

**ALMA MATER STUDIORUM – UNIVERSITA' DI
BOLOGNA SEDE DI CESENA
SECONDA FACOLTA' DI INGEGNERIA CON SEDE A
CESENA
CORSO DI LAUREA IN INGEGNERIA DEI SISTEMI E
DELLE TECNOLOGIE PER L'INFORMAZIONE**

Simulazione software basata su agenti

Elaborato in:
Sistemi intelligenti distribuiti LS

Relatore:
Prof. Andrea Omicini

Presentata da:
Paolo Angelini

Sessione II°

Anno Accademico 2005/06

Sommario

Sommario	3
1 Introduzione	7
2 Sulla Simulazione	9
2.1 I Modelli	10
2.2 La simulazione al computer	11
2.2.1 Metodologia per le simulazioni al computer	13
2.3 Tecniche di simulazione	15
2.3.1 Micro-Simulation e Macro-Simulation	15
2.3.2 Discrete Event Simulation	15
2.3.3 Dynamic Micro Simulation	16
2.3.4 Equazioni differenziali	17
2.3.5 Object Oriented Simulation	18
3 La Simulazione ad Agenti	19
3.1 Sintassi e semantica del concetto di agente	21
4 Modelli per la simulazione ad agenti	23
4.1 Cellular Automata	23
4.2 MultiLayered Multi Agent Situated System	26
5 Repast	33
5.1 Note tecniche	34
5.1.1 Installazione	34
5.1.2 Esecuzione	34
5.1.3 Librerie	35
5.2 Framework logico	35
5.2.1 Modellazione	35
5.2.2 Simulazione	36
5.3 Note sull'uso	37
5.3.1 La toolbar	37
5.3.2 Creazione del modello da simulare	39
5.3.2.1 Model Class	39
5.3.2.2 Space Class	44
5.3.2.3 Agent Class	45
5.3.3 Definizione di una simulazione	46
5.3.3.1 Lo scheduler	46
5.3.3.2 Le azioni	46
6 TuCSoN	49
6.1 A&A: un framework per la modellizzazione e la simulazione basato su TuCSoN	53
7 RePast v.s. TuCSoN	55
8 Conclusioni	59
9 Bibliografia	61

Indice delle figure

FIGURA 4-1 MMASS MODEL	27
FIGURA 5-1 REPAST: LA TOOLBAR.....	37
FIGURA 5-2 REPAST: GESTIONE DEI MODELLI DI SIMULAZIONE	37
FIGURA 5-3 REPAST: LA FINESTRA CON I PARAMETRI DELLA SIMULAZIONE	42
FIGURA 6-1 CENTRO DI TUPLE TuCSoN.....	51
FIGURA 6-2 NODO TuCSoN.....	52
FIGURA 6-3 TuCSoN	52
FIGURA 6-4 A&A: AGENTI (CERCHI), ARTEFATTI (QUADRATI) E WORKSPACES (AREE TRATTEGGIATE).....	53

1 Introduzione

La simulazione è un'attività presente in molti settori delle attività umane: vari campi dell'ingegneria, attività sociali, medicina, economia, eccetera. Attraverso il suo uso è possibile effettuare previsioni e analisi di sistemi complessi, capirne il comportamento e sottoporli ad azioni che nella realtà sarebbero difficili o estremamente costose.

Seguendo l'approccio più antico e tradizionale, si parte dalla modellazione del problema, e quindi delle varie entità coinvolte, arrivando alla scrittura di equazioni differenziali. Il modello si focalizza su aspetti globali, preferendo una "vista d'insieme" ad un'analisi basata sulle varie entità coinvolte. Questo paradigma è valido per sistemi relativamente "semplici", dove per semplicità si intende la presenza di un limitato numero e tipo di entità coinvolte le cui singole fluttuazioni sono tali da essere trascurabili nel loro insieme e la capacità di definire con termini matematici il comportamento globale. Per sistemi estesi ed articolati è difficile arrivare ad una modellazione di insieme attraverso l'utilizzo di equazioni differenziali. Occorre poi considerare che tecniche basate su modelli puramente matematici non riuscirebbero a mettere in luce aspetti importanti, quali per esempio l'interazione tra le parti che compongono il sistema da simulare, la loro singola evoluzione. Inoltre non è possibile osservare la formazione di comportamenti emergenti.

Con l' Agent Based Simulation (ABS) l'approccio alla simulazione cambia. Il sistema non viene più studiato e modellato nella sua interezza e in forma puramente matematica, ma attraverso il concetto di agente. La visione viene ribaltata e diventa bottom-up: si modella ogni singolo tipo di entità e l'ambiente in cui queste vivono. La modellazione è più semplice, intuitiva, scalabile e flessibile in quanto il lavoro svolto può essere facilmente riusato qualora il sistema da simulare presenti in futuro nuove entità. E' possibile effettuare diverse simulazioni relative allo stesso sistema in esame, variando parametri e condizioni iniziali. Il risultato è ottenuto attraverso l'interazione reciproca in quanto ogni entità viene a contatto con altre entità e con l'ambiente. Ciò permette di capire ed osservare la formazione di comportamenti emergenti. Tale approccio è indispensabile per lo studio di

sistemi complessi ove non è possibile predire a priori il comportamento e l'evoluzione globale e quella degli enti che lo compongono.

Con questa tesi parleremo degli scopi e dell'utilità della simulazione per poi discutere sulle caratteristiche dei vari approcci. Successivamente ci focalizzeremo su quello basato sull'utilizzo di agenti e discuteremo sul concetto di agente. Proseguiremo analizzando e discutendo alcuni modelli e metodologie per la ABS. La trattazione continuerà con lo studio di RePast, un tool per la scrittura e l'esecuzione di simulazioni in ottica ABM, di cui verranno descritte le caratteristiche tecniche più importanti, il modello concettuale e alcune note per l'uso. La stessa cosa verrà fatta per TuCSoN nell'ultimo capitolo di questa tesi.

2 Sulla Simulazione

Nelle scienze della natura, come la fisica e la chimica, le teorie formulate possono essere verificate direttamente in laboratorio consentendo agli studiosi di verificare la correttezza delle loro ipotesi. Gli esperimenti di questo tipo consentono di capire quali sono i fattori che danno origine a determinati fenomeni e di modificare alcuni aspetti e caratteristiche per osservare le conseguenze che si ripercuotono sul sistema. Tutto ciò è però limitato al solo studio di situazioni e sistemi semplici perché, in generale, quasi tutti i fenomeni che interessano le scienze della natura e dell'uomo danno origine a sistemi complessi [1]. Prima di effettuare il test in laboratorio è necessaria una lunga e costosa fase di analisi e di progetto dell'esperimento il quale finirebbe però per simulare uno o pochi aspetti del sistema oggetto di studio.

L'opportunità di verificare le teorie, di fare previsioni, non è invece disponibile per le scienze sociali quali l'antropologia, la sociologia, eccetera. Diversamente da quanto accade per le scienze della natura, non esiste nessun concetto di laboratorio e quindi non è possibile, in alcuni casi facile e conveniente, mettere in pratica concetti, idee e teorie. Il concetto di laboratorio è ancora di più assente per gli studiosi che si occupano di fenomeni e di realtà non direttamente accessibili: è il caso degli storici e degli archeologi. In altri casi il test in laboratorio sarebbe estremamente costoso, basti pensare ai progetti aerospaziali, a quelli meccanici, eccetera, le cui prove su una realizzazione prototipale finirebbero, in caso d'errori di progettazione, per danneggiare o distruggere assemblati dal costo elevato. A volte l'inconveniente è dato dal tempo perché il risultato finale ottenuto partendo da un'implementazione prototipale richiederebbe tempi di attesa troppo elevati. In questi casi più che un laboratorio "fisico" ne servirebbe uno "virtuale".

Da qui l'importanza della simulazione perché è in grado di fornire un potente mezzo per la comprensione e lo studio di alcune realtà scientifiche e di alcuni loro aspetti. Con la simulazione si possono quindi studiare tutti i sistemi, sia quelli semplici che quelli complessi; da questo punto di vista rappresenta per i ricercatori uno strumento decisamente più completo rispetto al laboratorio. La simulazione consente inoltre di elaborare modelli che mirano ad

ottenere particolari risposte, ad esempio conoscere quali saranno le conseguenze originate da un particolare insieme di fattori; per esempio, grazie a questo metodo, gli studiosi delle scienze sociali possono valutare gli effetti originati da particolari provvedimenti economici o sociali prima della loro applicazione nella realtà. Si tratta della tipica domanda “what if” a cui è possibile dare risposta modificando alcuni parametri e/o caratteristiche del modello.

2.1 I Modelli

La realtà da studiare viene descritta attraverso un modello che dovrà catturarne e metterne in evidenza gli aspetti d’interesse tralasciando quelli che risulterebbero essere insignificanti ed inutili. Dettaglio, precisione ed estensione del modello saranno rapportati allo scopo finale della simulazione. Se si è interessati a sapere qual è il risultato finale della simulazione in termini di risposte si/no o stato/valore finale si farà uso di un modello con certe caratteristiche, se si è interessati a capire quali sono i motivi ed i fattori che portano ad un certo stato finale del sistema il modello dovrà essere più fine ed accurato e dovrà tenere conto d’alcuni fattori che nel primo scenario non verrebbero considerati. Questa fase è estremamente importante per l’ottenimento di risultati validi e significativi. Un buon modello deve avere le seguenti caratteristiche:

- *Correttezza*: deve imitare correttamente gli aspetti di interesse della realtà studiata.
- *Precisione*: è misurata attraverso la differenza tra i risultati della simulazione e quelli della realtà.
- *Generalità*: deve essere in grado di descrivere correttamente varianti della realtà in esame e essere il più scalabile possibile. Ciò può essere reso possibile in parte attraverso l’uso di parametri.
- *Affidabilità*: a partire dal modello la simulazione deve fornire sempre risultati corretti per ogni condizione di utilizzo. In altre parole non deve

accadere per alcuni particolari configurazioni si ottengono risultati completamente sbagliati.

- *Costo*: è il costo richiesto per la creazione del modello e deve essere minimizzato.

I modelli vengono definiti:

- *Dinamici*: sono in grado di riprodurre l'evoluzione di un sistema soggetto al trascorrere del tempo.
- *Statici*: si assume che il sistema operi in un certo numero di istanti temporali ben definiti, come accade per le simulazioni secondo il modello Monte Carlo.
- *Deterministici*: non è presente alcun dato aleatorio. Tali modelli sollecitati con lo stesso input producono sempre lo stesso output.
- *Stocastici*: contengono variabili aleatorie.
- *Discreti*: il sistema evolve in certi istanti temporali. La dimensione temporale viene discretizzata.
- *Continui*: il sistema evolve in maniera continua con il passare del tempo. La dimensione temporale è un continuo.

2.2 La simulazione al computer

Seguendo gli approcci classici, la modellazione avviene attraverso la scrittura di equazioni differenziali. Si perviene ad uno o più sistemi di equazioni la cui risoluzione rappresenta il risultato della simulazione. Ciò penalizza fortemente gli studiosi di scienze per le quali non è possibile o realistico ricondursi a modelli puramente matematici la cui risoluzione risulta computazionalmente infattibile in molti casi.

Con l'avvento, la diffusione e la crescita computazionale dei calcolatori elettronici, la simulazione al computer offre a tutte le discipline un nuovo ed importante strumento di ricerca in quanto è possibile studiare vari aspetti della realtà tramite la creazione e la messa in esecuzione al computer di modelli accuratamente specificati. Si ha più facilità di astrazione, più flessibilità e

capacità espressiva grazie ai vari linguaggi di programmazione e alla definizione di pattern a cui ricondursi. Si perviene così alla creazione di una realtà virtuale in cui è possibile fare studi ed esperimenti altrimenti impossibili.

La simulazione al computer è considerata la “terza via” [2] disponibile agli studiosi delle scienze sociali, in una classificazione che comprende:

- modelli letterario-descrittivi: molto flessibili, ma la loro adeguatezza non può essere verificata con strumenti di calcolo
- modelli matematico-statistici: computabili, ma hanno un grave problema rappresentato dalle semplificazioni imposte che possono causare un allontanamento dalla realtà.

Con la simulazione al computer si ottiene un giusto compromesso tra flessibilità, capacità di modellare la realtà oggetto di studio e computabilità.

Una definizione del metodo simulativo al computer è fornita in [1]:

Le simulazioni sono teorie interpretative dei fenomeni della realtà formulate come un programma che gira in un computer. Le teorie della scienza mirano a identificare i meccanismi, i processi, i fattori che stanno dietro ai fenomeni e che spiegano i fenomeni, ce li fanno capire. Adottare il metodo della simulazione significa tradurre una teoria in un programma di computer, far girare il programma nel computer e verificare se il programma, cioè la simulazione, riproduce i fenomeni che la teoria intende spiegare. Le simulazioni sono il terzo strumento della scienza, uno strumento che si è aggiunto abbastanza di recente, cioè da quando computer veloci e potenti sono divenuti accessibili a qualunque scienziato, ai due strumenti di ricerca tradizionali: le teorie e gli esperimenti di laboratorio.

Con l'utilizzo del computer la simulazione è in grado di fornire nello stesso istante varie risposte e vari tipi di risposta, più precisamente gli utilizzatori possono avere diversi obiettivi [3] e se ne possono distinguere almeno tre:

1. La *proiezione*, cioè l'uso di modelli per prevedere gli avvenimenti futuri legati ad un particolare fenomeno.
2. La *sperimentazione*, nel caso in cui con la simulazione si intendano spiegare dei particolari fenomeni e si desideri verificare la qualità dei

modelli proposti. Il modello creato nel computer può essere usato in due modi: manipolando le variabili del modello, possibilità particolarmente indicata quando non si possono realizzare modifiche nella realtà oppure quando si intende valutare quali siano le cause scatenanti di particolari eventi, con l'obiettivo di utilizzare il computer come un vero e proprio “laboratorio” delle scienze sociali. Un secondo modo d'utilizzo è l'osservazione del comportamento del modello “così come è” e dei fenomeni che hanno origine da esso.

3. *L'esplorazione* del fenomeno costruito nel “laboratorio virtuale”, per osservare e studiare i diversi aspetti delle dinamiche sociali e l'evoluzione della cooperazione tra le entità che costituiscono il sistema.

2.2.1 Metodologia per le simulazioni al computer

Un buon approccio alla simulazione al computer deve seguire una valida metodologia [4] la quale si compone di determinate fasi; nel nostro caso citiamo le seguenti:

- *Raccolta dei requisiti*: in cui si identifica e chiarisce bene qual è lo scopo della simulazione, a quali domande deve dare risposta.
- *Creazione del modello*: in cui si costruisce un modello del sistema, descrivendo gli elementi da cui è composto e le relazioni che vi intercorrono.
- *Progetto e definizione dell'architettura del sistema software*: in cui si effettua una traslazione del modello in specifiche facilmente implementabili attraverso l'utilizzo di una o più tecnologie. In questa fase si cerca di identificare/utilizzare pattern di progettazione.
- *Implementazione*: in cui si scrive codice in uno o più linguaggi di programmazione.
- *Verification, Validation e Accreditation (VV&A)*: in cui si valutano i risultati ottenuti confrontandoli con eventuali risultati noti. In questa fase si verifica la validità e la bontà di quanto realizzato.

Questo processo produttivo è, come per quasi tutte le metodologie scientifiche, ciclico. Una volta che i requisiti sono stati ben definiti si itera tra le restanti

fasi, in particolare si raffina sempre di più il modello fino ad arrivare a quello più vicino e fedele alla realtà da simulare.

Tipicamente una simulazione viene concepita da un gruppo di persone, ognuna delle quali vanta capacità e competenze ben specifiche:

- *End-user*: è il committente della simulazione. Il suo compito è di chiarire il problema e quali sono i requisiti.
- *Domain expert*: è una persona (o un gruppo di persone) che ha più che buone conoscenze della realtà da simulare.
- *Modeler*: attraverso la consulenza dell'End-user e del Domain expert arriva alla costruzione di un modello della realtà.
- *Software Architect*: il suo compito è quello di definire dei pattern software e/o dei modelli simulativi per la definizione di un framework da utilizzare in lavori futuri.
- *Designer*: deve mappare il modello costruito dal Modeler in un progetto software. Tale progetto dovrebbe essere facilmente implementabile dal Developer.
- *Developer*: si occupa della fase implementativa.
- *Tester*: verifica la validità e la correttezza dell'applicativo costruito, il tutto in accordo con quanto stabilito dall'end-user.

Solitamente si ha che una o più di queste sette figure virtuali finiscono per coincidere con la stessa, o le stesse, persone fisiche. In linea di principio l'End-user può ricoprire il ruolo di tester e in molti casi anche quello di Domain expert nonché di Modeler. D'altra parte le figure di Software Architect e di Designer possono coincidere. In qualche caso, soprattutto quando il sistema software da realizzare è tutto sommato semplice, la persona che svolge suddette due funzioni può ricoprire anche il ruolo di Developer.

2.3 Tecniche di simulazione

2.3.1 Micro-Simulation e Macro-Simulation

Una simulazione può essere una micro-simulation o una macro-simulation indipendentemente dalla metodologia utilizzata; tutto dipende dal taglio, dal concentrarsi su una vista di insieme piuttosto che decentralizzare la descrizione sulle varie entità. Se il sistema viene analizzato, studiato e modellato nella sua globalità, in termini di popolazione piuttosto che degli individui che la compongono ci si trova di fronte ad una tecnica di macro-simulation. Nella realtà ogni entità fornisce un suo contributo al raggiungimento del risultato finale e, anziché considerare la somma di ognuna di esse, viene fatta una sorta di media ponderata dei vari contributi. Tutto viene come centralizzato, le entità sono viste come agglomerati inscindibili di una macrostruttura caratterizzata da un certo numero di parametri.

Quanto detto è l'esatto opposto di quello che accade per le tecniche di micro-simulation. Qui il modello cerca di specificare il contributo dato dalle singole entità ed è in grado di osservare e studiare la formazione di **comportamenti emergenti**; si tratta di situazioni imprevedibili a priori, sia qualitativamente che quantitativamente, il cui verificarsi è dato dalla reciproca interazione tra le varie entità, più precisamente quando il raggiungimento dello stato finale del sistema è funzione non lineare del contributo di ogni singolo ente. Con tecniche di macro-simulation ciò è impossibile, queste ultime vanno utilizzate quando le dinamiche della realtà sono dominate da precise leggi di vario tipo piuttosto che da scambio di informazioni e iterazioni tra le varie parti del sistema.

2.3.2 Discrete Event Simulation

Generalmente, ogni modello può essere visto come la specifica di un sistema in termini di stati ed eventi. La Discrete Event Simulation (DES) fa uso di tale principio basandosi sugli eventi a cui è soggetto il sistema simulato e quindi sugli effetti e sulle ripercussioni che questi hanno sul suo relativo stato. Nella DES il tempo è discretizzato, ciò si scontra con quanto accade invece nella Continuous Event Simulation (CES) dove gli eventi ed i cambiamenti di stato si

verificano, in linea di principio, in maniera continua con il passare del tempo. Tuttavia è possibile convertire modelli tempo continui in modelli tempo discreti considerando solamente gli istanti di inizio e di fine dei vari eventi. Da questo punto di vista la tecnica DES può essere intesa come un caso particolare di quella CES.

Esistono due approcci alla DES: time driver ed event driven. Nell'approccio time driven il tempo di simulazione avanza in maniera costante, attraverso step temporali ben definiti ed immutati. Nell'Event Driver il tempo avanza man mano che si verifica un evento. Quest'ultimo approccio è quello più diffuso perché in grado di percepire e gestire al meglio tutti gli eventi che interessano il sistema. Ciò lo rende più efficiente della versione time driver.

Fondamentale è la presenza di una event-list. Si tratta di una struttura centralizzata in cui gli eventi vengono memorizzati e marcati temporalmente (time stamped) per tenere traccia della relativa posizione temporale e del tempo che avanza. Il motore che gestisce l'esecuzione della simulazione non fa altro che prendere continuamente il primo evento dalla event-list, settare il simulation time al time stamp dell'evento considerato e simulare gli effetti sul sistema.

2.3.3 Dynamic Micro Simulation

L'idea che sta alla base della Dynamic Micro Simulation (DMS) è basata sugli effetti che il tempo ha sulle entità che compongono il sistema. Il focus è sulle singole entità piuttosto che sul sistema in generale anche se è possibile effettuare studi globali considerando questo ultimo come l'unica presente.

Ci si interessa ad un certo numero di caratteristiche delle varie entità. Il trascorrere del tempo fa sì che le caratteristiche di interesse siano soggette a transizioni da un valore ad un altro, e quindi da uno stato all'altro. Tali transizioni vengono descritte in termini probabilistici in base a regole scritte nel modello. Il tempo viene fatto avanzare in maniera discreta e a step temporali ben definiti. In ognuno di questi step vengono applicate le probabilità di transizione allo stato delle varie entità le quali evolveranno verso nuove configurazioni.

Il grande limite di questa tecnica di simulazione è rappresentato dal fatto che il comportamento di ogni entità è descritto in termini probabilistici, senza tenere in minima considerazione le loro intenzioni, decisioni, piani, eccetera. Inoltre non si considera minimamente l'interazione tra le varie entità, che possono dunque essere intese come isolate e separate le une dalle altre. Ciò non permette lo studio di comportamenti emergenti e più in generale la simulazione di realtà in cui il risultato finale è dato dall'interazione e dallo scambio di informazioni tra le parti. Tale tecnica è da evitare in situazioni soggette ad elevato dinamismo, in cui il comportamento delle varie entità può subire sostanziosi cambiamenti in seguito al verificarsi di particolari eventi, per esempio nei mercati finanziari.

2.3.4 Equazioni differenziali

Ne abbiamo parlato in precedenza, dicendo che si tratta della tecnica di simulazione più antica. Il sistema viene modellato attraverso la scrittura di equazioni differenziali e quindi in maniera puramente analitica. Storicamente è stata utilizzata da scienziati per lo studio di fenomeni naturali quali le leggi della meccanica, dell'elettromagnetismo, eccetera.

E' una tecnica di macro-simulation e il suo uso è rivolto a scenari facilmente descrivibili in termini matematici e da un punto di vista globale. Si riescono a fare delle previsioni, ma non a capire e studiare la formazione di situazioni e comportamenti emergenti. A volte è necessario approssimare e semplificare la realtà in maniera tale da potersi ricondurre a modelli noti di cui esistono tecniche risolutive ben collaudate; ciò ha come effetto collaterale quello di pervenire a risultati che seppur ben approssimati possono distare parecchio da quelli del mondo reale. Sistemi complessi ed architetture articolati sia per la struttura che per l'interazione tra le parti costituenti non possono essere studiati con questa tecnica onde evitare il raggiungimento di risultati poco attendibili e con un elevato grado di incertezza.

2.3.5 Object Oriented Simulation

La realtà viene modellata attraverso ADT, in maniera più o meno dettagliata in base alle esigenze del caso.

Attraverso la Object Oriented Simulation (OOS) è possibile godere della “structure preserving modelling” della realtà simulata; in altre parole le entità presenti nella realtà trovano un corrispettivo 1-1 con quelle virtuali della simulazione. Tutto ciò permette elevata pulizia concettuale, modellazione e progetto del sistema software più chiari ed intuitivi in quanto ciò che si definisce è effettivamente presente nella realtà. Osserviamo poi che ai vari livelli, e quindi da quello astratto e puramente modellistico giù fino a quello implementativo, l’idea di oggetto si conserva. Ciò non accade per tutte le altre tecniche di simulazione, e quindi anche in quella ad agenti, dove quanto definito nel modello finisce per essere sparso e distribuito nei vari costrutti messi a disposizione dalla tecnologia usata per in fase di implementazione.

Con la OOS è possibile arrivare allo studio di situazioni emergenti, inoltre viene supportata in maniera naturale la computazione distribuita perché tutto quel che riguarda l’evoluzione del sistema è decentralizzato, demandato alle singole entità e alla loro interazione.

Nel prossimo capitolo vedremo che le proprietà e le caratteristiche sopraelencate sono presenti anche nella ABS. La tecnica di simulazione ad oggetti è quella più vicina a quella ad agenti ma le differenze tra i due approcci sono concettualmente molto forti e significative. Un oggetto è un’entità dotata di un flusso di controllo che, diversamente da quanto accade per gli agenti ove è proprio, li attraversa ogni tanto rendendoli computazionalmente influenti solo in alcuni momenti. Inoltre un oggetto è puramente reattivo e quindi non presenta nessuna forma di pro-attività e autonomia comportamentale.

In seguito noteremo che tecnologie come Java consentono di avvicinarsi al concetto di agente e, dal momento che per questi non esiste una linguaggio di programmazione, è possibile emulare qualcuna delle loro caratteristiche attraverso tecniche basate sulla OOP.

3 La Simulazione ad Agenti

Si tratta di una tecnica intimamente di micro-simulation. Le entità vengono mappate su agenti, generalmente considerate come figure autonome. Il concetto di agente è molto vasto perché viene applicato in maniera diversa a seconda della realtà studiata. La percezione più diffusa vede un agente come un ente dalle seguenti caratteristiche:

- Individualità: un agente è un ente ben distinguibile dagli altri.
- Socialità: un agente interagisce più o meno intensamente con gli altri agenti.
- Autonomia: un agente esegue determinate azioni in base a propri ragionamenti.
- Goal oriented: un agente compie determinate azioni per il raggiungimento di un determinato obiettivo.
- Flessibilità: un agente modifica il proprio comportamento in base ai fatti che stanno accadendo.
- Collocazione nello spazio: un agente vive in un mondo.
- Mobilità: un agente ha o può avere la possibilità di muoversi all'interno del mondo in cui vive.

Tutte queste caratteristiche sono però diversamente sviluppate a seconda della realtà che si sta studiando [10]. In altre parole non c'è una netta distinzione tra agente ed oggetto, si fa piuttosto riferimento ad un'astrazione collocata all'interno di un intervallo continuo ai cui estremi si trovano il concetto di agente "ideale" e quello di oggetto. L'agente "reale" sarà tale da avere le diverse caratteristiche più prossime a quelle dell'agente ideale che a quelle dell'oggetto.

Con la ABS la realtà viene decentralizzata e l'evoluzione del sistema è data dalla interazione tra le varie entità e l'ambiente in cui queste vivono.

Analogamente a quanto accade per l'approccio ad oggetti, la ABS gode dell'importante proprietà di "structure preserving modelling". E' altresì possibile studiare la formazione di situazioni emergenti e, in aggiunta a quanto

accade per tecniche di simulazione ad oggetti, anche di comportamenti emergenti perché le entità in gioco sono dotate di un goal e di una propria autonomia comportamentale. Quest'ultima caratteristica permette lo studio di scenari caratterizzati da un elevato grado di dinamismo in cui alcuni avvenimenti hanno l'effetto di modificare drasticamente la politica comportamentale di un agente. In [5] leggiamo che, in linea di principio, un agente dovrebbe disporre di due diversi set di regole:

- Regole che riguardano il comportamento.
- Regole per modificare le regole che riguardano il comportamento.

In [6] viene effettuato un confronto con le tecniche di simulazione classiche, basate su modelli analitici, e viene messo in luce il fatto che la ABS può essere usata in tre casi distinti:

- Il fenomeno che si va a studiare può essere completamente descritto da equazioni risolvibili analiticamente o numericamente: in questo caso la simulazione può essere usata come metodo alternativo per arrivare agli stessi risultati o come strumento per presentarli in modo più facilmente comprensibile rispetto a quelli di un modello matematico.
- Il modello matematico adottato può essere descritto ma le sue equazioni non sono risolvibili oppure i risultati non sono stabili.
- Problemi per i quali non è possibile una spiegazione fondata su equazioni: la simulazione ad agenti è perciò l'unico metodo con cui è possibile studiare questi fenomeni.

La ABS trova una naturale e vantaggiosa applicazione in scenari in cui:

- La rappresentazione attraverso entità con le caratteristiche di agente è naturale ed immediata.
- Sono presenti comportamenti e decisioni.
- Il comportamento delle entità presenta dinamismo.
- Occorre focalizzare l'attenzione sull'interazione tra le varie entità piuttosto che sul sistema nella sua generalità.
- Quando il passato non è in grado di dare indizi e contributi significativi per la predizione del futuro.

- Il sistema presenta un grado di complessità troppo elevato a livello di interdipendenze tra le parti che lo compongono e dal punto di vista di modellazione top-down delle singole parti.

In letteratura si trovano diversi casi di applicazione della tecnica ABS alle scienze sociali oltre che a quelle ingegneristiche [5]:

- Antropologia: per lo studio di nascita, crescita e declino di alcune civiltà del passato. Più in particolare si è cercato di capire il perché del crollo della civiltà mesopotamica e la scomparsa degli Anasazi nel sud-est degli USA.
- Economia: per analizzare gli andamenti di alcuni mercati finanziari.
- Sociologia: per capire la formazione di identità nazionali e la nascita di Stati .
- Scienza cognitive: l'idea è quella di simulare gli stati emozionali a livello di interazione sociale tra gli individui.
- Biologia: per lo studio della formazione di strutture emergenti ed auto-organizzanti a livello molecolare.

3.1 Sintassi e semantica del concetto di agente

Il concetto di agente è molto vago [7], ci si limita a parlare di un'entità autonoma con tutta una serie di caratteristiche. Tale astrazione risulta molto comoda e vantaggiosa in fase di modellazione, tuttavia la terminologia utilizzata rappresenta una sorta di muto accordo a livello di sintassi, mentre a livello di semantica tutto varia nel migrare da un modello ad un altro.

Si elencano le proprietà e le caratteristiche tipiche di un agente, ma niente riguardo a come questi è fatto. Quello che manca è una visione comune ai livelli inferiori, uno standard de facto, una lingua franca relativamente alle tecnologie per l'implementazione di agenti e la costruzione di sistemi ad agenti.

Allo stato attuale tutti i concetti e le caratteristiche definite a livello sintattico non traslano correttamente a livello computazionale ed

implementativo. Il rischio che si corre è quello di costruire sistemi con caratteristiche diverse da quelle definite in fase di analisi e modellazione.

4 Modelli per la simulazione ad agenti

Un unico tipo modello e un' unica metodologia non sono in grado di coprire e soddisfare le esigenze simulative di tutte le possibili realtà oggetto di studio [19]. Ciò è facilmente comprensibile se si pensa che dal punto di vista sintattico, e dunque implementativo, non esiste uno standard de facto per la scrittura di software agent based.

Da qui la nascita di vari approcci dedicati a campi ristretti e ben specifici in ognuno dei quali il concetto di agente assume determinate caratteristiche e funzionalità che non hanno nulla a che vedere con quelle presenti in altri scenari.

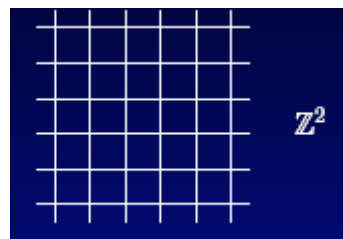
4.1 Cellular Automata

E' la prima metodologia conosciuta in grado di permettere lo studio di un sistema secondo un approccio basato sull'interazione di entità, in accordo con regole più o meno semplici in base alla situazione.

Il paradigma basato su Cellular Automata (CA) prevede:

- Uno spazio d -dimensionale suddiviso in celle omogenee (Z^d).

Nell'esempio consideriamo uno spazio bidimensionale ($d=2$):



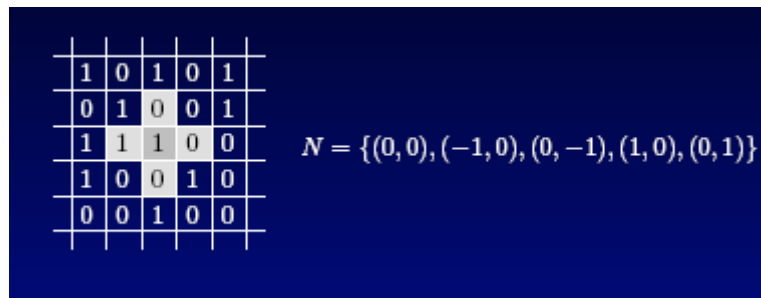
- Un certo numero di stati per ogni cella ($A=\{0,1,\dots,n\}$). Nell'esempio consideriamo stati binari ($A=\{0,1\}$):

1	0	1	0	1
0	1	0	0	1
1	1	1	0	0
1	0	0	1	0
0	0	1	0	0

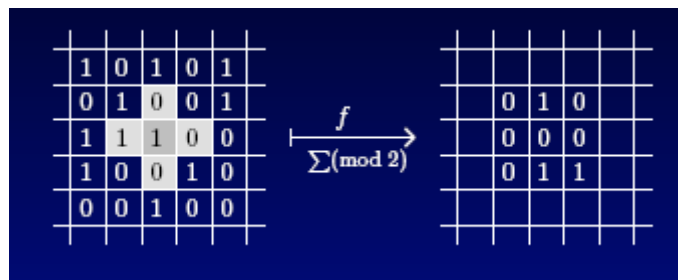
$A = \{0,1\}$

- Una regola per stabilire quali sono le celle vicine di ogni cella. Ogni cella avrà infatti delle celle vicine ($N \subset \mathbb{Z}^d$). Esistono 3 regole per definire i vicini:
 - Von Neumann: data una cella, le celle vicine sono quelle a nord, sud, ovest, est.
 - Moore: si estende la regola precedente considerando anche le celle a nord/est, nord/ovest, sud/est, sud/ovest
 - Extended Moore: si considerano le celle contenute in un quadrato di lato l a partire dalla cella in esame. La regola Moore è un caso particolare con $l=1$

Nell'esempio consideriamo si fa uso della regola di Von Neumann:



- Una regola per il cambiamento di stato di ogni cella ($f: A^{|\mathbb{N}|} \rightarrow A$). Questa regola è intesa come globale in quanto interessa tutte le celle della griglia. Nell'esempio il nuovo valore di ogni cella sarà dato dalla somma modulo 2 del valore delle celle vicine:



La comunicazione tra le celle è limitata all'interazione locale con quelle vicine. Lo stato di ogni cella cambia con il trascorrere del tempo in maniera sincrona con quello delle altre, in altre parole le celle evolvono simultaneamente. L'evoluzione globale del sistema risulta come un processo parallelo e sincronizzato. Generalmente ogni cella si modifica tenendo conto dello stato

delle proprie celle vicine. Qualora ciò non accadesse non sarebbe lecito parlare di CA.

L'Automa Cellulare può essere inteso come un Sistema Multi Agente in cui la struttura spaziale dell'ambiente è precisa e strutturata su di una griglia. Le celle sono omogenee e rappresentano gli agenti che sono dunque entità fisse. Il modello computazionale è decentralizzato e distribuito tra i vari agenti la cui interazione locale porta il sistema ad evolversi con il passare del tempo che avviene in maniera discreta.

Il concetto di agente è debole: la mobilità è azzerata, concetti come proattività e goal sono assenti, la comunicazione è drasticamente limitata e infine non esiste il concetto di tipo per via della proprietà di omogeneità delle celle. Inoltre l'ambiente non viene considerato. L'agente viene inteso come un automa a stati finiti, fisso, che interagisce solamente con i propri vicini e che esegue semplici operazioni in momenti ben definiti. Si tratta comunque di un'entità dotata di una sua individualità, di una propria autonomia, immersa in un contesto sociale e in grado di comunicare con i propri simili.

Modelli basati su Cellular Automata sono stati usati in maniera sempre più crescente in alcuni campi della chimica, della fisica, della geografia, della matematica e delle scienze sociali, inoltre sono state proposte varianti ed evoluzioni al modello iniziale. Tra le varie applicazioni citiamo:

- Lo studio dell'attività intracellulare a livello di cluster di celle nella biologia.
- Lo studio del sistema immunitario nella medicina.
- La simulazione del comportamento dei gas nella fisica.
- Lo studio dei fenomeni di cristallizzazione nella fisica e nella chimica.
- La simulazione della propagazione degli incendi boschivi.
- Lo studio degli ecosistemi naturali a livello di predatore e preda nell'ecologia.
- Metodo alternativo all'uso delle equazioni differenziali per la descrizione di fenomeni fisici ed elettromagnetici.

Il grande pregio della modellazione attraverso l'utilizzo di automi cellulare è dato dalla intrinseca modularità, regolarità e flessibilità su cui si basa il paradigma. Tutto si basa sul definire il numero di stati che può assumere ogni cella e il tipo di regola globale che governa la loro evoluzione. Inoltre attraverso gli automi cellulari è possibile costruire una macchina di Turing universale, ci troviamo quindi di fronte ad un modello computazionalmente Turing complete.

Tuttavia i CA presentano alcuni limiti, più precisamente:

- Impossibilità di definire agenti eterogenei. Ciò implica l'assenza di dinamismo e flessibilità in quanto gli agenti vengono considerati come entità comportamentalmente fisse ed immutabili.
- Impossibilità di definire l'interazione e la comunicazione a distanza tra gli agenti.
- Necessità di definire regole particolarmente complesse e celle con un elevato numero di stati, che in alcuni casi può crescere in maniera fattoriale, per lo studio di sistemi composti da entità dotate di comportamenti complessi ed articolati.
- Impossibilità di definire strutture spaziali complesse e di modellare l'ambiente in cui vivono gli agenti

In letteratura sono state proposte alcune varianti agli automi cellulari. Si tratta sostanzialmente di definire vari tipi di agenti, abbattendo quindi il vincolo di omogeneità, e di permettere la comunicazione a distanza

4.2 MultiLayered Multi Agent Situated System

MultiLayered Multi Agent Situated System (MMASS) è un framework grazie al quale è possibile descrivere, rappresentare e simulare sistemi complessi che richiedono la presa in considerazione delle caratteristiche spaziali dell'ambiente e di diverse forme di interazione tra i vari enti.

Questo modello può essere inteso come una forte estensione a quello basato su CA; vengono infatti rilassati i vincoli relativi all'omogeneità degli agenti e alle forme di interazione e comunicazione.

Il modello di un MMASS è facilmente rappresentabile con la seguente figura (una descrizione approfondita e dettagliata è presente in []):

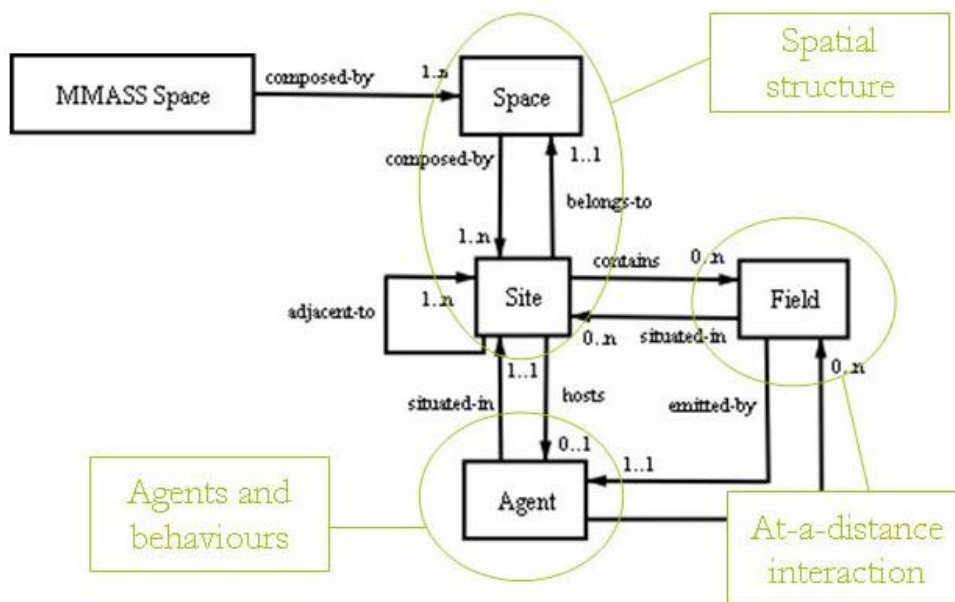


Figura 4-1 MMASS Model

Un MMASS è dato dall'insieme di un certo numero di Situated MAS tra di loro interagenti. Si definisce attraverso la tripla $\langle Space, F, A \rangle$ dove $Space$ modella l'ambiente, A e F insiemi finiti che comprendono rispettivamente gli agenti e i campi che si propagano e grazie ai quali avviene la comunicazione a distanza tra gli agenti. A e F sono insiemi finiti.

Uno spazio è dunque considerato come un grafo non orientato di siti. Un sito è, esattamente come avveniva per i CA, una struttura n -dimensionale suddivisa in cella. Ogni elemento $p \in P$ può contenere al più un agente e viene definito dalla tripla $\langle a_p, F_p, P_p \rangle$, dove $a_p \in A \cup \{\perp\}$ è l'agente che si trova in p ($\perp$ se non c'è nessun agente), $F_p \subset F$ è l'insieme dei campi attivi in p ($F_p = \emptyset$ se non ce n'è nessuno) e $P_p \subset P$ è l'insieme dei siti adiacenti a p .

I campi, attraverso la loro diffusione nello spazio, permettono la comunicazione a distanza tra agenti. Un campo con $f_\tau \in F$ viene emesso da un agente di tipo τ e si denota con il quartetto $\langle W_f, Diffusion_f, Compare_f, Compose_f \rangle$ dove con W_f , $Compare_f$, $Compose_f$ si denota l'insieme di valori che

il campo può assumere durante la sua diffusione nello spazio, $Diffusion_i$ è la funzione di distribuzione del campo su di un certo sito P e tiene conto dell'origine del campo e della sua intensità. Tale funzione può fornire più valori diversi perché tra il punto in cui prende origine il campo e il sito valutato possono esserci più cammini, anche di lunghezza diversa. $Compare_i$ è la funzione che restituisce un valore booleano a seconda che il valore del campo in un punto sia al di sopra o al di sotto della soglia di sensibilità dell'agente lì situato. Infine $Compose_i$ esprime come i campi si compongono tra di loro, ovviamente se sono dello stesso tipo, per ottenere un unico campo.

Un agente $a \in A$ viene identificato attraverso la tripla $\langle s, p, \tau \rangle$ dove $p \in P$ rappresenta il sito dello spazio in cui si trova, $s \in \Sigma_\tau$ denota lo stato in cui questi si trova e τ indica di che tipo è l'agente. Ogni tipo τ di agente viene caratterizzato da:

- Un insieme di stati Σ_τ che può assumere .
- Un insieme di azioni $Actions_\tau$ che può compiere.
- Una funzione $Perception_\tau$ che stabilisce, per ogni stato in cui può trovarsi, la sensibilità e il comportamento alla percezione di determinati campi .

Attraverso un'azione un agente specifica se e come questi cambierà il proprio stato o la propria posizione , come interagisce con gli altri agenti e come il contatto con gli agenti vicini è in grado di modificare il suo stato. In generale le motivazioni che conducono un agente a compiere un'azione sono due.

- Per modificare il proprio stato o la propria posizione. Si parla di azioni *intra-agent* in quanto non viene coinvolto nessun altro agente
- Per interagire con altri agenti. Si parla di azioni *inter-agent*.

Il modello prevede la presenza di quattro azioni:

- *Trigger*: determina come la percezione di eventuali campi determini una transizione di stato dell'agente.
- *Transport*: determina lo spostamento dell'agente nello spazio in base al tipo e all'intensità dei campi percepiti.
- *Emit*: determina l'emissione di campi da parte dell'agente in maniera asincrona.

- *Reaction*: del tutto analoga alla *Emit* da cui però si differenzia per il fatto che il tutto è rivolto ad uno o più agenti ben definiti che devono soddisfare al vincolo di contiguità rispetto all'agente che compie l'azione.

Rispetto ai CA, il modello proposto è molto più complesso ed articolato. Gli agenti vengono intesi come entità tra di loro non omogenee (esiste infatti il concetto di tipo di agente), dotate di un certo numero di stati. E' possibile modellare comportamenti più o meno evoluti, tenendo però conto del fatto che man mano che cresce la complessità il numero di stati necessari per rappresentare correttamente il tutto cresce in maniera piuttosto rapida. Il concetto di stato dell'agente è molto raffinato in quanto determina la sensibilità alla percezione di determinati tipi di campo e il proprio comportamento. Un agente è mobile e può spostarsi liberamente nello spazio e il suo comportamento è influenzato anche dalla posizione in cui questi si trova. Il cambiamento di stato avviene in maniera autonoma, non esiste una regola globale che stabilisce la stessa cosa per tutti gli agenti come invece avveniva per i CA. Più in generale, il comportamento di un agente si basa sul meccanismo *perception-deliberation-action*: si ha una percezione locale dell'ambiente grazie alla propagazione di campi, un processo di deliberation in cui si decide cosa fare tenendo conto del proprio stato, tipo e posizione nell'ambiente, e infine una fase in cui si esegue una determinata azione.

La comunicazione non è limitata a quella con i soli vicini ma può avvenire a distanza con tutti gli agenti che si trovano nello stesso ambiente attraverso la propagazione di campi. L'interazione può essere sia sincrona che asincrona. Quest'ultima è basata sul meccanismo di *emit-diffusion-perception*: si emette un campo con certe caratteristiche (intensità, contenuto informativo, eccetera), tale campo si propaga diffondendosi nell'ambiente e viene percepito dagli agenti con modalità che dipendono dallo stato in cui si trovano (ricordiamo che lo stato influenza la capacità di percezione). La comunicazione sincrona avviene solamente tra un agente ed i suoi vicini ed è diretta, non si fa uso di campi che si propagano.

Per quanto riguarda l'ambiente osserviamo che è teoricamente possibile modellare qualunque scenario. Analogamente a quanto accade per gli agenti,

non esiste nessun vincolo di omogeneità relativamente agli spazi ed ai siti di cui questi sono composti. Resta comunque il fatto che l'ambiente viene modellato in maniera discreta e può essere facilmente visualizzato come un insieme di griglie sovrapposte. L'interazione con gli agenti avviene in maniera indiretta in quanto pesantemente influenzati dalla rispettiva posizione spaziale. Inoltre la propagazione dei campi tiene conto della struttura spaziale e delle sue caratteristiche. E' possibile modellare la presenza di oggetti fisici attraverso la definizioni di particolari agenti in cui sono azzerate le caratteristiche tipiche, rendendoli quindi come entità statiche e puramente passive, ma comunque in grado di influenzare il comportamento delle entità attive, che indichiamo con il nome di agente, attraverso l'emissione di campi. Notiamo infatti che il concetto di agente secondo il modello Mmass presenta tutte le caratteristiche elencate nel capitolo precedente.

Il maggior difetto di un Mmass è dato dalla staticità della struttura spaziale e del numero di agenti che vi si trovano. Ciò non permette lo studio di sistemi dotati di dinamismo dal punto di visto logistico. E' però stata proposta un'estensione al modello attraverso la definizione delle seguenti azioni aggiuntive:

- AddSite(p): per aggiungere un sito.
- DeleteSite(p): per eliminare un sito (ovviamente non vi deve essere nessun agente posizionato).
- Clone(a): un agente crea un agente del suo stesso tipo specificandone stato e posizione.
- Clear(self): un agente si autoelimina.

Modelli Mmass trovano largo impiego nella ABS in quanto vengono forniti concetti ed un livello di astrazione tali da poter trattare lo studio di sistemi composti da entità indipendenti ed autonome che interagiscono tra di loro e con un ambiente più o meno complesso.

Un caso particolare di applicazione Mmass è il Situated Cellular Agents (SCA); si tratta di un Mmass con un solo ed unico spazio. In letteratura si trovano applicazioni basate su questo modello per lo studio del comportamento

delle folle. Gli interessi sono rivolti alla simulazione dei comportamenti emergenti in scenari come attraversamenti pedonali, stazioni della metropolitana, eccetera. Alcuni studi sono stati rivolti all'ottimizzazione di porte e via di fuga in locali pubblici come centri commerciali, cinema, biblioteche, eccetera.

Infine ricordiamo la possibilità di poter effettuare lo stesso tipo di simulazioni storicamente fatte servendosi di CA in quanto, diversamente da questi ultimi, si gode di una grande flessibilità nella definizione di agente ed interazione, il tutto perché un CA può essere inteso come un caso particolare e apprezzabilmente semplificato di MMAS.

5 Repast

RePast sta per “REcursive Porous Agent Simulation Toolkit”. Si tratta di un progetto open source sviluppato all’università di Chicago presso la Social Science Research. Attraverso l’uso di RePast è possibile effettuare simulazioni di sistemi di varia natura, in cui partecipano un certo numero di entità di un certo tipo che prendono il nome di agenti.

RePast non è l’unico toolkit che permette l’ABM, esistono altri framework, concettualmente molto simili, che sono, per citare i più famosi, Swarm e NetLogo. RePast si differenzia da questi due anche per il fatto che ha più di un’implementazione. Esistono infatti tre alternative per costruire simulazioni:

- Java.
- Py (utilizza un subset del linguaggio Python).
- .NET.

anche se, dal momento che RePast è stato concepito partendo da Java, la prima strada è la più sicura e supportata dalla comunità. L’uso di RePast non richiede lo studio di un’ulteriore linguaggio e quindi di imparare un’altra sintassi e semantica, è però richiesta una buona conoscenza di Java e comunque dei concetti della OOP. In questa tesi si farà riferimento alla sua versione Java.

RePast fornisce, oltre al motore per l’esecuzione delle simulazioni, tutta una serie di strumenti di base per la creazione e la gestione di queste:

- Tool per la visualizzazione grafica delle entità in gioco.
- Tool per la creazione di grafici.
- Possibilità di ottenere snapshots e movies della simulazione in corso.
- Librerie con varie funzionalità di carattere matematico e statistico.

5.1 Note tecniche

5.1.1 Installazione

RePast viene fornito per essere utilizzato su più di una piattaforma e cioè per sistemi Windows, Mac e Linux. Relativamente ai primi due viene fornito un installer, mentre per sistemi Linux occorre collocare il file `RePast.jar` all'interno del Classpath.

RePast si scarica liberamente da <http://repast.sourceforge.net> nella sezione download. Qui è possibile scegliere la piattaforma sul quale verrà utilizzato e, una volta effettuata tale selezione, il tipo di implementazione in quanto, oltre a quella in Java, è disponibile quella in .NET e quella in Py. Le versioni Java e .Net richiedono che nella macchina sia stato installato Java, mentre la versione .NET necessita del framework .NET. E' anche possibile scegliere la versione Suite che contiene tutte e tre le implementazioni. In tal caso è necessario che sulla macchina siano installati sia Java che il framework .NET. La versione suite impone l'installazione della versione .NET e permette di scegliere cosa fare per le restanti due.

5.1.2 Esecuzione

Una volta effettuata correttamente l'installazione, sulla piattaforma Windows viene aggiunta una voce nel menù di avvio che contiene da 1 a 3 voci a seconda di quante versioni sono state installate.

La versione Java presenta un link ad un file per l'avvio di RePast, una sottosezione contenente link agli esempi che vengono forniti e una seconda sottocartella con collegamenti alla documentazione. Si può avviare RePast facendo doppio click sul file “`repast.jar`” presente nella sottodirectory “RePast J” che si trova nella cartella principale “RePast 3” dove il tutto viene installato. E' altresì possibile lanciare RePast da linea di comando attraverso un'invocazione del tipo:

```
java -jar repast.jar
```

ipotizzando di trovarci nella sottodirectory “RePast J” presente nella directory dove il toolkit è stato installato

5.1.3 Librerie

Viene fornito tutto il codice sorgente sia di RePast che degli esempi. Provando a ricompilare i vari esempi si nota che in molti si fa uso del metodo `shuffle()` il quale richiede implicitamente la funzionalità di una libreria che però non viene fornita, più precisamente viene richiesta la presenza del tipo `cern.jet.random`. Tale libreria è reperibile sul web, si tratta anche in questo caso di codice open source liberamente scaricabile da <http://hoschek.web.cern.ch/hoschek/colt/download.htm>. Quello che serve è il file `colt.jar` che deve essere aggiunto al Java Build Path.

Ciò non accade per l’implementazione .NET dove suddetta libreria viene fornita.

5.2 Framework logico

5.2.1 Modellazione

RePast è Java based e quindi non è presente alcun costrutto relativo ai concetti della Agent Programming; ci si trova ad un livello implementativo squisitamente Object Oriented. Comportamento, proattività, goal, task, azioni, eccetera, non trovano nessun corrispettivo in fase di scrittura di una simulazione. Tutto viene in qualche modo emulato dagli strumenti messi a disposizione dalla OOP.

Una simulazione RePast è composta da almeno due diversi tipi di classe:

- **Agente:** descrive le caratteristiche ed il comportamento di un certo tipo di agenti. Possono essere presenti più classi di questo tipo, ognuna di esse modellerà un particolare tipo di agente partecipante alla simulazione.
- **Modello:** gestisce il setup della simulazione e quello che accade in ogni tic.

Solitamente a queste ne viene aggiunta una terza in cui vi è la descrizione dell'ambiente (**Space**) in cui vivono gli agenti. Una simulazione senza un ambiente in cui interagiscono gli agenti modellerebbe uno scenario estremamente semplice e demanderebbe l'evoluzione alla sola mutua interazione tra gli agenti in gioco.

Un Agente viene dunque emulato attraverso un ADT demandando il tutto a campi e metodi di classe. La stessa cosa avviene per l'ambiente. Un agente compirà determinate azioni e vi sarà un'eventuale mutua interazione con l'ambiente attraverso i relativi metodi. La stessa cosa accadrà tra agente e agente e quindi sia tra agenti dello stesso tipo che tra agenti modellisticamente diversi.

La model class deve obbedire a regole ben definite che saranno descritte nelle successive sezioni. Non esiste invece nessuna regola per l'architettura interna di un agente e dell'ambiente. Per esseri più precisi occorre dire che ci sono particolari regole implementative da seguire qualora si voglia una visualizzazione grafica di ciò che accade nell'evolvere della simulazione, ma queste non limitano la completa flessibilità offerta da RePast per quel che concerne la fase di definizione dell'ADT e dunque di modellazione della realtà.

Questa totale assenza di regole per quel che concerne la fase di definizione di agente ed ambiente consente potenzialmente la simulazione di ogni possibile realtà secondo le più disparate tecniche di modellazione (per esempio CA, MMASS, eccetera).

5.2.2 Simulazione

Una simulazione in ottica RePast consiste sostanzialmente di due passi. Il primo rappresenta uno stato di setup in cui si prepara la simulazione per la sua esecuzione, mentre nel secondo passo la simulazione viene eseguita.

Il running della simulazione avvenga in maniera discretizzata e cioè in stept temporali chiamati **tic**. Ciò permette di eseguire simulazioni passo-passo, in maniera controllata, consentendo all'utente di seguirne l'evoluzione sia globalmente che localmente, a livello di singola entità. In ogni tic i vari agenti eseguono un'azione, interagendo tra di loro e con l'ambiente, basandosi sui risultati delle azioni precedenti.

Tutto viene coordinato e gestito da uno **Scheduler**, una particolare entità che deve essere necessariamente presente per “dare vita” alla realtà modellata. Questa caratteristica di RePast è contraria ai principi dell’approccio a multiagente perché la messa in lista di un certo numero di azioni da eseguire ogni step temporale da un’entità coordinatrice ha come effetto quello di centralizzare tutto su tale entità e si perde la caratteristica di computazione distribuita.

5.3 Note sull’uso

5.3.1 La toolbar

Una volta che RePast viene lanciato si presenta la seguente control toolbar:



Figura 5-1 Repast: la toolbar

- **Loadmodel:** si apre una finestra che permette di caricare modelli di simulazione.

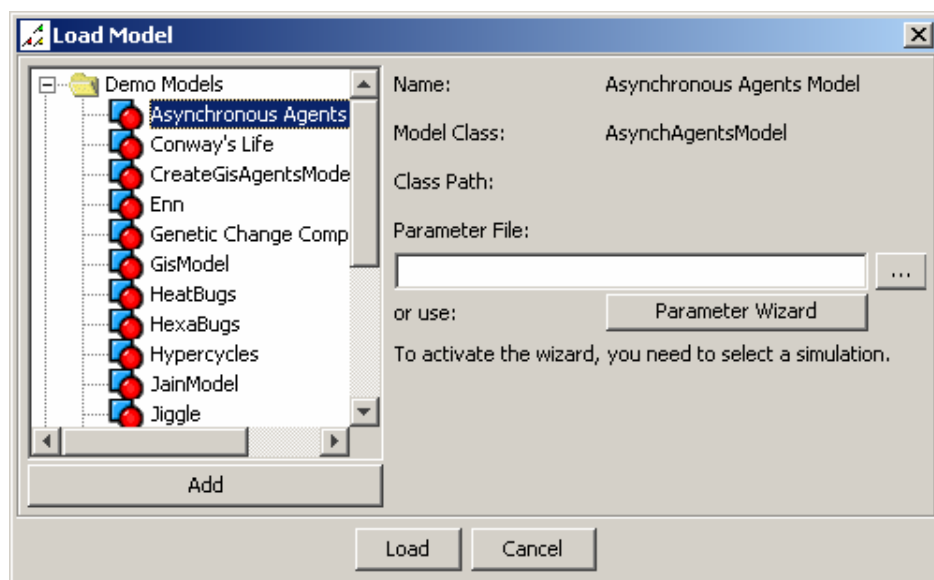


Figura 5-2 RePast: gestione dei modelli di simulazione

Quello che RePast chiede come input è un file .class che sia il risultato della compilazione di una classe Java contenente un modello di simulazione. Tali modelli devono rispettare delle regole implementative, se si tenta di dare in pasto a RePast un file .class qualunque verrà generato un messaggio di errore. Dallo snapshot si nota che i modelli sono suddivisi in varie cartelle, sostanzialmente una cartella con i modelli d'esempio forniti dal toolkit e due cartelle ove caricare i propri modelli. Per fare ciò i modelli devono essere aggiunti attraverso il pulsante Add (vedremo in seguito le caratteristiche che devono presentare i file per poter essere caricati). Ogni modello viene messo in esecuzione cliccando su Load. Al momento del caricamento di un modello è possibile specificare il valore iniziale di eventuali parametri di simulazione, che di solito sono presenti, come per esempio il numero iniziale di un certo tipo di agenti che partecipano alla simulazione.

- **Start Multi-Run:** per eseguire simulazioni in modalità batch, cioè senza la presenza di GUI grafiche.
- **Start:** per far partire una simulazione o per farla ripartire qualora fosse stata messa in pausa. La pressione di questo pulsante inizializza la simulazione e la esegue in tutti i suoi passi.
- **Stept:** per far ripartire una simulazione o per farla ripartire qualora fosse stata messa in pausa. A differenza del pulsante Start viene eseguito un solo passo della simulazione.
- **Initialize:** per inizializzare una simulazione. Tale pulsante è attivo solamente dopo che è stata caricata una simulazione. In seguito alla pressione di tale pulsante verrà inizializzata una nuova simulazione, e cioè verranno effettivamente istanziate le varie entità in gioco di cui si ha solo un riferimento fino a quel momento, la quale partirà solamente in seguito alla pressione del pulsante di start.
- **Stop:** per terminare la simulazione. In seguito alla pressione del pulsante di Stop diventano attivi i pulsanti di multi-run e quello di setup. La medesima simulazione può essere rieseguita attraverso la

pressione del pulsante setup che esegue codice atto a ripulire lo stato della simulazione da quanto fatto fino a quel momento, riportandolo in uno start state.

- **Pause:** per sospendere momentaneamente la simulazione. E' possibile farla ripartire normalmente come prima attraverso il pulsante Start o stept by stept attraverso il pulsante Stept.
- **Setup:** Il pulsante diventa attivo in seguito alla pressione del pulsante di Stop. Viene eseguito il codice presente all'interno del metodo Setup. Tale metodo deve essere presente altrimenti il modello viene riconosciuto come non valido all'atto del caricamento..
- **View Settings:** per visualizzare i parametri del modello.
- **Exit:** per chiudere RePast. Attivo e funzionante anche mentre è in corso una simulazione.

Riassumendo, per eseguire una simulazione, una volta che è stata correttamente caricata, si può cliccare sui pulsanti start, stept o initialize. Dopo che si è cliccato su initialize è necessario cliccare su stept o su start se si vuole eseguire, rispettivamente, uno o tutti i passi della simulazione. Qualora si decidesse di interrompere la simulazione attraverso il pulsante stop, per farla ripartire occorre cliccare sul pulsante setup e quindi resettarvi lo stato.

Generalmente la fase di begin avviene in passi, preoccupandosi del setup del modello, della Schedule e della GUI.

5.3.2 Creazione del modello da simulare

Come abbiamo accennato nelle precedenti sezioni, la costruzione di una simulazione avviene attraverso 3 tipi di classe.

5.3.2.1 Model Class

E' la classe che viene usata da RePast per il caricamento e la gestione di una simulazione. Verrà creato il modello della realtà da simulare attraverso oggetti che sono istanze per gli agenti di una o più agent class e per l'ambiente della space class. Oltre a ciò verranno gestite le varie azioni delle entità gioco, verrà eventualmente fornita una rappresentazione grafica loro e dell'ambiente

Le classi che contengono modelli devono implementare l'interfaccia `uchicago.src.sim.engine.SimModel`. Si tratta di una regola imposta da RePast in quanto nella model class si definisce la struttura della simulazione. Qualora tale regola venisse violata RePast non caricherebbe la relativa simulazione. Vengono fornite delle classi che implementano tale interfaccia:

- `uchicago.src.sim.engine.SimModelImpl`: è una classe astratta che implementa parzialmente i metodi dell'interfaccia sopra citata
- `uchicago.src.sim.engine.SimpleModel`: un'ulteriore specificazione della precedente classe ma a differenza di questa non è astratta e implementa tutti i metodi dell'interfaccia `SimModel`.

La model class che si intende creare può benissimo essere un sottotipo di una di queste due classi. E' però chiaro come alcuni dei metodi chiave per il setup e l'esecuzione della simulazione andranno ridefiniti in base al caso specifico.

Variabili di stato

Una model class sarà caratterizzata da un insieme di variabili il cui scopo è quello di rappresentare elementi della simulazione quali per esempio oggetti richiesti da RePast per l'esecuzione, oggetti per la visualizzazione grafica, entità coinvolte quali gli agenti e lo spazio in cui questi interagiscono. Un esempio di variabile necessaria per l'esecuzione dei vari passi della simulazione è quella adibita a contenere il riferimento ad un oggetto di tipo `Schedule`. Si può parlare dei riferimenti a strutture dati atte a contenere particolari tipi di entità che svolgono il ruolo di agenti in fase di simulazione e parametri di stato di cui parleremo in maniera dettagliata in seguito.

Setup

Il metodo **`public void setup()`** viene richiamato automaticamente quando si carica una model class, in seguito alla pressione del pulsanti setup e multi-run presenti nella toolbar. Viene preparato il modello per l'esecuzione di una nuova simulazione: tipicamente vengono deallocati, o eventualmente reistanziati a valori di default, gli agenti che partecipano alla simulazione ed eventuali entità GUI. La simulazione viene portata in uno start point da cui ripartire o partire nel caso in cui non si abbia effettuato ancora nessuna simulazione.

Indipendentemente da ciò, una volta avviato un passo della simulazione, verrà eseguito il codice presente nel metodo `begin`.

begin

Il metodo `public void begin()`: viene richiamato automaticamente in seguito alla pressione del pulsante `start` e, qualora la simulazione sia stata caricata ma non ancora avviata, dei pulsanti `stept` e `initialize`. All'interno di questo metodo viene scritto il codice relativo alle operazioni di inizializzazione, costruzione del modello e istanziamento di eventuali entità GUI. Una model classe deve rispettare certe regole implementative che sono dettate dai meccanismi della OOP, tuttavia è buona norma suddividere la fase di `begin` in tre fasi attraverso l'uso dei metodi:

```
public void buildModel(),  
public void buildSchedule()  
public void buildDisplay().
```

Si tratta di un'usanza tipica di chi scrive codice per simulazioni in RePast. Relativamente alle prime due fasi la classe `SimpleModel` presenta una sua implementazione attraverso i metodi `public void buildModel()` e `public void buildSchedule()`. Più in particolare il metodo `public void buildModel()` non esegue nessuna operazione a differenza di quanto invece accade per il metodo `public void buildSchedule()`.

Parametri di simulazione

Un modello di simulazione ha un certo numero di parametri che permettono di utilizzare la stessa model class per effettuare simulazioni in vari scenari di funzionamento, variando per esempio il numero di agenti coinvolti, eventuali loro strategie comportamentali, la quantità di risorse presente nell'ambiente virtuale e via discorrendo. Tali parametri verranno memorizzati attraverso opportune variabili. Affinché RePast li possa gestire correttamente, permettendo dunque di reinizializzare la simulazione senza doverla ricaricare, è necessario che i loro nomi vengano inseriti all'interno di un array di stringhe. Solitamente tale array viene chiamato `initParam`, si tratta di una convenzione che è bene rispettare. Occorre fare attenzione che il nome che si inserisce

nell'array coincida con il nome della variabile che rappresenta il parametro della model class. In caso contrario RePast non lo considererebbe e si sarebbe impossibilitati ad una sua variazione. Affinché tutto ciò sia possibile occorre che suddette operazioni vengano scritte all'interno del metodo **public String[] getInitParam()** (ecco perché è buona norma chiamare l'array di stringhe `initParam`). I vari parametri in gioco verranno visualizzati nella finestra dei settaggi, richiamabile, qualora non fosse presente, attraverso il pulsante View Settings nella toolbar di RePast.

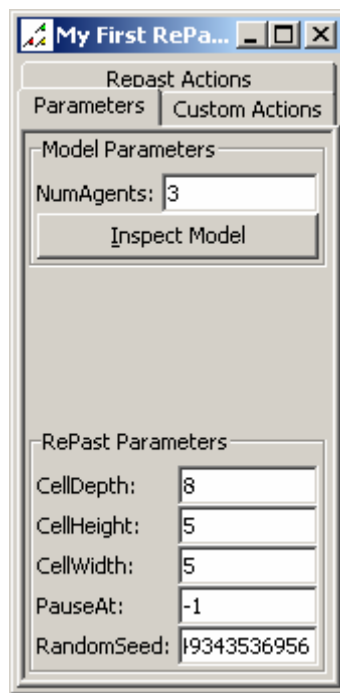


Figura 5-3 RePast: la finestra con i parametri della simulazione

Tali parametri dovranno essere visualizzati ed dovrà esserci la possibilità di modificarne il valore prima di iniziare e o riavviare la simulazione. A tal proposito sono necessari due metodi per ogni parametro in gioco: uno di **set** per modificarne il valore ed uno di **get** per la visualizzazione. Quando l'utente modifica il valore di un parametro viene automaticamente richiamato il relativo metodo di **set**, in maniera del tutto trasparente a chi ha scritto la model class della simulazione.

Senza il metodo di **get** il relativo parametro non verrà visualizzato e non sarà modificabile, inoltre l'inspector del modello lancerà un'eccezione che manderà in crash RePast.

Senza il metodo di **set** il relativo parametro verrà visualizzato ma non sarà modificabile, la stessa cosa accade se mancano entrambi i metodi.

Per modificare il valore di un parametro è sufficiente scrivere il nuovo valore nella relativa textbox. Qualora si scrivesse qualcosa di sintatticamente errato, come per esempio una stringa o un'espressione matematica dove è richiesto un valore intero, il sistema genera un'eccezione e il comportamento del sistema è imprevedibile a priori.

E' buona norma specificare un valore di default ai vari parametri in gioco, in caso contrario RePast assegnerebbe loro il valore di default 0 qualora si trattasse di un parametro di carattere numerico. E' usanza tipica servirsi di una variabile statica e non ereditabili da eventuali sottoclassi per ogni parametro di simulazione. Tale variabile finisce per essere una costante e i parametri di simulazioni vengono inizializzati al valore della relativa variabile

Schedule

Una `model class` dovrà avere almeno un oggetto di tipo `uchicago.src.sim.engine.Schedule` che serve per gestire l'esecuzione di azioni di tipo `BasicAction` ogni volta che viene scandito un clock di simulazione. Il metodo `public Schedule getSchedule()` impone la presenza di un'istanza di un oggetto di tale tipo.

Il metodo main

Tale metodo è, a differenza di quanto visto ad ora, opzionale. La simulazione può essere caricata attraverso il pulsante di load nella toolbar oppure attraverso linea di comando come argomento di `uchicago.src.sim.engine.SimInit`. All'interno di tale metodo è usanza scrivere codice molto simile al seguente:

```
public static void main(String[] args) {
    SimInit init = new SimInit();
    MyModel model = new MyModel();
    init.load(model, null, false);
}
```

Strumenti grafici per la visualizzazione

Una buona simulazione deve prevedere l'opportunità di fornire una rappresentazione grafica dell'ambiente e delle entità coinvolte nella

simulazione. Si fa uso degli strumenti messi a disposizione da RePast, in particolari dei costrutti presenti all'interno di **uchicago.src.sim.gui**. Solitamente ci si serve di `uchicago.src.sim.gui.DisplaySurface` che può essere pensato come un contenitore all'interno del quale vengono aggiunti elementi grafici per la visualizzazione. Si utilizza un elemento di natura grafica per ogni tipo di entità coinvolta. Per una descrizione più dettagliata ed esauriente occorre consultare le API e i vari esempi presenti.

Si nota che la visualizzazione è concettualmente legata allo spazio virtuale in cui vivono ed interagiscono le varie entità partecipanti alla simulazione ma non si tratta di un'operazione di modellazione. Tali operazioni devono essere esclusiva della space class, la relativa visualizzazione dovrà avvenire in fase di `buildDisplay()` e dunque nella model class.

Strumenti grafici per la visualizzazione

Una volta definita una rappresentazione grafica del sistema in esame è possibile, in fase di running della simulazione, visualizzare e modificare le caratteristiche (parametri e stato) delle varie entità attraverso l'uso di "sonde", comunemente chiamate probe.

Dal punto di vista concettuale ed ingegneristico, per quel che riguarda la struttura del sistema software, le modifiche da apportare sono veramente minime: è sufficiente utilizzare il metodo `addDisplayableProbeable` al posto di `addDisplayable`.

5.3.2.2 Space Class

Solitamente, negli scenari di simulazione più complessi ed articolati, lo spazio in cui vivono gli agenti viene modellato attraverso una classe. RePast non impone nessuna regola per la sua definizione.

La model class deve contenere un riferimento ad un'istanza di tale classe. Tale riferimento verrà assegnato al valore null in fase di **setup**. Occorre tenere infatti presente che all'interno di tale metodo deve essere scritto il codice ove si prepara il modello all'esecuzione di una nuova simulazione e quindi dove vengono deallocati agenti e GUI. La creazione vera e propria deve avvenire in fase di **begin** dove il modello viene reso operativo e funzionante. Dal momento che è buona norma suddividere le operazioni del metodo `begin` in tre metodi

che vengono da questo sequenzialmente eseguiti, tale operazione relativa alla creazione dello spazio viene collocata all'interno di `buildModel()`.

RePast mette a disposizione parecchi tipi di space attraverso il package **uchicago.src.sim.space**. In linea di principio si tratta di griglie bidimensionali con un numero di celle pari al prodotto delle dimensioni dei lati. Le varie entità che partecipano alla simulazione (e quindi oggetti di vario tipo) vengono solitamente collocate, in maniera più o meno random a seconda dei casi, nelle varie celle libere e durante la simulazione evolvono spostandosi da una all'altra, interagendo con le eventuali entità spaziali presenti. Ciò avviene servendosi dei vari metodi che vengono forniti dai vari costrutti del package e scrivendone dei propri in base alla modellazione che è stata effettuata.

La space class deve essere adibita alla modellazione dello spazio virtuale, non ci si preoccupa della visualizzazione che è un'operazione concettualmente estranea (si separa il contenuto dalla sua rappresentazione) e che viene quindi collocata nella model class, più precisamente nel metodo `buildDisplay()`.

5.3.2.3 Agent Class

Anche in questo caso RePast non impone nessuna regola per la scrittura di codice relativa agli agenti che partecipano alla simulazione i quali vengono modellati, tramite campi e metodi come qualunque ADT, all'interno di questa classe. La cosa è facilmente comprensibile in quanto gli agenti sono squisitamente simulation-specific, mentre la costruzione di un modello di simulazione deve essere regolata e seguire gli standard imposti dal tool. Occorre però precisare che, pur avendo libertà nell'implementazione di una agent class, è necessario anche qui seguire particolari accorgimenti se si vogliono rendere possibili operazioni quali la relativa visualizzazione grafica. Per esempio è necessario implementare i metodi dell'interfaccia `uchicago.src.sim.gui.Drawable`. RePast fornisce all'utente programmatore tutta una serie di librerie e strumenti per tale scopo. Per una descrizione più dettagliata ed esauriente occorre consultare le API e i vari esempi presenti.

5.3.3 Definizione di una simulazione

Una volta modellate le entità in gioco e lo spazio in cui queste interagiscono, una volta predisposte le varie GUI tutto è pronto. Se si prova a caricare la model class in esame RePast visualizzerà la sua toolbar e la finestra con i parametri della simulazione. Se si preme il pulsante initialize verrà eseguito il codice presente all'interno del metodo begin e quindi RePast provvederà all'istanziamento delle varie entità in gioco e delle GUI che forniranno una rappresentazione grafica dello stato iniziale da cui evolverà la simulazione. Se si tenta di avviare la simulazione, in uno o in tutti i passi attraverso i pulsanti stept e play, RePast genererà un messaggio di errore perché non è stata specificata nessuna azione da eseguire nei vari tic.

5.3.3.1 Lo scheduler

Nelle sezioni precedenti, mentre descrivevamo alcuni aspetti implementativi della model class, abbiamo accennato al fatto che tale classe doveva prevedere un campo privato atto a contenere un riferimento ad un oggetto di tipo `uchicago.src.sim.engine.Schedule`. Attraverso tale oggetto, che rappresenta in un certo senso il motore di RePast, verranno eseguite certe azioni (che dovranno essere opportunamente specificate) nei vari tic. Uno scheduler può potenzialmente eseguire le varie azioni con una granularità esprimibile in tic:

```
Schedule mySchedule = new Schedule(1); /*le azioni
verranno eseguite in ogni tic*/
Schedule anotherSchedule = new Schedule(5); /*le
azioni verranno eseguite ogni 5 tic*/
```

5.3.3.2 Le azioni

Occorre attenersi certe regole implementative per la definizione delle azioni che le varie parti devono compiere durante i tic della simulazione. Ogni azione dovrà scritta dentro di una inner class che andrà collocata all'interno del metodo `builSchedule()`. Tale inner class dovrà estendere la classe `uchicago.src.sim.engine.BasicAction` la quale contiene il metodo astratto `public void execute()` che dovrà dunque essere definito. E' all'interno di tale metodo che verrà scritto il codice relativo all'azione da compiere. Solitamente le azioni svolte sono implementata attraverso un opportuno mix ed un opportuna coordinazione delle azioni che agenti e ambiente sono in grado di

compiere. Affinché le varie azioni vengano eseguite dallo scheduler occorre dirglielo attraverso uno dei vari metodi che vengono messi a disposizione i quali forniscono varie funzionalità. E' possibile specificare che una determinata azione venga eseguita ogni tic, a partire da un certo tic, all'interno di un intervallo di tic, che cessi dopo un certo tic, eccetera.

Tutto ciò andrà fatto in fase di `buildSchedule()` e comunque all'interno del metodo di `begin()`.

6 TuCSoN

TuCSoN (Tuple Centres Spread Over the Network) nasce nel contesto della software engineering come modello e infrastruttura per sistemi di una certa complessità basati su agenti software.

Per agente si intende un'entità pro-attiva del sistema che compie autonomamente determinate azioni atte al raggiungimento di un qualche obiettivo. Tali azioni vengono eseguite in maniera asincrona e ciò implica una piena decentralizzazione e distribuzione del controllo, senza la presenza di un clock globale e di un qualche ente che esegue una qualche forma di scheduling. Per ogni agente il focus è sul comportamento osservabile, su quello che accade “esternamente” senza nessun interesse sulla sua struttura interna. Dal punto di vista implementativo non ci sono vincoli e specifiche, è possibile servirsi di diversi modelli e tecnologie.

L'astrazione di artefatto, su cui si basa TuCSoN, viene utilizzata come elemento base per modellare l'ambiente in cui vivono gli agenti del MAS. In maniera del tutto generale, un artefatto può essere inteso come un oggetto che incapsula una qualche funzione/utilità di cui si servono gli agenti, singolarmente o in gruppo, per l'esecuzione delle proprie attività. Si tratta dunque di componenti passivi a cui gli agenti accedono attraverso un'interfaccia d'uso che indica le operazioni a cui possono essere soggetti.

In TuCSoN l'interazione tra agenti avviene in maniera indiretta, attraverso particolari artefatti che vengono così chiamati **artefatti di coordinazione**. Essi forniscono meccanismi e mezzi per la comunicazione tra agenti, ma anche per adempiere a funzionalità più complesse quali la sincronizzazione e l'allocazione di task tra i vari agenti per il raggiungimento di un goal globale. TuCSoN è quindi un'**infrastruttura di coordinazione**, dove per infrastruttura di un sistema si intende un'invariante dell'environment, dunque persistente, in grado di fornire servizi e funzionalità non ottenibili da un singolo ente del sistema. Gli artefatti di coordinazione sono rappresentati da tuple centres (centri di tuple) che possono essere pensati come delle lavagne condivise dai vari agenti e su cui questi eseguono operazioni di inserimento e lettura di tuple secondo le seguenti primitive:

- *out(T)*: inserisce la tupla T.
- *in(TT)*: rimuove dallo spazio di tuple la tupla che fa match con il template TT. Se più di una tupla fa match ne viene scelta una a caso. Si dice che la *in(TT)* consuma una tupla. Si tratta di un'operazione sincrona perché l'agente rimane in attesa, dunque si blocca (operazione bloccante), fino a quando non è disponibile una tupla che fa match. Il risultato è sempre l'eliminazione di una tupla, il tempo di attesa varia in maniera imprevedibile.
- *rd(TT)*: legge, senza rimuovere, la tupla che fa match con il template TT. Anche questa primitiva è bloccante come la *in(TT)*.
- *inp(TT)*: come *in(TT)*, ma l'operazione fallisce se al momento non c'è nessuna tupla che fa match.
- *rdp(TT)*: come *rd(TT)*, ma l'operazione fallisce se al momento non c'è nessuna tupla che fa match.

Le tuple vengono codificate attraverso fatti Prolog, per esempio `sport(calcio)`. L'accesso è associativo, in altre parole una tupla viene cercata specificandone la struttura informativa con cui deve fare match e non il nome o l'indirizzo.

Ogni lavagna è reattiva. Ciò significa che è possibile specificare un comportamento del centro di tuple in seguito all'esecuzione di particolari operazioni. Tale comportamento viene codificato servendosi di ReSpecT (Reaction Specification Tuples), un linguaggio logico basato su Prolog, secondo fatti Prolog del tipo `reaction(Event,Body)`, dove *Body* rappresenta il set di operazioni da eseguire qualora si verificasse *Event*. Per esempio:

```
reaction(out(pp(X,Y)),  
         (out(p(X)), out(p(Y), in(pp(X,Y))) ) ).
```

in seguito all'operazione di inserimento di una tupla che fa match con il template `pp(X,Y)` il centro di tuple reagisce con l'inserimento di due nuove tuple `p(X)` e `p(Y)` seguito dalla rimozione della tupla inizialmente inserita.

Più in dettaglio, hanno effetto tutte le reazioni in cui sottoatomi di *Body* hanno successo, se almeno uno fallisce si effettua un'operazione di rollback, garantendo quindi la proprietà di atomicità. Si vede unicamente il risultato

finale, solo quello delle reazioni che vengono portate a termine con successo, e cioè quanto accade esternamente. Ogni centro di tuple può avere un vario numero di reaction, si possono poi verificare situazioni in cui viene eseguita un'operazione il cui template fa match con più reaction: in questi casi si assiste agli effetti di tutte secondo un ordine teoricamente non deterministico. Il comportamento di ogni lavagna non è fisso e cablato ma può essere modificato in base alle esigenze del momento. Infine ogni centro di tuple gode della caratteristica di ispezionabilità a run-time. Ciò significa che è possibile osservare ed analizzare il comportamento dei vari tuple centres senza dover arrestare il sistema: i vantaggi sono chiaramente comprensibili.

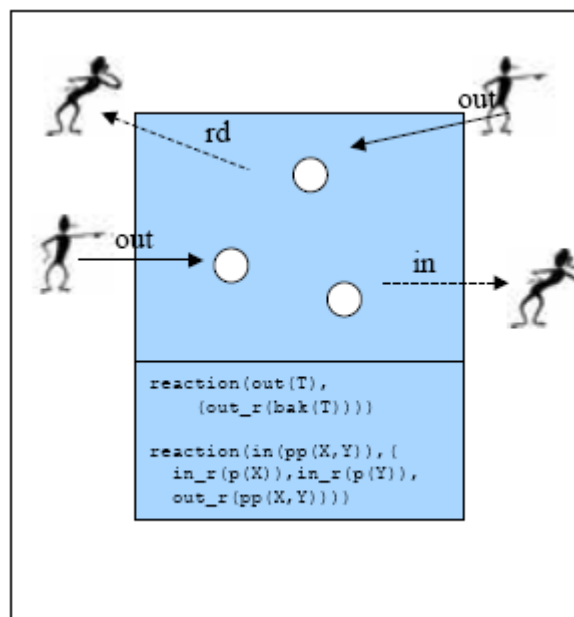


Figura 6-1 Centro di tuple TuCSoN

I vari centri di tuple si trovano all'interno di nodi TuCSoN (detti coordination contexts):

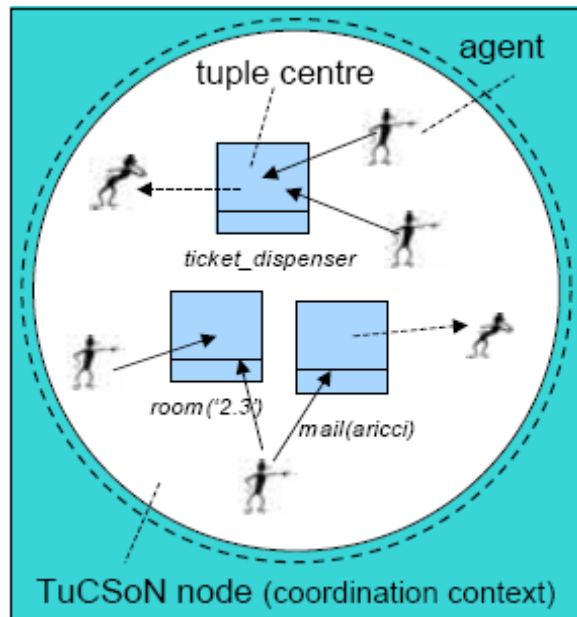


Figura 6-2 Nodo TuCSoN

In definitiva TuCSoN fornisce uno spazio di coordinazione che è dato dall'insieme dei vari nodi che sono sparsi nella rete:

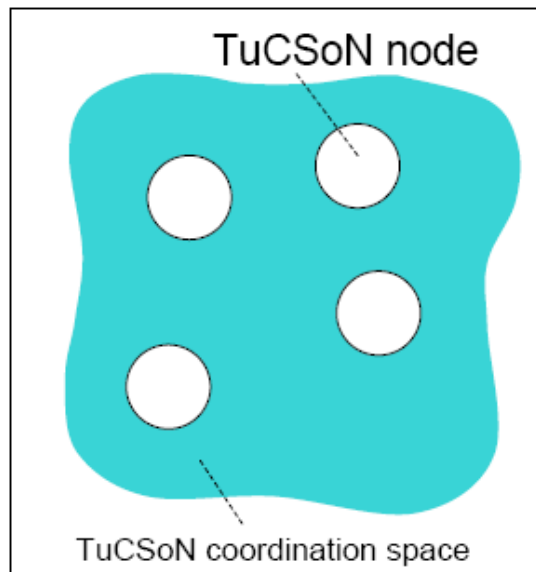


Figura 6-3 TuCSoN

Un'importante caratteristica di TuCSoN è l'ortogonalità tra la computazione a livello di agenti del MAS e la coordinazione. Ciò permette di usare la stessa infrastruttura di coordinazione per diversi MAS e di costruire sistemi in cui convivono ed interagiscono agenti modellati e implementati in diversi modi.

6.1 A&A: un framework per la modellizzazione e la simulazione basato su TuCSoN

In [20] viene presentato A&A, un framework basato sull'uso di agenti ed artefatti per la costruzione di sistemi ad agenti. Il modello concettuale prevede sostanzialmente la presenza di tre tipi di astrazioni:

- Agenti: per rappresentare le entità pro-attive del sistema che compiono determinate azioni in maniera autonoma.
- Artefatti: per rappresentare le entità passive del sistema di cui si servono gli agenti nel compiere le loro azioni e tutte le parti dell'ambiente che non possono essere considerate come agenti.
- Workspaces: sono contenitori logici di agenti ed artefatti, vengono utilizzati per definire il concetto di spazialità e di località: da un lato gli agenti possono interagire esclusivamente con gli artefatti presenti all'interno del loro workspace, da un altro lato agenti e artefatti possono appartenere a più di un workspace, tutto è relativo alla realtà da modellare.

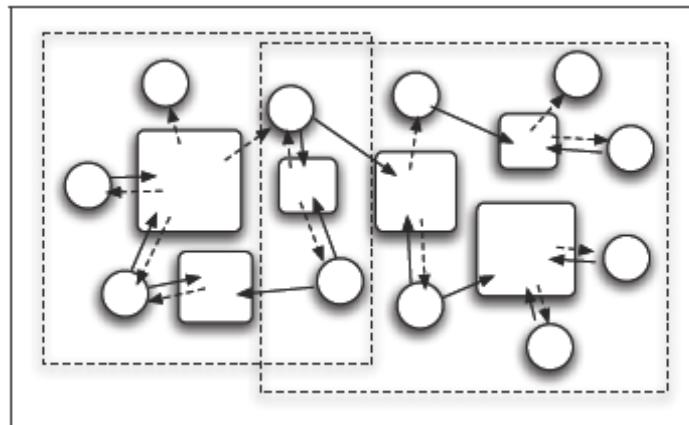


Figura 6-4 A&A: agenti (cerchi), artefatti (quadrati) e workspaces (aree tratteggiate)

Sempre in [20], per quel che riguarda l'implementazione ci si serve di TuCSoN. Esattamente come in A&A, TuCSoN considera gli agenti come le entità autonome del sistema e non fornisce alcun vincolo e alcuna specifica relativamente alla loro costruzione. Ogni artefatto trova il suo corrispettivo in

un centro di tuple e i vari agenti possono interagirvi attraverso un'interfaccia che viene modellata in base alle operazioni messe a disposizione da TuCSoN (out, rd, in, rdp, inp). Un workspace viene interpretato secondo uno o più nodi TuCSoN, ognuno dei quali contiene un insieme (possibilmente dinamico) di agenti e centri di tuple.

Stando a quanto detto, una simulazione consiste nel modellare il sistema in termini di agenti, workspaces e artefatti di coordinazione, specificando per questi ultimi eventuali comportamenti reattivi da tenere in seguito al verificarsi di certe operazioni. L'interazione tra gli agenti, e quindi non solo la comunicazione, viene mediata dai vari centri di tuple.

7 RePast v.s. TuCSoN

Iniziamo col dire che RePast è stato concepito con l'intento di creare un toolkit per la simulazione di sistemi complessi secondo l'approccio ABS, mentre TuCSoN è nato dalla computer engineering come infrastruttura per sistemi ad agenti; per tale motivo RePast mette a disposizione strumenti e utilità di vario genere che semplificano e facilitano il lavoro di creazione delle simulazioni, soprattutto per quel che concerne l'aspetto grafico e la definizione di modelli per lo sviluppo di particolari algoritmi, quali per esempio quelli genetici. Inoltre RePast può essere integrato in maniera del tutto automatica in versioni sia open source che commerciali di strumenti Geographical Information Systems [5] (GIS, si tratta di sistemi informativi applicati nell'ambito della geografia a livello di centri urbani).

Si tratta in entrambi i casi di due toolkit a cui possono avere accesso utenti con buone competenze nell'ambito della programmazione, più in particolare RePast impone la conoscenza dei principi della OOP in quanto la creazione di una simulazione consta nella scrittura di codice Java (in questa tesi si è fatto riferimento alla versione Java), mentre per poter utilizzare TuCSoN è necessario conoscere Prolog dal momento che ReSpecT, il linguaggio utilizzato per descrivere il comportamento dei centri di tuple, è un suo derivato. Ci troviamo dunque di fronte a strumenti a cui possono avere accesso utenti dotati di una certa familiarità con la programmazione; da questo punto di vista non si ha a che fare con toolkit di più alto livello quali per esempio NetLogo, dove la modellizzazione avviene in buona parte per via grafica, stile drag&drop, e in cui occorre imparare un linguaggio dedicato di gran lunga più semplice e accessibile all'utente comune.

Per quel che riguarda gli agenti, entrambi i toolkit non impongono nessuna specifica, ma si limitano a considerarli come entità autonome che compiono determinare azioni. Dal punto di vista implementativo TuCSoN è quello che lascia più libertà in quanto permette di utilizzare qualunque tecnologia. RePast invece cerca di emulare l'idea di agente attraverso i concetti ed i meccanismi della OOP, senza fornire nessuna regola e nessun vincolo di carattere implementativo, eccetto qualche specifica da seguire qualora si volesse fare

uso degli strumenti messi a disposizione per particolari operazioni, quali per esempio la visualizzazione grafica.

Relativamente all'ambiente, RePast continua a lasciare la più ampia libertà permettendone addirittura l'omissione, mentre in TuCSoN ci si serve di artefatti e quindi di entità che dal punto di vista concettuale non dispongono delle caratteristiche di autonomia e pro-attività. Diversamente da quanto accade per RePast, in TuCSoN non si può non considerare l'ambiente perchè il concetto di infrastruttura su cui esso è basato viene inteso come invariante dell'ambiente in cui vivono ed interagiscono le entità, ma soprattutto, se in RePast l'interazione può avvenire anche in maniera diretta, in TuCSoN si passa attraverso gli artefatti di coordinazione.

La natura di TuCSoN rende immediata la modellizzazione e la computazione distribuita. I varie centri di tuple si trovano all'interno di nodi TuCSoN che possono essere presenti su più calcolatori collegati in rete. E' quindi sufficiente che nei vari host sia installato un nodo TuCSoN. RePast fornisce funzionalità analoghe attraverso particolari librerie.

In RePast la definizione di una simulazione avviene all'interno di una `model class` in cui si specificano le entità in gioco e l'ambiente (che però può anche non essere presente) in cui queste vivono ed interagiscono. RePast impone delle regole per la costruzione di una classe di questo tipo. In TuCSoN non accade nulla di tutto ciò, il sistema viene modellato in maniera distribuita, attraverso la definizione dei vari agenti, dei centri di tuple e la relativa programmazione dei comportamenti da tenere in seguito al verificarsi di determinate azioni.

Affinché una simulazione possa essere eseguita RePast impone la presenza di un'istanza della classe `Scheduler`; tale entità si occupa dello scheduling delle azioni compiute dagli agenti. Lo scheduling viene gestito in maniera *time-driven*, secondo step temporali chiamati *tic*, inoltre l'utente ha il potere di decidere la granularità temporale e di definire i momenti/intervalli di tempo in cui i partecipanti compiono determinate azioni. L'esecuzione della simulazione viene quindi centralizzata su questa particolare entità e così facendo si perde in distribuzione e delocalizzazione, andando contro ai principi della computazione distribuita. Sotto questo punto di vista TuCSoN è completamente decentralizzato e distribuito, non esiste il concetto di clock

globale e la nozione di tempo globale: un agente ha un suo tempo locale ed esegue le proprie azioni in maniera asincrona a quelle degli altri. L'esecuzione della simulazioni avviene in maniera "spontanea", attraverso le azioni compiute dai vari agenti e le eventuali reazioni dei centri di tuple.

In RePast è possibile modificare i parametri degli agenti e di alcune parti dell'ambiente in fase di simulation-running attraverso gli strumenti e le funzionalità messe a disposizione (probe). In altre parole tutto il sistema è completamente ispezionabile. In TuCSoN avviene qualcosa di analogo, ma solo relativamente ai centri di tuple che possono essere ispezionati e riprogrammati a run-time.

8 Conclusioni

Con questa tesi ho effettuato una panoramica sui vari approcci della simulazione, focalizzando l'attenzione su quella basata sugli agenti.

Relativamente a questa tecnica sono state descritte alcune metodologie, cercando di capire per ognuna di esse cosa si intende per agente e i corrispettivi risvolti implementativi.

Successivamente sono stati analizzati e descritti RePast, un tool per la costruzione di software per la simulazione ad agenti, e Tucson, un'infrastruttura di coordinazione per MAS, cercando di capire per entrambi il modello concettuale e l'idea di agente su cui ci si è basati per poi effettuare un confronto.

9 Bibliografia

- [1] Parisi D.: *Se laboratorio è nel computer le scienze hanno un'arma in più*, Telèma, 5, pp. 28-30. (1999)
- [2] Gilbert N., Terna P.: *How to build and use agent-based models in social science*, Mind & Society, no. 1, pp. 57-72. (2000)
- [3] Edmonds B.: *Simulation and Complexity. How they can relate*, presentato al workshop su "Virtual Worlds of Precision – Computer based Simulations in the Natural and Social Sciences", Oxford, (2003)
- [4] Campos A. M. C., Canuto A. M. P., Fernandes J. H. C.: *Towards a methodology for developing agent-based simulations: the MASim methodology*
- [5] Macal C. M., North M. J.: *Tutorial on agent-based modelling and simulation*, Proceedings of the 2005 Winter Simulation Conference
- [6] Axtell R.: *Why agents? On the varied motivations for agent computing in the social sciences*, Working paper n°17 (2000), Centre on Social and Economic Dynamics, Brookings Institution, Washington
- [7] Drogoul A., Vanbergue D., Meurisse T.: *Multi-agent based simulation: where are the agents?*.
- [8] Paolucci P.: *Agent-based simulation in manufacturing: what is it for?*, AIROnews n.2/3/4 Summer/Autumn/Winter 2004
- [9] Parunak H., Savit R., Riolo R. L.: *Agent-base modeling vs. equation-based modeling: a case study and a user's guide*, Proceedings of multi-agent systems and agent-based simulation (MABS'98), 10-25, Springer, LNAI 1534, 1998
- [10] Davidsson P.: *Multi agent based simulation: beyond social simulation*

- [11] *Agent-based and system dynamics modelling: a call for cross study and joint research*, Priceedings of the 34th Hawaii international conference of system science (2001)
- [12] Allouche J. P., Courbage M., Skordev G.: *Notes on cellular automata*
- [13] Torbjorn H.: *Cellular Automata, not merely a new kind of science*
- [14] Orenstein P., Marantz Z., Goodman D.: *Application of cellular automata to modelling mobility and radio communication in wireless networks*
- [15] Gaungly N., Sikdar B. K., Deutsch A., Canright G., Chaudhuri P. P.: *A survey on cellular automata*
- [16] Rennard J. P.: *Introduction to Cellular Automata*, J. P. Rennard Phd (12/2000)
- [17] Bandini S., Manzoni S., Simone C.: *Situated cellular agents in non-uniform spaces*
- [18] Bandini S., Manzoni S., Simone C.: *Dealing with space in multi-agent systems: a model for situated MAS*
- [19] Bandini S., Federici M. L., Vizzari G.: *A methodology for crowd modelling with situated cellular agents*
- [20] Montagna S., Ricci A., Omicini A.: *A&A for modelling and engineering simulations in system biology: a first framework base on TuCSoN*
- [21] <http://repast.sourceforge.net>

- [22] Davidsson P.: *Agent Based Social Simulation: a computer science view*, Journal of Artificial Societies and Social Simulation, 5, 2002 available at <http://jasss.soc.surrey.ac.uk/5/1/7.html>.