Università degli studi di Bologna

Facoltà di Ingegneria
Corso di laurea magistrale in ingegneria e scienze informatiche

Sistemi Distribuiti

Zombie Outbreak

Neri Alessandro
Matricola: 0000741557

Trapanese Thomas
Matricola: 0000685720

Indice

1 Introduzione

Il progetto è realizzato nel contesto del corso di Sistemi Distribuiti e mira a dimostrare l'apprendimento delle tematiche trattate durante le lezioni.

Zombie Outbreak riprende i concetti già esposti nel progetto *Virus X*, nella quale si modellava la diffusione di una malattia. Durante l'analisi, basandoci sui problemi riscontrati nella loro implementazione, verrà rimodellata la simulazione ponendo al centro del sistema le città e le reazioni intraprese dal governo per contenere l'epidemia.

Nella realizzazione dell'applicativo, oltre ad utilizzare Jade come framework java per sistemi ad agenti, introdurremo TuCSon (e ReSpecT) come mezzo per la comunicazione e coordinazione e TuCSon for Jade come ponte tra le due tecnologie.

Nel corso della relazione cercheremo di spiegare dettagliatamente le modalità di utilizzo delle varie tecnologie, illustreremo la logica seguita in fase di progettazione, implementazione e gli eventuali problemi riscontrati in corso d'opera.

Infine proporremo delle possibili estensioni per la quale il progetto è stato predisposto e che rappresentano una buona base per eventuali evoluzioni future.

1.1 Requisiti

Realizzare una simulazione che mostri la diffusione del virus z. La simulazione deve essere temporizzata, distribuita e deve prevedere un'interfaccia grafica di controllo per la gestione delle configurazioni iniziali e per l'analisi dello stato della simulazione.

Punto cruciale saranno le azioni intraprese dal governo centrale per contrastare l'epidemia e l'analisi della sua diffusione tra le diverse città.

1.2 Analisi dei requisiti

Stando ai requisiti, la simulazione si focalizza sul modellare le misure di contrasto e la diffusione su grande scala. Ci è sembrato quindi molto sensato scegliere la città come entità base del nostro sistema. Le persone ed il numero di infetti saranno ricondotti ad essere una proprietà di quest'ultima.

Per gestire le azioni di contrasto necessitiamo di un'altra macro-entità: il governo. Questo si occuperà di analizzare lo stato del città e decidere quali azioni intraprendere.

Per la diffusione della malattia tra città sarà necessario implementare un algoritmo basato sulla distanza e quindi anche un'interfaccia grafica che permetta la definizione della distribuzione spaziale delle città.

1.3 Analisi del problema

La simulazione deve essere sia distribuita che temporizzata, questo implica la creazione di un meccanismo di sincronizzazione tra le varie entità.

La simulazione richiede l'implementazione di diversi algoritmi (diffusione all'interno della città, migrazione di persone verso altre città, eventuali reazioni, ecc..) questi dovranno essere eseguiti dagli agenti città in ogni unità di tempo.

Le azioni del governo centrale sono perfettamente implementabili come reaction in RespeCt.

Gli agenti città saranno presenti con una certa cardinalità e saranno quelli che avranno il maggior onere di calcoli, incentriamo quindi proprio su queste la "distribuzione" su varie macchine. Per distribuire gli agenti città su diversi nodi abbiamo due strade:

1. Distribuzione casuale
2. Introduzione di un qualche "raggruppamento"

Scegliamo di raggruppare le città in paesi, che verranno distribuiti su tutte le macchine partecipanti alla simulazione.

L'introduzione dei paesi rappresenta solo una suddivisione logica e non influisce sugli algoritmi ma solo sulla topologia del sistema.

2 Progettazione

Individuiamo i componenti principale dell'applicazione e analizziamone le principali caratteristiche.

1. Governo: avvia la simulazione e si occupa di gestire le azioni di contrasto all'epidemia
2. Città: é il componente centrale della simulazione, deve mappare il numero di cittadini, il numero di infetti, gestire eventuali cittadini in uscita e in arrivo e applicare le azioni decise dal governo centrale.
3. Timer: gestisce lo scorrere del tempo e raccoglie i dati dalle varie città a fine giornata
4. Setting: Intercetta modifiche alle configurazioni e le propaga alle altre entità

Governo

Il governo è l'entità di avvio della simulazione, è modellato come un agente JADE e comunica con il tuple centre grazie alle primitive della libreria T4j. L'entità "governo"

ha una componente che si occupa delle reazioni implementata come uno Spaw (*RiskResponseSpawn*) che viene chiamato alla fine di ogni giornata. Le possibili varie azioni vengono lanciate quando il numero di infetti supera una certa percentuale (configurabile dall'utente). Attualmente sono implementate:

1. Invio di task force: l'invio di una task force con l'intendo di ridurre il numero di infetti. La task force una volta arrivata in città rimane fino a quando non l'epidemia non è debellata. La percentuale di infezione a nella quale viene inviata la taskforce è cruciale per la riduzione del numero di infetti. La percentuale di default a cui viene inviata è 30%

2. Messa in quarantena: la città può essere messa in quarantena, questo comporta l'impossibilità per le persone di entrare o uscire dalla città. Viene implementata rimuovendo il behaviour relativo alla migrazione dalla città. La percentuale di default necessaria alla quarantena è 50%

3. Bombardamento: la città viene completamente annientata, grazie al lancio di una bomba nucleare il governo centrale azzera la popolazione della città debellando di fatto la malattia. Rappresenta la soluzione estrema ed infatti al percentuale di default è settata al 90%.

Città

Le città vengono modellate come agenti JADE. Ogni città appartiene logicamente ad un solo paese. Questi ultimi vengono mappati sui *Container* JADE.

Ogni agente dispone di un behaviour "giornaliero" implementato estendendo la classe *DailyBehaviour*. Ogni behaviour giornaliero è costituito da più sotto-behaviour, nel dettaglio:

1. MovementBehaviour: si occupa di gestire le migrazioni di persone tra le città. Ogni giorno un certo numero di cittadini (sia sani che infetti) si avviano verso un'altra città, il giorno di arrivo viene stabilito dall'algoritmo in base alla distanza tra le città.

   Il behaviour di occupa anche di controllare eventuali migranti in arrivo.

2. InfectionBehaviour: Gestisce la propagazione dell'infezione all'interno della città

3. DeadBehaviour: Calcola il numero di morti dovuti all'infezione.

4. TaskForceBehaviour: Se in città è presenta una task force avvia l'algoritmo che calcola il numero di infetti uccisi.

5. PublishBehaviour: Behaviour che si occupa della pubblicazione della tupla contenente lo stato della città.

Riportiamo uno schema grafico che illustra nel dettaglio la struttura dell'agente città.

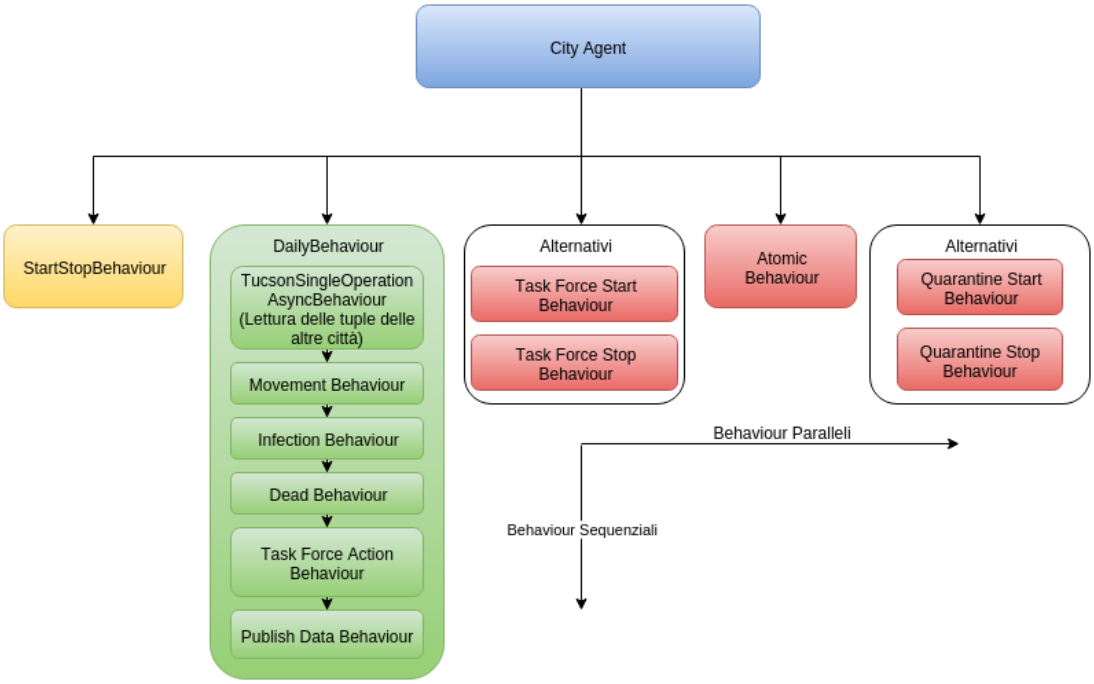

Figura 1: Schema dei behaviour dell'agente città

Oltre ai behaviour principali già presentati, la città incapsula altri comportamenti in grado di interpretare le azioni ordinate dal governo centrale.

Timer

Il timer è un agente Jade ed ha una forte interazione con il tuple centre. Gestisce lo scorrere del tempo, invia una tupla di start of day, attende che tutte le città abbiano completato il calcolo, infine invia la tupla di end of day.

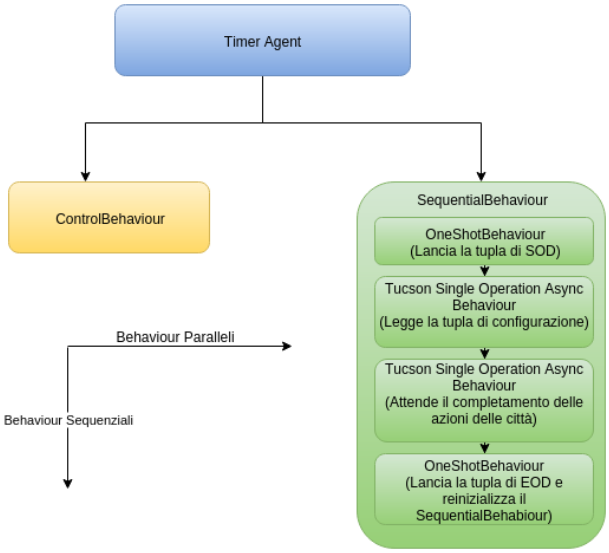

Figura 2: Schema dei behaviour dell'agente timer

Settings

L'agente Setting ha due compiti:

- Carica le reaction del governo
- Gestisce il cambio delle configurazioni utente

La gestione delle configurazioni è stata implementata tramite un bus di comunicazione. In questo modo la logica dell'applicativo è completamente staccata dalla sua rappresentazione grafica. Settings si aggancia all'evento di cambio delle impostazioni e pubblica la tupla con i nuovi settaggi.

2.1 Architettura del sistema

In questa sezione illustreremo la gestione delle principali problematiche architetturali.

2.2 La coordinazione

La gestione della coordinazione nel sistema avviene interamente tramite pubblicazione e lettura di tuple sul Tucson Tuple Centre. A scopo didattico, tuttavia, si è scelto di gestire le migrazioni delle persone e infetti tramite scambio di messaggi ACL JADE.

2.2.1 Jade Behaviours

Per la gestione dell'applicazione è stato utile implementare alcuni behaviour personalizzati elencati qui di seguito:

TucsonSingleOperationAsyncBehaviour

E' un Behaviour che si occupa di effettuare una lettura asincrona sul tucson tuple centre. Accetta nel costruttore le coordinate al tuple centre e l'azione da eseguire. Al suo interno incapsula le api di t4j in grado di comunicare con i servizi di TuCSon. In particolare il nostro behaviour sfrutterà una chiamata asincrona di tipo *inturrupt*. Questo significa che riceverà una notifica sul completamento dell'operazione verso TuCSoN.

Il behaviour, ricevuta la notifica, chiamerà un eventuale funzione di callback (nel caso si voglia effettuare della logica sul termine dell chiamata) e termina.

DailyBehaviour

E' una classe astratta che estende SequentialBehaviour.

Si occupa di gestire le operazioni quotidiane e può essere istanziato con un numero arbitrario di sub-behaviour, che rappresentano la sequenza di operazioni che la città esegue durante la giornata.

L'esecuzione viene avviata alla ricezione della tupla di Start Of Day (inviata dal timer agent).

Al termine dell'esecuzione aspetta la tupla di End of day e ricomincia il ciclo.

La lettura delle tuple di SOD e EOD avviene grazie al TucsonSingleOperationAsyncBehaviour.

HandleSimulationStateBehaviour

Estende parallel behaviour ed è implementato dagli agenti Città e Timer. Il behaviour rimane in *ascolto* di eventuali cambi di stato della simulazione leggendo la tupla *simulation(X)*. Di particolare importanza sono le funzioni di hook che fornisce agli agenti che lo utilizzano:

1. onSimStop();
2. onSimResume();
3. onSimPause();
4. onSimResume();

Facendo l'override di quei metodi è possibile specificare le azioni da eseguire sul cambio di stato dell'intera simulazione.

2.2.2 TucSon e RespecT

Grazie all'utilizzo del servizio per Jade "T4j", abilitiamo sugli agenti JADE la possibilità di comunicare tramite un tuple centre, sulla quale avverrà la maggior parte della coordinazione della simulazione.

In particolare:

1. Avvio/Pausa/Terminazione: tramite la pubblicazione della tupla *Simulation(X)* l'agente governo pilota lo stato corrente della simulazione.
2. Scandire dei giorni: tramite le pubblicazioni delle tuple di start/end of day: *sod(X)* e *eod(X)* da parte del timer
3. Cambio di impostazioni: eventuali cambi di impostazioni a runtime, saranno riportati sul tuple centre dall'agente *Settings* in modo da essere disponibili a tutti i componenti della simulazione
4. Applicazione delle misure di emergenza: alla fine di ogni giornata verrà lanciato uno *Spawn* in grado di valutare quale risposta all'infezione applicare su ogni città

5. Calcolo del numero di viaggiatori: è una tupla calcolata tramite "chained reactions". Ogni volta che la città pubblica il numero di partenze/arrivi il totale viene aggiornato automaticamente.

6. Statistiche giornaliere: alla fine di ogni giornata uno *Spawn* raccoglierà le informazioni di ogni città e del numero di viaggiatori totale e lancerà un evento in grado di aggiornare le informazioni nella GUI.

2.3 Gli algoritmi

Gli algoritmi implementati attualmente sono di 4 tipi:

1. Infection: L'algoritmo aggiorna il numero di infetti, il calcolo è basato su una funzione esponenziale che dipende da: un fattore di calcolo, dalla popolazione della città, dall'inverso del giorno di infezione.

2. Migration: Determina le migrazioni in partenza da una città verso tutte le altre, calcola il numero di migranti sani, infetti e il tempo di percorrenza, non viene eseguito se la città di partenza o quella di destinazione sono in quarantena.

3. Dead: L'algoritmo riduce la popolazione di infetti e sani in maniera equivalente, si basa su una funzione random che varia tra 2 fattori percentuali. Viene eseguita solo in presenza di infetti.

4. TaskForce: L'algoritmo riduce la popolazione di infetti di una quantità random indipendente dal numero di infetti o dalla popolazione della città.

2.4 La configurazione

Da interfaccia grafica è possibile agire su alcuni dei parametri della simulazione.

2.4.1 Il tempo virtuale

Uno slider che permette di modificare la velocità con cui si svolge la simulazione. L'agente timer, prima di avviare un nuovo giorno, leggerà questo valore e applicherà il tempo virtuale corrispondente.

2.4.2 Le soglie per le misure di emergenza

Tramite tre slider è possibile regolare le soglie (espresse in percentuale di infetti sul totale della popolazione cittadina) dopo cui verranno applicate le misure di emergenza. In particolare abbiamo la soglia per l'intervento della task force, dell'applicazione della quarantena e dello sgancio della bomba atomica.

2.5 La distribuzione su più nodi

La simulazione è stata progettata per permettere la distribuzione dei paesi (con relative città), su nodi distribuiti. Avviando il programma con l'opzione *-remote*, creiamo un nodo della simulazione in attesa di ricevere richieste (porta 9999).

Il servizio di avvio si accorgerà della presenza di nodi remoti e suddividerà i vari paesi (jade container) tra questi.

Quando viene richiesta la creazione di un container, il nodo remoto si occupa di istanziarlo e registrarlo sull'AMS. Quanto il container è registrato si procede con l'avvio di tutte le città remote presenti in quel paese. (Per maggiori dettagli si veda il package *mobility*)

A livello di coordinazione non viene apportata alcuna modifica. Il tuple centre rimane centralizzato ed ogni agente (locale e remoto) viene istanziato con i riferimenti per la corretta comunicazione.

2.6 La simulazione

La simulazione viene gestita interamente da una interfaccia grafica implementata in swing, la cui pagina principale presenta, in alto, uno slider che permette di definire la "velocità" della simulazione. Nella parte centrale sono presenti 3 tab, la prima permette di definire la configurazione delle città (Figura 3) per la simulazione, la seconda riporta il log della simulazione, la terza contiene altre configurazioni tra cui le soglie per le reaction del governo (task force, quarantena, bomba atomica) e la configurazione di altri nodi da utilizzare nella simulazione (Figura 4).

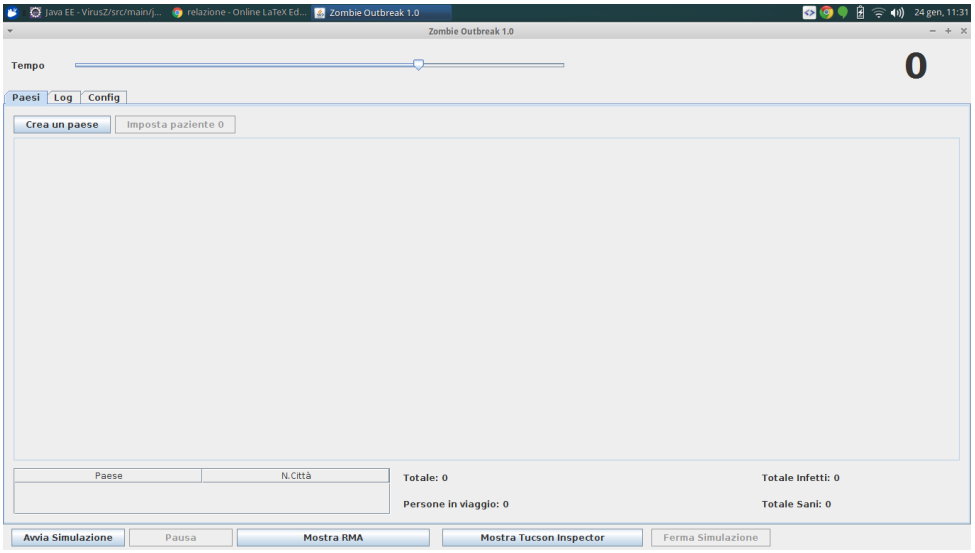

Figura 3: Schermata principale dell'applicazione

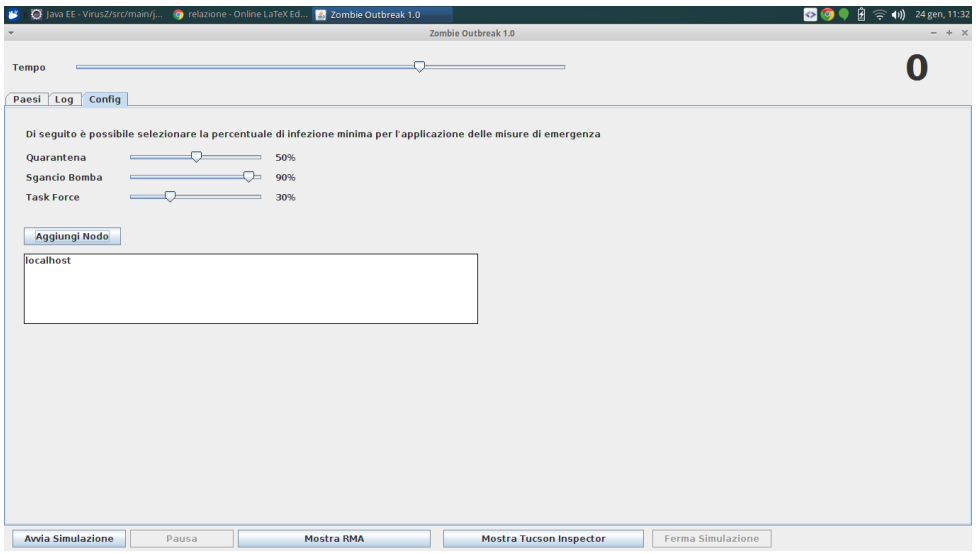

Figura 4: Schermata principale dell'applicazione

In basso sono sempre presenti i dati della simulazione: una tabella contente i paesi definiti e il numero di città per ognuno di essi; I dati relativi alla popolazione suddivisi per sani, infetti e in fase di migrazione.

Il primo requisito per avviare al simulazione è la presenza di almeno un città. Per definire le città bisogna premere il pulsante "Crea un paese", verrà presentata una input nella quale inserire un nome. A questo punto premendo sulla mappa verranno aggiunte delle città, nella versione attuale la popolazione viene caricata in maniera random e il nome delle città è dato dal nome del paese più un numero progressivo. Una volta terminata la definizione delle città premendo il pulsante "fine" la modifica diverrà effettiva. Si possono creare un numero arbitrario di paesi e di città ripetendo l'operazione.

Per la definizione dell'origine dell'epidemia si deve assegnare ad una o più città il paziente 0, premendo l'omonimo pulsante e selezionando, col mouse, la città nella quale lo si vuole posizionare.

Giunti a questo punto la simulazione può essere avviata, le configurazioni possono essere cambiate anche a simulazione avviata.

Quando la simulazione è in corso, la grafica verrà aggiornata alla fine di ogni giorno, su ogni città vengono indicati il numero totale di cittadini, il numero di infetti (con percentuale sul totale), lo stato della città (infetta, non infetta, quarantena, bombardata) e la presenza di una task force in città, indicata dalla dicitura "(TF)" in coda allo stato (Figura 5).

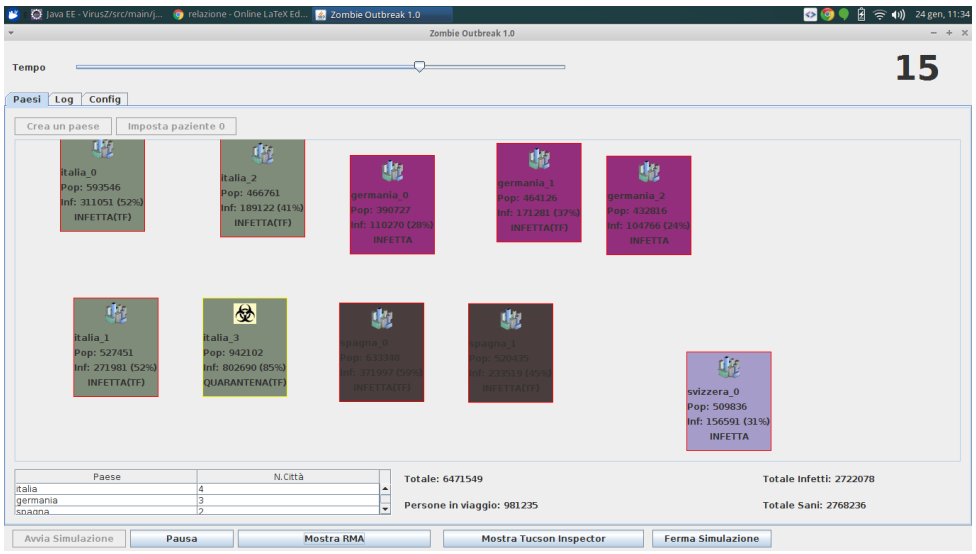

Figura 5: Schermata principale durante la simulazione

Nella pagina di log (Figura 6) si possono vedere le informazioni dettagliate su quanto avviene in tempo reale.

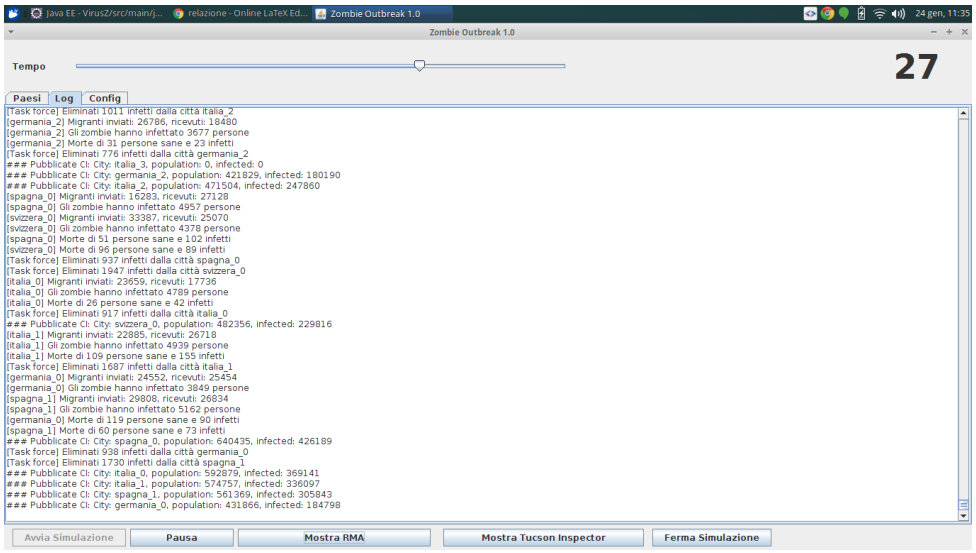

Figura 6: Pagina di log durante la simulazione

In ogni momento è possibile terminare o mettere in pausa la simulazione.

Sono inoltre disponibili i pulsanti di utilità "Mostra RMA" e "Mostra Tucson Inspector" che permettono di avviare agevolmente le utility di introspection disponibili per JADE e TuCSoN (Figura 7).

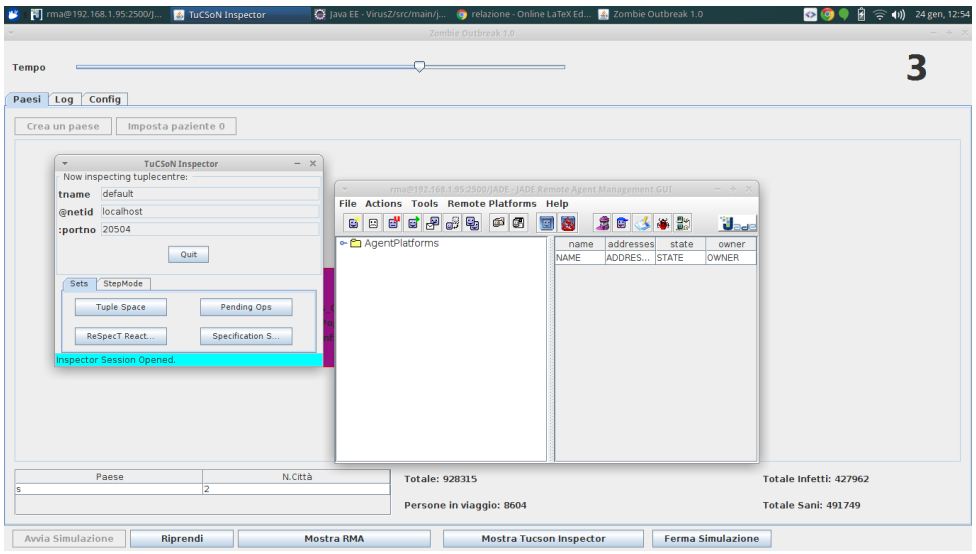

Figura 7: RMA e Tucson Inspector

2.7 Possibili evoluzioni

Durante la fase di test e collaudo sono emerse diverse modalità secondo cui si potrebbe proseguire nello sviluppo del progetto:

1. Quando distribuiamo i nodi potrebbe essere ottimale distribuire anche il tuple centre, decentralizzandolo dal nodo principale che sta eseguendo la simulazione.
2. Rendere gli algoritmi configurabili da interfaccia
3. Predisporre un salvataggio delle informazioni giornaliere, per dare la possibilità di navigare i diversi giorni a fine simulazione
4. Prevedere una struttura a plug-in per dare a gli utenti la possibilità di aggiungere le proprie implementazioni degli algoritmi
5. Dare agli utenti la possibilità di configurare il numero di abitanti in ogni città
6. Dare agli utenti la possibilità di interagire attivamente durante la simulazione, ad esempio forzando l'invio di una taskforce o lo sgancio di una bomba atomica.

3 Conclusioni

Durante lo sviluppo di zombie outbreak ci siamo posti come obiettivo, tramite la realizzazione di un simulatore della diffusione di una malattia altamente infettiva, quello di studiare e implementare un'architettura per sistemi distribuiti, analizzando nel dettaglio il *"paradigm-shift"* introdotto dal rappresentare i diversi componenti del sistema come entità pro-attive in grado di incapsulare specifici comportamenti: gli agenti.

Ci siamo imbattuti più volte nei principali problemi di coordinazione che ruotano attorno alla progettazione dei sistemi distribuiti (dalla gestione dell'avvio dei componenti fino alla gestione dell'unità di tempo del sistema, le giornate).

Questi sono stati prontamente risolti combinando la struttura degli agenti jade con la visione distribuita del sistema tucson e la relativa programmazione del tuple centre.

Abbiamo cercato di riutilizzare le esperienze di chi, prima di noi, aveva realizzato un progetto con queste tecnologie. Siamo partiti da un'idea introdotta precedentemente, quella della diffusione di un virus, e abbiamo fatto scuola dell'esperienza maturata in quel progetto per superarne i limiti ed introdurre nuovi aspetti.

Infine abbiamo predisposto il codice per considerevoli espansioni che, per non uscire dagli obiettivi di questo esame, non abbiamo implementato ma che abbiamo esposto nel corso della trattazione.

Speriamo di essere riusciti a coprire, in maniera sufficientemente esaustiva, i principali temi architetturali e tecnologici affrontati durante il corso, offrendo una panoramica su quelle che sono le modalità di interazione e coordinazione che governano un sistema distribuito multi-agente.

4 Reference

Source Code: https://bitbucket.org/trapthom/sistemi-distribuiti/src

Jade: http://jade.tilab.com/

TuCSoN: http://apice.unibo.it/xwiki/bin/view/TuCSoN/

TuCSoN4JADE: http://apice.unibo.it/xwiki/bin/view/TuCSoN/4JADE