Metodologie e architetture per l'autoorganizzazione nei MAS

Giacomo Pronti

Gennaio 2010

Indice

Introduzione

Questo articolo ha come obbiettivo quello di approfondire i concetti legati all'auto-organizzazione nei multi-agent system, e di mostrare come questa caratteristica possa essere introdotta direttamente dentro la fase di sviluppo tramite metodologie e modelli mirati. L'articolo si suddivide in 3 capitoli. Nel primo si cercherà di fornire tutte le conoscenze necessarie per comprendere al meglio il ruolo della self-organizazion nei sistemi complessi e tutti i sotto aspetti ad essa legati. Verranno introdotti concetti base come quelli della stigmergia, dei fenomeni emergenti, e verranno illustrati esempi biologici dai quali questi concetti si ispirano. Dopo aver fornito un discreto background concettuale, nel secondo capitolo motiveremo la scelta del paradigma ad agenti per progettare SOS, e i vari approcci studiati per aggiungere proprietà self-organazing ai sistema, a seconda del meccanismo da cui nascono queste proprietà di auto-organizzazione. Infine, nel terzo capitolo mostreremo alcune metodologie e metodi di sviluppo(alcuni più astratti,altri basati su casi studio specifici), e vedremo i vari problemi che devono essere affrontati quanto si vuole sviluppare, controllare e valutare un SOS. Chiuderemo l'articolo con una breve conclusione nel quale discuteremo del lavoro svolto e di possibili lavori futuri relativi a questo ambito.

Capitolo 1

Teoria di base

1.1 Descrizione capitolo

In questo capitolo cercheremo di introdurre e descrivere numerosi concetti teorici, su cui faremo largo riferimento nei capitoli successivi. Parleremo di:

1. Swarm intelligence
2. Decentralizzazione
3. Stigmergia
4. Auto-organizzazione
5. Fenomeni emergenti
6. Feedback positivo e negativo
7. Coordinazione,Cooperazione,Collaborazione e Deliberazione

Inoltre evidenzieremo la profonda e fondamentale differenza fra il comportamento del singolo individuo e il comportamento del gruppo, cioè come il *tutto* a volte possa valere molto di più della somma delle singole parti che lo compongono.

1.2 Swarm Intelligence

Per lunghi anni il comportamento collettivo degli insetti sociali è rimasto, per i naturalisti, un problema affascinante e misterioso. Tutto accadeva come se esistesse qualche agente particolarmente evoluto, che coordinasse le varie attività all'interno della popolazione, ma nessuno riusciva a dare una spiegazione valida a questi fenomeni.
Solo recentemente(a partire dagli ultimi 40anni) è stato possibile dimostrare che le regole che governano le interazioni fra gli insetti sono realizzate sulla base di informazioni locali, quindi senza alcuna conoscenza a livello globale. Nacque cosi una nuova scienza allo scopo di approfondire i concetti legati a questi fenomeni, e sfruttarne le caratteristiche anche in altri campi. La Swarm Intelligence(Swarm significa *sciame*, *flotta*) è proprio quella parte di IA che si occupa di studiare l'intelligenza collettiva emergente in gruppi di agenti semplici.

1.3 Stigmergia

La prima valida teoria (appurato che quelle precedenti non lo fossero!) per dare una spiegazione all'organizzazione degli insetti sociali nello svolgere le loro attività, è stata quella avanzata da Pierre-Paul Grassé [4], il quale introdusse il concetto di stigmergia per spiegare le attività di costruzione nelle termiti. Grassé mostrò che la coordinazione di queste attività non dipendeva dalle termiti stesse, ma bensì era inglobata nella struttura del nido in costruzione. In altre parole, le informazioni derivanti dall'evolversi dell'ambiente locale fornivano una specie di linea guida per il lavoro degli agenti operanti. Quando un individuo compie un azione (per esempio aggiunge un blocco di costruzione al nido), conseguentemente causa anche una variazione della configurazione locale del sistema, la quale a sua volta, in futuro, influenzerà le azioni di altri individui che si troveranno in quella zona. Questo processo conduce ad un' apparentemente perfetta coordinazione fra i lavoratori, dando l'impressione che la colonia stia evolvendo seguendo un piano di sviluppo conosciuto a priori.

1.4 Self-Organization

Il concetto di auto-organizzazione può avere essere definito in diversi modi a seconda del campo in cui viene analizzato. In generale si riferisce a un processo interno tramite cui un sistema tende ad aumentare,variare o mantenere costante il suo livello organizzativo. Questa organizzazione è *Self* in quanto è distribuita tra i componenti interni senza alcuna supervisione esterna. Una definizione molto diffusa di Self-organization è quella fornita da Camazine in un suo articolo del 2001:

> La self-organization è un processo nel quale l'emergere di pattern a livello globale è dovuto esclusivamente a numerose interazioni a livello locale tra i componenti del sistema. Inoltre, le interazioni avvengono sulla base di informazioni locali, senza alcuna conoscenza relativa al sistema globale.

L'emergenza è vista come la comparsa ad alto livello di una struttura, non esplicitamente rappresentata a basso livello. Le proprietà emergenti sono fortemente legate alla self-organization, ma nel caso generale, l'auto-organizzazione puà esistere anche senza emergenza e viceversa. Dal punto di vista dei sistemi artificiali, sono state adottate 2 definizioni per i sistemi self-organizing:

1. *Strong SOS*: sistemi che cambiano la loro organizzazione senza un esplicito controllo centralizzato (interno o esterno)

2. *Weaky SOS*: sistemi che si riorganizzano in relazione a un piano o un controllo interno.

1.4.1 proprietà

I processi auto-organizzati mostrano importanti proprietà, che li rendono ottimali per essere integrati nelle soluzioni dei sistemi più complessi. Possiamo affermare con certezza che:

1. I sistemi auto-organizzati sono dinamici.

2. I sistemi auto-organizzati sono robusti e adattabili agli eventi esterni **(proprietà chiave!)**

3. I sistemi auto-organizzati mostrano proprietà emergenti, cioè proprietà molto più complesse rispetto al semplice contributo di ogni singolo individuo.

4. Le interazioni, essendo non lineari, conducono a biforcazioni. Una biforcazione può esser vista come la comparsa, al variare di alcuni parametri di sistema, di una nuova soluzione stabile. Essa corrisponde a un sostanziale cambiamento all'interno del comportamento collettivo.

5. I sistemi auto-organizzati possono esser multi-stabili. Per multi-stabilità si intende che il sistema, in corrispondenza di un medesimo insieme di parametri, può portare a stati di stabilità differenti, a seconda delle condizioni iniziali e della casualità delle fluttuazioni occorse durante lo sviluppo del sistema stesso.

1.4.2 principi base

In un sistema generico possiamo vedere l'auto-organizzazione come un insieme di meccanismi dinamici per mezzo dei quali vengono a costruirsi delle strutture coerenti a livello globale, tramite interazioni locali fra i suoi componenti, e senza quindi supervisione a livello individuale. Questi meccanismi si basano su 4 principi fondamentali:

1. **Feedback positivo:** Si ottiene tramite l'esecuzione di semplici regole comportamentali che promuovono l'azione di un individuo. Per esempio il rilascio di feromone nelle formiche rappresenta un tipo di feedback positivo, allo scopo di fare emergere delle orme a livello globale disponibili per tutti gli individui della colonia.

2. **Feedback negativo:** Serve per controbilanciare il feedback positivo e per permettere la stabilizzazione di pattern collettivi. Restando nell'esempio delle formiche, possiamo notare questo fenomeno in diverse forme, quali l'evaporazione del feromone, la terminazione della fonte di cibo, la diminuzione degli individui della popolazione, ecc.

3. **Oscillazione stimoli:** Gli agenti nei sistemi Self-Organizing sono soliti svolgere azioni descrivibili stocasticamente. Alcune di queste oscillazioni sono la fonte necessaria per permettere la crescita e lo sviluppo di determinate strutture. Inoltre la casualità è spesso cruciale, in quanto permette di scoprire nuove soluzioni. Sempre nel caso delle formiche ad esempio, individui persi potrebbero trovare nuove fonti di cibo e permettere la creazioni di una nuova traccia da seguire.

4. **Interazioni multiple dirette o stigmergiche:** Sono le interazioni che avvengono tra i componenti del sistema, allo scopo di ottenere un'attività apparentemente deterministica e la nascita di pattern coerenti.

1.5 Emergent Phenomena

Il fenomeno dell'emergenza viene studiato sin dagli antichi greci, e ha portato alla nascita di 2 diverse scuole di pensiero. I *proto-emergentist* considerano il processo d'emergenza come una scatola nera. Solo input e output al livello piu basso possono esser riconosciuti, mentre non possiamo conoscere come gli ingressi vengano trasformati in uscite. Dal 1930 in poi, si iniziò ad investigare una nuova prospettiva, che diete origine al movimento dei *neo-emergentism*. Il suo scopo è quello di sviluppare metodi e tool che permettino di capire e riprodurre i processi che danno vita ai fenomeni emergenti. Anche per l'emergenza come per la self-organization sono state date più definizioni. Una buona definizione tecnica e fortemente legata alla computer science è quella data dal team SMAC e si basa su 2 punti:

1. *the subject*: Il goal di un sistema computazionale è realizzare una data funzione, che evolvendo nel tempo dovrà emergere.

2. *the condition*: Questa funzione è emergente se il codice del sistema non dipende in nessun modo dalla conoscenza di tale funzione.

Un fenomeno può definirsi emergente quando si può notare ad alto livello la produzione coerente di qualcosa di nuovo che non esisteva

precedentemente. Inoltre, tale fenomeno richiede almeno 2 livelli (micro e macro livello) e necessita di essere osservabile per lo meno a livello macro.

1.6 Edge of caos

Una catena di attività lineari garantisce la predicibilità di un qualsiasi fenomeno collettivo. Ovviamente un fenomeno emergente,in quanto non predicibile per definizione, necessiterà di attività non-lineari a livello micro per poter riscontrarsi. Essendo osservabile nel tempo, deve riuscire a mantenere una forma di auto-equilibrio. Per questo motivo l'emergenza può verificarsi solo quando il sistema si trova in uno stato intermedio tra ordine e disordine, chiamato *al confine del caos*, il quale è sempre lontano dall'equilibrio.

1.7 Caratteristiche dei comportamenti collettivi

I comportamenti collettivi possono essere caratterizzati dal modo in cui si organizzano i compiti all'interno del sistema. In questa ottica possiamo suddividerli in:

1. coordinazione
2. cooperazione
3. deliberazione
4. collaborazione

1.7.1 Coordination

La coordinazione è intesa come l'organizzazione, in spazio e tempo, necessaria ai vari task che cooperano per risolvere il problema. Questa funzione porta a una specifica distribuzione spazio-temporale degli individui, delle loro attività, e dei risultati ottenuti da ogni singola attività. La coordinazione è facilmente osservabile nel comportamento

delle formiche, le quali creando una rete di tracce olfattive, organizzano e ottimizzano la loro ricerca alimentare nello spazio tra il nido e la fonte di cibo stessa.

1.7.2 Cooperation

La cooperazione si manifesta quando degli individui realizzano insieme un compito che singolarmente non sarebbero in grado di svolgere. Essi devono combinare i loro sforzi per affrontare un problema comune, la cui complessità andrebbe aldilà delle loro singole capacità. Questa caratteristica si riscontra, per esempio, negli spostamenti di grossi blocchi materiali(robotica), e, in tutti quei casi in cui, un singolo agente risulti troppo debole per eseguire determinate operazioni senza l'aiuto di altri simili.

1.7.3 Deliberation

La deliberazione si presenta nei casi in cui i componenti di un sistema si trovino di fronte a varie opportunità, e si manifesta con una scelta collettiva di una di queste. Spesso la deliberazione è guidata dalla competizione che si instaura tra le risorse condivise, che corrispondono alle varie opportunità.

1.7.4 Collaboration

Per collaborazione si intende la possibilità di portare avanti attività differenti in modo simultaneo, tramite dei gruppi specializzati. Questa specializzazione, interna al sistema, può esser dovuta sia a una semplice differenziazione comportamentale, che a una sostanziale differenza morfologica, e può essere influenzata nel tempo da vari fattori.

Capitolo 2

Self-Organization nei MAS

2.1 Introduzione al capitolo

Molte proprietà osservate nei sistemi sel-organizing, quali soprattutto la robustessa e l'adattabilità ai cambiamenti, hanno condotto a numerose ricerche in diverse aree, nelle quali si sono utilizzati appunto sistemi auto-organizzati per risolvere problemi complessi. Abbiamo inoltre visto come l'auto-organizzazione e l'emergenza necessitino di:

1. Autonomia e incapsulamento del controllo
2. Percezioni e azioni locali
3. Ambienti distribuiti
4. Supporto all'organizzazione e alla cooperazione

E' evidente come il paradigma ad agenti si presenti ideale per rappresentare,simulare,e sviluppare sistemi self-organizing (da ora in poi abbreviati in SOS). Non esiste ancora un framework o una metodologia generale per lo sviluppo di SOS. Questa è una lacuna pesante, poichè è fondamentale per un ingegnere software avere metodi,tools e architetture standard sui quali modellare i propri sistemi. Come già visto precedentemente i meccanismi di auto-organizzazione possono aver luogo in seguito a diversi fattori, e per questo sono stati studiati vari approcci a seconda del principio teorico su cui si basano. In questo capitolo presenteremo meglio questi diversi approcci, evidenziandone caratteristiche, vantaggi e svantaggi.

2.2 Le 5 categorie

I ricercatori hanno sperimentato vari meccanismi per gestire l'auto-organizzazione e i fenomeni emergenti su differenti applicazioni. I differenti approcci si dividono in 5 classi, a seconda del meccanismo sul quale si basano:

1. Interazioni dirette tra agenti
2. Interazioni indirette e stigmergia
3. Rinforzo (feedback positivo) dei comportamenti
4. Cooperazione tra gli agenti
5. Scelta di un architettura generica

2.2.1 Meccanismo basato su interazioni dirette

Zambonelli nei suoi articoli(es:[1]) ha discusso differenti modi per ingegnerizzare l'auto-organizzazione. L'approccio proposto consiste nell'usare alcuni principi di base, come la localizzazione, l'invio broadcast, e le interazioni locali svolte dagli agenti al fine di fornire uno stato finale globale coerente. A differenza degli algoritmi classici, questi algoritmi dovranno assicurare di mantenere un desiderato stato stabile anche in caso di cambiamenti ambientali. Tipici esempi sono i meccanismi applicati nell'area del self-assembly e e della self-localization, dove oggetti mobili tramite interazioni locali e piani condivisi nell'ambiente danno origine a pattern spaziali.

2.2.2 Meccanismi basati sulla stigmergia

L'auto-organizzazione in questo caso emerge dall'interazioni indirette tra agenti, che porta a un comportamento globale coerente del sistema. Questo principio viene anche usato per ottenere la formazione di pattern non simmetrici nelle applicazioni self-assembly. Questi meccanismi possono essere valutati tramite sperimentazione, per esempio con delle fasi di simulazione e di prototipazione. In genere si integrano gli esperimenti simulativi dentro la metodologia per ingegnerizzare

questi sistemi. A causa della non linearità e della complessità dei fenomeni presenti i questi sistemi, non è possibile avere un controllo diretto del loro comportamento, ma si può solamente ottenere alcuni dati statistici(tramite esperimenti) sulla convergenza del sistema verso il comportamento globale desiderato.

2.2.3 Meccanismi basati sul rinforzo

La self-organization in questo caso si basa sulla capacità degli agenti di modificare dinamicamente i propri comportamenti in relazione ad alcuni rinforzi(feedback). Tale tecnica è legata a due principi di base: premiare o punire il comportamento di un agente. Come conseguenza ogni singolo agente può adattare le sue azioni e mostrare nel sistema una specificazione dei ruoli. In questi meccanismi gli individui dinamicamente scelgono un nuovo comportamento futuro legato ad una probabilità interna calcolata sulla base del suo stato attuale e di quello dell'ambiente circostante. Un tipico esempio è l'approccio descritto in questo articolo [9].

2.2.4 Meccanismi basati sulla cooperazione

Il framework dell'organizzazione Self-design(OSD) usa delle primitive per la composizione e decomposizione di agenti. Allo stato iniziale avremo un unico agente che gestisce tutto; dopo di che il sistema prova a rendersi cooperativo con l'ambiente tramite 2 operazioni:

1. **composizione:** 2 agenti diventano 1
2. **decomposizione:** 1 agente viene suddiviso in 2

Queste operazioni vengono alternate allo scopo di far adattare il sistema alle richieste dell'ambiente dinamico. (il tutto viene testato in fase di simulazione) E' importante notare come nessun agente ha una visione globale del sistema e non sia presente alcun controllo centralizzato. Ogni agente possiede l'abilità di auto-organizzazione, per esempio la capacità di adattare le sue interazioni con l'ambiente e gli altri agenti, a seconda della sua conoscenza individuale. Questo consente di effettuare delle modifiche dinamiche al sistema senza ricodificarlo

espressamente a livello globale. (tutti i cambiamenti avvengono al micro livello).

2.2.5 Meccanismi basati su architetture generiche

Una classe di meccanismi di auto-organizzazione sono basati su architetture o meta-modelli generici.

2.2.6 Pregi e difetti

I meccanismi basati sulla comunicazione diretta hanno il vantaggio di modellare comportamenti auto-organizzati con conoscenze sicure. Questo approccio dunque garantisce un forte controllo, ma purtroppo è applicabile solo per un numero limitato di applicazioni. I meccanismi basati sulla stigmergia non danno le stesse garanzie di quelli diretti, ma possiedono, a loro volta, altri grandi vantaggi. Per prima cosa sono riusabili, in quanto sono ispirati a fenomeni biologici esistenti e già studiati; inoltre in questi sistemi la simulazione può essere usata come parte integrante dell'implementazione, riducendo cosi i tempi di sviluppo. Un altro aspetto da tener conto è che,essendo i comportamenti individuali molto semplici, anche i costi di sviluppo sono bassi. I meccanismi basati sulla cooperazione consentono di trattare applicazioni che presentano comportamenti globali sia continui che discontinui. Lo svilupo del sistema risultante è robusto in quanto adattativo. Però possono esserci casi in cui è richiesto l'emergere di stati globali specifici come la posizione di un giocatore robot in un campo di calcio, e allora le molte possibili soluzioni offerte da questi meccanismi possono diventare un problema. Per questa ragione tali meccanismi vengono in genere usati quando, in seguito all'emergere di una soluzione globale, è necessario mantenerla stabile tramite auto-organizzazione senza la necessità di convergere ad altre soluzioni.

2.3 Problemi nell'introdurre self-organization nei MAS

Il problema centrale che un ingegnere si trova di pronte quando vuole progettare un SOS è ormai chiaro:

> Come programmare singoli agenti in modo che, nel loro insieme, si auto-organizzino?

L'ingegnerizzazione delle applicazioni self-organizing necessitano di strumenti per definire un goal globale, e per progettare i comportamenti locali che portano all'emergere del del goal stesso. Questo è un compito non facile, poichè il goal del sistema non è predicibile come funzione dei goal locali. Conseguentemente la fase di verifica diventa molto ardua se non realizzata tramite simulazione. Le metodologie software tradizionali sono insufficenti, poichè sono basate su interfacce statiche (già definite a design time) o su ontologie preesistenti. Queste tecniche rendono possibile definire un comportamento globale solo se esso è funzione delle varie parti del sistema(caso non compatibile con i SOS). Gli studi ingegneristici attuali, direttamente legati alla self-organization, consistono nel progettare algoritmi distribuiti ispirandosi a meccanismi naturali o biologici. Più recentemente stanno prendendo piede meccanismi non ispirati a fenomeni biologici, e si stanno sviluppando middleware a doc per sviluppare queste applicazioni self-organizing. Tuttavia, la fase di verifica e il metodo completo di ingegnerizzazione restano ancora problemi aperti. E' importante capire il vero significato di 'controllare' l'emergenza, per poter sfruttare tale controllo al fine di risolvere problemi complessi. Quando progettiamo un sistema artificiale, è necessario avere operazioni e tool che consentino a questi sistemi di produrre il fenomeno emergente desiderato. Inoltre l'ambiente gioca un ruolo fondamentale sia come media di coordinazione che come fonte di imprevedibilità per gli agenti. Il ruolo dell'ambiente in un SOS e la sua ingegnerizzazione, devono essere ben compresi per poter sviluppare il sistema. Le ricerche attualmente stanno dando vita a nuovi principi, teorie modelli, meccanismi e metodologie per l'ingegneria dei sistemi self-organizing con o senza fenomeni emergenti. E' importante esser consapevoli delle differenze, e distinguere soluzioni che si occupano solo di problemi di

auto-organizzazione, quelle che si occupano solo di fenomeni emergenti, e quelle che coinvolgono entrambi. Nel prossimo capitolo osserveremo alcune metodologie e architetture proposte per sviluppare sistemi multi-agenti auto-organizzati.

Capitolo 3

Un viaggio tra metodologie e architetture SOS

3.1 Descrizione capitolo

Gli sviluppatori di applicazioni self-organizing devono far fronte a un problema fondamentale, cioè quello di progettare entità a micro-livello, in modo da far emergere le proprietà desiderate dal sistema a livello globale(livello macro). In questo capitolo mostreremo diverse metodologie e architetture proposte per lo sviluppo di SOS. Alcune di esse saranno più generali, altre specifiche per alcuni casi studio, ma tutte mostreranno un processo di sviluppo molto diverso da quello suggerito dalle classiche metodologie MAS, in quanto i fenomeni emergenti e auto-organizzati si inseriscono in varie fasi di progetto, incidento in fase di analisi, di simulazione, e di verifica finale. Vogliamo far notare come sia importante comprendere da subito i vantaggi e svantaggi che si ottengono nell'implementare un SOS, in quanto la decisione di aggiungere auto-organizzazione in un sistema ha un impatto fortissimo sia nel processo di sviluppo sia nella struttura(e i costi) dell'applicazione finale.

3.2 Una metodologia generale

In questo paragrafo forniremo una metodologia adatta a costruire e controllare sistemi complessi e self-organazing. La metodologia proposta non fornisce soluzioni dirette ai problemi, ma solo un framework concettuale per guidare la ricerca della soluzione a tali problemi. Non si suggerisce una nuova soluzione pratica, ma un modo alternativo di approcciarsi a tali problemi. Gli standard e le ontologie sono necessarie (altrimenti ci sarebbe enorme caos tra gli ingegneri), però sono sviluppatre su concetti statici e non si adattano alla dinamicità dei requisiti futuri. **L'idea chiave è quella di inserire la possibilità di 'adattarsi' ai cambiamenti e alle necessità direttamente dentro la fase di sviluppo.**

3.2.1 il framework concettuale

Gli elementi di un sistema complesso interagiscono tra loro e le azioni di un elemento avranno effetti (diretti o indiretti) sugli altri elementi. Queste interazioni possono avere effetti negativi,positivi o neutri sul sistema. Ogni elemento e ogni sistema possono essere visti come agenti con goal e comportamenti mirati al raggiungimento di tali goal. Il completamento di un goal da parte di un agente può essere rappresentato tramite una variabile $\sigma_i \epsilon [0, 1]$. La soddisfazione totale del sistema sarà una funzione dei singoli o degli n agenti in relazione ai relativi pesi W_i, cioè all'importanza che hanno nel sistema. Secondo un approccio riduttivo, aumentando la qualità degli elementi del sistema aumenta anche la qualità globale del sistema. Ma è facile comprendere questo approccio sia errato nella maggior parte dei sistemi complessi, cioè in quei sistemi in cui un elemento può avvantaggiarsi o competere con altri elementi, e dove sarà quindi necessario soffermarsi sulle singole interazioni che avvengono tra i vari componenti del sistema stesso. Dopo avere inquadrato il problema viene spontaneo farsi alcune domande: Come posso determinare la funzione f(fitness) e i vari pesi W_i? La funzione f è molto difficile da ottenere, ma potremo accontentarci di una sua approssimazione. In alternativa potremmo pensare di togliere 1 ad 1 gli elementi dal sistema per osservare l'effetto ottenuto sul sistema, riusciendo cosi a ricavarmi i

pesi Wi tramite i quali ricostruire una f accettabile. Come posso massimizzare la qualità del sistema σ_{sys}? Anche questa operazione non è facile. L'idea sarebbe quella di minimizzare le interazioni negative e promuovere quelle positive. Però è impensabile dire a ogni singolo elemento cosa fare e come farlo. Posso solo vincolarne o modificarne il comportamento in modo da favorire il raggiungimento del suo goal danneggiando il meno possibile i goal degli altri elementi. Questi vincoli possono essere visti come dei **mediator**. I mediator sono determinati da un osservatore, interno o esterno al sistema. Un esempio di mediator possiamo osservarlo nel ruolo che svolgono i semafori e i segnali stradali nel traffico cittadino. Il ruolo del mediator è proprio quello di arbitrare tra i vari elementi di un sistema, di minimizzare i conflitti,le interferenze e le frizioni, e di massimizzare la cooperazione e la sinergia. Col termine **frizione**(friction) intendiamo l'evento che occorre qualora l'aumento di qualità di un elemento porta a una diminuzione della qualità globale del sistema. Una frizione negativa naturalmente implica una sinergia, cioè una qualità crescente sia per il singolo individuo che per il sistema in cui vive. Dopo avere introdotto i mediator possiamo porci il problema di massimizzare la qualità del sistema in questi termini: Come possiamo trovare mediator efficenti per un dato sistema? Il tentativo di rispondere a questa domanda è proprio l'obbiettivo della metodologia proposta. Il sistema è in continua evoluzione; di conseguenza anche la funzione di qualità e i vari goal saranno dinamici. La metodologia cerca di migliorare il controllo e la conoscenza futura del sistema L'obbiettivo è quello di identificare e diminuire i conflitti, garantendo l'aumento di σ_{sys}. E' molto importante in questo contesto la scelta della scala temporale, in quanto spesso sono necessari sacrifici nel breve periodo per garantire vantaggi a lungo termine. Il sistema dovrà essere osservato a 2 livelli; sia a quello locale che a quello globale, e cercando di comprendere come i comportamenti e i cambiamenti individuali possano incidere su quelli globali.(cosa molto difficile per i problemi complessi) Possiamo anche introdurre la **dipendenza** d di un elemento dal sistema, definendola come differenza tra la qualità dell'elemento all'interno del sistema e la qualità nel caso sia isolato. Se la dipendenza $d < 0$ significa che l'elemento si trova meglio fuori dal sistema. Possiamo usare la dipendenza appena introdotta per misurare l'integrazione $\tau \epsilon [-1, 1]$ di un sistema,

cioè una media tra le dipendenze degli elementi che lo compongono.

$$\tau = \frac{1}{n} * \sum d_i$$

Sfruttando tutti i concetti fin'ora introdotti, presenteremo ora i passi da seguire per uno sviluppo coerente di un sistema auto-organizzato.

3.2.2 La metodologia

La metodologia proposta contiene i seguenti step di sviluppo:

1. Rappresentazione
2. Modellazione
3. Simulazione
4. Applicazione
5. Valutazione

Tutti i livelli sono collegati con frecce bidirezionali, in quanto il progettista in ogni momento può aver la necessità di tornare ad uno step già affrontato, per fare delle riconsiderazioni.

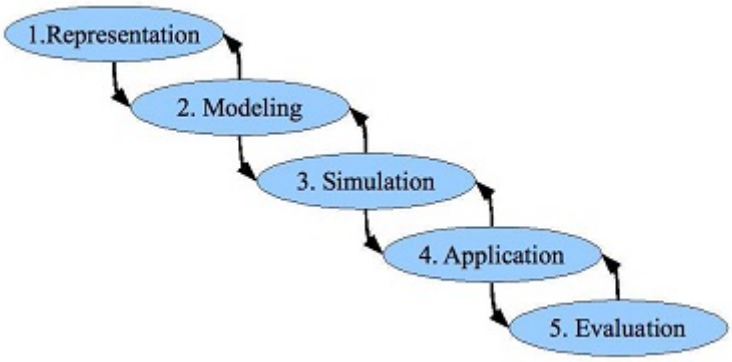


Figura 3.1: step principali della metodologia

Rappresentazione

L'obbiettivo di questo livello è sviluppare una 'specificazione' dei componenti del sistema. Ci sono diverse possibili rappresentazioni di un sistema. Il progettista sceglierà il vocabolario, il livello d'astrazione e la granularità che ritiene più opportuna a seconda della sua esperienza e dei benefici che tale approccio può fornire al dato sistema. Inoltre se possibile,dovrà suddividere il sistema in elementi identificando dei moduli il più possibile indipendenti, con goal e una chiara dinamica interna, e con poche interazioni con l'ambiente. Dividendo il sistema in moduli otterremo dei task indipendenti che potranno essere eseguiti contemporaneamente, e inoltre avremo un aumento di robustezza e adattabilità del sistema globale. A livello di rappresentazione verranno considerati minimo 2 livelli di astrazione(uno per gli elementi e uno per l'ambiente), ma nei sistemi più complessi possiamo averne molti di più.

Modellazione

I modelli idealmente dovrebbero essere molto semplici e chiari, e cercare di modellare il massimo numero di informazioni possibili. Un buon modello richiede una buona rappresentazione. La modellazione fornisce un meccanismo di controllo che assicura che il sistema adempie agli scopi per cui era stato pensato. Questo controllo, nel caso di sistemi auto-organizzati, dovrà essere interno e distribuito. Il meccanismo di controllo può essere visto come un mediator. Nei SOS purtroppo non possiamo avere un contro preciso, ma il sistema dovrà essere guidato per garantire continuamente i goal richiesti. Per sviluppare un controllo il deigner dovrà cercare un modo per minimizzare le frizioni, potendo cosi massimizzare la qualità globale del sistema. Il meccanismo di controllo dovrà essere 'adaptive', cioè riuscire ad adattarsi ai cambiamenti interni ed esterni al sistema, e 'active', cioè sempre attivo nella ricerca delle soluzioni(quindi non un controllo statico). In generale Il Controllo dovrà esplorare differenti alternative per provare costantemente ad aumentare la qualità del sistema minimizzando le frizioni e massimizzando la sinergia. Questa esplorazione deve essere svolta continuamente, poichè la dinamicità dell'ambiente dei SOS produce sempre nuove soluzioni. Sempre per migliorare la qualità del

sistema potremmo pensare di promuovere la cooperazione e la coordinazione tra gli elementi del sistema.Un buon controllo per garantire la coordinazione deve affrontare 2 aspetti importanti:

1. divisione del lavoro (gli agenti specializzati sono più performanti di quelli generici)

2. workflow (si deve minimizzare i tempi morti tra i vari task)

Simulazione

Lo scopo di questo livello è costurire una simulazione software che implementi il modello sviluppato nello stadio precedente, e testi differenti strategie dei mediator e diversi scenari. Questo è un livello fondamentale, in quanto il comportamento di un sistema complesso non può essere completamente dedotto dal modello; di conseguenza un modello verrà simulato prima ancora di essere completamente capito. Lo sviluppo della simulazione avviene in due fasi:

1. Si usa uno scenario astratto per testare i concetti principali sviluppati nel modello.

2. Solo dopo aver testato e approvato tali concetti, si aggiungono i dettagli alla simulazione.

Durante la simulazione bisognerebbe comparare le soluzioni trovate con gli approcci tradizionali, per accertarsi che l'integrazione della self-organization al sistema porti dei benefici reali. Il progettista dovrebbe inoltre sviluppare piu di un controllo da testare in fase di simulazione. La simulazione deve essere robusta prima di tantare un implementazione del sistema nel mondo reale.

Applicazione

Il ruolo di questo livello è semplicemente quello di usare nel mondo reale il modello sviluppato e testato precedentemente. Se stiamo trattando un sistema software, questo passaggio non sarà particolarmente difficile in quanto sarà gia stato sviluppato in fase di simulazione, mentre in caso contrario dovremo affrontare tutti i disturbi presenti nel mondo reale e non affrontati nel modello semplificato(Robotica). La

fattibilità dell'applicazione deve essere tenuta in considerazione durante tutto il processo di sviluppo, per evitare che una complessità o un costo elevato rendino inutili tutti gli sforzi dell'ingegniere. Un altro aspetto da non sottovalutare è quello dei legacy dei sistemi precedenti, in quanto il progettista sarà fortemente limitato dalle capacità dei vecchi sistemi.

Valutazione

Una volta terminata l'applicazione, le performance del nuovo sistema dovranno essere misurate e comparate con quelle del vecchio sistema, o con quelle desiderate all'avvio del progetto. Sotto possiamo osservare un grafico riassuntivo dei vari sottostep descritti.

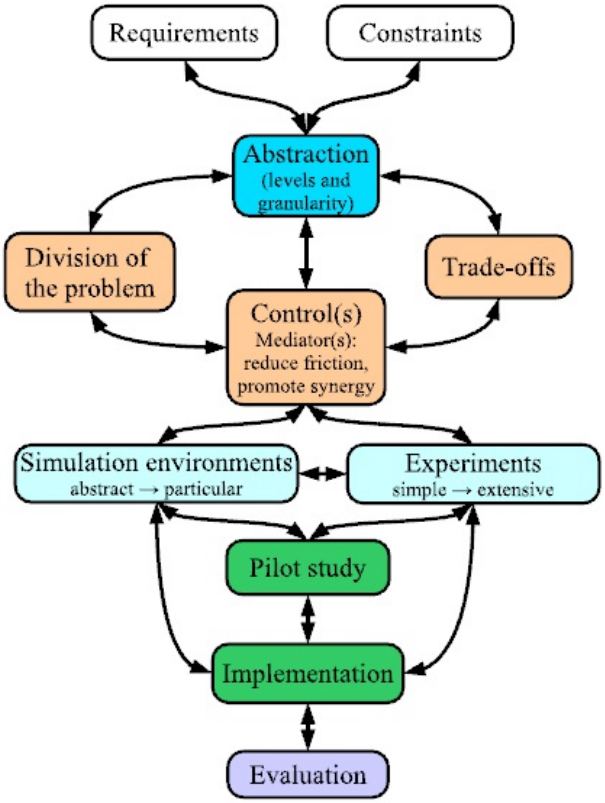


Figura 3.2: sottostep della metodologia

3.2.3 Conclusioni

Possiamo osservare dai diagrammi 3.1 e 3.2 che tutte le frecce sono bidirezionali, in quanto non è possibile prevedere a priori i risultati delle strategie implementate. Le proprietà emergenti, infatti, sono osservabili solo a posteriori. Questa metodologia, in principio, sarebbe applicabile a qualsiasi sistema, ma risulterebbe molto ridondante nei sistemi semplici o che avvengono in ambienti statici e predicibili (non è richiesta alcuna funzione adattiva). E' importante evidenziare come non sia strettamente necessario 'capire' una soluzione prima ancora che sia testata. In molti casi la comprensione è possibile solo dopo la fase di testing. Questa metodologia non è ne bottom-up ne top-down, o meglio è un pò entrambe. Sono necessari contemporaneamente sia un altro che un basso livello d'astrazione, quindi può essere vista come un metodologia con approccio multi-livello. La metodologia proposta non presenta risultati concreti, ma fornisce idee e strade che possono essere approfondite per produrli.

3.3 Architettura basata su stati emozionali

3.3.1 Teoria di base

In questo paragrafo proponiamo proponiamo un'architettura generica ibrida, nella quale gli operatori sono in grado di avere reazioni reattive o cognitive e di generare collettivamente un comportamento emergente, con l'obiettivo di migliorare la prestazione del sistema. Inoltre, considerando la cognizione come un fenomeno sociale, un operatore potrebbe sviluppare un comportamento individuale basato sul comportamento collettivo dei suoi vicini. L'architettura aggiunge la caratterizzazione dello stato emozionale degli operatori. Definiamo **architettura cognitiva** quel modello computazionale utile per lo studio del comportamento e la cognizione a livello individuale. Fornisce all'agente dei meccanismi 'decisionali'. Tra le varie architetture cognitive esistenti la più completa è SOAR([6]), la quale comprende una memoria di lavoro a lungo termine e meccanismi di apprendimento (es: rinforzi).

3.3.2 Descrizione architettura

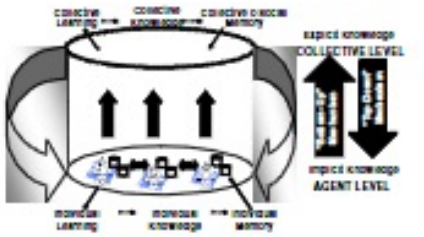


Figura 3.3: Tipi di apprendimento e conoscenza nel SOS

La 3.3 mostra come il processo di apprendimento ed acquisizione della conoscenza risieda completamente nell'architettura. Un agente aumenta la propria conoscenza attraverso un processo di apprendimento individuale, interagisce (socializza) con l'ambiente e con gli altri agenti usando l'informazione locale. Un meccanismo 'bottom-up' consente l'emersione della conoscenza collettiva esplicita, mentre un processo di feedback 'Top-down' promuove l'apprendimento individuale di questa nuova conoscenza collettiva emersa. Si crea cosi un processo circolare di causa-effetto della gestione della conoscenza generale, che si basa su:

- **socializzazione:** consiste nella condivisione delle esperienze attraverso le interazioni locali e richiede la traduzione di conoscenza implicita in concetti espliciti.
- **aggregazione:** l'agente crea conoscenza affidabile esplicita attraverso lo scambio di punti di vista, incontri, ecc.
- **appropriazione:** consiste nella traduzione della conoscenza esplicita all'interno di quella implicita.

architettura multiagente per sistemi auto-organizzati

L'architettura consente una coordinazione emergente tra gli agenti ibridi. È divisa in due livelli: individuale e collettivo (3.4). L'emersione collettiva cognitiva sorge da tre livelli di interazione:

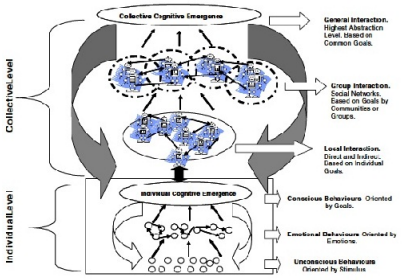


Figura 3.4: Tipi di apprendimento e conoscenza nel SOS

1. Livello di interazione locale che potrebbe essere diretto o indiretto (attraverso l'ambiente).
2. Livello di interazione di gruppo che coinvolge una rete sociale o un gruppo strutturato.
3. Livello di interazione generale che include l'intero set di agenti.

L'emersione individuale cognitiva, invece, si genera imitando il modo in cui agiscono i neuroni, quando generano una gamma di comportamenti, dall'inconscio al conscio. Ispirati da questi, il comportamento dell'agente è modellato su tre livelli, ognuno attivato o inibito a seconda dell'obiettivo dell'agente stesso:

1. comportamento inconscio o reattivo;
2. comportamento emotivo;
3. comportamento conscio.

Componenti dell'architettura

A livello collettivo l'architettura è composta da:

- **agente ibrido:** questo agente reagisce e ragiona a seconda della percezione, delle emozioni e dello stato dell'ambiente.
- **Meccanismi di feedback:** l'interazione tra i componenti può essere positiva (per promuovere la creazione di strutture e cambiamenti nel sistema) o negativa (per compensare il feedback positivo e contribuire a stabilizzare lo schema collettivo).

- **Campo medio:** rappresenta l'area circoscritta dagli agenti dentro l'ambiente per la coordinazione del comportamento.

Per quanto riguarda il livello individuale,l'architettura è costituita da quattro componenti: reattivo, cognitivo, comportamentale e sociale. Al fine di sfruttare la diversità e per favorire lo sviluppo di un emersione collettiva cognitiva, ogni agente può avere un comportamento ibrido: reattivo, emozionale - reattivo e cognitivo - reattivo, tra gli altri.

- **Componente reattivo:** si occupa del comportamento reattivo dell'agente. Il **Selettore delle reazioni** seleziona tra le diverse abitudini comportamentali quella che deve essere eseguita, secondo lo stato emotivo dell'agente stesso.

- **Componente comportamentale:** favorisce l'adattamento dell'agente al suo ambiente creando un modello interno per il mondo esterno. Ogni decisione dell'agente sarà basata sui suoi obiettivi individuali e collettivi, il suo stato emotivo e la conoscenza acquisita, individuale e collettiva. Tra i vari componenti in questa categoria abbiamo il **Configuratore emozionale** che manipola le emozioni dell'agente e il **Manager comportamentale** che suggerisce un tipo di comportamento basandosi sullo stato emotivo, gli obiettivi, la situazione dei vicini e dell'ambiente. La conoscenza, associata con l'emozione dell'agente, è memorizzata nel nella **Behavior KB**. Il ruolo delle emozioni è quello di suggerire il comportamento preferito dall'agente. Un agente può avere emozioni in tre aree differenti:

  1. **goal-based:** gioia, angoscia, speranza, paura;
  2. **basate sulle azioni degli altri:**rabbia, gratitudine, rimorso,ammirazione, disonore;
  3. **basate su gusti/atteggiamenti:** Amore, odio, piacere, antipatia)

  Inoltre ogni agente può avere un atteggiamento positivo, negativo o neutrale che influirà sull'intensità dell'emozione.

- **Componente cognitiva:** è responsabile della produzione del comportamento cognitivo degli agenti. E' formato dal **Configuratore obiettivi individuali** che crea la configurazione individuale degli obiettivi e delle priorità degli agenti e dal**Deliberator**, il quale è responsabile dei meccanismi cognitivi (apprendimento, ragionamento). La conoscenza generata è memorizzata nella **cognitive KB**.

- **Componente sociale:** promuove la coscienza tra gli agenti sul lavoro e l'esperienza degli altri. Specificamente, trae vantaggio dalle esperienze altrui (apprendimento sociale), al fine di evitare un lungo processo di apprendimento individuale di cose già ben apprese dai propri vicini. E' formato dal **Configuratore obiettivi collettivi** che configura le priorità e gli obiettivi collettivi degli agenti, e dal **Ragionatore sociale** che seleziona quale azione imitare e da quale agente sia basata negli obiettivi collettivi. La conoscenza delle decisioni prese dai vicini di un agente è memorizzata nella **social KB**.

- **Altri elementi generici:**

  - **Input System:** dota gli agenti di informazioni nel mondo in cui vivono. Questo sistema passa percezioni (input) in maniera parallela alle componenti: reattiva, comportamentale, cognitiva e sociale. Tutte interagiscono con quell'input, ma è il comportamento del componente stesso che deve stabilire chi abbia la maggior priorità per poter rispondere.
  - **Azioni:** sono regole di condizione-azione (if-then) o generate da un processo deliberativo.
  - **Output System:** sceglie l'azione indicata dal manager comportamentale nel caso in cui ci siano diverse risposte.

3.3.3 Conclusioni

Ricapitolando, l'architettura proposta è divisa in due livelli, uno individuale ed uno collettivo, e cerca di modellare i SOS generando conoscenza collettiva emersa dalle interazioni locali interne. E' un'architettura collettiva basata su tre fasi per la gestione della conoscenza, e

associa ad ogni singolo agente uno stato emotivo. Questo ci permette di avere un'architettura MAS in cui il comportamento dell'agente potrebbe rappresentare al meglio un'ampia gamma di situazioni umane. In conclusione potremmo dire che questa architettura offre un modello alternativo per rappresentare e meglio capire i processi self-organizing ed emergenti negli ambienti umani. Ulteriore lavoro in corso include la modellazione di sistemi quali Wikipedia, sviluppo di software gratuito e il comportamento collettivo dei pedoni.

3.4 Approccio basato su goal dinamici e specificazione ruoli

3.4.1 introduzione

Abbiamo già spiegato il perchè i MAS siano largamente utilizzati per modellare sistemi self-organizing, e come i meccanismi auto-organizzati possano emergere in vari modi. Per esempio dalle interazioni indirette tra gli agenti(approccio stigmergico) o da un comportamento cooperativo basato su interazioni locali, allo scopo di raggiungere un goal globale comune (approccio cooperativo). In questo paragrafo vedremo un approccio alternativo basato su goal dinamici, nel quale gli individui del sistema mostreranno una sorprendente abilità nell'adattarsi ai cambiamenti ambientali, e saranno in grado di valutare le varie opzioni possibili e scegliere la più performante. Con questa visione, i goal vengono associati a costrutti mentali con un proprio ciclo di vita. In questo modo possiamo associare all'agente uno stato motivazionale del goal, cioè rappresentare il livello di desiderio con cui l'agente tenta di raggiungere ogni singolo goal. Inoltre questo approccio fornisce un meccanismo che permette agli agenti di adattare i propri comportamenti dinamicamente, in base alle esperienze passate(ci sarà una memoria che tiene traccia degli adattamenti emersi nel tempo dalle interazioni locali). Questo meccanismo permette agli agenti di avere diversi *ruoli* durante il proprio ciclo di vita. Sarà dunque necessario definire un set di ruoli possibili e le varie condizioni che portano un agente a passare un ruolo ad un altro. Con questo approccio vengono modellate direttamente solo le proprietà a livello

micro del sistema, mentre i comportamenti globali auto-organizzati vengono lasciati emergere(come ovvio!). Gli individui dovranno essere modellati come agenti goal-oriented (le azioni interne sono regolate dai goal). Per fare questo adotteremo la metodologia INGENIAS, con relativi adattamenti per consentire l'inserimento di nuovi concetti (stati mentali,emozioni,ecc).

3.4.2 descrizione modello

Introduciamo ora il modello self-organizing proposto. L'auto-organizzazione in questo modello è associata all'abilità degli agenti di cambiare dinamicamente il proprio comportamento in relazione a meccanismi di rinforzo. Questo rinforzo deriva da feedback(positivi o negativi) che sono parte del sistema auto-organizzato stesso. Un esperienza passata positiva incentiva quel determinato comportamento dell'agente, mentre un esperienza negativa provoca l'effetto contrario. Invece di utilizzare una politica per le azioni (scelta di un azione in relazione al feedback), si applica una politica per i comportamenti: gli agenti sono in grado di scegliere dinamicamente un certo ruolo (il quale può implicare anche più task). Emerge cosi dal sistema una *specializzazione dei ruoli.*

modello dell'architettura dell'agente

Il comportamento adattabile visto sopra viene modellato direttamente nell'architettura del singolo agente. E' un architettura basata sui goal che un agente vuole raggiungere, i task che deve compiere per farlo, e il ruolo a cui si trova assegnato. Quindi la decisione di eseguire un dato task o adottare un dato ruolo è strettamente legata al goal che l'agente sta tentando di raggiungere. Si potrebbe pensare che non appena un agente abbandona un determinato goal per attivarne un altro, avvenga un cambiamento dinamico del suo comportamento interno. Tuttavia in questa architettura viene proposta un'altra soluzione che si basa su goal dinamici. Si introduce un particolare *bielef* chiamato **motivation**. Il compito della motivation è quello di rappresentare l'intensità con cui l'agente segua il suo obbiettivo, e in base a tale valore verrà specificarne il ruolo all'interno del sistema. La

variazione di motivation verrà garantita attraverso un feedback legato alle esperienze passate dell'agente stesso. Nella fig. 3.5 possiamo osservare tutti gli elementi del modello auto-organizzato. Possiamo

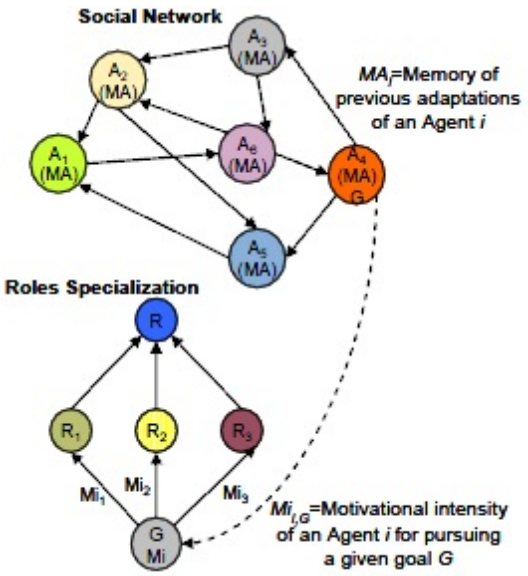


Figura 3.5

quindi affermare che il meccanismo è formato da:

1. Un belief **motivation**(Mi) che lega ai goal un intensità dinamica.

2. Una **selezione dinamica del comportamento** guidata dalla specializzazione dei ruoli basata sulla motivation.

3. L'aggiornamento della motivation attraverso un **meccanismo di rinforzo**(reinforcement mechanism), basato su esperienze passate,stato dell'ambiente e stato corrente degli agenti.

meccanismo di rinforzo

Nel sistema si forma una rete di interazioni tra gli agenti denominata *social network*. Questa rete fornisce agli agenti una *memoria di adattamento*(**MA**). La rete rappresenta l'esperienza e l'impatto(positivo o

negativo) di un agente nei confronti degli altri agenti. La formazione dei ruoli all'interno della rete è specificata dal progettista a design time, tuttavia l'evoluzione delle MA permette un comportamento non statico che si adatta nel tempo. Ad ogni Mi all'agente è associata una diversa strategia (ruolo). L'intensità della motivation Mi dell'agente i legata al goal G al tempo t è definita come funzione della memoria MA(passate esperienze verso lo stesso goal G),dello stato interno S e dello stato dell'ambiente percepito S_e.

$$Mi_{i,G,t} = f_i(MA_{i,G,t}, S_{i,t}, S_{e,t})$$

La funzione f_i può essere più o meno complessa, a seconda del dominio del problema che stiamo modellando. Ora, per rendere il modello adattabile, dobbiamo mappare i possibili comportamenti degli agenti(ruoli) con la corrispondente intensità Mi. Una prima associazione viene fatta a design time, mentre in seguito il tutto sarà legato esclusivamente all'evoluzione del sistema. Possiamo vedere il ruolo R scelto dall'agente come:

$$Ri = f_i(Mi_{i,G,t})$$

3.4.3 implementazione modello

L'obbiettivo del nostro modello è quello di osservare i comportamenti emergenti risultanti dalle interazioni nella *social network*, allo scopo di scoprire e analizzare la presenza di pattern sociali. Per creare il nostro framework implementativo sfruttiamo la metodologia INGENIAS, una metodologia agent-oriented che supporta ottimalmente la specificazione di strutture organizzate e dinamiche(come può essere il comportamento di un agente), e quindi si integra perfettamente con le caratteristiche presenti nei sistemi self-organizing. Il linguaggio del modello(MOF) è strutturato in 5 classi, che corrispondono ai diversi punti di vista in cui può essere osservato il sistema:

1. Organization
2. Agent
3. Goal-tasks

4. Interactions
5. Enviroment

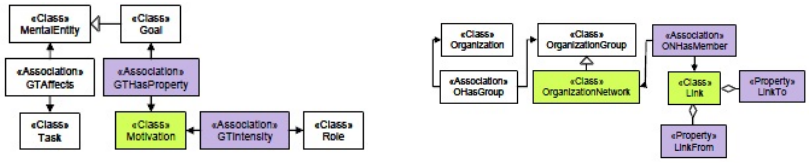
Per ulteriori informazioni su questa metodologia suggeriamo la lettura di questo articolo [7]. Per introdurre i nuovi concetti di adattamento visti precedentemente dobbiamo estendere alcuni parti del linguaggio di modellazione, in particolare quella relativa ai goal e all'organizzazione. Nel meta-modello *goal-tasks* verrà aggiunta l'intensità emozionale Mi come proprietà dell'entità *Goal*. Per farlo creiamo nel modello una nuova classe **Motivation** e due nuove associazioni:

- **GTHasProperty:** indica che il goal ha una proprietà chiamata *Motivation*;
- **GTIntensity:** associazione tra la classe *Motivation* e la classe(già esistente) *Role*.

Anche il meta-modello **Organization** dovrà essere esteso per includere una nuova entità **social network**, una struttura dinamica costituita da nodi (individui). La rete indicherà il modo in cui i nodi saranno dinamicamnete connessi tra loro. Per aggiungere questa nuova entità estenderemo il concetto di *Group*. La classe **organizationNetwork** deriverà da *organizationGroup*, e rappresenterà un gruppo dove gli agenti sono forniti della proprietà *node*, che gli consentirà di mantenere relazioni con altri agenti tramite la proprietà **LinkTo**. Dopo aver aggiornato il linguaggio del modello, non ci resterà che definire il mapping motivation-ruolo, come semplice tabella a due entrate o come funzione di Mi. Ora che abbiamo tutti gli elementi per modellare il sistema, dobbiamo definire un nuovo punto di vista del MAS, che chiameremo **Goal-Roles**. In questa classe rappresenteremo come un agente si incarica di raggiungere un dato goal, e come l'intensità motivazionale è legata al ruolo che l'agente sta svolgendo.

3.4.4 Conclusioni

In questo paragrafo abbiamo discusso un modello per self-organizing MAS, e testato una sua implementazione sfruttando un modeling language per MAS. Per farlo abbiamo sfruttato il linguaggio della metodologia INGENIAS, integrandola in vari punti per poter modellare



(a) Extended Goals-Tasks meta-model.

(b) Extended Organization meta-model.

Figura 3.6

i concetti di auto-organizzazione e di adattamento proposti. Vogliamo far notare come questo approccio sia particolarmente adatto per modellare sistemi di dimensioni ridotte, in quanto, in presenza di numerosi goal o tipi diversi di agenti, la specificazione di tutti i comportamenti interni diventa molto lunga. Si stanno svolgendo studi per poter integrare completamente l'intero meccanismo di adattamento dinamico all'intenro dei modelli di INGENIAS, consentendo cosi di utilizzare entità specifiche per modellare i feedback e la formazione dei ruoli nel sistema.

3.5 aproccio bio-ispirato

3.5.1 Introduzione

I sistemi biologici non solo sono importanti per i MAS, ma ispirano modelli per nuove aree software quali self-adaptation, self-protection,self-healing e meccanismi di coordinazione e cooperazione. Serve un metodo coerente per sviluppare i nuovi concetti, valutare i SOS, sviluppare in modo robusto tali sistemi, e decomporre i sistemi self-organizing in sottoproblemi. In questo paragrafo mostreremo l'utilizzo di un metodo bio-ispirato per modellare sistemi self-organizing da micro a macro livello e progettare comportamenti dinamici. Mostreremo il ruolo fondamentale che avrà l'ambiente durante il processo di sviluppo e come i pattern emergenti di self-organization possono essere osservati e valutati per la verifica del sistema.

3.5.2 Descrizione metodo e modello di rappresentazione

La metodologia proposta è stata pensata sfruttando come caso di studio un applicazione di rete auto-organizzata e autonoma. L'*Autonomic computing* è un area di ricerca in cui le soluzione self-organizing sono molto richieste, in quanto sarebbe bello avere un sistema decentralizzato in cui entità autonome cooperano per mantenere il sistema coerente in tutte le sue parti self-healing,self-configuring,self-protection ecc). In questo articolo vedremo solo una panoramica della tecnica proposta, mentre per un approfondimento più dettagliato si consiglia la lettura [3]. Nella nostra applicazione, sia i servizi forniti che il middleware saranno modellati come entità biologiche, ispirandosi alle api nelle colonie. Un **servizio** può essere visto come un agente sofrware distribuito e autonomo che segue semplici comportamenti biologici (replicazione, migrazione, morte, scambio energia), mentre la piattaforma **middleware** è vista come l'ambiente. Come nei sistemi biologici, anche nel nostro caso gli agenti e l'ambiente salvano e spendono energia per vivere. L'agente guadagna energia quando offre il suo servizio all'utente (o ad altri agenti) mentre la spende quando usa la rete e le risorse computazionali. Ogni piattaforma guadagna energia offrendo risorse agli agenti; energia che in seguito evaporerà nell'ambiente di rete. Una mancanza di energia può causare la morte dell'agente o della piattaforma. Come nei sistemi biologici, l'applicazione di rete mostra caratteristiche emergenti come scalabilità e adattamento alle varie situazioni. Queste proprietà emergono dalle interazioni tra agenti e piattaforme e non sono presenti nel singolo individuo. Agenti e piattaforme sono autonomi e decentralizzati, senza alcun controllo o informazione a livello globale del sistema. Il metodo proposto consiste in tre parti chiave:

1. Un meta-modello per strutturare le entità;

2. Un modello di rappresentazione (UML) per supportare il mapping tra macro e micro scala;

3. Un metodo per supportare la fase di verifica.

Il tutto viene preceduto da una fase di analisi del sistema, in cui bisognerà incorporare le seguenti attività: Ci sono 2 attività principali da incorporare nella fase di analisi:

- Definizione dei comportamenti globali self-org che dovranno essere composti da self-org mechanism pattern.
- Descrizione delle proprietà emergenti desiderate

In fase di analisi il progettista non conosce ancora come le interazioni locali possano dar vita alle proprietà emergenti richieste. Tuttavia, se vogliamo che il SOS converga, e verificare e ottimizzare il sistema con uno svilupo iterativo, queste interazioni locali sono critiche per lo sviluppo finale e devono essere approfondite il più possibile. Dopo l'analisi si passa alla fase di design, nella quale l'analista modella le proprietà emergenti definite nella fase precedente. La modellazione consiste in:

1. Definire le entità strutturali presentati nel meta-modello (mostrato in seguito);
2. Decomporre le proprietà emergenti nei comportamenti locali in legame ai pattern self-org;
3. Definire le soglie relative alle proprietà emergenti.

Tali soglie saranno basate sui vincoli richiesti a livello globale, e diventeranno gli input necessari per la simulazione durante la fase di verifica (in questa fase si richiede solo il riconoscimento dei parametri critici). Nel diagramma 3.7 possiamo vedere riassunte le varie fasi di sviluppo del metodo proposto.

Il meta-modello

Una buona procedura(non l'unica) per sviluppare con successo una tecnologia ad agenti è quella di fornire tool ingegneristici potenti che supportino i metodi industriali di sviluppo. L'obbiettivo del meta-modello proposto è quello di fornire delle fondamenta per l'ingegneria dei software SOS da integrare ai meta-modelli classici già conosciuti.

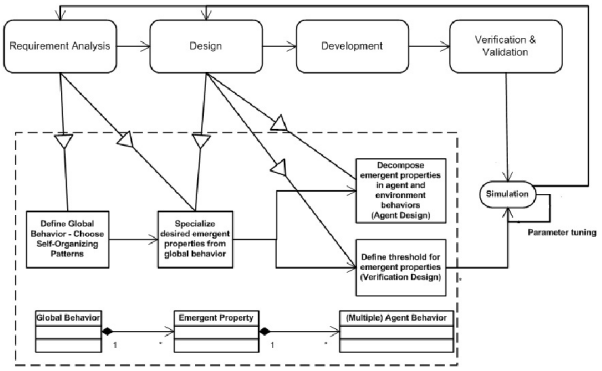


Figura 3.7: descrizione metodo

Questo meta-modello sarà basato sul meta-modello UML2. Esso riuserà tutte le notazioni dell'UML e lo estenderà per poter definire strutturalmente i sistemi SOS. Considereremo l'ambiente come astrazione del 1 ordine (cosi come gli agenti), in quanto fornisce agli agenti le condizioni per vivere, risultando cosi essenziale in tutti i sistemi MAS. Inoltre nei SOS è fondamentale la conoscenza dello stato globale del sistema, quindi risulta indispensabile considerare lo stato ambientale dentro il nostro modello. Nei sistemi SOS l'ambiente agisce autonomamente con comportamenti adattativi (proprio come gli agenti) e interagisce tramite reazioni o eventi di propagazione. Tali eventi sono:

- **emission:** interazione asincrona tra gli agenti e l'ambiente.
- **Trigger:** segnala il cambiamento di stato di un agente.
- **Movement:** segnala un agente in movimento nell'ambiente.
- **Reaction:** interazione sincrona tra gli agenti (con un vicino o con l'ambiente)
- **communication:** segnala un scambio di messaggio tra agenti

In UML una classe può essere attiva o passiva (tramite l'attributo isActive). Poichè un oggetto attivo è differente da un agente, è stata definita una nuova classe **Adaptive** per poter modellare agenti e ambiente.

modello di rappresentazione

Vediamo ora qualche dettaglio sulle nuove classi necessarie per poter rappresentare correttamente tutte le caratteristiche del nostro sistema SOS. Sarà necessario utilizzare sia un modello statico che uno dinamico. Come modello statico viene usato il Class Diagram UML, tramite il quale verranno definite tutte le strutture e le dipendenze statiche del sistema. In tale modello aggiungeremo una meta-classe **Adaptive**, tramite la quale potremo modellare tutte le caratteristiche di un agente software e dell'ambiente. Inoltre avremo due nuove strutture (input e output event) e un nuova caratteristica comportamentale (**action**). Un azione può venir eseguita dall'agente senza alcuna richiesta esplicita da parte di altri agenti, in quanto tali entità sono proattive e non puramente reattive come gli oggetti. Ogni singola azione è descritta specificandone le precondizioni e gli effetti in caso di completamento. Come modello dinamico ci si basa sullo State Machine diagram UML2, combinato con le azioni e le interazioni viste prima. L'obbiettivo è quello di modellare lo stato globale e lo stato dell'ambiente, ricordando che le proprietà macro sono definite da proprietà emergenti che nascono dalle interazioni e dai comportamenti interni al sistema. Lo State Machine diagram viene usato per descrivere tutti gli stati(iniziale,finale,semplice,composto) e le transizioni di una certa entità. E' utile per definire il ciclo di vita di un oggetto, o un ordine d'invocazione delle sue operazioni. Nel nostro caso ogni comportamento dell'agente(o dell'ambiente) è progettato per seguire una macchina a stati comportamentale. Ogni comportamento nel diagramma è in grado di comunicare con tutti gli altri diagrammi attraverso un canale di comunicazione. Il risultato di queste comunicazioni sarà l'apparizione delle proprietà emergenti desiderate. E' bene notare come un 'comportamento' sia formato da azioni, le quali saranno eseguite da input event(con relative precondizioni) e daranno luogo ad output event. I comportamenti andrebbero costruiti in parallelo tra loro, ma purtroppo la composizione dei diagrammi degli stati è sequenziale.

Metodo di verifica

I MAS self-organizing possono essere accettati a livello industriale solo se è possibile garantirne il loro comportamento emergente. Deve essere possibile 'misurare' e verificare l'auto-organizzazione in questi sistemi. Ci sono diverse caratteristiche di un sistema che possono venir monitorate a questo scopo, per esempio:

- Capacità di raggiungere un organizzazione tale da adempiere ai goal richiesti;
- Capacità di riorganizzarsi in seguito ad una perturbazione esterna;
- Livello di decentralizzazione del controllo (cetralizzato,decentralizzato,ibrido)
- Capacità di resistere alle perturbazioni (stabità, adattabilità)

La verifica del comportamento globale consiste nel determinare che il sistema svolga le funzioni richieste. Il problema chiave è come capire se un sistema auto-organizzato mostra il comportamento globale richiesto, considerando che è impensabile modellare tutte i possibili comportamenti interni(nel modello delle interazioni) e quindi dare una prova concreta della correttezza del sistema. Tuttavia, alcuni parametri del sistema saranno predicibili, e potranno dunque essere sfruttati per misurare i fenomeni emergenti. Ci sono diversi approcci per affrontare questo problema; ne vedremo due:

1. **configurazione dei parametri:** In questo approccio([2]) se un agente nel compiere determinate azioni critiche si comporta diversamente dalla media, possiamo decidere di approfondire il controllo sull'agente o negargli l'accesso alle risorse. Per farlo vengono settati alcuni parametri di simulazione (numero iniziale degli agenti, percentuale di agenti normali-anormali, ecc.) Tramite questo approccio siamo in grado di fare assunzioni sulle percentuali di comportamenti malevoli degli agenti, e sul grado di stabilità del sistema durante la sua evoluzione.
2. **verifica tramite equazioni matematiche:** Approccio proposto da DeWolf([10]), basato su algoritmi numerici. Le ricerche

hanno permesso di ottenere numerosi algoritmi di analisi numerica che, basandosi su valide teorie matematiche, supportano l'analisi dei sistemi dinamici e vengono applicati per ottenere modelli basati su equazioni formali. Questa tecnica permette una verifica più accurata dei risultati, se paragonata alla classica tecnica di simulazione e settaggio parametri a posteriori. Gli algoritmi di analisi in pratica guidano il processo simulativo, scegliendo iterativamente di quale simulazione si necessita, e con quali parametri e condizioni iniziali.

L'approccio di DeWolf nel complesso sembra il più appropriato per i nostri scopi. Per questo motivo l'idea è quella di estendere tale tecnica allo scopo di definire un nostro metodo per testare la convergenza del sistema, incorporando nel metodo di verifica dei planner. I **planner online** sono tool ottimali per verificare le proprietà globali dei self-organizing MAS, e indicare come i comportamenti locali perturbano il sistema modificandone il comportamento globale. Utilizzeremo inoltre un **autonomic manager** per verificare il livello di auto-organizzazione del sistema. Esso possiede come requisiti tutti i goal e le proprietà emergenti richieste al sistema. L'autonomic manager è in grado di percepire gli input e gli output event provenienti dagli agenti e dall'ambiente tramite un modulo di monitoraggio. Dopo aver analizzato le proprietà desiderate, si passa a un modulo di pianificazione in cui si studiano i vari step che dovrà affrontare il sistema per poter convergere a un dato stato ottimale durante la fase di esecuzione. Per questo scopo si suggeriscono metodi e algoritmi di ricerca real-time, come per esempio LRTA [5]. Ricapitolando, per poter sfruttare il nostro metodo di convergenza autonoma è necessario:

1. Definire tutte le azioni interne(azioni locali agenti e ambiente) ed esterne(input dall'esterno) del sistema.

2. Definire i goal, cioè le macro proprietà e le proprietà emergenti. I goal devono corrispondere almeno a un subset di stati del sistema.

3. Definire la valutazione dello stato. Si analizzeranno tutti gli stati globali raggiunti e si verificherà quali set di stati corrispondono a quelli di alcuni comportamenti desiderati.

3.5.3 Conclusioni

In questo paragrafo abbiamo proposto un approccio bio-ispirato, costituito da un metodo che permetta la rappresentazione delle proprietà macroscopiche desiderate, mappate all'interno dei comportamenti individuali degli agenti. Il metodo e il modello di rappresentazione vengono usati per ingegnerizzare i MAS che godono di proprietà emergenti e auto-organizzate. Riguardo il modello di rappresentazione, abbiamo suggerito una progettazione multi-livello a partire dalle proprietà macro fino ad arrivare a quelle micro, basandoci su un modello UML modificato (aggiunta di meta-classi per rappresentae agenti e ambiente). Le proprietà emergenti e i pattern di self-organization sono stati rappresentati attraverso le comunicazioni tra gli agenti e l'ambiente. I modelli possono incapsulare le proprietà emergenti a un certo livello di astrazione, ottenendo un buon grado di modularità. In aggiunta, abbiamo definito le interazioni attraverso dei diagrammi comportamentali, in modo da fornire un vista del flusso di controllo tra i nodi a livello micro. Per completare la metodologia ingeneristica, abbiamo introdotto un metodo di convergenza per la verifica delle proprietà del sistema. Questo metodo consiste nell'avere un approccio basato sull'*autonomic computing*. Per farlo, si è sfruttato il metodo di verifica suggerito da DeWolf, integrato tramite l'aggiunta di pianificatori online (online planner).

3.6 Approccio SodekoVS

3.6.1 Introduzione

I sistemi self-organizing tra le loro caratteristiche, hanno quella di mostrare una certa auto-adattabilità, che può consentirci di realizzare software che si controllano autonomamente a run-time senza supervisione esterna. Questo approccio innovativo si propone di considerare sia l'architettura del sistema che la metodologia di sviluppo software, come parti integranti nella costruzione del sistema. Maggiori informazioni su questo progetto possono essere ricercate qui [8]. Seguendo il processo consigliato, le dinamiche auto-organizzanti possono essere integrate dentro le applicazioni tramite moduli compositi, che distri-

buiscono il controllo tra le varie entità del sistema. L'idea di sviluppo viene supportata da un'architettura di riferimento, un modello di programmazione specifico, e una libreria di pattern self-organizing.

3.6.2 Descrizione architettura

Il progetto SodekoVS mira a fornire un processi auto-organizzati come elementi riusabili che lo sviluppatore può integrare facilmente nel suo progetto applicativo. Per farlo verrà fornita un'architettura di riferimento che offrirà un framework concettuale per la configurazione di tali processi. L'architettura è pensata per supportare lo sviluppo di applicazioni self-adaptive, cioè in grado di configurarsi autonomamente senza supervisione esterna. L'idea è quella di fare meno assunzioni possibili sugli elementi che saranno coordinati, in modo da permetterne l'integrazione con strutture specializzate.

Dinamiche self-organizing come componenti Software

Le diverse strategie di coordinazione e framework di simulazione complicano la visione delle dinamiche self-organizing come componenti software riusabili. Per favorire la categorizzazione e la combinazione di queste dinamiche emergenti, viene fornita una libreria di meccanismi di coordinazione, che comprendono un catalogo di pattern(come componenti riusabili) e i relativi problemi per cui sono pensati. Vediamo nel dettaglio l'architettura proposta 3.8. Essa è costituita da 3

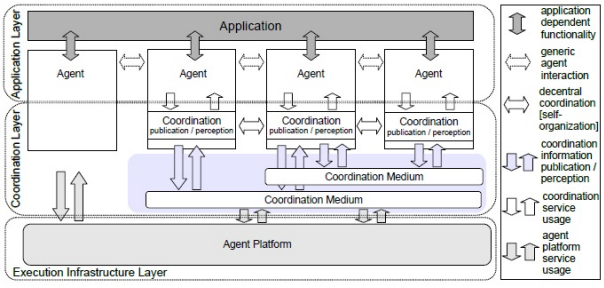


Figura 3.8: Architettura di riferimento

livelli d'astrazione.

- **Application layer:** Contiene le funzionalità standard dell'applicazione. Può avere un collegamento con gli agenti responsabili delle caratteristiche auto-organizzanti, ma solo mostrando altre funzionalità di sistema.

- **Coordination layer:** Consiste in un sottostrato di agenti, che può contenere uno o più *coordination media*. I meccanismi sono incapsulati nei vari *coordination media* e sono interfacciati con componenti che permettono la modifica dello stato interno degli agenti. Tramite questo modello è possibile attuare una strategia di coordinazione semplicemente:

  1. configurare l'informazione interna del componente da modificare;
  2. definire la dinamica del flusso informatico tra i componenti;
  3. Definire in che modo i componenti individuali modificano le loro azioni locali.

- **Infrastructure layer:** Fornisce i servizi base al coordination layer, quali la gestione e l'esecuzione degli agenti.

3.6.3 Descrizione metodologia

Vediamo ora quali sono le fasi di sviluppo necessarie per creare un sistema che integri proprietà di self-organization(3.9):

1. **Requirement:** Fase in cui vengono descritti i comportamenti desiderati dal sistema.

2. **Analysis:** Fase in cui si esaminano le istanze(o combinazioni) di metafore coordinative possono essere sfruttate (pattern).

3. **Design:** Fase in cui si modellano le strategie di coordinazione e si configurano i meccanismi usati per realizzarle.

4. **Implementation:** Fase in cui le dinamiche di coordinazione vengono implementate e integrate nell'applicazione software.

5. **Test:** Fase in cui le attività vengono sottoposte a una validazione simulation-based, la quale controlla se i requisiti richiesti sono stati raggiunti, e il sistema garantisce l'auto-organizzazione richiesta.

L'ostacolo principale che si incontra nel seguire questo percorso di sviluppo, consiste nel creare dei mezzi pratici(tool) per modellare le dinamiche self-organizing e facilitare cosi la fase di Design(scelta strategia, combinazioni tra vari metodi).

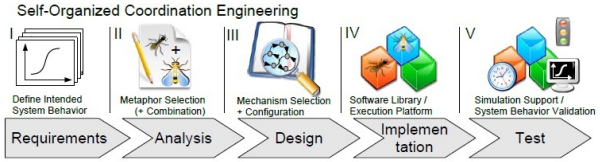


Figura 3.9: Processo di sviluppo della metodologia

Design delle dinamiche tramite strategie di coordinazione

Per facilitare la progettazione delle dinamiche emergenti sarebbe necessario possedere dei formalismi che consentino di allineare i modelli di coordinazione alle applicazioni sviluppate. L'utilizzo sistematico di strategie di coordinazione richiede che le linee guida per la selezione e combinazione durante la fase di Design siano derivate direttamente dalla descrizione delle strategie stesse. Un approccio per collegare i modelli di coordinazione alle entità di design è quello ispirato alla modellazione macro adottata nei concetti System Dynamics(**SD**). SD descrive i sistemi in termini di variabili di stato e relazioni tra esse, permettendo di osservare come gli elementi del sistema si influenzano tra loro, permettendo cosi una più facile descrizione dei loop di feedback interni. Trasferendo queste nozioni di modellazione dentro il design delle nostre applicazioni possiamo descrivere come le attività degli agenti influenzano gli altri individui e le proprietà dell'ambiente. Tramite questo approccio, la capacità di adattamento del sistema viene rappresentata come transizioni tra diversi stati macroscopici del sistema. Ora che abbiamo un metodo per descrivere le proprietà emergenti resta comunque un problema, in quanto scelta delle strategie di

coordinazione (selezione,composizione e coordinazione di meccanismi di coordinazione) non è abbastanza supportata nella letteratura odierna. Una eccezzione è descritta in un articolo di De Wolf([11]), dove i meccanismi di coordinazione vengono associati alle proprietà dei comportamenti macroscopici del sistema che sono in grado di far emergere. In questo modo possiamo facilitare la combinazione delle varie strategie coordinative, ed anticipare (ove possibile) i comportamenti globali tramite trattamenti matematici. Abbiamo cosi dimostrato come la modellazione delle dinamiche dell'applicazione può essere mappata direttamente in fase di design. Naturalmente questa fase di sviluppo è ancora in forte evoluzione e necessita di numerose migliorie per poter diventare sistematica e coerente. Sarebbe utile avere dei tool appositi che consentino una modellazione precisa mantendendo la consistenza a livello applicativo.

Validazione

Per la fase di valutazione dei risultati vengono suggeriti due differenti approcci:

1. **Verifica formale delle dinamiche:** Richiede l'utilizzo di concetti matematici per dimostrare a priori l'emergere di proprietà self-organizing

2. **studi di simulazione sistematica:** E' sempre applicabile e non richiede abilità matematiche particolari, quindi risulta più adatto per lo sviluppo di software.

Le applicazioni self-organizing possono essere validate sia con simulazioni *qualitative* che *quantitative*. Le simulazioni qualitative permettono di riconoscere precocemente se l'applicazione progettata è in grado di esibire i comportamenti globali desiderati, mentre quelle quantitative aiutano a configurare i parametri del sistema per raggiungere le performance ottimali. Gli sviluppatori derivano modelli di simulazione dal design dell'applicazione, esaminano le dinamiche emergenti e, a seconda dei risultati ottenuti, rivalutano la fase di design apportando le modifiche necessarie. Tramite il framework SodekoVS quindi, i progettisti svolgeranno simulazioni per testare se le combinazioni di strategie coordinative scelte sono in grado di garantire i requisiti del sistema.

Qualsiasi strategia coordinativa utilizzata deve essere necessariamente simulata per poter sia validare il comportamento qualitativo, che per regolare i parametri implementativi(simulazione quantitativa). La definizione degli esperimenti e l'interpretazione dei risultati ottenuti richiederebbe un controllo centralizzato che si scontrerebbe ai concetti di decentralizzazione alla base dei sistemi self-organizing. Per ovviare a questo problema il framework SodekoVS si propone di automatizzare questa fase, integrando l'abilità simulativa dentro i servizi di coordinazione offerti dal middleware(vedi figura 3.8).

3.6.4 Conclusioni

Abbiamo descritto un approccio per costruire sistemi self-organizing seguendo un processo ingegneristico sistematico, facente parte del progetto SodekoVS. Come punto di partenza abbiamo definito un nuovo framework concettuale, il quale fa riferimento ad una nuova architettura per applicazioni auto-organizzate e ad un nuovo metodo di sviluppo. L'architettura permette di definire le varie parti del sistema e le interazione che occorrono al suo interno, mentre la tecnica di sviluppo consente di mostrare quali attività devono essere svolte(e quali risultati ottenuti) durante le varie fasi di progetto. Come nella maggior parte delle metodologie per SOS, anche in questo progetto sono in atto ricerche per migliorare e standardizzare le fasi di progetto, cercando di ottenere una coerenza il più possibile generale. I problemi più importanti sui quali indagare riguardano principalmente la fase di Design delle attività coordinative(modelli di coordinazione riusabili), e la fase di validazione software tramite supporti simulativi.

Capitolo 4

Conclusione

In questo articolo abbiamo cercato di approfondire meglio quali sono i concetti legati all'auto-organizzazione, i vantaggi che comporta ai sistemi la presenza di tale proprietà, i problemi che derivano dalla sua introduzione nei MAS, e alcune metodologie d'esempio tramite cui progettare sistemi SOS. Nel terzo capitolo abbiamo mostrato solo alcune delle tante metodologie o architetture proposte per progettare in modo coerente sistemi self-organizing. Abbiamo potuto notare come gli strumenti a disposizione degli ingegneri software per affrontare questi problemi siano ancora molto limitati, e purtroppo anche molto specifici. L'obbiettivo sarebbe quello di trovare metodi,tool e architetture il più possibile generali, e una metodologia chiara è linerare che consenta di ottenere un processo di sviluppo coerente e collezzioni di pattern di coordinazione riusabili. Sarebbe un obbiettivo ambizioso, che però deve scontrarsi con 2 fattori importanti:

1. La grande complessità che governa i fenomeni emergenti e auto-organizzati rende difficile una classificazione del mondo coordinativo in pattern.

2. Come abbiamo visto nel capitolo 2, i fenomeni self-organizing possono manifestarsi in diversi modi(anche molto scorrelati tra loro), rendendo quasi impossibile l'avere una metodologia standard per coprire tutti i casi.

Per questi motivi la scelta più ovvia al momento è ancora quella di creare metodologie ad hoc per risolvere determinati problemi, cercan-

do di sfruttare i tool e i metodi a disposizione per modellare le varie fasi di sviluppo(ad un certo livello d'astrazione prefissato) e sfruttare al meglio le tecnologie a disposizione.

Bibliografia

[1] F.Zambonelli. Frontiers of self-organization for pervasive computing. In *Spray computers*, 2004.

[2] L. Gardelli, M. Viroli, and M. Casadei. Engineering the environment of self- organising multi-agent systems exploiting formal analysis tools. In AICA, editor, *Artificial Intelligence*, 2006.

[3] M. Gatti and C. D. Lucena. a bio-inspired method and representation model. In *Self-Organization and Emergent Behavior in Multi-Agents Systems*, 2008.

[4] P. Grassé. La reconstruction du nid et les coordinations interindividuelles chez bellicositermes natalensis et cubitermes sp. la théorie de la stigmergie : essai d'interprétation du comportement des termites constructeurs. In *Insectes Sociaux*, 1959.

[5] R. Korf. Real-time heuristic search. In *Artificial Intelligence*, 1990.

[6] J. Lehman. A gentle introduction to soar, an architecture for human cognition. 2006.

[7] J. Pavon, J. Gomez-Sanz, and R. Fuentes. The ingenias methodology and tools. In *Agent-oriented methodologies*, 2005.

[8] J. Sudeikat, L. Braubach, and A. Pokahr. The sodekovs approach. In *Systematically Enineering Self-Organizing System*, 2009.

[9] D. Weyns and T. Holvoet. Role based model for adaptive agents. 2004.

[10] T. D. Wolf. Analysing and engineering self-organising emergent applications. In AICA, editor, *Department of Computer Science*, 2007.

[11] T. D. Wolf and T. Holvoet. A taxonomy for self-properties in decentralised autonomic computing. In *Autonomic Computing*, 2006.