ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Ingegneria e Architettura
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

AGENTI AUTONOMI E ARTEFATTI PER LA SIMULAZIONE DEL TRAFFICO IN UNA SMART CITY

Relazione finale in
SISTEMI AUTONOMI

Relatore
Prof. ANDREA OMICINI

Presentata da
BRANDO MORDENTI

Anno Accademico 2016 – 2017

PAROLE CHIAVE

smart mobility

sistemi di trasporto intelligenti

simulazione

programmazione ad agenti

agenti e artefatti

La città è qualcosa di più di una congerie di singoli uomini e di servizi sociali, come strade, edifici, lampioni, linee tranviarie e via dicendo; essa è anche qualcosa di più di una semplice costellazione di istituzioni e di strumenti amministrativi, come tribunali, ospedali, scuole, polizia e funzionari di vario tipo. La città è piuttosto uno stato d'animo, un corpo di costumi e di tradizioni, di atteggiamenti e di sentimenti organizzati entro questi costumi e trasmessi mediante questa tradizione

Indice

Introduzione

Il dominio geografico oggetto di questo lavoro è costituito dall'area urbana, ovvero una zona caratterizzata da un'elevata densità di popolazione e da una massiccia presenza di attività commerciali, residenziali, ricreative e culturali. In questo contesto, i veicoli dei privati si spostano utilizzando la stessa infrastruttura stradale che usano i mezzi di trasporto pubblico, i veicoli di trasporto merci, veicoli dedicati alle forze dell'ordine, veicoli di soccorso, biciclette e pedoni. Risulta quindi intuitivo immaginare quale mole di traffico possa generarsi in determinate circostanze e – più in generale – all'aumentare dell'ampiezza e alla concentrazione di abitanti nelle aree urbane, con inevitabili conseguenze: crescita del numero di congestioni, disagi al regolare flusso del traffico e terribili influenze sull'ambiente.

Le ripercussioni negative sono note a tutti, tanto che alcune città – per le quali è emerso il termine *smart city* - hanno intrapreso iniziative con le potenzialità di portare benefici a livello sociale, ambientale ed economico. La *smart mobility* è l'insieme di tecnologie e progetti in grado di migliorare sotto diversi punti di vista la gestione del traffico urbano e grazie al diffondersi di tecnologie **ITS** (*Intelligent Transportation System*) e **IoT** (*Internet of Things*) che hanno avuto un grande sviluppo (specialmente negli ultimi anni), divenendo di fatto una delle tematiche più futuribili relativamente al campo delle *smart cities*.

Le aziende e le amministrazioni che intendono mettere in atto progetti o soluzioni di *smart mobility* hanno sicuramente l'esigenza di stimare a priori gli eventuali costi e benefici previsti per le loro applicazioni e sarebbe certamente utile poter farlo in relazione all'area geografica d'interesse. La mancanza di uno standard di valutazione e di approcci strutturati alla materia costituisce

un'ulteriore criticità che chi intende dare forma a un progetto simile deve considerare.

Lo scopo di questo studio è in un primo luogo quello di identificare e classificare le tecnologie e le pratiche connesse alla *smart mobility*, dopodichè verrà descritta l'analisi e la progettazione di un simulatore del traffico urbano in una determinata zona sfruttando le potenzialità della programmazione ad agenti.

Capitolo 1

Stato dell'arte

1.1 Smart City

Una città può essere definita *smart* quando gli investimenti effettuati in infrastrutture di comunicazione tradizionali (trasporti) e moderne (tecnologie dell'informazione e comunicazione) – riferite al capitale umano e sociale – assicurano uno sviluppo economico sostenibile e un'alta qualità della vita, una gestione sapiente delle risorse naturali, attraverso l'impegno e l'azione partecipativa. Il concetto di smart city si basa sull'utilizzo di infrastrutture di rete per migliorare l'efficienza economica e politica e consentire lo sviluppo sociale [1], culturale e urbano, in cui il termine infrastrutture indica i servizi resi disponibili alle imprese e ai residenti per il tempo libero, la qualità della vita e le tecnologie dell'informazione e la comunicazione (telefonia, TV satellitare, reti informatiche, e-commerce, internet). Viene portata in primo piano l'idea di una città cablata come il modello di sviluppo principale della connettività [2]. Le smart cities possono essere identificate e classificate in base a sei dimensioni principali: *smart economy, smart mobility, smart environment, smart people, smart living, smart governance.* [3]

1.2 Smart Mobility

La **smart mobility** rappresenta un punto cruciale per la realizzazione di una smart city e ne rappresenta un importante criterio di valutazione. La risorsa di cui si occupa la smart mobility è la rete stradale e lo scopo principale è quello di ottimizzarne la gestione e l'uso da parte dei cittadini, sia in termini di numero di veicoli che la attraversano sia in termini di tempi di percorrenza dei veicoli stessi. Gli obiettivi possono essere riassunti nelle seguenti categorie [4]:

- Riduzione dell'inquinamento
- Riduzione delle congestioni del traffico
- Aumento della sicurezza dei cittadini
- Riduzione dell'inquinamento acustico
- Aumento della velocità di trasferimento
- Riduzione dei costi di trasferimento

Le amministrazioni pubbliche sono i principali responsabili della promozione e dell'organizzazione della mobilità sostenibile, i cui interventi sono finalizzati alla promozione di metodologie di mobilità alternativa (a piedi, in bicicletta, con mezzi di trasporto pubblico, con mezzi di trasporto condivisi, ad esempio *car pooling* e *car sharing*).

Nel report per il *Sustainable Mobility Project 2.0* all'interno del *World Business Council for Sustainable Development* [5] gli autori descrivono 22 indicatori per i parametri e le metodologie usate dalle città per misurare le performance di *sustainable mobility*. La smart mobility si interseca con numerosi di essi, tra cui congestioni e ritardi, tempi di percorrenza, utilizzo di spazio di mobilità, accesso ai servizi di mobilità, sicurezza del traffico, comfort, connettività intermodale e rate di occupazione. Inoltre, essa si connette a un ampio range di tecnologie come la manifattura dei veicoli, sistemi di trasporto, logistica e tecnologie di comunicazione.

Alcune statistiche riportano che le città europee hanno un miglior transito pubblico e un miglior *focus* sulle soluzioni *low-carbon* rispetto alle altre città in giro per il mondo. Nel 2014 le città che più hanno investito sulle innovazioni a livello di infrastruttura sono Copenhaghen, Vienna, Londra, Barcellona, Stoccolma, Amburgo, Berlino, Parigi, Amsterdam e Helsinki. [6]

1.2.1 Mobility Patterns

Il monitoraggio dei *mobility patterns* può essere usato per studiare il comportamento dei guidatori per migliorare il flusso del traffico ed accrescere la sicurezza stradale. La figura 1.1 indica uno schema per l'acquisizione, archiviazione, processing e analisi di dati correlati alla mobilità utilizzando i sensori degli *smartphones*. Questo approccio permette agli utenti di beneficiare di servizi addizionali, come consigli su percorsi alternativi e feedback sull'andamento del traffico per tempi di percorrenza più corti e risparmio di carburante. [7]

Figura 1.1: Schema per l'acquisizione, storage, processing e analisi di dati di mobility usando i sensori degli smartphone

La tecnologia degli smartphone può anche essere applicata per dare una buona approssimazione del tasso di occupazione dei veicoli come parametro di smart mobility.

Gli *urban mobile data* permettono di sviluppare concetti in cui la sostituzione dell'uso privato dei veicoli con l'uso di mezzi pubblici (e la riduzione del

traffico) crea un flusso del traffico più efficiente, con la conseguente riduzione di emissioni di carbonio. L'obiettivo comune è quello di raggiungere un'ottimizzazione bilanciata dell'uso dei veicoli. Oltre a notevoli miglioramenti nella pianificazione del percorso in real-time, è importante che le soluzioni di smart mobility riguardanti questo aspetto diffondano informazioni accurate e attuali su una serie di parametri come lo stato del meteo, eventuali lavori di manutenzione, incidenti, eventi pubblici che possono incidere sulla scelta dei privati di utilizzare i mezzi pubblici o il proprio veicolo. [8]

L'uso di *mobile devices* nel contesto stradale da parte di automobilisti e altri utenti della strada è in costante crescita. Nel caso dei veicoli, l'industria dell'automobile ha concentrato gli sforzi su soluzioni *built-in* che forniscano gli avvertimenti necessari e la geolocalizzazione ma al contempo riducano le potenziali distrazioni del conducente. [9]

1.2.2 Vulnerable Road Users

Troppo spesso quando si considerano soluzioni di smart mobility non viene data l'adeguata rilevanza a pedoni e altri VRU (Vulnerable Road-Users). Anche in questo caso l'uso dello smartphone assume particolare importanza: da un lato può assistere i pedoni fornendogli informazioni, dall'altro può rappresentare un rischio per la loro sicurezza. Sono emerse alcune applicazioni mobile per minimizzare i potenziali pericoli per i VRU con pattern di comunicazione V2P e P2V (*Vehicle to Pedestrian, Pedestrian to Vehicle*) per lo scambio di informazioni finalizzato al miglioramento dell'uso della strada e della sicurezza.

Quello che è emerso da alcune ricerche è che la percezione e la comunicazione sono i due aspetti fondamentali per la sicurezza dei VRU; un sistema cooperativo con l'integrazione di entrambi gli elementi è stato proposto nel documento [10].

1.3 Aspetti di smart mobility

Nell'ambito dello studio e della ricerca sulla smart mobility risulta cruciale investigare sul ruolo dei cittadini – intesi come futuri utenti – per scoprire quali siano le loro effettive necessità e garantire un ambiente *citizen-friendly*. La conoscenza risultante dall'analisi di enormi quantità di dati, rilevati tramite la tecnologia e il trend positivo del *pervasive computing* - associato alla diffusione ormai totale degli smart devices - non fa che incentivare alla ricerca in un ambito che ambisce alla creazione di ampi benefici sociali.

1.3.1 Social

Il feedback dei cittadini e i consigli per il miglioramento dei servizi sono un requisito fondamentale per una città sostenibile ed efficiente. Le moderne tecnologie pervasive abilitano gli utenti alla condivisione di informazioni con le pubbliche autorità, rendendo possibile una *data-analysis* su dati provenienti da tecnologie di *crowd-sourcing*.

Le metodologie che catturano dati correlati alle preferenze dei cittadini e alle loro abitudini aiutano a identificarne e comprenderne bisogni e obiettivi. Ad esempio, tramite misure su variabili come **origine e destinazione** di un viaggio o il percorso consigliato. Alcune ricerche nel contesto cooperativo della smart mobility investigano sull'effetto del numero di veicoli e della capacità stradale sul livello di **congestione** del traffico, focalizzandosi sulla modifica delle scelte di percorso. Gli autori di tali ricerche hanno studiato la relazione tra la domanda di traffico e i tempi di percorrenza alla guida, stimando i benefici di diversi scenari. La conclusione a cui sono arrivati afferma che la considerazione sociale relativa alla scelta del percorso incide positivamente nelle città congestionate. [11]

1.3.2 Connettività

Per ricevere l'etichetta *smart* le città devono poter contare sulla connettività a banda larga. Una città *smart* quindi si differenzia dalle altre esiben-

do un assemblaggio di diverse componenti per analizzare e comprendere problemi urbani tramite **tecnologie innovative** in maniera effettiva e fruibile. Il *framework* concettuale copre diverse dimensioni, tra cui l'amministrazione urbana, l'organizzazione delle infrastrutture, i trasporti e l'energia.

C'è stato un drastico incremento nel numero di sistemi che si basano su dati raccolti da **sensori** fisici, in case, edifici e città. È proprio in questo contesto di *pervasive computing* che è emersa la possibilità di progettare applicazioni di smart city che basano il loro funzionamento su tecnologie intelligenti che simultaneamente risiedono in altre applicazioni e *comunicano* tra di loro. Questa integrazione delle tecnologie *ICT* nelle infrastrutture convenzionali è parte delle iniziative strategiche dei progetti internazionali congiunti della *Connected Smart Cities Network* [12]

1.3.3 Cooperazione

Come già introdotto in precedenza, i dati sul traffico urbano possono essere acquisiti tramite sensori in strada, nei mobile devices o nei veicoli. Lo scambio di informazioni attraverso **sistemi cooperativi** che trasmettono questi dati è fondamentale per accrescere la sicurezza delle strade. A tal proposito, gli ambienti urbani costituiscono il banco di prova necessario per effettuare esperimenti realistici con una massiccia quantità di dati preziosi. Questo permette di valutare diversi protocolli, tra veicoli e servizi. Ad esempio, con la progettazione e lo sviluppo di sistemi *See-Through* [13], sono stati effettuati esperimenti in condizioni reali per testare potenziali problematiche di connettività e ritardi nella trasmissione di dati utilizzando il protocollo standard di comunicazione wireless 802.11.

Figura 1.2: Esempio di cooperazione in cui i dati visivi sulla distanza di sicurezza sono forniti dal veicolo davanti in real-time

Nel tentativo comune di implementare processi di sicurezza e *decision-making* a livello individuale, altri approcci cooperativi potrebbero aumentare la visibilità dei conducenti, ad esempio con la distanza di sicurezza. Tramite l'acquisizione e il *processing* di immagini dalla camera posteriore, possono essere fornite informazioni riguardo all'appropriatezza della distanza di sicurezza. Ne troviamo un esempio in figura 1.2

1.3.4 Veicoli Autonomi

Le VANETs (*Vehicular Ad-Hoc Networks*) e le tecnologie ad esse correlate - come ad esempio le applicazioni sulle automobili autonome - cambieranno il futuro delle città. Mark Fields - direttore esecutivo della Ford - disse: *"Il 2016 sarà un anno rivoluzionario per l'automobilismo e i trasporti, nel quale vedremo progressi radicali che cambieranno il modo di muoversi"*.

Stando all'*International Organization for Road Accident Prevention* [14], l'errore umano causa il 90% degli incidenti stradali. Per alleviare il numero di incidenti, l'introduzione di veicoli autonomi nelle strade rappresenta un'opportunità per una miglior sicurezza, rendendo non necessario l'intervento del guidatore.

I **vantaggi** dei veicoli con guida autonoma sono molteplici: un flusso del traffico ininterrotto, la riduzione del consumo di energia, l'aumento di capacità delle strade a fronte di un decremento dell'impatto aerodinamico dei veicoli, e di conseguenza della riduzione della distanza tra essi. Ciò sarà garantito da sensori che controlleranno lo spazio tra i veicoli osservando le distanze di sicurezza. È importante anche notare come molti camion con caratteristiche dei sistemi autonomi si sono già diffusi in Europa, con ottimi risultati e benefici tangibili. I veicoli autonomi possono lavorare come nodi della VANET pur comunque rimanendo in grado di identificare situazioni anomale in cui è difficile adottare le misure appropriate (lavori nelle strade, incidenti, problemi alla rete). In queste situazioni, il controllo, il mapping e la localizzazione del veicolo potrebbero basarsi sui dati disponibili da altri veicoli nella stessa VANET. [15]

Un altro punto cruciale nella diffusione dei veicoli autonomi l'opinione e l'accettazione da parte dei cittadini. È stato dimostrato che l'utilità, l'affidabilità e la facilità d'uso percepita dagli utenti ha un impatto diretto sull'adozione o meno di uno specifico strumento. Non è chiaro se chi ama guidare accetterà di rinunciare a tale azione; quel che è certo è che la fiducia gioca un ruolo fondamentale nell'adozione delle auto autonome, a cui l'utente cede parte del proprio controllo. In questa linea di ricerca, l'interazione tra veicoli autonomi e altri utenti della strada è stata investigata in numerosi studi ma al momento è stata solamente basata su scenari simulati o sondaggi. In un esperimento, alcuni pedoni sono stati forniti di un'applicazione che li avvisava quando un veicolo autonomo si avvicinava, con lo scopo di dimostrargli l'effettiva sicurezza ed affidabilità dei veicoli autonomi. [16]

I potenziali **svantaggi** della diffusione dei veicoli autonomi – tolti i dubbi sull'accettazione da parte dei conducenti che non vogliono privarsi del piacere della guida, e quelli relativi alla sicurezza e a potenziali attacchi hacker - sono argomento di future ricerche. Per esempio, può il comfort degli AV causare un incremento della rilocazione della popolazione nelle periferie e causare problemi ambientali? Oppure, il comfort degli AV può causare un progressivo abbandono dei mezzi pubblici da parte dei cittadini?

La rivoluzione portata in grembo dalla diffusione dei veicoli autonomi non può non incidere su una disparata serie di aspetti legati alla mobilità urbana. Negli ultimi anni si stanno affermando diversi progetti di *car-sharing* in cui utenti mettono a disposizione di altri utenti il proprio veicolo: a questo proposito, gli AV offrono una soluzione innovativa e intelligente che può gradualmente ridefinire la mobilità individuale, creando nuove opportunità per il *car-sharing*.

In ultimo, un'altra tematica indissolubilmente legata agli AV è il ridisegno delle strade (ad esempio, aggiungendo corsie dedicate) e l'adattamento dell'infrastruttura, come il *marking* per i segnali stradali o per i VRU al fine di una ricognizione migliore. Non è ancora chiaro in quale direzione procedere, ma

quel che è certo è che questa tecnologia rappresenta una concreta opportunità di miglioramento.

1.4 Intelligent Transportation Systems

Con **ITS** (*Intelligent Transportation Systems*) s'intende l'integrazione delle conoscenze nel campo elettronico, informatico e delle telecomunicazioni con l'ingegneria dei trasporti, per la pianificazione, progettazione, esercizio, manutenzione e gestione dei sistemi di trasporto. Questa integrazione è finalizzata al miglioramento della sicurezza della guida e all'incolumità delle persone (*safety*), alla sicurezza e protezione dei veicoli e delle merci (*security*), alla qualità e all'efficienza dei sistemi di trasporto ottimizzando l'uso delle risorse naturali e rispettando l'ambiente. Gli ITS nascono dalla necessità di gestire i problemi causati dalla congestione del traffico attraverso una sinergia delle emergenti tecnologie informatiche per la simulazione, il controllo in tempo reale e le reti di comunicazione. [17]

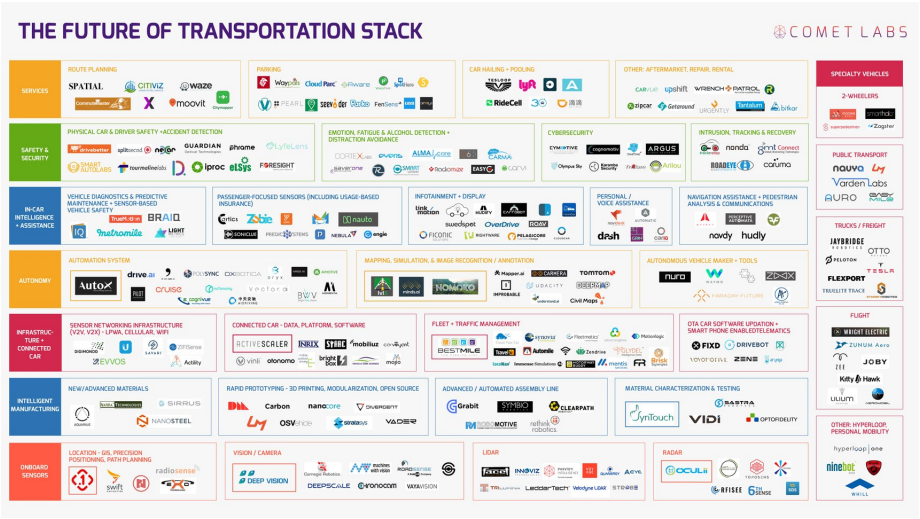

Figura 1.3: Mappa del futuro dello stack dei trasporti

1.4.1 Tecnologie di Intelligent Transportation

Gli ITS variano a seconda delle tecnologie applicate, dai sistemi di gestione base come navigatori satellitari, sistemi di controllo semaforici, rilevatori di velocità e telecamere fino ad applicazioni avanzate che integrano dati in tempo reale da fonti esterne come rilevatori meteorologici.

L'innovazione tecnologica porta in dote a questo ambito una numerosa serie di fattori abilitanti: le comunicazioni **wireless**, a corto raggio e ad ampio raggio (anche se queste ultime richiedono uno sviluppo di infrastrutture estensivo e costoso), tecnologie computazionali (con intelligenza artificiale) **embedded** nei veicoli, **floating car data**, ossia determinare l'andamento del traffico su una rete stradale tramite i dati geolocalizzati provenienti dai device mobili all'interno dei veicoli, che agiscono da sensori (con vantaggi di economicità, copertura e configurazione), tecnologie a sensori basati sulle infrastrutture (ossia installate/integrate nelle strade o in semafori, edifici, stazioni..) e individuazione video tramite telecamere.

È importante notare come lo sviluppo dell' **IoT** (Internet of Things) stia infatti aprendo la strada allo sviluppo di sistemi distribuiti che permettano di raccogliere dati sul traffico ed analizzarli. L'eventuale utlizzo di microcontrollori potrebbe permettere la mappatura fisica delle reti stradali e la distribuzione della computazione in modo tale da ideare soluzioni distribuite per la smart mobility. [17]

1.5 Agenti Autonomi

Secondo la tradizionale definizione, un agente intelligente è un sistema computazionale capace di effettuare azioni autonome e di percepire cose in un determinato ambiente. Gli agenti sono autonomi in quanto incapsulano il flusso di controllo. Il controllo non passa oltre i confini dell'agente: solo la conoscenza, le informazioni e i dati attraversano questi confini. Gli agenti non hanno interfaccia e non possono essere nè invocati nè controllati. I **MAS** (*Multi-Agent System*) possono essere concepiti come un'aggregazione di molti

centri di controllo distinti che interagiscono tra di essi scambiandosi informazioni. Il concetto di autonomia copre diversi aspetti degli agenti: si può distinguere l'autonomia *sociale* (rispetto agli altri agenti), *interattiva* (rispetto all'ambiente), *artificiale* (rispetto agli umani) e *morale* (rispetto a sè stessi).

Gli agenti sono pensati per cambiare l'ambiente in cui agiscono per raggiungere gli obiettivi preposti in fase di design. La percezione è il processo con cui l'agente prende cognizione dello stato dell'ambiente, in modo tale da adattare il proprio comportamento in base ad esso. L'azione è la funzione che rappresenta il processo di *decision-making* dell'agente e mappa sequenze di percezioni ad un'azione.

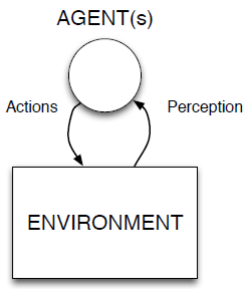

Figura 1.4: Schema delle interazioni tra l'agente e l'ambiente in cui opera

1.5.1 Comportamento

Il comportamento di un agente può essere descritto con quattro caratteristiche principali: reattivo, situato, proattivo, sociale.

Reactivity

I domini delle applicazioni del mondo reale sono caratterizzati da condizioni altamente dinamiche: cambi di situazione, informazioni incomplete, risorse scarse e azioni che hanno effetti non deterministici. Un sistema reattivo mantiene un'interazione costante con l'ambiente rispondendo ai cambiamenti che avvengono in esso con tempistiche utili. Gli agenti puramente reattivi sono quelli che decidono che cosa fare senza fare riferimento alla propria storia, basandosi esclusivamente sul presente.

Situatedness

I modelli reattivi e *stateless* non sono abbastanza per entità coinvolte in ambienti dinamici in cui si affrontano contnuamente eventi esterni che richiedono comportamenti di risposta adeguati. Un agente quindi possiede un proprio modello di azione ed è strettamente legato all'ambiente in cui vive e interagisce effettuando le azioni di cui è capace.

Proactiveness

Gli agenti devono essere in grado di adattarsi all'ambiente e ai cambiamenti, adottando un comportamento *goal-oriented*. La proattività è un approccio generativo: un agente genera i propri obiettivi e prova a raggiungerli, incapsula il proprio controllo e le regole per governarlo e prende l'iniziativa facendo in modo che qualcosa succeda – piuttosto che aspettare passivamente che qualcosa succeda.

Social ability

I MAS sono mondi complessi ed articolati e di conseguenza alcuni *goals* degli agenti potrebbero essere raggiungibili solo tramite la cooperazione con altri. La *social ability* degli agenti è l'abilità degli agenti di interagire con altri agenti (e possibilmente anche gli umani) tramite qualche sorta di mezzo di comunicazione (*signal, speech, artifacts*).

1.5.2 Ragionamento

Il processo del ragionamento degli agenti si può dividere in due categorie: il ragionamento **pratico** e il ragionamento **epistemico**. Il ragionamento *pratico* è diretto alle azioni, è il processo che porta a scoprire che cosa fare raggiungendo ciò che si desidera. Consiste di due principali attività cognitive: la *deliberazione*, ossia quando l'agente effettua decisioni sullo stato delle cose che desidera raggiungere e il ragionamento *a mezzi fini*, quando l'agente effettua decisione sul come raggiungere questo stato delle cose. Il risultato della fase di deliberazione sono le intenzioni (cosa l'agente vuole raggiungere, o co-

sa vuole fare) mentre il risultato del ragionamento a mezzi fini è la selezione di un determinato percorso di azioni (il flusso di azioni che l'agente pensa di adottare per raggiungere le sue intenzioni). Il ragionamento *epistemico*, ovvero il ragionamento diretto alla conoscenza, è il processo di aggiornamento delle informazioni rimpiazzando dati vecchi ed inconsistenti con dati aggiornati.

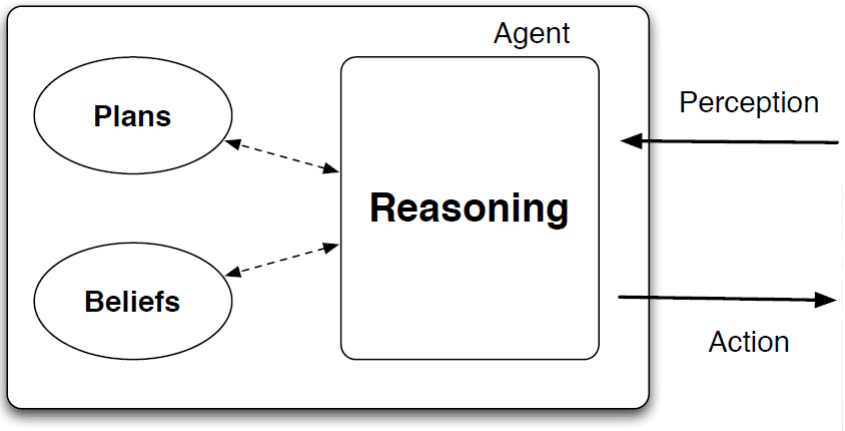

Figura 1.5: Schema del processo di ragionamento degli agenti

1.6 Paradigma A&A

Nella società umana quasi ogni contesto lavorativo in cui c'è cooperazione comprende differenti tipologie di oggetti, **strumenti o artefatti** in generale che gli umani utilizzano e condividono a sostegno delle proprie attività. Basandosi su teorie proprie delle *Human Sciences* e alle corrispettive discipline in Computer Science (ad esempio, *Computer Supported Cooperative Work o Human Computer Interaction*), questi strumenti hanno un ruolo centrale nel determinare il successo o il fallimento delle attività, giocando un ruolo fondamentale nella semplificazione di *task* complessi e più in generale nel progettare soluzioni che possano scalare con la complessità delle attività.

Analogamente al caso degli umani, anche i **MAS** (Multi-Agent System) possono ampiamente beneficiare dalla definizione e dalla sistematica interrogazione del concetto di ambiente di lavoro (*working environment*) in grado di

definire una topologia dell'ambiente computazionale, composto da diverse tipologie di artefatti, costruiti dinamicamente, condivisi ed utilizzati dagli agenti a supporto delle proprie attività lavorative.

È quindi necessario introdurre un *framework* concettuale che possa integrare tali nozioni con gli agenti: l'**A&A framework** (Agents and Artifacts). Tale framework evidenzia l'importanza dell'ambiente di lavoro in un contesto di agenti cognitivi, e presenta due caratteristiche fondamentali: **astrazioni** di carattere generale, ossia lo scopo è quello di trovare un set di astrazioni concettuali – analoghe agli agenti – che siano generali abbastanza per poter definire architetture concrete per ambienti di sviluppo e **cognitivo**, cioè le proprietà di tale ambiente devono essere concepite per venire appositamente sfruttate dagli agenti cognitivi.

Un *artefatto* nasce per essere progettato da un *MAS designer* per incapsulare qualche tipo di funzione, in questo caso sinonimo di finalità prevista. Essi sono i componenti fondamentali per la costruzione di un ambiente di lavoro complesso. Si possono distinguere due tipologie principali di artefatti: le **risorse** e gli **strumenti** (*tools*). Le prime sono la sorgente primaria e l'obiettivo delle attività degli agenti, i tools invece sono artefatti usati come strumenti per raggiungere gli obiettivi o eseguire qualche task.

L'astrazione di artefatto porta a una nozione di **uso** che è la tipologia di relazione di base tra agente e artefatto, oltre alla creazione e alla disposizione. Di conseguenza, la nozione di interfaccia d'uso è definita come il set di base di operazioni, di stati osservabili e di eventi che l'artefatto espone, rendendoli di fatto utilizzabili dagli agenti. Questa astrazione riproduce chiaramente il modo in cui gli umani usano i propri artefatti.

È inoltre fondamentale definire un **manuale** dell'artefatto, che tipicamente contiene informazioni sulla funzione dell'artefatto, come può essere usato e come agire in caso di malfunzionamento. Questi concetti sono essenziali per supportare gli umani nella comprensione di quale tipo di artefatto può essere utile per il proprio lavoro e su come possono essere usati al lato pratico.

Partendo dal modello astratto A&A sono emerse alcune tecnologie con lo scopo di avere framework concreti per prototipare applicazioni basate sui MAS. Una di queste è **CARtAgO** (*Common Artifact Infrastructure for Agent Open environment*), framework che essenzialmente fornisce la possibilità di definire nuovi tipi di artefatto, *API* adeguate per far interagire gli agenti con artefatti e *workspaces* e il supporto *runtime* per la gestione dinamica dei *workspaces*.[18]

Capitolo 2

Motivazioni, obiettivi e requisiti del sistema

2.1 Motivazioni

Uno studio condotto nel 2014 dal DESA (*Department of Economic and Social Affairs*, organo delle nazioni unite) sostiene che entro il 2050 il 70% della popolazione mondiale vivrà nelle città, che vedranno un incremento dai 2,5 miliardi di persone (dato del 2009) ai 5,2 stimati del 2050 [19]. La revisione (*World Urbanization Prospects*) sostiene che la maggiore crescita urbana – circa il 90% - avrà luogo in India, Cina e Nigeria. Gran parte della crescita stimata avverrà in paesi di regioni in via di sviluppo, particolarmente in Africa. Queste nazioni dovranno affrontare numerose sfide relative alle necessità che l'insediamento urbano porta con sè: le infrastrutture, i trasporti, l'energia, il lavoro, le abitazioni e servizi primari come l'educazione e la sanità. Tokyo – già oggi la più popolosa città al mondo – si prevede che avrà quasi 38 milioni di abitanti, seguita da Delhi, Shanghai, Città del Messico, San Paolo e Mumbai.

Figura 2.1: Vista dall'alto di Tokyo, che ad oggi conta 15 milioni di abitanti

Nel Rapporto Nazionale sullo sviluppo urbano sostenibile, consegnato dal Governo italiano alle Nazioni Unite nel luglio 2015 [20] vengono evidenziate alcune possibili soluzioni al problema dell'urbanizzazione e alle sue conseguenze sull'ambiente e sulla vita dei cittadini. In particolare, viene sottolineata l'esigenza di una "rivoluzione amministrativa che ponga in primo piano l'informazione, la trasparenza e il coinvolgimento dei cittadini nell'azione pubblica", dove "devono essere accelerati gli investimenti per assicurare l'adeguamento delle reti infrastrutturali nella mobilità e dell'IT". I campi indicati come prioritari sono la *green economy*, l'attenzione alle emissioni di carbonio, la riduzione dei consumi energetici, le *smart cities*, la messa in sicurezza dei territori, l'abbattimento della produzione di emissioni e rifiuti, la mobilità sostenibile.

Questi dati sono importanti ed è fondamentale concentrare risorse e sforzi notevoli nell'evoluzione e nello sviluppo delle città, ma quando si parla delle città come luoghi del cambiamento e dello sviluppo e del progresso dell'umanità non si può non far notare come forse ci si dimentichi del concetto stesso

di città. La città (dal latino civitas, da cui deriva anche civiltà) può essere definita come una concentrazione di popolazioni e funzioni, dotata di strutture stabili e di un territorio. Si differenzia da un paese o da un villaggio per dimensione, densità di popolazione, importanza legale, frutto di un processo più o meno lungo di urbanizzazione. Le origini delle città si possono datare intorno all'8000 A.C. in Mesopotamia, seguite dalle polis dell'antica Grecia, l'Impero romano e altri innumerevoli esempi. Ma concentrandosi sull'età moderna, guardando al periodo della rivoluzione scientifica, della rivoluzione artitistica, del rinascimento e della nascita del concetto di Europa (circa tra il 1300 e il 1600) si nota che in quel momento solamente tra il 7% e il 13% della popolazione viveva in città, mentre il resto della popolazione viveva nelle campagne. Eppure è innegabile il fatto che i centri catalizzatori del progresso siano stati proprio le città, e che quindi nell'età moderna le città siano sempre state il luogo dove si è sviluppata la cultura, la coscienza comune e l'innovazione tecnologica. Con queste premesse risulta intuitivo capire perchè i progetti relativi alle *smart cities* siano uno dei punti cruciali della crescita e dell'innovazione delle città del presente e del futuro.

2.2 Obiettivi

Gli strumenti e le tecnologie delle *smart cities* sono molteplici e rendono possibile l'apertura di un ventaglio di possibilità pressochè smisurato per l'ideazione e la progettazione di applicazioni in tale ambito. La mancanza di approcci strutturati alla materia rende possibile l'esplorazione di soluzioni che siano in grado di combinare diverse opportunità tecnologiche. L'obiettivo di questo lavoro è quello di valutare questi strumenti e queste tecnologie, studiarne la potenziale effettiva utilità ed applicabilità al dominio delle *smart cities*. Le tecnologie che verranno trattate sono solo una piccola parte dell'enorme offerta di *framework* e metodologie che possono supportare la smart mobility; l'obiettivo è descriverne alcune ed integrarle con lo scopo di costruire un *middleware* costituito da più parti, astraendo rispetto a determinate problematiche e concentrandosi su una visione d'insieme che possa indicare diverse soluzioni relative al dominio d'interesse. Ad esempio, si può pensare di considerare un

simulatore per riprodurre differenti situazioni e scenari del traffico urbano e parametrizzarlo in base alle carattreristiche del dominio urbano preso come oggetto. O ancora, si potrebbe pensare a una piattaforma ad agenti per emulare il comportamento dei cittadini che utilizzano la rete stradale, come automobilisti, autisti di mezzi pubblici, ciclisti e pedoni. Il tutto in un contesto – quello delle *smart cities* – dove le tecnologie e i framework che potrebbero in qualche modo essere utili ad applicazioni relative al campo sono veramente tanti: *Internet of Things*, comunicazioni *wireless*, uso di sensori, intelligenza artificiale, *floating car data*, *machine learning*, *big data*. La sfida è quindi quella di porre l'attenzione sui concetti e le astrazioni che non sulle caratterstiche particolari di una o dell'altra tecnologia. Per esempio, si può prendere un *middleware* qualunque (ad esempio ad agenti) ed utilizzarlo per simulare il comportamento del flusso del traffico urbano in una zona geografica di interesse, componendo propriamente un sistema complesso.

2.3 Requisiti

Il sistema che si intende sviluppare consiste in una piattaforma per la simulazione del traffico stradale all'interno di un'infrastruttura urbana. Ponendo come base un progetto preesistente che permette di restituire a un utente che ne effettua la richiesta il miglior percorso tra due punti all'interno di una rete stradale e una piattaforma iniziale sviluppata in Jason con alcuni agenti che ricalcano le caratteristiche di determinate classi di automobilisti, l'obiettivo è quello di effettuare una simulazione del traffico urbano in una città ad un livello di dettaglio più elevato, tenendo conto anche di ciclisti, pedoni e mezzi di trasporto pubblico e delle relative interazioni con semafori, incroci, fermate e stazioni. Il riferimento temporale è l'arco di una settimana.

La simulazione deve prevedere diverse tipologie di utenti che si muovono nella rete stradale: automobilisti (a loro volta suddivisi in più categorie in base alle caratteristiche e alle abitudini stradali), veicoli pubblici (autobus e tram), veicoli speciali (polizia, ambulanza) biciclette, pedoni.

Gli automobilisti devono possedere caratteristiche che li differenziano da altri automobilisti appartenenti a una categoria diversa, ognuno con determinate abitudini e un set di luoghi in cui si reca durante il corso della giornata.

Gli utenti devono essere autonomi: una volta conosciuti i luoghi e gli orari in cui si devono muovere, essi devono spostarsi all'interno dell'area e compiere le azioni e raggiungere gli obiettivi prefissati in ogni giornata in maniera del tutto autonoma.

La simulazione deve prevedere le interazioni degli utenti con semafori, stazioni e fermate di tram e autobus.

In particolare, tutti i veicoli devono rendersi conto di quando si trovano in prossimità di un semaforo ed interpretare correttamente le indicazioni fornite da esso (ovvero dover aspettare o dover ripartire). I veicoli speciali devono inoltre essere in grado di intervenire sullo stato dei semafori, ad esempio bloccandoli in caso di emergenza. I semafori devono inoltre poter assumere una funzione attiva in determinate situazioni di emergenza, ad esempio indicando ai veicoli l'interruzione di una tratta e reindirizzandoli verso una determinata direzione.

I pedoni decideranno se effettuare il tragitto a piedi oppure recandosi presso una stazione o una fermata dell'autobus o del tram. Le stazioni (e più in generale, tutte le fermate intermedie delle tratte dei mezzi pubblici) devono gestire l'arrivo di nuovi passeggeri ed interfacciarsi con i mezzi pubblici in modo tale da metterli al corrente del numero di passeggeri che saliranno su di essi presso quella stazione. Esse devono anche poter segnalare – nel caso ne si verificasse l'eventualità – la propria impossibilità a svolgere la propria regolare funzione o la presenza di guasti e problemi.

Capitolo 3

Analisi

3.1 Analisi dei requisiti

Partendo dai requisiti del sistema, in questa sezione verranno definite le specifiche dei requisiti e formalizzate le funzionalità che il sistema deve fornire.

Il sistema consiste di una piattaforma per simulare il traffico stradale all'interno di una determinata area urbana partendo da un'applicazione già esistente in grado di restituire il miglior percorso tra due punti e il relativo tempo di percorrenza. Trattandosi di un'applicazione in cui bisogna *simulare* il flusso del traffico risulta totalmente nulla l'interazione dell'utente con il sistema, in quanto tutte le interazioni avvengono all'interno dei confini del sistema stesso.

Una specifica importante è la presenza contemporanea di diversi profili di *road-users* all'interno della simulazione. Sarà opportuno quindi nelle fasi successive effettuare scelte che consentano di poter caratterizzare fortemente le varie categorie di utenti all'interno della simulazione e poterne differenziare aspetti e comportamenti. Il caso degli automobilisti induce a pensare a un'ulteriore scissione (da categoria a sotto-categoria) in quanto dovranno essere presenti automobilisti con abitudini e caratteristiche diverse tra loro.

Gli utenti devono essere autonomi. Questo significa che dovranno essere forniti di una conoscenza iniziale sui luoghi che intendono visitare e sui giorni

e gli orari in cui devono effettuare i propri movimenti all'interno dell'area urbana. Indirettamente, questa specifica induce a considerare un importante requisito non funzionale: bisogna introdurre all'interno della simulazione il concetto di tempo, per coordinare le attività degli utenti e permettergli di poter effettuare le proprie azioni in maniera autonoma.

Il sistema prevede che ogni categoria di utente della strada interagisca con elementi del dominio stradale: semafori, stazioni e fermate di tram e autobus. L'interazione con i semafori riguarda tutte le categorie di utenti e prevede che una volta giunti in prossimità di un semaforo l'utente aspetti fino a che il semaforo non gli indichi di ripartire. L'interazione con le stazioni e le fermate di tram e autobus riguarda i pedoni e i mezzi pubblici.

Per quanto riguarda i pedoni, essi devono valutare in fase di inizializzazione se è possibile recarsi a piedi nel posto in cui intendono andare oppure se gli conviene recarsi presso la fermata più vicina ed utilizzare i mezzi pubblici, *registrandosi* presso la stazione/fermata e comunicando la propria destinazione.

I mezzi pubblici invece dovranno essere forniti in fase di inizializzazione delle informazioni di base riguardo agli orari e alle tratte di competenza ed effettuare il proprio servizio trasportando i passeggeri da una stazione/fermata all'altra in base alle esigenze di questi ultimi.

I mezzi speciali – polizia e ambulanza – devono essere in grado di reagire alle situazioni di emergenza. Quando si verifica una situazione di emergenza questi veicoli devono effettuare una richiesta di percorso dalla posizione attuale alla posizione dove si è verificata l'emergenza e poter essere in grado di *bloccare* i semafori che si trovano lungo il percorso per poter agevolare il proprio passaggio.

3.2 Scenari

Per organizzare e comprendere più in dettaglio le diverse tipologie di situazioni che si manifesteranno durante il funzionamento dell'applicazione, è risultato utile definire una serie di scenari utlizzando il formalismo UML dei diagrammi di sequenza. Di seguito verrà anche definito un breve glossario per comprendere i termini utilizzati nei diagrammi

3.2.1 Glossario

- **User**: per User si intende qualsiasi utente della simulazione, ossia automobilisti, veicoli speciali (*special vehicles*), mezzi pubblici (nei diagrammi indicati come *bus*), pedoni (*pedestrian*).

- **Bus Stop**: rappresenta il concetto di stazione o fermata dei mezzi pubblici, luogo in cui i pedoni si registrano per aspettare il transito dei mezzi pubblici.

- **Semaphore**: concetto di semaforo, dove i veicoli transitano o si fermano in base allo stato

- **Server**: l'applicazione in grado di restituire il miglior percorso tra due punti a chi ne effettua richiesta

- **System**: modella tutte le interazioni e le comunicazioni tra gli utenti della simulazione e il sistema, eccetto le richieste di percorso di cui si occupa il server

3.2.2 Request Path Scenario

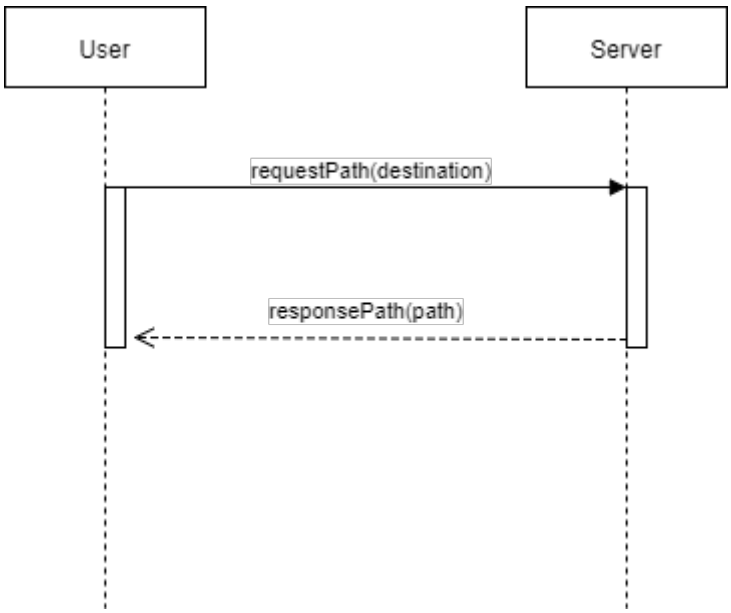

Figura 3.1: Request Path Scenario

Request Path Scenario	
ID	Request Path
Attori	User (ogni categoria), System
Descrizione	L'utente richiede il miglior percorso tra la propria posizione e un'altra posizione, il server risponde con il miglior percorso
Main Scenario	La richiesta del miglior percorso è l'operazione effettuata autonomamente da ogni veicolo che intende effettuare uno spostamento

3.2.3 User Initialize Scenario

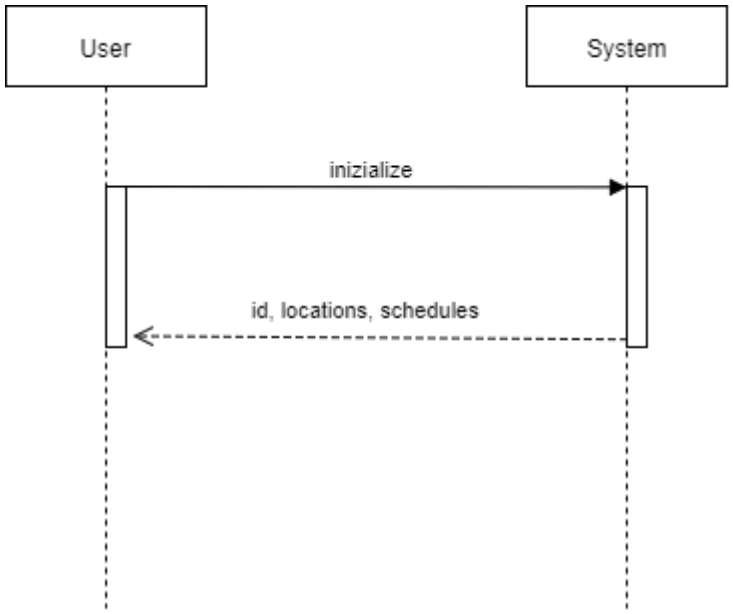

Figura 3.2: User Initialize Scenario

User Initialize Scenario	
ID	User Initialize
Attori	User (ogni categoria), System
Descrizione	L'utente si registra al sistema che gli assegna un id e gli fornisce la conoscenza di base di cui necessita per le gli obiettivi e le attività da svolgere
Main Scenario	L'utente contatta il sistema per ottenere la lista delle coordinate dei luoghi che deve visitare e la lista degli orari in cui deve svolgere le proprie attività

3.2.4 Clock Setting Scenario

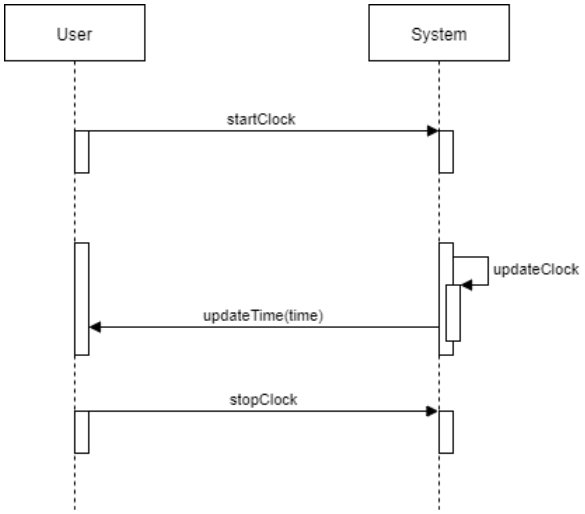

Figura 3.3: Clock Setting Scenario

Clock Setting Scenario	
ID	Clock Setting
Attori	User (ogni categoria), System
Descrizione	Ogni utente richiede al sistema di ricevere aggiornamenti ogni volta che il tempo viene aggiornato
Main Scenario	L'utente dopo la fase di inizializzazione comunica al sistema che vuole essere aggiornato continuamente sul cambio dell'orario. Il sistema avrà un clock che con un proprio flusso di controllo aggiornerà continuamente la conoscenza relativa allo stato attuale del tempo degli utenti che hanno richiesto di essere aggiornati. Quando l'utente non necessiterà più di essere aggiornato, potrà stoppare il clock

3.2.5 Semaphore Interaction Scenario

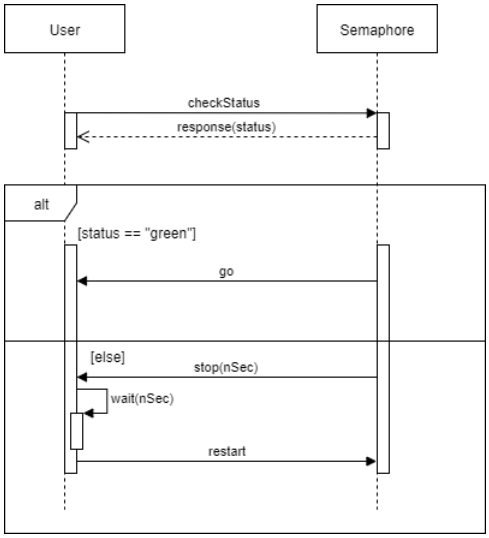

Figura 3.4: Semaphore Interaction Scenario

Semaphore Interaction Scenario	
ID	Semaphore Interaction
Attori	User (tutti i veicoli), Semaphore
Descrizione	I veicoli che si avvicinano a un semaforo ne richiedono lo stato attuale e in base alla risposta transitano oppure attendono
Main Scenario	L'utente mentre si muove lungo il percorso può trovarsi in un punto in cui è presente un semaforo. In questo caso controlla in che stato si trova attualmente il semaforo (rosso, verde, giallo) e nel caso in cui sia verde transita, altrimenti il semaforo comunica di fermarsi con il relativo tempo di attesa. Dopo aver aspettato quel tempo, l'utente può ripartire

3.2.6 Pedestrian Interaction Scenario

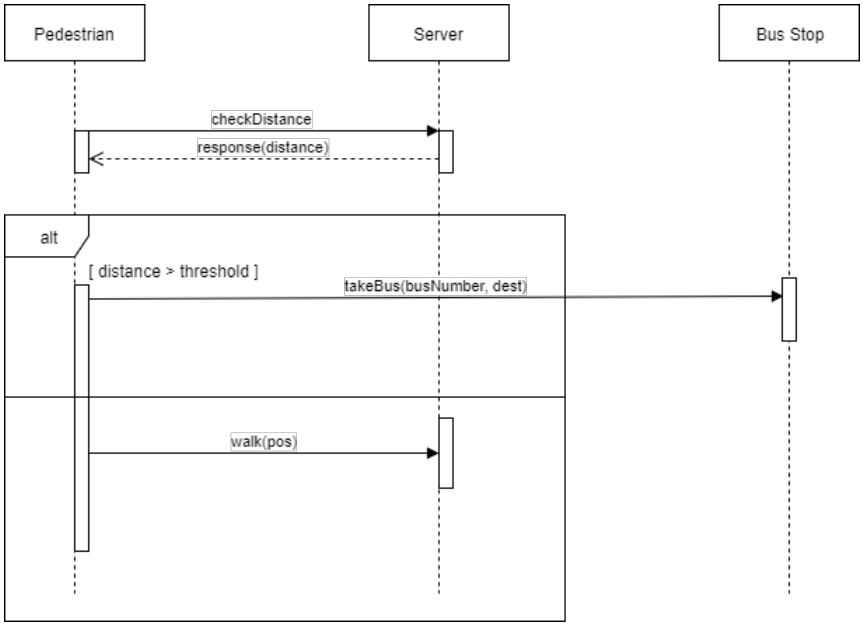

Figura 3.5: Pedestrian Interaction Scenario

Pedestrian Interaction Scenario	
ID	Pedestrian Interactions
Attori	Pedestrian, Server, Bus Stop
Descrizione	L'utente valuta se effettuare il percorso a piedi oppure utilizzando i mezzi pubblici
Main Scenario	I pedoni richiedono al server la distanza che vogliono percorrere e la confrontano con una soglia massima entro la quale decidono di percorrere la tratta a piedi. In caso contrario, si recano presso la stazione / fermata dell'autobus più vicina comunicando il numero dell'autobus e la destinazione

3.2.7 Bus Interaction Scenario

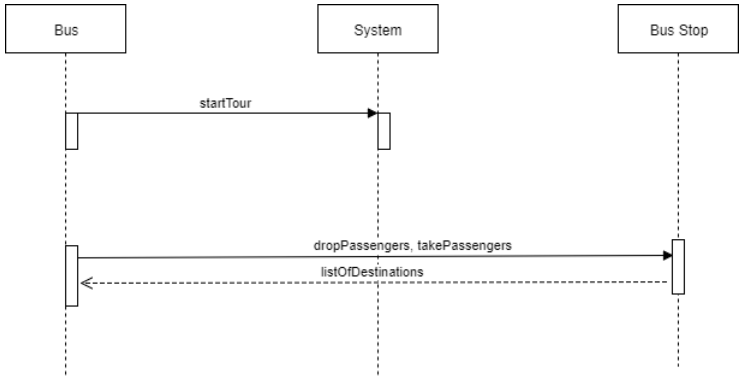

Figura 3.6: Bus Interaction Scenario

Bus Interaction Scenario	
ID	Bus Interaction
Attori	Bus, System, Bus Stop
Descrizione	I mezzi pubblici durante le proprie tratte raccolgono e rilasciano i pedoni alle fermate o alle stazioni
Main Scenario	Al bus in fase di inizializzazione viene fornita la conoscenza sugli orari e sulle stazioni e le fermate in cui deve transitare. Dopo aver iniziato la propria tratta, il bus interagisce di volta in volta con le fermate rilasciando i pedoni che devono scendere in quel punto e facendo salire coloro che hanno manifestato l'intenzione di usare quel bus, con le relative destinazioni

3.2.8 Emergency Scenario

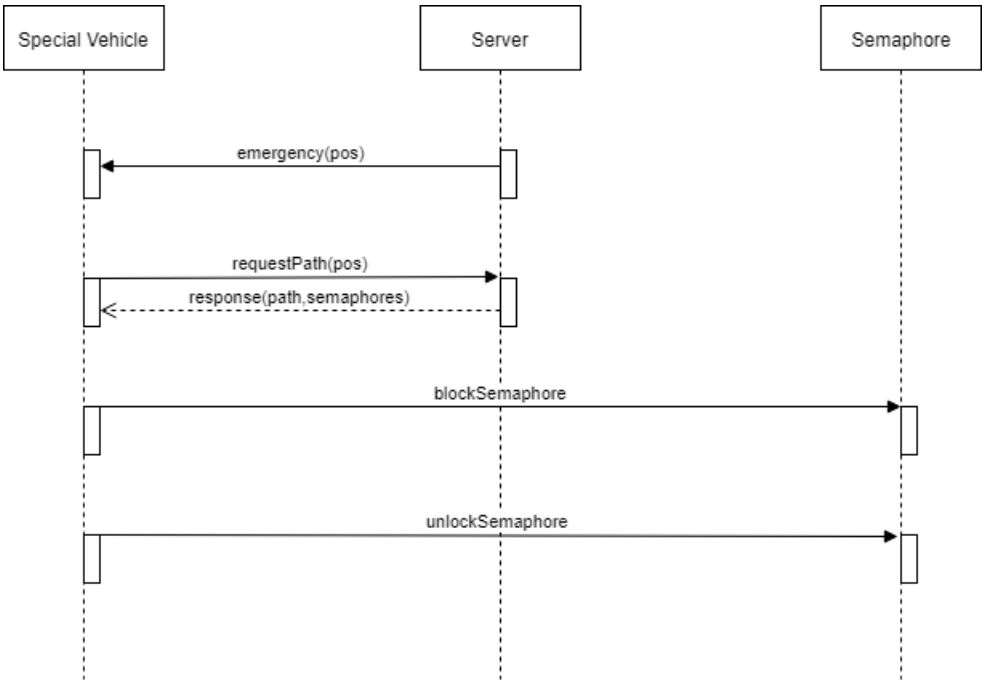

Figura 3.7: Emergency Scenario

Emergency Scenario	
ID	Emergency
Attori	Special Vehicle, Server, Semaphore
Descrizione	I veicoli speciali agiscono sullo stato dei semafori nelle situazioni di emergenza
Main Scenario	Quando i veicoli speciali come polizia ed ambulanza ricevono un messaggio di emergenza, effettuano una richiesta di percorso in cui viene indicata la lista di semafori presenti nel tragitto. I veicoli speciali quindi bloccano tali semafori per agevolare il proprio transito, dopodichè una volta giunti a destinazione li sbloccano facendoli tornare al proprio regolare funzionamento

3.3 Analisi del problema

La piattaforma ad agenti presa come punto di partenza per il sistema è basata su Jason, framework che estende AgentSpeak e consente la creazione di sistemi multi-agente (MAS). Uno dei più noti approcci allo sviluppo di agenti cognitivi è l'architettura BDI (Beliefs, Desires, Intentions). AgentSpeak è stato uno dei più influenti linguaggi astratti BDI-based. Jason costituisce un'estensione di AgentSpeak, implementata in Java.

L'accento – in Jason – è quindi posto sugli agenti e sulle loro azioni, mentre i requisiti del sistema inducono a pensare che oltre agli agenti sia opportuno modellare il concetto di ambiente di lavoro (workspace) in cui gli agenti si muovono e una serie di oggetti e strumenti che gli agenti possano accedere, condividere, interrogare e in qualche modo supportare le attività.

A tale scopo, viene naturale pensare a una struttura concettuale che possa in un qualche modo integrare il concetto di agenti autonomi con quello di *open-space environment* e *workspace* con strumenti e oggetti appositamente progettati per agevolare le attività degli agenti e che possano essere dinamicamente creati ed utilizzati da essi.

Una possibile soluzione a questa esigenza è rappresentata dal framework A&A (Agents and Artifacts) che introduce il concetto di artefatto come astrazione di prima classe per modellare e progettare ambienti computazionali MAS. La definizione di un ambiente di lavoro (*workspace*) permette di esplicitare una topologia dell'ambiente computazionale, popolato di agenti ed artefatti di svariata tipologia e utilizzati dagli agenti a proprio supporto.

Adottando questo modello di riferimento, risulta abbastanza logico mappare concettualmente gli agenti con le varie categorie di utenti che si muoveranno all'interno del contesto urbano, luogo della simulazione; oltre alle categorie di automobilisti presenti nella piattaforma di partenza, sono associabili al concetto di agente anche i mezzi di trasporto pubblici come tram e autobus, mezzi speciali come polizia o ambulanza, pedoni, biciclette e motocicli.

Gli artefatti sono in grado di racchiudere un'ampia selezione di strumenti a supporto delle attività degli agenti, ma rappresentano anche entità vere e proprie in grado di modellare elementi propri del dominio applicativo. È necessario precisare che il concetto di artefatto nasce per modellare entità reattive e passive (e non proattive, come nel caso degli elementi): ciò si adatta particolarmente ad elementi come semafori, stazioni e fermate dei mezzi pubblici.

Gli artefatti quindi – oltre a modellare elementi propri del contesto urbano – sono astrazioni su cui poter progettare un'ampia selezione di strumenti a supporto degli agenti: artefatti in grado di supportare la fase di inizializzazione degli agenti, artefatti che gestiscano la comunicazione e le richieste di percorso degli agenti, artefatti che incapsulino il concetto di tempo. È fondamentale ai fini di ottenere un sistema solido la corretta progettazione dell'interfaccia d'uso di tali artefatti, ossia la tipologia di relazione di base che lega gli agenti con gli artefatti.

Pensando in termini architetturali, alla luce delle precedenti considerazioni si possono distinguere quindi due macro-componenti: il server illustrato nel capitolo 4 (Background) che restituisce agli utenti che lo richiedono il miglior percorso tra due punti e il Multi-Agent system nel quale saranno inclusi anche gli artefatti che gli agenti utilizzano. In fase progettuale sarà necessario considerare anche la forma di comunicazione tra le due parti.

3.4 Architettura Logica

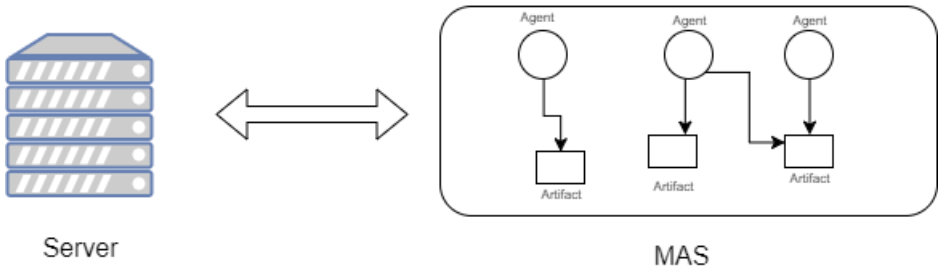

Figura 3.8: Schema che mostra l'architettura logica

Per una migliore comprensione delle due macro-componenti risulta utile definire un modello in termini di struttura, interazioni e comportamento.

3.4.1 Server

Nel server è racchiusa la conoscenza del dominio urbano preso come ambito della simulazione. Vengono memorizzate le posizioni di tutti i nodi nella mappa, quali sono i nodi vicini a cui è collegato e quali sono i tempi di percorrenza tra uno e l'altro.

- **Struttura**: il server è atomico, non si distinguono sottoparti
- **Interazioni**: viene contattato e risponde alle richieste degli agenti
- **Comportamento**: rimane in attesa delle richieste degli utenti, calcola il percorso migliore e risponde

3.4.2 MAS

Il MAS è la macro-componente nella quale operano gli agenti autonomi che effettuano le richieste di percorso. Ogni agente si interfaccia con diversi artefatti all'interno degli *workspaces*; alcuni artefatti rappresentano degli elementi del dominio stradale urbano (semafori, stazioni, fermate dei mezzi pubblici), altri sono strutture di supporto agli agenti che permettono di ottenere la conoscenza iniziale riguardo alla propria posizione, alla posizione dei luoghi rilevanti

per ciascun agente (ad esempio la posizione del proprio ufficio o di altri luoghi di interesse che l'agente dovrà prima o poi visitare nell'arco della simulazione) e agli orari in cui devono svolgere le proprie attività. Anche la gestione delle interazioni con il server è gestita tramite un artefatto, che riceve le richieste degli agenti e le inoltra al server rimanendo in attesa e restituendo il risultato della richiesta all'agente che ne ha effettuato la richiesta

- **Struttura**: il MAS è composto da un set di tipologie di agenti – ognuna delle quali comprende un numero di agenti che può variare in base a parametri relativi all'area della mappa presa in considerazione - e da artefatti, ossia entità passive relative al dominio urbano e di supporto agli agenti

- **Interazioni**: la principale forma d'interazione del MAS con il resto del sistema è la richiesta che viene proattivamente effettuata dall'agente a un apposito artefatto che la inoltra al server e ne gestisce la risposta

- **Comportamento**: il MAS è un'entità sostanzialmente proattiva: gli agenti possiedono la conoscenza di base che gli permette di sapere in ogni momento quali azioni compiere, in quale momento e quando effettuare le richieste al server, basando le proprie azioni successive sulle risposte ricevute.

Capitolo 4

Background

Il sistema preso come punto di partenza della simulazione è un'applicazione Java che consente ad utenti che ne fanno richiesta di ottenere il tempo di percorrenza stimato riguardo ad un determinato percorso. Ne verranno descritte le caratteristiche principali e gli accorgimenti effettuati per integrarlo con la piattaforma utilizzata per la simulazione.

L'applicazione ha l'obiettivo di prevenire ed individuare congestioni all'interno di una rete stradale urbana. Il sistema nasce con la seguente struttura: è composto da un server che gestisce le richieste iniziali di elaborazione del percorso, da una serie di device infrastrutturali posizionati all'interno della rete in prossimità di ogni incrocio e da un device utente messo in dotazione per ogni veicolo e dotato di localizzatore GPS.

Nel momento in cui un utente vuole iniziare un percorso all'interno della rete urbana, invia attraverso il device una richiesta al server il quale risponde con una serie di possibili percorsi. Tali percorsi includono gli ID dei device infrastrutturali da contattare per conoscere il tempo di percorrenza. Il server conosce la posizione di tutti i device infrastrutturali e come sono collegati tra loro: in questo modo è in grado di determinare quali siano i percorsi più veloci, senza considerare però le eventuali congestioni.

L'utente a questo punto deve contattare il primo device infrastrutturale di ogni percorso che ha ricevuto, ognuno di questi contatta il secondo e così si prosegue nella comunicazione fino a che l'ultimo device del percorso che restituisce il tempo di percorrenza all'utente calcolando le congestioni previste. L'utente valuta il percorso più veloce in base alle risposte e dichiara la sua effettiva volontà di percorrerlo al primo device del percorso che a sua volta comunicherà con il secondo il passaggio previsto dell'utente, e così via come in precedenza.

È opportuno a questo punto precisare che – per la simulazione – i device utente vengono rimpiazzati dagli agenti autonomi facenti parte di un MAS e la rete di device infrastrutturali viene incorporata interamente nel server, il quale manterrà tutte le informazioni necessarie (posizione di ognuno, collegamenti e tempi di percorrenza tra uno e l'altro) per soddisfare le richieste degli agenti, semplificando di fatto la comunicazione. Nella simulazione si astrae dalla gestione delle congestoni delegando al server (e non all'infrastruttura distribuita sul territorio) la funzionalità di restituire a chi ne fa richiesta il miglior percorso tra due punti con il relativo tempo di percorrenza basato sui dati e i parametri del server.

Capitolo 5

Design dell'architettura

In questa sezione verrà illustrato il processo di progettazione e design dell'architettura progettuale, che costituisce un modello del sistema ispirato ai requisiti che ha l'obiettivo di identificare e descrivere i macro sotto-sistemi in cui il sistema si articola e il processo che ha portato al loro design. L'architettura di progetto è basata sull'architettura logica descritta in fase di analisi. Risponde alla necessità di mappare ogni componente definito in fase di analisi nella sezione 3.4 a livello logico in una concreta realizzazione a livello fisico.

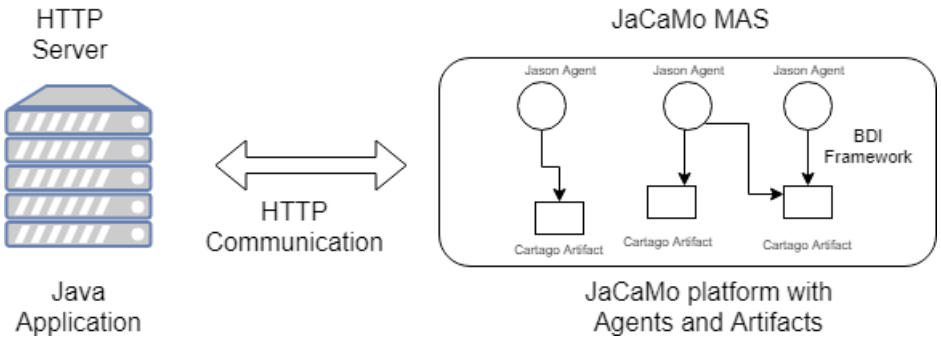

Figura 5.1: Architettura di progetto

L'applicazione è composta da due macro-componenti principali: un'applicazione Java che svolge la funzione del Server (introdotta nel capitolo **??**) e un MAS sviluppato tramite la piattaforma *JaCaMo* con artefatti e agenti autonomi che effetuano richieste assumendo il ruolo del Client. Le due macro-componenti hanno la principale forma di interazione nelle richieste HTTP con-

tenenti l'interrogazione su quale sia il miglior percorso tra due nodi appartenti alla mappa e quale sia il tempo di percorrenza tra essi.

La mappa rappresenta il dominio dell'area urbana presa in considerazione ed è espressa come una matrice nella quale ogni punto ha due coordinate (x,y) e ogni incrocio tra due strade ed ogni fermata o stazione dei mezzi pubblici rappresenta un punto significativo. Ognuno di questi punti rappresenta un nodo di un grafo e il Server conosce quali sono i nodi vicini e quali sono i tempi di percorrenza tra uno e l'altro e - in questo modo - è in grado di calcolare i tempi di percorrenza tra i nodi.

Il processo di design risponde alla necessità di stabilire un **mapping** tra i concetti emersi in fase di analisi e una concreta realizzazione di essi in fase di progettazione. La fase di analisi ci ha fornito in output una serie di astrazioni di base che in fase di design dovranno trovare una attuazione pratica. Queste astrazioni (descritte nella sezione Glossario) sono costitute dall'**User** (di cui si possono distinguere diverse sotto-categorie: **Vehicle, Special Vehicle, Bus, Pedestrian**), da elementi come **Bus Stop** e **Semaphore**, dal **Server** e dal **System**. Risulta utile differenziare sin da subito il Server dal System in quanto il Server è una componente del sistema esterna al Multi-Agent system con cui gli agenti si interfacciano e comunicano solo ed esclusivamente nell'evenienza in cui debbano effettuare richieste di percorso – mentre per System si intende tutta la parte interna al MAS che fornisce supporto agli agenti durante il loro ciclo di vita (inizializzazione, gestione del tempo, gestione delle attività quotidiane).

Risulta evidente che il MAS rappresenta la macro-componente del sistema più ampia e complessa e per comprendere meglio le diverse sotto-parti in cui il MAS si articola è necessario elencare e descrivere le diverse astrazioni di base definite in fase di analisi e realizzate concretamente tramite la piattaforma JaCaMo.

5.1 Server

Il server è realizzato tramite un'applicazione Java che gestisce la zona urbana di competenza con una mappa dei nodi presenti e dei collegamenti tra un nodo e l'altro con i relativi tempi di percorrenza. Il server è un HTTP Server che si mette in attesa di messaggi e prevede un metodo per gestire le richieste di miglior percorso da parte degli agenti, basandosi su un algoritmo *shortest path*.

5.2 HTTP Communication

La scelta di utilizzare il protocollo HTTP per gestire la comunicazione è giustificata dal fatto che la comunicazione tra gli Agenti e il Server è sincrona: negli scenari in cui viene fatto uso di HTTP, l'agente che effettua la richiesta deve rimanere in attesa della risposta da parte del Server.

5.3 Agenti Autonomi

Gli agenti autonomi sono la tecnologia utilizzata per realizzare gli User della simulazione. Il concetto di agente autonomo è particolarmente adatto a modellare un'astrazione come quella dell'utente della strada nell'ambito di una simulazione. Quando si parla dell'infrastruttura stradale nel suo complesso risulta logico mapparla in un *workspace* dove appunto i protagonisti sono automobilisti, mezzi pubblici e ciclisti o pedoni con una forte autonomia e che quindi possono essere descritti efficacemente dagli agenti autonomi. Nella sezione 1.5 si è parlato di agenti intelligenti come sistemi in grado di "effettuare azioni autonome e percepire cose in un determinato ambiente", con diversi aspetti di autonomia (autonomia sociale, interattiva, artificiale e morale). Gli agenti sono pensati per raggiungere degli obiettivi preposti in fase di design e si dicono cognitivi in quanto prendono cognizione dello stato dell'ambiente tramite l'atto della percezione. Ripetendo queste parole e questi concetti non si può fare a meno di notare come ricordino in maniera decisa il comportamento degli automobilisti, e come risultato di questa corrispondenza la scelta

è ricaduta proprio sugli agenti autonomi per mappare le astrazioni definite in fase di analisi con una concreta realizzazione nelle fasi di design e sviluppo.

Di seguito verranno indicate le principali categorie di agenti all'interno della simulazione.

- **Business Man**

  L'agente Business Man durante la settimana si reca a lavoro e mangia al ristorante nella pausa pranzo. Durante il weekend, va al cinema il sabato sera e allo stadio alla domenica pomeriggio.

  Conoscenza iniziale: posizione di casa, dell'ufficio, dello stadio, del cinema, del ristorante. Orari delle attività.

- **Housewife**

  L'agente Housewife durante la settimana accompagna i figli a scuola alla mattina e – se lo status del frigorifero lo rende necessario – va al supermercato a fare la spesa. Poi va a prendere i figli a scuola e nel pomeriggio li accompagna nei luoghi di altre attività e li va a prendere. Durante il weekend, va dalla parrucchiera o dall'estetista e a fare shopping.

  Conoscenza iniziale: posizione di casa, della scuola, del supermercato, dell'estetista/parrucchiera, del centro commerciale, delle attività che frequentano i figli. Orari delle attività.

- **Universitary Student**

  L'agente Universitary Student durante la settimana va all'università, una volta uscito in palestra e poi in certe serate va al bar. Nel weekend, va in discoteca al sabato sera.

  Conoscenza iniziale: posizione di casa, università, palestra, bar, discoteca. Orari delle attività.

- **Bus**

  L'agente Bus ogni giorno effettua una tratta a determinati orari trasportando i passeggeri che raccoglie alle fermate lungo il tragitto

  Conoscenza iniziale: posizione di ogni fermata della tratta, orari in cui inizia e finisce la tratta

- **Pedestrian**

  L'agente Pedestrian in fase di inizializzione effettua un controllo sulla distanza che lo separa dal proprio ufficio e se la distanza è sotto a una soglia predefinita effettua il tragitto a piedi altrimenti si reca alla fermata più vicina e aspetta il mezzo pubblico che lo porta nella fermata più vicina alla posizione dell'ufficio.

  Conoscenza iniziale: posizione di casa, dell'ufficio. Orari lavorativi

- **Special Vehicle**

  L'agente Special Vehicle si attiva quando riceve il messaggio *emergency* dal sistema e si reca immediatamente nella posizione in cui si è verificata l'emergenza bloccando i semafori che si trovano lungo il percorso per agevolare il passaggio. Una volta finito lo stato di emergenza, sblocca tali semafori rendendoli nuovamente opearativi

  Conoscenza iniziale: posizione della *base* (distretto per la polizia, pronto soccorso per l'ambulanza), lista dei semafori

In fase di design del MAS, sono state effettuate esplorazioni e riflessioni sui principali design pattern per sistemi multi-agente. Il framework BDI (Belief-Desire-Intention) è risultato uno dei più adatti al caso di studio della simulazione.

5.4 BDI Framework

Il framework BDI è uno dei più popolari framework per le tecnologie ad agenti definito da Rao e Georgeff. Gli aspetti principali su cui si basa questo approccio sono quelli di *belief* (conoscenza), *desire* (desiderio) e *intention* (intenzione); è per questo che spesso ci si riferisce agli agenti di tale framework come BDI-Agents.

- **Beliefs**: rappresentano in qualsiasi momento la conoscenza del mondo da parte dell'agente, incluse le informazioni sullo stato attuale dell'ambiente inferite da sensori, messaggi da altri agenti o informazioni interne. Nel sistema, la conoscenza iniziale viene fornita agli agenti tramite un artefatto che gestisce l'inizializzazione di questi ultimi e in base alla categoria di agente gli fornisce informazioni utili durante l'arco della simulazione. Queste informazioni riguardano le posizioni dei luoghi da raggiungere e gli orari in cui svolgere le proprie attività. L'elemento del tempo è la chiave per comprendere la gestione della coordinazione degli agenti all'interno del sistema.

- **Desires**: rappresentano uno stato del mondo che gli agenti provano a raggiungere

- **Intentions**: sono i mezzi scelti dall'agente per raggiungere i desires dell'agente, generalmente implementati tramite *plans* e *post-conditions*. Queste *intentions*, relative ai *desires*, possono essere pensate come eseguibili in concorrenza, con una sola intention attiva in ogni momento.

Oltre a queste componenti, il modello BDI include una libreria di *plan*, ossia un set di ricette che rappresentano la conoscenza procedurale dell'agente e una coda di eventi, dove sono memorizzati gli eventi (percepiti dall'ambiente oppure auto-generati dall'agente) e gli obiettivi interni (generati dall'agente stesso mentre cerca di risolvere un desire). I *plan* degli agenti devono essere generati nel momento del design, mentre a *run-time* vengono selezionati per l'esecuzione. È importante notare che – adottando un approccio basato sull'interazione degli agenti con gli artefatti – la grande maggioranza delle in-

terazioni degli agenti con l'ambiente circostante e con il sistema sarà costituita – appunto – dalla comunicazione agente-artefatto.

5.5 Artefatti

Gli artefatti rappresentano una parte fondamentale del MAS e rendono possibile la coordinazione tra gli agenti, la loro inizializzazione, la gestione delle loro richieste e delle interazioni con gli oggetti e gli elementi del dominio urbano come semafori, stazioni e fermate. Come già introdotto nella sezione Paradigma A&A, gli artefatti sono entità *stateful, function-oriented* e non autonome. Sono controllabili ed osservabili dagli agenti e modellano gli strumenti e le risorse che gli agenti usano. Queste funzionalità vengono incapsulate all'interno degli artefatti, che vengono poi strutturati in *workspaces*.

A differenza degli agenti, gli artefatti – che sono entità che forniscono funzioni e servizi – in determinati casi necessitano di fare riferimento a una memoria condivisa nella quale ci sono i riferimenti spaziali relativi alla mappa e alle posizioni dei luoghi di interesse. Di seguito vengono elencati i principali artefatti:

- **Semaphore**: artefatto situato che modella il semaforo. Interagisce direttamente con gli agenti che si trovano in prossimità di esso indicando di fermarsi nel caso il proprio stato sia *rosso* e di aspettare per un determinato numero di secondi, che varia di semaforo in semaforo. L'interfaccia d'uso prevede anche la possibilità (per i veicoli speciali) di bloccare i semafori di una determinata tratta nel caso in cui si verifichi un'emergenza

- **Bus Stop**: artefatto situato che modella fermate e stazioni dei mezzi pubblici. Interagisce sia con l'agente Bus sia con l'agente Pedestrian. I pedoni che intendono utilizzare i mezzi pubblici si recano presso la fermata più vicina alla propria posizione attuale. Una volta giunti alla fermata si registrano comunicando all'artefatto il numero identificativo del mezzo che intendono utilizzare e la posizione della propria destinazione. I mezzi

pubblici – quando si trovano in prossimità di una fermata o stazione che appartiene alla propria tratta – effettuano la fermata e prima scaricano i passeggeri a bordo che hanno come destinazione la stessa posizione della Bus Stop, poi caricano i passeggeri che si sono registrati per quel mezzo e l'artefatto comunica la destinazione di ogni passeggero

- **Clock**: artefatto che gestisce la coordinazione degli agenti. Il Clock nella sua interfaccia d'uso espone l'operazione *start* che ogni agente deve effettuare per poter essere al corrente del tempo. Il Clock definisce quindi proprietà osservabili dagli agenti (giorno, ora, minuto, secondo) che aggiorna continuamente tramite un'operazione interna. Quando un agente non necessità più di essere consapevole del tempo, può effettuare l'operazione *stop*

- **Initializer**: artefatto che gestisce l'inizializzazione degli agenti. La prima operazione che effettua un agente è initialize in cui l'artefatto lo informa sul suo *id* e sulla sua posizione.

- **Request Manager**: artefatto che gestisce le interazioni degli agenti con il Server. Le operazioni principali sono *requestPath* con il quale i veicoli richiedono il percorso al Server e *checkBus* con cui i pedoni controllano se gli convenga o meno prendere i mezzi per recarsi a destinazione. L'artefatto prevede delle operazioni interne per calcolare i tempi di percorrenza dei percorsi.

  È possibile definire una tassonomia degli artefatti per categorizzarli e comprendere l'ampio ventaglio di possibilità che offrono.

5.5.1 Boundary Artifacts

I *boundary artifacts* (da *bound*, confine, limite) sono una categoria di artefatti usati per caratterizzare e controllare la presenza di un agente all'interno di un contesto organizzativo. Possono essere considerati come oggetti che forniscono un'interfaccia tra l'agente e l'ambiente in cui agisce. In questo caso, stazioni, fermate e semafori sono classificabili come

boundary artifacts, ovvero elementi che appartengono al dominio spaziale preso in oggetto nel sistema ma che non sono proattivi e *goal-oriented* come gli agenti, consentendo di mantenere separate queste astrazioni di prima classe e strutturare il sistema in maniera efficiente. Un'altra importante caratteristica degli artefatti che semafori, stazioni e fermate rispecchiano è quella dell'estensione spaziale: dato un MAS con una topologia, lo stesso tipo di artefatto può coprire più nodi, risultando concettualmente e fisicamente distribuito.

5.5.2 Social Artifacts

I *social artifacts* sono la categoria di artefatti progettati per fornire funzionalità per strutturare e gestire le interazioni all'interno di un MAS. Appartengono a questa categoria artefatti come l'orologio e come il gestore di richieste. Il primo è responsabile della coordinazione degli agenti, facendo in modo che gli agenti possano conoscere lo stato osservabile e quindi essere consapevoli dello stato attuale dell'orologio per poter coordinare le proprie attività quotidiane che si svolgono a determinati orari forniti all'agente nella sua conoscenza iniziale. Il secondo gestisce le interazioni degli agenti con il server, ad esempio le richieste di miglior percorso oppure il controllo che effettua il pedone sull'effettiva possibilità di percorrere la tratta a piedi oppure se gli è necessario usare i mezzi pubblici.

Può essere considerato social anche l'artefatto *Initializer* che fornisce agli agenti la conoscenza di base che gli agenti necessitano: questo artefatto assegna un identificativo univoco ad ogni agente e in base alla tipologia di agente fornisce tutte le informazioni che l'agente sfrutterà durante la simulazione. Principalmente, la conoscenza di base di ogni agente consiste del set di posizioni in cui si trovano i luoghi che intende visitare e negli orari programmati per le proprie attività.

Capitolo 6

Workplan

Le attività principali per la realizzazione del progetto sono identificate nella seguente WBS

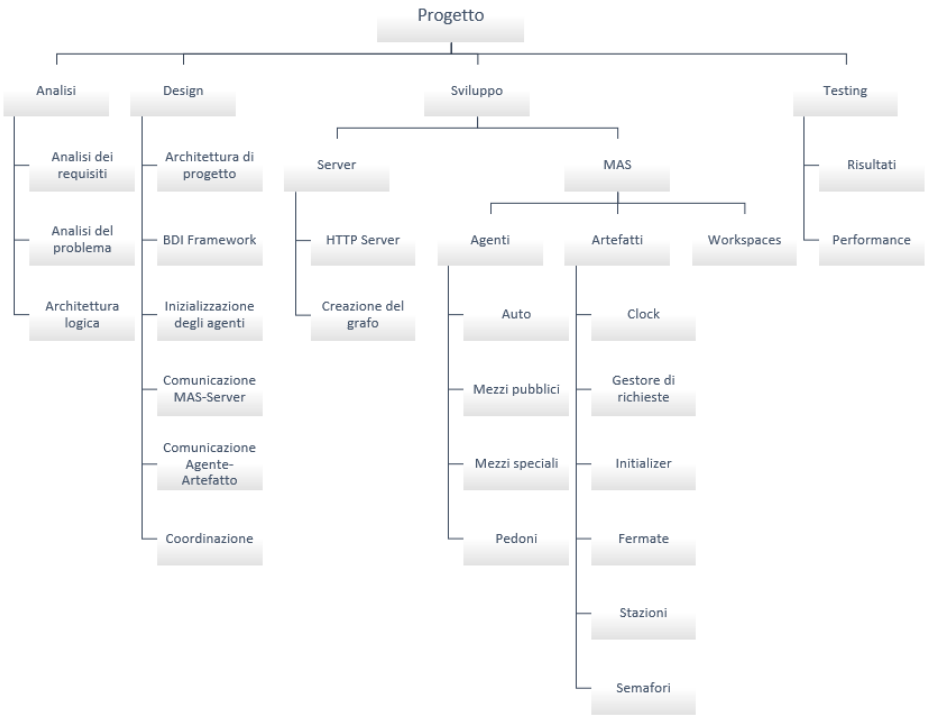

Figura 6.1: WBS

Capitolo 7

Implementazione

In fase di implementazione è stato realizzato quanto definito nella fase di design. Il Server è un'applicazione Java con un HTTP Server in ascolto sulla porta 8000, utilizzando la libreria `com.sun.net.httpserver`. Gli agenti in questo caso si comportano come un tipico HTTP Client e tramite il metodo statico *POST* fornito dalla classe *HttpUtils* inviano i messaggi di richiesta di percorso. I messaggi tra gli agenti e il Server vengono scambiati utilizzando il formato JSON che consente di incapsulare e strutturare i dati all'interno dei messaggi.

HttpUtils

```
package utils.http;

import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.HttpURLConnection;
import java.net.URL;
```

```
public class HttpUtils {

   private static final String REQUEST =
       "http://127.0.0.1:8000";

   public static String POST(String msg) throws IOException {
      String response = "";
      URL url = new URL(REQUEST);
      HttpURLConnection urlConnection = (HttpURLConnection)
          url.openConnection();
      urlConnection.setDoOutput(true);

      OutputStreamWriter wr = new
          OutputStreamWriter(urlConnection.getOutputStream());
      wr.write(msg);
      wr.flush();
      wr.close();

      InputStream in = new
          BufferedInputStream(urlConnection.getInputStream());
      response = readStream(in);
      in.close();
      urlConnection.disconnect();

      return response;
   }

   private static String readStream(InputStream inputStream)
       throws IOException {
      BufferedReader reader = new BufferedReader(new
          InputStreamReader(inputStream));
      StringBuilder sb = new StringBuilder();
      String content;
      while ((content = reader.readLine()) != null) {
         sb.append(content);
```

```
        }
        return sb.toString();
    }

}
```

Il MAS è stato realizzato tramite una piattaforma *JaCaMo* con artefatti e agenti con una specifica organizzazione. La piattaforma permette di definire diversi *workspaces*, intesi come set di agenti e artefatti che interagiscono tra loro. Nel file di configurazione JaCaMo vengono creati gli agenti a cui viene assegnato un nome e vengono definiti i *beliefs* iniziali e gli artefatti che l'agente ha bisogno di conoscere (*focus*). All'interno di un *workspace* è possibile creare gli artefatti.

7.1 Agenti

Gli agenti sono agenti Jason basati sul framework BDI descritto nella sezione 5.4. Tutti gli agenti corrispondenti a veicoli di privati prevedono una fase di inizializzazione in cui un artefatto apposito li inizializza assegnandogli una posizione (coincidente con la casa dell'agente) e le informazioni utili per le attività (i luoghi e gli orari delle attività) che l'agente aggiunge ai propri *beliefs*. I *plans* dell'agente indicano il comportamento che l'agente deve tenere durante la simulazione. I seguenti esempi di agenti mostrano la struttura tipica degli agenti Jason.

Gli agenti inoltre efettuano un'operazione interna (*leaveForPlace*) che gli consente di capire quando è il momento giusto per partire in anticipo e raggiungere un luogo entro un determinato orario. Ad esempio, l'agente *Business Man* inizia il proprio turno di lavoro alle 09:00. Con questa operazione l'agente controlla che – aggiungendo il tempo stimato per raggiungere il luogo all'orario attuale – non sfori l'orario limite entro il quale presentarsi al lavoro.

7.1.1 Leave For Place

```
package jia;

import model.Model;
import jason.asSemantics.*;
import jason.asSyntax.*;

public class leaveForPlace extends DefaultInternalAction {

   private static final long serialVersionUID = 1L;
   private static final int CELL_SECONDS = 10;
   private static final int THRESHOLD = 15;

   @Override
   public Object execute(TransitionSystem ts, Unifier un,
       Term[] args) throws Exception {
      Model model = Model.getInstance();
      String start = args[0].toString();
      String startNode = start.replaceAll(String.valueOf('"'),
          "");
      String end = args[1].toString();
      String endNode = end.replaceAll(String.valueOf('"'), "");
      String hour =
          args[2].toString().replaceAll(String.valueOf('"'), "");
      int h = Integer.parseInt(hour);
      String min =
          args[3].toString().replaceAll(String.valueOf('"'), "");
      int m = Integer.parseInt(min);
      int distance = getDistanceInMinutes(startNode, endNode);
      int totMinsModel = model.getHour() * 60 +
          model.getMinute();
      int totMinsPlace = h * 60 + m;
      int timeGap = Math.abs(totMinsModel - totMinsPlace);
      if (totMinsModel > totMinsPlace) {
```

```
            return false;
        } else if (totMinsModel + distance > totMinsPlace +
            THRESHOLD) {
            return false;
        }
        if (Math.abs(timeGap - distance) <= THRESHOLD) {
            return true;
        } else
            return false;
    }

    private int getDistanceInMinutes(String cell1, String cell2)
        {
        String[] c1 = cell1.split("-");
        int lat1 = Integer.parseInt(c1[0]);
        int lng1 = Integer.parseInt(c1[1]);
        String[] c2 = cell2.split("-");
        int lat2 = Integer.parseInt(c2[0]);
        int lng2 = Integer.parseInt(c2[1]);
        return (Math.abs(lat1 - lat2) + Math.abs(lng1 - lng2)) *
            CELL_SECONDS / 60;
    }
}
```

7.1.2 Business Man

L'agente *Business Man* viene a conoscenza dei luoghi e dell'orario di lavoro tramite l'operazione `initBusinessman`. Quello che segue è la lista dei *plan* che costituiscono tutto ciò l'agente svolge durante la simulazione. Gran parte dei *plan* vengono *triggerati* dalla percezione del `tick(D,S,M,Sec)` emesso dall'artefatto *Clock*, ovvero il modo in cui l'agente prende cognizione del tempo.

```
/* Beliefs */
```

```
stadiumStartSchedule(14,30).
stadiumEndSchedule(17,00).

/* Goals */
!init.
!getLocations.

/* Plans */

+!init<- initialize.
+!getLocations <- initBusinessMan.

+idIs(I) <- .print("> My Name is ",I); +id(I).

+location(L) <- .print("> My Home is at: ",L); +home(L);
   +loc(L).

+officeAt(L) <- .print("> Office at:",L); +office(L).

+officeStartMorningAt(H,M) <- .print("> Starting Morning
   at:",H,":",M);
                    +officeStartMorning(H,M).

+officeEndMorningAt(H,M) <- .print("> Ending Morning
   at:",H,":",M);
                  +officeEndMorning(H,M).

+officeStartNoonAt(H,M) <- .print("> Starting Noon
   at:",H,":",M);
                  +officeStartNoon(H,M).

+officeEndNoonAt(H,M) <- .print("> Ending Noon at:",H,":",M);
                +officeEndNoon(H,M).

+restaurantAt(L) <- .print("> Restaurant at: ",L);
```

```
            +restaurant(L).

+stadiumAt(L) <- .print("> Stadium at:",L);
            +stadium(L);
            start.

+tick(D,H,M,Sec) : stop(S) & S > 0
<-    -+stop(S-1).

+tick(D,H,M,Sec) : stop(S) & S == 0
<-    -stop(_);
   +moving.

+tick(D,H,M,Sec) : D > 5 & not weekend
<-    +weekend.

+tick(D,H,M,Sec) : not weekend & loc(L) & office(O) & not L ==
    O & id(I) & not stop(_) & Sec = 0 &
           officeStartMorning(Hour,Minute) &
               jia.leaveForPlace(L,O,Hour,Minute) & not working
<- .print("It is ",H,":",M," of day ",D);
   .print("> GOING TO OFFICE (MORNING) - I'm at",L,"Dest
      is:",O);
   requestPath(L,O,I,H,M);
   +dest(O);
   +moving;
   +working.

+tick(D,H,M,Sec) : not weekend & id(I) & loc(L) & restaurant(R)
    & officeEndMorning(Hour,Minute)
               & H==Hour & M==Minute & Sec==0
<- -working;
   .print("> Going to restaurant at:",R);
   requestPath(L,R,I,H,M);
   +dest(R);
```

```
   +moving.

+tick(D,H,M,Sec) : not weekend & loc(L) & office(O) & not L ==
    O & id(I) & not stop(_) & Sec == 0 &
            officeStartNoon(Hour,Minute) &
                jia.leaveForPlace(L,O,Hour,Minute) & not working
<-  .print("It is ",H,":",M," of day ",D);
   .print("> GOING TO OFFICE (AFTERNOON) - I'm at",L,"Dest
      is:",O);
   requestPath(L,O,I,H,M);
   +dest(O);
   +moving;
   +working.

+tick(D,H,M,Sec) : not weekend & id(I) & loc(L) & home(Home) &
    officeEndNoon(Hour,Minute)
               & H==Hour & M==Minute & Sec==0 & working
<-    -working;
   .print("It is ",H,":",M,":",Sec," of day ",D);
   .print("> Going home:");
   requestPath(L,Home,I,H,M);
   +dest(Home);
   +moving.

+tick(D,H,M,Sec) : stadiumStartSchedule(Hour,Minute) & H=Hour &
    M=Minute & Sec = 0
               & id(I) & loc(L) & stadium(S) & D = 7
<- .print("It is ",H,":",M,":",Sec," of day ",D);
   .print("> Going to the Stadium");
   requestPath(L,S,I,H,M);
   +dest(S);
   +moving.


+tick(D,H,M,Sec) : stadiumEndSchedule(Hour,Minute) & H=Hour &
```

```
    M=Minute & Sec = 0
              & id(I) & loc(L) & home(Home) & D = 7
<- .print("It is ",H,":",M,":",Sec," of day ",D);
   .print("> Going home from the stadium");
   requestPath(L,Home,I,H,M);
   +dest(Home);
   +moving.

+nextNodeIs(N,T)
<-    -+nextNode(N,T).

+tick(D,H,M,Sec) : moving & nextNode(N,T) & T > 0
   <- -+nextNode(N,T-1).

+tick(D,H,M,Sec) : moving & nextNode(N,T) & dest(Dest) & not
    N==Dest & T == 0
   <- checkSemaphore(N);
      next.

+arriving : dest(D) & hour(H) & minute(M) & second(S)
<-  .print("Arriving at dest",D," ",H,":",M,":",S);
   -moving;
   -dest(D);
   -+loc(D) .

+tick(D,H,M,Sec) : moving & nextNode(N,T) & dest(Dest) &
    N==Dest & T == 0
   <- .print("Arrived at destination: ", Dest).

+stop(S) <- -moving;
        +stop(S).

{ include("$jacamoJar/templates/common-cartago.asl") }
{ include("$jacamoJar/templates/common-moise.asl") }
```

7.1.3 Bus

L'agente *Bus* viene inizializzato con gli orari delle proprie corse e la lista di fermate da effettuare lungo il tragitto. Mentre si muove lungo la tratta, quando giunge in un punto significativo `P` della mappa effettua tre controlli: `checkSemaphore(P)` per controllare se è presente un semaforo, `checkBusStop(P)` per controllare se in quel punto è presente una fermata appartenente alla propria tratta (e nel caso caricare i passeggeri che intendono salire) e `dropPassengers(passengers,P)` per far scendere i passeggeri che intendono scendere in quel punto.

```
/* Initial beliefs and rules */
passengers(0).

/* Initial goals */

!init.
!getLocations.

/* Plans */

+!init <- initialize.
+!getLocations <- initBus.

+idIs(I)
<- .print("> My Name is ",I);
   +id(I).

+startTimeAt(H,M)
<-    .print("> Starting at:",H,":",M);
   +startTime(H,M).

+endTimeAt(H,M)
<-    .print("> Ending at:",H,":",M);
   +endTime(H,M).
```

```
+startBusStopAt(L)
<-    .print("> Starting from: ",L);
   +startBusStop(L);
   +loc(L).

+endBusStopAt(L,D)
<-    .print("> Ending to: ",L);
   +endBusStop(L);
   +destinations(D);
   start.

+nextNodeIs(N,T)
<-    -+nextNode(N,T).

+updatePassengers(N,L) : passengers(P)
<-    -moving;
   -+passengers(N+P);
   +taking(N);
   -+destinations(L).

+tick(D,H,M,Sec) : D > 5
<-    +weekend.

+tick(D,H,M,Sec) : not weekend & endBusStop(Dest) & id(I) & Sec
    == 0 & not stop(_) & loc(L)
              & startTime(Hour,Minute) & H = Hour & M = Minute
                  & not moving
<- .print("> Starting tour at:",H,":",M);
   requestPath(L,Dest,I,H,M);
   +dest(Dest);
   +moving.

+tick(D,H,M,Sec) : Sec = 0 & moving & nextNode(N,T) &
    dest(Dest) & not N==Dest
```

```
            & T == 0 & passengers(P) & destinations(L)
<- .print("> Moving towards: ",N);
   .print("> Passengers on board: ",P);
   checkSemaphore(N);
   checkBusStop(N);
   dropPassengers(L,N);
   next.

+tick(D,H,M,Sec) : Sec = 0 & moving & nextNode(N,T) &
    dest(Dest) & N==Dest & T == 0
            & passengers(P) & startBusStop(B) & not done
<-    .print("Arrived at ",Dest," with ",P," passengers");
   -moving;
   -dest(Dest);
   -+loc(Dest);
   +done.

+tick(D,H,M,Sec) : not weekend & startBusStop(Dest) & id(I) &
    Sec == 0 & not stop(_) & loc(L)
             & not moving & done
<-    .print("Going back ");
   .print("> Starting tour at:",H,":",M);
   requestPath(L,Dest,I,H,M);
   +dest(Dest);
   +moving.


+tick(D,H,M,Sec) : moving & nextNode(N,T) & T > 0
<- -+nextNode(N,T-1).

+arriving : dest(D) & hour(H) & minute(M) & second(S)
<- .print("Arriving at dest",D," ",H,":",M,":",S);
   -moving;
   -dest(D);
   -+loc(D) .
```

```
+tick(D,H,M,Sec) : stop(S) & S > 0
<-    -+stop(S-1).

+tick(D,H,M,Sec) : taking(N) & N > 0
<-    .print("Taking ",N, " passengers");
   -+taking(N-3).

+tick(D,H,M,Sec) : taking(N) & N <= 0
<- -taking(_);
   +moving.

+tick(D,H,M,Sec) : stop(S) & S == 0
<-    -stop(_);
   +moving.

+stop(S)
<- -moving;
   +stop(S).



{ include("$jacamoJar/templates/common-cartago.asl") }
{ include("$jacamoJar/templates/common-moise.asl") }
```

7.1.4 Pedestrian

L'agente *Pedestrian* effettua il controllo `checkBus` per capire se recarsi alla *Bus Stop* più vicina e prendere i mezzi pubblici. In questo caso, una volta giunto alla *Bus Stop* comunica quale *Bus* vuole prendere e qual è la sua destinazione con l'operazione `takeBus(bus,destination)`

```
/* Initial beliefs and rules */
!init.
!getLocations.
```

```
/* Initial goals */

+!init <- initialize.
+!getLocations <- initPedestrian.

+idIs(I)
<-    .print("> My Name is ",I);
   +id(I).

+homeAt(L)
<-    .print("> My Home is at: ",L);
   +home(L);
   +loc(L).

+officeAt(L) : home(H)
<-    .print("> My Office is at:",L);
   +office(L);
   checkBus(H,L).

+startTimeAt(H,M)
<-    .print("> Starting at:",H,":",M);
   +startTime(H,M).

+endTimeAt(H,M)
<-    .print("> Ending at:",H,":",M);
   +endTime(H,M);
   start.

+takeTheBus(N,S)
<-    .print("> I will take the bus");
   +bus(N);
   +busStop(S).

+tick(D,H,M,Sec) : D > 5
<-    +weekend.
```

```
+tick(D,H,M,Sec) : not weekend & loc(L) & office(O) & not L ==
    O & id(I) & Sec = 0 & bus(N) &
            startTime(Hour,Minute) & (H*60 + M +30) == (Hour*60
                + Minute) & not working & busStop(BS)
<-.print("It is ",H,":",M," of day ",D);
   .print("> GOING TO Take the bus - I'm at",L,"Dest is:",O);
   takeBus(N,BS,O);
   +moving;
   +working.

+tick(D,H,M,Sec) : not weekend & home(Home) & office(O) & id(I)
    & Sec = 0 & bus(N) &
            endTime(Hour,Minute) & H == Hour & M == Minute &
                working & busStop(O)
<- .print("It is ",H,":",M," of day ",D);
   .print("> GOING Home - I'm at",O,"Dest is:",Home);
   takeBus(N,O,Home);
   +moving;
   +working.


{ include("$jacamoJar/templates/common-cartago.asl") }
{ include("$jacamoJar/templates/common-moise.asl") }
```

7.1.5 Police

L'agente *Police* - una volta percepito lo stato di emergenza - reagisce richiedendo il percorso tra la propria posizione e la posizione in cui si è verificata l'emergenza e bloccando i semafori che si trovano su tale percorso con l'operazione `blockSemaphores(S)`. Una volta effettuata questa operazione si avvia verso il luogo dell'emergenza e una volta giunto in quel punto sblocca i semafori con l'operazione `unblockSemaphores(S)`.

```
/* Goals */

!init.
!getLocations.

/* Plans */

+!init
<-    initialize.

+!getLocations
<-    initPolice.

+idIs(I)
<- .print("> My Name is ",I);
   +id(I).

+location(L)
<- .print("> My District is at: ",L);
   +district(L);
   +loc(L).

+semaphoresAt(S)
<- .print("> Getting list of semaphores");
   +semaphores(S);
   start.

+nextNodeIs(N,T)
<- +nextNode(N,T).

+tick(D,H,M,Sec) : stop(S) & S > 0 & not blockedSem(B)
<-    -+stop(S-1).

+tick(D,H,M,Sec) : stop(S) & S == 0 & not blockedSem(B)
<-    -stop(_);
```

```
   +moving.

+tick(D,H,M,Sec) : D > 5 & not weekend
<-    +weekend.

+emergencyAt(E,D,H,M,Sec) : loc(L) & id(I) & not L == E & not
    handling
                & semaphores(S) & not dest(_) & not blocking
<-    +emergency(E);
   .print("> Emergency: Police is at ",L," and emergency is at:
       ",E);
   getSemaphoresToBlock(L,E,I,H,M,S);
   +blocking.

+semaphoresToBlock(S) : not blocked
<- .print("Blocking semaphores");
   blockSemaphores(S); +blocked.

+blockedSem
<- +blockedSemaphores.

+tick(D,H,M,Sec) : blockedSemaphores & loc(L) & id(I) &
    emergency(E) & not L == E
        & not handling & blocked & not dest(_) & not moving
<- requestPath(L,E,I,H,M);
   +dest(E);
   +moving;
   +handling.

+tick(D,H,M,Sec) : loc(L) & emergency(E) & L==E & handling
<- unblockSemaphores;
   -blocked;
   -handling.

+tick(D,H,M,Sec) : moving & nextNode(N,T) & T > 0
```

```
<- -+nextNode(N,T-1).

+tick(D,H,M,Sec) : moving & nextNode(N,T) & dest(Dest) & not
    N==Dest & T == 0 & blocked
<- next.

+arriving : dest(D) & hour(H) & minute(M) & second(S)
<-    .print("Arriving at dest",D," ",H,":",M,":",S);
   -moving;
   -dest(D);
   -+loc(D) .

+tick(D,H,M,Sec) : moving & nextNode(N,T) & dest(Dest) &
    N==Dest & T == 0
<- .print("Arrived at destination: ", Dest).

+stop(S)
<- -moving;
   +stop(S).

{ include("$jacamoJar/templates/common-cartago.asl") }
{ include("$jacamoJar/templates/common-moise.asl") }
```

7.2 Artefatti

Gli artefatti sono estensioni della classe Java *Artifact* e permettono di gestire le operazioni effettuate dagli agenti. Quando un metodo della classe è preceduto da `@OPERATION` significa che corrisponde a una possibile operazione fornita dall'artefatto agli agenti che si trovano nel suo *workspace*. Gli artefatti fanno riferimento a un'istanza di Model (ottenuta tramite il pattern *Singleton*) nella quale sono contenute le informazioni relative alle coordinate delle celle della mappa in cui sono posizionati i punti di interesse del sistema.

7.2.1 Semaphore

L'artefatto che modella il semaforo offre agli agenti la funzionalità di base `checkSemaphore` con la quale essi si rendono conto se si trovano nei pressi di un semaforo e se lo stato del semaforo è impostato a 0 l'artefatto effettua una `signal` con cui avverte l'agente di fermarsi e di aspettare un tempo predefinito. L'artefatto inoltre prevede l'operazione `blockSemaphores(S)` che consente ai veicoli speciali di bloccare un set `S` di semafori per agevolare il proprio transito nel caso di emergenza. È prevista inoltre un'operazione interna `switchStatus` con un ciclo che cambia lo stato di ogni semaforo una volta trascorso un lasso di tempo predefinito. Chiaramente, se un veicolo speciale ha bloccato determinati semafori, essi non saranno inficiati dal cambiamento di stato fino a che lo stesso veicolo speciale (una volta giunto a destinazione) avrà *sbloccato* tali semafori.

```
package artifacts;

import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Random;

import cartago.*;
import model.Model;

public class Semaphore extends Artifact {
   final static long SWITCH_TIME = 1000;
   private Model model = Model.getInstance();
   private List<String> semaphores;
   private Map<String, Integer> status;
   boolean block = false;
   boolean flag = false;

   void init() {
```

```
    this.semaphores = this.model.getSemaphores();
    this.status = new HashMap<String, Integer>();
    for (String s : this.semaphores) {
        Random r = new Random();
        status.put(s, r.nextInt(2));
    }
    execInternalOp("switchStatus");
}

@OPERATION
void checkSemaphore(String s) {
    AgentId agId = getCurrentOpAgentId();
    if (s != null) {
        if (this.semaphores.contains(s) && this.status.get(s)
            == 0) {
            System.out.println("> Agent " + agId + " on
                semaphore " + s);
            signal(agId, "stop", 10);
        }
    }
}

@INTERNAL_OPERATION
void switchStatus() {
    while (!block) {
        await_time(SWITCH_TIME);
        for (String s : status.keySet()) {
            if (status.get(s) == 0) {
                status.put(s, 1);
            } else {
                status.put(s, 0);
            }
        }
        // System.out.println("> SEMAPHORES NON
            BLOCKED:"+status.toString());
```

```
    }
}

@INTERNAL_OPERATION
void switchStatusBlocked(List<String> blocked) {
    while (block) {

        await_time(SWITCH_TIME);
        for (String s : this.status.keySet()) {
            if (!blocked.contains(s)) {
                if (status.get(s) == 0) {
                    status.put(s, 1);
                } else {
                    status.put(s, 0);
                }
            }
        }

        // System.out.println("> SEMAPHORES
            BLOCKED:"+status.toString());
        signal("blockedSem");
    }
}

@OPERATION
void blockSemaphores(List<String> toBlock) {
    this.block = true;
    for (String s : this.status.keySet()) {
        if (toBlock.contains(s)) {
            this.status.put(s, 0);
        }
    }
    execInternalOp("switchStatusBlocked", toBlock);
}
```

```
    @OPERATION
    void unblockSemaphores() {
        this.block = false;
        execInternalOp("switchStatus");
    }
}
```

7.2.2 Bus Stop

L'artefatto Bus Stop gestisce la presenza di stazioni e fermate all'interno della mappa. Le fermate sono distribuite nella mappa e ogni mezzo pubblico possiede un percorso con un set di fermate in cui effettua la sosta e invoca le operazioni `checkBusStop` per fare salire i passeggeri presenti in quella fermata (con la relativa lista di destinazioni) e `dropPassengers` per far scendere i passeggeri già presenti sul bus che avevano selezionato quella fermata come destinazione. Un'altra operazione prevista dall'artefatto è `takeBus`, invocata dai pedoni che non intendono percorrere il proprio percorso a piedi e si sono *registrati* presso quella specifica fermata.

```
package artifacts;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import cartago.*;
import model.Model;
import model.Pair;

public class BusStop extends Artifact {
    private Model model = Model.getInstance();
    // Pair: first is bus number, second is destinations
    private List<Pair<String, String>> destinations;
```

```
// Map: 1st is number of passengers, 2nd is coordinates of
    the busstop
private Map<String, Integer> busStops;

void init() {
   this.destinations = new ArrayList<Pair<String, String>>();
   this.busStops = new HashMap<String, Integer>();
   for (String s : model.getBusStops()) {
      this.busStops.put(s, 0);
   }
}

@OPERATION
void checkBusStop(String s) {
   if (this.busStops.containsKey(s)) {
      AgentId agId = this.getCurrentOpAgentId();
      List<String> dest =
          this.getDestinations(agId.toString());
      System.out.println(">>BUSSTOP: destination list is " +
          dest.toString());
      int nPass = this.busStops.get(s);
      System.out.println(">>BUSSTOP: npass at" + s + " is" +
          nPass);
      if (nPass > 0) {
         System.out.println(">>BUSSTOP: Taking " + nPass + "
             from: " + s);
         signal(agId, "updatePassengers", nPass, dest);
         int oldValue = this.busStops.get(s);
         this.busStops.replace(s, oldValue, oldValue - nPass);

      }
   }
}

@OPERATION
```

```
void takeBus(String bus, String origin, String dest) {
   int oldValue = this.busStops.get(origin) + 1;
   this.busStops.remove(origin);
   this.busStops.put(origin, oldValue);
   for (String s : this.busStops.keySet()) {
      int passengers = this.busStops.get(s);
      System.out.println("> Busstop " + s + " have " +
          passengers + " passengers");
   }
   System.out.println(
        ">>> I will take bus: " + bus + " at: " + origin + "
            / There are: " + oldValue++ + "passengers");
   this.destinations.add(new Pair<String, String>(bus,
       dest));
}

@OPERATION
void dropPassengers(List<String> destinations, String pos) {
   List<String> newDestinations = new ArrayList<String>();
   newDestinations = destinations;
   System.out.println(newDestinations.size() + " is the
       size, MyPos is: " + pos);
   boolean cond = false;
   for (Pair<String, String> p : this.destinations) {
      if (p.getSecond().equals(pos))
         cond = true;
   }
   if (!newDestinations.isEmpty() && cond) {
      AgentId agId = this.getCurrentOpAgentId();
      int count = 0;
      List<String> toRemove = new ArrayList<String>();
      for (String d : newDestinations) {
         System.out.println("Destination:" + d + ", myPos:" +
             pos);
         if (d.equals(pos)) {
```

```
                toRemove.add(d);
                count++;
            }
        }
        newDestinations.removeAll(toRemove);
        int oldValue = this.busStops.get(pos);
        this.busStops.remove(pos, oldValue);
        oldValue -= count;
        this.busStops.put(pos, oldValue);
        if (count > 0) {
            System.out.println("Bus " + agId.toString() + "
                dropped " + count + " passengers at: " + pos);
            signal(agId, "updatePassengers", oldValue,
                newDestinations);
        }
    }
  }

  private List<String> getDestinations(String bus) {
    List<String> dest = new ArrayList<String>();
    for (Pair<String, String> p : destinations) {
        if (p.getFirst().equals(bus)) {
            dest.add((String) p.getSecond());
        }
    }
    return dest;
  }

}
```

7.2.3 Clock

La sincronizzazione a livello temporale degli agenti viene fornita dall'artefatto Clock che tramite l'operazione `work` aggiorna continuamente con

le proprietà osservabili (giorno, ora, minuto, secondo) utilizzando la primitiva `await-time` che sospende l'esecuzione dell'operazione per un determinato tempo ed effettua una signal (`tick`) che gli agenti percepiscono e in seguito alla quale effettuano le proprie attività.

```
package model;

import cartago.*;

public class Clock extends Artifact {
   private AgentId policeId;
   boolean working;
   final static long TICK_TIME = 10;
   private static final int STARTING_DAY = 1; // From 1
       (Monday) to 7 (Sunday)
   private static final int STARTING_HOUR = 6; // from 0 to 23
   private static final int STARTING_MINUTE = 0;
   private static final int STARTING_SECOND = 0;
   private int day, hour, minute, second;
   private Model m = Model.getInstance();

   void init() {
      working = false;
   }

   @OPERATION
   void start() {
      if (!working) {
         this.policeId = this.getCurrentOpAgentId();
         working = true;
         execInternalOp("work");
         this.day = STARTING_DAY;
         this.hour = STARTING_HOUR;
         this.minute = STARTING_MINUTE;
         this.second = STARTING_SECOND;
         this.m.updateClock(this.day, this.hour, this.minute,
```

```
            this.second);
        this.defineObsProperty("day", this.day);
        this.defineObsProperty("hour", this.hour);
        this.defineObsProperty("minute", this.minute);
        this.defineObsProperty("second", this.second);
    }
}

@OPERATION
void stop() {
    working = false;
}

@INTERNAL_OPERATION
void work() {
    while (working) {
        signal("tick", day, hour, minute, second);
        await_time(TICK_TIME);
        if (second < 49) {
            second += 10;
        } else {
            this.second = 0;
            if (minute < 59) {
                this.minute++;
                this.m.updateClock(this.day, this.hour,
                    this.minute, this.second);
            } else {
                this.minute = 0;
                if (hour < 23) {
                    if (hour == 11 && minute == 0) {
                        signal(policeId, "emergencyAt", "5-10",
                            this.day, this.hour, this.minute,
                            this.second);
                    }
                    this.hour++;
```

```
                System.out.println("> It's " + hour + ":00 of
                    day" + day);
                this.m.updateClock(this.day, this.hour,
                    this.minute, this.second);
            } else {
                if (day < 7) {
                    this.hour = 0;
                    this.day++;
                    this.m.updateClock(this.day, this.hour,
                        this.minute, this.second);
                } else {
                    working = false;
                }
            }
        }
        getObsProperty("day").updateValue(this.day);
        getObsProperty("hour").updateValue(this.hour);
        getObsProperty("minute").updateValue(this.minute);
        getObsProperty("second").updateValue(this.second);
    }
  }
}
```

7.2.4 Request Manager

La comunicazione degli agenti con il Server viene gestita da un'artefatto che prende in carico le richieste di percorso (`requestPath`). Oltre a restituire agli agenti il miglior percorso per la tratta che devono effettuare, l'artefatto *Request Manager* fornisce agli agenti altre due funzionalità in cui non è presente l'interazione con il Server, ma è comunque necessaria la conoscenza riguardo alla mappa (che l'artefatto ottiene tramite l'istanza di *Model*.

La prima operazione - `next` – viene invocata dagli agenti mentre stanno effettuando la tratta quando si trovano in una delle celle significative (gli incroci) e l'artefatto risponde con una `signal` in cui li mette al corrente del prossimo incrocio da raggiungere; in questo modo, una volta raggiunto ogni incrocio, l'agente controlla se in quel punto è presente un semaforo. La seconda operazione - `checkBus` – viene invocata solo dai pedoni che controllano se è necessario prendere i mezzi pubblici per recarsi a destinazione confrontando la distanza con una soglia predefinita.

```
package artifacts;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import cartago.AgentId;
import cartago.Artifact;
import cartago.OPERATION;
import interfaces.msg.IRequestBestPathMsg;
import model.Model;
import msg.RequestBestPathMsg;
import utils.http.HttpUtils;
import utils.json.JSONMessagingUtils;

public class RequestManager extends Artifact {
   public static final String REQUEST_BEST_PATH =
       "requestbestpath";
   private static final String BETWEEN_NODES = "betweennodes";
   private static final String PATH_LIST = "pathlist";
   private static final String PATH = "path";
   private static final int CELL_SECONDS = 10;
```

```
private static final int THRESHOLD = 300;
private AgentId policeId;
int index;
String res;

@OPERATION
void requestPath(String start, String end, String typo, Byte
    h, Byte m) throws Exception {
   Model model = Model.getInstance();
   this.index = 0;
   start = start.replace(String.valueOf('"'), "");
   end = end.replace(String.valueOf('"'), "");
   typo = typo.replace(String.valueOf('"'), "");
   String time = model.getDay() + ":" + h.byteValue();
   IRequestBestPathMsg msg = new
       RequestBestPathMsg(REQUEST_BEST_PATH, start, end,
       typo, time);
   String req =
       JSONMessagingUtils.getStringfromRequestBestPathMsg(msg);
   this.res = HttpUtils.POST(req);
   String s = this.getPaths(res).get(index);
   int t = this.getPathTimes(res).get(index);
   index++;
   System.out.println(this.getPaths(res).toString());
   System.out.println(this.getPathTimes(res).toString());
   signal("nextNodeIs", s, t);
}

@OPERATION
void getSemaphoresToBlock(String l, String e, String typo,
    Byte h, Byte m, List<String> sem)
      throws JSONException, IOException {
   Model model = Model.getInstance();
   this.policeId = this.getCurrentOpAgentId();
   this.index = 0;
```

```
    l = l.replace(String.valueOf('"'), "");
    e = e.replace(String.valueOf('"'), "");
    typo = typo.replace(String.valueOf('"'), "");
    String time = model.getDay() + ":" + h.byteValue();
    IRequestBestPathMsg msg = new
        RequestBestPathMsg(REQUEST_BEST_PATH, l, e, typo,
        time);
    String req =
        JSONMessagingUtils.getStringfromRequestBestPathMsg(msg);
    this.res = HttpUtils.POST(req);
    index++;
    List<String> newSem = new ArrayList<String>();
    for (String ss : this.getPaths(res)) {
       if (sem.contains(ss)) {
          newSem.add(ss);
       }
    }
    System.out.println("Semaphores:" + newSem.toString());
    AgentId agId = this.getCurrentOpAgentId();
    signal(agId, "semaphoresToBlock", newSem);
}

@OPERATION
void next() throws JSONException {
    if (index < this.getPathTimes(res).size()) {
       String s = this.getPaths(res).get(index);
       int t = this.getPathTimes(res).get(index);
       index++;
       signal("nextNodeIs", s, t);
    } else {
       signal("arriving");
    }
}

private int getTravelTime(String response) throws
```

```
     JSONException {
    JSONObject obj = new JSONObject(response);
    JSONArray paths = obj.getJSONArray(PATH_LIST);
    JSONArray betArray =
        paths.getJSONObject(0).getJSONArray(BETWEEN_NODES);
    int numCells = 0;
    for (int i = 0; i < betArray.length(); i++) {
       JSONArray a = betArray.getJSONArray(i);
       numCells += a.length() + 1;
    }
    return numCells * CELL_SECONDS / 6;
}

private List<String> getPaths(String response) throws
     JSONException {
    List<String> pathList = new ArrayList<String>();
    JSONObject obj = new JSONObject(response);
    JSONArray paths = obj.getJSONArray(PATH_LIST);
    JSONArray path =
        paths.getJSONObject(0).getJSONArray(PATH);
    for (int i = 0; i < path.length(); i++) {
       pathList.add(path.getJSONObject(i).getString("id"));
    }
    return pathList;
}

private List<Integer> getPathTimes(String response) throws
     JSONException {
    List<Integer> pathTimes = new ArrayList<Integer>();
    JSONObject obj = new JSONObject(response);
    JSONArray paths = obj.getJSONArray(PATH_LIST);
    JSONArray betArray =
        paths.getJSONObject(0).getJSONArray(BETWEEN_NODES);
    for (int j = 0; j < betArray.length(); j++) {
       JSONArray a = betArray.getJSONArray(j);
```

```
            pathTimes.add(a.length() * CELL_SECONDS / 6);
        }
        pathTimes.add(0, 0);
        return pathTimes;
    }

    @OPERATION
    void checkBus(String origin, String dest) {
        int o1 = Integer.parseInt(origin.split("-")[0]);
        int o2 = Integer.parseInt(origin.split("-")[1]);
        int d1 = Integer.parseInt(dest.split("-")[0]);
        int d2 = Integer.parseInt(dest.split("-")[1]);
        int diff1 = Math.abs(o1 - d1);
        int diff2 = Math.abs(o2 - d2);

        if (diff1 + diff2 > THRESHOLD) {
            Model model = Model.getInstance();
            String bs = model.getNearestBusStop(origin);
            System.out.println("Bus needed");
            signal(this.getCurrentOpAgentId(), "takeTheBus",
                "bus21", bs);
        }
    }

}
```

7.2.5 Initializer

L'artefatto *Initializer* è responsabile dell'inizializzazione degli agenti. Quando gli agenti invocano le rispettive operazioni di inizializzazione (`initBus`, `initPedestrian`, `initPolice` per citarne alcune) l'artefatto risponde con una serie di `signal` con tutte le informazioni utili per la simulazione (orari, posizioni dei luoghi e qualsiasi altro tipo di informazioni utili), a lui accessibili tramite il Model.

```
package model;

import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.ThreadLocalRandom;
import cartago.*;

public class Initializer extends Artifact {
   Model m = Model.getInstance();
   AgentId policeId;

   void init() {
      m = Model.getInstance();
   }

   @OPERATION
   void initialize() {
      AgentId agId = this.getCurrentOpAgentId();
      System.out.println(">>>>> " + agId.toString());
      String l = m.getRandomLocation();
      this.m.addAgent(agId.toString());
      this.m.addLocation(l);
      signal(agId, "idIs", agId.toString());
      signal(agId, "location", l);
   }

   @OPERATION
   void initBusinessMan() {
      AgentId agId = this.getCurrentOpAgentId();
      String office = m.getLocation("office").toString();
      String restaurant =
          m.getLocation("restaurant").toString();
      String stadium = m.getLocation("stadium").toString();
      Pair<Integer, Integer> startTimeMorning = new
```

```
        Pair<Integer, Integer>
       (ThreadLocalRandom.current().nextInt(0, 2) + 8,
          ThreadLocalRandom.current().nextInt(0, 2) * 30);
    Pair<Integer, Integer> endTimeMorning = new Pair<Integer,
        Integer>
       (ThreadLocalRandom.current().nextInt(0, 2) + 13,
          ThreadLocalRandom.current().nextInt(0, 2) * 30);
    Pair<Integer, Integer> startTimeNoon = new Pair<Integer,
        Integer>
       (ThreadLocalRandom.current().nextInt(0, 2) + 15,
          ThreadLocalRandom.current().nextInt(0, 2) * 30);
    Pair<Integer, Integer> endTimeNoon = new Pair<Integer,
        Integer>
       (ThreadLocalRandom.current().nextInt(0, 2) + 18,
          ThreadLocalRandom.current().nextInt(0, 2) * 30);
    signal(agId, "officeAt", office);
    signal(agId, "officeStartMorningAt",
        startTimeMorning.getFirst(),
        startTimeMorning.getSecond());
    signal(agId, "officeEndMorningAt",
        endTimeMorning.getFirst(), endTimeMorning.getSecond());
    signal(agId, "officeStartNoonAt",
        startTimeNoon.getFirst(), startTimeNoon.getSecond());
    signal(agId, "officeEndNoonAt", endTimeNoon.getFirst(),
        endTimeNoon.getSecond());
    signal(agId, "restaurantAt", restaurant);
    signal(agId, "stadiumAt", stadium);
}

@OPERATION
void initHousewife() {
    AgentId agId = this.getCurrentOpAgentId();
    String school = m.getLocation("school").toString();
    String supermarket =
        m.getLocation("supermarket").toString();
```

```
    String hairdresser =
        m.getLocation("hairdresser").toString();
    String beautysalon =
        m.getLocation("beautysalon").toString();
    String activity = m.getLocation("stadium").toString();
    String store = m.getLocation("store").toString();
    Pair<Integer, Integer> startTimeNoon = new Pair<Integer,
        Integer>(
          ThreadLocalRandom.current().nextInt(0, 2) + 15,
              ThreadLocalRandom.current().nextInt(0, 2) * 30);
    Pair<Integer, Integer> endTimeNoon = new Pair<Integer,
        Integer>(ThreadLocalRandom.current().nextInt(0, 2) +
        18,
          ThreadLocalRandom.current().nextInt(0, 2) * 30);
    signal(agId, "schoolAt", school);
    signal(agId, "supermarketAt", supermarket);
    signal(agId, "hairdresserAt", hairdresser);
    signal(agId, "beautySalonAt", beautysalon);
    signal(agId, "activityAt", activity);
    signal(agId, "storeAt", store);
    signal(agId, "activityStartAt", startTimeNoon.getFirst(),
        startTimeNoon.getSecond());
    signal(agId, "activityEndAt", endTimeNoon.getFirst(),
        endTimeNoon.getSecond());
}

@OPERATION
void initBus() {
    AgentId agId = this.getCurrentOpAgentId();
    List<String> busPath = m.getBusPath(agId.toString());
    String startBusStop = busPath.get(0);
    String endBusStop = busPath.get(busPath.size());
    Pair<Integer, Integer> startTime = new Pair<Integer,
        Integer>
    (ThreadLocalRandom.current().nextInt(0, 4) + 8,
```

```
        ThreadLocalRandom.current().nextInt(0, 2) * 30);
    Pair<Integer, Integer> endTime = new Pair<Integer,
        Integer>
    (ThreadLocalRandom.current().nextInt(0, 4) + 16,
        ThreadLocalRandom.current().nextInt(0, 2) * 30);
    List<String> destinations = new ArrayList<String>();
    signal(agId, "startTimeAt", startTime.getFirst(),
        startTime.getSecond());
    signal(agId, "endTimeAt", endTime.getFirst(),
        endTime.getSecond());
    signal(agId, "startBusStopAt", startBusStop);
    signal(agId, "endBusStopAt", endBusStop, destinations);
}

@OPERATION
void initPedestrian() {
    AgentId agId = this.getCurrentOpAgentId();
    String home = m.getRandomLocation();
    String office = m.getLocation("office").toString();
    Pair<Integer, Integer> startTime = new Pair<Integer,
        Integer>
    (ThreadLocalRandom.current().nextInt(0, 4) + 8,
        ThreadLocalRandom.current().nextInt(0, 2) * 30);
    Pair<Integer, Integer> endTime = new Pair<Integer,
        Integer>
    (ThreadLocalRandom.current().nextInt(0, 4) + 16,
        ThreadLocalRandom.current().nextInt(0, 2) * 30);
    signal(agId, "startTimeAt", startTime.getFirst(),
        startTime.getSecond());
    signal(agId, "endTimeAt", endTime.getFirst(),
        endTime.getSecond());
    signal(agId, "homeAt", home);
    signal(agId, "officeAt", office);

}
```

```
    @OPERATION
    void initPolice() {
        this.policeId = getCurrentOpAgentId();
        signal(policeId, "semaphoresAt", m.getSemaphores());
    }

}
```

Capitolo 8

Validazione

In questa fase viene descritto il testing del sistema e la validazione relativa alle funzionalità definite nei requisiti, analizzando l'output nei vari scenari della simulazione utlizzando la **MAS console**. La configurazione del sistema viene effettuata tramite un file con estensione .jcm dove si definisce il workspace del sistema, gli agenti e gli artefatti.

8.0.1 Inizializzazione

Nella fase di inizializzazione si nota come l'agente prenda conoscenza riguardo alle informazioni che gli servono per effettuare le azioni durante la simulazione. L'esempio in figura con un agente di tipo *Business Man* mostra come gli vengono forniti gli orari e le coordinate dei punti che gli interessano.

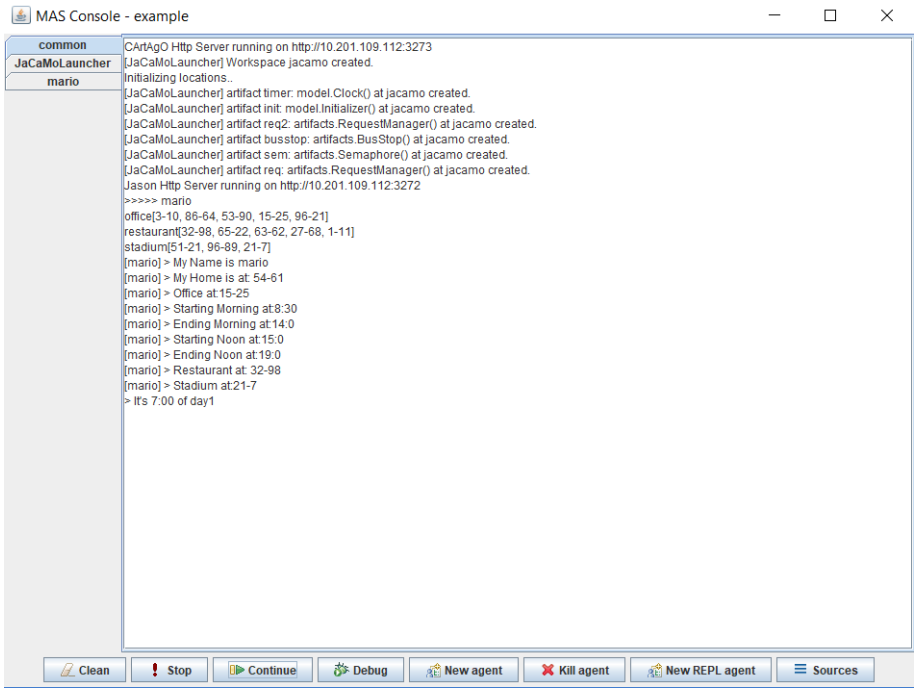

Figura 8.1: Inizializzazione dell'agente

8.0.2 Richiesta e svolgimento del percorso

Una volta ottenuta la conoscenza iniziale e giunto l'orario in cui deve effettuare il percorso, l'utente richiede il percorso e si muove verso la destinazione. Vengono visualizzati i punti della mappa intermedi tra l'origine e la destinazione. Quando l'utente si ritrova in prossimità di un punto in cui c'è un semaforo rosso, si ferma e aspetta che sia verde.

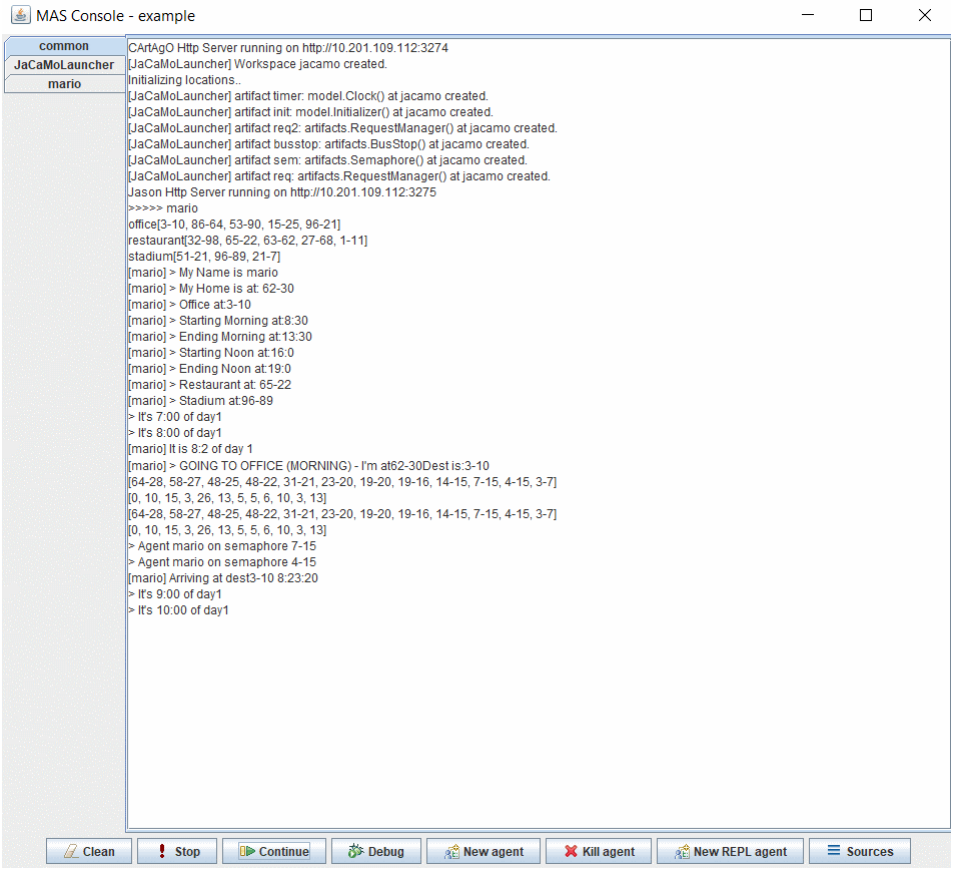

Figura 8.2: Richiesta e svolgimento del percorso

8.0.3 Svolgimento delle attività quotidiane

Lo svolgimento delle attività quotidiane degli agenti è scandito dal `tick` del Clock e in questo esempio si può osservare l'agente *Business Man* effettuare le richieste di percorso per recarsi a lavoro alla mattina, andare a ristorante per pranzo, tornare al lavoro al pomeriggio e poi ritornare a casa una volta finito il lavoro.

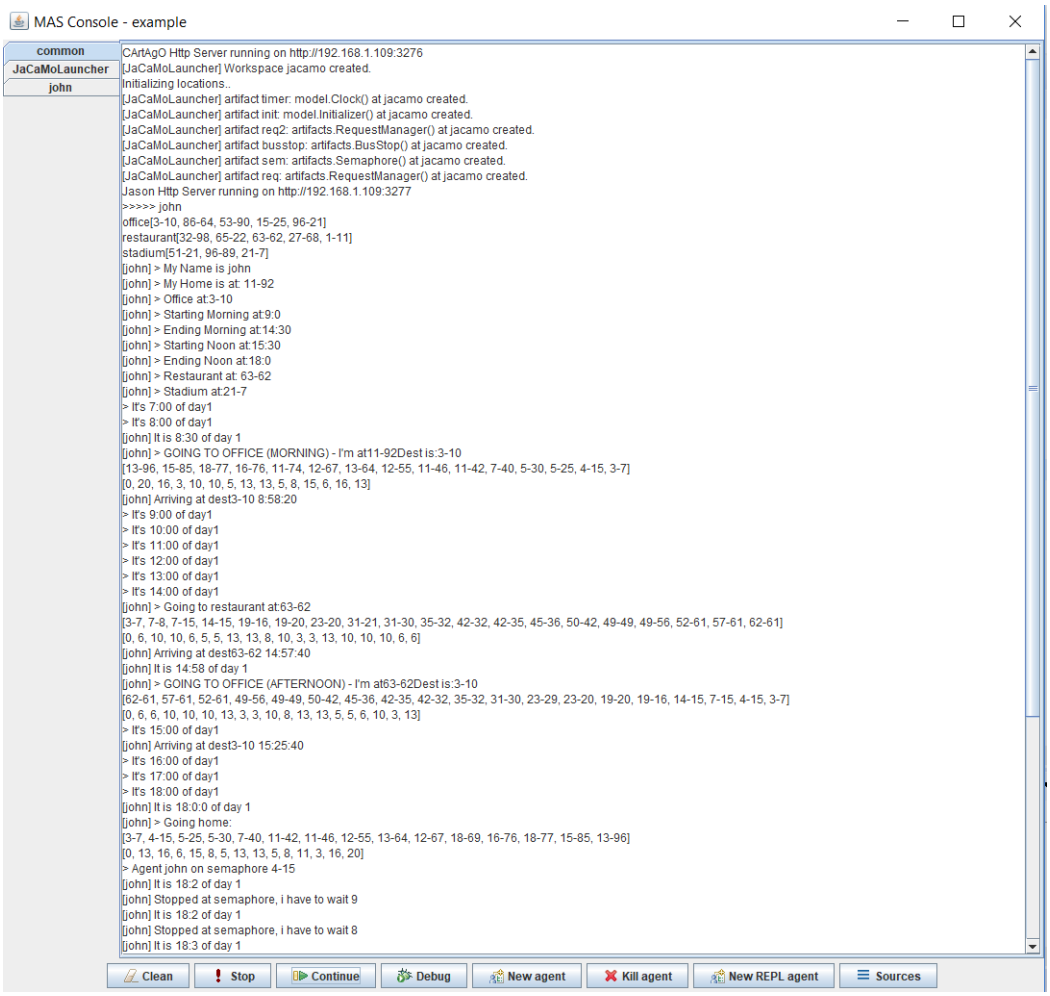

Figura 8.3: Svolgimento delle attività quotidiane

8.0.4 Gestione dell'emergenza da parte dei veicoli speciali

Quando un veicolo speciale (nell'esempio l'agente *Police*) riceve un segnale di emergenza si attiva richiedendo la lista di semafori interposti tra la posizione attuale del veicolo e il punto dell'emergenza e poi li blocca in modo da agevolare il proprio passaggio. Le *log* sullo stato dei semafori sono effettuate dall'artefatto *Semaphore* che mostra in output se alcuni semafori sono bloccati e lo stato di tutti i semafori (1 se verde, 0 se ros-

so). Una volta che il veicolo speciale raggiunge il luogo dell'emergenza sblocca i semafori ed essi tornano a funzionare regolarmente.

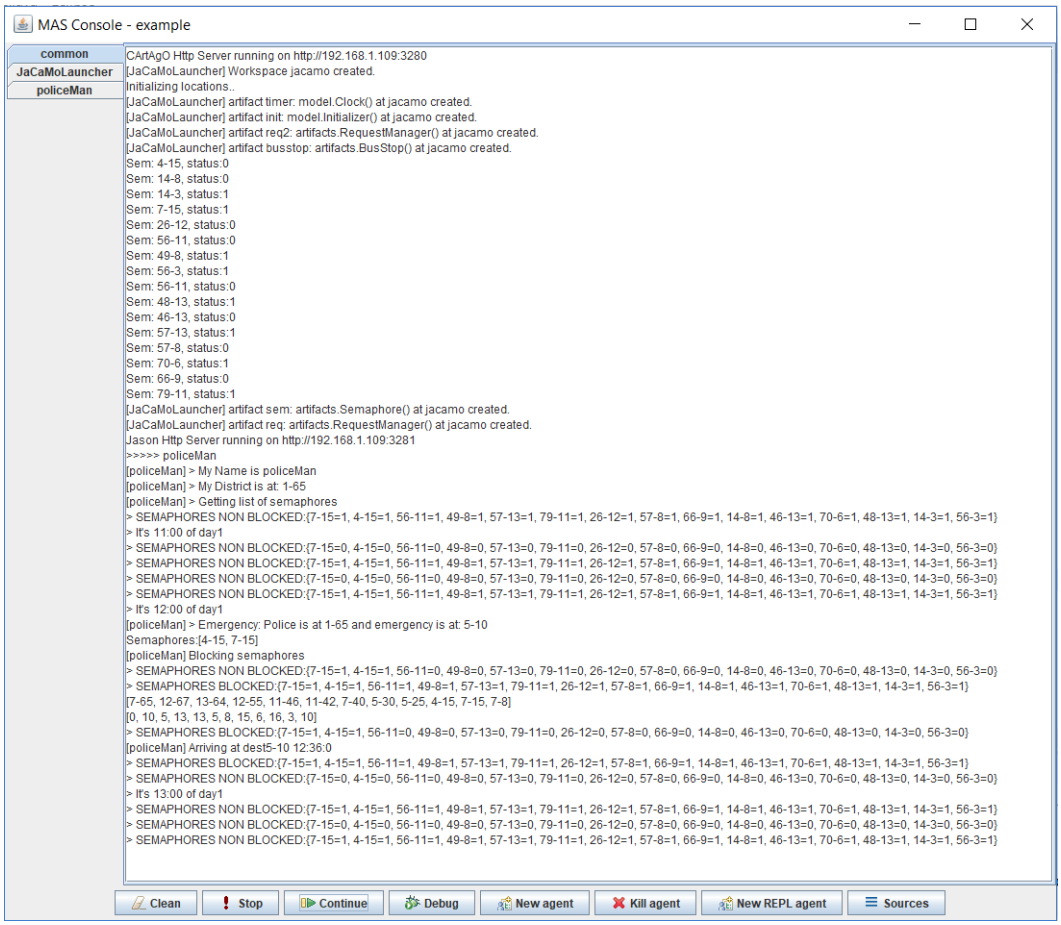

Figura 8.4: Gestione dell'emergenza da parte dei veicoli speciali

La simulazione è stata effettuata utilizzando un notebook portatile con un processore *Intel i7 Quadcore* con CPU a 2.40GHz e 8 Gb di RAM. In fase di configurazione è stata settata una mappa di 100x100 celle con i collegamenti tra le celle significative sul Server e sono stati lanciati 100 agenti di tipo *Business Man*, 100 di tipo *Housewife*, 100 di tipo *Universitary Student*, 10 di tipo *Bus*, 10 di tipo *Special Vehicle* e 200 di tipo *Pedestrian*. Al fine di misurare le prestazioni del Server è stata effettuata un'altra simulazione in cui vengono effettuate 500 richieste di percorso contemporanee da parte degli Agenti e si è notato che il tempo di attesa

della risposta cresce di pari passo con il numero di Agenti che effettuano la richiesta. Negli ultimi Agenti che hanno effettuato la richiesta il tempo massimo di attesa riscontrato è di circa 4 secondi, tempo di attesa che in una situazione estrema con un carico computazionale notevolmente superiore alla norma risulta accettabile.

Conclusioni

Il lavoro svolto offre alcuni interessanti spunti di riflessione riguardo alle possibili soluzioni attuabili nel campo della *smart mobility*. Se da un canto è vero che non si può prescindere da una serie di fattori abilitanti quali la connettività a banda larga, lo sfruttamento delle tecnologie mobile, embedded e sensoristica, è altrettanto vero che la mancanza di un approccio strutturato alla materia e il ventaglio enorme di possibilità offerto dall'evoluzione tecnologica e dal continuo emergere di nuovi framework rendono possibile e in certi casi necessaria l'esplorazione e l'applicazione di queste nuove tecnologie a supporto della *smart mobility*.

La *smart mobility* è un aspetto delle smart cities di grande attualità, che negli ultimi anni è passato in primo piano e che molto probabilmente rivoluzionerà il modo di intendere il traffico urbano nei decenni a venire (basti pensare all'impatto rivoluzionario che avranno i veicoli autonomi nell'immaginario comune). Il rischio però - e, da un punto di vista differente, anche la grande sfida - è che l'eterogeneità e la peculiarità di certi contesti urbani rispetto ad altri possano causare problemi di parametrizzazione e fare in modo che soluzioni di *smart mobility* ottime in un contesto non funzionino con altrettanta efficienza in un altro contesto.

Lo scopo di questo lavoro era quello di utilizzare le funzionalità di un framework ad agenti (ed artefatti) per simulare l'andamento del traffico in una determinata zona e il comportamento degli utenti della strada, in modo tale da fornire una sorta di middleware a supporto delle scelte

di chiunque decida di dedicare tempo e risorse a un progetto di *smart mobility*, in modo tale da stimare a priori il comportamento dei cittadini e condurre analisi sui dati raccolti nel corso della simulazione.

L'adozione del paradigma ad agenti autonomi e artefatti, oltre a modellare in maniera efficiente il dominio urbano per ricreare una vera e propria simulazione, ha dato vita a una *MAS-Based simulation*, integrazione che sicuramente rappresenta una soluzione interessante per quanto riguarda la *smart mobility*, dove gli utenti sono fortemente autonomi e gli elementi dell'infrastruttura stradale trovano una ideale corrispondenza con gli artefatti. Si può tranquillamente affermare che questo lavoro può rappresentare un punto di partenza (e non di arrivo) per la modellazione e la simulazione di una struttura urbana tramite un framework ad agenti e, parlando più in generale, rappresentare un piccolo mattoncino nella grande opera di costruzione e ampliamento delle smart cities.

8.1 Sviluppi Futuri

La moltitudine di tematiche differenti tra loro trattate e la vastità di gran parte di esse induce a considerare quelli che potrebbero essere gli sviluppi futuri e quali sono stati i punti in cui il lavoro della tesi si è arrestato, consapevolmente della grandezza della mole di lavoro aperta da ognuno di questi scenari.

In primo luogo, bisogna considerare le richieste del miglior percorso degli utenti all'interno di un quadro più ampio, ossia quello dei sistemi di navigazione intelligente. Esistono infatti al giorno d'oggi diversi software in grado di fornire il *routing* ai veicoli e agli utenti che ne fanno richiesta e identificare le congestioni del traffico in *real-time*. Una delle sfide più avvincenti nel futuro di questo ambito è sicuramente quella di basarsi su dati e richieste di percorso effettivi anticipando la formazione di una congestione.

L'effettivo successo di molti progetti di *smart mobility* dipenderà dalla possibilità di utilizzare e impiantare device fisici distribuiti in un contesto urbano. Questo punto richiede studi approfonditi riguardo a costi, performance e manutenibilità, i quali però non sono stati oggetto di questa tesi.

Le tipologie di agenti sviluppate nel sistema coprono solo una parte della vastissima gamma di profili diversi di cittadini che popolano le zone urbane. A tal proposito, sarebbe certamente utile, al fine di aumentare il livello di precisione ed accuratezza della simulazione, ampliare il numero delle tipologie di agenti del sistema e caratterizzare più dettagliatamente ogni tipologia allo scopo di ottenere una varietà di agenti più eterogenea possibile e aderente alla realtà di ogni dominio specifico in cui si intende effettuare la simulazione.

Allo stesso modo, il numero di artefatti presenti all'interno della simulazione può essere ampliato e ad esempio sarebbe interessante modellare altri elementi dell'arredo urbano, come la segnaletica verticale (i cartelli stradali) e quella orizziontale (gli attraversamenti pedonali) per accrescere il livello di dettaglio e di automazione della simulazione.

L'importanza dei mezzi pubblici e dei veicoli di emergenza e delle forze dell'ordine sono argomenti di cruciale importanza in ogni città, ed in particolare nelle smart cities. In questa tesi sono stati ipotizzati alcuni scenari di base sia per le interazioni dei primi con i pedoni, sia nella gestione di situazioni di emergenza nel caso dei secondi. È opportuno notare però che ogni contesto urbano presenta caratteristiche diverse dagli altri e quindi per poter adattare la simulazione a contesti diversi sarebbe necessario andare ad ampliare e ridefinire la struttura e il comportamento degli agenti che modellano i mezzi pubblici e i veicoli speciali, ad esempio introducendo la cooperazione con altri agenti in grado di segnalare situazioni di emergenza.

Ringraziamenti

Prima di tutto, ringrazio il mio relatore Andrea Omicini per la serietà, la professionalità e i preziosi consigli che mi hanno permesso di portare a termine questo lavoro. Ringrazio la mia famiglia, mia sorella Adria Nora, i miei zii Fabrizio e Franca, i miei nonni, Tiberio e Giovanni e soprattutto i miei genitori Federica e Giorgio che mi hanno guidato e supportato sin dal primo giorno di questo lungo percorso scolastico. Ringrazio i miei compagni di avventura, Matteo, Filippo, Cristian, Andrea, Alex, Davide, Luca e Lorenzo, senza i quali sarebbe stato tutto molto più difficile e noioso. Ringrazio la mia ragazza Martina, persona splendida che mi è stata affianco in questi anni. Ringrazio tutti i miei amici per aiutarmi a godermi i momenti belli e superare quelli più difficili.

Bibliografia

[1] R. G Hollands, *Will the real smart city please stand up?, in City, vol. 12, nº 3, 2008, pp. 303–320.*

[2] Komninos Nicos, *Intelligent cities: innovation, knowledge systems and digital spaces, London, Spon Press, 2002..*

[3] Sarwant Singh, *Smart Cities. A 1.5 Trillion Dollars Market Opportunity. Forbes. Retrieved 4 November 2014.*

[4] Clara Benevolo, Renata Paola Dameri, Beatrice D'Auria, *Smart Mobility in Smart City - Action Taxonomy, ICT Intensity and Public Benefits*

[5] W. Mobile, *Methodology and indicator calculation method for sustainable urban mobility, 2015,* `http://wbcsdservers.org/images/Mobilityindicators.pdf`

[6] B. Cohen, *The 10 smartest cities in europe," 2015,* `http://www.fastcoexist.com/3024721/the-10-smartest-cities-in-europ#2.`

[7] J. Goncalves, J. S. Goncalves, R. J. Rossetti, and C. Olaverri-Monreal, *Smartphone sensor platform to study traffic conditions and assess driving performance, in 17th International Conference on Intelligent Transportation Systems (ITSC). IEEE, 2014, pp. 2596–2601.*

[8] D. Pickeral, *Smarter transportation infrastructure means smarter choices, 2014,* `http://insights-on-business.com/government/`

smartertransportation-infrastructure-means-smarter-choices/
.

[9] C. Olaverri-Monreal, A. E. Hasan, J. Bulut, M. Korber, and K. Bengler, *Impact of in-vehicle displays location preferences on drivers' performance and gaze, IEEE Transactions on Intelligent Transportation Systems, vol. 15, no. 4, pp. 1770–1780, 2014.*

[10] P. Merdrignac, O. Shagdar, I. Ben Jemaa, and F. Nashashibi, *Study on perception and communication systems for safety of vulnerable road users, in 18th International Conference on Intelligent Transportation Systems (ITSC). IEEE, 2015, pp. 1876–1881*

[11] S. Colak, A. Lima, and M. C. Gonzalez, *Understanding congested travel in urban areas, Nature Communications, vol. 7, 2016.*

[12] Connected Smart Cities, *strategic initiatives* ***http://www.oascities.org/strategic-initiatives/.***

[13] P. Gomes, C. Olaverri-Monreal, and M. Ferreira, *Making vehicles transparent through v2v video streaming, IEEE Transactions on Intelligent Transportation Systems*

[14] Transportation Systems, *vol. 13, no. 2, pp. 930–938, 2012 NHTSA, "Traffic safety facts, research note."*

[15] M. Elbanhawi, M. Simic, and R. Jazar, *In the passenger seat: Investigating ride comfort measures in autonomous cars, IEEE Intelligent Transportation Systems Magazine, vol. 7, no. 3, pp. 4–17, 2015.*

[16] A. Hussein, F. Garcia, J. Armingol, and C. Olaverri-Monreal, *P2V and V2P Communication for Pedestrian Warning on the basis of Autonomous Vehicles, in 19th International Conference on Intelligent Transportation Systems (ITSC), in press. IEEE, 2016.*

[17] Wikipedia, *Intelligent transportation system.* ***https://it.wikipedia.org/wiki/Intelligent_transportation_system***

[18] A. Omicini, A. Ricci, M. Viroli, *"Give Agents their Artifacts": The AA Approach for Engineering Working Environments in MAS*

[19] Presidenza del Consiglio dei Ministri, *Habitat III Italy's National Report,* *http://www.governo.it/sites/governo.it/files/UN_HABITAT_III_ITALY_NATIONAL_REPORT_IT.pdf*

[20] United Nations, *World's population increasingly urban with more than half living in urban areas* *http://www.un.org/en/development/desa/news/population/world-urbanization-prospects-2014.html*