Self-Organization & MAS: Autonomic Computing

Approfondimento per il corso di:
Sistemi Multiagente LS

Massimo Bartolini

Indice

1 Introduzione **5**
1.1 Auto-organizzazione 5
1.2 Caratteristiche di un sistema auto-organizzante 6

2 Autonomic Computing **7**
2.1 Che cos' è l'Autonomic Computing 7
2.2 Principi fondamentali 8
2.2.1 Auto-configurazione 9
2.2.2 Auto-riparazione 9
2.2.3 Auto-ottimizzazione 10
2.2.4 Auto-protezione 10
2.2.5 Caratteristiche minori 10
2.2.6 Livelli di automazione 11
2.3 Architettura 12
2.3.1 Managed resource 13
2.3.2 Autonomic Manager 13
2.3.3 Touchpoint 14
2.3.4 Interazione tra autonomic element 15

3 Autonomic Computing e Agenti **16**
3.1 Integrazione tra Autonomic Computing e MAS 16
3.2 Architetture agent-based per Autonomic Computing 17
3.2.1 Primo modello 17
3.2.2 Secondo modello 19
3.2.3 Terzo modello 20

Bibliografia **23**

Capitolo 1

Introduzione

In questo capitolo introduttivo viene semplicemente descritto il concetto di auto-organizzazione e vengono elencate le caratteristiche principali di un sistema auto-organizzante.

1.1 Auto-organizzazione

L'auto-organizzazione costituisce un approccio efficiente per affrontare la complessità dei sistemi moderni, come testimoniato dal crescente interesse della comunità scientifica. Un approccio auto-organizzante permette lo sviluppo di sistemi che esibiscono dinamiche complesse e che si adattano alle perturbazioni ambientali senza richiedere una conoscenza completa delle condizioni circostanti. Un sistema sviluppato seguendo i principi dell'auto-organizzazione produce pattern e dinamiche globali attraverso l'interazione locale dei suoi componenti.

Lo sviluppo di sistemi auto-organizzanti (SOS) è guidato da principi diversi rispetto all'ingegneria tradizionale. Tipicamente, gli ingegneri progettano sistemi come composizione di elementi più semplici, sia nel caso di astrazioni software che di dispositivi fisici, dove le regole di composizione dipendono dal paradigma di riferimento, ma che comunque producono risultati predicibili. Al contrario, i sistemi auto-organizzanti mostrano dinamiche non-lineari, le quali possono essere difficilmente catturate da modelli deterministici, e che sebbene robusti nei confronti di perturbazioni esterne, sono piuttosto sensibili ai cambiamenti di parametri interni.

Inoltre, i principi dell'auto-organizzazione sono attualmente al centro di progetti di ricerca che potrebbero avere rilevanza industriale in un futuro prossimo come, ad esempio, l'Autonomic Computing.

1.2 Caratteristiche di un sistema auto-organizzante

Di regola i sistemi auto-organizzati possiedono quattro caratteristiche principali:

- *Autonomia*: i sistemi auto-organizzanti sono autonomi se le relazioni e interazioni che definiscono il sistema come un tutt'uno sono determinate solamente dal sistema stesso.

- *Complessità*: sono complessi i sistemi le cui parti si intrecciano l'una all'altra tramite relazioni mutuali in permanente cambiamento. Le parti possono cambiare similmente ogni volta. La complessità rende più difficile la descrizione e la previsione del comportamento dei sistemi nella loro interezza.

- *Dinamicità*: l'auto-organizzazione deve essere intesa come un processo e non come uno stato finale.

- *Adattività*: le perturbazioni possono avvenire sia all'interno del sistema, sia nell'ambiente circostante; un sistema auto-organizzante deve essere in grado di rispondere adeguatamente a tali cambiamenti.

Capitolo 2

Autonomic Computing

In questo capitolo si fornisce una descrizione degli apetti principali dell'Autonomic Computing, tra cui il concetto di self-management e l'architettura.

2.1 Che cos' è l'Autonomic Computing

La complessità dei sistemi IT continua inesorabilmente a crescere. Di conseguenza essi sono sempre più proni a errori e la loro gestione è più che mai difficile. La ricerca di un paradigma che si occupi specificamente di governare questa crescente complessità è diventata ormai una delle più grandi sfide dell'industria IT. L'Autonomic Computing si pone come obiettivo quello di risolvere questa grande sfida.

L'aggettivo *autonomo* caratterizza entità che eseguono azioni involontariamente, senza un controllo cosciente. In psicologia, esso è legato alle attività controllate del sistema nervoso centrale. Con questa metafora, si vuole presentare la caratteristica principale dell'Autonomic Computing, ovvero l'abilità di gestire l'infrastruttura ICT attraverso hardware e software che autonomamente (senza controllo umano) e dinamicamente risponde ai requisiti del business. Questo significa avere un sistema IT che è in grado di auto-ripararsi, auto-configurarsi, auto-ottimizzarsi, auto-proteggersi e comportarsi in accordo a determinati livelli di servizio e determinate policy impostate dall'amministratore. Proprio come il sistema nervoso centrale risponde alle richieste del corpo, l'Autonomic Computing risponde alle esigenze del business. L'introduzione dell'Autonomic Computing in questo determinato contesto storico è dettato principalmente da due tipologie di aspetti:

- Gli aspetti legati al business: oggigiorno il mercato è molto veloce, c'è una continua richiesta di sistemi reattivi e spesso il continuo cambiamento dell'infrastruttura IT è richiesto dal modello di business adottato dall'azienda.

- Gli aspetti tecnici: l'elevata complessità di gestione resta uno dei principali attriti al progresso dell'industria IT. L'elevata interconnessione ed eterogeneità rende difficile per i tecnici anticipare e modellare le differenti interazioni tra i vari componenti. Diventerà presto troppo difficile e costoso installare, configurare, ottimizzare, manutenere e integrare sistemi IT.

La necessità di Autonomic Computing arriva quindi sia dal contesto di business attuale sia dalle condizioni dell'infrastruttura IT. Altro aspetto fondamentale è che le tecnologie odierne sono abbastanza mature per abilitare un tale paradigma.

2.2 Principi fondamentali

Un concetto centrale alla base del paradigma dell'Autonomic Computing è il *self-management*, ovvero la capacità di alcuni sistemi di funzionare per 24 ore al giorno 7 giorni su 7 senza il controllo umano. Il self-management permette ai sistemi autonomici di modificare il loro comportamento a fronte di cambiamenti di componenti, del carico di lavoro, della domanda e delle modificazioni dell'ambiente di esecuzione.
È possibile scomporre il self-management in quattro aspetti fondamentali:

- Auto-configurazione (self-confiuring).
- Auto-riparazione (self-healing).
- Auto-ottimizzazione (self-optimizing).
- Auto-protezione (auto-protecting).

Vi sono inoltre caratteristiche minori dei sistemi autonomici quali: consapevolezza di sè (self-awareness), apertura (openness), consapevolezza del contesto (context-awareness), comportamento anticipatorio (anticipatory behavior).

Nel seguito ognuno di questi aspetti verrà analizzato nel dettaglio; si introdurranno inoltre i livelli di automazione attraverso i quali un normale sistema IT può diventare un sistema autonomico.

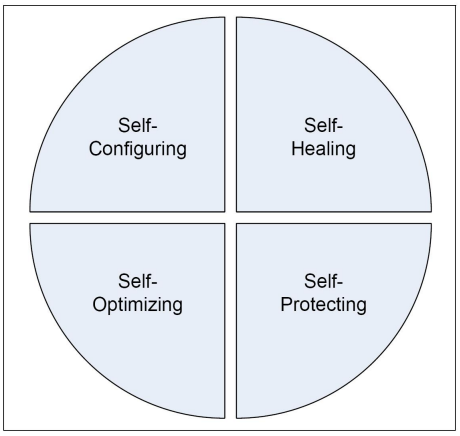

Figura 2.1: I quattro aspetti principali dell'Autonomic Computing

2.2.1 Auto-configurazione

L'auto-configurazione permette di adattare il sistema ai veloci cambiamenti dell'ambiente esterno, aumentando così la sua velocità di risposta.
L'eterogeneità, caratteristica degli attuali sistemi IT, rende difficili e proni a errori i processi di installazione, configurazione e integrazione.

La visione che ci offre l'Autonomic Computing è quella di componenti e sistemi che si auto-configurano seguendo delle politiche di alto livello definite dall'amministratore. L'idea di politica racchiude il motto dell'Autonomic Computing, ovvero l'amministratore deve essere conscio del cosa fare, non del come farlo: il 'come' è determinato dal sistema.

Ad esempio, l'introduzione di un nuovo componente non deve essere a carico del sistemista, piuttosto deve essere il componente stesso a determinare la configurazione ottimale e a effettuare determinate azioni per facilitare l'integrazione con gli altri componenti.

2.2.2 Auto-riparazione

L'auto-riparazione permette di abilitare la resilienza del business. I sistemi autonomici rilevano, diagnosticano e agiscono per prevenire la loro distruzione.

Negli attuali sistemi IT, sempre più spesso interconnessi, è difficile determinare le condizioni e le cause degli errori. Ad esempio, in dipartimenti di grandi corporazioni esistono team che hanno il solo scopo di identificare,

tracciare e determinare le cause principali di errori. Alcuni problemi possono richiedere settimane per essere diagnosticati e risolti, altre volte questi stessi problemi scompaiono rendendo impossibile una qualsiasi diagnosi.

I sistemi autonomici determinano, diagnosticano e talvolta riparano problemi derivanti da insuccessi software e hardware. Componenti dedicati a questi scopi analizzano di continuo i file di log ed effettuano determinate azioni per risolvere i problemi.

2.2.3 Auto-ottimizzazione

L'auto-ottimizzazione permette di ottenere un'efficienza operativa mediante il tuning delle risorse e del carico di lavoro per massimizzare l'utilizzo dell'infrastruttura IT.

Nella visione dell'Autonomic Computing i componenti e il sistema sono proattivi, cioè prendono iniziativa e senza il controllo umano decidono cosa fare a run-time, e ricercano continuamente opportunità per migliorare le loro performance.

2.2.4 Auto-protezione

L'auto-protezione permette di mettere al sicuro informazioni e risorse anticipando, rilevando, identificando e proteggendo attivamente il sistema contro attacchi maliziosi e non. I moderni sistemi IT gestiscono solo manualmente la detection e il recovery da attacchi e insuccessi hardware e software. Ad esempio, nonostante l'esistenza di firewall e tool per il rilevamento delle intrusioni, è necessaria la presenza di un umano per decidere di proteggere il sistema da attacchi maliziosi o da involontarie failure a cascata.

I sistemi autonomici sono in grado di auto-proteggersi in due sensi. Un primo tipo di protezione, quella sistemica, difende il sistema da problemi su larga scala derivanti da attacchi maliziosi o failure a cascata che rimangono a lungo non corrette. Un secondo tipo di protezione anticipa i problemi sulla base di report iniziali derivanti dalle informazioni acquisite mediante sensori in modo da eliminare o mitigare future cause di errore.

2.2.5 Caratteristiche minori

Si presentano nel seguito alcune caratteristiche minori ([4] e [5]) che contraddistinguono gli autonomic system.

La consapevolezza di sè caratterizza sistemi che sono in grado di informarsi sul loro stato e sul loro comportamento. Nel caso dei sistemi autonomici

questa proprietà rende possibile attuare politiche di self-management in caso di condizioni anomale.

La consapevolezza del contesto indica che ciascun sistema autonomico deve essere consapevole dell'ambiente di esecuzione e del contesto in cui esso viene eseguito. Questo significa che un tale sistema deve essere pronto a reagire in caso di cambiamenti dell'ambiente.

L'apertura caratterizza i sistemi autonomici perchè essi devono tipicamente operare in ambienti eterogenei. Inoltre, il numero e la tipologia di componenti di tali ambienti non è stabilita a priori e di conseguenza ogni sistema autonomico deve essere pronto a far fronte a un numero illimitato di situazioni non conosciute a design-time.

Il comportamento anticipatorio caratterizza i sistemi autonomici in quanto essi devono prevedere e anticipare le condizioni critiche. L'esempio classico di questo tipo di comportamento è un sistema che effettui previsioni delle risorse necessarie in futuro. Questo permette a un sistema autonomico, congiuntamente all'auto-ottimizzazione, di ottimizzare l'utilizzo dell'infrastruttura IT. Questa caratteristica, inoltre, aggiunge pro-attività al self-management.

2.2.6 Livelli di automazione

IBM definisce vari livelli di automazione di un sistema IT. Partendo dal livello più basso, in cui un team di persone si occupa della gestione delle risorse IT, si arriva fino a un ultimo livello in cui poche persone si occupano unicamente della definizione degli obiettivi globali che il sistema autonomico stesso soddisferà.
In particolare esistono cinque livelli di automazione [2]:

1. *Basic*, la capacità di gestire l'infrastruttura e l'ambiente risiede principalmente in esseri umani a cui viene richiesto consulto anche per le procedure ordinarie.
2. *Managed*, esistono tool di scripting e logging che automatizzano alcune routine di esecuzione e di reporting. Specialisti analizzano le informazioni acquisite mediante i tool per fini di pianificazione e decisione.
3. *Predictive*, all'interno del sistema esistono meccanismi per i quali da alcune condizioni scaturiscono avvisi preventivi. Una base di conoscenza raccomanda le azioni più appropriate a seconda delle condizioni. La risoluzione del problema è quindi immagazzinata all'interno di uno storage di situazioni comuni che ha lo scopo di raccogliere l'esperienza maturata.

4. *Adaptive*, il sistema è adattivo, riesce a eseguire azioni opportune in base alle diverse situazioni presentatesi; sono presenti, inoltre, capacità di previsione.

5. *Autonomic*, il sistema è guidato da politiche di alto livello che specificano gli obiettivi globali impostati dagli amministratori. Le azioni specifiche per soddisfare queste richieste vengono determinate dal sistema senza l'intervento umano.

Ognuno di questi livelli descrive una fase attraverso la quale una comune infrastruttura IT può diventare un sistema autonomico. Pertanto, si può parlare di evoluzione dell'infrastruttura più che di una vera e propria rivoluzione.

2.3 Architettura

In questo paragrafo si analizzeranno nel dettaglio le entità principali di un sistema autonomico: gli autonomic element (Figura 2.2) che hanno come scopo quello di fare da wrapper alle ordinarie risorse del sistema IT. Ogni autonomic element ha tre costituenti fondamentali: una *managed resource*, un *touchpoint* e un *autonomic manager*.

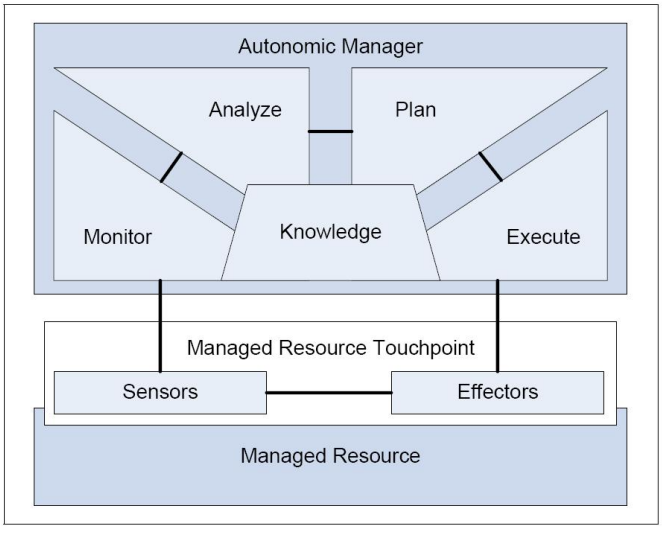

Figura 2.2: Loop di controllo dell' Autonomic Computing

2.3.1 Managed resource

Le managed resource costituiscono i componenti di un'infrastruttura IT, come ad esempio server, database, file. È interessante notare che le managed resource possono essere anche degli elementi di tipo aggregato, come cluster di server, un'intera applicazione o anche una business unit. Il concetto di managed resource, quindi, è altamente scalabile.

2.3.2 Autonomic Manager

Un autonomic manager è un componente che si occupa della gestione delle managed resource. In particolare, ogni autonomic manager ha lo scopo di collezionare, filtrare e controllare i dati provenienti dalle managed resource tramite i sensori. Inoltre, al fine di far fronte a situazioni impreviste, l'autonomic manager può modellare, usando algoritmi di macchine learning, il componente amministrato acquisendo conoscenza su di esso. Questa conoscenza rende possibile predire situazioni critiche future.
All'interno di ogni autonomic manager è presente un loop di controllo, il cosiddetto MAPE. L'acronimo MAPE deriva dalle lettere iniziali di ognuna delle quattro fasi che si succedono in sequenza: *Monitor*, *Analyze*, *Plan*, *Execute*. La Figura 2.2 mostra queste fasi ed evidenzia una base di conoscenza o knowledge base che è condivisa tra di esse durante l'intero ciclo.

Nella fase di monitor l'autonomic manager riceve gli eventi individuati dai sensori che sono connessi alla managed resource. La knowledge base viene utilizzata dall'autonomic manager per filtrare gli eventi critici.

Dopo aver appurato che l'evento ricevuto è di interesse, nella fase di analyze, l'autonomic manager utilizza la knowledge base per individuare quale sia l'obiettivo da raggiungere.
Mediante Policy Validations e Policy Resolution si ricava la politica da rispettare. Un Analysis Engine, infine, ricevendo in input la politica e l'evento determina quale sia il goal del componente. Per questo scopo sono tipicamente impiegati dei rule engine.

Una volta che l'evento e l'obiettivo del componente sono noti, la fase di plan determina quali azioni svolgere per raggiungere l'obiettivo.
In questa fase si utilizza il cosiddetto Symptom Database (anch'esso contenuto nella knowledge base) che contiene informazioni circa i sintomi (eventi o serie di eventi) e i rimedi (azioni da effettuare). Il Plan Generators si occupa della generazione dei piani e successivamente un Policy Interpreter e un Policy Transforms controllano che il piano sia in linea con le politiche del componente.

Formulato il piano di azione, infine, segue una fase di execute dove tramite gli attuatori si esegue concretamente l'azione pianificata.
In questa fase un Workflow Engine controlla che le azioni effettuate producano concretamente i risultati per cui erano state pianificate. Nel caso di sistemi complessi, uno Scheduler Engine effettua il monitoring attivo, tenendo in considerazione anche i tempi, e un Distribution Engine permette di delegare determinate azioni ad altri componenti.

Per completezza, si riportano i dettagli che caratterizzano la knowledge base. Il suo ruolo è quello di rappresentare i dati condivisi tra le quattro fasi citate in precedenza. Alcuni dei dati contenuti nella knowledge base sono ad esempio: i log degli eventi critici ricevuti dalla managed resource, le informazioni sulla topologia del sistema e la policy.

2.3.3 Touchpoint

In un sistema autonomico i touchpoint rappresentano le interfacce di gestione per le managed resource, ovvero per i componenti che devono essere amministrati. Il touchpoint rappresenta un modo per fornire un punto di accesso singolo, standardizzato, indipendente dal tipo di risorsa gestita. Ad esempio, un touchpoint può essere utilizzato per gestire router, application server, middleware, etc.
All'interno un touchpoint è costituito da una manageability interface, che include una sensor interface e una effector interface, e permette di interagire con la risorsa gestita (managed resource).
L'interfaccia mette a disposizione gli usuali meccanismi di gestione (esecuzione di comandi, logging, supporto di eventi, etc) ed espone informazioni riguardanti la risorsa controllata (identificazione, metadati associati a essa, metriche, configurazione e dettagli sullo stato). È interessante notare che gli attuatori e i sensori permettono un'ampia gamma di tipologie di comunicazione tra autonomic manager e managed resource. È possibile utilizzare i sensori per accedere allo stato della risorsa controllata, con due meccanismi: *Request-Response* (richiesta da parte dell'autonomic manager e risposta diretta da parte della managed resource) oppure *Send notification* (all'occorrenza la managed resource invia degli eventi significativi all'autonomic manager). Gli attuatori, invece, possono essere utilizzati bidirezionalmente: un meccanismo di *Perform-Operation* permette di eseguire operazioni per cambiare lo stato della risorsa controllata, un meccanismo di *Solicit-Response* permette, invece, a una risorsa controllata di comunicare con il suo manager.

2.3.4 Interazione tra autonomic element

In alcuni articoli sull'autonomic computing [3] si adotta una visione in cui ogni autonomic element è un agente che fornisce e consuma servizi, da questa interazione emerge appunto il self-management. Implicitamente si assume quindi una visione peer-to-peer, non gerarchica, dove gli agenti sono: autonomi e goal-oriented, perchè eseguono le richieste degli altri agenti solo se queste sono consistenti con i propri goal; pro-attivi perchè prendono iniziativa e senza il controllo umano decidono cosa fare a run-time.

Altri articoli [2], tuttavia, si focalizzano invece su una versione gerarchica in cui ogni autonomic manager può gestire come managed resource un altro autonomic element. E' bene dunque fare alcune precisazioni:

- La versione gerarchica non ha il difetto di avere un single point of failure come si potrebbe pensare. Infatti, un autonomic manager che gestisce altri autonomic element, in virtù della sua funzione, potrebbe essere replicato garantendo maggiore affidabilità al sistema.

- La versione gerarchica e quella peer-to-peer non sono in conflitto. Nella pratica si utilizza la versione gerarchica, la versione ad agenti viene utilizzata più come riferimento concettuale che pragmatico, tuttavia le due versioni non sono in conflitto. Si potrebbe, ad esempio, avere un autonomic system che sfrutti le proprietà dei sistemi peer-to-peer per eleggere un autonomic manager leader che prenda gerarchicamente il controllo degli altri autonomic manager.

Capitolo 3

Autonomic Computing e Agenti

In questo capitolo vengono presentati diversi approcci basati sul paradigma ad agenti applicabili all'Autonomic Computing.

3.1 Integrazione tra Autonomic Computing e MAS

Comunemente un autonomic system viene definito "un sistema capace di auto-gestirsi in accordo a determinate policy impostate dall'amministratore"[3]. Questa definizione assomiglia molto alla classica definizione di agente software, cioè "un sistema informatico situato in un certo ambiente e capace di eseguire azioni autonomamente, all'interno di questo ambiente, al fine di raggiungere i propri obiettivi"[11].
Si possono quindi facilmente identificare i singoli agenti con i singoli autonomic element: entità autonome, adattive, che possono agire sia reattivamente sia proattivamente, percependo l'ambiente in cui si trovano e interagendo tra di loro per soddisfare scopi individuali. Quindi sia nel campo dell'Autonomic Computing, sia nel paradigma ad agenti, queste singole entità autonome gestiscono il proprio comportamento, le loro interazioni con l'ambiente e con le altre entità autonome per realizzare goal specifici e globali.
La maggior parte degli autonomic system coinvolgono molteplici sistemi e la maggior parte dei sistemi ad agenti coinvolgono molteplici agenti. Di conseguenza è possibile identificare gli autonomic system con sistemi multiagente.

Alcuni dei primi articoli su Autonomic Computing [3,6] descrivono delle possibili architetture che sono fortemente agent-oriented. Tali sistemi vengono presentati come una collezione di entità interagenti chiamate *autonomic element*. Ogni autonomic element è analogo a un agente, nel senso che gestisce il proprio comportamento agendo sulle informazioni ottenute dai propri sensori in accordo con le policy e gli accordi stabiliti con gli altri

autonomic element. Il comportamento autonomo dell'intero sistema deriva dalle interazioni tra questi autonomic element così come il comportamento di un sistema multiagente deriva dalle interazioni tra i singoli agenti. E' dunque possibile utilizzare diverse tecnologie e metodi propri del paradigma multiagente per cercare di descrivere l'architettura di un autonomic system.

3.2 Architetture agent-based per Autonomic Computing

In passato il problema della gestione dei sistemi informatici veniva affrontato soprattutto utilizzando approcci centralizzati e gerarchici.
Oggigiorno, invece, i sistemi ICT sono caratterizzati da fattori come l'interconnessione, l'eterogeneità e l'alto dinamismo. Di conseguenza, la loro gestione diventa particolarmente complessa e costosa e l'approccio centralizzato appare non adatto. Da qui l'idea di un approccio decentralizzato basato su sistemi multiagente si presta meglio a definire un'architettura per l'autonomic computing.

Diversi approcci e modelli architetturali emergono dalle ricerche effettute in tale ambito. Di seguito ne verranno presentati alcuni.

3.2.1 Primo modello

In un suo articolo [7] G. Pour illustra un progetto di un'architettura autonomica.
Questa architettura consiste di tre livelli principali (Figura 3.1), ciascuno dei quali è costituito da differenti sottosistemi (Figura 3.2).

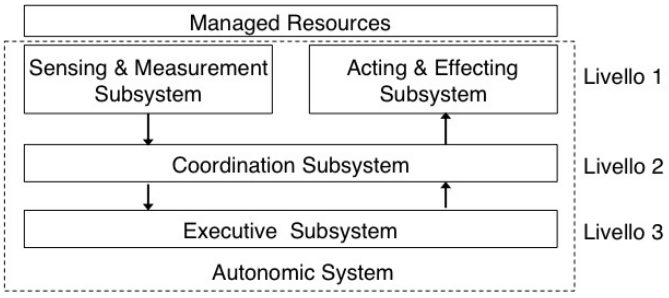

Figura 3.1: Progetto di un'architettura autonomica di alto-livello

- Il livello 1 comprende i sottosistemi *Sensing & Measurement*, adibiti alla raccolta e alla misura dei dati di interesse, e *Acting & Effecting*, in grado di modificare l'ambiente in accordo con i comandi ricevuti dall'autonomic manager.

- Il livello 2 comprende i sottosistemi che riguardano la coordinazione, tra cui *Information Management*, *Knowledge Management*, *Analysis*, *Monitoring* e *Plan Execution*.

- Il livello 3 è composto invece dai sottosistemi esecutivi, quali l'identificazione degli eventi, il planning, la gestione dei rischi , la gestione della sicurezza.

Ogni sottosistema gestisce il proprio stato interno, il proprio comportamento e le interazioni con l'ambiente utilizzando segnali e messaggi.
Ogni sottosistema si compone di un insieme di agenti, con differenti funzionalità e goal a seconda del livello in cui si trovano. Ad esempio gli agenti del Livello 1 agiscono localmente, sono reattivi, autonomi, capaci di svolgere prontamente azioni preprogrammate che richiedono risposta immediata. Gli agenti del livello 2 invece dispongono di una base di conoscenza euristica per svolgere differenti attività di coordinazione. Nel livello 3 gli agenti hanno goal e piani espliciti per raggiungere i loro obiettivi.

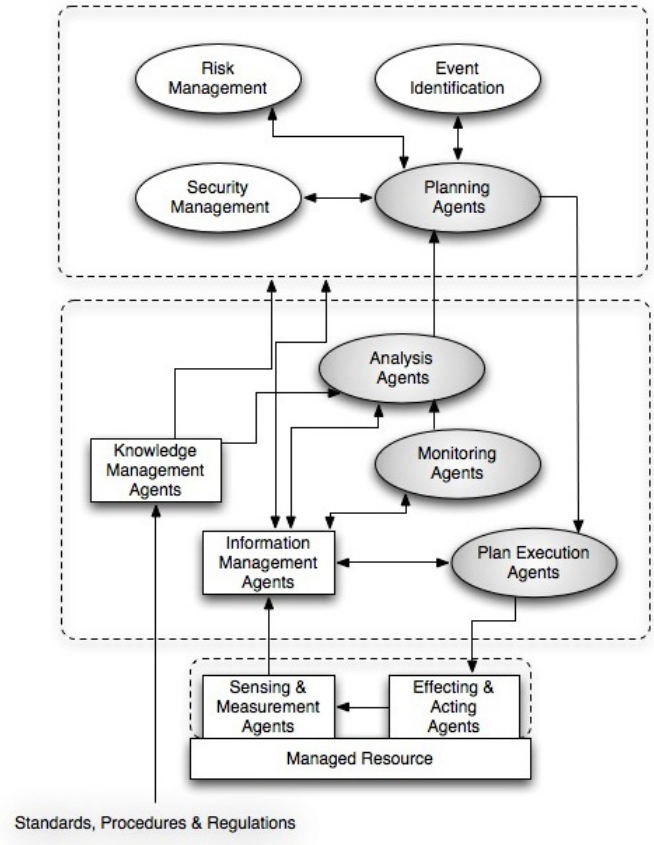

Figura 3.2: Sotto-sistemi principali dell'architettura autonomica

3.2.2 Secondo modello

IBM ha sviluppato un'architettura software decentralizzata chiamata Unity [6] che si propone di definire un sistema in grado di auto-gestirsi attraverso l'interazione di una popolazione di agenti autonomi denominati autonomic element.
Gli obiettivi specifici di Unity sono: permettere a un sistema di auto-configurarsi durante l'inizializzazione a runtime; sviluppare dei design pattern per l'auto-riparazione all'interno del sistema; auto-ottimizzare in tempo reale l'uso di risorse computazionali distribuite in accordo con gli obiettivi da raggiungere.

Struttura

La struttura essenziale di Unity segue quanto descritto da Kephart e Chess [3]. Ogni componente di Unity è un autonomic element. Gli autonomic element si distinguono in risorse(database, server, storage system), elementi di più alto livello con una qualche autorità di gestione ed elementi di supporto ad altri elementi nello svolgimento delle loto attività(policy repostitory, sentinel, brookers, registry).
Ogni autonomic element è responsabile del suo comportamento autonomo. Innanzitutto deve sapersi auto-gestire, cioè deve essere capace di auto-configurarsi, di auto-ripararsi in caso di malfunzionamenti interni, di auto-ottimizzare il proprio comportamento e di auto-proteggersi da eventuali attacchi esterni.
In secondo luogo un autonomic element deve essere capace di instaurare e mantenere delle relazioni con altri autonomic element, quelli a cui fornisce dei servizi e quelli che forniscono servizi a lui. Questo richiede una descrizione accurata dei servizi offerti e delle modalità di accesso da parte degli altri autonomic element. Deve inoltre essere in grado di negoziare con gli altri elementi per stabilire degli accordi e di rifiutare richieste di servizi che potrebbero violare accordi presi precedentemente.

Unity può supportare diversi application environment logicamente separati, ognuno dei quali fornisce un servizio distinto. Ogni application environment è rappresentato da un elemento chiamato *application manager* che è responsabile della gestione dell'ambiente, della ricerca delle risorse necessarie a raggiungere determinati obiettivi e della comunicazione con gli altri elementi.

Gli elementi di tipo *resource arbiter* sono responsabili dell'allocazione ottima delle risorse.

Gli elementi di tipo *registry* consentono ai diversi elementi di localizzar-

ne altri con cui essi hanno bisogno di interagire.

Gli elementi di tipo *policy repository* supportano delle interfacce che permettono agli amministratori di inserire delle policy di alto livello che guidano le operazioni del sistema.

Gli elementi di tipo *sentinel* permettono di monitorare il funzionamento di un elemento.

"Goal-driven Self-assembly"

All'interno di Unity è stata sperimentata una tecnica chiamata "Goal-driven Self-assembly".
Idealmente ogni autonomic element, quando inizia a eseguire per la prima volta, conosce solo una descrizione di alto livello di ciò che deve essere svolto. Quando ogni elemento viene inizializzato, contatta il registry per localizzare gli elementi esistenti che possono fornire i servizi di cui ha bisogno. A questo punto contatta gli elementi appena localizzati, e instaura delle relazioni con essi per ricevere i servizi necessari. Una volta che l'elemento ha ottenuto tutte le risorse necessarie si registra a sua volta nel registry, in modo da essere identificato da altri elementi.

3.2.3 Terzo modello

In questo ultimo modello proposto [9], gli autonomic manager, che fanno parte degli autonomic element (Figura 3.3), vengono rappresentati come degli agenti, che gestiscono servizi e altri agenti di più basso livello.

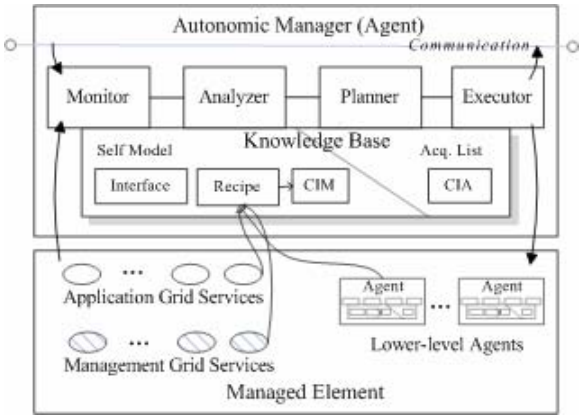

Figura 3.3: Struttura di un autonomic element

Un autonomic system, composto da autonomic element basati su agenti, può essere visto come un sistema multi-agente. Ogni agente divulga agli

altri agenti le proprie *capability* e i propri *interest* e, a sua volta, registra queste stesse informazioni degli altri. La *capability* di un agente è ciò che l'agente è capace di fare, mentre l'*interest* è una qualche informazione che l'agente considera importante e necessaria per iniziare ed eseguire un'azione.

L'architettura di questo sistema auto-organizzato è mostrata in Figura 3.4.

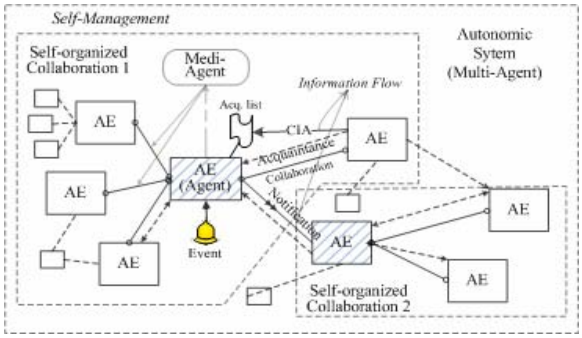

Figura 3.4: Architettura del sistema

Il sistema può essere definito da tre componenti:

- un insieme di autonomic element
- un insieme di relazioni tra autonomic element
- un meccanismo che permetta al sistema di auto-gestirsi

Autonomic element

Ogni autonomic element è composto da:

- AM (*Autonomic Manager*): un agente
- ME (*Managed Elements*): elementi come applicazioni, servizi o agenti di più basso livello
- IR (*Internal Relationship*): relazioni interne tra AM e ME

Le relazioni IR sono principalmente descritte nel *Self-Model* (SM), che definisce come l'agente manager integra, diffonde, invoca, monitora e controlla i managed element. Il self-model si compone di tre parti: *Recipe*, *Interface* e *CIM (Capability and Interest Model)*

- *Recipe* definisce le linee guida per realizzare attività complicate (servizi composti).

- *Interface* descrive come utilizzare e controllare i managed element. Senza tale descrizione gli agenti non potrebbero invocare e gestire le risorse di basso livello.

- *CIM (Capability and Interest Model).* In un sistema complesso, la realizzazione di diverse attività richiede una certa collaborazione e cooperazione tra gli elementi. A tal fine, gli autonomic element hanno bisogno di manifestare e notificare ciò che sono in grado di fare e ciò cui sono interessati. Ciascun agente, dunque, si crea un *Capability and Interest Advertisement (CIA)* che verrà comunicato agli altri agenti.

Relazioni tra autonomic element

Vengono presentate tre tipologie di relazioni tra gli autonomic element:

- *Conoscenza.* Relazione che si verifica quando un autonomic element riceve i CIA resi noti dagli altri; tali informazioni vengono memorizzate unitamente agli autonomic element che le hanno diffuse. Ogni agente dispone, quindi, di una base di conoscenza in cui sono elencati tutti gli elementi di cui è a conoscenza. In altre parole, dati due autonomic element A e B, A fa parte della base di conoscenza di B, se B conosce le funzionalità e gli interessi di A. B, a questo punto, può richiedere i servizi che A è in grado di offrire o comunicare ad A le informazioni che necessita.

- *Cooperazione.* Relazione stabilita per realizzare obiettivi comuni. Ogni elemento realizza un sub-task parziale e lavora insieme agli altri membri per soddisfare il goal collettivo.

- *Notifica.* Relazione che si verifica quando un elemento possiede informazioni richieste da un altro elemento; in questo caso viene inviato un messaggio contenente tali informazioni all'elemento interessato.

Bibliografia

[1] M. Parashar, S. Hariri. *Autonomic Computing - Concepts, Infrastructure, and Applications*

[2] IBM. *An architectural blueprint for autonomic computing.* IBM Autonomic Computing White paper, 2006.

[3] J. Kephart, D. Chess. *The Vision of Autonomic Computing.* IEEE Computer 36(1): 41-50, 2003.

[4] P. Horn. *Autonomic Computing: IBM's Perspective on the State of Information Technology.* Technical Report, IBM Corporation, October 15, 2001.

[5] M. Salehie, L. Tahvildari. *Autonomic computing: emerging trends and open problems.* Proceedings of the 2005 workshop on Design and evolution of autonomic application software, pages 1 - 7, 2005.

[6] S. R. White, J. R. Hanson, I. Whalley, D. M. Chess, J. O. Kephart. *An Architectural Approach to Autonomic Computing.* Proceedings of the 1st International Conference on Autonomic Computing (ICAC'04), IEEE Computer Society Press, pp 2-9, May 2004.

[7] G. Tesauro, D. M. Chess, W. E. Walsh, R. Das, I. Whalley, J. O. Kephart, and S. R. White. *A multiagent systems approach to autonomic computing.* Autonomous Agents and Multi-Agent Systems, 2004.

[8] G. Pour. *Expanding the Possibilities for Enterprise Computing: Multi-Agent Autonomic Architectures.* Proceedings of the 10th IEEE on International Enterprise Distributed Object Computing Conference Workshops, 2006.

[9] H. Guo, J. Gao, P. Zhu, F. Zhang. *A Self-Organized Model of Agent-Enabling Autonomic Computing for Grid Environment.* Intelligent Control and Automation, 2006.

[10] F. M. T. Brazier, J. O. Kephart, H. Van Dyke Parunak, M. N. Huhns. *Agents and Service-Oriented Computing for Autonomic Computing: A*

Research Agenda. IEEE Internet Computing, vol. 13, no. 3, pp. 82-87, May/June 2009.

[11] N. R. Jennings, K. Sycara, M. Wooldridge. *A Roadmap of Agent Research and Development*. Autonomous Agents and Multi-Agent Systems, vol 1, pp 7-38, 1998.

[12] L. Gardelli. *Ingegneria di Sistemi Auto-organizzanti con il Paradigma Multiagente*. Tesi di dottorato, 2008.