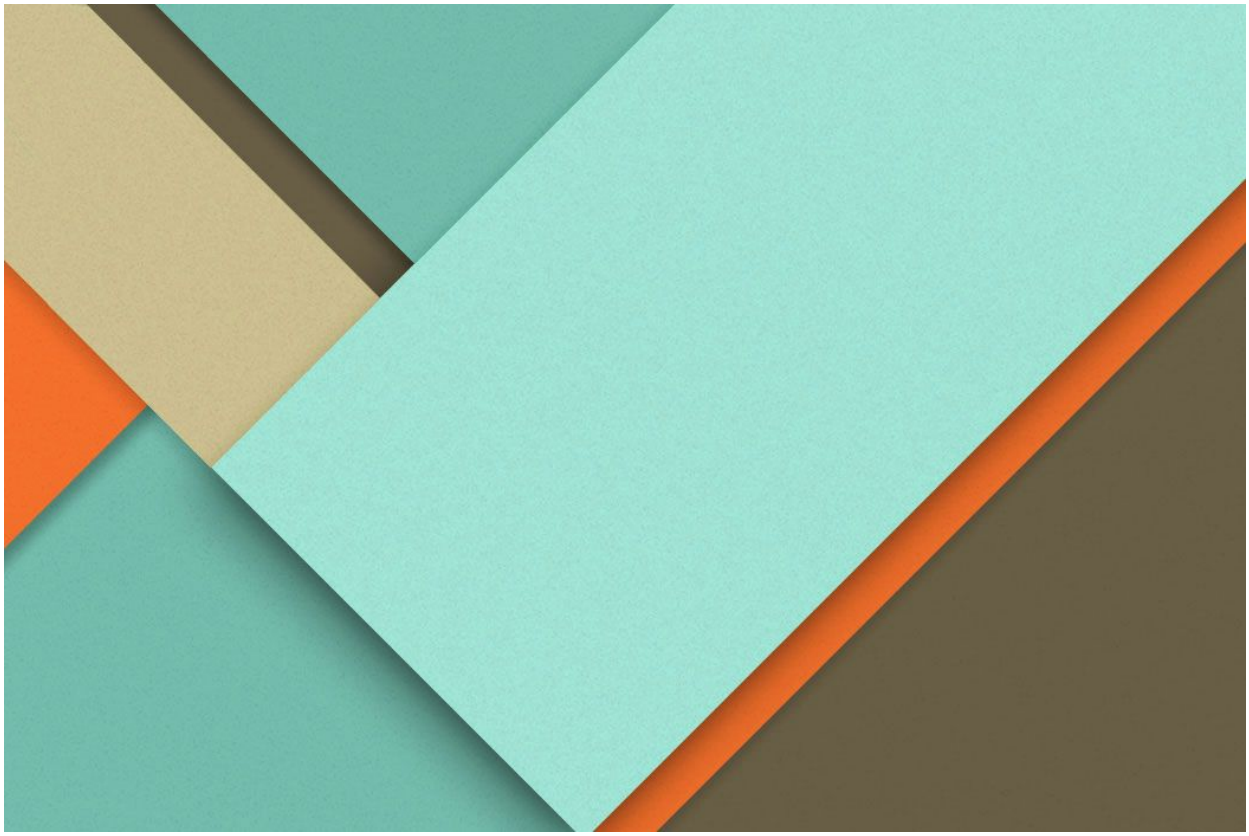


Virus X

Progetto di Sistemi Distribuiti

Mattia Barbaresi
mattia.barbaresi@studio.unibo.it

Matteo Bocchetti
matteo.bocchetti@studio.unibo.it

Francesco Maughelli
francesco.maughelli@studio.unibo.it

Introduzione

Lo scopo principale di questo progetto è quello di realizzare un simulatore che implementi correttamente un modello, da noi proposto, di diffusione di una malattia all'interno di una popolazione urbana. La finalità del progetto è stata quindi duplice: da una parte quella di affrontare sia la difficoltà di progettazione e di successiva implementazione di tutto l'applicativo; dall'altra come caso di studio per testare l'efficienza, la scalabilità e la solidità del sistema software distribuito JADE (Java Agent DEvelopment Framework), sviluppato dal gruppo di http://jade.tilab.com/.

E' stato scelto questo framework per le sue peculiarità multi-agente e per la facilità con cui mette a disposizione le dovute strutture dati e primitive orientate da subito ad operare con entità complesse e reattive che possono essere dotate di comportamenti propri. Un sistema JADE può essere distribuito su più macchine indipendentemente dal proprio sistema operativo ed il tutto può essere controllato mediante una gui remota. Tutto il framework di JADE è basato sulle specifiche FIPA (standard della IEEE che promuove sia l'utilizzo di tecnologia basata sugli agenti, sia la loro integrazione con gli altri standard).

Modello

Il modello tenuto in considerazione è stato pensato per poter essere utilizzato in un contesto dove verranno impiegati gli agenti e dunque alcune delle sue semplificazioni sono dovute proprio alla necessità di dover affrontare questa architettura. La malattia e l'individuo sono i soggetti centrali all'interno del nostro modello; un individuo è, infatti, l'unità minima che viene misurata e su cui viene applicata una malattia. Per poter riuscire ad implementare queste esigenze è stato scelto di semplificare le caratteristiche di base dei nostri soggetti fondamentali al minimo. Tale scelta faciliterà non solo l'implementazione iniziale ma getterà le basi per un eventuale ampliamento del simulatore nel futuro.

I soggetti primari che compongono il modello sono:

1. **Malattia:** è caratterizzata da un nome, una durata in giorni, un periodo di incubazione ed un tasso di infettività percentuale. Sia l'incubazione che la durata hanno anche una tolleranza percentuale che ne modifica i rispettivi valori nel margine massimo stabilito. All'inizio di ogni simulazione verranno scelti degli individui all'interno dello scenario che avranno già contratto la malattia.

2. **Persona:** il soggetto fondamentale del modello. Verrà creata una popolazione di individui che agiranno indipendentemente e che avranno il ruolo sia di "cavie", sia di portatori sani oppure di malati infettivi. La persona è altresì il mezzo con cui viene veicolata la malattia all'interno del simulatore ed è il più complesso dei soggetti.

3. **Quartiere:** la topologia di base che dovrà mappare il modello in questione è stata pensata per essere quanto più simile ad una città, suddivisa in diversi quartieri. Ogni individuo, infatti, con il passare del tempo si sposterà da un quartiere ad un altro e l'unità minima dove può essere infettato da un altro individuo è collocata all'interno dello stesso quartiere. Questo rappresenta un vincolo rilassato per poter meglio descrivere il problema; un individuo malato che transiti in un quartiere infatti "genera" una probabilità di infezione che verrà calcolata singolarmente in ogni persona.

4. **Città:** diversi quartieri andranno a comporre delle vere e proprie città. Tuttavia, mentre per spostarsi internamente è sufficiente un mero "passo" da parte di un individuo i collegamenti tra città verranno invece gestiti da autostrade caratterizzate da una portata massima.

5. **Tempo:** la gestione del tempo simulato (virtuale) deve essere in qualche modo sincronizzata tra i vari soggetti che compongono la simulazione. La nozione di "simulazione" in quanto tale definisce che, a parità di input e scenario iniziale, una computazione di una certa durata virtuale deve garantire gli stessi identici risultati e deve essere altresì deterministica.

Pianificazione

Il modello e la piattaforma JADE:

Il modello così proposto richiede ovviamente una solida integrazione con le caratteristiche architetturali e fisiche della JADE platform. Dato che le peculiarità e le potenzialità della piattaforma non sono note in anticipo abbiamo dovuto affrontare dei casi di studio ad-hoc per la risoluzione di dipendenze interne e di vincoli di progetto.

Un'altra incognita è se la platform così strutturata possa *effettivamente* sopportare un carico di lavoro multi-agente abbastanza "affollato". Per ovviare a tali ipotesi valuteremo il risultato ottenuto sia dal punto di vista di una singola macchina, sia attraverso soluzioni più o meno distribuite.

Architettura di JADE:

JADE è costituito essenzialmente da uno o più runtime environment chiamati *container* all'interno dei quali è possibile creare degli agenti. Un set di container è definito come *platform* e rappresenta il nodo centrale del sistema JADE all'interno della macchina. I container devono, infatti, essere "registrati" presso una platform ed una volta che quest'operazione è completa tutti gli agenti presenti possono comunicare tra di loro in maniera trasparente.

I due componenti principali nella piattaforma sono AMS e DF:

L'*Agent Management System* provvede al servizio di naming (ovvero si assicura che ogni agente nella platform abbia un nome univoco) ed allo stesso tempo svolge il ruolo di amministratore della piattaforma (può creare e distruggere agenti, tra le altre cose).

Il *Directory Facilitator* invece gestisce il servizio delle "pagine gialle" per tutti gli agenti, ovvero fornisce l'identità di coloro che possono risolvere un determinato servizio specifico (precedentemente registrato).

Coesione concetti modello / platform:

Sulla base dei requisiti sopra descritti, assume un ruolocentrale la costruzione di una topologia su cui possa essere basata la simulazione, tale da rispettare i vincoli del modello ed implementare le logiche architetturali di JADE. La prima criticità affrontata è stata quella della rappresentazione dell'astrazione urbanistica: l'architettura di JADE non è gerarchica ed è costituita solamente da un livello (container e platform). La visione urbana del nostro modello invece può imporre diverse stratificazioni (quartiere, città, regione, nazione, continente e mondo), ognuna di queste deve poter essere analizzata. E' stato scelto, dunque, di tenere come unità minima il quartiere (corrispondente ai *container*), una città infatti rappresenterà solo una "organizzazione virtuale" di più container. Seguendo questo prototipo la tassonomia dei container sarà del tipo "ROMA1, ROMA2.. ROMAn", dove tutti i quartieri con suffisso "roma" rappresentano i quartieri della città. Per rappresentare i successivi livelli di dettaglio delle informazioni (regionali, nazionali o mondiali) verranno effettuate delle aggregazioni topologiche con codici identificativi che serviranno ad identificare dati parziali del sistema ad ogni iterazione.

Al centro del modello c'è la persona, che, come anticipato, deve essere in grado di spostarsi da un quartiere all'altro indipendentemente. Dato che la gerarchia di JADE è

"flat" ci occorre un supervisore che si accerti che questi spostamenti vengano effettuati congruamente al modello, questo ruolo verrà ricoperto dall'agente sindaco. Sarà presente un sindaco per ogni città,il quale oltre a ricevere dati statistici per l'utilizzo successivo sarà anche il garante degli spostamenti inter-quartiere o inter-città effettuati dalle persone.

Tra le varie città vengono istaurati collegamenti fissi: le autostrade, che presentano proprietà particolari. Sarà, dunque, collocato per ogni autostrada un agente atto a controllarne la capienza e a gestire il ritardo generato dal trasporto di un individuo da una città all'altra.

Insieme a questi "protagonisti" del modello ci dovremo avvalere anche di alcuni agenti di supporto per la raccolta ed elaborazione dei dati numerici e per la visualizzazione di tutto il sistema. Un agente statistico si occuperà di raccogliere, elaborare e reinviare i dati mentre un agente dedicato alla gui riceverà questi aggiornamenti così che possano essere visualizzati a schermo.

Topology

Lo studio effettuato sulla topologia è stato pensato per adattarsi alla JADE platform e quindi per aumentare la coesione tra il modello e l'architettura. Innanzitutto era necessario trovare un modo per mappare i quartieri all'interno delle città. Si è deciso che ogni città avrà 9 quartieri in tutto (e quindi 9 *container* a rappresentarli) mentre il numero delle città potrà essere aumentato arbitrariamente. All'interno dello scenario saranno, inoltre, richieste le quantità di persone che andranno create ed i relativi malati iniziali, questi potranno essere distribuiti differentemente in 3 maniere diverse: ***CITY*** *(tutti i malati nella stessa città),* ***DISTRICT*** (tutti i malati in un unico quartiere) *ed* ***UNCLUSTERED*** (malati distribuiti). Vi sono poi tre istanze di topologie che creano le città dislocandole differentemente a livello geografico: *RING* (le città vengono interconesse ad anello, una autostrada connette la città successiva a quella precedente), ***MESH*** (ogni città è collegata a tutte le altre attraverso una maglia completa di collegamenti autostradali) e ***CLUSTER*** (le città vengono create a blocchi "insulari" di 3, solo nelle città di uscita vengono create autostrade verso le altre isole). Le autostrade sono canali di comunicazione che permettono il passaggio da un quartiere specifico di una città (chiamato porta) ad un quartiere di un'altra città in maniera asincrona e bidirezionale.

La disposizione dei quartieri all'interno delle città verrà gestita da un'apposita mappa (Tabella 1) che serve ad assegnare un indice numerico atto ad identificare la loro posizione spaziale all'interno della città.

Tabella 1

1	2	3
4	5	6
7	8	9

La vicinanza di alcuni quartieri rispetto ad altri ne determinerà la possibilità o meno da parte di un individuo di spostarvisi. Secondo questo esempio il quartiere "1" sarà vicino solo ai quartieri 2 e 4. Dunque i suoi abitanti possono muoversi solo verso uno di questi vicini. La mappatura della vicinanza di ogni quartiere verrà mantenuta dai rispettivi Mayor di città (all'interno di un container dedicato).

Progettazione

La progettazione di tutto il simulatore ha richiesto un'attenta analisi per poter capire quali esigenze del modello erano più vincolanti e quali potevano essere rilassate. Molti vincoli, infatti, devono essere trasposti non solo nell'implementazione dei singoli comportamenti degli agenti o della logica del simulatore ma anche nell'operazione di inizializzazione di tutto lo scenario creato.

Il lavoro di progettazione ha necessitato in primis una importante fase di comprensione delle strutture e caratteristiche del framework JADE. Inoltre, dato che le potenzialità e il know-how di come effettivamente implementare il modello non erano chiari dall'inizio, sono state effettuate fasi successive di ridefinizione del target finale in funzione di quanto scoperto durante l'approfondimento.

Descriveremo quì di seguito i principali problemi e vincoli affrontati durante la fase iniziale :

I. In un primo momento, è stato ipotizzato di utilizzare più platforms ed il plugin di mobility di JADE per effettuare gli scambi intra-platform (questa visione era utile in quanto riusciva a dare 2 livelli di gerarchia iniziale invece di uno solo). Questa ipotesi è stata scartata sia in virtù dei problemi di comunicazione tra le due platform (ricevono la richiesta di spostamento di un agente ma non riescono ad inviarne i dati), sia perchè la sola presenza dell'agente responsabile della gestione del plugin aumenta il carico sulla CPU del 20-25% in idle.

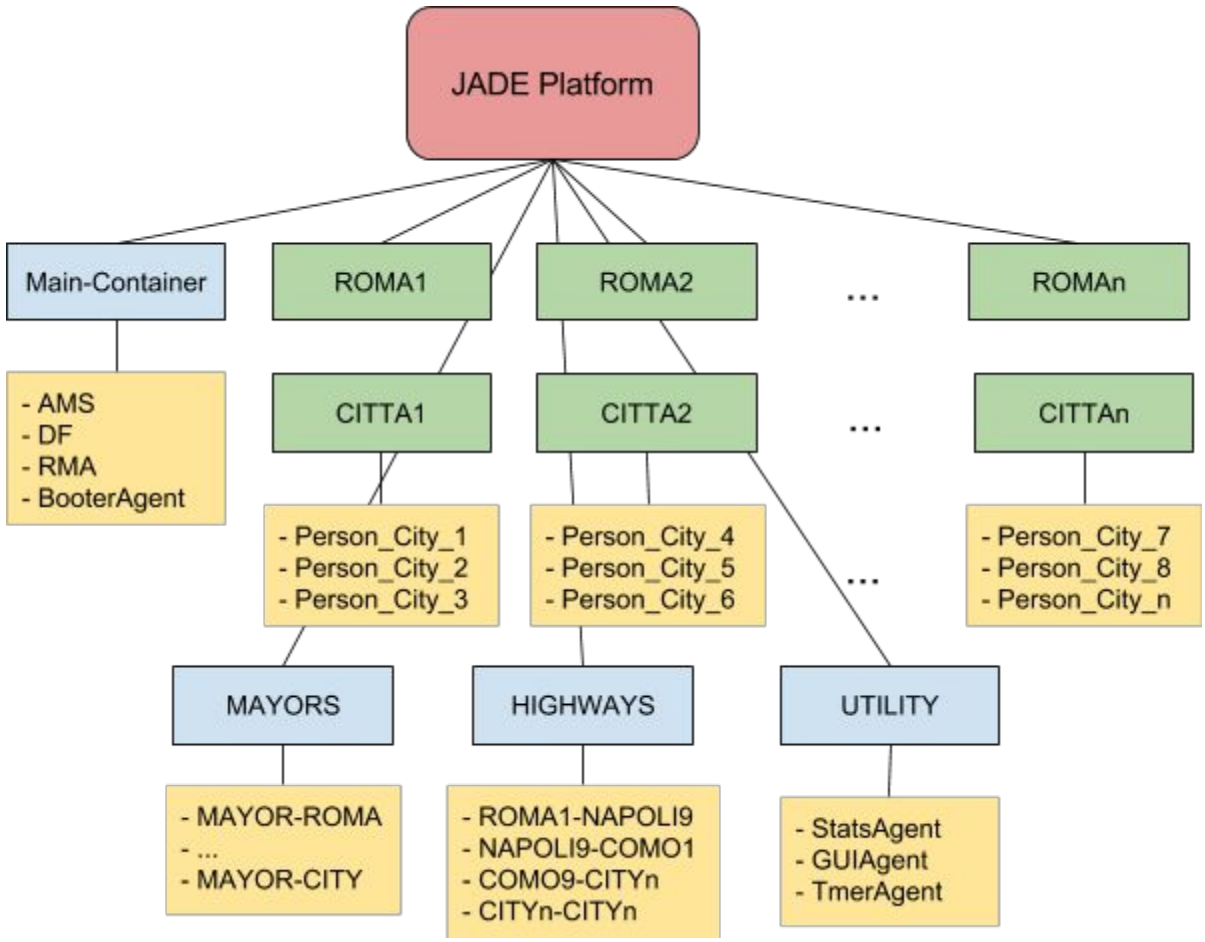
II. La capacità di implementare dei behaviours (o comportamenti) all'interno degli agenti stessi fornisce quello che per definizione è il comportamento reattivo di tali oggetti. Mediante questi costrutti possiamo modellare l'indipendenza della persona e la sua capacià di spostarsi all'interno del simulatore.

III. E' stata testata la capacità di avviare container remoti da altre macchine con la registrazione presso una platform centrale, residente su un diverso host, con esito positivo; la possibilità di scalare il simulatore quindi può essere regolata proprio su questo parametro.

IV. Lo scambio di messaggi risulta essere il nodo centrale di tutto il simulatore. Infatti, dopo l'implementazione, occorrerà analizzare l'overhead dato dall'invio di tutti i vari messaggi per stabilire se il sistema è in grado di gestirli. A differenza di altre situazioni nel nostro modello la persona è una fonte di molti messaggi in un dato istante.

Prima Analisi

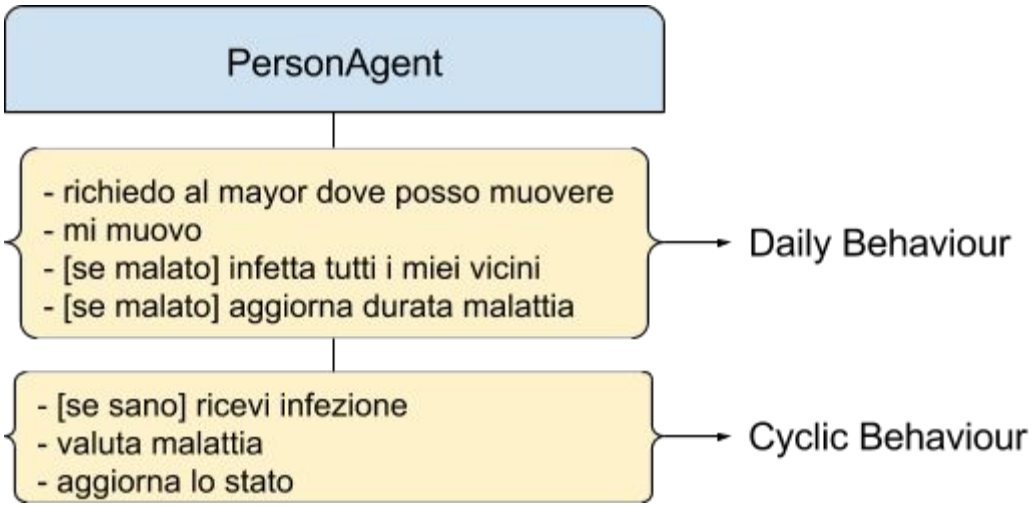
Di seguito il risultato visivo della prima fase di analisi. Questa immagine mostra l'organizzazione della topologia e la disposizione degli agenti all'interno dei container, la loro tassonomia e la struttura del simulatore.



Il simulatore si presenterà con una finestra di input dati dove sarà possibile impostare i parametri iniziali relativi alla nuova istanza (si definisce lo *scenario*). Terminato il setup il *BooterAgent* si occuperà di inizializzare tutto il sistema fino allo stato iniziale.

Progettazione dei behaviours

La seconda fase affrontata è stata quella della stesura dei vari behaviours dei singoli agenti; in questa fase l'interazione tra agenti risulta fondamentale per determinare la sequenzialità delle operazioni coinvolte. Anche questa fase è stata soggetta ad almeno 3 reiterazioni del progetto con successivi raffinamenti ed aggiunte, collegate all'insorgere di nuove esigenze vincolate al modello utilizzato. I behaviour quì esposti sono raggruppati per funzionalità, ma, in realtà, nel codice sono presenti più sotto-behaviours, che insieme costituiscono il comportamento finale dell'agente in questione.

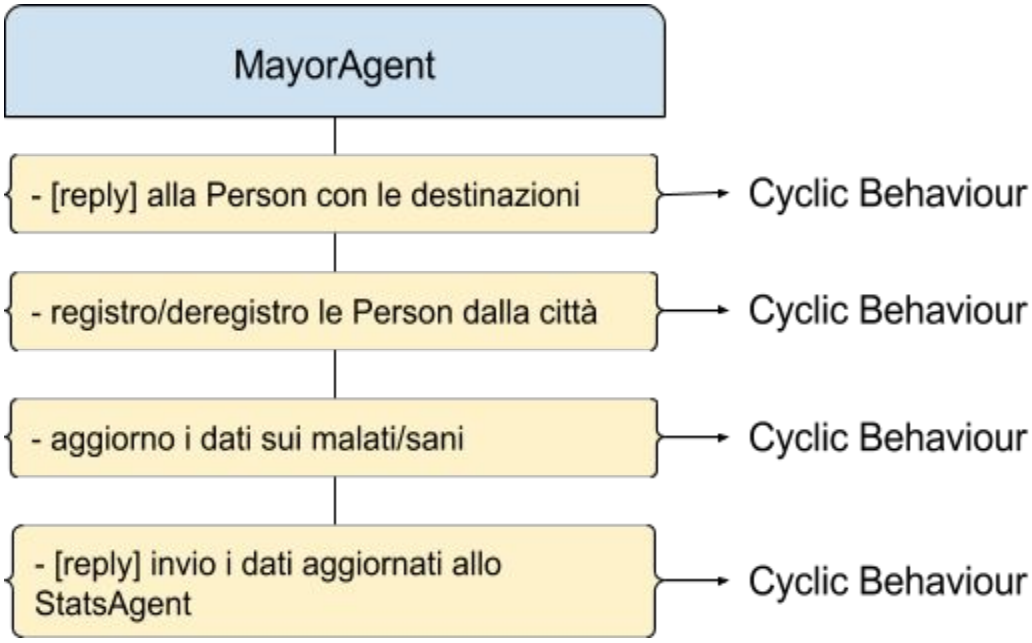


La persona è definita in funzione di un behaviour giornaliero atomico, che le permette di muoversi e successivamente ricevere o inviare l'infezione a seconda del suo stato di contagio. Una volta contagiato l'individuo ha un tempo di incubazione all'interno del quale non può essere infettato nuovamente né infettare a sua volta. Ogni daily "round" questo contatore viene incrementato fino all'insorgere della malattia ed al conseguente cambio di stato in malato. Al termine del decorso della malattia la persona si curerà ed otterrà un certo grado di immunizzazione alla malattia (settato per ora a totale). Ad ogni

cambio di stato la person invierà un aggiornamento al proprio Mayor di città in modo che possa aggiornare i dati statistici.

La seconda tipologia di behaviours dell'agente persona concerne la capacità di ricevere in maniera asincrona tutti quei messaggi che riguardano lo stato dell'individuo. In ogni momento, infatti, la persona potrà ricevere un messaggio di contagio (proveniente da un agente infetto all'interno dello stesso container) che determinerà la probabilità di ammalarsi a sua volta.

Lo start/stop, invece, è il messaggio dato dalla gui per fermare o far ripartire tutta la simulazione.



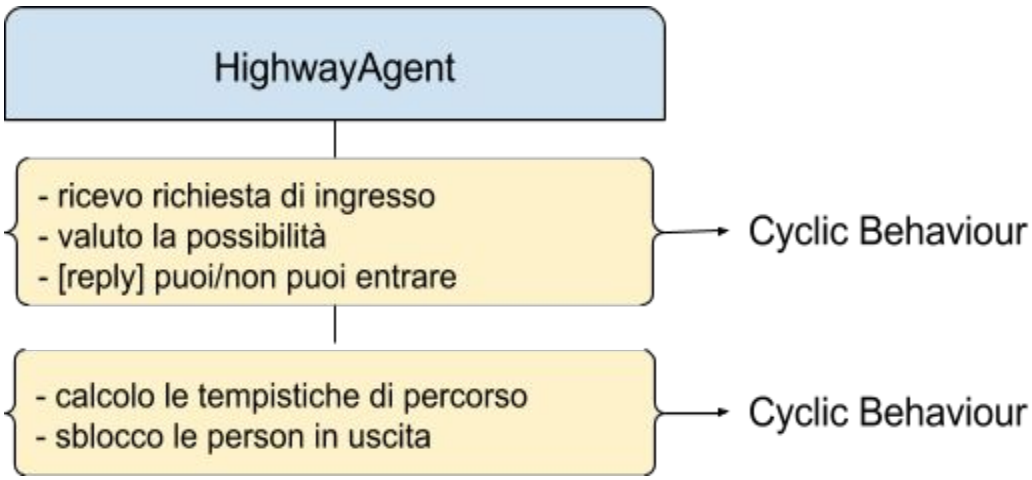
Il sindaco, o *MayorAgent*, è l'entità passiva preposta al controllo della viabilità urbana di una città. Questo agente si è reso necessario per poter mappare correttamente ed in maniera scalabile l'esigenza del modello relativa ai quartieri delle città. La posizione geografica dei quartieri, infatti, incide sulla capacità di spostamento delle persone: come nella vita reale è possibile spostarsi da un quartiere all'altro solo se sono confinanti. In fase di inizializzazione il *BooterAgent* crea una *HashMap* per ogni città che determina i suoi collegamenti urbani statici e la assegna al Mayor preposto. Tale hashmap viene poi aggiornata per includere anche i collegamenti inter-urbani (ovvero con le autostrade), fornendo di fatto la possibilità di spostarsi da una città all'altra. Il *MayorAgent* è, dunque, un agente passivo, che risponde ai bisogni di altri agenti e per ognuno di essi implementa un behaviour specifico (tutti quanti di tipo *Cyclic*, ovvero sempre disponibili).

Il primo behaviour serve a consegnare ad ogni persona che ne fa richiesta la lista delle destinazioni raggiungibili in base alla propria posizione attuale. Il Mayor estrae dalla HashMap le destinazioni possibili e le invia come risposta alla persona, che sceglierà dunque dove muoversi.

Il secondo behaviour resta in ascolto di notifiche da parte di tutte le persone che si muovono sia tra i quartieri sia tra le città. In entrambi i casi il Mayor aggiorna i dati numerici che fanno riferimento alle statistiche, le persone vengono quindi censite ad ogni spostamento (in caso di uscita dalla città verranno censiti in uscita dal Mayor di partenza e registrati da quello di arrivo).

Il terzo behaviour aggiorna le statistiche dei quartieri (e di riflesso della città) per quanto riguarda la quantità di abitanti totali sani, incubati e malati. Questa lista di statistiche viene tenuta sempre in costante aggiornamento perchè rappresenta l'unico modo per avere una "snapshot" del simulatore in un dato istante.

L'ultimo behaviour è quello preposto all'invio del pacchetto dati allo *StatsAgent*. Alla fine di ogni round questo agente richiederà a tutti i Mayors i nuovi dati che verranno poi mostrati nella GUI. I Mayors, quindi, provvedono a organizzare ed inviare tutti i loro dati a tale agente che provvederà alla loro elaborazione.

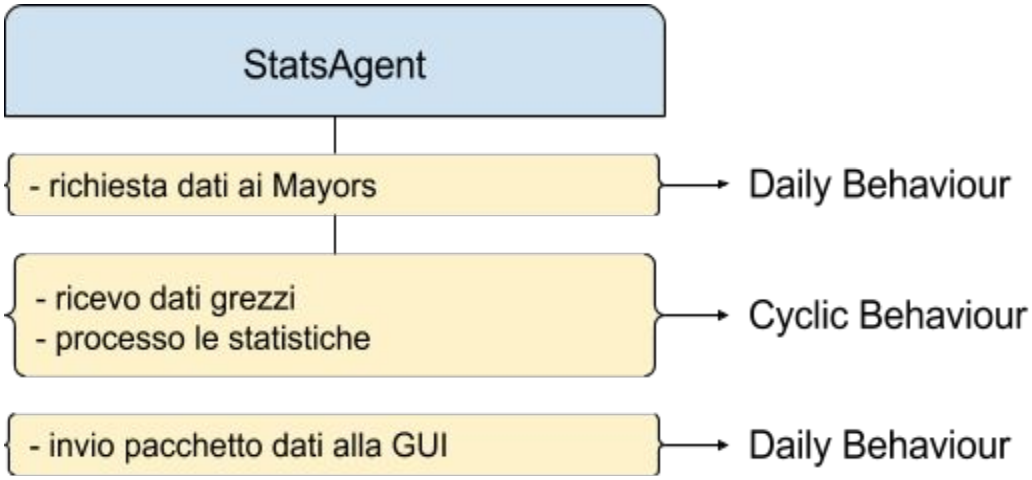


L'autostradiere, o *HighwayAgent*, è l'agente che mantiene il controllo di una autostrada e ne supervisiona l'andamento. E' presente un *HighwayAgent* per ogni autostrada (intesa come arco che connette due quartieri di città differenti) e la comunicazione può avvenire in maniera bidirezionale. Ogni canale, definito di andata e di ritorno, ha una capacità e un tempo di percorrenza (quando è raggiunto il limite di capacità nessuna persona può procedere per quella autostrada fino a che non si liberano dei posti). I tempi di percorrenza sono in funzione dei round giornalieri; mentre gli spostamenti tra quartieri

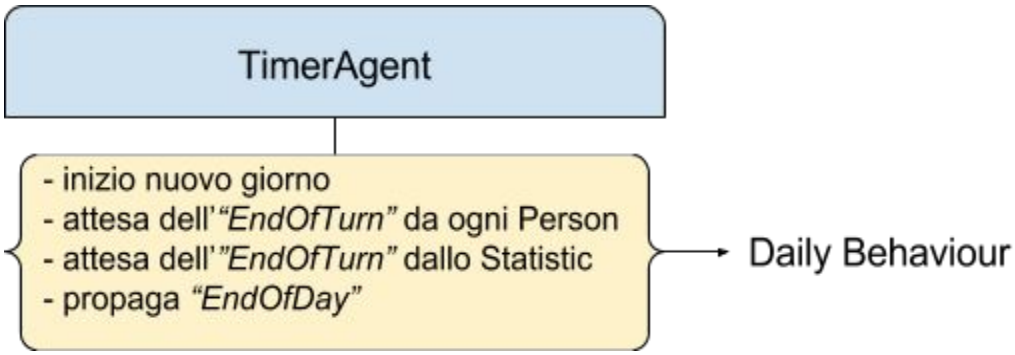
impiegano un tempo 0 (istantanei) per essere portati a termine, quelli autostradali necessitano almeno di 1 round (o più) per il transito. Ogni persona in transito mantiene il suo stato attuale (modificando solo i tempi di incubazione o decorso laddove infetta, in funzione dei round che passano) e riprende l'esecuzione del movimento una volta che arriva nel nuovo quartiere della nuova città.

Il primo behaviour implementato dall'autostradiere gestisce la richiesta da parte di una persone di entrare in autostrada. Dopo aver fatto un controllo sulla capienza *l'HighwayAgent* invia la risposta (positiva o negativa) alla persona che quindi opterà per entrare nell'arco, oppure in caso negativo sceglierà un'altra destinazione urbana. In caso inizi il viaggio verrà aggiornata la nuova capienza nel corrispettivo senso di marcia.

Il secondo behaviour si occupa di "sbloccare" quelle *PersonAgent* in transito; giunte al termine della durata del viaggio vengono aggiornati i relativi contantori e la persona riprende la sua esecuzione all'interno del nuovo container di arrivo.



L'agente statistico raccoglie i dati intermedi dai Mayors e dalle Highways, li processa e genera dei pacchetti di report che vengono inviati giornalmente alla GUI per poter essere visualizzati in maniera aggiornata. Dato che lo statistico è l'ultimo agente che entra in campo durante il ciclo di vita giornaliero del simulatore è garantito che i suoi dati siano aggiornati all'ultimo istante valido di simulazione prima del nuovo round. La richiesta dei dati ai Mayor viene effettuata in coda a tutte le attività delle persone.



La nozione di tempo *simulato* è fondamentale all'interno di un progetto di simulazione. A tale scopo si è resa necessaria la modellazione di un agente Timer che introducesse un meccanismo di sincronia tra gli agenti persona. La sincronia è dovuta al fatto che ogni persona compie una serie di azioni che devono essere giornalmente atomiche ma la cui valutazione richiede di misurare il tempo trascorso in maniera univoca (ad esempio, il decorso di una malattia in giorni). Il timer, una volta inizializzato, si occupa di scandire l'inizio di ogni nuovo giorno simulato; le persone quindi svolgeranno le loro attività fino all'invio del loro messaggio di *End Of Turn*. Il timer raccoglie una quantità precisa di questi messaggi (uguale al numero di persone presenti) dopodichè attende la fine del turno anche da parte dello statistico (che deve raccogliere i dati numerici).

Risultati

Si è giunti ad una versione finale del simulatore dopo numerosi test e correzioni che ci hanno permesso di osservare il modello funzionante costruito sopra l'architettura JADE. La soluzione finale implementa, dunque, una singola platform all'interno della quale prendono posto svariati container (main, di supporto ed urbani) con una quantità crescente di agenti. La difficoltà principale è stata quella legata al carico di lavoro dell'agente **DF** (*Directory Facilitator*) che provvede alla gestione delle pagine gialle e di tutti i servizi registrati nella platform. La quantità di informazioni che veniva richiesta al DF era oltremisura e questo *polling* determinava non solo scarse prestazioni ma anche ritardi nella consegna di tutti i messaggi che passavano per tale agente. Tramite delle ottimizzazioni effettuate a fronte di questo problema siamo giunti ad una versione funzionante che riesce a far iterare l'esperienza di simulazione correttamente seguendo le parametrizzazioni date al sistema.

Sono state testate alcune istanze di esempio con un numero di città variabili ed un numero di agenti sempre più crescente (fino a 100) e, sebbene il simulatore tenda a rallentare per istanze molto popolate, riesce comunque a svolgere il suo compito celermente fino al raggiungimento di una situazione stabile, determinata dalla guarigione di tutte le persone e dalla loro immunizzazione. Unico limite rimasto all'interno della soluzione è rappresentato da una situazione di stallo del programma che può verificarsi ad intervalli differenti rispetto ad ogni istanza di simulazione e che determina il blocco di tutto il simulatore. Sono stati effettuati molti tentativi per capire la natura del problema ma ogni prova ha dato come esito una de-sincronizzazione tra alcuni agenti chiave ed i loro messaggi di risposta al timer. Il problema, però, si manifesta allo stesso modo ma in maniera differente al cambio della macchina di esecuzione. La parallelizzazione degli agenti e il modo in cui vengono schedulati i loro behaviour incide fortemente su tale problematica che risulta in una situazione di concorrenza tra agenti e messaggi. Sono state messe in campo alcune misure atte a prevenire questa complicazione (sono stati serializzati i behaviours responsabili dell'invio di conferme al Timer) ma questo non è bastato a risolvere l'errore.

Soluzioni e sviluppi

Verrà descritta quì di seguito la migliore soluzione candidata alla risolvere il problema principale del simulatore. Questa implementazione garantisce un secondo step di studio più approfondito e prevede una serie di modifiche al codice che dovrebbe essere in grado di limitare l'influenza negativa sia del DF attualmente utilizzato, sia delle scarse prestazioni che subisce il programma.

Questi cenni di soluzione sono stati anche il frutto del nostro scambio con Caire che ci ha aiutato nel capire in quale direzione potessimo andare per terminare il nostro attuale lavoro.

1. Sostituire l'attuale *SequentialBehaviour* principale del *PersonAgent* (che alterna i diversi sub-behaviours delle sue azioni) con un FSMBehaviour. Questo particolare behaviour esegue tutti i suoi sotto-behaviour secondo una *Finite State Machine* che può essere implementata sulle basi dei cambiamenti di stato della persona (sana, malata ed infetta). Questa implementazione (estendibile anche allo *StatisticAgent* per quanto riguarda il suo invio dati) potrebbe essere la soluzione al problema di mancata sincronia che intercorre dopo una certa soglia perchè dovrebbe garantire l'atomicità degli step all'interno degli stati.
2. Decentralizzare il lavoro del DF su più agenti speciali che possano prendere in carico la parte dei servizi che è relativa all'infezione dei presenti all'interno di un quartiere (ispirato come modello a quello che già fa il Mayor). Si può prendere in considerazione l'idea di decentralizzare a livello di città, oppure anche di singolo quartiere. Da valutare è anche l'ipotesi di poter scrivere due (o più) DF

personalizzati tramite l'estensione della classe *DFAgent* invece che della semplice *Agent*.

3. Rivedere nei punti critici di invio dei segnali per la sincronizzazione (dotati dei conversationId: "EOT", "EOS" ed "EOD") l'implementazione della receive() -- block() alla luce del fatto che questa chiamata diventa "bloccante" solo quando la action() del metodo ritorna il controllo.
4. Per i servizi rimasti a gestione del DF (non è necessario devolvere tutto a qualcun'altro, il DF ha comunque una soglia di funzionamento alta) controllare l'implementazione dei nuovi servizi: il cambio dei services repentino infatti è di difficile gestione e a volte presenta delle instabilità durante l'esecuzione del programma.

Conclusioni

A fronte di quello che è stato analizzato, visto e realizzato nel corso del progetto abbiamo capito come JADE sia un framework che implementa in maniera molto utile gli agenti e ne declina il loro paradigma in maniera precisa e puntuale. Il fatto che questo framework possa garantire scalabilità nell'utilizzo e una distribuzione a livello di rete lo rende uno strumento molto capace, soprattutto grazie all'utilizzo dello standard **FIPA** per il *message passing* e di ontologie specifiche. Per quanto riguarda, invece, il modello preso in esame (lo sviluppo e l'analisi della diffusione di una malattia all'interno di una popolazione localizzata a livello urbanistico) ci sono diversi punti d'incontro che possono essere sfruttati tra JADE e il problema considerato.

Tuttavia le criticità maggiori emerse sono quì riassunte:

1. E' stato confermato che il *Directory Facilitator* a lungo andare sarà il problema principale perchè anche a fronte dell'introduzione della temporizzazione (e quindi della sincronia tra agenti) le prestazioni non sono aumentate della quantità sperata. Si è evidenziato che il DF da solo, con l'implementazione dei propri servizi, non è sufficiente a garantire un buon risultato perchè rimane un nodo sovraccarico di questo modello, data la frequenza delle richieste che gli vengono inoltrate (in alcune istanze i messaggi superano la soglia di *over threshold*, perchè il DF è oberato di lavoro).

2. Il tempo rappresenta un concetto fondamentale e con esso la sincronia. Senza considerare tali aspetti, inizialmente era stata presa in esame un'implementazione dove ogni agente teneva privatamente traccia della durata dei suoi stati (malattia, infezione etc.) ma questo approccio si è rivelato subito fallimentare oltre che

costoso. La sincronia rappresenta un vincolo che non può essere rilassato nella simulazione.

3. Una delle caratteristiche principali del modello di JADE è il fatto che sia "logicamente appiattito", ovvero non esistono modi per modellare gerarchie tra agenti. Il nostro modello, invece, richiede una topologia che per definizione è gerarchica (quartieri, città, regioni, etc.). Per risolvere questo problema sono stati messi in campo degli agenti speciali (i Mayor e le Highway) che servono ad implementare aspetti tecnici che non sarebbero potuti essere modellati altrimenti. Tramite questa "decentralizzazione" è stato possibile realizzare la disposizione geografica di quartieri e città (la mappa dei quartieri è mantenuta dai Mayors che provvedono a passarla alle persone su richiesta così da poter determinare *dove* possono muoversi). Tramite le Highways, invece, sono stati modellati dei canali di comunicazione che intercorrono tra due città. Tali canali sono soggetti a portata massima in funzione di un tempo necessario al transito, ovvero un tempo in cui gli agenti coinvolti sono in viaggio e quindi non reperibili. Il fatto che il movimento sia supervisionato ne garantisce la correttezza logica e, inoltre, ci è servito per tenere traccia delle statistiche numeriche (raccolte in prima istanza proprio da questi due agenti) oltre che valutare il comportamento delle persone tra le città ed i quartieri.

Queste considerazioni insieme ad altre difficoltà tecniche (legate alla mera implementazione e gestione non fluida dei metodi che dovrebbero essere bloccanti all'interno dell'agente) hanno portato alla conclusione che questo framework non si adatti bene all'implementazione del modello proposto. Il lavoro di studio effettuato, comunque, ha messo in luce pregi e difetti sia della scelta del modello, che di quella del framework, e, ancora più importante, ha contribuito alla maturazione della capacità di reagire di fronte alle difficoltà. L'aspetto più rilevante riguarda le diverse discrepanze che possono sorgere a fronte di una modellazione corretta dal punto di vista logico ma non coerente da quello fisico ed architetturale di un framework. Alcuni concetti come la temporizzazione e la gestione dei behaviours hanno richiesto un massiccio restauro dopo che il loro funzionamento non era stato del tutto spiegato dalle guide rese disponibili. In un'ottica in cui un framework di programmazione dovesse diventare uno standard il supporto della comunità è fondamentale: i dubbi più difficili che abbiamo avuto hanno incontrato la pronta risposta del creatore di JADE, Caire Giovanni che ci ha illuminato su alcune incomprensioni. La scarsa diffusione e la relativa mancanza di supporto e di esempi approfonditi su JADE, però, si fanno sentire non appena occorre implementare agenti più complessi dal punto di vista operazionale, rendendo questo framework di difficile impiego per problemi guidati da molti vincoli logici come quelli imposti, ad esempio, da un modello come il nostro.