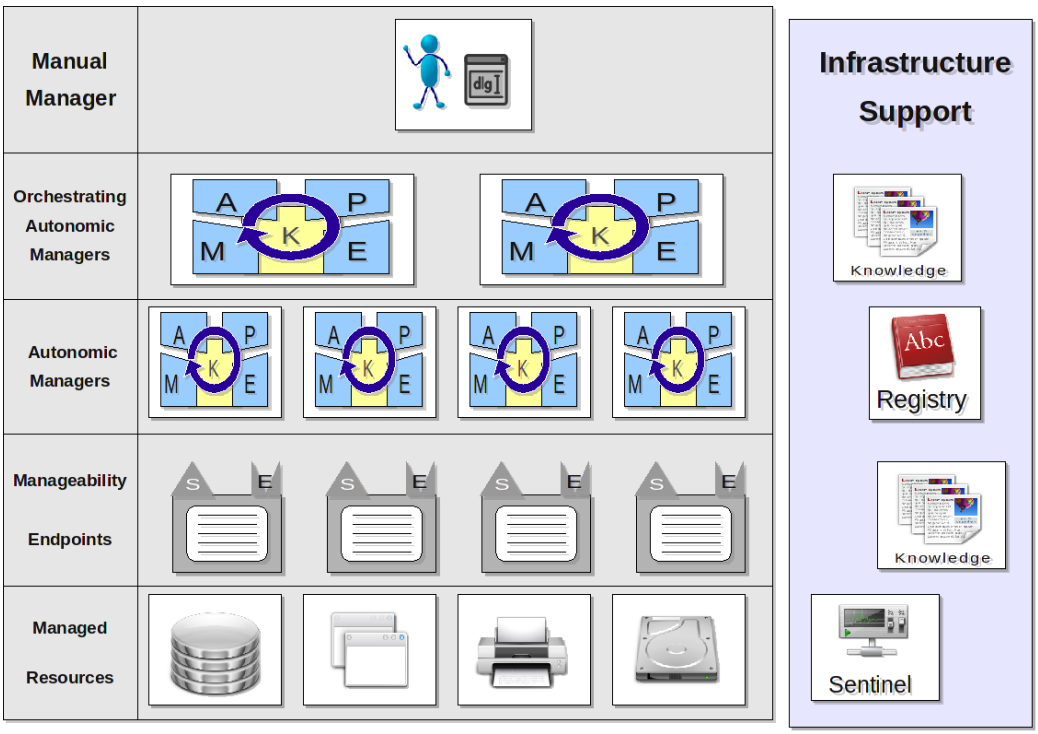

Figura 1: Architettura di un sistema autonomico

2.2.1 Managed Resource

Al livello più basso dell'architettura di un sistema autonomico poniamo le risorse del sistema informatico delle quali vogliano cercare di automatizzare il funzionamento; come è possibile vedere nella Figura 1 fra queste risorse possiamo trovare elementi largamente eterogenei: si va da database a applicativi software, da stampanti a dispositivi di storage, ma possiamo includere in questo elenco anche dispositivi accessibili via Bluetooth (Smartphone e PDA per esempio) oppure il controllo di macchine virtuali attive su un server remoto.

2.2.2 Manageability Endpoint

Ciascuna risorsa che vuole essere gestita attraverso un approccio autonomico in un sistema informatico, offre spesso una serie interminabile di operazioni attuabili su di essa, molte delle quali non sono necessarie per rendere a tale componente del sistema le proprietà *Self-** desiderate.

Un Manageability Endpoint rappresenta dunque una possibile interfaccia di

accesso alla risorsa gestita: come nel paradigma object-oriented ciascun oggetto poteva essere utilizzato attraverso differenti interfacce, è possibile qui definire diverse interfacce d'uso della risorsa; è possibile fare in modo che le funzionalità necessarie per ciascuna delle proprietà *Self-** definite per essa siano racchiuse in un distinto Manageability Endpoint, che verrà poi gestito da un distinto Autonomic Manager.

In tale modo è possibile lasciare libera la scelta al progettista del sistema autonomico di suddividere la gestione automatizzata della risorsa su più Autonomic Manager o delegarla interamente ad uno soltanto: ciò dovrà essere deciso in base alla complessità delle funzioni autonomiche che si vuole assegnare a quella determinata risorsa.

Il Manageability Endpoint offre due differenti tipi di accesso alla risorsa:

Sensors vengono utilizzati per ispezionare lo stato della risorsa stessa (rilevare lo spazio libero su un dispositivo di archiviazione per esempio).

Effectors vengono utilizzati per modificare lo stato della risorsa o per far sì che essa compia una certa operazione (lo spegnimento di una macchina virtuale per esempio).

2.2.3 Autonomic Manager

L'elemento centrale nell'architettura di un sistema autonomico è l'Autonomic Manager, il quale ha l'effettivo compito di automatizzare determinati compiti sulle risorse del sistema; in esso vengono incapsulate le capacità necessarie per fornire ad un sistema informatico le proprietà *Self-** precedentemente elencate.

Ciascun manager ha un accesso limitato a tale risorsa, come definito nel paragrafo relativo al Manageability Endpoint ed esso è definito attraverso l'uso di sensori (Sensors) e attuatori (Effectors): i primi per prelevare informazioni relativi allo stato della risorsa, mentre i secondi per modificare tale stato in accordo con le politiche di automazione assegnate.

L'attività di un Autonomic Manager è rappresentata principalmente da un ciclo di controllo intelligente sulla risorsa gestita composta da 4 fasi ben distinte come osservabile nella Figura 2.

La prima fase, detta **Monitor**, consiste nel rilevare tutti gli eventi generati dal Manageability Endpoint relativo alla risorsa gestita: questi eventi possono essere relativi a cambiamenti di stato interni oppure ad avvii/conclusioni dell'esecuzione di operazioni; la rilevazione di tali eventi avviene mediante la ciclica interrogazione dei sensori del Manageability Endpoint.

La fase di **Analyze** ha invece il compito di filtrare l'insieme degli eventi raccolti nella fase precedente e di constatare se la risorsa gestita è in una condizione tale da richiedere un intervento su di essa: in questa fase si effettua un'analisi

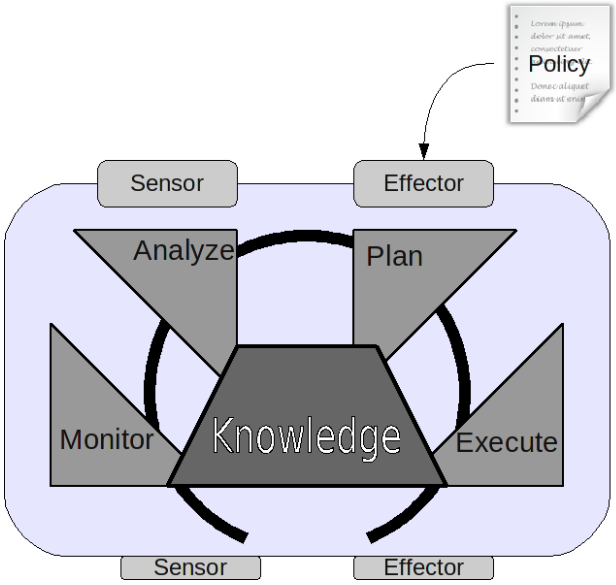

Figura 2: Architettura di un autonomic manager secondo la visione IBM [3]

complessa a partire dai dati storici che l'Autonomic Manager dispone, correlati con le nuove informazioni ottenute nella prima fase.

Una volta rilevata una situazione non coerente ciò che è stato definito nelle politiche dell'Autonomic Manager, la fase di **Plan** fornisce quel meccanismo in grado di definire una sequenza di azioni da effettuare sulla risorsa in modo da perseguire gli obiettivi prefissati; per raggiunengere tale scopo sarà necessario attingere da più fonti di informazioni: la conoscenza locale dell'Autonomic Manager, la conoscenza globale disponibile all'interno del sistema in cui il manager opera ed infine le informazioni specificate nella politica assegnata all'Autonomic Manager.

Nell'ultima fase, quella di **Execute**, si eseguono le operazioni individuate al punto precedente mediante l'uso degli attuatori messi a disposizione dal Manageability Endpoint: in questa fase è opportuno verificare constantemente eventuali cambiamenti dinamici della risorsa, i quali potrebbero rendere i piani elaborati non più validi.

Il progettista di un sistema autonomico non è vincolato a fare in modo che tutte le fasi vengano svolte dall'Autonomic Manager, in quanto spesso l'amministratore del sistema preferisce avere il controllo manuale delle risorse in molte circostanze: in tal caso spesso le fasi di **Plan** e di **Execute** vengono delegate all'intervento manuale e il compito dell'Autonomic Manager è quello di notificare ad una console le situazioni d'interesse che rileva attraverso le fasi di **Monitor** e **Analyze**.

2.2.4 Manual Manager

Nella visione di Autonomic Computing qui illustrata, il Manual Manager rappresenta la modalità attraverso cui l'utente umano può definire ed esprimere le politiche del sistema, molto spesso un'applicazione dotata di una interfaccia grafica.

Le possibilità di interazione che vengono fornite all'utente umano per interagire con il sistema sono qui di fondamentale importanza: attraverso una console di gestione l'utente deve poter monitorare il funzionamento del sistema, intervenire sulle politiche di gestione automatica del sistema, le quali saranno di più alto livello, quanto i meccanismi autonomici implementati saranno sofisticati ed efficaci.

L'amministratore del sistema potrebbe d'altro canto decidere per rendere l'attività del Manual Manager dinamica, ovvero fare in modo che a seconda delle circostanze certe operazioni possano essere decise in maniera autonoma dal sistema, mentre in altre ciò deve avvenire con la supervisione/intervento di un'operatore umano.

Un'importante vantaggio che si può ottenere attraverso la costruzione di un'unica interfaccia di gestione del sistema è un ridotto utilizzo di tempo sia per l'apprendimento che per l'utilizzo della stessa da parte dell'operatore, spesso costretto attualmente a utilizzare e conoscere molteplici applicazioni di gestione e controllo.

2.2.5 Supporto infrastrutturale

Gli elementi fin qui descritti rappresentano l'essenza di un sistema autonomico, ma vi sono altri componenti che potrebbero essere utili per il raggiungimento degli scopi prefissati.

Sotto questo punto di vista IBM [3] include in questa categoria solo le *Knowledge Sources*, mentre altri [7] includono altri componenti più complessi come *Registry*, *Broker*, *Aggregator*, *Negotiator*, *Sentinel*.

Nella Figura 1 sono stati inseriti gli elementi ritenuti maggiormente significativi per coadiuvare i componenti principali di un sistema autonomico alla realizzazione delle funzionalità richieste.

Knowledge Source : nei sistemi eterogenei quali i moderni sistemi IT è opportuno memorizzare informazioni di interesse comune in strutture ben organizzate. Nell'architettura di sistemi autonomici fin qui descritta abbiamo già

accennato al fatto che la conoscenza deve essere necessariamente distribuita a più livelli e in particolare abbiamo mostrato come per l'Autonomic Manager sia utile disporre di una conoscenza locale relativa al risorsa gestita e poter ricavare informazioni dalle politiche di funzionamento impartitegli da un livello superiore; ad un livello sistemico invece può essere utile disporre di una conoscenza globale accessibile attraverso un'interfaccia standard, in modo da poter ricavare informazioni topologiche, nuovi piani per la risoluzione dei problemi, politiche generali e così via.

Registry : in un sistema autonomico sono di rilevante importanza le interazioni che si possono instaurare fra Autonomic Manager; ad esempio se uno di essi rileva che la risorsa che gestisce potrebbe essere più efficiente o necessita lo svolgimento di una determinata operazione da parte di un'altra si potrebbe instaurare un'interazione fra gli stessi Autonomic Manager. Compito del **Registry** sarebbe poter fornire un modo per contattare un Autonomic Manager che gestisce una particolare risorsa e d'altro canto pubblicare informazioni relative alle funzionalità che un Autonomic Manager può svolgere.

Sentinel : il compito di un elemento **Sentinel** in un sistema autonomico è quello di monitorare determinate situazioni di interesse per gli Autonomic Manager non relative alla risorsa che gestiscono direttamente, ma piuttosto relative allo stato e alle azioni che svolgono altri componenti del sistema autonomico; ad esempio si potrebbe far sì che un Sentinel avverta un'Autonomic Manager quando può delegare ad una risorsa precedentemente non disponibile; è opportuno notare che l'informazione relativa la disponibilità di tale risorsa dovrà essere recuperata dal relativo Autonomic Manager.

2.2.6 Aspetti avanzati

Una volta delineata l'architettura di base e descritti i componenti principali della stessa è opportuno effettuare alcune considerazioni ulteriori su di essa.

In particolare occorre investigare il ruolo dell'Orchestrating Autonomic Manager: esso rappresenta un Autonomic Manager di alto livello in grado di governare il funzionamento dei Resource Autonomic Manager attraverso l'assegnazione delle relative politiche di automazione.

La Figura 3 rappresenta lo schema architetturale dell'insieme composto da Autonomic Manager e Manageability Endpoint che è possibile definire come Autonomic Element; esso dispone di Sensors e Effectors come un qualsiasi Manageability Endpoint ed è possibile organizzare un sistema autonomico realizzando Orchestrating Autonomic Manager che analizzino il funzionamento di uno

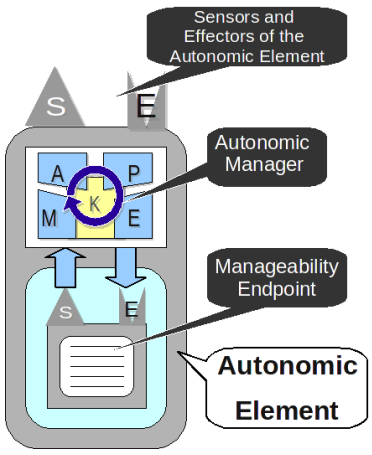

Figura 3: Architettura di un Autonomic Element

o più Autonomic Element e ne influenzino il comportamento come desiderato dall'amministratore del sistema informatico, come osservabile nella Figura 4.

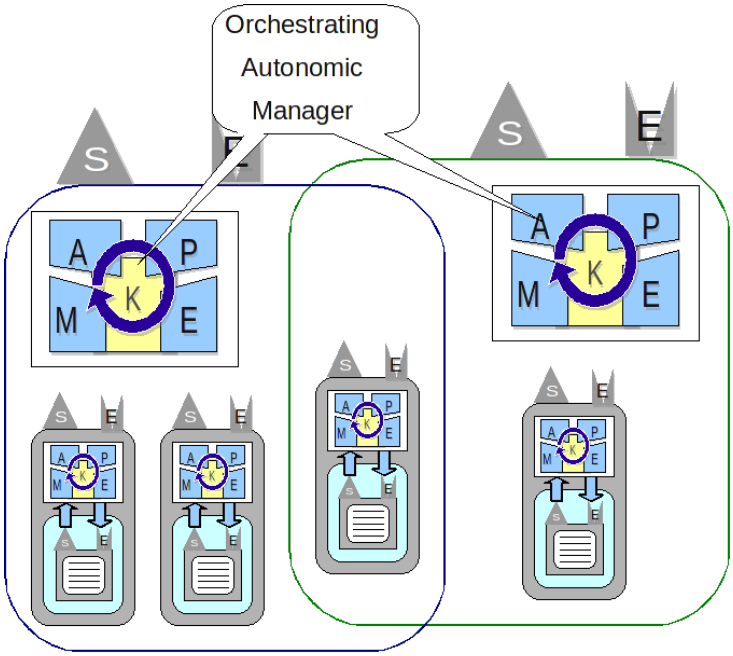

Figura 4: Composizione "gerarchica" di Autonomic Manager: due Orchestrating Autonomic Manager operano su uno o più Autonomic Manager ciascuno

La visione fin qui descritta, la quale sintetizza ciò che è stato definito in letteratura, mostra come l'architettura sia stata fortemente influenzata dai paradigmi di programmazione utilizzati nel mainstream, in particolare dalla programmazione

ad oggetti: tutti i componenti principali dell'Autonomic Computing espongono un'interfaccia di sensori e di attuatori, perfino quando essi vengono dotati di un certo livello di autonomia propria; tale interfaccia infatti ricorda quella del paradigma object oriented che prevede esclusivamente interazioni di tipo *request-response* (accesso ai sensori) o di tipo *fire-and-forget* (accesso agli attuatori).

Interazioni aventi un livello di astrazione maggiore non vengono prese in considerazione, piuttosto sono stati introdotti in molti testi componenti aggiuntivi (quali Sentinel, Broker, Negotiator) cui viene delegato il compito di gestire tale complessità, in modo da non dover "appesantire" i componenti centrali dell'architettura.

Nal capitolo successivo vedremo come possa essere possibile sottrarsi a questi vincoli, basandosi su un differente paradigma di progettazione; ciò dovrà necessariamente portare alla modifica dei punti architetturali strettamente legati al modello di programmazione adottato.

2.3 Evoluzione dei sistemi di Autonomic Computing

Introdurre proprietà *Self-** all'interno di un sistema informatico, soprattutto nel caso in cui esso sia già esistente, comporta numerosi rischi, anche nel caso in cui i componenti autonomici progettati svolgessero bene il proprio compito.

Un sistema completamente autonomico prevede infatti che l'amministratore del sistema fornisca esclusivamente le politiche di business e i vari componenti autonomici siano in grado di interpretarle e metterle in atto, mentre nella maggioranza dei sistemi informatici attuali l'amministratore e gli operatori IT devono svolgere manualmente la maggior parte delle operazioni necessarie al corretto funzionamento del sistema.

Il problema principale risiede nella definizione delle corrette politiche per i componenti autonomici, processo che può indurre facilmente in errore essendo ampio il divario d'astrazione fra un sistema gestito manualmente ed uno gestito in maniera autonomica: è difficile modellare attraverso politiche l'insieme di ragionamenti e considerazioni che l'operatore umano svolge prima di effettuare una qualsivoglia operazione, senza incorrere in errori, almeno fino a quando non venga raggiunta una certa esperienza in tale processo.

L'architettura definita per i componenti di un sistema autonomico rende però possibile un passaggio graduale per l'inserimento di funzionalità autonomiche nei sistemi informatici; ciò avviene principalmente su 2 dimensioni:

Funzionalità : capacità autonomiche fornite ai componenti del sistema (vedi Tabella 1).

Scope : componenti di un sistema informatico dotati di capacità autonomiche (vedi Tabella 2).

Livello	Funzionalità
Manuale	Gli operatori IT svolgono le operazioni di gestione in maniera completamente manuale.
Controllo	I sistemi per la gestione automatizzano la raccolta di informazioni utili presso i Manageability Endpoint in modo da ridurre i tempi necessari per l'amministratore di sistema a rilevare eventuali problemi.
Analisi	A questo livello sono introdotti ulteriori meccanismi autonomici che permettono di correlare le informazioni ricavate da più Manageability Endpoint e di fornire suggerimenti sulla configurazione ottimale del sistema all'amministratore, al quale spetterà il compito di scegliere se attuare o meno tali modifiche al sistema.
Ciclo di Controllo	Il sistema può automaticamente effettuare modifiche basate sulle informazioni e sulla conoscenza su ciò che sta avvenendo nel sistema stesso; alcune operazioni possono essere comunque svolte in maniera manuale dagli operatori IT.
Ciclo di Controllo con Politiche di Business	L'intera infrastruttura IT viene gestita in maniera autonoma basando il comportamento su politiche e obiettivi di business ben definiti. Gli utenti interagiscono con il Manual Manager solo per monitorare il funzionamento del sistema ed eventualmente modificare gli obiettivi e le politiche utilizzate.

Tabella 1: Evoluzione di un sistema autonomico: dimensione relativa alle funzionalità.

2.4 Standard per l'Autonomic Computing

I sistemi informatici utilizzati in ambito aziendale, soprattutto se queste sono di medie/grandi dimensioni, coinvolgono un'eterogeneità di componenti hardware e software che devono interoperare sia internamente, sia con risorse esterne all'infrastruttura IT dell'azienda.

Un'implementazione proprietaria chiusa delle funzionalità autonomiche di un sistema sarebbe, tornando alla metafora del sistema nervoso umano, come poter mantenere un battito cardiaco costante, senza però regolarne la frequenza quando il corpo umano è sottoposto ad uno stress fisico: l'interazione e lo scambio di informazioni fra le risorse di un sistema autonomico sono elementi base per rendere un sistema dotato delle proprietà *Self-**.

A tale proposito appare saggio l'introduzione e l'adozione di standard aperti che permettano di astrarre dallo specifico software o hardware utilizzato all'in-

Livello	Scope
Sotto-componente	A questo livello solo una porzione di risorse sono gestite in maniera autonomica, per esempio una macchina virtuale in un server o alcune applicazioni tipo Web Services.
Componente Singolo	Si gestisce in maniera autonomica un'intera risorsa indipendente, come un server o tutto l'insieme di Web Services presenti nel sistema.
Insieme omogeneo di componenti	Vengono gestite in maniera autonomica molteplici istanze di un tipo si risorsa, ad esempio un insieme di server dotati dello stesso sistema operativo.
Insieme eterogeneo di componenti	Le funzionalità autonomiche sono introdotte in una parte considerevole del sistema informatico, ma non nella sua totalità.
Sistema	Le funzionalità autonomiche sono introdotte nella totalità di risorse hardware e software che compongono il sistema informatico dell'azienda.

Tabella 2: Evoluzione di un sistema autonomico: dimensione relativo allo scope.

terno del sistema; si ritiene utile presentare le caratteristiche di quelli ritenuti più adatti alla causa dell'Autonomic Computing

Web Services Distributed Management (WSDM) Lo standard per la gestione distribuita attraverso Web Services (Web Services Distribution Management) sviluppato da OASIS e ratificato nel Marzo 2005 è probabilmente lo standard chiave nell'ambito dell'Autonomic Computing.

Obiettivo dello standard è quello di rendere possibile all'accesso di risorse all'interno di un'infrastruttura IT distribuita attraverso l'uso di Web Services e scambio di messaggi SOAP.

Tale specifica consta di 2 parti principali:

Management Using Web Services (MUWS) : questa parte fornisce specifiche per accedere alle risorse attraverso l'uso di Web Services; in particolare viene definito il modo in cui ciascuna risorsa può essere univocamente identificata, come è possibile ottenere le relative proprietà ed infine vengono definite le cosiddette "Capabilities", ovvero un modo per definire formalmente le operazioni eseguibili su tale risorsa, definendo così un'interfaccia di gestione completa per la risorsa.

Management Of Web Services (MOWS) : i Web Services sono già ampiamente

utilizzati in molte realtà aziendali per fornire servizi di business, sia internamente che esternamente al sistema informatico dell'azienda; questa parte di specifica definisce le modalità con cui è possibile accedere a funzioni di gestione di uno o più Web Services. La gestione di Web Services (MOWS) rappresenta un caso applicativo per la gestione attraverso Web Services (MUWS) di risorse che sono elementi dell'architettura dei Web Services e per raggiungere tale scopo si basa sull'architettura, definizioni e specifiche di MUWS, ovvero la prima parte dell'intera specifica WSDM.

La prima parte della specifica (MUWS) è decisamente più significativa per quanto riguarda l'Autonomic Computing in quanto è una specifica generale applicabile a qualsiasi tipo di risorsa hardware e software: l'applicabilità di tale specifica risorse particolari come altri Web Services (ovvero Web Services che governano Web Services) è definita nella seconda parte (MOWS).

Solution Deployment Descriptor (SDD) In ambito Autonomic Computing è opportuno citare anche al lavoro svolto da un comitato tecnico di OASIS, il quale mira a costruire un XML Schema in grado di definire le caratteristiche di un'"Installable Unit" (IU), la quale racchiude tutte le informazioni chiave necessarie al deployment, alla configurazione e al mantenimento di un'applicativo software.

Tale specifica appare chiaramente correlata alla proprietà di *Self-Configuring*, una delle principali proprietà desiderate in un sistema autonomico.

Nelle sezioni successive non si affronterà la questione dell'integrazione di tali standard nell'architettura per Autonomic Computing che verrà definita seguendo un differente approccio a quello tradizionale, ma al termine di esse sarà già possibile delineare un piano di lavoro per raggiungere tale scopo senza eccessivi sforzi concettuali, ma esclusivamente di natura tecnica.

3 Un approccio ad agenti per sistemi self-managing

3.1 Introduzione al paradigma A&A e agenti BDI

Lo sviluppo e la diffusione di sistemi computazionali distribuiti e multi-processore ha fatto sì che il modello di programmazione utilizzato per lo sviluppo di applicazioni software utilizzato negli ultimi 2 decenni del secolo scorso necessitasse di alcune estensioni per poter sfruttare a pieno le potenzialità delle nuove tecnologie che si sono affacciate sul mercato.

Il modello di programmazione basato su oggetti, attualmente utilizzato, si basa infatti su un unico flusso di controllo che viene "passato" a strutture dati opportunamente organizzate chiamate oggetti; per adattare tale paradigma ad un ambito distribuito in cui sono presenti molteplici flussi di controllo sono nati nuovi modelli di programmazione che estendono quello basato su oggetti, come ad esempio quello basato su componenti.

Tali soluzioni non affrontano però la radice del problema, ovvero il fatto che il livello delle astrazioni su cui si basano tali modelli di programmazione non è ormai più sufficiente, ma è necessario un paradigma che definisca astrazioni in grado di incapsulare direttamente molteplici flussi di controllo.

Il modello di programmazione, studiato in ricerca per riuscire in tale scopo, è quello basato su agenti, una nuova astrazione in grado di operare in maniera autonoma la quale incapsula (almeno) uno o più flussi di controllo; tale modello però non è stato ancora univocamente definito ed accettato in letteratura e molteplici varianti sono state presentate nel corso degli ultimi anni.

In questo lavoro si è deciso di utilizzare un particolare modello ad agenti per la creazione di sistemi autonomici, in particolare quello definito dal paradigma "Agenti e Artefatti" (A&A): esso fonda le proprie radici sulle concezioni proprie della Teoria dell'Attività e della Cognizione Distribuita, ambiti di studio della psicologia umana, per lo sviluppo di applicazioni concorrenti distribuite.

Il modello A&A cerca di rappresentare un sistema software come un tipico ambiente lavorativo umano dove vi sono persone (**Agenti**) le quali sono entità autonome che operano utilizzando e producendo strumenti (**Artefatti**) nell'ambito di un ambiente lavorativo comune (**Workspace**).

Le entità di primo grado individuate in questo particolare modello di programmazione ad agenti sono dunque [4]:

Agenti Un agente è un'entità pro-attiva in grado di svolgeere attività in maniera autonoma.

Artefatti Un artefatto rappresenta un componente passivo del sistema, uno strumenti intenzionalmente costruito, condiviso, usato dagli agenti a supporto o fine delle loro attività.

Workspace Il workspace è un'entità concettuale utilizzata come contenitore di agenti e artefatti, utilizzata per definire la topologia dei sistemi e poter definire la località dell'entità coinvolte.

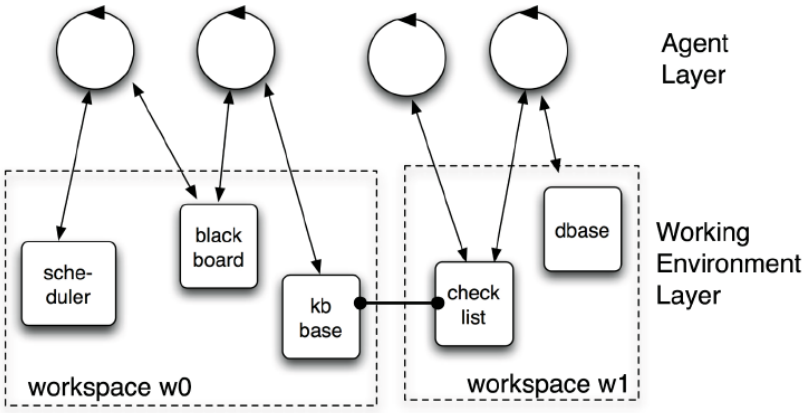

Figura 5: Rappresentazione astratta delle astrazioni del paradigma Agenti & Artefatti

Un aspetto di fondamentale importanza per descrivere un modello di programmazione, dopo aver definito le astrazioni di prima classe, è sicuramente definire opportunamente le interazioni e di quale tipo occorrono fra le astrazioni definite; nel nostro caso sono le seguenti:

- Gli agenti *parlano* con altri agenti
- Gli agenti *usano* gli artefatti
- Gli artefatti *si collegano* con altri artefatti
- Gli artefatti *si manifestano* agli agenti

3.1.1 Agenti BDI

La nozione di agente non è stata ancora definita univocamente in letteratura, nè tantomeno un'architettura formale con la quale costruire entità di questo tipo.

In questa sede si prende in considerazione un particolare tipo di architettura denominata BDI (Beliefs, Desires and Intentions) basata sui seguenti costrutti[1]:

- **Beliefs**, ovvero le informazioni a disposizione dell'agente in un determinato istante, le quali potrebbero essere non coerenti con lo stato attuale del sistema.

- **Desires**, ovvero tutti i possibili obiettivi che un agente *può* mirare a raggiungere: influenzano le azioni dell'agente, ma non costringono esso al raggiungimento degli stessi.
- **Intentions**, ovvero l'insieme di azioni che l'agente ha deciso di voler eseguire.

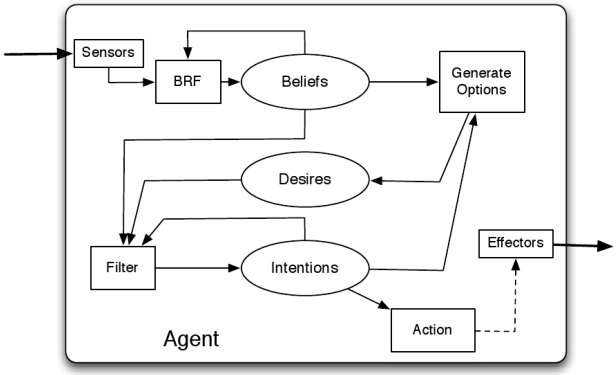

Figura 6: Architettura base di un agente BDI

Gli eventi che avvengono nel sistema concorrono alla modifica dello stato mentale dell'agente; tra essi distinguiamo quelli interni, ovvero che riguardano modifiche agli obiettivi o alle informazioni dell'agente e quelli esterni che sono dovuti ad azioni quali scambi di messaggi o segnalazioni da parte di altri agenti.

Nel 1996 Anand Rao definì un linguaggio astratto per la programmazione di agenti BDI, chiamato AgentSpeak(L) [5], i cui costrutti principali sono, leggermente discostanti dal modello base BDI, i seguenti:

Beliefs stato corrente delle informazioni che un agente dispone nei riguardi dell'ambiente e degli altri agenti.

Goals obiettivi che l'agente mira a raggiungere, effettuando un ragionamento pratico (Practical Reasoning) sugli eventi interni ed esterni che percepisce.

Plans insieme di procedure che un agente ha a disposizione per raggiungere i propri obiettivi.

L'architettura di un agente AgentSpeak(L) è caratterizzata inoltre da 4 principali componenti, com'è è possibile osservare in maniera dettagliata nella Figura 7.

- Una base di conoscenza, denominata Belief Base, che raccoglie l'insieme dei Beliefs dell'agente.

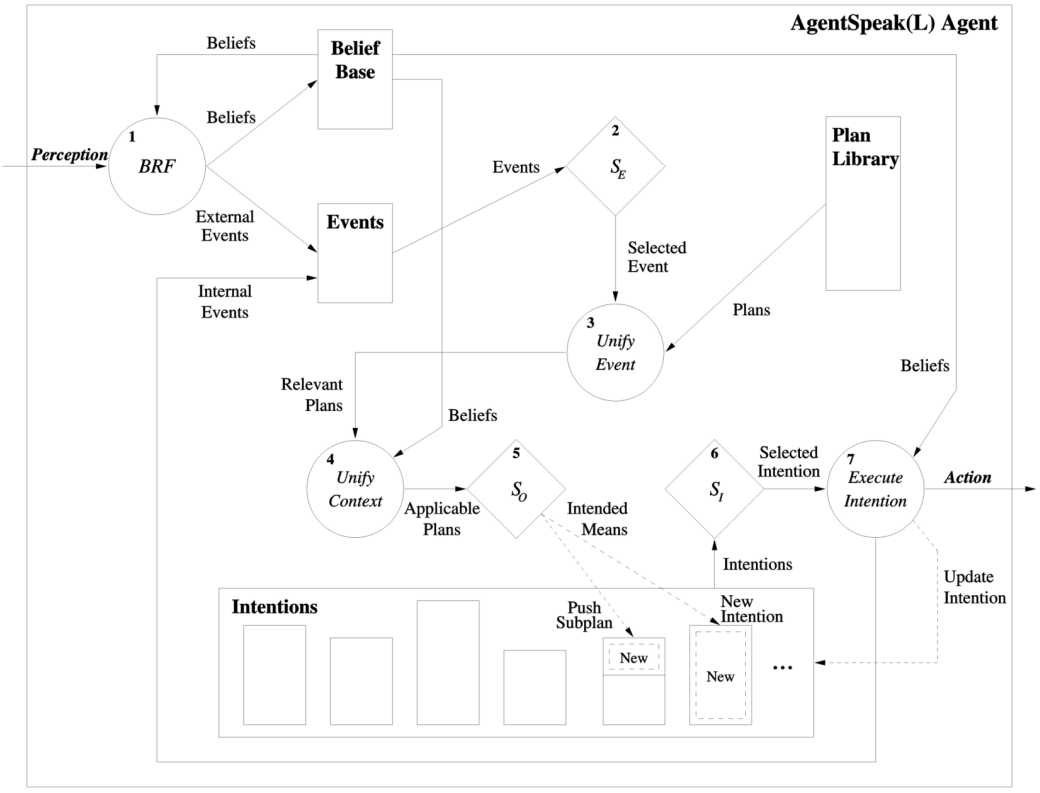

Figura 7: Architettura dettagliata di un agente AgentSpeak(L) [5]

- L'insieme dei piani di cui dispone l'agente, denominato Plan Library.
- Una struttura per raccogliere gli eventi con cui l'agente potrà effettuare il proprio ragionamento.
- L'insieme delle intenzioni che l'azioni intende eseguire e dal quale andrà a selezionare l'intenzione scelta ad un determinato istante.

Il gruppo di ricerca guidato da Rafael H. Bordini e Jomi F. Hubner ha definito un linguaggio, Jason, il quale è una variante di AgentSpeak(L) che implementa ed estende la sua semantica operazionale[1].

Esso non è un linguaggio astratto come AgentSpeak, ma ne è stato sviluppato un framework basato su Java per la programmazione di sistemi multi-agente.

Le estensioni ad AgentSpeak(L) riguardano sia la semantica operazionale vera e propria, sia l'integrazione con sistemi legacy, rendendo Jason un linguaggio di programmazione più pratico facendo uso di Java per implementare meccanismi di basso livello; fra queste estensioni possiamo notare:

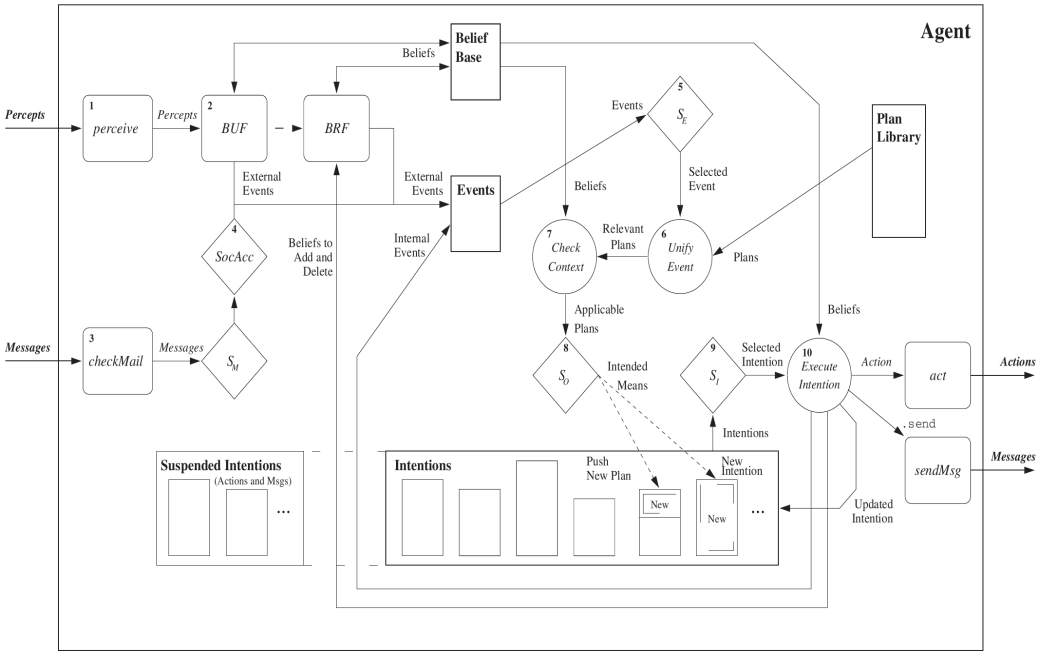

Figura 8: Architettura dettagliata di un agente Jason tratto da [1]: è possibile osservare i passi che compongono il ciclo di ragionamento dell'agente

- I predicati utilizzati dagli agenti possono essere annotati, fornendo informazioni aggiuntive quali la sorgente o il grado di certezza di validità dell'informazione.
- I piani possono allo stesso modo essere annotati per inserire dati utili nella funzione di selezione quali il grado di successo, l'efficacia attesa e così via.
- La negazione forte, denotata con il simbolo ˜, permette di denotare quali informazioni l'agente crede non siano vere, permettendo entrmabe le assunzioni di mondo chiuso e aperto.
- Gestione dei fallimenti durante l'esecuzione dei piani attraverso eventi di cancellazione di goal.
- Supporto all'utilizzo di azioni interne il cui codice viene eseguito in maniera atomica; esse sono utili per modellare l'interazione con artefatti in ambienti A&A e integrare elegantemente sistemi legacy.
- Alta configurabilità dell'architettura dell'agente da parte del programmatore, che può creare varianti dell'architettura di base per modificare ad esempio le funzioni di percezione o di selezione dei piani a suo piacimento.

- Definizione di un sistema di comunicazione fra agenti di alto livello ispirato da KQML,il quale definisce l'atto di comunicazione attraverso contenuto e "performative" (affermazione, richiesta o domanda).

Il confronto fra la Figura 8 e la Figura 7 mostra le differenze fra i cicli di ragionamento di un agente Jason e un agente AgentSpeak, in particolare si può notare la maggiore enfasi sull'elaborazione delle informazioni scambiate attraverso messaggi fra agenti in Jason, elemento architetturale trascurato in AgentSpeak(L).

Jason fornisce un framework completo per la programmazione di sistemi multi-agente, infatti è possibile definire le caratteristiche dell'ambiente in cui gli agenti andranno ad operare attraverso un opportuno file di configurazione; attraverso di esso è possibile definire, oltre agli agenti e al numero di istanze di essi che agiscono nel sistema, anche l'infrastruttura di comunicazione e il tipo di ambiente utilizzato.

La configurabilità di Jason permette di creare nuove tipologie di ambiente (estendendo la classe base `Environment` fornita), rendendo possibile una facile integrazione con altre infrastrutture.

3.1.2 Artefatti e Workspace

Nell'introduzione al paradigma A&A si è già mostrato come gli agenti non sono soli nei sistemi multi-agente, ma operano in un determinato ambiente operativo, il quale nel nostro meta-modello è composto da 2 particolari astrazioni fortemente correlate fra loro.

Un artefatto rappresenta un'entità passiva che incapsula un insieme di funzionalità controllabili e osservabili da parte degli agenti, le quali sono esplicitamente progettate dai programmatori del sistema multi-agente.

A differenza degli agenti, gli artefatti non dispongono di alcun tipo di autonomia, in quanto il proprio scopo è quello di servire le attività degli agenti e in tal senso sono computazionalmente reattivi, in quanto reagiscono esclusivamente agli stimoli esterni di agenti.

Gli artefatti si possono suddivedere principalmente in 2 categorie:

Artefatti sociali : i quali hanno il compito di regolare le interazioni sociali fra agenti e artefatti all'interno di un sistema multi-agente: sono strumenti utili alla coordinazione e alla cooperazione fra agenti nell'ambito dello svolgimento delle proprie attività.

Artefatti ambientali : astraggono l'accesso a risorse hardware/software esterne al sistema multi-agente; permettono quindi agli agenti di interagire con altre componenti del sistema informatico, come ad esempio Web Services, macchine virtuali, interfacce grafiche o basi di dati. Essi dunque possono

simulare il mondo reale, fornire un'interfaccia a risorse esterne o fungere da wrapper per l'interazione con altre tecnologie esistenti come schematizzato nella Figura 9.

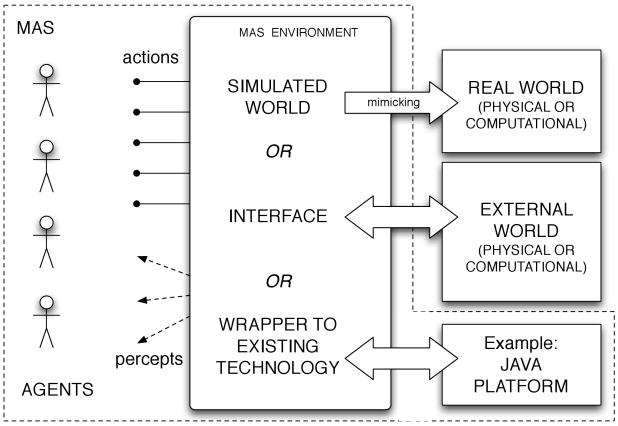

Figura 9: Ruoli degli artefatti ambientali nella modellazione dell'interazione fra agenti e risorse esterne

Un artefatto è caratterizzato dalle *operazioni* che mette a disposizione degli agenti mediante un'opportuna *interfaccia d'uso*; ciascuna operazione può essere suddivisa in più passi computazionali, la cui esecuzione è intervallata *guardie* che consistono nella verifica di determinate condizioni prima di procedere con il passo successivo: questo permette l'esecuzione concorrente di più operazioni da parte dello stesso artefatto, alternando l'esecuzione dei singoli passi atomici.

Un artefatto può essere dotato inoltre di *proprietà osservabili*, le quali possono essere rilevate dagli agenti, senza che questi necessariamente abbiano eseguito alcuna operazione su tale artefatto.

L'esecuzione di un'operazione può provocare sia una modifica nello stato interno dell'artefatto (non osservabile), sia l'emissione di un insieme di *eventi osservabili* che possono essere percepiti da agenti che stiano usando o osservabdo tale artefatto.

Aspetti avanzati nell'architettura di un artefatto sono il *manuale* d'uso dell'artefatto il quale descrive funzione, interfaccia d'uso e *istruzioni operative* dello stesso.

Fra le possibili interazioni fra astrazioni del meta-modello A&A abbiamo citato la possibilità di collegare fra loro 2 artefatti: a tale scopo ciascun artefatto espone una propria *interfaccia di collegamento* in modo da poter rendere possibile la composizione di artefatti.

Questa architettura interna di un artefatto appena descritta è stata implementata in una piattaforma basata su Java, chiamata CARTAGO e sviluppata dal grup-

po di ricerca del DEIS a Cesena [6], ed è rappresentabile secondo lo schema in Figura 10.

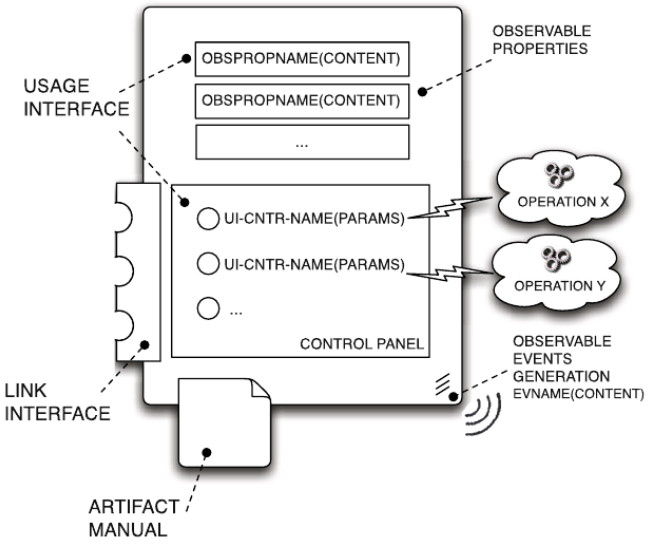

Figura 10: Rappresentazione astratta dell'architettura interna di un artefatto [6]

Un workspace è di fatto un'entità logica atta a contenere in maniera organizzata gli artefatti creati, utilizzati e manipolati dagli agenti durante le proprie attività e, come rappresentato in Figura 5; esso può essere utile a definire la topologia di un sistema multi-agente in ambito distribuito, in cui in ciascun nodo sono posizionati uno o più workspace.

Ciascun workspace in CARTAGO fornisce alcuni artefatti predefiniti:

Factory : artefatto utilizzato per costruire dinamicamente nuovi artefatti situati nel workspace.

Registry : artefatto funge da registro in grado di mantenere traccia degli artefatti attualmente situati nel workspace, creati attraverso l'artefatto *Factory*. Esso viene utilizzato dagli agenti per recuperare le singole istanze dei artefatti in modo da poter operare su di esse.

Security-Registry : questo artefatto viene utilizzato per gestire i ruoli disponibili e le relative politiche di controllo di accesso. CARTAGO supporta infatti un modello di sicurezza basato su ruoli RBAC.

L'infrastruttura CARTAGO permette agli agenti di entrare ed agire su più workspace, anche se questi sono posizionati su nodi di rete differenti; inoltre vi è la possibilità di collegare artefatti situati in differenti spazi di lavoro, attraverso

un opportuno utilizzo delle interfacce di collegamento proprie di ciascun artefatto coinvolto.

Riassumendo è possibile affermare che tale piattaforma permette di astrarre completamente gli aspetti tecnici della comunicazione in ambiente distribuito, lasciando liberi i programmatori di concentrarsi sugli aspetti di business logic di agenti e artefatti.

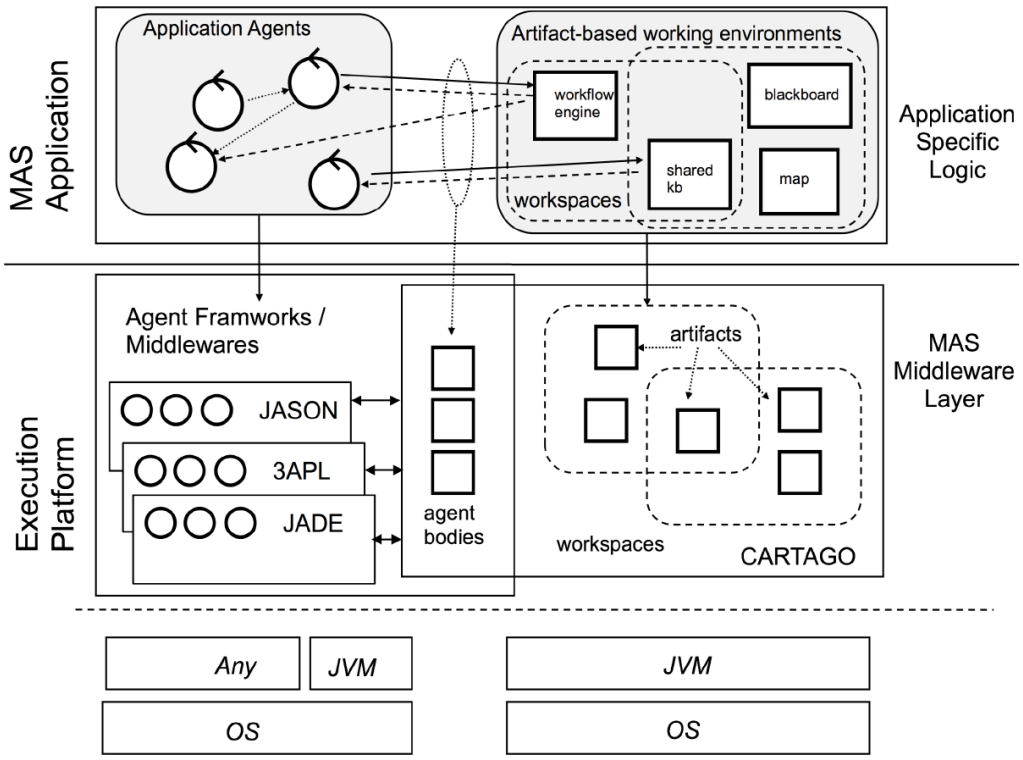

Figura 11: Rappresentazione architetturale della piattaforma CARTAGO [6]

L'architettura di tale piattaforma è schematizzata in Figura 11: è possibile osservare come è prevista l'integrazione con diversi modelli di programmazione per agenti fra cui quello che considereremo nell'ambito di questo lavoro, ovvero Jason.

3.2 Artefatti come Manageability Endpoint

L'elemento cardine dell'architettura dell'Autonomic Computing è senza ombra di dubbio l'Autonomic Manager, ma in questo studio si ritiene opportuno analizzare in prima istanza la realizzazione di Manageability Endpoint attraverso l'uso di artefatti CARTAGO.

Tale precedenza è dovuta al fatto che non è possibile definire le modalità con cui un Autonomic Manager possa gestire una risorsa, senza che prima venga definito come esso possa accedere alla risorsa e ciò avviene tramite l'utilizzo di Manageability Endpoint.

Secondo la definizione di Manageability Endpoint che si trova in letteratura, in particolare quella più esaustiva di IBM [3], tale componente deve esporre, attraverso la propria interfaccia, da un lato lo stato interno della risorsa e dall'altro le operazioni di gestione consentite all'Autonomic Manager.

L'interfaccia, come detto nella sezione 2.2.2, è composta da:

Sensors per accedere allo stato interno.

Effectors per utilizzare le operazioni di gestione della risorsa.

L'architettura di un artefatto, nel modello A&A e soprattutto nella relativa implementazione in CARTAGO, può essere rappresentata attraverso lo schema in Figura 10, dalla quale possiamo notare una naturale corrispondenza con le caratteristiche di un Manageability Endpoint.

Lo stato interno di una risorsa gestita attravero un Manageability Endpoint può dunque essere rappresentato attraverso una serie di *proprietà osservabili* di un artefatto; CARTAGO offre 2 possibilità per accedere a tali proprietà:

- `observeProperty(AId,PropertyName,?PropertyValue)`: azione che effettua una *request-response* sull' artefatto, richiesta avente come oggetto il valore di tale proprietà, il quale viene associato alla variabile specificata in `PropertyValue`.
- `focus(AId,[SId],+Filter,?InitObsPropertyValueList)`: questa azione invece effettua un'osservazione continua dell'artefatto, mettendo in atto un'interazione di tipo *publih-subscribe* in cui l'agente si registra presso l'artefatto per ricevere informazioni sullo stato di tali proprietà, quando una di queste subisce una modifica: l'agente quindi rileverà una serie di eventi relativi a cambiamenti di stato della risorsa.

Il parametro opzionale `SId` rappresenta l'identificatore di un sensore dell'agente: nel caso esso sia utilizzato, l'agente dovrà recuperare il valore della proprietà mediante l'uso dell'operazione `sense` sul sensore stesso, altrimenti tale valore sarà reso automaticamente osservabile all'agente (Jason in questo caso) mediante una percezione del tipo:

```
+PropName(PropValue)[AnnotationList].
```

Possiamo dunque definire concettualmente 2 tipologie di *Sensors* per i Manageability Endpoint realizzati con artefatti CARTAGO:

One Shot Sensors osservazione singola di una proprietà della risorsa.

Continuous Sensors osservazione continua dello stato della risorsa.

L'utilizzo dei *Continuous Sensors* potrebbe rivelarsi molto utile nella fase di **Monitor** dell'Autonomic Manager, mentre i *One Shot Sensors* potrebbero rivelarsi utili nella altre fasi, soprattutto in quella di **Analyze**.

Le operazioni esposte nelle interfacce dagli artefatti in CARTAGO possono essere utilizzate da agenti Jason attraverso l'operazione

```
use(AId,OpName,UIControlName(Params),[SId],+Timeout,+Filter)
```

il parametro opzionale `SensorId` distingue 2 differenti modalità per l'agente di recuperare l'esito dell'azione.

Allo stesso modo della osservazione di proprietà, nel caso in cui infatti venga utilizzato un particolare sensore, l'agente dovrà intenzionalmente recuperare l'evento rilevato agendo sul sensore mediante l'operazione di `sense` , mentre se tale parametro non viene specificato, tutti gli eventi generati dall'artefatto a seguito dell'esecuzione di tale operazione saranno resi automaticamente osservabili dall'agente mediante la generazione di un evento interno del tipo:

```
+Event[AnnotationList]
```

Fra le informazioni presenti nelle annotazioni troviamo per esempio l'artefatto sorgente dell'evento e l'operazione eseguita su di essa.

Gli eventi predefiniti generati dagli artefatti sono:

- `op_exec_started(OpName)`
- `op_exec_completed(OpName)`
- `op_exec_failed(OpName)`

3.3 Agenti Jason come Autonomic Manager

La scelta di utilizzare un modello di programmazione ad agenti per progettare sistemi autonomici dovrebbe apparire ben chiara, in quanto l'incapsulamento di un flusso di controllo all'interno di un'entità autonoma pro-attiva corrisponde a pieno all'idea di ciclo di controllo intelligente che sta alla base del ruolo di Autonomic Manager.

L'idea di utilizzare agenti BDI per coprire tale ruolo deve essere giustificata con motivazioni più specifiche sia da un lato prettamente teorico, spiegando come

l'architettura interna e il ciclo di ragionamento di un agente Jason siano adatti a svolgere tali compiti, sia da un lato pratico, mostrando come un programmatore di agenti Jason possa progettare Autonomic Manager.

L'obiettivo è quello di mostrare come sia possibile effettuare tale operazione, ma non dimostrare che questa metodologia sia la migliore o la più efficace, essendo un primo tentativo di studio dell'applicabilità di tali strumenti per lo sviluppo di sistemi autonomici.

3.3.1 Ciclo MAPE-K nel ciclo di ragionamento di un agente Jason

Il ciclo di controllo intelligente che caratterizza un Autonomic Manager è composto da 4 fasi distinte ovvero **Monitor**, **Analyze**, **Plan** e **Execute**.

Il ciclo di ragionamento di un agente Jason (vedi Figura 8) invece è composto da 10 passi computazionali, i quali possono svolgere alcune delle funzioni previste dalle fasi di un Autonomic Manager; ciò non significa assolutamente che un ciclo di un agente Jason corrisponda ad un ciclo MAPE in maniera diretta, in quanto mettere in atto un piano per risolvere un determinato problema comporta spesso una serie numerosa di azioni, mentre è noto che in ogni singolo ciclo di ragionamento di un agente Jason viene svolto al più un passo del piano scelto.

11

Monitor In letteratura questa fase è utile per raccogliere le informazioni provenienti dal Manageability Endpoint assegnato, mentre utilizzando agenti Jason è possibile rilevare anche eventi generati da altre entità come ad esempio altri Autonomic Manager; in questa fase possiamo infatti raccogliere i seguenti passi del ciclo di ragionamento di un agente Jason:

1) *Percezione dell'ambiente*; in ambito autonomico l'ambiente percepito è spesso il Manageability Endpoint di competenza del Manager, ma è possibile considerare anche percezioni proveniente dall'artefatto che incapsula l'interfaccia utente dell'operatore di sistema.

2) *Aggiornamento della base di conoscenza*.

3) *Ricezione di comunicazioni da parte di altre entità*; in particolare nel nostro caso di studio questi possono essere sia Autonomic Manager che gestiscono anch'essi una determinata risorsa, sia manager di coordinazione definiti come Orchestrating Autonomic Manager nella sezione precedente.

Analyze Compito di questa fase è riuscire a rilevare particolari condizioni per la risorsa gestita utilizzando le informazioni a disposizioni del manager; un agente Jason può recuperare le informazioni utili a tale scopo da tre fonti distinte: la

propria base di conoscenza, comunicazioni dirette ricevute da altri agenti o interrogazione di artefatti sociali di coordinazione presenti nello spazio di lavoro considerato.

Tralasciando il recupero di informazioni da quest'ultima fonte, la quale richiede un impegno computazionale da parte dell'agente ben maggiore di un singolo passo del proprio ciclo di ragionamento, possiamo ritenere attinenti a questa fase i seguenti passi:

4) *Selezione dei messaggi "socialmente accettabili"*; passo in cui vengono filtrate le comunicazioni da parte di altri agenti.

5) *Selezione di un evento*; vi deve essere un ultimo evento che faccia sì che tutte le condizioni di una particolare situazione risultino verificate e si possa procedere cone la gestione autonomica di tale situazione.

6) *Recupero di tutti i piani rilevanti*; dal nostro punto di vista questo passo appartiene maggiormente alla fase di Analyze, sia a quella di Plan, in quanto non si effettua alcuna operazione di pianificazione di azioni da eseguire, ma piuttosto si effettua un analisi dei piani rilevanti, ovvero attinenti ad una situazione di interesse per la risorsa gestita.

Plan Questa fase comporta la scelta delle azioni necessarie alla risoluzione dei problemi rilevati in seguito alla analisi della risorsa gestita; la letteratura relativa all'Autonomic Computing non definisce in maniera specifica le modalità con cui tale pianificazione debba avvenire, mentre nell'ambito dell'intelligenza artificiale esistono diverse tecniche a riguardo.

Il Practical Reasoning nel modello BDI (e quindi anche in Jason) è quella parte di ciclo di ragionamento che permette all'agente di determinare la sequenza di azioni che andrà a compiere ed esso è composto da 2 attività principali, *Deliberation* e *Mean-Ends Reasoning*, le quali ricoprono entrambe un ruolo nei compiti che abbiamo assegnato agli Autonomic Manager.

La fase di *Deliberation* stabilisce quale debba essere l'obiettivo dell'agente fra le alternative disponibili; tale fase in un sistema autonomico assume un ruolo importante quanto un problema può essere risolto seguendo differenti strade.

La fase di *Mean-Ends Reasoning* invece definisce, date le circostanze ambientali rilevate dall'agente, una serie di azioni da mettere in atto per raggiungere l'obiettivo scelto; l'agente secondo questo modello deve essere fornito di piani parziali predefiniti, in quanto la fase di pianificazione di un agente Jason consiste nel comporre in maniera efficace tali piani: ciò è previsto anche nell'architettura dell'Autonomic Manager definita in letteratura, in quanto essa prevede l'utilizzo di una conoscenza interna a tale componente che possiamo associare all'insieme di piani definiti per un agente Jason.

I passi computazionali che possiamo associare a tale fase sono i seguenti:

7) *Determinazione dei piani applicabili*; questo passo corrisponde alla fase di *Deliberation* di un agente BDI.

8) *Selezione di un piano applicabile*; questo passo corrisponde invece alla fase di *Mean-Ends Reasoning* di un agente BDI.

Execute Dal punto di vista concettuale l'analisi di questa fase in ottica di agenti Jason non sembra presentare grosse difficoltà, però è opportuno effettuare alcune precisazioni.

Dalla visione fornita dalla letteratura relativa ai sistemi autonomici sembra emergere come in questa fase si debbano eseguire in sequenza tutte le azioni che compongono il piano elaborato alla fase precedente, seppur verificando di passo in passo le condizioni per la realizzazione di tali scopi.

Un agente Jason offre molto di più di quanto necessario per implementare un comportamento di questo tipo, in particolare esso è in grado di istanziare più piani e inserirli in una coda di esecuzione: ad ogni ciclo di ragionamento esso sceglie una sola azione facente parte del piano in cima alla coda di esecuzione [1], controlla che le condizioni per l'esecuzione di essa siano valide, la esegue ed infine rimette il piano istanziato (chiamato intenzione) in fondo a tale coda.

Questo meccanismo fa sì che sia possibile realizzare Autonomic Manager in grado di gestire molteplici situazioni di interesse per la risorsa gestita, ad esempio relative a differenti proprietà *Self-**; inoltre personalizzando la funzione di selezione delle intenzioni è possibile assegnare differenti priorità e in tale maniera potrebbe essere una buona soluzione definire una priorità alta ai piani relativi a proprietà quali *Self-Protecting* o *Self-Healing*, piuttosto che *Self-Optimising*.

I passi computazionali che possiamo associare a tale fase sono i seguenti:

9) *Selezione dell'intenzione per l'esecuzione*.

10) *Esecuzione di un passo del piano istanziato selezionato*.

3.3.2 Implementare le funzioni di un Autonomic Manager in Jason

In questa sezione si analizzerà in che modo possa essere programmato il corpo di un agente Jason per implementare le funzionalità autonomiche desiderate.

Le tre dimensioni in cui si articola il corpo di un agente Jason sono quelle di **Beliefs**, **Goals** e **Plans**, mentre il compito attribuito da un Autonomic Manager è un ciclo di controllo intelligente suddivibile nelle 4 fasi ampiamente descritte in precedenza.

[1] A meno di una diversa configurazione data dal programmatore all'agente Jason

Nella sezione 2.3 abbiamo mostrato come l'introduzione di funzionalità autonomiche in un sistema informatico possa avvenire per gradi e un metodo per analizzare l'implementazione di Autonomic Manager potrebbe essere quello di procedere cercando di realizzare progressivamente tali livelli con agenti Jason.

Manuale A livello **Manuale** il sistema informatico è privo di funzionalità autonomiche e, volendo realizzare tali funzioni di gestione con il paradigma mostrato, gli agenti Jason avrebbero il compito di rilevare le richieste fatte dall'operatore del sistema tramite GUI o Console, elaborarle agendo sul Manageability Endpoint, eventualmente ottenere il risultato dell'operazione e inviare tale esito all'operatore che ha fatto la precedente richiesta: l'agente Jason funge in tal senso da Bridge fra la risorsa e l'interfaccia dell'operatore.

Non si intende analizzare più a fondo questa eventualità, in quanto non è necessario utilizzare un nuovo approccio ad agenti per realizzare tale sistema e soprattutto è possibile farlo in maniera efficiente utilizzando vecchi modelli di programmazione già affermati.

Controllo A livello **Controllo** gli Autonomic Manager devono implementare la funzione di **Monitor** della risorsa attraverso l'interfaccia esposta dal Manageability Endpoint.

La funzione principale svolta dall'Autonomic Manager è quello di rilevare particolari condizioni nello stato della risorsa gestita e notificarle all'operatore tramite l'interfaccia fornita (GUI o Console).

Le operazioni iniziali che deve compiere un Autonomic Manager sono 3:

- La fase di inizializzazione che consiste nell'entrare all'interno dello spazio di lavoro e recuperare il riferimento al Manageability Endpoint di competenza: tale fase potrà essere eventualmente coadiuvata dall'utilizzo di un file di configurazione e/o la cooperazione con un agente avente tale compito.
- Il recupero del riferimento all'artefatto che verrà utilizzato dall'Autonomic Manager per notificare le situazioni rilevate all'operatore di sistema.
- L'osservazione continua dello stato della risorsa mediante l'uso dell'operazione `focus` fornita dalla libreria che integra CARTAGO in Jason.

Per fare in modo che l'Autonomic Manager esegua tali operazioni è necessario imporre nella Belief Base un Goal del tipo:

```
!init.
```

e definire un piano strutturato in linea di massima come segue:

```
+!init: true <-
 cartago.lookupWsp("AMspace",WspId,"localhost");
 cartago.joinWsp(WspId);
 cartago.lookupArtifact("ResNameME",ResMEId);
 +resource(ResMEId);
 cartago.lookupArtifact("OperatorUI",UIId);
 +user_interface(UIId);
 cartago.focus(ResMEId).
```

Dato che nell'operazione di `focus` non è stato specificato alcun sensore, eventuali aggiornamenti alle proprietà osservabili definite nel Manageability Endpoint saranno automaticamente rese osservabili all'agente sotto forma di percezioni: per far sì che l'Autonomic Manager monitori la risorsa è sufficiente definire piani innescati da tali percezioni.

Tali piani verificheranno nel contesto che tale proprietà assuma un valore di interesse per le funzionalità autonomiche desiderate, mentre nel corpo del piano si utilizzerà l'artefatto che fornisce l'accesso all'interfaccia utente per poter passare tale informazione all'operatore; un piano di questo genere può essere rappresentato con una struttura simile a quella seguente:

```
+property1(Val) : Val < MinValue <-
   -+resource(property1,Val);
   ?user_interface(UIId);
   cartago.use(UIId, notify(property1,Val,MinValue)).
```

Nel piano è importante sottolineare come non si effettua esclusivamente la notifica all'interfaccia utente dello stato di tale parametro della risors, ma viene inoltre effettuato un aggiornamento della Belief Base, in modo che l'Autonomic Manager disponga di dati sempre coerenti con lo stato della risorsa di cui è responsabile.

Nel corpo dell'agente appariranno una serie di piani di questo genere, tanti quante sono le condizioni delle proprietà della risorsa interessanti per le funzionalità autonomiche introdotte.

La fase di **Monitor** può essere realizzata con gli elementi descritti in precedenza, però rimane indefinita la modalità con cui l'operatore può successivamente operare su tale risorsa, ammettendo che le funzioni di gestione della stessa siano definite attraverso opportune operazioni dell'artefatto che costituisce il Manageability Endpoint; in particolare si aprono 2 possibilità:

- Fare in modo che attraverso l'interfaccia utente dell'operatore si possa agire direttamente sul Manageability Endpoint, sfruttando la possibilità di collegare artefatti offerta da CARTAGO.

- Rendere l'Autonomic Manager sensibile ad eventi generati dall'interfaccia utente, effettuando un monitoraggio anche su tale artefatto, e assegnare a quest'ultimo il compito di operare sul Manageability Endpoint.

Tali soluzioni presentano entrambe vantaggi e svantaggi: nel primo caso si otterebbe una semplificazione delle interazioni del sistema, in quanto vi sarebbe un passaggio in meno per operare sulla risorsa, ma non si rispetterebbe il principio concettuale definito nella letteratura relativa all'Autonomic Computing, il quale evidenzia come l'unica entità ad operare su un Manageability Endpoint debba essere un Autonomic Manager. Nel secondo caso avremmo viceversa un rispetto rigoroso dell'architettura dell'Autonomic Computing, a dispetto di un incremento dei compiti e delle responsabilità attribuite all'Autonomic Manager.

A questo proposito riteniamo opportuno optare per la seconda possibilità, in quanto in tale maniera vi è un incapsulamento della responsabilità di gestione della risorsa interamente a carico dell'Autonomic Manager in modo da rendere l'architettura globale del sistema autonomico più semplice e lineare, a scapito di una complessità maggiore dei singoli elementi; tale scelta inoltre predispone l'agente Jason all'implementazione delle successive fasi del ciclo MAPE-K, in particolare quella di **Execute**.

L'implementazione di questa funzione di monitoraggio dell'interfaccia utente da parte dell'Autonomic Manager richiede 2 passi: innanzitutto occorre aggiungere alla fine del corpo del piano di inizializzazione una formula del tipo

`cartago.focus(UIId).`

In seconda istanza è necessario aggiungere i piani che gestiscano gli input immessi dall'operatore nell'interfaccia utente; questi dovranno essere notificati da parte dell'artefatto GUI attraverso proprietà osservabili, il cui aggiornamento innescherà l'esecuzione di piani da parte dell'Autonomic Manager; i parametri verrano recuperati dall'agente attraverso la rilevazione del valore di altre proprietà osservabili dell'artefatto.

Un piano di questo tipo assumerà una struttura in linea di massima simile a quella che segue:

```
+opName(_)[source(GUI)]:true <-
                cartago.lookupArtifact(GUI,UIId);
                cartago.observeProperty(UIId,op_par1(Par1));
                cartago.observeProperty(UIId,op_par2(Par2));
                ?resource(ResMEId);
                cartago.use(ResMEId,operation(Par1,Par2)).
```

A discrezione del progettista sarà possibile affinare tali piani, in modo che venga notificata all'operatore la corretta o meno esecuzione dell'operazione all'-operatore.

Un problema che si evince da questo tipo di soluzione è la complessità che assume l'artefatto che incapsula l'interfaccia utente, in quanto viene definita una proprietà osservabile per ciascuna operazione eseguibile su ogni risorsa e per ciascun parametro della stessa: appare evidente che se attraverso tale interfaccia avviene la gestione di molte risorse, ciascuna delle quali dispone una molteplicità di operazioni di controllo, la struttura dell'artefatto risultante sarà decisamente complessa.

Una possibile soluzione a tale inconveniente potrebbe essere quella di scomporre tale interfaccia in più artefatti ognuno che si occupi dell'interazione con una sola risorsa e fare in modo che appaia in maniera unica all'operatore.

Analisi Il livello di **Analisi** si differenzia dal precedente dal fatto che l'Autonomic Manager deve essere in grado di correlare le informazioni rilevate sulla risorsa relative ad una singola proprietà o evento, con i dati a disposizione dell'Autonomic Manager, i quali possono essere relativi sia a fonti di conoscenza esterne (artefatti sociali per esempio), sia ad altre proprietà osservabili della risorsa gestita.

La struttura dell'Autonomic Manager descritta per il livello precedente non muta in sostanza, però si rendono necessarie modifiche al corpo dei piani implementati per effettuare un'analisi avanzata delle condizioni della risorsa: tale analisi può essere effettuata in 2 modi:

- Inserire nel contesto del piano di ulteriori condizioni relative alla Belief Base dell'agente.
- Inserire nel corpo del piano goal che spingano l'agente a verificare le altre condizioni attraverso l'esecuzione di piani aventi lo scopo di ottenere tali informazioni.

Utilizzare in maniera esclusiva la prima possibilità è utile nei casi in cui l'analisi della risorsa non richieda che l'Autonomic Manager debba recuperare informazioni esterne alla risorsa stessa.

La notifica all'interfaccia utente non sarà quindi la notifica del valore di una singola proprietà delle risorsa, ma dovrà rappresentare un'informazione completa della situazione generale, in modo che l'operatore possa capire in che modo agire su di essa in maniera agile.

Un piano di questo tipo potrebbe essere strutturato come segue:

```
+property(Val) : Val < MinValue &
                 resource(prop2,Prop2val) &
                 Prop2val ==’Prop2_relevant_val’ &
                 resource(prop3,Prop3val) &
```

```
                Prop3val ==’Prop3_relevant_val’ <-

                ?user_interface(UIId);
                cartago.use(UIId, notify(Info,Details)).
                //Info = [’Notify’|’Warning’|’Error’|’Problem’]
```

La seconda possibilità mette in condizione l’agente di effettuare un’analisi avanzata della risorsa e in generale dell’ambiente in cui essa opera: vi saranno piani specifici per poter recuperare determinate informazioni; ciò può avvenire attraverso l’uso di artefatti sociali in cui vengono memorizzati tali dati o attraverso la comunicazione diretta con altri agenti.

Per quanto riguarda l’uso di artefatti per il recupero di informazioni, è preferibile l’uso dei sensori, in modo da mantenere compatto il codice relativo a tale piano, mentre

```
+property(Val):Val < MinValue <-
                !check(info1);
                ?info(info1,Info1Value);
                Info1Value == ’Info1_relevant_val’;
                !check(info2);
                ?info(info2,Info2Value);

                ?user_interface(UIId);
                cartago.use(UIId, notify(Info,Details)).
                //Info = [’Notify’|’Warning’|’Error’|’Problem’]

+!check(info1):true <-
                ?knowledge_artifact(KBId);
                cartago.use(KBId,retrieve(info1),mySensor0);
                cartago.sense(mySensor0,info1(Value));
                -+info(info1,Value).

+!check(info2):true <-
                ?agentWithInfo2(AgentRef);
                .send(AgentRef,askOne,info(info2,Value),Answer,5000);
                Answer=/=’timeout’;
                -+Answer. //Answer unifies with info(info2,AnswerValue)
```

In questo caso definiamo un piano specifico per la verifica di ciascuno dei 2 parametri, differenti da quello che ha innescato l’analisi, in cui nel primo si effettua un’interazione di tipo *request-response* con un artefatto sociale il quale rappresenta un contenitore di conoscenza condiviso nello spazio di lavoro degli agenti.

L'aggiornamento delle informazioni contenute in esso potrà essere effettuato sia dai singoli Autonomic Manager durante l'esecuzione delle proprie intenzioni, sia eventualmente da uno o più Orchestrating Autonomic Manager ai quali viene assegnato il compito di collezionare una serie di informazioni dalle varie componenti del sistema, elaborarli e immagazzinari in maniera intelligibile per gli altri Autonomic Manager.

Nel secondo piano di verifica l'interazione avviene invece in maniera diretta con un altro Autonomic Manager, mediante l'uso dell'azione interna `send` con un attesa temporanea della risposta dell'interlocutore: non viene sospesa l'esecuzione dell'intero operato dell'Autonomic Manager, ma solo l'esecuzione del piano istanziato contenente tale interazione.

Questa possibilità può essere particolarmente utile nel caso in cui in uno stesso spazio di lavoro risiedano più istanze della stessa risorsa (e.g. Virtual Machine) e un Autonomic Manager può chiedere ad un suo pari il carico di lavoro della risorsa da lui gestita: nel caso questa sia bassa può suggerire all'operatore di spostare parte del carico di lavoro della propria risorsa a quella più scarica; al livello successivo vedremo come tale operazione possa invece essere resa completamente trasparente e non sia necessaria l'approvazione dell'operatore.

La struttura dei piani per rilevare le azioni dell'operatore sulla interfaccia utente rimangono inalterati, compresa l'esecuzione di operazioni sul Manageability Endpoint.

Ciclo di Controllo Questo livello si differenzia dal precedente per il fatto che l'Autonomic Manager, oltre ad effettuare un'analisi complessa della risorsa e dell'ambiente in cui essa opera, ha il compito anche di elaborare un possibile piano per risolvere il problema rilevato o in ogni caso migliorare le condizioni di lavoro della risorsa.

Tale piano dovrà tenere conto delle politiche di business assegnate a ciascun Autonomic Manager dall'amministratore e in ogni caso dovrà ottenere l'approvazione dell'operatore prima di essere messo in atto; la notifica all'interfaccia utente dovrà in questo caso fornire un'informazione completa riguardante sia il problema riscontrati, sia la serie di operazioni necessarie suggerite dall'Autonomic Manager per la risoluzione dello stesso, ciò non modifica in ogni caso la modalità di interazione fra Autonomic Manager e interfaccia utente.

Le 2 questione più rilevanti da considerare a questo punto sono:

- Fare in modo che l'Autonomic Manager recepisca le politiche di business e sia incline a modificare il proprio comportamento a seguito di cambiamenti nelle politiche stesse.

- Costruire la libreria di piani dell'agente in maniera tale che, attraverso il

Practical Reasoning, si possa giungere alla definizione di una sequenza di azioni valida per la risoluzione del problema e da sottoporre all'attenzione dell'operatore.

Una politica di business ha lo stesso ruolo di un file di configurazione: in avvio dell'esecuzione delle proprie attività l'Autonomic Manager deve configurarsi per svolgere correttamente il proprio compito considerando il particolare contesto in cui si trova, in quanto lo si vuole progettare in maniera riusabile.

Riusabilità in questo ambito significa per ciascuna classe di risorsa sarà necessario progettare un unico Autonomic Manager di cui ne verrà creata un'istanza per ciascuna risorsa reale della stessa classe presente nel sistema informatico.

Le singole istanze dello stesso tipo di risorsa potranno infatti operare in differenti condizioni e in diverse modalità, in quanto in ogni spazio di lavoro potrebbero esservi esigenze specifiche e il compito delle politiche di business è quello di permettere all'Autonomic Manager di adattarsi e di operare correttamente nell'ambiente in cui si trova.

La soluzione proposta consiste nella creazione di un artefatto personale per l'Autonomic Manager nel quale incapsulare tali politiche; esso andrà creato in fase di inizializzazione e andrà a caricare le politiche di business definite dall'amministratore di sistema su una risorsa fisica esterna (per esempio un file XML o un file di proprietà Java).

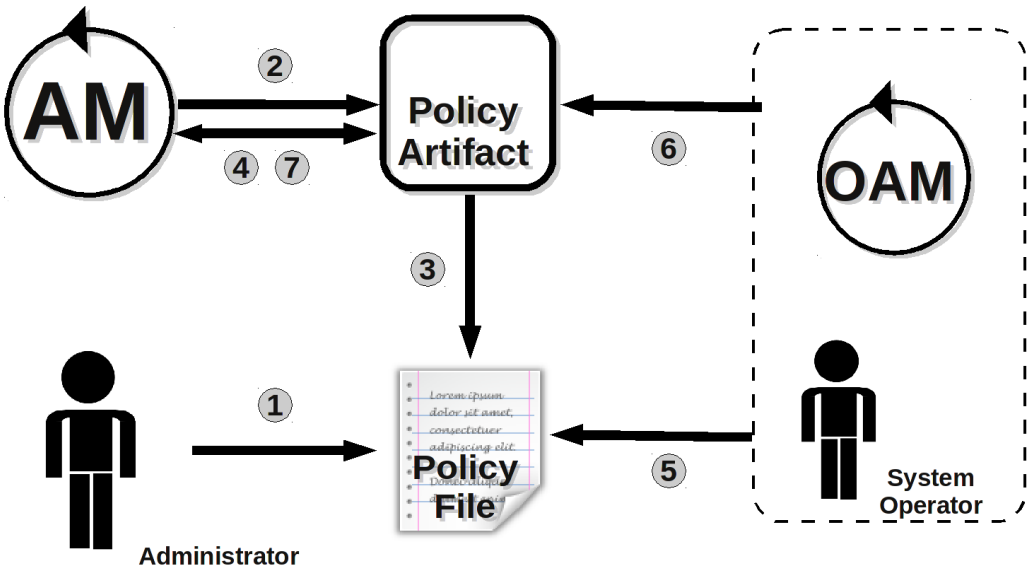

Figura 12: Fasi relative al caricamento e all'aggiornamento delle politiche di business di un Autonomic Manager

Quest'operazione, come mostrato in Figura 12, sarà caratterizzata dalle seguenti fasi:

1. L'amministratore crea il file delle politiche con i parametri base.
2. L'Autonomic Manager crea il Policy Artifact e comincia l'osservazione di tale artefatto.
3. Il Policy Artifact carica le informazioni contenute dentro il file delle politiche durante la propria operazione di `INIT`.
4. Il Policy Artifact segnala all'Autonomic Manager la necessità di aggiornare le politiche di business: ciò avviene mediante la specifica di quali piani l'Autonomic Manager deve attivare e i parametri da utilizzare in essi.
5. Durante lo svolgimento delle attività del sistema si può rendere necessaria la modifica delle politiche di business dell'Autonomic Manager. Ciò può avvenire mediante una modifica manuale da parte dell'operatore o da un'entità autonomica di più alto livello come gli Orchestrating Autonomic Manager.
6. L'entità che ha apportato tale modifica, forzerà l'aggiornamento del Policy Artifact in modo che esso rimanga coerente con le nuove politiche di business stabilite.
7. Come al punto 4, il Policy Artifact segnala all'Autonomic Manager la necessità di un aggiornamento delle politiche di business.

Il file delle politiche di business dovrà contenere essenzialmente due diverse tipologie di informazioni: da un lato informazioni a partire delle quali l'Autonomic Manager possa elaborare e adattare la propria libreria di piani, mentre dall'altro i parametri di configurazione da aggiungere come predicati nella propria Belief Base, i quali verranno poi utilizzati negli stessi piani elaborati.

Il processo di elaborazione delle informazioni che porta alla costruzione/modifica della libreria dei piani dell'agente Jason necessita uno studio specifico molto elaborato che va dalla strutturazione delle informazioni all'interno del file delle politiche fino al processamento delle stesse in modo da ottenere modifiche alla libreria di piani dell'Autonomic Manager, nei termini di aggiunta, rimozione o modifica di piani.

Il livello attuale di maturità di questo approccio, soprattutto dal punto di vista pratico, è ancora troppo basso per poter delineare in maniera efficace tale processo, per cui si è deciso per il momento di inserire nel file delle politiche direttamente gli interi piani che dovranno essere attivati, eventualmente sostituendo alcuni di quelli presenti.

Il piano relativo alle attività iniziali dell'Autonomic Manager dovrà dunque comprendere le azioni elencate al punto 2 della lista precedente e potrebbe assumere una forma del tipo:

```
+!init: true <-
  cartago.lookupWsp("AMspace",WspId,"localhost");
  cartago.joinWsp(WspId);

  +!make_Policy_Artifact;

  +!discover_ME;

  +!discover_GUI.

+!make_Policy_Artifact: true <-
  .my_name(Name);
  .concat("AM_",Name,CName);
  .concat(CName,"_Pol_Art",PolArtName);

  cartago.makeArtifact(PolArtName,
                       "artifacts.pol.PolicyArtifact",
                       ["/path_to/PolicyFile"]);
  +policy_artifact(PolArtName);
  cartago.focus(PolArtName);

+!discover_ME : true <-
  cartago.lookupArtifact("ResNameME",ResMEId);
  +resource(ResMEId);
  cartago.focus(ResMEId);

+!discover_GUI : true <-
  cartago.lookupArtifact("OperatorUI",UIId);
  +user_interface(UIId);
  cartago.focus(UIId);
```

L'interazione fra Autonomic Manager e il Policy Artifact per il recupero delle politiche di business può essere modellata secondo due differenti schemi:

- Il Policy Artifact segnala esclusivamente la disponibilità delle politiche aggiornate all'Autonomic Manager, cui spetta poi il compito di interrogare l'artefatto per recuperare piani e parametri di configurazione, attraverso la tipica interazione fra agenti e artefatti `use-sense`.

- Il Policy Artifact segnala singolarmente i piani da attivare e rimuovere all'agente attraverso eventi osservabili all'agente, che dovranno essere opportunamente gestiti.

Per il momento prendiamo in considerazione la seconda possibilità e vediamo come dovrebbe essere strutturato il corpo dell'Autonomic Manager relativamente a tali operazioni; vediamo come la possibilità di assegnare etichette ai piani fornita da Jason e quelle di aggiungere e rimuovere piani attraverso l'uso di azioni interne siano di grande aiuto per tale scopo.

Il Policy Artifact dovrà segnalare 3 differenti tipologie di eventi:

- Aggiunta di un nuovo piano: nel caso esso sia già presente esso verrà prima rimosso e poi aggiunto con le nuove specifiche.
- Rimozione di un piano: se nel corpo dell'agente non esiste un piano avente l'etichetta specificata tale evento non produrrà alcun effetto.
- Aggiornamento di un parametro di configurazione.

A seguito di questi eventi l'agente dovrà eseguire alcune operazioni in modo tale da adeguarsi alle nuove specifiche; i piani aventi tale scopo potranno essere strutturati in linea di massima come segue:

```
+add_plan(PLabel,PBody)[source(PAId)]: true <-
                              ?policy_artifact(PAId);
                              .remove_plan(PLabel);
                              .add_plan(PBody).

+remove_plan(PLabel)[source(PAId)]: true <-
                              ?policy_artifact(PAId);
                              .remove_plan(PLabel).

+update_param(Param)[source(PAId)]: true <-
                              ?policy_artifact(PAId);
                              -+Param.
```

Un modo efficiente per etichettare i piani di un Autonomic Manager potrebbe essere quello di creare etichette nella forma:

```
@self-*_property(plan_function,plan_name)
```

in tale modo si applica una organizzazione dei piani a seconda dello scopo a cui

asseriscono, determinato dalla proprietà *Self-** a cui appartengono; inoltre è possibile disattivare in maniera istantanea tutti i piani relativi ad un'intera proprietà o ad una specifica funzione della stessa, mentre sarà necessario un singolo inserimento per ogni piano da aggiungere.

La fase di pianificazione, nel senso inteso dalla letteratura relativa all'Autonomic Computing e non il Practical Reasoning dell'agente Jason, comporta la necessità di progettare all'interno del corpo dell'agente Jason piani che portino a determinare una possibile soluzione di un problema, partendo dalle condizioni che lo hanno generato; a tale scopo può essere utile introdurre la nozione di **necessità** (**need()**) come il risultato finale della fase di analisi del ciclo MAPE-K di un Autonomic Manager.

A termine della suddetta fase un Autonomic Manager ora non deve più notificare direttamente l'interfaccia utente dell'operatore di sistema, bensì produrre una possibile serie di operazione che l'Autonomic Manager per migliorare lo stato di produttività della risorsa.

Una prima modifica ai piani che portavano al completamento della fase di analisi potrebbe essere dunque l'aggiunta alla Belief Base di predicati che rappresentino le necessità per la risoluzione del problema, che saranno utilizzati per innescare le fasi successive della pianificazione; inoltre, per poter sottoporre al termine della pianificazione la soluzione all'operatore del sistema, è opportuno memorizzare in maniera ordinata ciascuna operazione consigliata all'interno della memoria dell'agente.

La struttura del corpo dell'agente relativa alla pianificazione potrà dunque assumere una struttura del seguente tipo:

```
 +property(prop_name,Val):Val < prop_low_limit(prop_name,MinValue) <-
                ?last_problem_id(N);

                ?prop_high_limit(prop_name,MaxValue);
                +prob_sol(N+1,[]);
                +prob_needs(N+1,1);
                +prob_descr(N+1,info_prob,details);
                need(prop_name,Val,(MinValue+MaxValue)/2,N+1);
                -+last_problem_id(N+1).

+need(prop_name,CurrVal,DesiredValue,ProbId): other_cond(....) <-

                ?prob_sol(ProbId,L);
                .concat(L,[op(op1name,params)],LL);
                .concat(LL,[op(op2name,params)],FL);
                -+prob_sol(ProbId,FL);
```

```
                ?prob_needs(ProbId,N);
                -+prob_needs(ProbId,N-1).

+prob_needs(ProbId,0) : true <-
                ?prob_sol(ProbId,OpList);

                ?user_interface(UIId);
                ?prob_descr(ProbId,Info,Details);
                cartago.use(UIId,
                            notify(problem(Info,Details),
                            solution(OpList));
                //Info = ['Notify'|'Warning'|'Error'|'Problem']

                -prob_needs(ProbId,_);
                -prob_sol(ProbId,_).
```

In questo semplice esempio un Autonomic Manager rileva una particolare proprietà avente un valore troppo basso e genera una necessità che viene gestita da un apposito piano il quale aggiunge una serie di operazioni alla soluzione del problema, la quale verrà consigliata all'operatore tramite notifica nell'interfaccia utente; è possibile tramite questa struttura gestire condizioni in cui vi siano più necessità per la risorsa: la notifica all'interfaccia utente avverrà solamente nel momento in cui saranno definite operazioni che soddisfino tutte le necessità.

Ciclo di Controllo con Politiche di Business Le politiche di business sono già state introdotte al livello precedente, ma ora determinano l'effettiva esecuzione di operazioni sulla risorsa gestita, in quanto una volta determinata una possibile soluzione ad un problema, coerente con tali politiche, essa verrà direttamente messa in atto dall'Autonomic Manager, senza la specifica approvazione dell'operatore del sistema.

L'interfaccia utente a questo livello verrà utilizzata per 2 scopi: il primo è quello di monitorare parametri di gestione del sistema e mostrare lo storico delle operazioni effettuate, mentre il secondo consiste nella gestione delle politiche di sistema, adattandole a tempo di esecuzione alle esigenze decise dall'amministratore o responsabile del caso.

La sostanziale modifica rispetto al livello precedente da apportare al corpo dell'Autonomic Manager riguarda l'esecuzione delle operazioni necessarie alla risoluzione del problema individuato, memorizzate come abbiamo visto in precedenza in una lista; essa può essere implementata come segue:

```
+prob_needs(ProbId,0) : true <-
                ?prob_sol(ProbId,OpList);
```

```
                !execute(ProbId,Oplist).


+!execute(ProbId,[op(Name,Params)|L]) : true <-
                ?resource(ResMEId);
                cartago.use(ResMEId,Name(Params));
                !execute(ProbId,L).

+!execute(ProbId,[]) : true <-
                ?prob_descr(ProbId,Info,Details);
                ?prob_sol(ProbId,OpList);
                ?user_interface(UIId);
                cartago.use(UIId,
                            notify(problem(Info,Details),
                            solution(OpList));
                //Info = ['Notify'|'Warning'|'Error'|'Problem']
                -prob_descr(ProbId,_,_);
                -prob_needs(ProbId,_);
                -prob_sol(ProbId,_).
```

Lo schema di progettazione dei piani mostrato è ampiamente incompleto, in quanto ad esempio per gli ultimi due livelli di autonomicità si presentano problematiche da gestire in maniera approfondita.

Ad esempio vi sono i casi in cui un Autonomic Manager non riesca a determinare una soluzione per una singola necessità di un problema (il quale potrebbe eventualmente presentarne più di una): in queste circostanze è opportuno gestire il comportamento dell'agente attraverso l'uso di timeout e gestione di piani falliti, funzionalità offerte da Jason.

3.4 Autonomic Computing attraverso Multi-Agent Systems

Nei precedenti punti di questa sezione abbiamo visto com'è possibile creare entità che forniscano funzionalità autonomiche ad un sistema informatico, ma non è ancora stato chiarito come è possibile integrare queste in un sistema informatico già esistente o in ogni caso realizzato con paradigmi di programmazione object-based.

Tale integrazione comporta una problematica di rilievo per la corretta efficacia delle funzionalità autonomiche, in quanto le risorse che costituiscono il sistema, siano esse di tipo fisico come dispositivi hardware o di tipo virtuale come applicazioni software, sono caratterizzate da una natura passiva; d'altro canto anche i

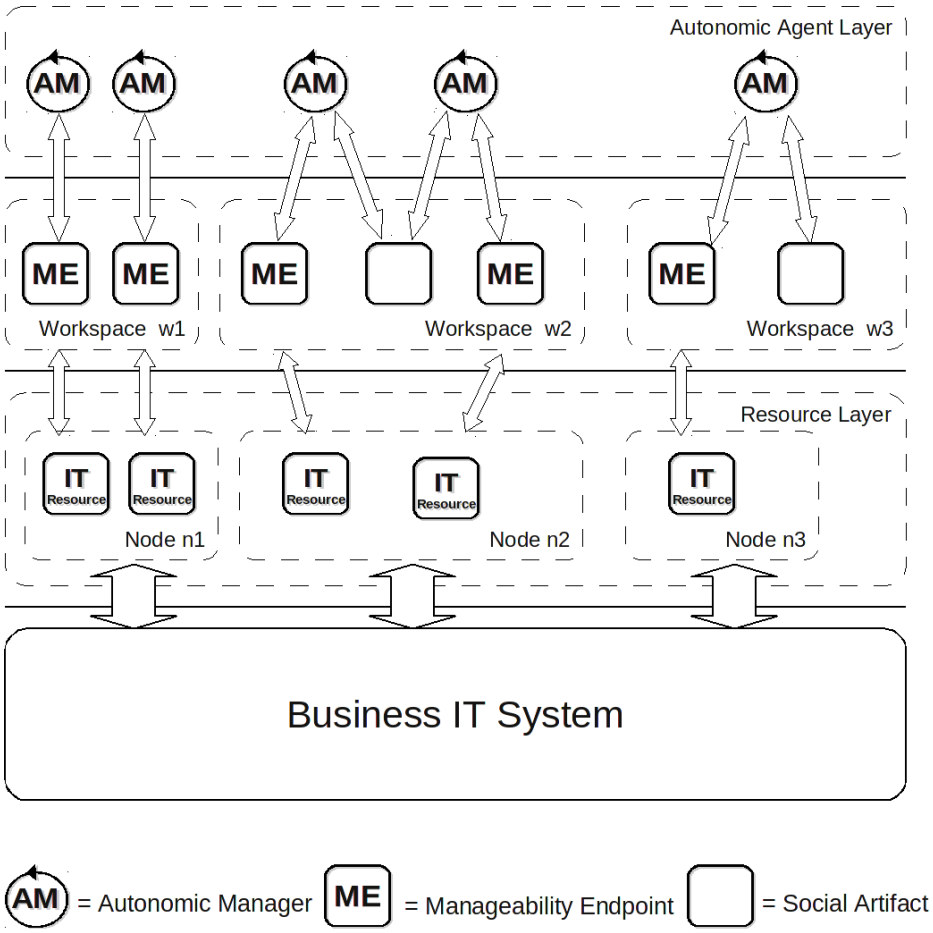

Figura 13: Rappresentazione di un sistema informatico dotato di funzionalità autonomiche realizzate con Jason e CARTAGO.

Manageability Endpoint sono entità di tipo passivo e fungono da interfacce fra gli Autonomic Manager e le risorse stesse.

In uno scenario del genere, mostrato in Figura 13, quando l'esecuzione delle normali funzioni di business del sistema informatico portano a sostanziali modifiche delle proprietà interne delle risorse, tali cambiamenti non possano essere recepiti in alcun modo dai Manageability Endpoint e di conseguenza nemmeno dagli Autonomic Manager, il cui ruolo si fonda appunto sul monitorare le proprietà della risorsa assegnatagli.

A tale proposito si potrebbe ritenere necessario intervenire nella architettura generale del sistema informatico per rendere possibile un'effettiva integrazione fra le funzioni di business e quelle autonomiche.

Sono possibili molteplici soluzioni architetturali che permettono di ovviare a tale problema di consistenza di informazioni fra Manageability Endpoint e risorsa reale gestita.

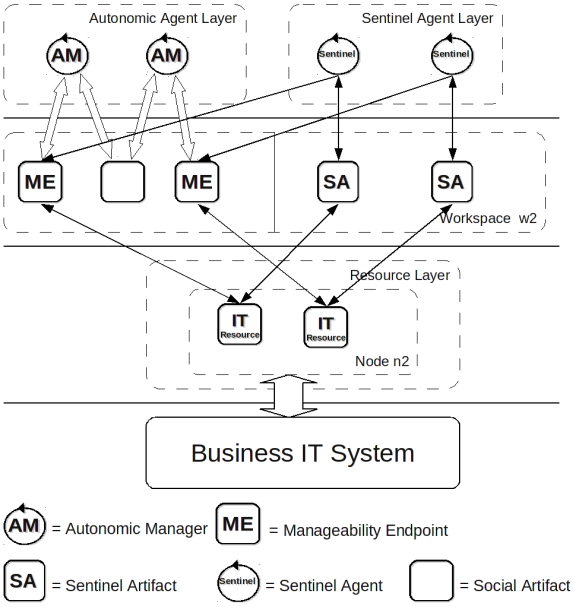

Figura 14: Integrazione fra le funzionalità autonomiche e le funzionalità di business in un sistema informatico: soluzione che adotta agenti di monitoraggio delle risorse.

- Inserire un insieme di agenti (chiamati **Sentinel**) a cui viene assegnata una risorsa ciascuno, aventi il compito di monitorare le proprietà di interesse dal punto di vista di gestione autonomica e aggiornare le proprietà osservabili del Manageability Endpoint; l'accesso alla risorsa dovrà avvenire mediante nuo specifico artefatto (vedi Figura 14).

- Realizzare l'intera parte del sistema informatico che si occupa di funzioni di business attraverso l'uso del paradigma basato ad agenti, in particolare con l'uso di agenti Jason e dell'infrastruttura fornita da CARTAGO (vedi Figura 15). Questa soluzione dipende dalla possibilità di utilizzare tali risorse attraverso questo nuovo modello di programmazione.

- Creare un livello di astrazione intermedio che funga da intermediario fra le funzionalità di business e le risorse stesse. Tale livello costituito da agenti e artefatti farà in modo che le richieste di business vengano inoltrate alle risorse e faranno in modo che a seguito delle stesse i Manageability Endpoint aggiornino le informazioni di cui devono disporre (vediamo Figura 16).

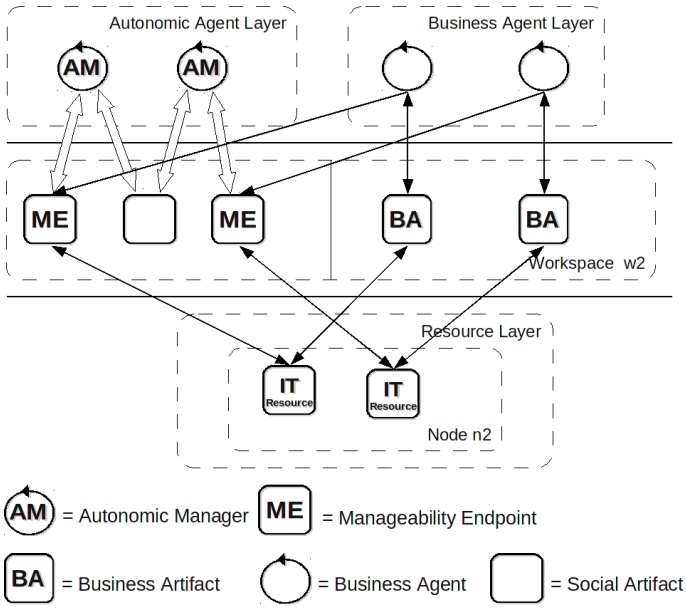

Figura 15: Integrazione fra le funzionalità autonomiche e le funzionalità di business in un sistema informatico: soluzione in cui le funzioni di business vengono realizzate attraverso un modello di programmazione basato ad agenti

La soluzione più lineare in grado di mantenere separata la parte di sistema informatico relativa alla produttività a quella relativa a funzionalità autonomiche di autogestione, mantenendo inalterato l'utilizzo dei sistemi legacy, è sicuramente quella rappresentata in Figura 14.

Tale problematica è in realtà stata già riscontrata dagli sviluppatori della piattaforma CARTAGO, in quanto si è già analizzato ampiamente il caso in cui un artefatto debba fungere da interfaccia o wrapper rispettivamente a risorse o tecnologie esistenti, come già illustrato nella Figura 9.

La definizione di artefatto come entità passiva utilizzata come strumento dagli agenti pensa in prima istanza che essi non possano incapsulare un flusso di controllo interno, proprietà propria esclusivamente degli agenti; in realtà esistono strumenti comunemente utilizzati dagli esseri umani che dispongono di un ciclo interno di esecuzione, vedi il classico esempio dell'orologio.

Un orologio, considerando la Teoria dell'Attività su cui si basa il paradigma A&A, non si può certo classificare come agente, in quanto non è assolutamente proattivo e non dispone di alcun tipo di autonomia; è lecito dunque considerarlo come un artefatto a cui è stato attribuito da un agente lo svolgimento di uno o più processi interni, come ad esempio lo scorrere del tempo di un orologio.

Con CARTAGO è dunque possibile fare in modo che un Manageability Endpoint disponga di un processo interno che ciclicamente vada ad interrogare i

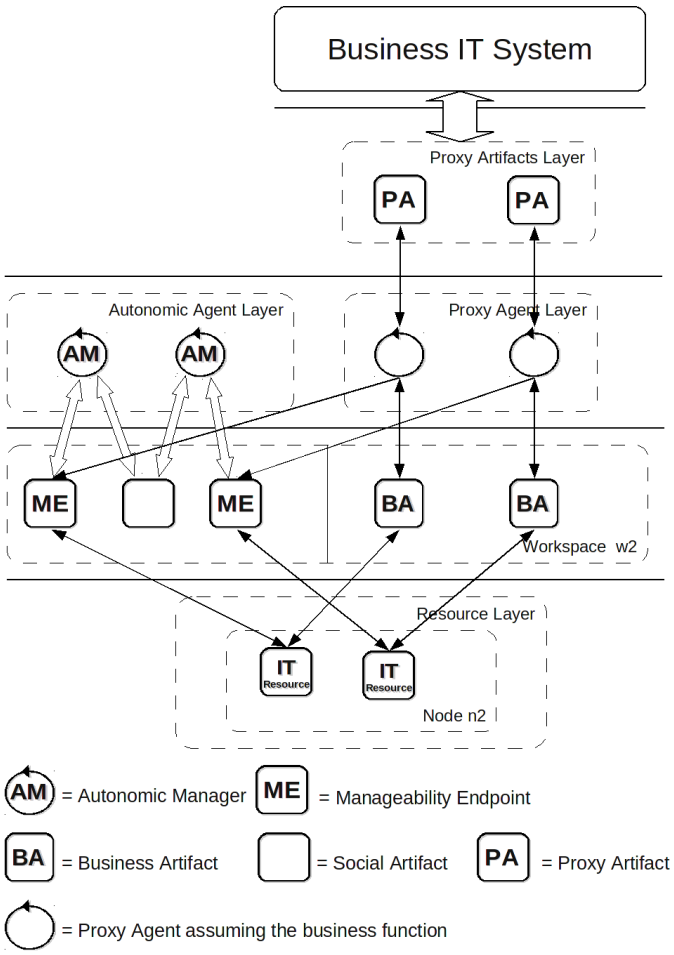

Figura 16: Integrazione fra le funzionalità autonomiche e le funzionalità di business in un sistema informatico: soluzione che introduce un livello di agenti e artefatti che funge da proxy fra il precedente sistema informatico e le risorse.

parametri di interesse della risorsa e li metta a disposizione in maniera aggiornata all'Autonomic Manager di competenza, il quale potrà, a discrezione del progettista, intervenire per modificare i tempi di aggiornamento di tali informazioni, giungendo ad un compromesso fra validità temporale delle informazioni e peso computazionale dell'aggiornamento stesso.

L'esempio di architettura generale presentata nella Figura 13, può dunque essere modificata e resa definitiva annotando la ciclicità nell'interazione fra Manageability Endpoint come illustrato in figura Figura 17.

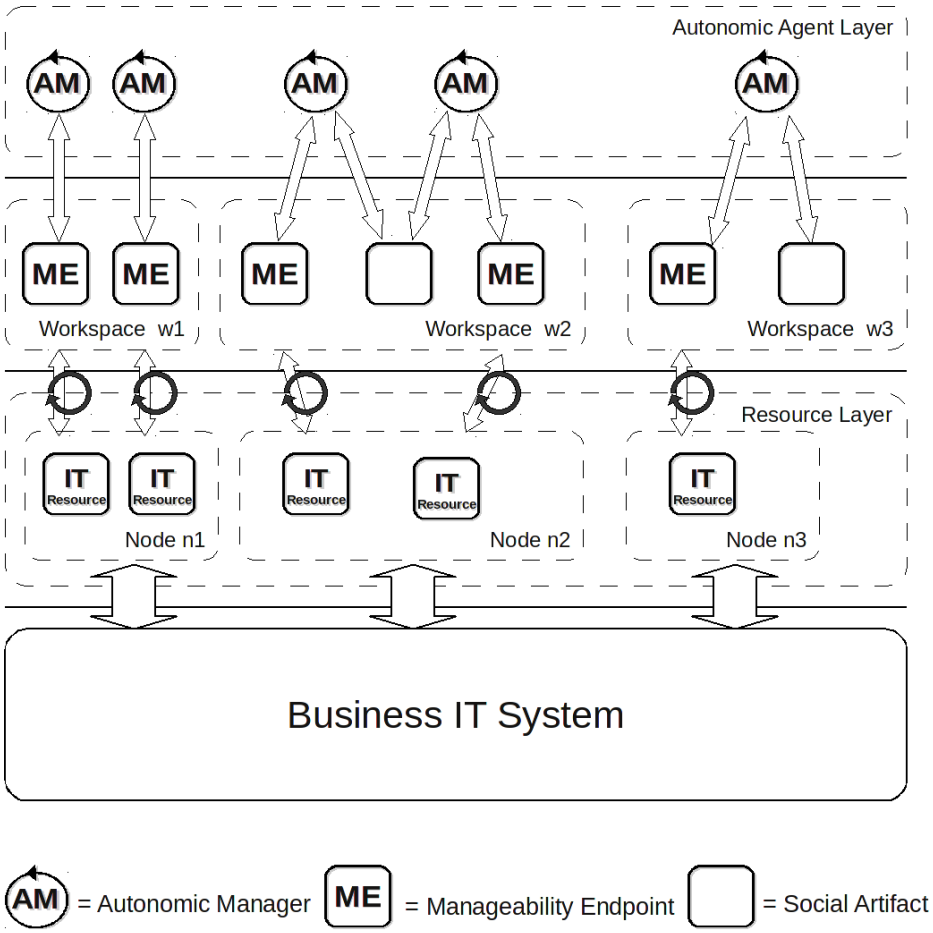

Figura 17: Rappresentazione di un sistema informatico dotato di funzionalità autonomiche realizzate con Jason e CARTAGO. E' opportuno sottolineare l'interazione ciclica svolta da un Manageability Endpoint sulla risorsa gestita per il recupero di informazioni aggiornate.

4 Sviluppi Futuri e Conclusioni

4.1 Sviluppi Futuri

Questo primo studio sull'Autonomic Computing è servito a delineare un approccio di base sulla possibile risoluzione del problema attraverso l'utilizzo di un paradigma ad agenti basato sul modello A&A.

Molti aspetti relativi all'Autonomic Computing trattati in letteratura non sono stati analizzati o sono stati analizzati solo in parte e elenchiamo qui i punti di sviluppo principali.

Analisi dell'efficacia attraverso simulazioni reali Il primo punto su cui focalizzare lo sviluppo di tale approccio riguarda la analisi dell'efficacia e, in secondo luogo, dell'efficienza dello stesso nell'ambito dell'applicazione dello stesso in casi di studio reali.

In particolare si delinea un'analisi per gradi, che consiste dapprima nell'effettuare i test del caso su sistemi fittizi di costituiti da un numero ristretto di componenti, passando ad una simulazione su scala ridotta di un vero sistema informatico, fino a giunegere all'effettiva sperimentazione su un sistema informatico reale.

Sviluppo degli Orchestrating Autonomic Manager Un elemento che potrebbe ricoprire un ruolo importante nella architettura di un sistema autonomico è quella dell'Orchestrating Autonomic Manager, il cui compito risiede nel conferire all'intera struttura autonomica funzionalità di auto gestione, piuttosto che a una singola risorsa come i classici Autonomic Manager ampiamente analizzati.

Esso è stato analizzato solo quando è stato identificato come una possibile entità cui spetta il compito di modificare il file delle politiche degli Autonomic Manager, ma non sono state analizzate le modalità con cui esso giunge a determinare tali modifiche.

Un'idea esplorabile èquella di utilizzare come fonti di dati le *Knowledege Sources* condivise fra gli Autonomic Manager e a seconda di politiche di business globali, non relative ad una singola risorsa, agire in modo da modificare il comportamento di uno o più Autonomic Manager.

Progettazione di componenti a supporto dell'infrastruttura L'architettura generale di un sistema di Autonomic Computing elaborata analizzando le diverse fonti presenti in letteratura è costituita, oltre che dai componenti principali analizzati, da elementi infrastrutturali a supporto dell'attività degli Autonomic Manager non analizzati in dettaglio in questo studio.

Registry : questo componente mira a fornire informazioni relative agli elementi autonomici presenti nel sistema, quali ad esempio gli Autonomic Manager attivi o i Manageability Endpoint gestiti. L'infrastruttura CARTAGO da un lato permette un agile recupero di artefatti (e quindi Manageability Endpoint), però tale supporto non è sufficiente: potrebbe essere utile disporre di un meccanismo per ottenere riferimenti agli Autonomic Manager che operano nel sistema e informazioni sulla loro attuale gestione delle risorse, ponendo particolare attenzione anche all'introduzione di un sistema di permesse per gestire tali circostanze.

Sentinel : il compito assegnato in letteratura a questo elemento e descritto nella Sezione 2.2.5, può essere incorporato nell'insieme di attività svolte dagli Orchestrating Autonomic Manager, in quanto l'utilizzo di un modello di programmazione ad agenti favorisce il monitoring autonomo di determinate circostanze.

Knowledege Sources : le informazioni globali di sistema sono di rilevante importanze in particolare nella fase di *Analyze* degli Autonomic Manager e possono essere memorizzate in artefatti sociali condivisi; un possibile sviluppo di tale approccio potrebbe prevedere l'utilizzo di media di coordinazione più elaborati quali i centri di tuple.

Introduzione di standard Gli standard possono aiutare ad unificare per tipo la progettazione di funzioni autonomiche relative alla gestione di risorse; in particolare lo standard WSDM, ovvero Web Service Distributed Management, può semplificare lo sviluppo di Manageability Endpoint i quali potrebbero accedere, attraverso l'uso della tecnologia dei Web Services, alle risorse gestite.

SDD, Solution Deployement Descriptor, può invece risultare utile nel caso in cui si vogliano attribuire proprietà *Self-Configuring* al sistema, utilizzando lo standard nelle azioni svolte dagli Autonomic Manager.

Realizzazione di una piattaforma infrastrutturale ad-hoc Nel caso la sperimentazione pratica di questo approccio fornisca conferma dell'efficacia di tale metodologia per sistemi di Autonomic Computing, si può delineare la possibilità di progettare un layer infrastrutturale per la progettazione dei componenti autonomici nei sistemi informatici, raccogliendo le operazioni e strutture comuni delle entità che compongono l'architettura di un sistema di Autonomic Computing definita.

In particolare si possono creare entità caratterizzate da un livello d'astrazione superiore ad agenti Jason e artefatti CARTAGO, che rappresentino la base su cui modellare Autonomic Manager e Manageability Endpoint, automatizzando

processi quali il recupero delle politiche di business per l'Autonomic Manager o l'interfacciamento con la risorsa reale per quanto concerne il Manageability Endpoint.

4.2 Conclusioni

La soluzione proposta per la realizzazione di sistemi informatici dotati di funzionalità autonomiche è ancora ad un livello primordiale di sviluppo, come è possibile notare dai molteplici possibili ambiti di sviluppi futuri elencati nella Sezione 4.1.

Nonostante molti aspetti di piccola-media importanza siano stati affrontati solamente in maniera marginale, da questo primo studio sembra che la realizzazione di sistemi autonomici possa essere un ottimo campo di applicazione di modelli di programmazione ad agenti, in particolare delle tecnologie JaCa (Jason con CARTAGO), la quale inoltre fornisce una facile integrazione con i sistemi legacy grazie alle fondamenta radicate in una tecnologia così ampiamente diffusa quale Java.

L'efficacia di questa metodologia deve essere in ogni caso verificata sperimentalmente su sistemi reali, ma per poterla realizzare in maniera utile è necessaria un'ampia conoscenza del dominio applicativo; a tale proposito questo ambito potrebbe offrire ampie opportunità per collaborazioni fra università e aziende, che in questi ultimi tempi si trovano ad affrontare grossi problemi di scalabilità dei propri sistemi.

In conclusione non possiamo affermare con assoluta certezza la validità di tale approccio per sistemi informatici autonomici, ma sicuramente vale la pena approfondire tale studio perchè mostra le potenzialità per affrontare in maniera efficiente la "Crisi della complessità" che molti sistemi informatici stanno affrontando o sono prossimi dall'affrontare.

Riferimenti bibliografici

[1] BORDINI, R. H., HUBNER, J. F., AND WOOLDRIDGE, M. *Programming Multi-Agent Systems in AgentSpeak using Jason*. Wiley, 2007.

[2] HARIRI, S., KHARGHARIA, B., CHEN, H., YANG, J., ZHANG, Y., PARASHAR, M., AND LIU, H. The autonomic computing paradigm. *Cluster Computing 9*, 1 (2006), 5–17.

[3] IBM. An architectural blueprint for autonomic computing. *IBM Research Grop on Autonomic Computing* (2006), 1–36.

[4] OMICINI, A., RICCI, A., AND VIROLI, M. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems 17*, 3 (Dec. 2008), 432–456. Special Issue on Foundations, Advanced Topics and Industrial Perspectives of Multi-Agent Systems.

[5] RAO, A. S. AgentSpeak(L): BDI agents speak out in a logical computable language. In *Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World* (Eindhoven, The Netherlands, 1996), R. van Hoe, Ed.

[6] RICCI, A., VIROLI, M., AND OMICINI, A. The a&aprogramming model and technology for developing agent environments in mas. In *PROMAS* (2007), pp. 89–106.

[7] WHITE, S. R., HANSON, J. E., WHALLEY, I., CHESS, D. M., AND KEPHART, J. O. An architectural approach to autonomic computing. *Autonomic Computing, International Conference on 0* (2004), 2–9.