

EvoMAS - Sistema Gerarchico Adattativo ad Agenti

Maria Letizia Moretti marialetizia.moretti@studio.unibo.it
Davide Vichi davide.vichi@studio.unibo.it

27 marzo 2011

Indice

1	Modello di Riferimento	1
2	Analisi dei Requisiti	3
3	Analisi del Problema	3
3.1	Modello dei ruoli	3
3.2	Modello delle risorse	6
3.3	Modello delle interazioni	9
3.4	Tool di simulazione	10
4	Design	12
4.1	Tipologia di Agenti	13
4.2	Analisi delle Tecnologie	14
4.2.1	TuCSon come alternativa dominante	15
4.3	Workspace	17
4.4	Sviluppo degli Artefatti	17
5	Scenario Applicativo: Gestione files di rete	21
6	Conclusioni	24
6.1	Possibili estensioni	24

Elenco delle figure

1	Upload post-morte	2
2	Stati di un AI	4
3	Stati di un AS	5
4	Esempio di ciclo di vita di una specie	7
5	Migrazione	8
6	Esempio Interazione AS	12
7	Rete a stella dei TC	18
8	Finestra di inizializzazione	24
9	Finestra principale	25
10	Finestra di focus sul singolo habitat	26

Elenco delle tabelle

1	Classificazione Tasks	6
2	Raggruppamento Servizi	9
3	Tasks e Informazioni	11
4	Artefatti e Metodi	20
5	Simulazioni comparative	22

Sommario

L'utilizzo di algoritmi e procedimenti nature-inspired è sempre stato d'aiuto nel determinare il set di valori ottimi dei parametri di un sistema ma anche per fornire strategie d'azione efficienti per il compimento di un dato task. Nell'ambito dei sistemi multi-agente, ovvero in sistemi complessi comprendenti fra le parti entità autonome chiamate *agenti* con un proprio flusso di controllo dove è caratteristica rilevante la proattività, può essere sensato quindi *agire* nel miglior modo in accordo ai criteri dettati dal sistema stesso. Nella relazione seguente verrà proposto un modello di sistema che pur astraendo dalle molteplici complessità presenti in natura, conterrà i concetti cardine (come riproduzione, selezione, migrazione, estinzione) utilizzati normalmente nella descrizione di azioni delle entità naturali. Tali termini costituiranno l'essenza del comportamento dei processi autonomi che implementeremo all'interno del giusto ambiente simulato, al fine di poter tracciare un'andamento complessivo di sistema applicabile ove possa essere migliorativo. Verrà quindi descritto il modello di riferimento, analizzato in dettaglio nella sua struttura, i vari comportamenti e le interazioni delle entità presenti, mostrata una possibile via implementativa comprendente aspetti di interazione utente e infine, grazie all'utilizzo del simulatore, ne verrà testata l'efficienza all'interno di uno scenario concreto.

1 Modello di Riferimento

Abbiamo un sistema complesso composto da due tipi di agenti:

- Agenti-Individuo
- Agenti-Specie

Agente-Individuo L'AI¹ è la più piccola entità computazionale del sistema. Il suo comportamento è lontanamente attribuibile al ciclo di vita di una cellula somatica. Possiede una propria struttura dati contenente: ID, tipo di specie, indicatore di fame... L' *indicatore di fame* è un valore intero indicante la quantità di risorsa necessaria per la sua riproduzione.

Ciclo di vita

- Per ogni specie esiste un *individuo primordiale* da cui hanno origine le discendenze.
- Il processo mediante il quale una cellula somatica si riproduce è detto *mitosi*. Esso avviene in varie fasi ed ha lo scopo di trasmettere l'informazione genetica alle cellule figlie in modo che abbiano le stesse caratteristiche e le stesse funzioni della cellula genitrice. In realtà, nella nostra simulazione avremo una *mitosi semplice* dove in base alle risorse messe a disposizione e all'indicatore di fame, l'individuo potrà scindersi in più nuovi individui della stessa specie con valori di fame determinati stocasticamente.
- Ogni nuovo individuo nato possiede un *contributo per la specie* unitario. L'AS (cioè quello di livello gerarchico superiore) ha il compito di ottimizzare il rapporto complessivo *contributi/fame* della propria specie selezionando così, fra i *neonati*, quello con indicatore di fame inferiore.

Agente-Specie Questi particolari agenti rappresentano una specie all'interno del sistema. Possiedono un *contributo* determinato dalla somma dei contributi forniti dai singoli agenti-individuo che gli appartengono e sono associati ad una particolare tipologia di risorsa. Ogni AS nasce in un pre-determinato habitat dove è presente una certa quantità di risorsa che sarà utilizzata per il mantenimento dei propri individui. Il *contributo* equivale alla quantità di individui posseduti nel tempo. Per massimizzare il proprio

¹D'ora in poi per semplicità di linguaggio si identificheranno gli agenti individuo col nome AI e gli agenti specie col nome AS.

contributo, un AS dovrà collezionare il maggior numero di individui possibili nel rispetto della quantità di risorsa disponibile per il loro mantenimento. Quindi, risulta chiara una politica di selezione basata sul miglior rapporto *contributo/fame* per ogni nuova coppia di individui generati dal processo di mitosi. Se la quantità di risorsa a disposizione nell'habitat dell'AS è insufficiente per permettere la riproduzione all'individuo selezionato, abbiamo due diversi comportamenti:

- **Se il pool ha cardinalità maggiore di 1:** L'AI viene terminato e opzionalmente trasformato in risorsa utilizzabile dentro l'habitat dagli individui della stessa specie, o che comunque necessitano di quel tipo di risorsa (fig. 1).
- **Se il pool ha cardinalità uguale a 1:** per la sopravvivenza della specie, l'AS migrerà con l'ultimo AI rimasto in un nuovo habitat dove è possibile trovare maggiori risorse (fig. 5).
- L'AS muore se migrando in tutti gli habitat non riesce a trovare risorse a sufficienza.

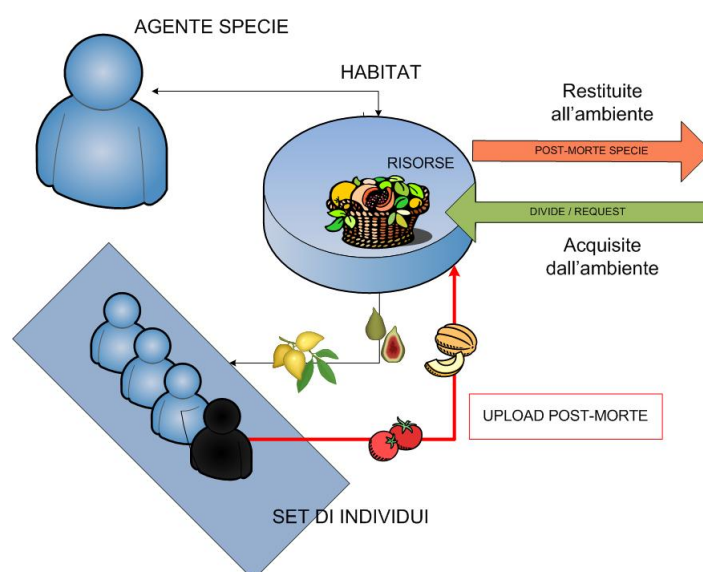


Figura 1: Upload post-morte. Se l'AI non possiede sufficienti risorse terminerà convertendosi in risorsa utilizzabile

In riferimento alla metodologia SODA, seguono le varie fasi per lo sviluppo del sistema.

2 Analisi dei Requisiti

L'obiettivo del progetto consiste nel realizzare un simulatore di ambiente EvoMAS. Questa applicazione deve essere in grado di osservare il comportamento del modello simulato nel tempo, al variare di alcuni parametri, quali:

- Numerosità delle specie presenti nell'ambiente;
- Numerosità degli habitat di cui è composto l'ambiente;
- Tipi di risorse e numerosità iniziale delle stesse;
- Numero di figli generati ad ogni passo dagli individui grazie alla riproduzione: fisso, casuale;
- Esistenza o meno di parassiti, ovvero di individui che non si riproducono e si limitano a consumare risorse;
- Riconversione di un individuo morto in risorsa;
- Indice di fame di ogni individuo generato: fisso, casuale;
- Scelta dell'habitat in cui effettuare la migrazione: fisso, casuale;
- Scelta dell'habitat di partenza, ovvero stabilire se ogni specie nasce nello stesso habitat o in habitat diversi.

I requisiti appena enunciati implicano lo sviluppo di un ambiente di simulazione fornito di interfaccia grafica, che permetta la personalizzazione dei parametri. Il modello da simulare dovrà rispettare quello enunciato nell'introduzione, perciò occorrerà sviluppare un'architettura attraverso una tecnologia ad agenti.

3 Analisi del Problema

3.1 Modello dei ruoli

Agente Individuo Una volta generato, ogni AI attende un comando dal proprio AS che può essere di terminazione, di autorizzazione a riprodursi o di migrazione. Se il comando è di migrazione, l'AI migrerà nel nuovo habitat a lui indicato e tornerà in attesa di nuove istruzioni. Se il comando è di autorizzazione a riprodursi, seguirà uno stato di controllo degli A.U.A.²

²A.U.A. significa Alimento Unitario Astratto. Vedi sezione 3.2 pag.6

dell'habitat per constatare l'effettiva possibilità di supplire alla propria fame. Se gli A.U.A. dovessero essere sufficienti, tali alimenti vengono eliminati dall'ambiente e l'AI si riproduce in N figli. E' qui possibile identificare un nuovo stato come lo "stato di vita e/o di morte dei propri figli". Semanticamente in questo caso, l'AI non muore, ma si scinde in più figli e così via; potremmo parlare quindi di *Stato Ricorsivo* di un AI (diagramma degli stati Fig.2).

L'AI presenta due tasks individuali. Qualora venga in possesso dell'autorizzazione dal proprio AS e se nell'habitat ci sono abbastanza A.U.A. per soddisfare il proprio indice di fame, allora l'AI eseguirà il task di *riproduzione*. Se per volere del proprio AS o per mancanza di A.U.A. la riproduzione non può avere luogo, l'AI eseguirà il task di *morte* convertendosi in una quantità di risorsa utilizzabile pari al suo indice di fame.

Il ruolo sociale è quello di fornire il *contributo* alla sua specie derivato dalla sua presenza nel gruppo. Il comportamento emergente da un insieme di AI di una stessa specie è quello di mettere a disposizione all'AS il maggior numero di contributi possibili.

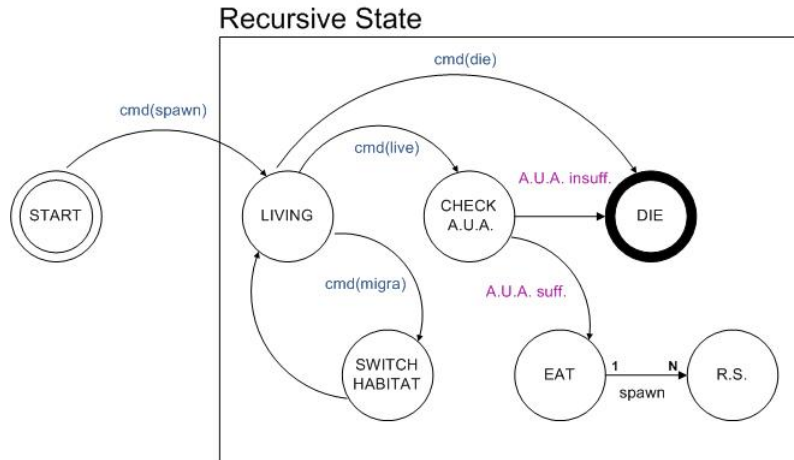


Figura 2: Stati di un AI. Cambiamenti di stato di un AI a seguito delle interazioni. In blu le informazioni da AS e in viola quelle dall'ambiente

Agente Specie Una volta avvenuto il partizionamento delle risorse, il ciclo di vita delle specie viene avviato. Ogni AS prenderà coscienza degli habitats presenti nell'ambiente, genererà l'AI primordiale ed eseguirà ciclicamente un algoritmo di selezione al fine di autorizzare ad ogni ciclo l'AI con l'indice di fame minore (quindi con fitness maggiore). Se dopo una verifica degli

A.U.A. nell'habitat questi risultassero insufficienti a sfamare l'AI selezionato, possono seguire due diversi comportamenti a seconda che l'AI sia l'unico della specie o meno. Nel primo caso, avviene un tentativo di migrazione in un nuovo habitat, nel secondo caso si autorizza ugualmente l'AI a riprodursi portandolo però ad un esito deterministico di terminazione. Se la migrazione della specie ha successo, si ri-autorizza il miglior individuo alla riproduzione sperando che nel nuovo habitat possa avere maggiori possibilità. Se invece la migrazione fallisce (perchè gli habitat erano già stati esplorati tutti) si autorizza comunque l'individuo alla riproduzione conoscendone però già l'esito fallimentare. La specie si estingue quando non sono più presenti AI in vita a causa di insufficienti A.U.A. nonostante tutti i tentativi di migrazione (diagramma degli stati in Fig.3).

Ricapitolando, l'AS gestisce individualmente aspetti riguardanti la popolazione della sua specie (seleziona sequenzialmente gli individui che possono riprodursi, fig.4) con lo scopo di massimizzare il rapporto fra il *contributo* complessivo e l'esigenza di risorse dei vari AI. Inoltre, Gli AS sono mobili, in grado di cambiare habitat per mancanza di risorse (fig. 5). Il comportamento emergente dal sistema è quello di far sopravvivere nel tempo il maggior numero di specie possibile.

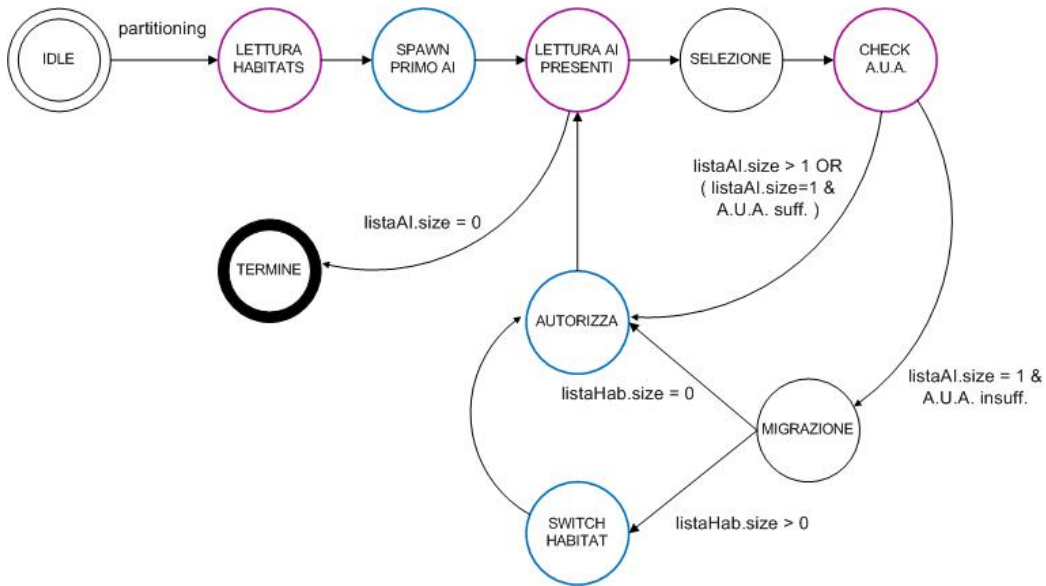


Figura 3: Stati di un AS. In viola gli stati la cui funzione interna richiede interazioni con l'ambiente, mentre in quelli in blu avviene l'interazione coi propri AI

É possibile notare come tutti i task (dalla riproduzione dell'AI alla sopravvivenza degli AS) siano strettamente connessi in modo tale che l'uno sia la conseguenza dell'altro. Non é quindi possibile scindere i task modificandone le caratteristiche pensando che tali modifiche non si ripercuotano sul comportamento complessivo del sistema (in Tab.1 abbiamo una schematizzazione di tutti i tasks).

	Task Individuali	Ruolo Sociale	Task Sociali
AI	Riproduzione. Morte	Fornire contributo unitario	Massimizzare contributi
AS	Far nascere il primo AI, Autorizzare riproduzione, Migrare	Mostare/mettere a disposizione il contributo della specie	Sopravvivere

Tabella 1: Classificazione Tasks. Ogni agente può avere tasks sociali, individuali e un ruolo all'interno della società di suoi simili.

3.2 Modello delle risorse

A.U.A. (Alimento Unitario Astratto) Sono le risorse utilizzate dagli AI (la cui concretizzazione varia a seconda della tecnologia utilizzata) sfruttate e consumate per la riproduzione. Gli AS non hanno un bisogno diretto di tali unità di risorsa; se vogliamo, il nutrimento degli AS é calcolabile in A.U.A. necessarie a mantenere in vita il proprio sottosistema di AI. In riferimento ad un approccio naturalistico, si nota quanto possa essere sottile il confine fra agente e ambiente. In effetti, un essere vivente deceduto diventa a sua volta parte dell'ambiente (sempre che non lo sia stato anche in vita) e come tale può essere riutilizzato dagli altri individui.

Modello degli Artefatti Seguendo il meta-modello A&A possiamo identificare, in maniera astratta sotto il nome di *artefatti*, dei componenti passivi condivisi, manipolati e usati dagli agenti per supportare le loro attività.

Innanzitutto identifichiamo tutti i servizi osservabili dall'ambiente.

Generazione di un AI Sia AS che AI possono accedere a tale servizio. AS vi accederà la prima volta per generare l'AI primordiale. AI vi accederà sotto autorizzazione dell'AS per generare la discendenza.

Consumo di risorse ogni AI, dopo autorizzazione per la riproduzione, consumerà le risorse presenti nell'habitat (in quantità sufficiente).

Comunicazione AS-AI Gli AS devono comunicare con gli AI per autorizzarli a riprodursi, terminarsi o effettuare una migrazione.

Upload post-morte L'ambiente convertirà in risorsa utile ogni AI non in grado di riprodursi. Le risorse generate saranno in quantità pari all'indice di fame dell'AI deceduto.

E' ora possibile raggruppare tali servizi in artefatti in base alle caratteristiche comuni. Una possibile suddivisione può essere la seguente (Tab.2):

- *Inspector*: consente agli agenti di osservare gli elementi presenti nell'ambiente e di quanti habitats è composto il sistema.
- *AI Manager*: fornisce gli strumenti per la generazione e la coordinazione degli AI.

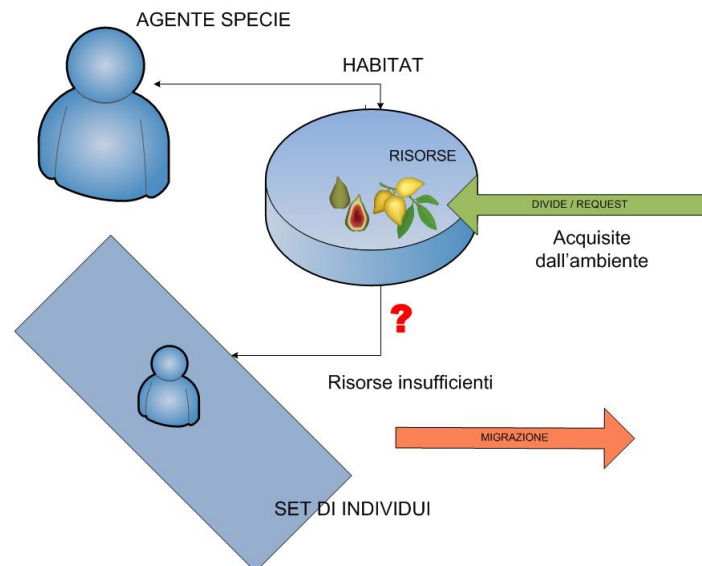


Figura 5: Migrazione. Dinanzi alla mancanza di risorse per l'ultimo AI rimasto in vita, la specie migra in un nuovo habitat

- *A.U.A. Manager*: riorganizza le quantità di A.U.A. a seguito di particolari eventi degli AI.

ARTEFATTO	SERVIZIO	PERMESSO	Input	Output	
				Successo	Fallimento
<i>Inspector</i>	<i>Conoscenza habitat</i>	AS	Richiesta da AS	Lista habitats al richiedente	Lista vuota
	<i>Conoscenza A.U.A.</i>	AI / AS	Richiesta da AS, Richiesta da AI	Quantità degli A.U.A.	Nessuna informazione
	<i>Conoscenza AI</i>	AS	Richiesta da AS	Lista AI viventi entro l'habitat	Lista vuota
<i>AI Manager</i>	<i>Generazione di un AI</i>	AI / AS	Comando da AS, Comando da AI	Nuovo AI nell'habitat della richiesta	Nessuna informazione
	<i>Comunicazione AS-AI</i>	AI / AS	Comando da AS	AI cambierà stato in funzione al comando	Nessuna informazione
<i>A.U.A. Manager</i>	<i>Partizionamento</i>	Agente Esterno	Comando esterno di suddivisione	Habitats contenenti A.U.A.	Habitats non contenenti A.U.A.
	<i>Consumo di risorse</i>	AI	Richiesta da AI	Decremento risorse dall'habitat, Riproduzione del richiedente	Nessuna informazione
	<i>Upload post-morte</i>	AI	Terminazione di un AI	Incremento A.U.A. dell'habitat	Nessuna informazione

Tabella 2: Raggruppamento Servizi. I servizi sono raccolti in artefatti in base alle caratteristiche e agli scopi. Ogni servizio necessita di particolari informazioni di input e fornirà specifici output a seconda del successo o fallimento.

3.3 Modello delle interazioni

Il protocollo delle interazioni associato agli AI In riferimento al diagramma degli stati in Fig.2 possiamo notare che le interazioni di un AI avvengono con il proprio AS e con l'ambiente.

La comunicazione con l'AS è unilaterale (si ricevono solamente autorizzazioni o comandi di terminazione e migrazione senza che l'AI comunichi nulla all'AS) e mediata dall'artefatto *AI Manager*. Una volta nello stato "*Check A.U.A.*" l'AI utilizzerà l'artefatto *Inspector* per verificare le quantità e l'artefatto *A.U.A. Manager* per consumare le risorse utili alla riproduzione. *A.U.A. Manager* viene utilizzato anche per effettuare l'upload delle risorse dell'habitat subito prima che l'AI debba terminarsi.

Il protocollo delle interazioni associato agli AS Da comando innescato da agente esterno (quale l'utente) tramite *A.U.A. Manager* viene avviato il ciclo di vita del sistema. Ne segue per l'AS la lettura dell'habitat con *Inspector* e la generazione del primo individuo tramite un comando verso *AI Manager*.

Sequenzialmente AS ispezionerà i propri AI e verrà effettuato un controllo sulle risorse con *Inspector*. A seguito di particolari condizioni (es. Ultimo AI rimasto con indice di fame maggiore delle risorse presenti) verrà effettuata la migrazione in un secondo habitat mai visitato. La migrazione viene comandata a tutti gli AI della specie ed è mediata con *AI Manager*. Lo spostamento dell'AS in un nuovo habitat segue quello dei propri AI (Fig.6).

Quindi, in riferimento al diagramma degli stati in Fig.3 gli stati in viola richiedono l'utilizzo di *Inspector* mentre quelli in blu l'utilizzo di *AI Manager*.

Non c'è un'interazione né inter-AI né inter-AS anche se della stessa specie. I flussi informativi compaiono solo fra AS e AI attraversando l'ambiente. Tuttavia, condividendo lo stesso habitat, il comportamento di una specie (e quindi dei suoi AI) può dipendere fortemente da quello delle altre: per esempio il caso in cui nello stesso habitat coesistano assieme due specie diverse che necessitano dello stesso tipo di risorsa per sopravvivere.

Nella Tabella 3 sono indicati per ogni agente i tasks individuali, le informazioni ad essi necessarie e quelle fornite a seguito del loro completamento.

Il protocollo delle interazioni relativo agli artefatti si trova in Tabella 2.

3.4 Tool di simulazione

Il modello di cui sopra deve essere simulato in un ambiente che ne permetta l'osservazione e il controllo. Dai requisiti si evince la necessità di un'interfaccia in grado di comunicare all'utente le informazioni riguardo la simulazione e di permetterne l'avvio, l'arresto e la personalizzazione. Occorrerà pertanto aggiungere un'entità al sistema in grado di comunicare informazioni temporali ad AS e AI di procedere con il loro ciclo di vita in accordo con la volontà dell'utente. Modifichiamo perciò il modello aggiungendo le seguenti caratteristiche:

- Al modello dei ruoli aggiungeremo un nuovo tipo di agente, l'*Agente Controller*, che si occuperà di comunicare con l'utente e di inviare agli agenti le informazioni di simulazione.
- Tra i servizi osservabili dall'ambiente avremo anche un servizio che permette di propagare agli agenti le informazioni di progresso quali

AGENTE	TASK INDIVIDUALE	INFORMAZIONI	
		NECESSARIE	FORNITE
Individuo	Riproduzione	Comando "live" da AS	Aggiornamento contributi
		Quantità A.U.A.	Aggiornamento A.U.A.
			Comando generazione AI
	Morte	Comando "die" da AS	Upload A.U.A.
		Quantità A.U.A.	Notifica cambio di stato
Specie	Nascita primo AI	Avvenuta partizione	Comando generazione AI
	Autorizzare	Quantità A.U.A.	Comando "live" da AS
		AI nell'habitat	
	Migrare	Quantità A.U.A.	Comunicazione nuovo habitat
		AI nell'habitat	Comando "migra" da AS verso AI
		Habitats esistenti	

Tabella 3: Tasks e Informazioni. A prescindere da chi le mette a disposizione, ogni task necessita di informazioni per essere avviato. La sua esecuzione fornirà nuove informazioni.

start e stop, nonché un servizio in grado di tenere traccia del tempo occorso per la simulazione. Questi servizi possono essere raggruppati in un ulteriore artefatto che prende il nome di *Simulator*.

- Una simulazione si può ritenere terminata nel momento in cui non c'è più nessuna specie attiva nel sistema. Per tenere traccia di questa informazione, aggiungeremo un servizio di conteggio delle specie vive, da aggiungere all'artefatto *Inspector*.
- Per quanto riguarda il modello delle interazioni, l'AC comunica ad AS e AI, grazie all'artefatto *Simulator*, informazioni quali "start", "stop", "reset" e così via. Grazie a *Inspector* invece, gli AS comunicano indirettamente all'AC la loro attività.

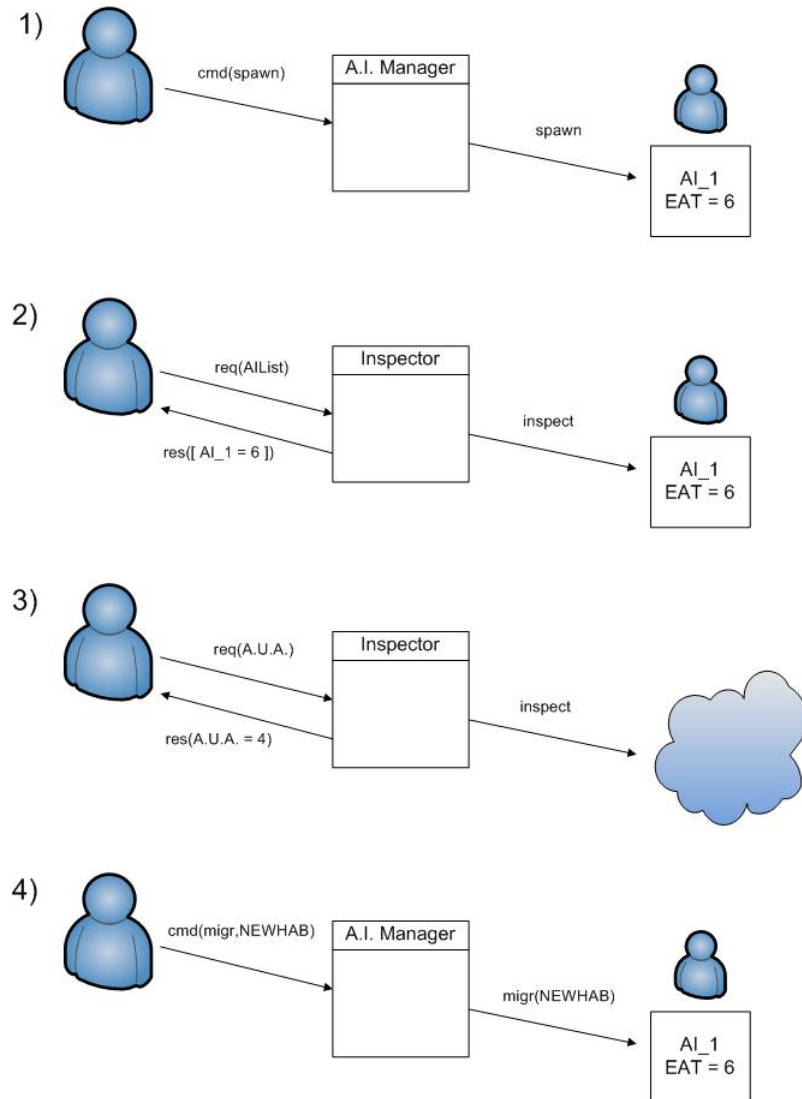


Figura 6: Esempio di Interazione AS tramite artefatti. Alla generazione del primo AI non ci sono già abbastanza risorse per permettere la riproduzione (AI richiede 6 mentre nell'ambiente ci sono 4 risorse), la specie migra.

4 Design

In accordo alle caratteristiche e ai componenti del sistema emersi dalla fase di analisi, dovrà essere selezionata la tecnologia più adatta alla sua implementazione. In particolare dovranno essere disponibili tools in grado di:

- Implementare classi di agenti con le predeterminate caratteristiche.
- Sviluppare un ambiente costituito da più habitats.
- Rendere possibile la comunicazione fra agenti.
- Rendere possibili azioni pragmatiche sull'habitat come il consumo e l'upload di A.U.A.
- Permettere la creazione dinamica di nuovi agenti.
- Fornire all'utente una vista completa dell'intero sistema con la possibilità di focus a livello di habitat.

4.1 Tipologia di Agenti

Gli agenti sono entità autonome. L'autonomia computazionale deriva dall'incapsulamento del controllo il quale rimane circoscritto all'interno dell'agente stesso; per questo motivo non possono essere invocati, non hanno interfaccia e non possono essere controllati. L'autonomia è la proprietà base per la definizione di ogni agente dalla quale discendono ulteriori proprietà. Identifichiamo quindi le correlazioni fra proprietà generali e quelle particolari degli individui e delle specie.

Pro-attività Gli agenti sono pro-attivi: eseguono azioni, provocano eventi piuttosto che attendere che qualcosa accada. In realtà un agente autonomo potrebbe essere costruito e scelto per reagire solamente agli eventi esterni. Può attendere che qualcosa accada come attitudine permanente (quindi puramente reattivo) o come scelta temporanea opportunistica.

Nel nostro sistema compaiono due tipologie di agenti entrambe pro-attive anche se una più pro-attiva dell'altra. Negli AS predomina l'aspetto pro-attivo: a parte il trigger di avvio iniziale è l'agente stesso che a seconda delle condizioni esterne decide l'azione da compiere (come la selezione o la migrazione). Gli AI invece sono prevalentemente reattivi ovvero reagiscono ad impulsi esterni determinati dall'interazione (dai comandi) del proprio AS (come l'autorizzazione a riprodursi, l'ordine di migrare...).

Situatedness & Azioni Ogni agente è correlato da modelli di azioni che sono strettamente connessi al luogo in cui avvengono. Il concetto di azione non ha senso se non si specifica tale contesto. Il contesto d'azione

è dato dagli habitats che con i loro A.U.A. variabili vanno implicitamente a determinare le azioni sia di AS che di AI. Ogni AS cambia lo stato dei propri AI fornendo particolari tipi di informazione. Gli AI cambiano lo stato dell'ambiente (dell'habitat) tramite azioni pragmatiche virtuali come il consumo o l'upload di A.U.A.

Goal Autonomia significa auto-governo, auto-regolazione e auto-determinazione. Non implica la necessità di goal. Si possono prevedere però casi goal-oriented e task-oriented dove goal e task sono determinanti per il controllo.

In EvoMAS il goal di ogni agente non è rappresentabile dallo stato finale di un'ipotetica macchina a stati interni. Paradossalmente, se lo stato finale è la terminazione di un agente, massimizzare il tempo di raggiungimento dello stato finale significa perseguire il goal principale: la sopravvivenza. Il concetto di task si può utilizzare nella misura in cui, per sopravvivere, è necessario che ogni agente (a seconda che sia AS o AI) compia particolari azioni: come la migrazione per gli AS o il consumo di risorse per gli AI.

In riferimento alle considerazioni sopra possiamo classificare gli AI e AS come: autonomi, proattivi, reattivi al cambiamento, sociali e quindi appartenenti alla categoria Weak. Se per agenti Strong intendiamo dei sistemi intenzionali il cui comportamento è esprimibile e prevedibile tramite stati e attitudini mentali come *Belief*, *Desires* e *Intentions* e la cui architettura astratta prevede strutture dati che riflettono tale logica, per quanto le entità in gioco mimino il comportamento di esseri viventi, si ritiene superfluo percorrere tale via implementativa in quanto la complessità computazionale risultante si mantiene generalmente su un livello piuttosto basso.

4.2 Analisi delle Tecnologie

Esistono molte tecnologie più o meno assestate utilizzabili per l'implementazione del sistema presente. La prima è CArtAgO; grazie all'implementazione del modello A&A è possibile in maniera elegante e ordinata gestire le interazioni fra i due tipi di agenti. Abbiamo però limitazioni sull'aspetto principale del sistema, ovvero la possibilità di creare da parte di alcuni agenti, altri agenti (si pensi alla riproduzione) o comunque creare dinamicamente agenti in generale.

L'utilizzo di CArtAgO affrontato nel corso è stato accoppiato a Jason, il quale propone un'architettura BDI per gli agenti. Jason necessita di cono-

scere a priori quali agenti saranno presenti nel sistema; aspetto che contrasta con le nostre caratteristiche.

Un'ulteriore tecnologia è JADE che può interfacciarsi a Jason e quindi anche a CArtaGO. Anche in JADE però rimane il problema visto sopra; infatti gli agenti devono essere creati e lanciati da shell o tramite uno script che li possa inserire singolarmente all'interno del framework.

TuCSon è un'infrastruttura che fornisce servizi di comunicazione e coordinazione di componenti software distribuiti, che incapsulano un proprio flusso di controllo (agenti). Per lo scopo, la sua infrastruttura mette a disposizione *centri di tuple*, che sono spazi di tuple con un comportamento reattivo programmabile dinamicamente tramite un linguaggio logic-based, chiamato ReSpecT.

4.2.1 TuCSon come alternativa dominante

L'infrastruttura TuCSon aderisce notevolmente al modello d'interesse e soddisfa i nostri requisiti.

Implementare classi di agenti TuCSon presenta un'interfaccia di programmazione che permette di sfruttare i suoi servizi da applicazioni scritte in Java o TuProlog. Per accedere all'infrastruttura deve essere acquisito un *agent coordination context* (ACC). Un ACC è una sorta di interfaccia tramite la quale un agente può invocare le primitive di coordinazione sui centri di tuple appartenenti ad un'organizzazione.

Gli agenti possono essere quindi implementati con due linguaggi differenti quali Java e TuProlog. Visto che entrambi gli agenti sono di tipo Weak si abbandona l'idea del linguaggio dichiarativo e si sceglie l'implementazione in Java.

Sviluppare un ambiente costituito da più habitats La topologia di TuCSon prevede molti livelli di aggregazione (centri di tuple sono collezionati in nodi di infrastruttura, distribuiti in una rete, organizzati in domini articolati...) che vanno oltre gli scopi del nostro sistema. Infatti è sufficiente spostare l'attenzione sui TC e le loro interazioni a livello di singolo nodo.

E' evidente un forte mapping fra il modello di habitat EvoMAS e il centro di tuple proposto in TuCSon. In fase di inizializzazione verranno istanziati tanti TC quanti gli habitats voluti, ognuno dei quali avrà un identificativo, una certa quantità di risorse ecc...

Gli agenti che popolano un dato habitat possederanno un riferimento

al TC associato e potranno quindi inserirne all'interno opportune tuple per l'interazione con l'ambiente e con gli altri agenti associati al TC. Tramite l'uso delle tuple è inoltre possibile per ogni agente sottoscrivere la propria presenza nel TC aggiornando anche informazioni sul proprio stato (es: "living" oppure "died").

Rendere possibile la comunicazione fra agenti Tramite il paradigma degli spazi di tuple, la comunicazione avviene tramite l'estrazione e l'inserimento di tuple nel medium di coordinazione che nel nostro caso è rappresentato dal TC. Notare come il TC venga semanticamente associato all'habitat e al medium di coordinazione. Nel nostro caso abbiamo Infatti un esempio di comunicazione stigmergica, dove gli individui del sistema comunicano fra loro modificando l'ambiente circostante.

Rendere possibili azioni pragmatiche sull'habitat Ogni TC conterrà al suo interno particolari tuple rappresentanti le componenti ambientali. In particolare avremo le quantità e il tipo di A.U.A. che gli individui di una specie utilizzeranno per nutrirsi e quindi riprodursi. Ogni individuo potrà quindi leggere e modificare tali valori agendo virtualmente sulle quantità di risorsa sia nel caso di consumo che di upload. Per marcare la separazione fra le azioni pragmatiche sull'ambiente e le interazioni fra agenti e per rendere più trasparente e flessibile il protocollo di coordinazione, viene sfruttato il comportamento reattivo programmabile dei TC. Ogni azione sugli A.U.A. dell'habitat avviene a seguito di reazioni di specifiche ReSpect innescate dall'inserimento nel TC di semplici tuple dagli agenti.

Permettere la creazione dinamica di nuovi agenti Quando un AI trova abbastanza A.U.A. per riprodursi non solo deve attuare modifiche all'habitat ma effettuare anche lo *spawn* dei propri figli. A prescindere dalla nostra scelta di utilizzo di ReSpect, mentre per l'aggiornamento potevamo semplicemente operare a livello di primitive di coordinazione *in* o *out*, ora è invece indispensabile un tool ulteriore che va oltre i semplici concetti del paradigma. Tramite le API TuCSon siamo in grado di permettere ad ogni agente, a fronte di particolari condizioni, lo spawn di nuovi agenti ma ancora una volta si è preferito mantenere semplicità nelle interazioni delegando quindi l'azione fisica dello spawning a specifiche ReSpecT presenti in ogni TC innescate da comandi lato-agente.

Fornire all'utente una vista completa dell'intero sistema Interpretando i centri di tuple secondo il framework concettuale A&A possiamo defi-

nirli *artefatti di coordinazione* e come tali possono godere delle seguenti proprietà: *inspectability* - la capacità di essere osservati e controllati sul proprio stato, struttura e comportamento a runtime, *forgeability* - la capacità di essere adattati e modificati a runtime in accordo a nuovi requisiti, *linkability* - la capacità di essere collegati assieme determinando una composizione dinamica in grado di supplire a funzionalità più complesse, *situatedness* - la proprietà di essere immersi in un ambiente e di essere quindi reattivi ai suoi cambiamenti.

ReSpecT fornisce i tools per soddisfare ognuna di queste proprietà; la *linkability* permette ad esempio di aggregare gli stati e le strutture di ogni TC in un unico centro separato ispezionabile dall'utente che vuole analizzare l'intero sistema.

4.3 Workspace

Il Workspace è il contenitore concettuale di agenti e artefatti, utile per definire la topologia dell'ambiente e il principio di località. In EvoMAS sono presenti tanti *habitats* diversi, che devono comunque essere connessi affinché le risorse possano essere partizionate e il sistema globalmente analizzato. Si pensa di utilizzare una topologia di rete a stella dove ogni *TC habitat* è connesso ad un TC centrale che chiameremo *Environment* (Fig.7).

Environment conterrà tuple informative dinamicamente aggiornabili riguardanti lo stato delle specie, la loro posizione, il numero di AI di ogni specie, il numero complessivo di contributi, le tipologie di habitat e la quantità di risorse in essi contenute. Nei TC *Habitat* le tuple dettagliano la composizione delle specie, come gli AI riferiti al TC, il loro stato, i figli che hanno generato, i contributi complessivi di ogni specie ivi presente, quali specie sono immigrate e quali emigrate. Non è necessario l'uso di tutti i modelli messi a disposizione da TuCSoN per la creazione di reti; nel sistema corrente è sufficiente implementare i TC all'interno di un unico nodo. Tuttavia, avremmo un comportamento equivalente se il sistema fosse distribuito su più nodi diversi.

4.4 Sviluppo degli Artefatti

Abbiamo visto i motivi per i quali i centri di tuple possono essere definiti *artefatti di coordinazione*. Come tali, mediando fra agenti e fra agenti e ambiente, mettono a disposizione un set molto vasto di operazioni.

Gli artefatti del modello EvoMAS emersi in fase di analisi possono essere per la maggior parte inglobati all'interno di ogni TC del sistema e mappati quindi in sezioni di codice ReSpecT. Ogni TC disporrà di un multiset di reazioni

suddivisibile per tipologia di servizio. Il multiset caricato varierà a seconda che il TC rappresenti un habitat o sia *Environment*.

Per lo sviluppo si sceglie quindi di modularizzare in metodi Java le funzionalità degli artefatti EvoMAS che verranno poi caricate in fase di inizializzazione sui rispettivi TC.

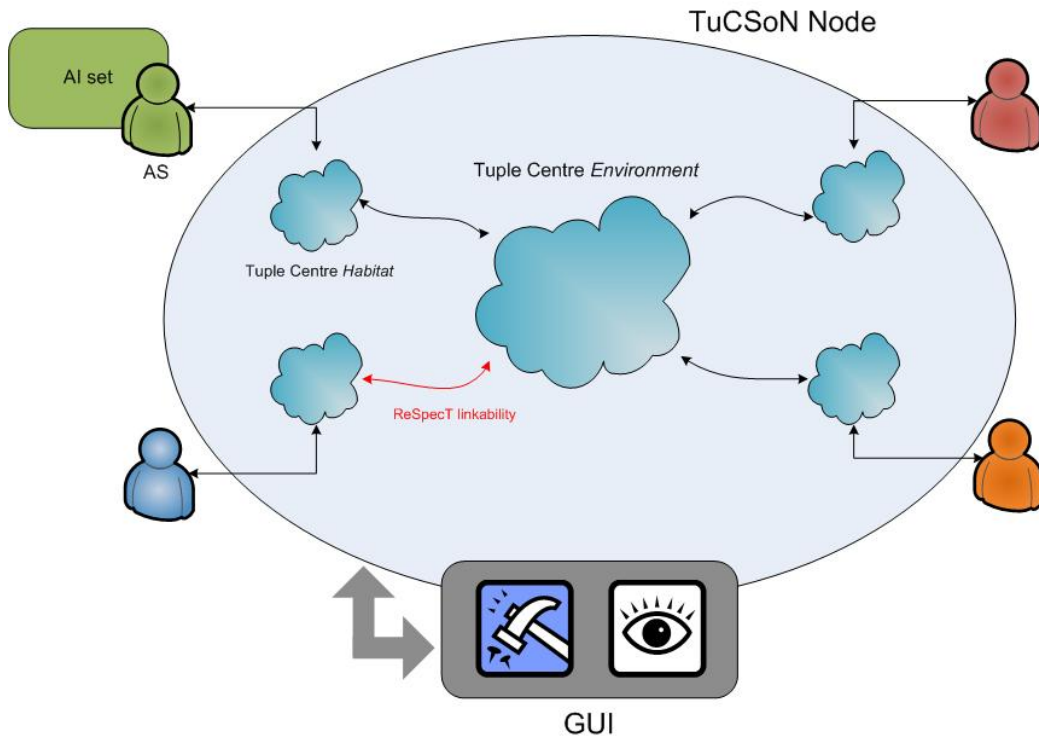


Figura 7: Rete a stella dei TC. Le informazioni presenti in *Environment* vengono mantenute consistenti con quelle presenti in ogni TC grazie all'utilizzo di reazioni ReSpecT. Tramite GUI l'utente può monitorare e interagire globalmente.

Lato *Environment* Segue la lista dei metodi utilizzati in *Environment* e una loro descrizione.

partitioning() Set di specifiche per suddividere equamente gli A.U.A. iniziali su ogni habitat e per lanciare il sistema.

updating() Specifiche di aggiornamento delle tuple di monitoring di ogni habitat e dello stato di ogni specie.

req_hab() Specifica che fornisce alla specie richiedente una lista degli habitats visitabili.

died() Specifiche che controllano la complessiva terminazione di tutte le specie presenti nel sistema.

play() Specifiche di utility per il controllo temporale lato-utente. Ad ogni step, il comando **play** da Controller viene propagato a tutte le specie non estinte che possono quindi proseguire nello step successivo. Viene gestita l'informazione temporale a seguito dei comandi **pause** e **retrieve**.

reset() Gestione del ripristino coordinato fra specie a seguito del comando **reset** sui parametri iniziali.

Lato Habitat Segue la lista dei metodi utilizzati in ogni TC costituente un habitat e una loro descrizione.

rd_all() Specifiche ReSpecT di utility per facilitare la lettura di una lista di tuple con stesse caratteristiche.

in_all() Specifiche ReSpecT di utility per facilitare l'estrazione di una lista di tuple con stesse caratteristiche.

packup() Quando un AS decide di migrare da un habitat all'altro, deve prima ripulire l'habitat corrente dai suoi contributi accumulati e poi comandare ad ogni AI la volontà di migrazione aggiornando il proprio stato in *Environment*.

packdown() Avvenuto il cambio di contesto a seguito della migrazione, ogni AS dovrà inserire nel nuovo TC le informazioni sui contributi estratte dal vecchio habitat come anche gli stati dei propri AI.

rd_hab() Specifica complementare a **req_hab** per la lettura degli habitat visitabili.

spawning() Set di specifiche per semplificare lato-agente lo spawn di un nuovo AI e per aggiornare i contributi e le quantità degli AI correnti.

multispawn() Generalizzazione del metodo precedente con la possibilità di parametrizzare il numero dei figli.

update_res() Specifica per l'aggiornamento locale e globale degli A.U.A. di ogni habitat a seguito dei consumi degli AI.

dietores() Set di specifiche che permettono l'opzionalità dell'upload post-morte per ogni AI e inseriscono in *Environment* l'informazione di estinzione della specie a seguito della sua terminazione.

`clean()` Specifiche per la pulizia degli habitats a seguito del comando `reset`.

Nella Tabella 4 vediamo il mapping fra i metodi sopra e gli artefatti necessari.

ARTEFATTI	METODI JAVA
<i>Inspector</i>	<code>req_hab()</code>
	<code>rd_hab()</code>
	<code>died()</code>
<i>AI Manager</i>	<code>packup()</code>
	<code>packdown()</code>
	<code>spawning()</code>
	<code>multispawn()</code>
<i>A.U.A. Manager</i>	<code>partitioning()</code>
	<code>updating()</code>
	<code>update_res()</code>
	<code>dietores()</code>
<i>Simulator</i>	<code>play()</code>
	<code>reset()</code>
	<code>clean()</code>

Tabella 4: Artefatti e Metodi. Ad ogni artefatto rilevato in fase di analisi sono associati più metodi Java contenenti set di specifiche ReSpecT. I metodi non coprono interamente tutte le funzionalità degli artefatti e possono essere semplici utilities per l'interazione utente.

5 Scenario Applicativo: Gestione files di rete

L'infrastruttura e le caratteristiche comportamentali del sistema EvoMAS lo rendono particolarmente adatto all'interno di scenari applicativi di rete. Uno fra questi è la gestione di files.

Assumiamo di avere una rete di host uno dei quali rappresenta l'amministratore. Per un dato motivo è interesse dell'amministratore acquisire, eliminare o modificare uno o più determinati tipi di files che sono sparsi su tutta la rete; come farlo nel minor tempo possibile a prescindere dal tipo di ordinamento locale dei files?

Ogni rete a seconda della propria struttura e di quella degli host può adottare tecniche euristiche *ad-hoc* per affrontare il problema. La metafora del sistema EvoMAS suggerisce una strategia nature-inspired di valenza generale che astrae dal tipo di rete e dall'implementazione degli host.

Possiamo quindi enumerare i collegamenti fra gli elementi del sistema e quelli del problema:

- **Individuo** : processo che gestisce un certo numero di files (come eliminazione, spedizione all'amministrazione, modifica...).
- **Specie** : team di individui che si spostano su ogni host. Ogni specie tratta una tipologia diversa di files.
- **A.U.A.** : i files da gestire.

Quindi, partendo da un individuo primordiale con un dato indice di fame, verranno trattati tanti files quanto indicato dal suo indice e seguirà così la mitosi in nuovi individui. Se sono presenti sufficienti files per i nuovi AI figli, il ciclo riproduttivo continua nell'habitat, altrimenti avverrà la migrazione della specie in altri nodi. La fitness della specie è in funzione alle quantità di files da trattare in rete. La specie si estingue quando non ci sono più risorse a sufficienza per i propri individui ovvero quando tutti i files dell'intera rete (o la maggior parte di essi) sono stati trattati. Per comprendere l'efficacia di questa scelta, confrontiamola con un'alternativa.

Una strada percorribile dall'amministratore (assumendo di voler mantenere il disaccoppiamento dalle scelte implementative) può essere la creazione di un processo (chiamato Sweeper) che, scansionando ogni nodo, ricerca con particolari tecniche tutti i files di interesse.

Quanti files ci sono su ogni host? Il processo Sweeper secondo la terminologia EvoMAS avrebbe indice di fame pari al numero dei files di interesse presenti nel nodo. Tale quantità è sconosciuta a priori e la sua determinazione necessita di un tempo di computazione non indifferente. EvoMAS crea invece individui con indice di fame predeterminato senza il bisogno di scoprire le quantità totali.

Lavoro concorrente Sweeper esegue sequenzialmente la ricerca di un numero di files che è sconosciuto. Tale ricerca ha fine solo con la scansione dell'intero disco dell'host. Con EvoMAS non solo le ricerche si concludono più rapidamente (una volta raggiunto l'indice di fame) ma avvengono parallelamente da più processi.

Adattattività Il sistema EvoMAS è auto-organizzante. Il numero di individui presenti nella rete è direttamente proporzionale al numero di files di interesse. Man mano che il numero di files da trattare diminuisce, il sistema perderà individui (per mancanza di risorsa) fino alla sua completa estinzione. L'equilibrio del sistema è quindi raggiunto quando saranno scomparsi sia tutti gli Individui che tutti i files da trattare. Tale stato si raggiunge con una dinamica implicita che emerge dal sistema mentre il processo Sweeper necessiterebbe della definizione di politiche di terminazione.

Residui Il sistema EvoMAS a fine esecuzione potrebbe aver mancato di trattare alcuni files (si pensi al caso in cui l'ultimo individuo in vita di una specie abbia bisogno di 7 A.U.A. mentre ne sono presenti solamente 6). Nello Sweeper questo difetto non è presente.

	Offspring range	Hunger Index range	Time (ms)	Steps
	[1,3]	[1,3]	252008	246
EvoMAS	[1,3]	[1,10]	124719	121
	[1,3]	[3,15]	54056	53
Sweeper	[1,1]	[1,1]	205645	202

Tabella 5: Simulazioni comparative fra le due strategie con 200 A.U.A. disposti su 4 habitats. Il sistema EvoMAS acquista efficienza all'aumentare dell'indice di fame di ogni suo individuo. La quantità dei residui rimanenti sarà tanto più alta quanto più alto è il valore del minimo indice di fame.

Da un'analisi conclusiva è evidente la predominanza della strategia EvoMAS su tecniche più triviali come l'utilizzo del processo Sweeper. Tuttavia, EvoMAS presenta delle imperfezioni di esecuzione (il caso dei residui)

che possono essere comunque riparate con l'aggiunta di parti o modifiche euristiche.

6 Conclusioni

Si è così ottenuta un'applicazione in grado di simulare il modello proposto. Il sistema EvoMAS è stato implementato su TuCSoN e la GUI di utilizzo è stata scritta in Java, linguaggio scelto anche per la realizzazione degli agenti. Seguono alcuni snapshot dell'applicazione realizzata: in Figura 8 viene mostrata la finestra di inizializzazione che permette di personalizzare i parametri caratteristici di ogni simulazione. In Figura 9 è possibile osservare l'interfaccia principale, che contiene gli strumenti per l'interazione con l'utente durante la simulazione e la vista complessiva del sistema; è possibile anche il focus sui singoli habitat cliccando sul circolo che li rappresenta (Figura 10).

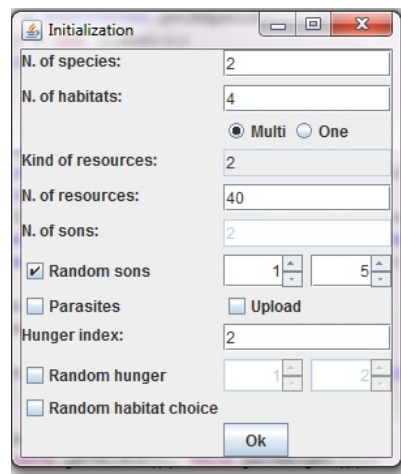


Figura 8: Finestra di inizializzazione

6.1 Possibili estensioni

Esistono molte possibilità di estensione del sistema ottenuto.

- Ad esempio si può estendere il modello corrente:
 - disaccoppiando le specie dai singoli habitat, ovvero permettendo la presenza di individui della stessa specie in più luoghi;
 - inserendo un criterio di selezione più articolato rispetto all'indice di fame;
 - ...

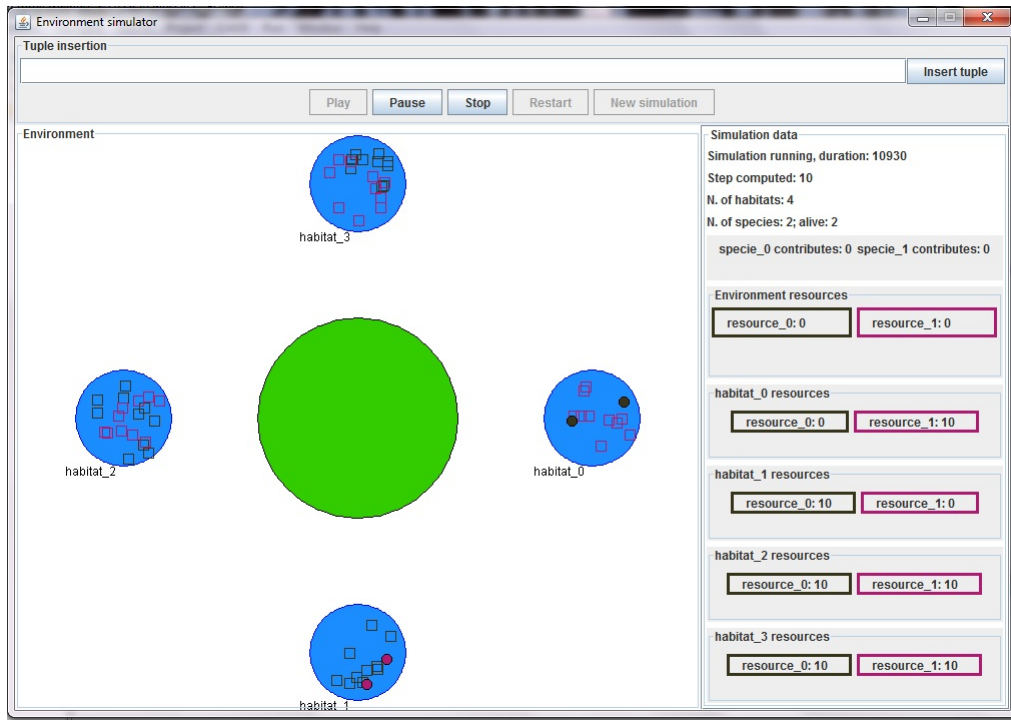


Figura 9: Finestra principale

- L'interfaccia potrebbe essere ampliata permettendo una ulteriore parametrizzazione delle caratteristiche della simulazione e la visualizzazione di statistiche a grana più fine.
- Ragionando sull'*Agente Controller* che si occupa di monitorare l'andamento della simulazione, potrebbe essere interessante, attraverso l'utilizzo di tecniche di controllo adeguate, il concepimento di un meta-agente di livello più alto, il cui compito sia quello di eseguire il tuning dei parametri di simulazione in base ad un certo criterio. Questo criterio potrebbe essere più o meno premiante rispetto a una certa caratteristica della simulazione, come il tempo di vita del sistema o il numero di contributi di ogni specie. Il compito del meta-agente consisterebbe perciò nel controllo di un certo numero di simulazioni, volte al miglioramento dei parametri allo scopo di perseguire l'obiettivo suggerito dal criterio stesso. Al termine del processo, l'agente dovrebbe essere in grado di dimostrare di aver migliorato il sistema e di fornire una serie di statistiche sull'evoluzione dei parametri.

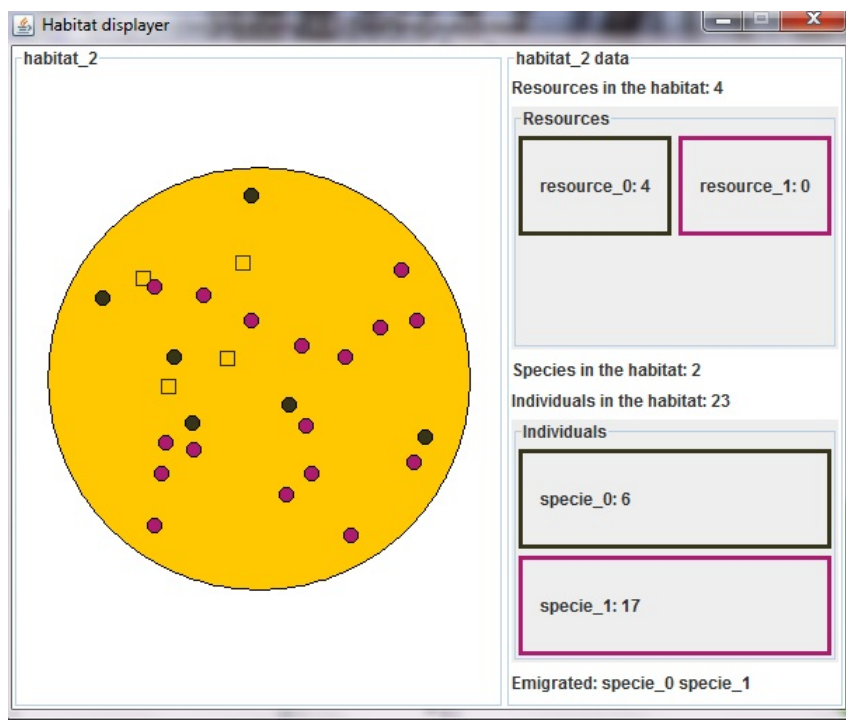


Figura 10: Finestra che permette il focus su ogni singolo habitat

Riferimenti bibliografici

- [1] A. Omicini: *SODA: Societies and Infrastructures in the Analysis and Design of Agent-based Systems*
- [2] A. Omicini: *Formal ReSpecT in the A&A Perspective*
- [3] F.Zambonelli, M. Viroli: *Towards Eternally Adaptive Pervasive Service Ecosystems*
- [4] F. Zambonelli, N. R. Jennings, M. Wooldridge: *Developing Multiagent Systems: The Gaia Methodology*