ALMA MATER STUDIORUM
UNIVERSITÀ DEGLI STUDI DI BOLOGNA

Seconda Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica

L'UTILIZZO DELLO STANDARD WSDM PER LA COSTRUZIONE DI SISTEMI SELF-* AD AGENTI, NELL'AMBITO DELL'AUTONOMIC COMPUTING

Elaborata nel corso di: Sistemi Multiagente

A cura di: MICHELE MORGAGNI

ANNO ACCADEMICO 2010–2011

PAROLE CHIAVE

autonomic computing

WSDM

risorse

agente

gestione automatizzata

Indice

Introduzione

L'evoluzione del software, le nuove tecnologie e le astrazioni sempre di piú alto livello, disponibili in campo informatico, hanno portato alla necessitá di creare sistemi e applicazioni sempre piú complesse, composte molto spesso da tante parti di grandi dimensioni e disposte in ambito distribuito. Inoltre tali sistemi hanno requisiti di funzionamento 24/7 e questo, unitamente alla loro vastitá e complessitá, rende praticamente impossibile la manutenzione ed il controllo da parte di utenti umani. A completare il quadro troviamo requisiti di funzionamento su piattaforme completamente eterogenee, integrazione di parti attraverso internet (anche per la comunicazione con eventuali utenti interessati ai servizi), e la necessitá di inserire a run-time parti nemmeno ipotizzabili a design-time.

L'Autonomic Computing tenta di risolvere tutte queste problematiche, cercando di creare sistemi in grado di gestirsi autonomamente, ragionando su scale dimensionali e temporali completamente impraticabili per gli utenti umani; delle sue origini e dei suoi aspetti fondamentali parleró nel primo capitolo.

L'Autonomic Computing inizialmente é nato solamente con specifiche di controllo dei sistemi e per la pianificazione e la gestione di momenti di ripristino automatizzato in sistemi pre-esistenti; in seguito ha poi variato i suoi obiettivi, spostandosi verso il tentativo di garantire comportamenti autonomi stabili, spostando cosí la sua attenzione verso altre aree dell'informatica: come la programmazione ad agenti, l'architettura distribuita ed il P2P. Il secondo capitolo tratterá queste parti.

La vasta varietá di standard legati al mondo di internet ha portato alla costituzione dello standard WSDM (Web Services Distributed Management), che si rivela nell'ambito dell'autonomic computing co-

me un'importante pedina, utile soprattutto per l'integrazione di nuove parti nel sistema a run-time; dei suoi aspetti fondamentali e del suo utilizzo in questo campo parleró nel terzo capitolo.

Il modello riportato nei primi tre capitoli dell'articolo presenta svariati aspetti, ed altrettante problematiche nascono dai tentativi di realizzazione pratica di sistemi in questo ambito, nel quarto capitolo mi focalizzeró su questi, oltre a dare un'immagine dello stato attuale in questo campo.

Capitolo 1

I fondamenti dell'Autonomic Computing

L'Autonomic Computing ha come scopo la realizzazione di sistemi self-* per la gestione automatizzata di sistemi particolarmente complessi, tali da richiedere uno sforzo sovradimensionato per degli operatori umani.

Per self-* si intendono sistemi capaci di reagire autonomamente a stimoli ed eventi (interni ed esterni) in modo da garantire in primo luogo alcuni aspetti fondamentali: la sopravvivenza dell'intero sistema, il suo buon funzionamento in relazione alla non predicibilitá di ció che potrebbe accadergli, la resistenza ad errori ed attacchi esterni; ed in secondo luogo ad altri aspetti importanti, legati ad una riduzione del dispendio economico e temporale oltre che ad un miglioramento della qualitá del sistema: integrazione di nuove parti e loro configurazione, ottimizzazione del comportamento del sistema, capacitá di prendere decisioni in anticipo e con benefici a lungo termine.

L'obiettivo é quello di costruire sistemi particolarmente robusti, capaci di garantire il loro buon funzionamento indipendentemente da eventi interni ed esterni al sistema: attacchi, errori, problemi di performance, aggiunta di nuovi componenti; per questo motivo le prime idee relative a questo paradigma sono state ispirate dal comportamento del sistema nervoso umano, il quale é capace di reagire molto rapidamente, garantendo il buon funzionamento di tutti i meccanismi interni al corpo umano, e mantenendo alcuni valori fondamentali (pressione

sanguigna, temperatura corporea, ...) all'interno di opportuni range ottimali.

Non a caso alcune riflessioni relative all'Autonomic Computing [1] presenti in letteratura prendono direttamente spunto dal sistema nervoso, realizzando per alcuni aspetti quasi una simulazione di questo sistema; altri approcci invece sintetizzano solamente quelli che sono gli aspetti fondamentali, adattando poi le parti rimanenti a quello che é il reale obiettivo di questo studio (ben diverso dalla simulazione di sistemi biologici): la creazione di sistemi tecnologici (informatici, software, di telecomunicazione, ...) con un certo grado di autonomia, capaci di funzionare mettendo in atto diversi aspetti relativi all'auto-gestione.

1.1 Aspetti fondamentali per la sopravvivenza e il buon funzionamento di un Autonomic Computing System

Gli aspetti trattati in questa sezione [2] [3] sono mirati al mantenimento del sistema in una condizione di buon funzionamento, in relazione alla molteplicitá di eventi interni ed esterni che possono minarne il funzionamento. Da un punto di vista esterno al sistema non é importante come questo avvenga, ma solamente che sia garantito un certo livello di funzionamento del sistema (comportamenti e performance accettabili), e che la reazione ad eventi particolari avvenga rapidamente: cioé prima che il sistema venga compromesso, e contemporaneamente riducendo al minimo l'impatto economico dovuto ad errori o attacchi esterni (in questa parte delle specifiche é piú importante la reattivitá del sistema rispetto alla sua pro-attivitá). Tra questi aspetti troviamo la consapevolezza da parte del sistema di se stesso (e delle sue parti) oltre che dell'ambiente in cui é situato, la capacitá di auto-proteggersi e di auto-ripararsi in caso di problemi.

1.1.1 Self-Awareness (consapevolezza di se)

La consapevolezza di se e delle sue parti permette al sistema di riconoscere i propri obiettivi ed il comportamento atteso: in questo modo puó monitorarsi e verificare che i suoi comportamenti siano adeguati

in relazione ai propri obiettivi e politiche di alto livello, e che il suo stato interno sia consistente; inoltre puó rendersi conto dei propri malfunzionamenti e reagire opportunamente (personalmente o attraverso la richiesta di contributo dall'esterno).

1.1.2 Contextually-Aware (consapevolezza del contesto)

La conoscenza del contesto in cui si trova il sistema permette un piú facile adattamento del sistema ai diversi ambienti in cui si trova; comporta inoltre la capacitá di reagire opportunamente ai cambiamenti esterni, attraverso la percezione di stimoli provenienti dall'ambiente, ragionando sulla situazione specifica e pianificando le operazioni ottimali da eseguire.

1.1.3 Self-Protecting (auto-protezione)

L'auto-protezione di un sistema non riguarda solamente aspetti legati ai tentativi esterni maliziosi, ma consiste anche nella reazione ad errori o a problemi interni ad esso. Il sistema deve essere resistente ad attacchi intenzionali esterni, individuandoli e rendendoli inefficaci attraverso tecniche appartenenti all'ambito della sicurezza informatica: in alcuni casi l'individuazione di questi attacchi potrebbe rivelarsi molto complessa e richiedere sforzi particolari (ad esempio per attacchi con come obiettivo il denial of service). Il sistema deve essere altrettando robusto ad errori interni, essere in grado di rimediare prontamente e a run-time (molto spesso abbiamo a che fare con sistemi in esecuzione 24/7), e correggere problemi che potrebbero portare ad una propagazione degli errori in cascata. Inoltre un buon sistema dovrebbe essere anche capace di prevedere problemi prima ancora del loro verificarsi, grazie all'aggregazione di diversi dati (ad esempio legati alle performance del sistema), e di agire preventivamente in modo da annullare i danni derivanti.

1.1.4 Self-Healing (auto-riparazione)

La capacitá di auto-ripararsi comporta innanzitutto la necessitá di individuare danni e malfunzionamenti: il sistema deve riuscire ad individuare le cause di errore (l'interconnessione attuale dei sistemi IT rende tutto ció piú difficile) attraverso tecniche complesse capaci, ad esempio, di analizzare i file di log e ripristinare la parti del sistema compromesse (é quindi necessario anche un meccanismo periodico di backup delle diverse parti di un sistema, oltre alla possibilitá di ripristino a run-time ed in modo consistente rispetto allo stato attuale del sistema).

1.2 Aspetti fondamentali all'aumento della qualitá di un Autonomic Computing System

Gli aspetti trattati in questa sezione [2] [3] hanno un ruolo meno legato alla sopravvivenza del sistema, ma importantissimi per un miglioramento del sistema in termini di manutenzione, messa in esercizio e sfruttamento ottimale delle potenzialitá. Questa parte é fondamentale per una riduzione dei costi e dei tempi legati agli aspetti del sistema sopracitati permettendo, ad esempio, l'aggiunta e la configurazione automatica e a run-time di nuove componenti, oppure l'ottimizzazione di parametri all'interno di sistemi molto complessi, in modo da adattarli alle diverse condizioni dell'ambiente in cui é situato (ottimizzazione dei parametri non solo off-line ma anche on-line). Inoltre il sistema deve essere in grado di prevedere eventi futuri e risorse che serviranno in seguito, agendo preventivamente ed in modo pro-attivo: questi aspetti comportano la necessitá da parte del sistema di essere aperto ed in grado di auto-configurare parti nuove e vecchie, di auto-ottimizzarsi e capace di attuare comportamenti anticipatori.

1.2.1 Openness (apertura)

Un Autonomic System deve essere portabile il piú possibile su diverse piattaforme hardware e software, deve interagire e comunicare

con un insieme particolarmente eterogeneo di ambienti ed altri sistemi software, oltre a dover affrontare a run-time svariate situazioni non considerate a design-time: tra queste troviamo la necessitá di aggiungere e rimuovere parti e risorse in modo autonomo, e a seconda degli andamenti dell'ambiente e dell'intero sistema. Per tutti questi motivi é fondamentale pensare e realizzare il software con un occhio di riguardo verso gli standard che rappresentano lo stato dell'arte (piú sono gli standard che il sistema riconosce piú il sistema é capace di interagire con diverse piattaforme e sistemi, piú il sistema é aperto verso l'esterno).

1.2.2 Self-Configuring (auto-configurazione)

Come giá detto riguardo l'apertura, un sistema deve essere in grado di aggiungere e rimuovere parti e risorse a run-time, a seconda delle svariate situazioni che possono verificarsi, non tutte predicibili a design-time; il grado di apertura di un sistema influenza questi aspetti, ma le nuove parti rimangono pur sempre da configurare (e ció comporta nei sistemi tradizionali un considerevole dispendio di personale e tempo). I sistemi auto-configuranti non necessitano di intervento umano per l'aggiunta di nuovi componenti, questi ultimi saranno in grado di configurarsi da soli partendo dalla conoscenza di se e da politiche di alto livello specificate dall'amministratore (queste indicano cosa deve fare il componente e non come), di comunicare al sistema le nuove funzionalitá messe a disposizione e come utilizzarle.

1.2.3 Self-Optimizing (auto-ottimizzazione)

I sistemi complessi contengono svariati parametri di funzionamento con influenze spesso non-lineari, questo spesso comporta lunghi periodi di tuning precedenti alla messa in esercizio del sistema (oppure aggiornati manualmente nel caso ci si renda conto di una degradazione delle performance). I sistemi auto-ottimizzanti sono in grado di cambiare il valore di questi parametri a run-time e con una frequenza molto maggiore (eventualmente anche in modo quasi continuo), non solo reattivamente rispetto al rilevamento del degrado di performance, ma anche pro-attivamente nei confronti delle politiche del sistema.

1.2.4 Anticipatory Behaviour (comportamento anticipatorio)

Il comportamento anticipatorio di un sistema ricopre diversi aspetti trattati precedentemente (auto-protezione, auto-ottimizzazione, ...) e permette di evitare grossi dispendi economici prevedendo, ad esempio, la necessitá di risorse future per le quali i tempi di negoziazione e messa in esecuzione sono particolarmente lunghi, oppure semplicemente adattando molto velocemente i parametri del sistema in modo da non rilevare mai dei cali delle performance. Questo aspetto é inoltre legato strettamente alla pro-attivitá del sistema, dando la possibilitá al sistema di perseguire degli obiettivi a lungo termine in relazione agli eventi futuri previsti dal sistema.

1.3 Il paradigma derivato dal sistema nervoso umano

L'Autonomic Computing prende spunto dal funzionamento del corpo umano e dal suo sistema nervoso, alcuni articoli presenti in letteratura mantengono questa relazione in modo molto stretto, cercando di costruire sistemi i cui meccanismi di funzionamento sono direttamente ricavati dal comportamento del corpo umano in risposta ad eventi esterni quali variazioni di temperatura, attacchi patogeni o altro. Caratteristica del corpo umano é quella di avere svariati parametri (pressione sanguigna, concentrazione del glucosio nel sangue, ...) i cui valori devono essere garantiti all'interno di opportuni intervalli: per mantenere il corpo in buono stato il nostro sistema nervoso reagisce opportunamente appena rileva valori a rischio, ad esempio, immettendo nel sangue opportune sostanze capaci di riportare il piú velocemente possibile l'organismo ad uno stato di equilibrio. Il principio é quindi quello di reagire ad ogni evento esterno con un comportamento tale da avere effetti contrari.

Inoltre gli studi che si muovono in questa direzione mettono in evidenza due tipologie di eventi (interni o esterni al sistema): la prima categoria consiste in eventi piccoli e molto frequenti, con scarso impatto sull'organismo e a cui é necessario reagire per garantire l'ot-

timalitá del comportamento e del funzionamento del sistema, effettuando piccoli aggiustamenti e variazioni di opportuni parametri. La seconda categoria contiene altri tipi di eventi, molto meno frequenti e con impatto decisamente di grado superiore, quali possono essere gravi patologie del corpo umano, che nell'equivalente metafora tecnologica si traducono in malfunzionamenti di parti fondamentali del sistema; che comportano reazioni di piú ampia portata (backup, sostituzione di parti, aggiunta di nuove componenti, ...). Un meccanismo di questo tipo sará quindi caratterizzato dalla presenza di un ambiente con cui il sistema puó interagire in due modi fondamentalmente diversi: sará presente una parte reattiva del sistema collegata all'ambiente attraverso sensori e attuatori, capaci di rilevare tutte le piccole variazioni e reagire immediatamente; e un'altra parte caratterizzata da quelle che sono le variabili essenziali del sistema, influenzate direttamente sia dall'ambiente che dallo stato interno, che comportano una reazione da parte del sistema solo nel caso in cui i loro valori fuoriescano da opportuni intervalli (limiti fisiologici nell'equivalente biologico); la dimensione di questi intervalli influenza la portata delle reazioni, e dipende dalla stabilitá specifica del sistema.

Altre considerazioni sono derivate dallo studio del sistema nervoso umano, da cui si prende spunto per le sue capacitá adattative intrinseche, che vengono poi riprodotte (almeno nei loro aspetti essenziali) in diversi ambiti dell'Autonomous Computing. Il sistema nervoso umano si divide in Sistema Nervoso Centrale e Sistema Nervoso Periferico, quest'ultimo ha il compito di mediare le interazioni tra il mondo esterno ed il Sistema Nervoso Centrale: in una direzione ció avviene attraverso la percezione di stimoli provenienti dall'esterno rilevati attraverso opportuni recettori; mentre nell'altra direzione viene attuata attraverso i muscoli e le ghiandole (ad esempio) controllati da segnali provenienti dall'interno; il Sistema Nervoso Centrale si divide a sua volta in sistema nervoso sensoriale e sistema nervoso autonomo, quest'ultimo ha il compito di gestire le attivitá involontarie del nostro organismo, quali ad esempio il battito cardiaco. Inoltre é opportuno distinguere tra ambiente interno ed esterno, entrambi possono produrre stimoli ai quali il sistema nervoso deve reagire in modo piú o meno reattivo; in ogni caso l'obiettivo é quello di garantire la stabilitá e il buon funzionamento della globalitá del sistema, indipendentemente

dalle diverse sotto-parti, con la possibilitá di bilanciare malfunzionamenti ed errori (nei casi semplici), o di ripristinare o sostituire intere parti (nei casi piú problematici).

Capitolo 2

Le diverse dimensioni dell'Autonomic Computing

Le specifiche fondamentali dell'Autonomic Computing sono tali da definire un'insieme di funzionalitá del sistema (o capacitá) senza focalizzarsi sulle tecniche implementative e le diverse scelte realizzative. Per via di questa libertá in letteratura possiamo notare una gran variabilitá di approcci, con differenze attribuibili principalmente alle piattaforme e alle diverse conoscenze ed interessi degli sviluppatori. Le diverse dimensioni che ne derivano sollevano problematiche quali l'utilitá di un approccio ad agenti rispetto ad altri (ad esempio rispetto al piú affermato paradigma ad oggetti), o l'organizzazione dei componenti di un sistema (ad esempio struttura gerarchica contro struttura peer-to-peer), o la distribuzione delle risorse (con correlate tutte le problematiche di consistenza, coesistenza temporale e di integrazione tra supporti eterogenei).

2.1 L'approccio ad agenti

Questo tipo di approccio é particolarmente adatto a modellare le entitá costituenti degli Autonomic Computing Systems [3]. La possibilitá di avere a livello concettuale e implementativo l'astrazione di agente, ovvero di un'entitá autonoma, é particolarmente adatta alla realizzazione di sistemi self-* (che sono per definizione di natura autonoma) e che

devono necessariamente essere sia reattivi che pro-attivi per svolgere opportunamente i loro compiti.

Gli agenti specifici dell'Autonomic Computing seguono un modello composto strutturalmente da un insieme di Autonomic Elements, ognuno suddividibile in Resource, Autonomic Manager e Touchpoint. Questi tre elementi rappresentano l'architettura fondamentale di questi componenti, in particolare l'Autonomic Manager rappresenta la parte aggiuntiva di questo paradigma rispetto ai modelli precedentemente utilizzati, volta ad aggiungere le funzionalitá di nostro interesse; mentre la Managed Resource rappresenta ció che é tipico del sistema e che vogliamo migliorare rispetto a realizzazioni piú tradizionali (ovvero la nostra risorsa vera e propria: ad esempio un database, un file, un dispositivo).

Un Autonomic Manager ha quindi il compito di gestire la risorsa: questo consiste nell'ottenere informazioni riguardanti quest'ultima attraverso un Touchpoint (ovvero un'interfaccia con funzionalitá tali da facilitarne il compito, tali da permettere l'estrazione di dati e la modifica dello stato della risorsa o del suo comportamento in modo agevole), analizzare i dati appena ottenuti, pianificare le successive operazioni possibili e definire quali tra queste sono le ottimali in relazione allo stato della risorsa, ed infine eseguire opportune operazioni verso l'esterno (ad esempio altri Autonomic Elements) o sulla risorsa (eseguite sempre attraverso il TouchPoint). Un Autonomic Manager ha quindi tipicamente un comportamento ciclico chiamato MAPE, acronimo delle sue fasi principali: Monitor, Analyze, Plan, Execute; le fasi Monitor ed Execute avvengono rispettivamente grazie all'utilizzo di sensori ed attuatori disposti sulla risorsa, contenuti o controllati dallo stesso Touchpoint, i quali solitamente mettono a disposizione del Manager operazioni e funzionalitá piú o meno evolute e/o di alto livello.

Il ruolo di WSDM (che verrá trattato approfonditamente nei capitoli successivi) va a collocarsi proprio in queste interfaccie di comunicazione tra risorse e manager, garantendo strumenti validi e standardizzati tali da facilitare le operazioni sulle risorse, oltre a supportare interazioni di varie tipologie adatte a situazioni ed esigenze di comunicazione diverse. Tra queste le piú comuni sono la comunicazione di tipo Request-Response, in cui a fronte di una richiesta specifica del manager vengono restituire in risposta le informazioni richieste; oppu-

re la modalitá Register-Notify in cui il master specifica il suo interesse riguardo determinati argomenti (topics) attraverso una registrazione, mentre il Touchpoint si prende carico di notificare al master eventuali eventi significativi appartenenti alle categorie specificate.

2.2 Struttura gerarchica vs P2P

Uno degli aspetti di grande importanza consiste nella scelta della struttura di alto livello del sistema, che deve essere realizzato in modo da bilanciare opportunamente gli aspetti positivi e negativi dei vari approcci possibili: gerarchici o tra pari (P2P), molto spesso trovando la scelta migliore in una soluzione ibrida [4].

In un approccio completamente gerarchico tutto viene gestito da un singolo master, tale da rappresentare il punto fondamentale del sistema, il cui buon funzionamento é sufficiente a garantirne la stabilitá (diventando punto fondamentale di monitoraggio e mantenendo il controllo e il potere decisionale sui livelli inferiori della gerarchia), ma contemporaneamente tale da rappresentare un enorme punto di vulnerabilitá (infatti un malfunzionamento a questo livello potrebbe avere effetti catastrofici per l'intero sistema). Con una struttura gerarchica abbiamo quindi un'unica entitá complessa, potente ma anche molto delicata, facilmente ispezionabile dall'esterno e tale da rappresentare un efficace tramite tra l'esterno e le varie parti del sistema, con requisiti molto elevati di robustezza, tali da non permettere errori.

Un approccio completamente diverso é il P2P, in cui le entitá non percepiscono affatto elementi master capaci di assegnare compiti ad elementi slave, tantomeno vi saranno referenti per problemi ed eventi avvenuti a livello inferiore, non essendoci affatto una struttura gerarchica: ogni elemento (peer) prenderá decisioni singolarmente e indipendenti, a fronte della comunicazione con gli altri elementi del sistema e del proprio stato interno, ma senza sottostare passivamente a nessun ordine proveniente da altre entitá; questi sistemi sono in genere molto piú robusti, ma anche piú complessi e piú difficili da sviluppare, perché necessitano di politiche complesse di alto livello, che molto spesso possono essere realizzate piú facilmente attraverso l'utilizzo di paradigmi ad agenti.

Capitolo 3

Ruolo e possibili utilizzi di WSDM

Lo standard WSDM (Web Services Distributed Management) é standard OASIS dal 2005 e si occupa della gestione di risorse attraverso l'utilizzo di Web Services, con il compito di permettere l'accesso ad esse attaverso modalitá standardizzate, ma lasciando un buon grado di libertá riguardo allo sviluppo di tali interfaccie, grazie ai diversi aspetti e funzionalitá che lo sviluppatore puó decidere se utilizzare o meno. L'utilizzo di Web Services permette di sfruttare tutti i grandi vantaggi che comportano in ambito distribuito; WSDM inoltre ha come obiettivo la gestione automatizzata di queste risorse, adattandosi molto bene all'utilizzo che ne dobbiamo fare nell'ambito dell'autonomic computing, giá specificato nei capitoli precedenti: si pone come tramite tra gli Autonomi Manager e le risorse, situandosi quindi in corrispondenza dei Touchpoints.

Questo capitolo non vuole riprendere tutti gli aspetti di WSDM, ma solamente quelli di maggior interesse in relazione all'argomento trattato, rimando alla mia tesi di laurea triennale [9] o alle relative fonti [7] [8] per maggiori informazioni riguardanti questo standard.

3.1 Gli aspetti fondanti di WSDM

L'utilizzo di Web Services per la gestione di risorse garantisce di per se un insieme di vantaggi e aspetti, giá tipici dei sistemi distribuiti, tra

i quali l'interoperabilitá, il disaccoppiamento, l'indipendenza dalle diverse implementazioni possibili, la trasparenza; inoltre tra gli obiettivi degli sviluppatori di questo standard troviamo la volontá di costruire interfacce di accesso alle risorse sotto forma di WS (Resource Orientation), nascondendo quindi le diverse implementazioni possibili interne alla risorsa , forzando l'accesso e la visibilitá dall'esterno al passaggio attraverso l'interfaccia (Agent architecture agnostic). Lo standard é stato pensato per adattarsi ad un insieme molto vasto di problematiche di tipo diverso, per questo motivo é composto da tante parti che il progettista puó decidere di attuare o meno, in relazione alla problematica specifica di cui si sta occupando; molte specifiche consistono in possibilitá date al progettista piuttosto che in vincoli a cui sottostare necessariamente, potendo cosí adattarsi efficacemente a casa di complessitá anche molto diversa (Composability). Le specifiche sono pensate per permettere altrettando efficentemente l'ispezione di nuove risorse o entitá sia a design-time che a run-time, a seconda che all'avvio del sistema si abbiano a disposizione dati relativi alle risorse future oppure nessuna informazione (Design-time and run-time inspection). La grande versatilitá dello standard é dovuta anche dalla possibilitá di definire i propri modelli attraverso l'utilizzo di XML Schema, in diverse delle situazioni sopra citate.

WSDM si poggia su diversi standard o sistemi pre-esistenti nell'ambito dei Web Services, ed utilizza i diversi aspetti di maggior utilitá nell'ambito della gestione di risorse, sia attraverso specifiche di alto livello riguardanti i WS (tra cui ad esempio WS-Resource Framework, WS-Notification, WS-Addressing) sia attraverso specifiche necessarie per il funzionamento generale dei WS (SOAP, WSDL, XML, XML Schema).

3.2 Le modalitá di interazione supportate

WSDM prevede diverse tipologie di interazione con le interfaccie previste da questo standard, e ad ognuna di esse associa un insieme di MEPs (Message Exchange Patterns) standard, a cui é possibile affiancarne altri liberamente creati dagli sviluppatori dei singoli sistemi.

La prima modalitá di interazione prevede una richiesta esplicita da parte del manager riguardo a una o piú proprietá della risorsa, ed una risposta da parte della risorsa, le MEPs standardizzate corrispondenti sono:

- GetResourceProperty: richiede una singola proprietá
- GetMultipleResourceProperty: richiede piú proprietá con un'unica richiesta
- QueryResourceProperty: effettua una query partendo da uno specifico template
- QueryRelationshipByType: richiede informazioni sulle relazione tra risorsa ed altre, in modo da raggiungere risorse collegate a quella interrogata

La seconda modalitá prevede la modifica di una o piú proprietá della risorsa, avviene con un'unica MEPs attraverso la quale si puó modificare la risorsa in qualsiasi modo:

- SetResourceProperty: modifica una o piú risorse

La terza modalitá prevede la sottoscrizione da parte del manager ad un argomento, e la notifica da parte della risorsa o di un intermediario (un Notification Broker) degli eventi significativi appartenenti a tali argomenti, le MEPS standardizzate sono:

- Subscribe: utilizzata dal manager per indicare il suo interesse riguardo un determinato topic
- GetCurrentMessage: per richiedere l'ultima notifica relativa ad un determinato argomento
- PauseSubscription: per arrestare temporaneamente la notifica di messaggi relativi ad una determinata sottoscrizione
- ResumeSubscription: per ripristinare la notifica dopo una PauseSubscription
- Notify: utilizzata dalla risorsa per notificare le notizie al manager

- RegisterPublisher: per registrare una risorsa presso un broker
- Destroy: per eliminare la registrazione presso un broker

Le diverse modalitá di interazione unitamente alle relative MEPs consistono in strumenti utili in mano a manager agenteschi, con capacitá superiori rispetto a elementi software tradizionali; inoltre é da tenere bene in considerazione il fatto che lo standard é stato pensato in modo aperto, permettendo agli sviluppatori di creare le proprie MEPs, ed in questo modo perfezionare quelle giá esistenti oppure crearne di specifiche per l'ambiente o caso di studio particolare.

3.3 Advertisement e Discovering

Questi aspetti di WSDM sono trattati nella documentazione in modo rapido e secondario rispetto ad altre parti, ma nel nostro caso hanno un aspetto molto importante, perché garantiscono sistemi molto efficaci riguardo l'integrazione di nuove parti nel sistema. Come abbiamo detto nei capitoli precedenti, il nostro sistema viene progettato senza una consapevolezza precisa riguardo a come dovrá evolversi ed adattarsi durante il suo lungo tempo di vita: per questo il sistema deve essere in grado di procurarsi nuove componenti software nel momento in cui ne senta la necessitá (in modo sufficientemente rapido da non comportare perdite economiche ed in efficenza), oppure di considerare immediatamente nuove risorse aggiuntive nel momento in cui vengono rese disponibili ai diversi componenti (ad esempio un componente del sistema puó rilevare la necessitá di una nuova risorsa e richiederla per tempo, immediatamente tutti gli altri componenti che possono trarre vantaggio da quest'ultima devono conoscerne l'esistenza, in modo da poterla utilizzare se necessario).

Le tecniche di Advertisement hanno il compito di portare a conoscenza dei vari componenti del sistema l'esistenza di risorse e le relative modalitá di interazione. Tutto ció avviene grazie all'esistenza di opportuni topic di interesse (vedi terza modalitá di interazione tra manager e risorse) la cui sottoscrizione permette di ricevere informazioni relative alla creazione e distruzione di risorse ed endpoints. Questo meccanismo puó venire ulteriormente raffinato attraverso la

creazione di diversi topic, relativi alla creazione e distruzione di risorse di tipologie e con funzionalitá specifiche (ad esempio un componente con il singolo compito di salvare dei report puó essere interessato solamente a risorse consistenti in aree di memoria in cui poter scrivere). Lo standard prevede anche l'esistenza di registri in cui inserire i dati relativi alle risorse, esplorabili da parte delle diverse componenti del sistema, oltre alla possibilitá di risalire a risorse attraverso l'utilizzo della MUWS Relationship Capability.

3.4 Le componenti di una WS-Resource

Lo standard prevede l'esistenza di almeno quattro file di diverso tipo, necessari per la gestione della risorsa attraverso l'utilizzo della WS-Resource:

- L'interfaccia del WebService descritta in WSDL (tipica di tutti i WS)
- Il Resource Property Document (RPD) in formato XML, contenente le caratteristiche statiche della risorsa, alle quali componenti esterni possono essere interessati
- Un file XML Schema tale da descivere la formattazione ed il contenuto del RPD
- Un metadata document contenente le caratteristiche dinamiche della risorsa
- Eventualmente altri file XML Schema necessari a descrivere le informazioni scambiate tra WS-Resorce e la risorsa vera e propria

Capitolo 4

Problematiche di realizzazione e stato attuale dell'autonomic computing

Questa sezione tratterá quelli che sono gli interrogativi piú importanti in cui uno sviluppatore di sistemi software, legati all'autonomic computing e al mondo agentesco, é destinato ad imbattersi; inoltre porteró alla luce diversi aspetti e considerazioni derivanti dagli sviluppi effettivi di sistemi di questo tipo, relative ad implementazioni pratiche realizzate da persone nel settore (alcune di esse aventi a che fare anche con lo standard WSDM).

Ho suddiviso questo capitolo in una prima parte piú generale relativa all'autonomic computing realizzata attraverso l'uso di agenti, in una seconda parte riguardante lo standard WSDM utilizzato in questo campo, ed infine nell'analisi di alcuni casi di studio e delle problematiche sollevate da questi ultimi.

4.1 Problematiche relative all'AC ad agenti

Le politiche di un sistema possono essere esaminate a diversi livelli, nel campo dell'AC interessano almeno due livelli (agli antipodi uno dall'altro): in una visione di alto livello (in cui il sistema viene esa-

minato nella sua interezza) possiamo individuare diverse macro-parti, tipicamente é possibile individuare una parte che costituisce il cuore dell'intero sistema e che é a conoscenza di quelle che sono le politiche di alto livello, questa parte puó non consistere in un punto di centralizzazione, infatti puó essere composta da tante piccole entitá distribuite capaci di comunicare tra loro e con compiti diversi; questa parte solitamente non si occupa di effettuare compiti ma si limita a delegarli o distribuirli alle parti del sistema di piú basso livello. Queste ultime hanno visioni piú limitate e possono essere passive o attive, comportando cosí modelli architetturali anche molto diversi: entitá passive porteranno ad un modello di tipo Master-Worker e si limiteranno ad eseguire i compiti provenienti dall'alto, in modo efficace ed efficente, senza conoscere le politiche del sistema; mentre entitá attive avranno accesso a sotto-politiche derivate dalla parte master del sistema, oppure verranno informate attraverso comunicazioni dei compiti da svolgere, mantenendo la possibilitá di decidere se effettuare o no operazioni, e libere di scegliere le modalitá piú adatte secondo la propria natura decisionale, e dipendenti dalla propria idea di sistema; in entrambi i casi rimane un livello di trasparenza tra i diversi livelli: le entitá di basso livello non sono a conoscenza né dell'architettura né delle politiche di quelle di alto livello, mentre quelle di basso livello nascondono la loro struttura interna, la loro logica ed il loro funzionamento, garantendo all'esterno solamente la visione dei risultati ottenuti e di opportuni feedback. Posizionandosi a livelli cosí diversi si nota una grande differenza in scala temporale: ad alto livello il sistema ragiona in modo piú lento e complesso, prendendo decisioni a lungo termine, con natura di tipo proattiva e con eventi significativi poco frequenti, relativi all'aggregazione di molti eventi di piú basso livello (ad esempio l'andamento del sistema effettuato con cadenza giornaliera, settimanale o mensile); mentre tra le entitá di basso livello abbiamo comunicazioni molto piú frequenti, cicli di esecuzione molto piú brevi ed eventi significativi a cui é necessario reagire in modo reattivo (ad esempio i segnali provenienti da sensori posti sulle risorse), senza nemmeno informare le parti di piú alto livello.

I sistemi autonomi sono per loro natura particolarmente complessi e lo sviluppo di questi, in ambito distribuito, aperto, con forti requisiti di buon funzionamento ed ispezionabilitá, comporta molto spesso il

passaggio in secondo piano degli aspetti di robustezza e adattativitá legati all'utilizzo di architetture P2P. Per questo motivo attualmente nell'Autonomic Computing é molto piú frequente trovare sistemi con un cuore centralizzato (unico manager che si fa carico dei compiti piú complessi e delicati) oppure sistemi P2P con un unico manager definito tramite elezione (in questo modo il sistema é sostanzialmente centralizzato ma é piú resistente, potendo eleggere un nuovo manager in caso di difficoltá); questi sistemi ricalcano quindi il modello Master-Worker, con molte entitá Slave semplici solitamente passive e capaci di occuparsi di problemi mono-caratteristica, inoltre sono maggiormente ispezionabili perché é sufficiente posizionarsi in corrispondenza di un'unica entitá per avere un quadro ampio di tutto il sistema. Un'altra variazione semplificata del modello ideale prevede entitá attive sia per i compiti di pianificazione e ragionamento sia per i compiti di piú basso livello, utilizzando peró un'unica base di conoscenza accessibile da tutti i livelli (seppur molto spesso con privilegi differenti), nella quale anche contratti e vincoli vengono inseriti.

Uno tra gli obiettivi principali dell'AC consiste nell'apertura dei sistemi e nella capacitá di procurarsi risorse e servizi autonomamente, tutto ció in realtá apre diverse problematiche relative non solo alla notifica di nuove risorse utilizzabili, ma anche alla contrattazione per ottenerne l'accesso (a volte anche particolarmente complesse e tali da richiedere l'utilizzo di un utente umano per sottoscrivere contratti) attraverso ontologie non ancora esistenti o affermate; inoltre nei casi in cui i tempi di negoziazione siano molto lunghi é necessario che il sistema sia particolarmente proattivo e riesca a prevedere le future aggiunte (che molto spesso comportano anche investimenti economici) per tempo e con un alto grado di infallibilitá.

L'aspetto forse piú importante da tenere in considerazione quando si parla di sistemi con componenti attivi, é il fatto di avere a che fare con entitá indipendenti: questo comporta dei vantaggi, tra cui la capacitá di parti del sistema di poter accorgersi di errori a livello superiore (sono sufficientemente intelligenti da accorgersi di problemi ai livelli superiori nella gerarchia); ma anche degli svantaggi in quanto la complessitá delle diverse parti é molto piú alta e molto spesso non é nemmeno possibile testare le diverse parti del sistema (se non in piccole simulazioni in ambienti ridotti), comportando dei

rischi soprattutto economici. Per questo motivo é molto difficile trovare progetti di questo genere in un campo cosí profondamente legato al fattore economico, in cui si tende a minimizzare i rischi, e nel quale si tendono a preferire tecnologie ed architetture piú tradizionali e maggiormente monitorabili rispetto a sistemi piú innovativi (in questo campo potrebbero risultare molto utili anche algoritmi di learning oppure sistemi capaci di comportamenti emergenti, ma i loro transitori lunghi e oscillanti spesso finiscono per scoraggiare molti sviluppatori).

Un'ultimo aspetto non trascurabile consiste nel maggior utilizzo di standard normalmente utilizzati nel mondo dei Web Services, oltre alla costruzione delle diverse parti seguendo un'architettura a componenti, giá pensando a design-time al compito delle diverse parti in relazione agli obiettivi dell'intero sistema: tutto ció rende molto faticoso il compito di chiunque voglia modificare un progetto giá esistente per adattarsi a queste nuove caratteristiche, mentre é molto piú agevole costruire sistemi da zero, rendendo perció dura la strada a chiunque voglia utilizzare questi aspetti dell'autonomic computing in un sistema pre-esistente.

4.2 Problematiche relative a WSDM

Pur essendo nato con obiettivi principalmente diversi da quelli trattati in questo articolo, lo standard WSDM é scritto in modo sufficientemente generico ed adattabile da permettere diverse possibilitá di utilizzo anche in questo campo [5], soprattutto grazie alla sua natura a componenti ed alla possibilitá di inserire le proprie personalizzazioni in caso di necessitá; nel nostro caso vi sono ben due utilizzi diversi che possiamo farne: il primo consiste nella mediazione tra risorse ed agenti, utilizzandolo quindi come punto di accesso alle risorse da parte dell'esterno o da altri agenti; il secondo invece consiste nell'interazione tra i diversi agenti, sfruttando quindi gli aspetti relativi alla standardizzazione di protocolli di comunicazione, in modo da rendere gli agenti capaci di scambiare messaggi secondo diverse modalitá e di avere un protocollo prestabilito per la comunicazione, anche per poter interagire con eventuali nuovi elementi (risorse o entitá) che entreranno a far parte del sistema in futuro. La genericitá con cui é scritto lo

standard non porta peró solamente benefici [5]: lasciando in mano ai progettisti architetture potenti ma senza nessun supporto o consiglio relativamente agli aspetti pratici, si creano difficoltá al momento della realizzazione; non sono presenti linee guida riguardanti l'implementazione, lasciando cosí una vasta scelta di possibilitá ai progettisti, ma anche una grande varietá di rischi ed interpretazioni possibili particolarmente diverse, anche solo tra i diversi componenti dello stesso team di lavoro.

Uno degli aspetti piú critici relativi alla programmazione ad agenti consiste nella definizione del ciclo di lavoro di ogni singolo agente: la problematica non é banale e le difficoltá sono dovute principalemente alla necessitá di trovare il giusto bilanciamento tra reattvitá e proattivitá: un agente deve scegliere opportune politiche riguardanti l'ordine con cui reagire e considerare gli eventi ed i messaggi provenienti dall'esterno (scegliere quale stimolo esterno gestire per primo, attraverso prioritá o altri meccanismi), la lunghezza dei tempi di pianificazione ed esecuzione, in modo da poter effettuare ragionamenti sufficientemente accurati, ma anche sufficientemente veloci da non perdere in termini di reattivitá. Uno dei pregi di WSDM consiste nel permettere di effettuare quasiasi scelta in quanto alla gestione degli stimoli esterni, senza dover considerare particolari vincoli relativi alle comunicazioni, grazie alle diverse modalitá che supporta lo standard tali da adattarsi facilmente ad ogni caso.

WSDM permette diversi benefici (in particolare in termini di apertura ed adattativitá) comportando peró nel sistema un calo delle performance, legati all'utilizzo di meccanismi piú complessi e di un maggior numero di livelli nei protocolli di comunicazione, inoltre comporta tempi piú lunghi per l'analisi e l'implementazione del sistema, una maggior quantitá di codice, e la necessitá di sviluppatori conoscitori dello standard e capaci di realizzarlo in modo pratico (come abbiamo visto é un problema non da poco). Per questi motivi non tutti i casi si addicono a questo tipo di architettura ed é necessaria una fondamentale analisi preventiva, mirata ad evitare grossi investimenti in termini di tempo e forza lavoro, in problemi in cui questa soluzione finirebbe per non rivelarsi efficace.

4.3 Lo stato attuale

Attualmente non sono moltissimi i sistemi reali realizzati utilizzando le specifiche dell'autonomic computing, ancora meno sono i sistemi riguardo cui si possono trovare informazioni relative alla realizzazione ed al funzionamento: le seguenti considerazioni sono derivate dalla scarsa documentazione disponibile.

I sistemi che si basano sull'autonomic computing attualmente hanno un'architettura gerarchica, molto spesso con un'unico master tramite cui l'intero sistema puó essere facilmente ispezionato e controllato; inoltre si tende a minimizzare le entitá attive del sistema prediligendo tecnologie riguardo cui gli sviluppatori hanno maggior conoscenza ed esperienza; molto spesso le uniche entitá attive consistono esclusivamente in uno o piú master disposti gerarchicamente (tutti finiscono per sottostare ad un'unico master in cima alla gerarchia) oppure comandati da un Arbiter, col compito di avviare i diversi componenti e di risolvere eventuali conflitti tra le varie parti; ed in alcuni casi da alcune Sentinels con il compito di monitorare le diverse parti del sistema, verificarne il corretto funzionamento e riferire in casi di problemi direttamente all'Arbiter.

Per dare un'idea concreta di quello che é lo stato attuale, descriveró gli aspetti piú interessanti di uno dei progetti che piú si avvicina alle considerazioni trattate in questo articolo nel campo dell'autonomic computing, ottenuto in casa IBM e di nome Unity [4]; in questo modo esamineremo quali degli aspetti trattati nei capitoli precedenti sono giá stati riscontrati in progetti reali.

- L'accesso alle risorse e la comunicazione tra entitá diverse avviene necessariamente attraverso interfacce documentate, evitando cosí punti di accesso multipli alle risorse e consentendo solo operazioni ritenute opportune.

- Tutte le interazioni tra ogni singolo agente e l'ambiente circostante (compresi altri agenti) sono effettuate attraverso l'uso di un Application Manager, che si occupa di ricavare le informazioni utili all'agente, oltre ad effettuare le operazioni dell'agente sull'esterno.

- E' presente un Resource Arbiter con il compito di gestire le allocazioni delle risorse: ogni volta che un'entitá richiede una risorsa, l'Arbiter esamina tutte le diverse possibilitá ed effettua la scelta ottimale sia per il funzionamento del richiedente sia per l'intero sistema.

- Sono presenti diversi altri componenti necessari per il funzionamento del sistema: dei server, dei registri, dei policy repository (contenenti gli obiettivi ed i goal di alto livello), delle sentinelle (con il compito di individuare eventuali malfunzionamenti) e degli OSContainer (che rappresentano i singoli host su cui mettere in esecuzione le diverse parti del sistema).

- Il sistema é controllato e monitorato attraverso un'unica interfaccia utente attraverso cui é possibile esaminarne lo stato (molto spesso attraverso un utente umano che agisce a polling), o specificare tutte le politiche di alto livello.

- Il sistema é capace di autoconfigurare le diverse parti che lo compongono: attraverso l'utilizzo di specifiche di alto livello e di registri contenenti tutte le risorse ed i servizi disponibili, ogni parte é in grado di configurarsi autonomamente o attraverso l'intervento di opportuni Arbiters.

- Le parti del sistema sono duplicate in modo da resistere meglio a guasti e malfunzionamenti, ma anche per evitare punti di centralizzazione; oltre a risorse ed entitá, anche eventuali sottoscrizioni sono duplicate perché considerate di vitale importanza; il sistema é anche in grado di ripristinare parti di se stesso in caso di errori irrisolvibili attraverso backup automatizzati (self-healing).

- Le diverse parti attive del sistema sono inizializzate e fatte partire una alla volta da un Arbiter, per evitare conflitti al momento dell'avvio; lo stesso Arbiter a sistema avviato si occupa di risolvere problemi e malfunzionamenti (ad esempio autorizzando il ripristino di alcune parti) nel momento in cui vengono individuati dalle Sentinels.

- Una delle problematiche principali consiste nella definizione dei goal del sistema, ed una quantificazione dei benefici e delle priorità con cui perseguirli. Tra le varie possibilitá é possibile specificarli in termini economici, ma molto spesso non é la scelta migliore in questi sistemi, in cui anche la reattivitá e le performance ricoprono ruoli fondamentali; questa rimane una delle problematiche principali, ed anche Unity é molto sensibile alle diverse politiche che si possono effettuare a questo livello.

- L'Arbiter rappresenta l'apice della gerarchia del sistema, la sua visione del sistema é peró limitata alle interfacce messe a disposizione dalle altre entitá e risorse (trasparenza), quindi ragiona utilizzando tecniche di inferenza ed algoritmi di machine learning, utilizzando i feedback provenienti dai livelli inferiori della gerarchia, e molto spesso utilizzando scale temporali diverse (in modo da reagire sia reattivamente che proattivamente).

Questo progetto non tocca tutti gli aspetti trattati nell'articolo, ma possiamo individuare diversi punti di interesse, inoltre possiamo notare che l'implementazione pratica di sistemi di questo tipo comporta la creazione di molte parti attive diverse, ognuna di una certa complessitá.

Capitolo 5

Conclusioni

Lo scopo di questo articolo vuole essere quello di descrivere le basi e lo stato attuale dell'autonomic computing; inoltre ho cercato di ipotizzare uno scenario, in questo campo, in cui lo standard WSDM (Web Services Distributed Management) viene affiancato agli agenti, con l'intenzione di far utilizzare a questi ultimi tutte le funzionalitá messe a disposizione dallo standard. Ció avviene grazie alla suddivisione del sistema in diversi componenti, attivi o passivi, organizzati gerarchicamente oppure secondo altre architetture: ciascun componente puó sfruttare i benefici dei Web Services a proprio vantaggio.

Nel primo capitolo ho cercato di descrivere gli aspetti fondamentali dell'Autonomic Computing, suddividendoli tra necessari alla sopravvivenza del sistema e utili per un incremento della qualitá e delle performance; questa suddivisione é estremamente utile per paragonare i comportamenti delle parti di questi sistemi al sistema nervoso umano, il quale garantisce sia la sopravvivenza dell'organismo, sia la reazione a problemi di piú piccola portata, in modo rapido ma meno prioritario.

Nel secondo capitolo ho descritto l'utilitá di un approccio ad agenti in questo contesto, oltre alla struttura specifica degli agenti utilizzati in questo campo. Inoltre mi sono posto il problema della struttura architetturale e gerarchica del sistema, mettendo in luce vantaggi e svantaggi di sistemi centralizzati e P2P, e giungendo infine ad ipotizzare una soluzione ibrida come soluzione risolutiva.

Nel terzo capitolo ho parlato dello standard WSDM, focalizzandomi sulle parti fondamentali per lo scenario ipotizzato in questo ar-

ticolo. Tra gli aspetti piú utili possiamo trovare diverse modalitá di interazione, in modo da potersi adattare ad una casistica di problemi piú ampia possibile, oltre a poter utilizzare lo standard come supporto per l'inserimento agevole di parti a run-time.

Nel quarto capitolo ho evidenziato alcuni aspetti relativi all'utilizzo di agenti nel campo dell'Autonomic Computing, ma anche relativi allo standard WSDM, concentrandomi su problemi e benefici. Inoltre sempre in questo capitolo ho cercato di descrivere lo stato attuale in questo campo, soffermandomi in particolare su un progetto di casa IBM di nome Unity, particolarmente vicino allo scenario descritto. Quest'ultima parte ha l'utilitá di sostenere la realizzabilitá di sistemi di questo tipo, ancora poco conosciuti e con un'incredibile complessitá relativa ad aspetti sia architetturali che implementativi, seppur con ampie speranze per quanto riguarda la composizione di software capace di auto-gestirsi e auto-monitorarsi.

Bibliografia

[1] Salim Hariri, Bithika Khargharia, Houping Chen, Jingmei Yang, Yeliang Zhang: *The Autonomic Computing Paradigm* Springer Science - 2006

[2] Jeffrey O. Kephart, David M. Chess: *The Vision of Autonomic Computing* IEEE Computer Society - 2003

[3] Marco Cioffi: *Autonomic Computing* Politecnico di Milano - 2006

[4] Gerald Tesauro, David M. Chess, William E. Walsh, Rajarshi Das, Alla Segal, Ian Whalley, Jeffrey O. Kephart, Steve R. White: *A Multi-Agent Systems Approach to Autonomic Computing* IBM T. J. Watson Research center - 2004

[5] P. Martin, W. Powley, K. Wilson, W.Tian, T. Xu, J. Zebedee: *The WSDM of Autonomic Computing: Experiences in Implementing Autonomic Web Services* School of Computing Queen's University Kingston

[6] Alexandru Cicortas, Victoria Iordan: *A Multi-Agent Framework for execution of Complex Applications* Acta Polytechnica Hungarica Vol.3 No.3 - 2006

[7] OASIS: *Web Services Distributed Management: MUWS Primer* - Febbraio 2006

[8] OASIS: *Web Services Distributed Management: MOWS Primer* - Febbraio 2006

[9] Michele Morgagni: *Lo standard WSDM per la gestione di risorse a livello applicativo: panoramica e sperimentazioni* Tesi di laurea triennale - 2008