
Università degli Studi di Salerno

Facoltà di Scienze Matematiche Fisiche e Naturali

---

Tesi di Laurea di I livello in

Informatica

# Progettazione assistita di simulazioni agent-based: l'architettura di Agent Modeling Platform

**Relatore**
Prof. Vittorio Scarano

**Candidato**
Francesco Farina
Matricola 0512100694

---

Anno Accademico 2012-2013

# Dediche e ringraziamenti

*Ai miei genitori, a mia sorella Giovanna, alla mia famiglia ed a tutti i miei amici per aver sempre creduto in me.*

*Al mio caro amico e collega Marco, a Daniele e a Iole che hanno vissuto con me questi intensi mesi di tirocinio.*

*A tutti i membri dell'ISISLab, in particolare al mio relatore Prof. Vittorio Scarano, al Prof. Gennaro Cordasco ed al Dott. Luca Vicidomini.*

Questa tesi è stata sviluppata in 

# Indice

# Capitolo 1

# Introduzione

Nell'ultimo decennio, lo studio dei sistemi multiagente (ABM) ha acquisito grande attenzione ed è diventato uno strumento sempre più diffuso in molti ambiti di ricerca come biologia, economia, ecologia, scienze sociali.
I sistemi multiagente nascono con la necessità di rappresentare determinate realtà di interesse, in cui in un ambiente vengono situati degli agenti, ovvero entità autonome in grado di prendere decisioni ed interagire tra loro secondo una logica definita dal sistema. La simulazione basata su agenti è un software capace di modellare i comportamenti di un sistema e viene utilizzata per analizzare sistemi complessi, come il controllo del traffico, l'infezione di virus, il comportamento di una rete, senza la necessità di doverli costruire.
Vivendo in un mondo molto complesso anche i sistemi da analizzare risultano complessi, e costruire prototipi potrebbe essere oneroso. Grazie alla continua evoluzione del settore informatico, si possiedono le capacità di modellare e prevedere approssimativamente l'evoluzione di sistemi molto complessi su larga scala. La modellazione può essere suddivisa in tre fasi:

- definizione della struttura degli agenti e dei reciproci rapporti, che si possono instaurare;
- creazione del modello che rappresenta la realtà mediante gli agenti;
- implementazione software dei modelli di simulazione ed analisi dei risultati emergenti.

Mentre la prima e la seconda fase possono essere portate a termine da uno o più modellatori, la terza necessita di almeno uno sviluppatore. Ciò è dovuto al fatto che l'implementazione software richiede conoscenze nel campo della programmazione orientata agli oggetti, dei toolkit di simulazione e tutto

quel che concerne un processo di sviluppo software.
Tali conoscenze non rientrano nel bagaglio culturale della maggior parte dei modellatori, per cui avere un nuovo modo di implementare il modello potrebbe essere di supporto a tutta la comunità dell'*agent-based modeling*.

Questa idea è alla base della **Agent Modeling Platform**, progetto Eclipse open source rilasciato sotto licenza EPL (Eclipse Public Lincense), sviluppato da Miles Parker, attualmente Senior Solutions Architect / Information Systems Business Analyst presso Tasktop Technologies e President and Chief Software Architect presso Metascape.

La piattaforma AMP ha lo scopo di offrire strumenti atti alla rappresentazione, creazione, modifica, generazione, esecuzione e visualizzazione di modelli basati su agenti. Essa offre la possibilità, grazie a tali strumenti, di costruire un modello utilizzando componenti grafiche e quindi astraendosi dalla reale implementazione software. In questo modo, l'utente della piattaforma crea un *metamodello* che rispecchia il prodotto delle prime due fasi della modellazione, senza dover prendersi carico dei vari problemi legati al codice, anzi potendone completamente ignorare l'effettiva implementazione software.
Il metamodello, una volta ultimato, può essere eseguito all'interno dell'ambiente Eclipse, così come l'analisi dei dati prodotti. Questo approccio fornisce al modellatore un ambiente unico e completo, in cui svolgere tutte le operazioni che sarebbero state effettuate attraverso la scrittura di software apposito.

L'obiettivo di questo lavoro di tesi è quello di descrivere ed analizzare dettagliatamente la *Agent Modeling Platform*, la sua architettura e le numerose tecnologie su cui si fonda, evidenziando in particolar modo le conoscenze acquisite, necessarie alla comprensione delle caratteristiche e delle problematiche riscontrate nello studio di questo ampio ed elaborato progetto.

# Capitolo 2

# Alcuni concetti base

Per comprendere a fondo questo lavoro di tesi è necessario introdurre alcuni concetti, come la simulazione ad agenti, i modelli comportamentali, le diverse piattaforme e linguaggi, che consentono di costruire simulazioni ad agenti.

## 2.1 Modellazione basata su agenti

Vivendo in un mondo in continua evoluzione, anche i sistemi da analizzare, quali un impianto elettrico, il comportamento di una rete, diventano sempre più complessi e quindi sempre più difficili da simulare. L'Agent-based modeling è la risposta alla necessità di rappresentare tali sistemi in maniera realistica. La modellazione è un processo di astrazione, che comporta necessariamente numerose approssimazioni basate su delle assunzioni, la cui validità deve continuamente essere messa in discussione. Un modello è un'approssimazione della realtà, dalla quale vengono scelti i componenti principali del sistema, magari decidendo di escludere dalla rappresentazione quei componenti che sono ritenuti irrilevanti per lo studio del fenomeno di interesse, quindi individuando i principali meccanismi di funzionamento, scegliendo una rappresentazione matematico-algoritmica di tali meccanismi, ecc. Si parla di modellazione basata su agenti, ma che cosa è un agente? *un agente è una componente autonoma, che rappresenta gli oggetti fisici o logici all'interno del sistema, in grado di agire al fine di raggiungere i propri obiettivi, ed essere in grado di interagire con altri agenti, quando non possiede conoscenze e competenze per raggiungere da solo i suoi obiettivi.* [1]
Le caratteristiche di un agente sono:

- **Autonomia**, ossia la capacità del gruppo di operare senza direttive esterne, nel senso che esso è in grado di effettuare le proprie scelte e prendere le proprie decisioni.
- **Capacità di comunicare**, ovvero interagire con altri agenti, magari cooperando nel perseguimento degli obiettivi comuni, scambiando informazioni e conoscenze. La maggior parte dei protocolli di interazione fra agenti consistono nel contendere lo spazio ed evitare collisioni.
- **Reattività**, un agente può interagire con l'ambiente che lo contiene. Ogni agente valuta singolarmente la sua situazione e prende decisioni sulla base di una serie di regole.
- **Proattività**, un agente può avere degli specifici obiettivi, che ne guidano il comportamento. Generando così, eventi nell'ambiente circostante, tali da poter iniziare delle interazioni con altri agenti, coordinare le attività di differenti agenti, stimolandoli a produrre determinate risposte.
- **Nozioni mentali**, un agente può possedere la capacità di apprendere ed adattare il proprio comportamento, in base alle esperienze accumulate. Tale apprendimento consente di modellare sistemi dinamici, che evolvono nel tempo. In seguito a tale processo gli agenti sono in grado di modificare il loro stato interno ed adattarsi alla nuova situazione modificando l'intero sistema.

## 2.2 Simulazione ad agenti

Le simulazioni basate su agenti sono uno strumento di grande utilità per lo studio di sistemi complessi legati agli esseri viventi. E' possibile creare un mondo virtuale basato su agenti autonomi, che interagiscono tra loro. Impostando uno stato iniziale, si può osservare come evolve il sistema, e come i fenomeni emergenti spesso si rivelino di una complessità straordinaria. La simulazione è uno strumento che permette di stimare, ad esempio, l'evoluzione di una popolazione batterica o la diffusione di un'epidemia. Esistono differenti approcci alla simulazione:

- *bottom-up*, in cui la simulazione ha inizio dagli attori del modello;
- *top-down*, dove si cerca di modellare la situazione da simulare nel complessivo, mediante l'utilizzo di mezzi matematici, quali, ad esempio, le equazioni differenziali.

### 2.2.1 Esempio di simulazione

Le simulazione ad agenti, come presentato in precedenza, modellano e permettono di studiare e stimare situazioni reali, in cui il sistema analizzato evolve nel tempo. Per comprendere maggiormente l'uso pratico di quest'ultime, verrà analizzato *Boids* di Craig Reynolds. Da Wikipedia: [2] *Boids è un programma di vita artificiale sviluppato nel 1986, che simula il comportamento di uno stormo di uccelli. Il nome si riferisce a birds-like object, ma la sua pronuncia evoca quella di uccello nell'accento stereotipato di New York.* Simulare uno stormo di uccelli è come simulare un'elaborazione di un sistema di particelle, con gli uccelli al posto delle particelle. Come la maggior parte delle simulazioni di intelligenza artificiale, Boids è un esempio di comportamento emergente; la complessità di Boids nasce dall'interazione dei singoli agenti (i boids, in questo caso), che rispettano un'insieme di semplici regole, cosicchè ogni agente possa scegliere un proprio percorso. Ognuno di essi è considerato come un attore indipendente, che naviga secondo la propria percezione locale di un ambiente dinamico, sfruttando le

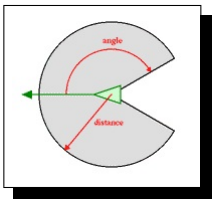


Figura 2.1: Vicinato di un flocker

leggi della fisica simulata che ne guidano il movimento, ed un insieme di comportamenti programmati da chi ha progettato la simulazione. Ogni boid ha un accesso diretto alla descrizione geometrica dell'intera scena, ma lo stormo richiede che esso reagisca solo ai suoi vicini presenti in un certo raggio attorno a lui. Il vicinato è caratterizzato da una distanza (avendo come origine la posizione del boid) e da un angolo (vedi fig. 2.1), misurato dalla direzione di volo del boid. I boids che si trovano al di fuori di quest'area non infuiscono sul comportamento del boid in oggetto. Il vicinato potrebbe essere considerato un modello di percezione limitata, ma è probabilmente più corretto pensare ad esso, come la regione in cui un gruppo di boids infuenza il cammino di un altro. Il modello di base è composto da tre semplici comportamenti di guida, che descrivono come un singolo boid si possa spostare, in base alle posizioni ed alla velocità dei boids del suo vicinato:

- **separazione**: virare per evitare un sovraffollamento locale;
- **allineamento**: virare verso la direzione media dei boids del vicinato;
- **coesione**: muoversi verso la posizione media (il centro di massa) dei boids del vicinato.

Un modello più complesso può essere realizzato con l'aggiunta di nuove regole comportamentali, come quelle di evitare ostacoli e cercare obiettivi. Il movimento dei Boids può essere caratterizzato sia caotico che ordinato. Comportamenti inattesi, come la scissione dello stormo e la successiva riunione dopo aver evitato ostacoli, possono essere considerati emergenti. La cosa forse più sconcertante è la forte impressione di un controllo intenzionale e centralizzato. Eppure tutte le prove indicano che il movimento dello stormo è semplicemente il risultato complessivo delle azioni individuali dei boids, ognuno dei quali interagisce esclusivamente sulla base della propria percezione locale del mondo.

# Capitolo 3

# Piattaforme per simulazioni agent-based

Negli ultimi anni, la diffusione e l'adozione di approcci alla modellazione ed alla simulazione basata su agenti, ha spinto la comunità dell'*agent-based modeling* a sviluppare diversi toolkit che consentono lo sviluppo di applicazioni basate su agenti. Tali toolkit forniscono astrazioni e meccanismi per costruire definizioni di agenti e degli ambienti in cui essi operano, in maniera da favorire la loro interazione. Inoltre mettono a disposizione del modellatore funzionalità addizionali come la gestione della simulazione, la visualizzazione, il monitoraggio e l'acquisizione dei dati riguardanti le dinamiche simulate.

## 3.1 Principali categorie

Sempre più toolkit emergono nel panorama della modellazione agent-based, ed ognuno porta con sè un ventaglio di caratteristiche, che lo differenzia dagli altri. In generale, essi possono essere raggruppati in categorie a seconda delle caratteristiche comuni.

### 3.1.1 General Purpose Framework

Una prima categoria può essere rappresentata dall'insieme di piattaforme che forniscono *general purpose frameworks*, in cui gli agenti sono considerati come astrazioni "passive", cioè una sorta di strutture dati manipolate da uno specifico processo di simulazione. In generale si tratta di toolkit, in cui i parametri fondamentali del modello vengono costruiti mediante una

interfaccia grafica, mentre il comportamento degli agenti viene specificato mediante il linguaggio, che il toolkit stesso mette a disposizione. Tale linguaggio presenta un elevato livello d'astrazione, similmente alla lingua parlata, e caratteristiche fondamentali per l'utilizzo da parte di un modellatore di simulazioni, che non abbia particolari skill di programmazione. Il toolkit più popolare appartenente a questa categoria è senz'altro **NetLogo** (vedi sez. 3.2).

### 3.1.2 General Purpose Programming Framework

Un'altra categoria comprende quei toolkit che racchiudono gli stessi concetti dei precedenti, fornendo strumenti di supporto al modellatore che vanno dalla gestione della simulazione alla visualizzazione grafica, ma tali strumenti sono basati su linguaggi di programmazione 'general purpose'. Nella maggior parte dei casi, la natura *object oriented* dei linguaggi alla base dei toolkit, offre la possibilità di aggiungere elementi computazionali, che rendono gli agenti più autonomi e molto più vicini alla definizione teorica di agente, offrendo in questo modo l'incapsulamento di stato ed azioni di quest'ultimo all'interno della sua definizione.
La scelta di utilizzare un linguaggio di programmazione *general purpose* presenta vantaggi e svantaggi: da un lato, rende la scelta di questo ambiente meno probabile da parte di modellatori, che non hanno esperienza nel campo della programmazione, mentre dall'altro semplifica l'integrazione di librerie esterne preesistenti. Probabilmente **Swarm** [3] rappresenta l'antenato comune di tutti i membri di questa categoria, tra cui i più diffusi e degni di nota sono **Ascape** (vedi sez. 3.4), **MASON** (vedi sez. 3.3) e **Repast** (vedi sez. 3.5).

### 3.1.3 Graphic Modeling Framework

L'ultima categoria riunisce le piattaforme che hanno lo scopo di ridurre la distanza tra i modelli basati su agenti e le loro implementazioni. Alcune di queste sono caratterizzati da un'interfaccia "punta e clicca" capace di creare definizioni del comportamento degli agenti e dell'implementazione del modello. In questo genere di toolkit, tutti i componenti di un modello di simulazione vengono astratti e resi come elementi grafici, che il modellatore può utilizzare per costruire il suo modello personalizzato. Tali piattaforme consentono di costruire modelli di simulazione, anche a chi non ha mai avuto esperienze di programmazione, grazie alla forte intuitività delle interfacce. Per ottenere tale caratteristica il livello di astrazione deve essere elevato,

fattore che porta inevitabilmente alla perdita di dettaglio, nei confronti delle piattaforme citate in precedenza. La **Agent Modeling Platform** (AMP) appartiene a questa categoria.

## 3.2 NetLogo

NetLogo nasce nel 1999 dallo sviluppo di StarLogo di cui mantiene gli elementi base quali i tipi di agenti (turtles, patches, observer) e l'intuitività della rappresentazione, ma aggiunge nuove caratteristiche significative ed innovazioni nel linguaggio e nell'interfaccia utente.
Unisce la caratteristica di essere uno strumento di programmazione molto potente a quella di possedere un'interfaccia semplice e di immediato utilizzo, grazie anche ad un efficace supporto grafico e diverse modalità di interazione per la programmazione.
NetLogo è un ambiente di programmazione di modelli per la simulazione di fenomeni sociali e naturali. É particolarmente adatto alla modellazione di sistemi complessi che evolvono nel tempo. Chi programma può dare istruzioni a centinaia o migliaia di agenti indipendenti, ma che agiscono simultaneamente e ciò rende possibile esplorare la connessione tra i comportamenti degli individui, e le strutture che emergono dalla loro interazione.
Facilitano l'utilizzo di NetLogo, informazioni e tutorials molto estesi e una Models Library [5], che è un'ampia collezione di simulazioni pronte per essere usate e che possono costituire lo spunto per la costruzione di modelli più complessi a partire dall'utilizzo e la modifica di meccanismi base. Queste simulazioni fanno riferimento a diverse aree delle scienze sociali e naturali quali la biologia, la medicina, la chimica, la fisica, la matematica, l'economia e la sociologia.

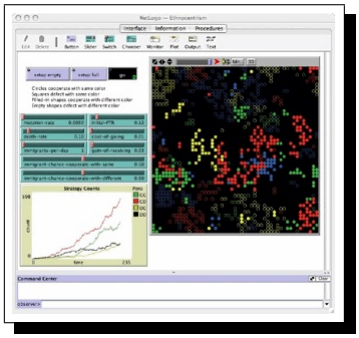

Figura 3.1: Esecuzione di una simulazione in NetLogo

## 3.3 MASON

MASON è un toolkit open source progettato per effettuare simulazioni basate su agenti, utilizzando una singola macchina, ed è in grado di supportare simulazioni aventi un elevato numero di agenti. MASON è modulare e consistente, presenta un alto grado di separazione e di indipendenza tra ogni componente del sistema.
Inoltre tale toolkit è scritto in Java, in modo da poter collocarsi in ambienti eterogenei, con un'occhio di riguardo verso l'efficienza, in quanto Java possiede la reputazione di essere relativamente lento. Tale obiettivo viene raggiunto, sostituendo le librerie considerate la fonte dei problemi circa la lentezza, con altre equivalenti e più efficienti.
MASON fornisce inoltre un'interfaccia grafica, che permette la visualizzazione in tempo reale di simulazioni in esecuzione.

### 3.3.1 Architettura

Il toolkit è scritto utilizzando il paradigma MVC (Model-View-Controller). Tale pattern è basato sulla separazione dei lavori che i diversi componenti software devono compiere. In figura 3.2 viene mostrata la struttura del pattern.

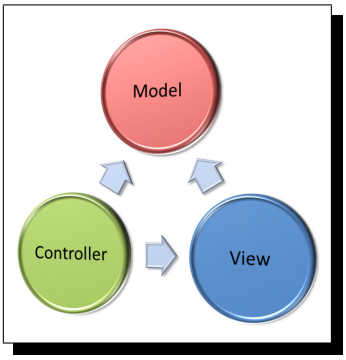


Figura 3.2: Struttura del pattern MVC

Il *model layer* fornisce i metodi per accedere ai dati utili all'applicazione, il *view layer* visualizza i dati contenuti nel model e si occupa dell'interazione con utenti e agenti ed infine il *controller layer* riceve i comandi dell'utente (in genere attraverso il view) e li attua modificando lo stato degli altri due

componenti. In Mason i tre livelli svolgono le seguenti funzionalità (come da Figura 3.3) :

- ***Model Layer***: è il core di MASON, non presenta dipendenze dal layer di visualizzazione. È costituito da classi, che comprendono una coda ad eventi discreti ed un insieme di campi, che vengono utilizzati per contenere oggetti ed associarli a locazioni specifiche.
- ***Visualization Layer***: permette sia la visualizzazione che la manipolazione del modello.
- ***Utility Layer***: costituito da widgets per l'interfaccia utente e programmi di utilità di altro genere.

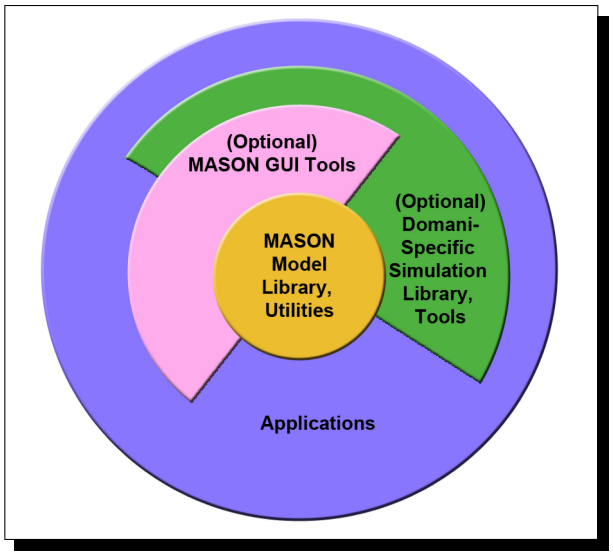


Figura 3.3: Livelli di architettura di MASON

Siccome il Model Layer è separato dal Visualization Layer, le simulazioni in MASON possono essere eseguite con o senza l'ausilio di interfaccia grafica, ed è possibile anche passare da una modalità ad un'altra durante l'esecuzione della simulazione.

### 3.3.2 Struttura di una generica simulazione

La creazione di un modello di simulazione è un passo essenziale per lo sviluppo di una simulazione basata su agenti. In MASON il modello di simulazione viene definito come sottoclasse *sim.engine.SimState* e tale istanza viene mantenuta come stato globale dell'intera simulazione.
**SimState** contiene **uno schedulatore ad eventi discreti**, istanza della classe
*sim.engine.Schedule*, consente di processare gli agenti per dar loro la possibilità di comportarsi in una data maniera. Inoltre, è fondamentale specificare un'entità che rappresenti la definizione degli agenti, con i relativi comportamenti. Tale entità deve implementare l'interfaccia *sim.engine.Steppable*, in particolare il metodo *step()*, nel quale viene esplicitato il comportamento che l'agente possiede. L'esecuzione di una simulazione MASON consiste di un ciclo elaborato, in cui vengono compiuti i seguenti passi:

- viene creata un'istanza della classe rappresentante il modello di simulazione ed inizializza il generatore di numeri pseudo-casuali, in base ad un particolare numero, chiamato *seed*;
- viene invocata il metodo *start()* dell'istanza creata, per effettuare inizializzare il modello;
- ripetutamente lo schedulatore ad eventi discreti richiama il metodo *step*(SimState state), in modo che gli agenti memorizzati nello schedulatore siano processati;
- il ciclo termina quando lo schedulatore non ha più agenti da elaborare.

Le simulazioni in MASON possono essere eseguite con o senza l'ausilio di interfaccia grafica, come detto in precedenza, in quanto il modello è separato dalla visualizzazione, per cui sono presenti due classi primarie per ciascuno:

- la classe del modello principale è sottoclasse di *sim.engine.SimState*;
- la classe incaricata della visualizzazione è sottoclasse di *sim.engine.GUIState*.

**GUIState** è composto dai seguenti oggetti:

- un *display*, che si occupa della rappresentazione del campo, *field portrayal*, in cui gli agenti operano, permettendone la visualizzazione e l'ispezione;
- un oggetto in grado di creare e mostrare una console, componente grafico che funga da gestore grafico della simulazione, consentendo all'utente di controllare il ciclo di vita della simulazione;
- lo stato del modello (SimState) della corrente simulazione.

Di seguito viene riportato un diagramma delle classi di una generica simulazione scritta in MASON:

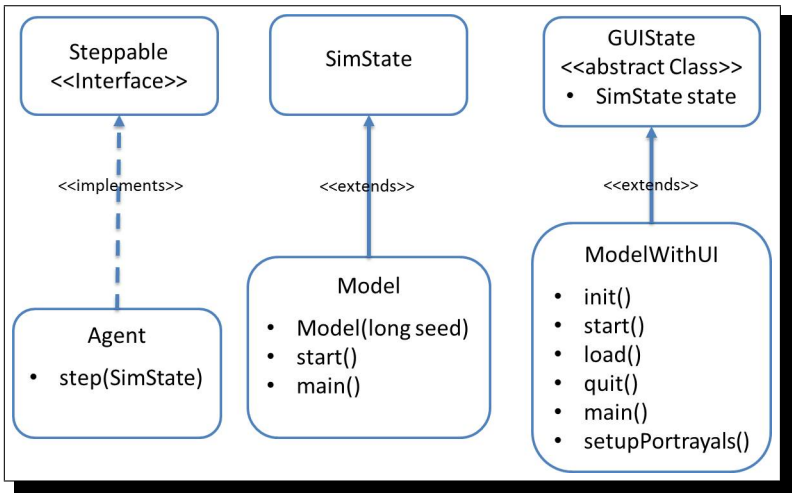


Figura 3.4: Diagramma delle classi di una generica simulazione MASON

## 3.4 Ascape

Ascape è un framework scritto in linguaggio Java per lo sviluppo e l'esplorazione di sofisticati modelli basati su agenti. Il nome "ASCAPE" introduce due concetti fondamentali alla base del framework: agenti e *scapes*.
Il termine *scape* deriva da Sugarscape, il modello implementato da Joshua M. Epstein e Robert L. Axtell, in cui si realizzano le idee espresse nel loro famoso libro *Growing Artificial Societies* [6]. Tale termine fa riferimento ad un territorio in cui gli agenti vivono.

In Ascape, l'idea di uno scape è stata generalizzata fino ad includere qualsiasi collezione di dati. Ascape promuove l'idea di dare pari importanza allo scape tanto quanto all'agente. Tale idea alla base del framework, semplifica l'implementazione e l'esecuzione di comportamenti, la raccolta di statistiche e la componibilità dei modelli, integrando le astrazioni di spazio e vicinato. Lo sviluppatore non necessita, quindi, di concentrarsi sulla realizzazione dell'infrastruttura della simulazione, ma bensì egli può lavorare sin da subito sul modello.
Ascape rafforza l'idea di scape, infatti ogni modello possiede uno scape "radice". Uno scape fornisce un modo di organizzare agenti in un insieme ed è *esso stesso considerato un agente*, in maniera tale da poter costruire gerarchie di scape, offrendo ai modelli la capacità di essere composti con altri.

Sebbene i modelli Ascape siano scritti in Java, il framework, avvalendosi della progettazione basata sul pattern *Model-View-Controller*, mette a disposizione dei modellatori, che non hanno conoscenze di tale linguaggio, strumenti grafici (vedi Fig. 3.5) per apportare variazioni al modello intuitivamente e dinamicamente, durante l'esecuzione di quest'ultimo. Inoltre, l'interfaccia utente permette di catturare filmati dell'esecuzione della simulazione e di osservare diagrammi in tempo reale: tali diagrammi si basano sui dati statistici collezionati automaticamente dal framework, esportabili in fogli di calcolo.

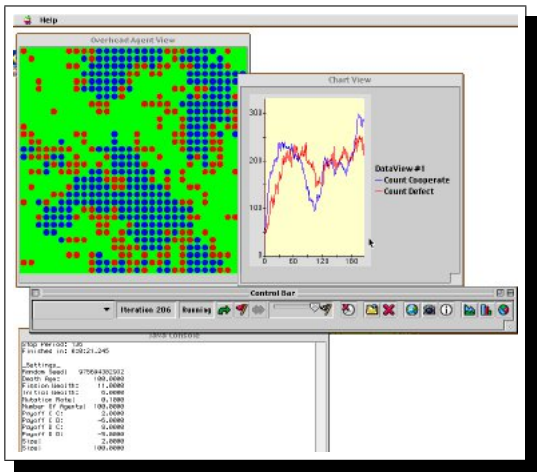


Figura 3.5: Ambiente di esecuzione di un modello Ascape

## 3.5 Repast Simphony

Il *REcursive Porous Agent Simulation Toolkit* è stato sviluppato nel 2000 presso l'Università di Chicago ed in seguito reso, dall'Argonne National Laboratory, un'infrastruttura software in grado di supportare la *"rapid social science discovery"*. Versioni successive hanno introdotto nel sistema la capacità di gestire lo sviluppo di applicazioni di simulazione ad agenti su larga scala.
Repast è ora sotto il controllo della *Repast Organization for Architecture and Design* ( ROAD ), associazione no-profit, i cui membri dirigenti fanno parte di organizzazioni governative, accademiche ed industriali. Il principale mantainer dello sviluppo di Repast è l'Argonne National Laboratory, ma molti gruppi sono attivi nello sviluppo e nel 2011, grazie alla Google Summer of Code, più di 20 modelli di simulazione sono stati aggiunti alla suite.

Repast Simphony è stato progettato ponendo particolare attenzione alle astrazioni fondamentali. Il codice risultante costruisce un'architettura orientata ai plug-in altamente modulare. Ciò consente di sostituire quando necessario le componenti individuali, come ad esempio il networking, il logging o lo scheduling.
La piattaforma è formata da più layer, permettendo così di implementare ognuno di questi con particolari plug-in, al fine di poter personalizzare la piattaforma e mantenere la compatibilità con versioni precedenti.

I modelli di simulazione possono essere implementati in Java: Repast offre strumenti di supporto avanzati allo sviluppatore, dalle strutture che astraggono lo spazio in cui vivono gli agenti, alla raccolta di dati automatica durante l'esecuzione della simulazione. Inoltre, particolarmente importante, è la possibilità di ottenere proiezioni geografiche grazie al Geographic Information System (GIS), in combinazione alle librerie OpenGL che permettono di avere una resa grafica accurata sia in 2D che in 3D (vedi Fig. 3.6).

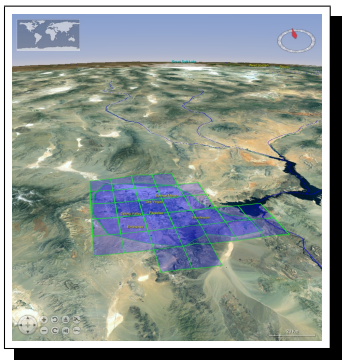

Figura 3.6: Integrazione GIS in Repast Simphony

# Capitolo 4

# Eclipse

Gli ambienti di sviluppo integrati vengono in aiuto del modellatore per la realizzazione di simulazioni basate su agenti. In generale gli ambienti di sviluppo integrati, **IDE** (*Integrated Development Environment*), sono progettati per massimizzare la produttività di chi sviluppa software, fornendo un unico programma che ingloba tutte le funzionalità offerte, attraverso interfacce grafiche.
Tra i più diffusi IDE specifici per le piattaforme, *Microsoft Visual Studio* sviluppato da Microsoft, supporta diversi tipi di linguaggi, quali C, C++, etc., e permette la realizzazione di applicazioni, siti e servizi web; mentre *Xcode*, sviluppato da Apple, consente di sviluppare software per Mac OS X ed iOS.
Per quanto concerne gli IDE non legati alla piattaforma fra i principali, **Eclipse**, ambiente di sviluppo integrato multi-linguaggio e multipiattaforma, viene utilizzato per la produzione di software. Completo IDE, per il linguaggio Java (JDT, *Java Development Tools*) e C++ (CDT, *C/C++ Development Tools*), per la gestione di XML(*eXtensible Markup Language*), Javascript, PHP ed inoltre per la progettazione grafica di GUI (*Graphical User Interface*) per un'applicazione JAVA (Eclipse VE, *Visual Editor*), e non solo.
É progettato per funzionare su più sistemi operativi, fornendo una robusta integrazione con ogni software sottostante. Tale programma è scritto in linguaggio Java, ma anziché basare la sua GUI su Swing, il toolkit grafico di *Sun Microsystems*, si appoggia a SWT, librerie di nuova concezione che conferiscono ad Eclipse un'elevata reattività.

## 4.1 Cenni storici

I leader nel settore industriale, IBM, Borland, MERANT, QNX Software Systems, Rational Software, Red Hat, SuSE, TogetherSoft e Webgain, formarono nel novembre 2001 l'iniziale *eclipse.org*.
Alla fine del 2003, il consorzio ebbe una crescita di oltre 80 membri ed all'inizio del successivo anno, il consorzio Eclipse annunciò la riorganizzazione del progetto in una società senza scopi di lucro. Quello che originariamente era il consorzio, formatosi quando IBM rilasciò la piattaforma Eclipse come *Open Source*, è divenuto un organismo indipendente, che attualmente guida l'evoluzione della piattaforma a vantaggio di chi fornisce software e degli utenti finali.
Tale società no-profit, la *Eclipse Foundation*, fu appunto creata per consentire a venditori neutrali ed alla community dell'open source di affacciarsi sul progetto Eclipse.
Tutta la tecnologia ed il codice sorgente fornito e sviluppato da questa comunità è reso disponibile, tramite la *Eclipse Public License* (EPL).

## 4.2 Architettura

La piattaforma Eclipse è basata principalmente sul concetto di *plug-in*. Un plug-in è un componente software, che aggiunge specifiche funzionalità ad un sistema. La piattaforma è strutturata in sottosistemi, che vengono implementati in uno o più plug-in, aggiungendo così comportamenti alla stessa, attraverso i cosiddetti *extension point*. Tali punti di estensione consentono a chi sviluppa software sia di aggiungere nuove caratteristiche, che personalizzare e specializzare le componenti esistenti.
Tale modularità permette alla piattaforma di rendere trasparente la complessità nella gestione di diversi ambienti d'esecuzione e sistemi operativi, consentendo agli sviluppatori di concentrarsi sul loro compito, senza preoccuparsi di tali problematiche.
Di seguito viene mostrata un'immagine raffigurante l'architettura di Eclipse.

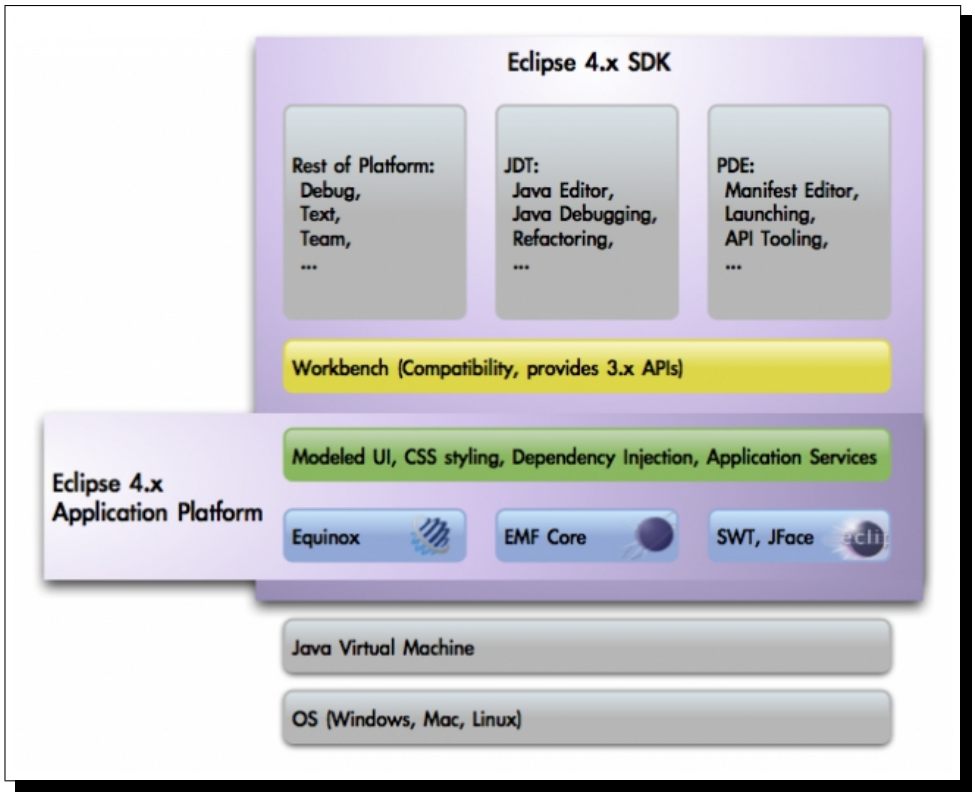


Figura 4.1: L'architettura di Eclipse

L'ecosistema di Eclipse è formato da diverse componenti. Il *Workbench* è l'elemento grafico più familiare per gli utenti dell'Eclipse platform, il quale fornisce le strutture che organizzano come Eclipse debba apparire all'utente. Tale componente consiste di prospettive, viste, editors ed utilizza API di SWT (*Standard Widget Toolkit*) e JFace.
Il *Plug-in Development Environment* (PDE) fornisce strumenti per creare, sviluppare e testare plug-in. Inoltre offre un insieme di strumenti per lo sviluppo di componenti OSGi, *Open Service Gateway initiative*.
Il framework OSGi rappresenta un sistema modulare, dinamico ed orientato ai servizi, che definisce la gestione del modello del ciclo di vita del software, moduli ed ambiente d'esecuzione. L'implementazione Eclipse di tale framework OSGi è **Equinox**.


Figura 4.2: Il logo della OSGi Alliance e di Equinox

Equinox nacque, come progetto Eclipse, con l'obiettivo di sostituire il modello a componenti di Eclipse con uno già esistente, oltre a fornire il supporto per plug-in dinamici. La scelta ricadde su OSGi, che aggiunse, in particolare, un sistema di versioni semantico per la gestione di dipendenze, fornendo la possibilità di specificare versioni di librerie, che fossero compatibili per diverse componenti, ottenendo così un quadro di modularità che mancava ad Eclipse.
Il *Java Development Tools* (JDT) consente agli utenti di scrivere, compilare, testare, effettuare il debug e modificare programmi scritti nel linguaggio di programmazione, *Java*. Il JDT si avvale di molti dei punti di estensione della piattaforma e di framework inclusi nel PDE. Java Development Tools può essere considerato come un insieme di plug-in, che aggiunge specifici comportamenti, viste ed editor Java a modelli di piattaforme generiche.

## 4.3 Estendibilità: Plug-in

Oltre ad essere uno dei più diffusi IDE, Eclipse è una piattaforma facilmente estendibile, attraverso componenti indipendenti e collegabili, chiamati *Eclipse plug-in*. Il meccanismo di base di estendibilità in Eclipse si basa sul concetto che i nuovi plug-in possono aggiungere nuove caratteristiche ai plug-in esistenti.
Nel dettaglio, un Eclipse plug-in consiste di codice Java, racchiuso all'interno di una libreria, denominata *Jar*. Con il passaggio a OSGi, un Eclipse plug-in divenne noto come *OSGi bundle*. Un plugin ed un bundle forniscono entrambi un sottoinsieme modulare di funzionalità, descritto con metadati attraverso un file apposito.
Le dipendenze, i pacchetti da esportare, le estensioni ed i punti di estensione vengono descritti in un file scritto in linguaggio XML. In tale file, *plugin.xml*,

sono contenute tutte le informazioni necessarie all'esecuzione del plugin e alla modalità con cui il modulo deve essere richiamato. Con il passaggio a OSGi bundle le informazioni aggiuntive vengono descritte in un OSGi manifest file, chiamato *MANIFEST.MF*. Per supportare questo cambiamento, PDE ha fornito un nuovo editor all'interno di Eclipse.

Il compilatore Java non è in grado di comprendere bundle e manifest OSGi, il PDE si assume il compito di analizzare le informazioni contenute all'interno delle componenti OSGi. Inoltre si occupa di incapsulare i bundle in pacchetti ed assegnargli un determinato percorso, in questa maniera si ottiene modularità e le risorse necessarie ai bundle vengono allocate solamente al momento del bisogno. Tali pacchetti hanno un ciclo di vita, gestito da determinate classi Java, gli *Activator*, i quali fornisco dei metodi per installazione, avviamento, arresto e disinstallazione.

# Capitolo 5

# Agent Modeling Platform

La **Agent Modeling Platform** è un progetto Eclipse open source, rilasciato con licenza pubblica Eclipse (EPL), il cui obiettivo è quello di fornire framework estensibili e strumenti atti alla rappresentazione, creazione, modifica, generazione, esecuzione e visualizzazione di modelli basati su agenti. I modelli basati su agenti condividono alcune caratteristiche con i modelli orientati agli oggetti, a cui aggiungono proprietà peculiari come:

- **Spazialità**: i modelli hanno espliciti ambienti in cui gli agenti interagiscono. Un ambiente non deve necessariamente deve un paesaggio fisico, può essere rappresentato da relazioni spaziali che includono reti sociali o posizioni interne alla logica di un sistema.
- **Temporalità**: col passare del tempo, i modelli mutano il loro stato. Nella maggior parte dei casi il tempo è discretizzato.
- **Autonomia**: i comportamenti degli agenti sono attivati indipendentemente da cosa fanno gli altri agenti.
- **Eterogeneità**: gli agenti possono condividere le definizioni dei comportamenti, ma i loro stati e le azioni apparenti sono distinti.
- **Collettività**: i modelli contengono vaste comunità di agenti, che esibiscono comportamenti collaborativi e competitivi.
- **Emersività**: gli agenti mostrano macro-comportamenti che non sarebbero stati dedotti dalle definizioni dei micro-comportamenti di ciascuno di essi.


Figura 5.1: Logo del progetto AMP

## 5.1 Dominio applicativo di AMP

Lo scopo principale su cui la Agent Modeling Platform ripone maggior attenzione è l'**Agent Based Modeling** (ABM), tecnica usata per esplorare fenomeni complessi in molti ambiti, tra cui l'economia, le scienze sociali, la bio-medicina, l'ecologia e le operazioni di business. La piattaforma Eclipse fornisce alcune funzionalità che la rendono ideale per una ABM Platform.

Nonostante l'Agent Based Modeling sia l'obiettivo principale di **AMP**, ciò non confina le possibilità della piattaforma solamente a quell'ambito. Molti tipi di oggetti godono delle proprietà descritte in precedenza, rendendo possibile l'utilizzo degli agenti in tanti altri contesti. Quindi, il supporto che AMP offre alla *meta-modellazione* è potenzialmente adatto o estendibile ad un gran numero di approcci al di là dell'ABM: regole di business, generiche interazioni tra oggetti, ed i tradizionali modelli ad eventi discreti. Allo stesso modo, il supporto all'esecuzione ed alla resa grafica offerto da AMP può essere sfruttato per modellare sistemi naturali.

Dunque, il dominio applicativo della piattaforma è senza dubbio vasto, ciò è confermato anche dal grande interesse scatenato all'interno della comunità dell'Agent Based Modeling.

## 5.2 Ambienti di simulazione supportati da AMP

La Agent Modeling Platform è un progetto Eclipse, ciò significa che il suo sviluppo è suddiviso in fasi così come definito nel processo di sviluppo [9] (vedi Fig. 5.2). Attualmente AMP è nella fase di Incubation, fase in cui viene realizzato il progetto, implementate le sue tecnologie ed aggiunte le funzionalità fino ad ottenere un progetto open source perfettamente funzionante.

Essendo ancora in sviluppo, la piattaforma offre supporto per alcuni tra i

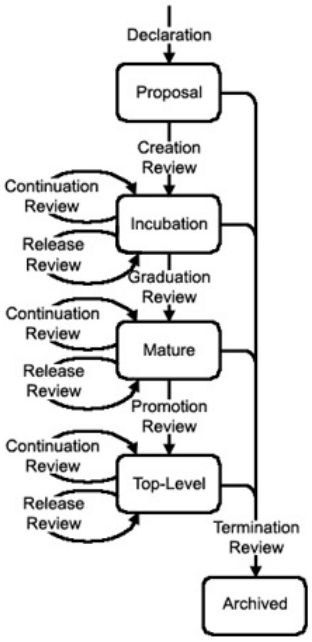


Figura 5.2: Fasi del processo di sviluppo di un progetto Eclipse

maggiori ambienti di simulazione basata su agenti: *Ascape, Escape e Repast Simphony.*

### 5.2.1 Escape

**Escape** è un ambiente di modellazione *agent-based* open source basato su *Ascape*. Costituito da un framework di esecuzione ed uno di visualizzazione grafica, che hanno consentito una semplice integrazione all'interno dell'ecosistema di AMP, anche grazie alle potenzialità di Eclipse. Inoltre esso fornisce un completo insieme di servizi per l'esecuzione e l'esplorazione dei modelli, così come un gran numero di modelli d'esempio. L'implementazione AMP di Escape permette ai modellatori, una volta realizzato il modello, di eseguirlo all'interno dell'ambiente di sviluppo stesso, senza doversi affidare ad ambienti esterni.

## 5.3 Installazione in Eclipse

La piattaforma Eclipse fonda la sua architettura sul concetto di plug-in. Tali plug-in possono essere scoperti ed installati tramite gli strumenti offerti dalla piattaforma:

- **Eclipse Marketplace**: la *Eclipse Foundation* mette a disposizione degli utenti un sito web chiamato, Eclipse Marketplace (http://marketplace.eclipse.org/), che fornisce l'elenco completo di tutte le soluzioni basate su Eclipse. Ogni soluzione può specificare il proprio repository per facilitarne l'installazione.
  In questo modo, gli utenti Eclipse hanno un unico catalogo in cui ricercare la soluzione desiderata, per poterla installare in maniera semplice ed immediata. Inoltre, la piattaforma integra un Marketplace Client per interfacciarsi con il sito web, rendendo l'esperienza di ricerca ed installazione sempre più completa ed integrata.

- **Update Sites**: la piattaforma Eclipse fornisce un meccanismo di installazione e gestione degli aggiornamenti chiamato *Eclipse p2*. Questo meccanismo permette di aggiornare le applicazioni Eclipse e di installare nuove funzionalità.
  Queste caratteristiche sono destinate sia ad intere applicazioni, che a singole componenti, denominate *installable units*. Le unità possono essere raggruppate in *p2 repository*, che vengono identificati da una URI (Uniform Resource Identifier) e possono far riferimento a risorse presenti su file system locale o su webserver. L'Eclipse p2 repository è anche conosciuto con il termine *"Update Site"*.

### 5.3.1 Installazione da Update Sites

La modalità di installazione consigliata all'utente della Agent Modeling Platform è quella tramite Update Sites, compatibile con la release di Eclipse versione 3.7.2 Indigo (2011).
AMP mette a disposizione update sites per ogni "ramo" del suo processo di sviluppo, il principale riguarda le release stabili http://download.eclipse.org/amp/updates/releases. Per utilizzarlo è necessario recarsi all'apposita sezione in Eclipse: *Help* → *Install New Software*, aggiungere l'url agli update sites disponibili e selezionare, quindi, le componenti desiderate.

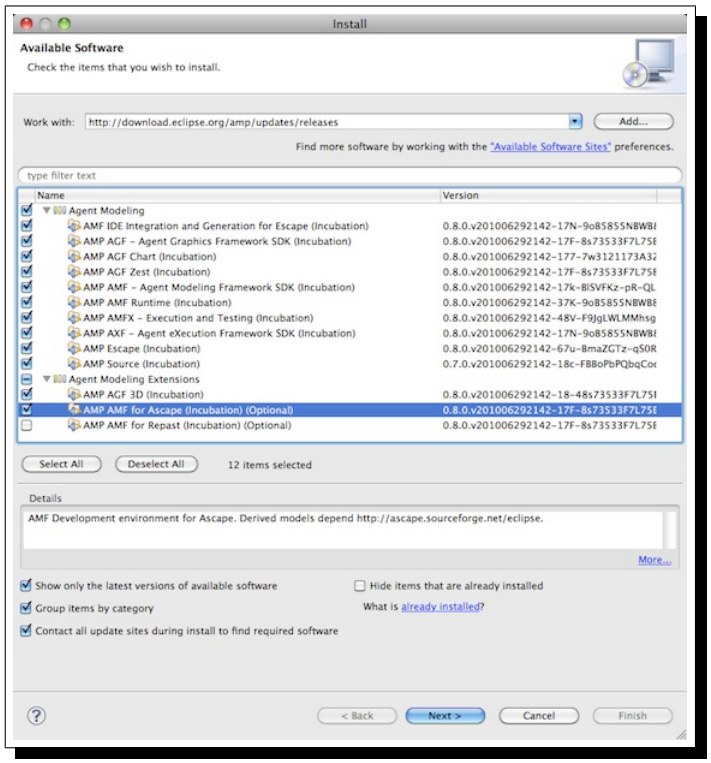


Figura 5.3: Finestra di installazione da update site

### 5.3.2 Compatibilità

La Agent Modeling Platform, come accennato in precedenza, prevede piena compatibilità con Eclipse Indigo 3.7, versione ormai superata da varie release successive: Eclipse Juno 4.2, Eclipse Kepler 4.3 e nel 2014 dalla nuova versione, attualmente in sviluppo, Eclipse Luna 4.4.
Quindi, per poter lavorare su AMP e non confinare gli utenti e gli sviluppatori alla versione 3.7, il lavoro di tesi ha necessitato che la piattaforma fosse compatibile con release recenti.
Questa parte preliminare del lavoro, ha previsto un'analisi approfondita della piattaforma per comprenderne le dipendenze da altri framework ed, in seguito, la risoluzione manuale di quelle che generavano conflitti.
Tale parte del lavoro, ha portato la piattaforma ad essere compatibile con tutte le versioni di Eclipse sopraccitate, ed è stata resa pubblica una guida all'installazione disponibile a questo indirizzo https://www.dropbox.com/s/u53jc15136szdxa/MASON4AMPGuide.pdf.

## 5.4 Esperienza Utente

La Agent Modeling Platform si integra perfettamente con l'interfaccia di Eclipse, grazie alla possibilità offerta dall'IDE di personalizzare le varie componenti (dette *views*) che vengono mostrate all'utente. Per realizzare quanto appena detto, Eclipse permette di definire delle *Perspective* in cui specificare ogni singola *view* da mostrare, così da comporre progressivamente tutta l'interfaccia grafica.

### 5.4.1 Interfaccia grafica

AMP offre due prospettive principali per lavorare con i modelli basati su agenti:

- **Agent Modeling**: supporta tutti gli aspetti legati allo sviluppo di modelli, incluso l'editing e la generazione automatica dell'intero codice e della documentazione.
- **Agent Execution**: supporta le visualizzazione, la gestione e l'esplorazione di una simulazione in esecuzione.

Caratteristica importante della piattaforma è la capacità di eseguire modelli direttamente nell'ambiente, in cui essi vengono sviluppati. Infatti, la prospettiva **Agent Execution** viene attivata dinamicamente al lancio di una simulazione.

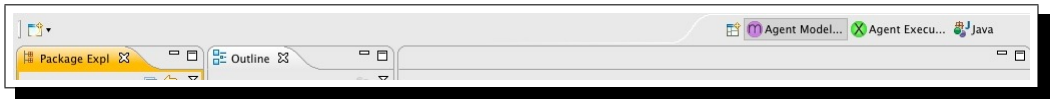


Figura 5.4: Eclipse permette di scegliere la prospettiva relativa alla modellazione fornita da AMP

### 5.4.2 Modellazione

La prospettiva di Agent Modeling, come accennato, si concentra sullo sviluppo di modelli, mettendo a dispozione vari strumenti, a partire dalla creazione di un nuovo progetto. Infatti, come mostrato in figura 5.5, alla creazione di un nuovo progetto vengono visualizzate tutte le piattaforme supportate e per cui è installato l'ambiente necessario in AMP.
In seguito alla creazione del progetto, è semplice aggiungere al suo interno un nuovo modello di simulazione selezionando un *Agent Model (MetaABM)*.

Tale modello è rappresentato da un unico file con estensione *.metaabm*, che contiene tutte le definizioni dello spazio e degli agenti, inclusi i loro comportamenti.

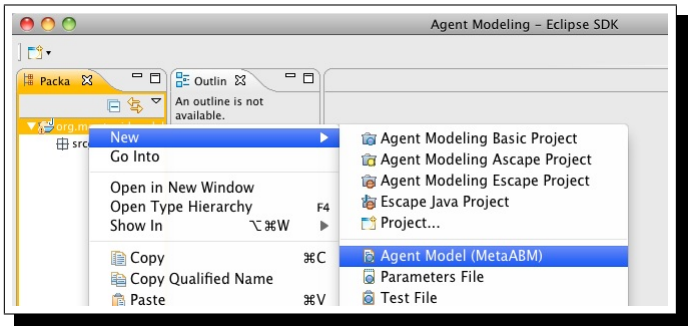


Figura 5.5: Creazione di progetti per ogni piattaforma supportata

### Viste

Il *workbench*, quindi, è suddiviso in viste in funzione della prospettiva scelta. Nella prospettiva di modellazione, il focus principale si concentra sulla view *"Editor"* (vedi Fig. 5.6), che offre la possibilità di apportare modifiche ad un qualsiasi modello o file sorgente.
La vista dedicata alle proprietà del modello, ha un comportamento contestuale, in quanto varia il suo contenuto in funzione di ciò che l'utente seleziona dall'editor. Permette di visualizzare e modificare dettagli specifici dell'oggetto correntemente selezionato, come ad esempio gli attributi per esso definiti.

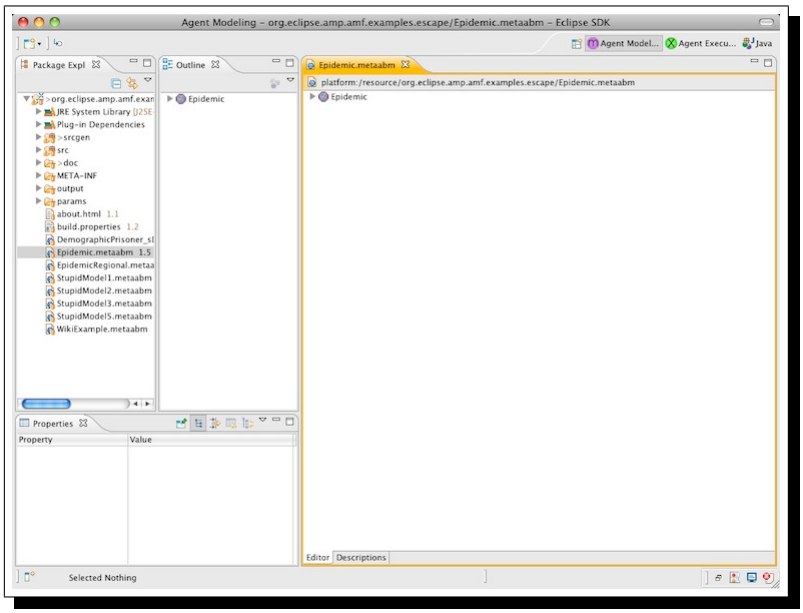


Figura 5.6: Come si presenta il workbench nella prospettiva Agent Modeling

### Editor Gerarchico

L'editor della prospettiva di modellazione è definito nell'*Agent Modeling Framework*, che viene analizzato nel capitolo dedicato all'architettura di AMP. Esso fornisce un vasto insieme di funzionalità utili alla creazione e gestione di modelli AMF e si basa sugli "Edit tools" dell'*Eclipse Modeling Framework* [10].
L'editor presenta una struttura gerarchica, più precisamente ad albero, in cui il modello si compone di nodi ognuno dei quali rappresenta un'entità del modello, come ad esempio agenti, azioni e spazi. Aggiungere un componente

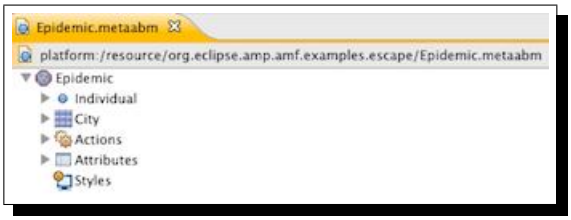


Figura 5.7: Il contesto è la radice dell'albero

al modello, considerando la sua struttura ad albero, significa aggiungere un

nodo figlio ad un nodo preesistente. Tale nodo può essere di vari tipi, come mostrato dalla figura 5.8, inoltre la scelta del tipo è contestuale in funzione del nodo padre.

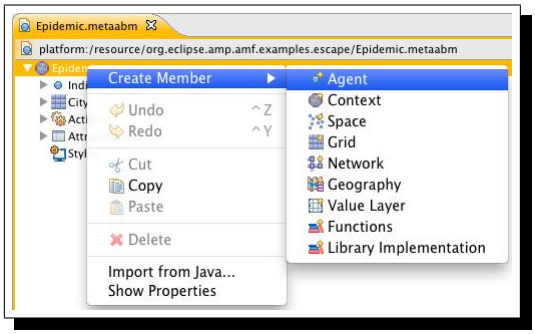


Figura 5.8: Aggiunta di un nuovo nodo

### 5.4.3 Azioni e Funzioni

Ogni componente può avere dei comportamenti che vengono specificati in un unico nodo di tipo *Action*. Le azioni sono un aspetto chiave e relativamente complesso dell'editor, anche dovuto al fatto che la relazione che intercorre tra ognuna di esse forma un grafo, che non è possibile rappresentare all'interno di un editor gerarchico. Un'azione viene aggiunta come ogni altra componente, aggiungendola come nodo figlio di chi la compie.

#### Tipi di azioni

Le azioni sono catalogate in vari ruoli, in modo da diversificare le caratteristiche di un agente, per rispecchiare quella che è la sua definizione nel modello teorico (vedi Fig. 5.9).

- **Selection**: una selezione definisce un particolare insieme di agenti con cui si vuole interagire. Le selezioni sono formate da azioni di tipo *Select*, *Query* e *Logic*. Ogni selezione fa riferimento al contesto in cui ne è stato definito il comportamento.

- **Command**: un comando può a sua volta essere di tre tipi. Evaluta-zione e settaggio di stato (*State*), spostamento nello spazio (*Space*) ed operazioni in reti di agenti (*Network*).

- **Builder**: speciale categoria di azioni che viene utilizzata per creare spazi. Questo tipo di comandi viene invocato dalle *Build Actions*, definite come nodi figli della radice del modello.

- **Other**: prevede un'unica azione di tipo *Method*, che consiste nell'includere del codice arbitrario all'interno di un metodo, che viene autogenerato. Tale codice può essere scritto in linguaggio Java, così come in qualsiasi altro linguaggio, in base a come esso viene generato per la determinata piattaforma scelta. Un'azione di questo tipo dovrebbe essere sempre evitata, ma può essere presa in considerazione quando sembra non esserci un altro modo per costruire una funzionalità equivalente, utilizzando le azioni native del framework, ad esempio utilizzare API di una libreria esterna.

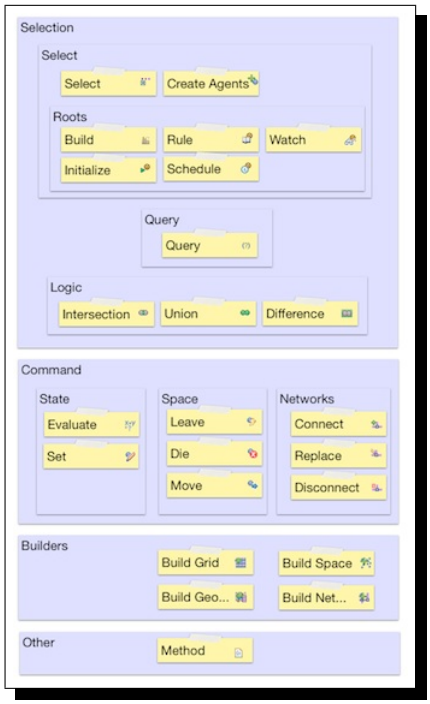


Figura 5.9: Possibili azioni a disposizione di un agente

### Gruppi di funzioni

Il framework di modellazione mette a disposizione funzioni generiche atte ad effettuare calcoli, confronti ed espressioni matematiche più complesse. La libreria delle funzioni può essere facilmente estesa, per aggiungerne di nuove, offrendo maggiori capacità alla piattaforma. I principali gruppi in cui è possibile catalogare le funzioni sono:

- **Logiche**: operazioni di confronto tra valori.
- **Numeriche**: supporto per vari operatori matematici.
- **Spaziali**: funzionalità fondamentali per i modelli ad agenti, per la scoperta dello spazio in cui ci si trova, come ad esempio l'interrogazione del vicinato.
- **Random**: generazione di valori pseudo-casuali, in particolare distribuzioni numeriche.
- **Grafiche**: supporto per la definizione di colore e forma di un agente. Queste funzioni sono combinate a definizioni di stile per la visualizzazione, consentendo di essere definite una sola volta e riutilizzate per più agenti, senza dover apportare alcuna modifica.
- **Temporali**: valori legati al tempo d'esecuzione della simulazione.
- **Matematiche**: supporto per un vasto numero di funzioni matematiche molto specifiche e ben testate, appartenenti alla libreria *Java Math*.
- **Liste**: trattamento di dati in forma vettoriale.
- **Distribuzione**: distribuzioni numeriche casuali uniformi, che facilitano operazioni come l'inizializzazione di parametri.

### 5.4.4 Building del Modello

Nei più comuni ambienti di sviluppo, si è abituati ad un passo di compilazione che è successivo a quello di implementazione. Eclipse e la Agent Modeling Platform offrono la possibilità di effettuare la compilazione automatica. Ciò significa che semplicemente salvando le modifiche apportate al modello, viene eseguita la compilazione, operazione a carico della piattaforma, di cui l'utente non deve avere assumersi nessuna responsabilità.
Quello che viene compilato dipende dal progetto su cui si lavora. Ad esempio, se si salva un modello creato in un *Agent Modeling Escape Project* avvengono in automatico i seguenti step:

1. Il *builder* specifico per Escape genera il codice Java corrispondente al modello, che utilizza la API Escape, aggiungendo inoltre il supporto alla grafica ed alla visualizzazione 3D.

2. Il builder della documentazione genera i documenti contenenti informazioni relative al modello, in formato HTML.

3. Il codice prodotto al passo 1 viene elaborato dal compilatore Java.

4. Il progetto viene "impacchettato" per poter essere utilizzato come un Eclipse plug-in.

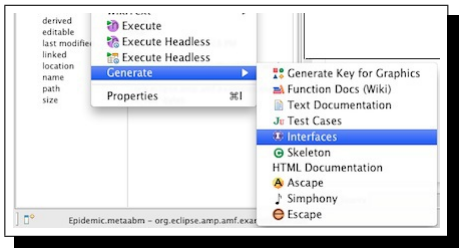


Figura 5.10: Generazione automatica di diverse feature

### Documentazione

La documentazione è un aspetto importante di qualsiasi progetto. Avere la possibilità di poterla generare automaticamente a partire da un modello, al semplice salvataggio di quest'ultimo è un vantaggio notevole.
Tali documenti vengono generati in formato HTML come pagine web statiche (vedi Fig.5.11). La piattaforma produce sia la versione completa che una semplificata. In generale, ogni componente aggiunta al modello, prevede una proprietà *Description*, che l'utente può utilizzare per far sì che la documentazione includa tale definizione. Inoltre, se un attributo possiede un valore predefinito, tale valore è indicato nella documentazione.

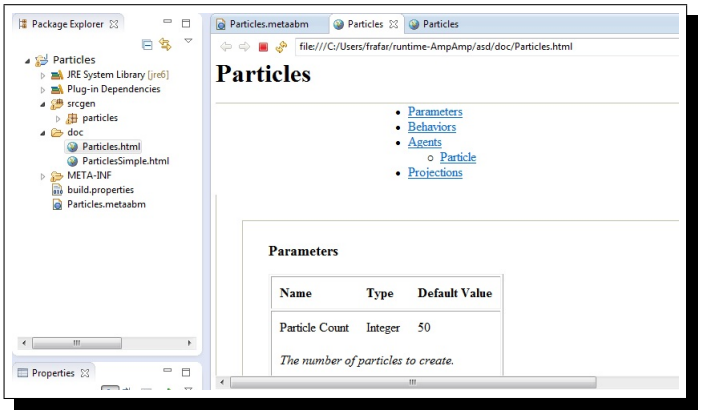


Figura 5.11: Breve scorcio della documentazione generata da AMP

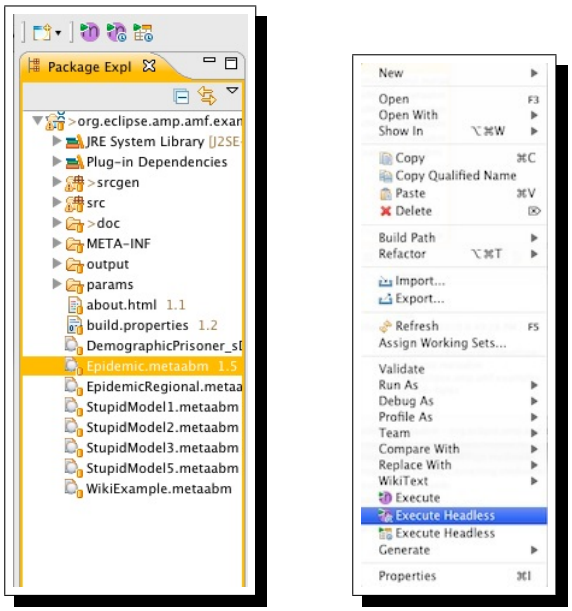


Figura 5.12: Strumenti di esecuzione: toolbar e menu contestuale

### 5.4.5 Esecuzione del Modello

Eseguire modelli compatibili con la piattaforma è molto semplice. Quando il file *.metaabm* relativo al modello viene selezionato, non è importante in quale prospettiva di Eclipse ci si trovi, la barra degli strumenti ed il menu contestuale sono aggiornati per rispecchiare il file selezionato ed offrire gli strumenti adeguati alla sua gestione. Così come mostrato nelle immagini sottostanti, i tasti d'esecuzione vengono mostrati all'utente. Sono presenti tre modalità d'esecuzione:

- **Execute**: manda in esecuzione il modello con la grafica predefinita (solitamente 2D), aprendo la prospettiva di *Agent Execution*.
- **Execute 3D**: manda in esecuzione il modello con visualizzazione 3D, aprendo la prospettiva di *Agent Execution*.
- **Execute Headless**: esegue il modello senza ausilio di grafica, aprendo la prospettiva di *Agent Execution*.
- **Execute Headless (Data)**: manda in esecuzione il modello collezionando i dati che produce, visualizzando il tutto.

Una volta che la simulazione è stata mandata in esecuzione, Eclipse cambia dinamicamente prospettiva, passando da quella di modellazione a quella di esecuzione: la *Agent Execution Perspective*. Quest'ultima dispone di una toolbar, che permette di controllare l'esecuzione del modello. Come mostra-

Figura 5.13: I controlli dell'esecuzione

to in Fig. 5.13, viene offerta all'utente la possibilità di eseguire, far ripartire, mettere in pausa, far eseguire un singolo passo, terminare e chiudere l'esecuzione di un modello. La piattaforma è anche capace di eseguire più modelli contemporaneamente, garantendo all'utente di poterli controllare singolarmente.
Inoltre, viene data la possibilità di poter rallentare o velocizzare l'esecuzione: lo *slider* in Fig. 5.13 permette di effettuare intuitivamente tale operazione. In realtà, esso non modifica realmente la velocità di esecuzione, ma il numero di volte che la visualizzazione viene resa a schermo, in quanto una simulazione potrebbe completare anche qualche migliaia di iterazioni in un secondo. Aggiornando la visualizzazione meno frequentemente, si può notare un effetto *speed-up* nell'esecuzione. Invece, rallentando la simulazione vengono effettuate delle pause tra le iterazioni della simulazione.

### Viste e Proprietà

Un concetto importante nell'ambiente del workbench è il *"modello attivo"*. Il modello attivo, è quel modello che attualmente viene gestito dai controlli della toolbar. Mentre possono esserci più modelli in esecuzione, solamente un modello può essere attivo.
L'attività di un modello è corredata anche dalle *views*, di cui si compone la prospettiva di esecuzione. Tali viste permettono la gestione e l'esplorazione del modello durante la sua esecuzione.
In aggiunta, viene offerta la possibilità di ispezionare gli agenti. In particolare, grazie alla vista *Properties*, se si vuole venire a conoscenza di maggiori informazioni su un agente basta selezionare un agente dalla visualizzazione grafica 2D o da qualsiasi altra visualizzazione che consenta la selezione degli agenti, per poi consultare la vista Properties (vedi Fig. 5.14).

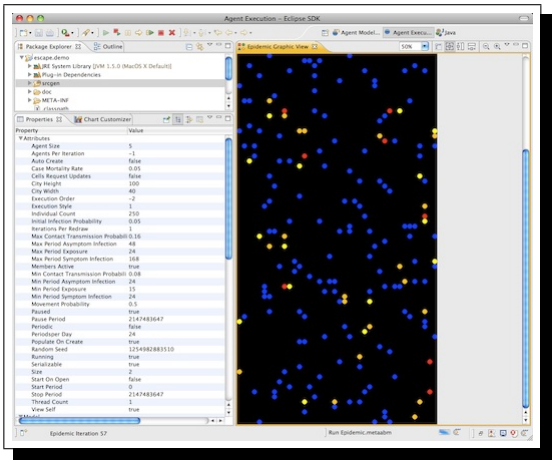

Figura 5.14: Vista delle proprietà di un agente

### Visualizzazione e Grafici

La Agent Modeling Platform è progettata per supportare diversi tipi di visualizzazioni.

La **Graphic 2D** view (vedi Fig. 5.15) è il modo più comune per lavorare con un modello basato su agenti, ed è facilmente generata e visualizzata a partire dal modello.

La **Graphic 3D** view (vedi Fig. 5.16) fornisce una rappresentazione tri-

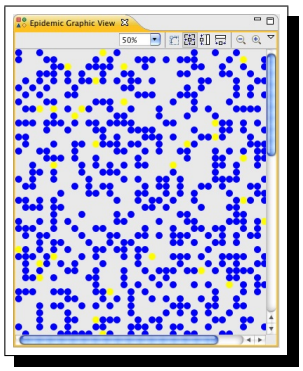


Figura 5.15: Visualizzazione in due dimensioni

dimensionale del modello in esecuzione. Tale modalità di visualizzazione mette a disposizione diversi controlli, grazie ai quali esplorare il modello in profondità, dall'alto e da vari punti di vista.

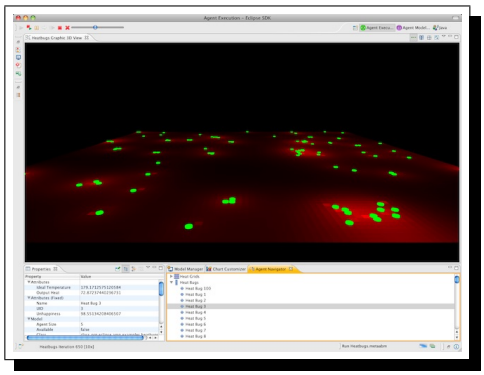

Figura 5.16: Visualizzazione in tre dimensioni

La **Graph View** permette di analizzare le relazioni che intercorrono tra gli agenti, evidenziando quelle che vengono chiamate *"reti sociali"*, che si vanno ad imporre tra un insieme di agenti. Tale vista (vedi Fig. 5.17) però, nel caso di modelli molto complessi, richiede del tempo per essere costruita a causa del gran numero di relazioni che costituisce il grafo.

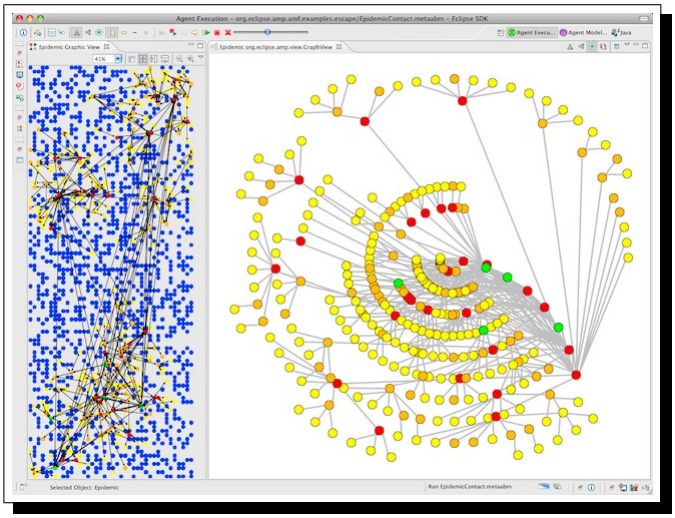

Figura 5.17: Graph view che mostra i contagi avvenuti tra gli agenti del modello Epidemic

Altra importante modalità di visualizzazione riguarda la **reportistica in tempo reale**. AMP offre la possibilità di creare grafici dinamicamente durante l'esecuzione del modello.
Può essere visualizzato qualsiasi valore per ogni attributo degli agenti, che abbiano impostato la raccolta dei dati (*gather data*) su quel determinato attributo. Tale capacità fornisce uno strumento potente e semplice da utilizzare, per esplorare modelli interattivamente.
I vari tipi di grafici supportati vengono mostrati nell'immagine seguente.

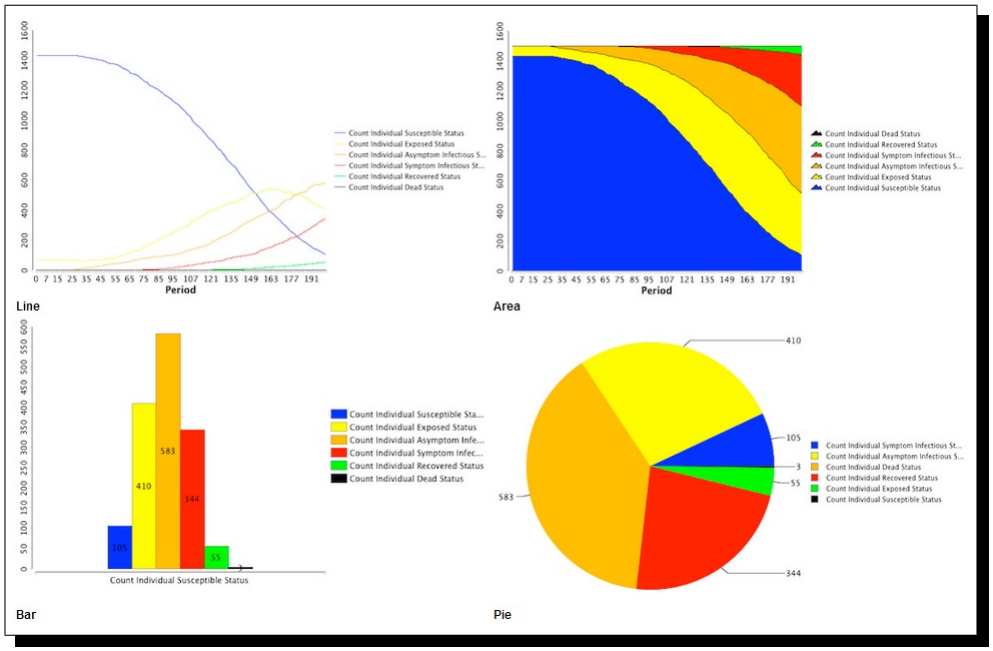


Figura 5.18: Grafici supportati: curve, aree, barre e torta

# Capitolo 6

# Architettura della piattaforma

La *Agent Modeling Platform* presenta un gran numero di caratteristiche, che per essere integrate e funzionanti, necessitano di un'architettura ben definita alla base.
Il lavoro di questa tesi si concentra sull'illustrazione di tale architettura, analizzando gli strati che la compongono ed evidenziando le funzionalità che ognuno di questi offre, in che modo riesce ad offrirle e di quali strumenti necessita per farlo.

## 6.1 Overview

La piattaforma *AMP* non è costruita come un blocco monolitico, anzi, essa è stata progettata in modo tale che sia costituita da una infrastruttura modulare, che possa offrire specifici strumenti in grado di supportare la modellazione basata su agenti ed altri tipi di tecnologie ad agenti.
Così come per il progetto Eclipse stesso, la visione globale di AMP è quella di un insieme di componenti debolmente accoppiati, ma allo stesso tempo in stretta collaborazione, che si integrano e permettono di costruire un grande e variegato ecosistema di strumenti ed approcci diversi.
Il diagramma architetturale in Fig. 6.1 mostra i *layers* e le relazioni che intercorrono tra AMP, Eclipse ed il mondo esterno.
Ogni layer viene analizzato in dettaglio in seguito.

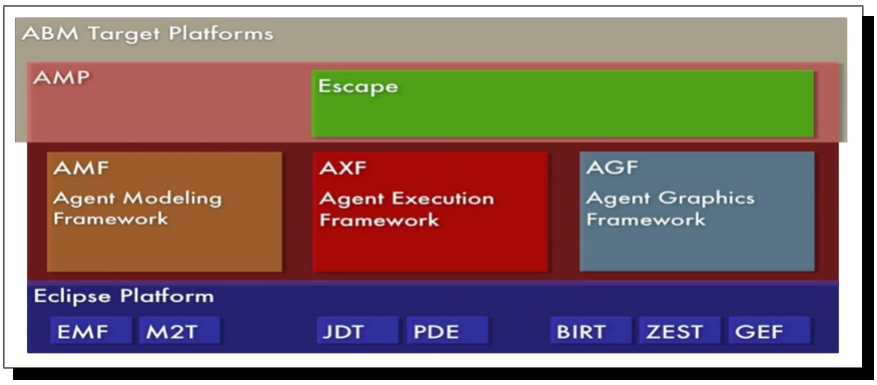


Figura 6.1: Diagramma architetturale

## 6.2 Eclipse Platform

La Agent Modeling Platform poggia su Eclipse ed, in particolare, su alcuni framework, considerati i "pilastri" che la sorreggono.

### 6.2.1 Eclipse Modeling Framework

L'*Eclipse Modeling Framework* è un progetto Eclipse che offre strumenti di modellazione e generazione del codice, per costruire tools e applicazioni basate su modelli di dati strutturati. A partire dalla definizione di un modello, specificata in **XMI** (XML Metadata Interchange), **EMF** mette a disposizione strumenti e supporto alla generazione di un insieme di classi Java relative a quel modello, oltre ad un insieme di classi *adapter*, che permettono la visualizzazione e la modifica del modello attraverso un editor.
I prodotti dell'Eclipse Modeling Framework includono la rispettiva definizione **XSD** (XML Schema Definition). Tale definizione fornisce un modello e le API per la manipolazione di componenti di uno schema XML, garantendo l'accesso alla rappresentazione DOM dello schema sottostante.

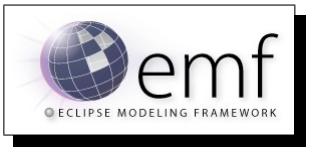


Figura 6.2: Logo dell'Eclipse Modeling Framework

### 6.2.2 Model To Text

Il progetto **Model To Text** (M2T) si concentra sulla generazione automatica di artefatti testuali a partire dai modelli. Presenta tre obiettivi principali che sono: fornire implementazioni di standard industriali ed Eclipse standard "defacto" ai motori che effettuano la traduzione di modelli in testo, offrire strumenti di sviluppo per i linguaggi di generazione del codice ed infine rappresentare un'infrastruttura comune per i linguaggi di questa categoria.
La piattaforma AMP, utilizza **Xpand**, linguaggio tipizzato staticamente atto alla definizione di *template*, che presenta caratteristiche di invocazione polimorfica di template, programmazione orientata agli aspetti, estensioni funzionali, un flessibile sistema di astrazione dei tipi, trasformazione di modelli, validazione di questi ultimi e molto altro.
Xpand include, inoltre, un editor che offre colorazione della sintassi, evidenziazione degli errori, navigazione nel codice, refactoring e completamento del codice.


Figura 6.3: Logo del linguaggio Xpand

### 6.2.3 Java Development Tools

Il progetto **JDT** fornisce un insieme di plug-in che aggiungono le capacità di avere un completo Java IDE alla piattaforma Eclipse. Ogni plug-in offerto dal JDT mette a disposizione degli sviluppatori API, che permettono di estenderli ulteriormente per personalizzarne le funzionalità.
In breve, i plugin del JDT vengono raggruppati in:

- *JDT APT*: aggiunge il supporto alle annotazioni di processazione ai progetti a partire da Java 5 in poi.
- *JDT Core*: definisce tutta l'infrastruttura che non riguarda l'interfaccia grafica, includendo ad esempio il building incrementale ed API per la navigazione dell'albero di un progetto Java.

- *JDT Debug*: aggiunge il supporto al processo di debug e funziona con ogni *Java Virtual Machine* che sia Java Platform Debugger Architecture-compatibile.
- *JDT Text*: fornisce un completo editor Java.
- *JDT UI*: aggiunge al workbench caratteristiche prettamente specifiche del linguaggio Java, come il *Package Explorer* e *wizard* per la crazione di elementi Java.

### 6.2.4 Plug-in Development Environment

Il progetto **PDE** fornisce strumenti per creare, sviluppare, testare e distribuire Eclipse plug-in e molti altri prodotti Eclipse. Inoltre è l'ambiente di sviluppo ideale per la programmazione di componenti OSGi indipendenti da Eclipse.
Il Plug-in Development Environment si caratterizza di tre componenti principali:

- *PDE Build*: strumenti basati su Apache Ant e script per l'automazione del processo di compilazione.
- *PDE UI*: editor, modelli ed altro necessario alla semplificazione dello sviluppo di plug-in Eclipse.
- *PDE API Tools*: strumenti offerti all'Eclipse IDE ed al processo di compilazione per manutenere le API.

### 6.2.5 Business Intelligence and Reporting Tools

**BIRT** è un progetto open source basato su Eclipse che offre un sistema di reportistica per applicazioni web, in particolare quelle basate su Java e Java Enterprise Edition.
BIRT è costituito da due componenti principali: un *report designer* integrato in Eclipse, che offre gli strumenti per costruire la rappresentazione desiderata del grafico; l'altra principale componente è il *runtime*, attiva durante l'esecuzione dell'applicazione. BIRT, inoltre, offre un motore di reportistica capace di integrare grafici ad una applicazione indipendente.

### 6.2.6 Graphical Editing Framework

Il progetto **Graphical Editing Framework** (GEF) fornisce tecnologie atte alla realizzazione di editor grafici e viste per il workbench di Eclipse. É composto nel complesso da tre componenti:

- *Draw2d*: toolkit di layout e rendering per mostrare componenti grafici su un Canvas SWT (Standard Widget Toolkit).
- *GEF (MVC)*: framework interattivo costruito sulla base di Draw2d, che favorisce l'implementazione di completi editor grafici per il workbench di Eclipse.
- *Zest*: toolkit di visualizzazione basato du Draw2d, che permette di implementare viste per il workbench di Eclipse.

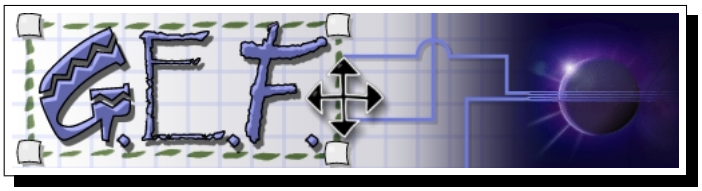


Figura 6.4: Logo del Graphical Editing Framework

### 6.2.7 Zest: The Eclipse Visualization Toolkit

**Zest**: *The Eclipse Visualization Toolkit*, è un insieme di componenti di visualizzazione realizzate per Eclipse. L'intera libreria Zest è stata sviluppata in SWT/Draw2d e si integra perfettamente in Eclipse, in quanto, è stato modellato dopo l'avvento di JFace, quindi tutte le sue view sono conformi agli stessi standard e convenzioni delle view esisitenti di Eclipse. Ciò significa che le applicazioni preesistenti possono utilizzare le componenti di Zest, senza dover apportare modifiche, oltre a nuove caratteristiche come il layout basato su grafo che consente di mettere in relazione le componenti.

## 6.3 Agent Modeling Framework

In questa sezione viene presentato il design dell'*Agent Modeling Framework* e come può essere utilizzato per create modelli che siano trasparenti, componibili ed adattabili. In linea generale, un modello basato su agenti (ABM), è formato da circa alcune componenti fondamentali: agenti, attributi, spazi ed azioni.
I primi tre si riferiscono a componenti strutturali, mentre le azioni definiscono comportamenti. I modelli ad agenti, inoltre, posseggono degli stili, che sono realizzati da specifici tipi di azioni, usate per determinare come rappresentare un agente in una visualizzazione. Infine, le azioni fanno uso di funzioni per adempiere il loro compito.

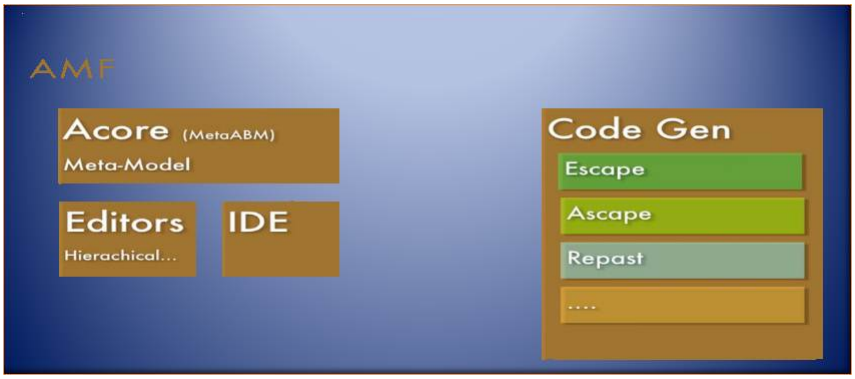


Figura 6.5: Le componenti dell'Agent Modeling Framework

L'ambiente Eclipse fornisce molte caratteristiche utili che la rendono ideale per una piattaforma fondata sull'Agent Based Modeling. L'*Agent Modeling Framework* offre strumenti potenti e di semplice utilizzo per la progettazione di modelli basati su agenti, compresa una rappresentazione comune, editor, generatori e l'ambiente di sviluppo.
**AMF** dispone di rappresentazioni ad alto livello per i comuni costrutti dell'ABM, introducendo vari modi di rappresentare agenti ed i loro comportamenti. Questo framework è stato progettato con lo scopo di permettere l'esplorazione di modelli complessi in maniera intuitiva.
Uno dei principali obiettivi di progettazione è quello di fornire strumenti che possano essere utilizzati anche da chi non è avvezzo alla programmazione, per creare modelli sofisticati. Questo tipo di sviluppo è detto *Model-Driven Software Development*, ed è una buona tecnica per aumentare la produttività degli sviluppatori indipendentemente dal livello di preparazione.

La componente su cui è sviluppato l'*Agent Modeling Framework* è **"Acore"** (vedi Fig. 6.5), ed in particolare la sua implementazione attualmente utilizzata, chiamata **"MetaABM"**. I modelli AMF sono detti *meta-modelli* in quanto sono utilizzati per definire come i modelli basati su agenti sono essi stessi modellati. AMF, quindi, è l'analogo dell'*Eclipse Modeling Framework*, ma rivolto alla progettazione ed all'esecuzione di modelli ad agenti.
Acore e MetaABM sono definiti in modelli di EMF *(Ecore)*, e forniscono una rappresentazione più diretta e ad alto livello di agenti, includendo caratteristiche spaziali, comportamentali e funzionali, sufficienti a generare modelli completi ed eseguibili per un ambiente di simulazione compatibile (chiamato *target platform*).
AMF è pienamente integrato con l'Eclipse IDE, ma i modelli Acore non ne necessitano di alcuna funzionalità, in quanto non hanno dipendenze da nessuna tecnologia particolare a parte XML/XSD.

### 6.3.1 Il metamodello Acore

**Acore** quindi è il modello che definisce i concetti fondamentali dell'ABM, che potranno essere utilizzati nella costruzione di simulazioni nella Agent Modeling Platform. Una breve panoramica mette in evidenza:

- **Named Entities**: è l'entità più generica. Ogni entità ha bisogno di essere referenziata. L'utente può utilizzare un'etichetta, mentre il software la identifica con un *ID*. Presenta anche una descrizione che viene inclusa nella documentazione.

- **Agenti**: astrazione di un agente che esiste in uno o più spazi. Esso esegue comportamenti che ne modificano lo stato. Le azioni sono rappresentate da metodi, mentre lo stato da attributi. Un agente è rappresentato graficamente da uno stile.

- **Attributi**: rappresentano lo stato interno di un agente. Possono avere tipo "primitivo", cioè *Boolean, Integer, Real, Symbol*

- **Spazi**: gli spazi forniscono un ambiente in cui gli agenti possono avere una locazione fisica o logica grazie alla quale possano interagire sia l'uno con l'altro che con l'ambiente. Un agente può esistere in più di uno spazio contemporaneamente, ed un dato agente non necessariamente deve esistere in ogni spazio del modello. Gli spazi supportati sono: *continui, griglie, network e geografici (GIS)*. I confini degli spazi possono essere toroidali o meno, si può inoltre scegliere se in una stessa

locazione possono essere presenti più agenti allo stesso istante. Infine, sono supportati tre tipi diversi di vicinato di un agente: *Euclidiano, Moore, Von-Neumann.*

Il diagramma in Fig. 6.6 mostra maggiori dettagli sul design strutturale.

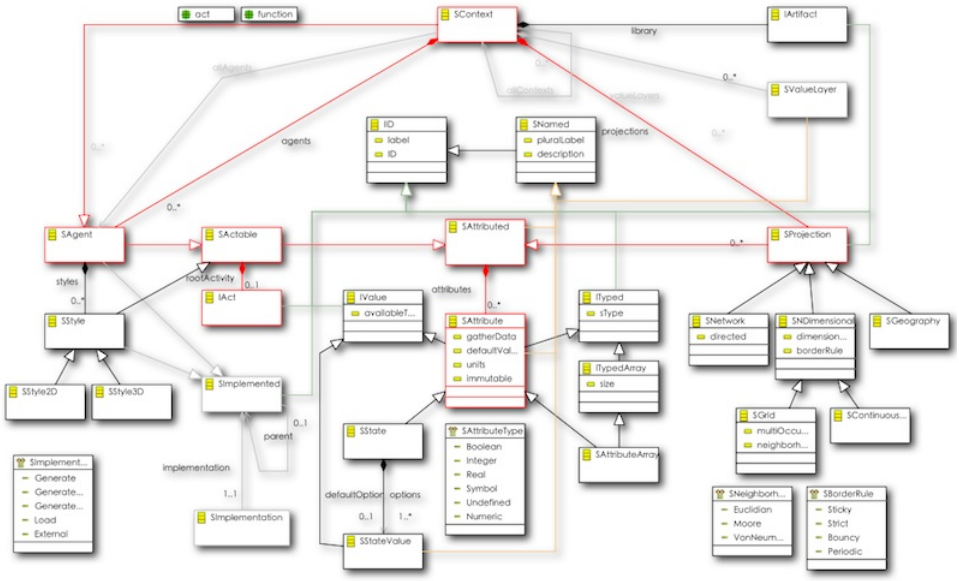


Figura 6.6: Diagramma UML delle classi di Acore

### 6.3.2 Acore in AMP: MetaABM

Il metamodello **MetaABM** implementa il modello *Acore*, le cui interazioni fondamentali sono evidenziate in rosso in Fig. 6.6 e la cui struttura segue un pattern di composizione. Nel dettaglio:

1. Ogni modello ha alla radice dell'albero che lo rappresenta un Context. I Context sono agenti capaci di contenere altri agenti.

2. Al tempo di design i contesti contengono definizioni di agenti. A runtime, il contesto conterrà istanze di agenti definite in *SContext* ed in *SAgent*.

3. Gli agenti hanno degli attributi. Gli attributi del contesto spesso rappresentano parametri di input per gli agenti contenuti.

4. I contesti possono contenere *Projection* (spazi), che rappresentano spazi di interazione per gli agenti, griglie, spazi continui o grafi.

5. Gli agenti supportano anche il tipo *Actable* e possono avere svariati comportamenti chiamati "Azioni".

6. Gli stili, chiamati *Styles*, forniscono un meccanismo per definire comportamenti di visualizzazione per gli agenti.

Le istanze di questo metamodello sono effettivamente i modelli che l'utente costruisce grazie all'editor gerarchico. In esso vengono specificate tutte le definizioni che sono contenute nel modello, dagli spazi utilizzati ai comportamenti degli agenti. Si tratta di un file con estensione *.metaabm* scritto in linguaggio XML-based, in cui è descritto l'intero modello di simulazione.
Ad ogni salvataggio all'interno dell'editor, il framework AMF provvede a tradurre ciò che l'utente ha costruito graficamente in codice XML, costruendo il modello in maniera incrementale. Il modello viene quindi validato secondo le regole definite nei file della componente Acore viste in precedenza, e se corretto si prosegue con la fase di generazione del codice.

### 6.3.3 Generazione del codice

La *metamodellazione* effettuata grazie ad EMF, derivata nella componente **Acore**, consente di costruire la componente di generazione automatica del codice chiamata **CodeGen** (vedi Fig.6.5).
Oltre alle definizioni astratte presenti in Acore (in particolare si fa riferimento all'implementazione MetaABM), CodeGen si basa sulle tecnologie offerte dal **Model To Text** Framework. Al salvataggio di un modello creato dall'utente grazie all'editor gerarchico, viene aggiornato e validato il file *.metaabm* che lo rappresenta, a questo punto entra in gioco la componente CodeGen: il modello metaabm viene "analizzato" ed in base alla *target platform* desiderata si vanno ad invocare gli specifici Eclipse plugin, addetti all'aggiornamento del progetto in cui si sta lavorando, invocando a loro volta i plugin di generazione del codice.

La generazione del codice si basa su file di template e di workflow. I file di template sono file **Xpand/Xtend**, che permettono di specificare come una classe *auto-generata*, debba essere strutturata e da cosa debba essere composta, tutto ciò operando sulle definizioni astratte delle entità definite da Acore.

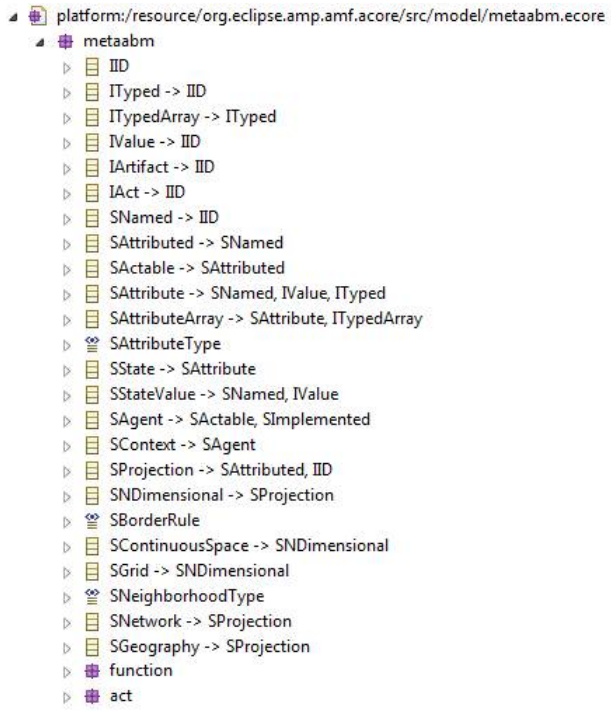


Figura 6.7: Modello Ecore del metamodello MetaABM

Un file **Xpand/Xtend**, per una classe auto-generata, è basato sul meccanismo di *DEFINE-EXPAND-AROUND*, attraverso il quale è possibile specificare delle definizioni di codice che possono essere espanse in punti precisi. Il file vero e proprio viene creato tramite la clausola *FILE*.

- **DEFINE**: definisce un blocco di template. É la più piccola unità identificabile in un file di template. Il tag consiste di un nome ed una lista di parametri separati da virgole, seguito dal nome della classe del metamodello per cui il template è definito.

```
1 «DEFINE templateName(formalParameterList) FOR MetaClass»
2         a sequence of statements
3 «ENDDEFINE»
```

  I template possono essere visti come speciali metodi della metaclasse di riferimento. Infatti, c'è sempre un parametro *this* implicito, che può essere utilizzato per riferirsi al modello sottostante. Il corpo di un template può contenere una sequenza di altre istruzioni, incluso qualsiasi tipo di testo.

Xpand supporta il polimorfismo, per cui se ci sono due templates (due blocchi DEFINE) con lo stesso nome, definiti per due metaclassi discendenti dalla stessa superclasse, Xpand utilizzerà il template corrispondente alla 'sottoclasse' che lo ha invocato.

- **EXPAND**: espande un blocco DEFINE, in un contesto differente, eseguendolo e riportandone l'output generato in corrispondenza della riga in cui viene invocata. É simile al concetto di subroutine.

```
1 «EXPAND definitionName [(parameterList)]
2  [FOR expression | FOREACH expression [SEPARATOR expression]][ONFILECLOSE]»
```

- **AROUND**: permette di aggiungere e modificare templates in maniera non invasiva, offrendo la possibilità di non dover apportare modifiche al template originario. Tale funzionalità di *Aspect Oriented Programming* è ottenuta grazie all'utilizzo del *workflow engine*.

```
1 «AROUND qualifiedDefinitionName(parameterList)? FOR type»
2  a sequence of statements
3 «ENDAROUND»
```

- **FILE**: istruzione fondamentale, che permette di ridirezionare in un file l'output prodotto dal blocco di istruzioni contenute nel suo corpo.

```
1 «FILE expression [outletName]»
2    a sequence of statements
3 «ENDFILE»
```

Il suo target è un file nel file system, il cui nome è specificato dall'espressione contenuta nel tag *[outletName]*. Tale espressione può essere una qualsiasi stringa, anche costruita grazie all'operatore di concatenazione +. Il file prodotto, viene creato nella directory di lavoro del generatore Xpand; è possibile specificare un percorso a partire da tale directory utilizzando il simbolo "/".

### Il plugin di generazione del codice

Il principale plugin del framework AMF, che si occupa della generazione di codice e documentazione, è identificato nella piattaforma AMP dall'identificativo ***org.eclipse.amp.amf.gen***.

I file di template Xpand/Xtend si basano sull'implementazione *MetaABM* del modello Acore, e sono concentrati nel package *metaabm.tmpl*: il più importante è **Java.xpt** (vedi Fig. 6.9). Esso rappresenta il file contenente la definizione dello scheletro che una classe Java debba avere, per costituire la base di partenza a cui aggiungere ciò che è prodotto dall'espansione delle

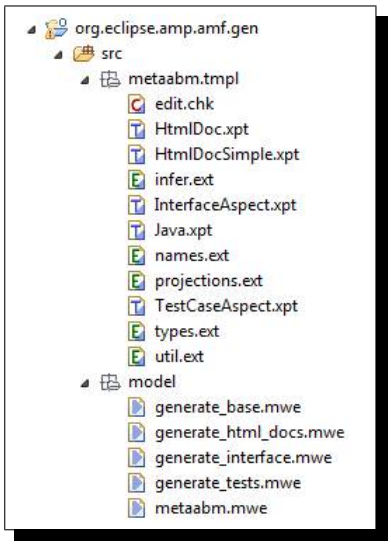


Figura 6.8: Il plugin principale della componente di generazione del codice in AMF

definizioni contenute nei file di template specifici per la piattaforma.

***HtmlDoc.xpt*** ed ***HtmlDocSimple.xpt*** forniscono i template di generazione delle pagine HTML contenenti la documentazione sia in formato completo che semplificato. Infine, ***names.ext***, ***types.ext*** ed ***infer.ext*** sono dei file Xtend di estensione che offrono funzioni ausiliarie al riguardo di, rispettivamente, manipolazione di ID ed etichette delle entità del modello, recupero di informazioni sulla gerarchia del modello e gestione del comportamento del modello/agenti.

Avendo analizzato i file di template, si procede con il considerare i **file di workflow**, contenuti nel package *model*. Quest'ultimi, consentono l'applicazione delle regole specificate nei file di template, che formeranno le componenti di una simulazione. In generale viene eseguito un unico file di workflow, rappresentativo per ogni piattaforma, che porta con sè, in senso figurato, tutte le definizioni presenti nei template, ad una componente, *Xpand Code Generator*, adibita all'elaborazione del prodotto finale.
Un file di workflow è un documento scritto in linguaggio *XML-based*, processato dal *Modeling Workflow Engine (MWE)* per configurare i generatori di codice *Xtext/Xpand* (vedi Fig. 6.10). Ai file *mwe* è associata una classe,

```
«IMPORT metaabm»
«IMPORT metaabm::act»
«IMPORT metaabm::function»
«IMPORT emf»

«EXTENSION metaabm::tmpl::names»
«EXTENSION metaabm::tmpl::types»
«EXTENSION metaabm::tmpl::infer»
«EXTENSION metaabm::tmpl::util»

«REM»Author: Miles Parker «ENDREM»

«DEFINE Model FOR SContext-»
    «EXPAND Files-»
    «EXPAND Model FOREACH agents-»
«ENDDEFINE»

«DEFINE Model FOR SAgent-»
    «EXPAND Files-»
«ENDDEFINE»

«DEFINE Files FOR SAgent-»
    «EXPAND ClassFile-»
    «EXPAND ClassFile FOREACH styles-»
    «EXPAND ClassFile FOREACH attributes.typeSelect(SState)-»
«ENDDEFINE»

«DEFINE ClassFile FOR IID-»
    «IF generate()-»
        «FILE javaFileLoc()-»
            «EXPAND ClassText»
        «ENDFILE-»
    «ENDIF-»
«ENDDEFINE»
```

Figura 6.9: Breve scorcio del file Java.xpt

chiamata *builder*, che si occupa di specificare i documenti da far eseguire e predisporre l'ambiente alla generazione di codice.
In particolare:

- ***generate_base.mwe***: indica al generatore Xpand il metamodello di riferimento ed il punto di partenza della generazione delle classi Java (Java.xpt).
- ***generate_html_docs.mwe***: generazione di documentazione HTML (HtmlDocs.xpt, HtmlDocsSimple.xpt).
- ***metaabm.mwe***: parsing e checking del modello metaabm.

```
<component id="javaGenerator" class="org.eclipse.xpand2.Generator" skipOnErrors="true" fileEncoding="iso-8859-1">
    <metaModel idRef="metaabm"/>
    <expand value="metaabm::tmpl::Java::Model FOR model"/>
```

Figura 6.10: Breve scorcio del file generate_base.mwe

Il plugin ***org.eclipse.amp.amf.gen*** è rappresentato dalla classe Java **AMFGenPlugin.java** (vedi Fig. 6.11). Si tratta di un plug-in *activator* che controlla il ciclo di vita del plug-in. Possiede un *PLUGIN_ID* che può essere acceduto dall'esterno, in particolare per ottenere accesso ai file di

```
public class AMFGenPlugin extends Plugin {

    // The plug-in ID
    public static final String PLUGIN_ID = "org.eclipse.amp.amf.gen";
```

Figura 6.11: Dichiarazione della Classe AMFGenPlugin

workflow ed invocarne l'esecuzione. Come accennato in precedenza, i file di workflow necessitano di classi *builder* che ne invochino l'esecuzione. In particolare, è necessario mettere in relazione il plug-in di generazione del codice con l'IDE. Per far ciò, la *Agent Modeling Platform* fornisce il plug-in ***org.eclipse.amp.amf.gen.ide*** (vedi Fig.6.12). Tutte le *target*

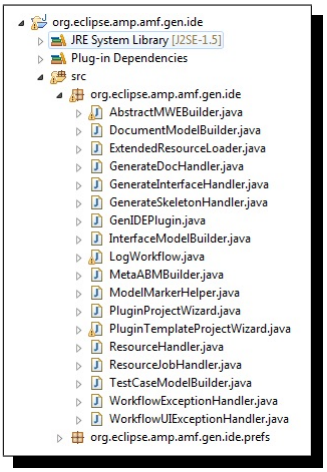


Figura 6.12: Struttura del plug-in org.eclipse.amp.amf.gen.ide

*platform* integrate in AMP, necessitano di plug-in che interagiscono con *org.eclipse.amp.amf.gen* ed *org.eclipse.amp.amf.gen.ide*. Tale interazione avviene nella classe Java che permette la creazione del progetto specifico per la target platform: si tratta di una classe che estende *PluginTemplateProjectWizard*, in cui vengono implementati dei metodi che garantiscono l'integrazione delle varie componenti. I metodi fondamentali sono *getDependenciesForManifest* che consente di specificare tutte le dipendenze necessarie alla target platform per poter eseguire il codice generato, e *getBuilders* che permette di aggiungere al progetto, builder adatti alla target platform andando a indicare gli ID dei plug-in da eseguire.
Ogni target platform include un builder personalizzato, in cui il metodo *loadPlugins* permette di far uso del plug-in di generazione del codice appropriato. Il builder è utilizzato in maniera intelligente da un *handler* che

monitora i cambiamenti di stato del modello metaabm su cui l'utente sta lavorando. Ogniqualvolta il modello subisce una modifica, l'handler invoca il metodo *handleModifiedResource* del builder di default, cioè ***AbstractMWEBuilder*** parte del plug-in *org.eclipse.amp.amf.gen.ide* (vedi Fig. 6.12), il quale carica le proprietà del modello che sono state recuperate dai file di workflow, invoca il metodo *generateModel(resource)*, facendo eseguire al Modeling Workflow Engine il workflow adatto alla target platform sul modello (la risorsa in questione), utilizzando le proprietà appena caricate, per poi aggiornare il progetto dell'utente (vedi Fig. 6.13).

```
public void handleModifiedResource(IResource resource) {
    if (resource instanceof IFile && resource.getName().endsWith(sourceExtension)) {
        loadPropertiesFromResource(resource);
        generateModel(resource);
        try {
            resource.getProject().refreshLocal(IResource.DEPTH_INFINITE, null);
```

Figura 6.13: Istruzioni fondamentali del metodo *handleModifiedResource*

## 6.4 Agent Execution Framework

L'*Agent Execution Framework* fornisce interfacce "core", servizi ed interfaccia grafica per l'esecuzione del ciclo di vita del modello, l'aggregazione degli agenti, il controllo della visualizzazione e della gestione dell'esecuzione, e l'interazione con il workbench.

Per rispettare i canoni imposti dall'implementazione globale che utilizza un approccio neutrale, il framework **AXF** non è concepito per fornire supporto all'esecuzione, ma invece agisce come un layer di astrazione tra l'ambiente Eclipse ed i vari toolkit di esecuzione dei modelli.
Le diverse implementazioni forniscono già motori di scheduling interni al modello e funzionalità di raccolta dati, che vengono collegate insieme ed utilizzate a runtime, attraverso i cosiddetti *adapters* (vedi Fig. 6.14).

Grazie al fatto che i framework di modellazione (AMF) e di esecuzione

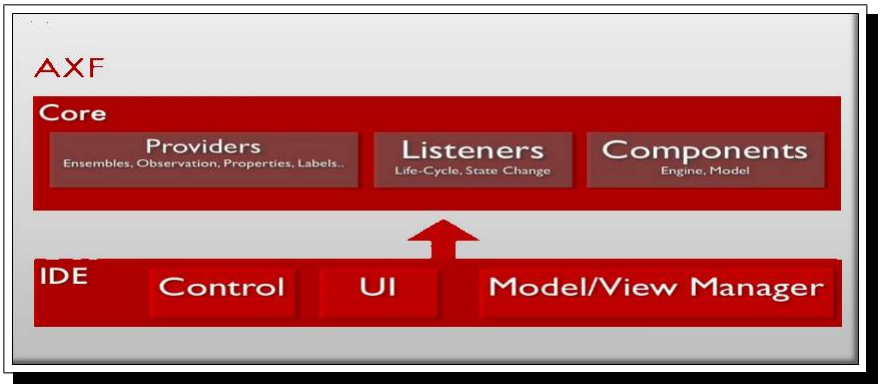


Figura 6.14: Le componenti dell'Agent Execution Framework

(AXF) non presentano dipendenze tra di essi, l'*Agent Execution Framework* fornisce tutte le *feature* descritte in maniera indipendente e modulare. Così come AMF, il framework AXF non è limitato ai modellatori di simulazioni basate su agenti.

Chiunque necessiti di esecuzione, gestione e visualizzazione, di collezioni di oggetti interagenti, il tutto all'interno del workbench di Eclipse può affidarsi ai servizi offerti dall'*Agent Execution Framework*.

## 6.5 Agent Graphic Framework

Il framework che si occupa della parte grafica è l'***Agent Graphic Framework***, il quale si basa ed estende strumenti della piattaforma Eclipse come **GEF**, **Draw2D**, **Zest** ed altre tecnologie per supportare la visualizzazione in *real-time* e l'interazione con i modelli ad agenti.

**AGF** fornisce il supporto alla visualizzazione di spazi in due e tre dimensioni, oltre alle strutture a grafo per la visualizzazioni di *network* tra gli agenti. Come le altre componenti di AMP, l'obiettivo di progettazione del framework grafico è di fornire una infrastruttura estensibile (vedi Fig. 6.15), garantendo a chi ne fa uso di poter creare in maniera semplice le proprie viste e parti dell'editor.
**AGF** include, inoltre, due componenti:

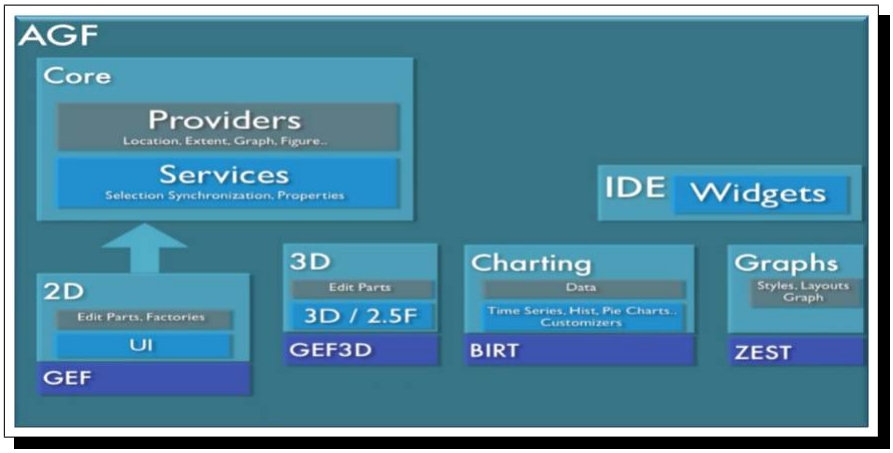


Figura 6.15: Le componenti dell'Agent Graphic Framework

- **AGF-Chart**: funzionalità che permette di costruire grafici in maniera dinamica, a partire dai dati prodotti dalla simulazione in esecuzione, fornendo strumenti interattivi per l'utente che ne garantiscano una semplice lettura ed analisi.
- **AGF-Viz**: si occupa della visualizzazione dinamica di strutture che aggregano informazioni "dense", come grafi e strutture n-dimensionali.

## 6.6 Target Platform Layer

Il *Target Platform Layer* poggia sulla Agent Modeling Platform, e rappresenta il livello in cui vanno ad aggiungersi le nuove target platform. Una target platform consiste di un toolkit di simulazione basato su agenti, a cui sono state aggiunte alcune caratteristiche proprie dello sviluppo di plug-in Eclipse, per far sì che possa integrare le sue capacità nei vari moduli messi a disposizione da AMP.

Un toolkit di simulazione è composto, in linea generale, da layer ognuno dei quali viene implementato da classi raggruppate in package. Per essere integrato in AMP, è necessario rendere il toolkit (quindi tutte le relative classi) un Eclipse plug-in, in modo da essere visto come una sola entità dall'ambiente: nella *Agent Modeling Platform* tali plug-in rispettano la forma ***org."toolkitname".core***.
L'integrazione vera e propria all'interno della piattaforma, invece, avviene fornendo plug-in che poggiano sulle componenti fornite da AMP (plug-in a loro volta) *"org.eclipse.amp.amf.gen"* per la generazione automatica del codice, *"org.eclipse.amp.amf.ide"* per l'integrazione nell'ide, *"org.eclipse.amp.axf"* per quanto riguarda gli adapters atti all'esecuzione dei nuovi modelli ed infine *"org.eclipse.amp.amf.agf"* per l'integrazione grafica nel workbench di Eclipse.
In particolare, per quanto riguarda il framework AMF, il nuovo toolkit che sta per essere integrato deve fornire i plug-in *org.eclipse.amp.amf.gen.toolkitname*, in cui definire i workflow necessari all'invocazione dei template di generazione specifici ed *org.eclipse.amp.amf.ide.toolkitname* che fornisce gli strumenti di integrazione nell'IDE, dalla creazione di un progetto che sia predisposto al supporto della nuova target platform, alla possibilità di generare il codice dal modello ad ogni salvataggio dell'utente.

L'integrazione di una nuova target platform, in particolare *MASON*, viene trattata nella tesi dal titolo "*Progettazione assistita di simulazioni agent-based: l'integrazione di una nuova target platform in Agent Modeling Platform*" scritta da Marco Amoruso [12].

# Capitolo 7

# Conclusioni e sviluppi futuri

La *Agent Modeling Platform* offre un ambiente completamente compatibile con Eclipse, l'ambiente di sviluppo più conosciuto e supportato, introducendo l'idea di avere una rappresentazione univoca del modello di simulazione. Inoltre grazie alla sua modularità consente l'integrazione di nuove tecnologie come nuovi toolkit di simulazione, ma anche tecnologie che non siano strettamente correlate alla simulazione di modelli basati su agenti.

In generale, AMP ha ottime potenzialità di poter diventare la piattaforma di riferimento per i modellatori, che non possiedono grosse esperienze di programmazione in linguaggi *Object Oriented*. Tale capacità è ottenuta anche grazie alla sua architettura fondata sul concetto di modularità, il quale è reso possibile da Eclipse, dai plug-in e dal livello d'astrazione offerto dai layer che la compongono.

Sicuramente coloro i quali hanno le capacità di poter scrivere il modello desiderato, scrivendone il codice ed utilizzando le librerie messe a disposizione da un toolkit, vedono questa piattaforma come uno strumento alquanto macchinoso, ma essa non è intesa per offrire supporto ai programmatori, anzi, essa è frutto di uno sforzo di progettazione che ha unificato molte tecnologie diverse, che potesse andare incontro alla comunità dei modellatori di simulazioni basate su agenti.

La piattaforma, però, non presenta solo vantaggi. Il principale è rappresentato dall'editor gerarchico che non sempre è d'aiuto al modellatore, a causa della rigida struttura ad albero che a volte pone un limite alla modellazione, precludendo possibilità che sarebbero state possibili ed implementate in ma-

niera molto più semplice andando a scrivere poche righe di codice, ma ciò è dovuto all'alto livello d'astrazione necessario per poter poi derivare da quel modello una simulazione scritta per un qualsiasi framework di simulazione. É bene sottolineare che l'editor presenta ancora alcuni difetti: elementi annidati non vengono mostrati sempre nell'ordine corretto, così come il *drag 'n' drop* degli stessi, spesso non viene effettuato.
Inoltre, la generazione di codice a partire dal modello, non è rispecchiata nel senso opposto: ciò significa che qualora l'utente modifichi le classi contenenti il codice, il modello non sarà aggiornato, anzi le classi verranno sovrascritte perdendo così le modifiche apportate.

Per quanto riguarda il futuro della piattaforma, sarebbe una caratteristica apprezzata, se l'editor gerarchico venisse abbandonato in favore di uno più flessibile, ed inoltre avere a disposizione molte più *target platform*, come ad esempio *MASON* e *NetLogo*, per far sì che AMP sia sempre più nota ed utilizzata da tutta la comunità dell'*agent-based modeling*.

# Bibliografia


[1] W.Shen, D.H. Norrie, Agent-Based Systems for Intelligent Manufacturing: A State-of-the-Art Survey, Knowledge and Information Systems, an Int. Jour. 1, 2, (1999) 129-156. 2.1

[2] Boids http://en.wikipedia.org/wiki/Boids 2.2.1

[3] Swarm www.santafe.edu/media/workingpapers/96-06-042.pdf 3.1.2

[4] MASON http://cs.gmu.edu/~eclab/projects/mason/

[5] NetLogo Models Library http://ccl.northwestern.edu/netlogo/models/ 3.2

[6] Axtell, R. L. & Epstein, J. M. (1996) Growing Artificial Societies: Social Science from the Bottom Up (Brookings Institution Press/MIT Press, Cambridge, MA) 3.4

[7] M.E. Inchiosa & M.T. Parker, (2002) Overcoming design and development challenges in agent-based modeling using ASCAPE http://www.pnas.org/content/99/suppl_3/7304.full.pdf+html

[8] M.J. North, N.T. Collier, J. Ozik, E.R. Tatara, C.M. Macal, M. Bragen and P. Sydelko, (2013) Complex adaptive systems modeling with Repast Simphony http://www.casmodeling.com/content/1/1/3

[9] Eclipse Development Process https://www.eclipse.org/projects/dev_process/ 5.2

[10] Eclipse Modeling Framework http://www.eclipse.org/modeling/emf/ 5.4.2

[11] D. Steinberg, F. Budinsky, M. Paternostro, E. Merks, EMF: Eclipse Modeling Framework 2nd Edition, Addison-Wesley Professional

[12] M.Amoruso, (2013) Progettazione assistita di simulazioni agent-based: l'integrazione di una nuova target platform in Agent Modeling Platform 6.6