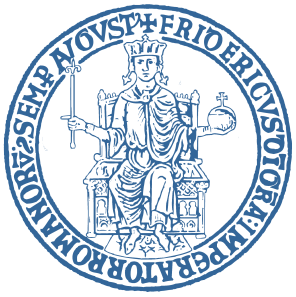
UNIVERSITÀ DEGLI STUDI DI NAPOLI
"FEDERICO II"



DIPARTIMENTO DI INGEGNERIA ELETTRICA
E TECNOLOGIE DELL'INFORMAZIONE

Corso di Laurea in Informatica

Tesi di laurea in Informatica

Un sistema multi-agente di supporto alle decisioni per lo smistamento di flussi croceristici in città

Candidato:
Pol Santamaria Mateu

Relatori:
Ch.ma Dott.ssa Silvia Rossi
Ch.ma Dott.ssa Claudia Di Napoli

ANNO ACCADEMICO 2014/2015

Abstract

0.0.1 Abstract

In questa tesi sarà studiata una metodologia per affrontare l'organizzazione e gestione degli arrivi delle crociere alle città basata sui Sistemi di Supporto alle Decisioni e con l'inclusione dei Sistemi Multi-Agente. La proposta ha l'intenzione d'aiutare a limitare le congestioni nella città derivate dell'arrivo di un gran numero di croceristi.

Nella tesi si studiano i diversi fattori che intervengono nel problema, soluzioni a problemi simili e casi reali d'implementazione. Un analisi dei requisiti viene condotto per poi progettare il design del sistema proposto. Ulteriormente si analizza l'implementazione di un prototipo a dimostrazione delle idee espresse in maniera teorica.

Riconoscimenti

Ringrazio i miei relatori, la Prof.ssa Silvia Rossi e la Ricerc. Claudia di Napoli per la pazienza che hanno avuto, sopratutto nelle lunghe correzioni linguistiche.

Anche i miei amici meritano un riconoscimento per avermi incoraggiato nei momenti di difficoltà e essersi mostrati comprensivi.

Infine un ringraziamento particolare va ai miei genitori che mi hanno sostenuto durante tutto il percorso di studi.

Indice

Lista delle Tabelle

Lista delle Figure

Capitolo 1

Visione generale del progetto

1.1 Preambolo

In questo lavoro di tesi sarà presentata una metodologia per affrontare problemi complessi che richiedono l'intervento di persone umane per prendere decisioni.

Il software che struttura questo tipo di problemi è conosciuto come Decision Support System, ovvero Sistema di Supporto alle Decisioni. La idea base è quella di offrire agli utenti la possibilità di prendere decisioni avendo conoscenza delle alternative e delle possibile conseguenze. In questo lavoro il DSS sarà progettato secondo la metodologia dei Sistemi Multi-Agente in base alla quale un compito viene suddiviso in piccole parti, ognuna gestita da un sistema autonomo e interagente con le altre.

1.2 Motivazione

C'è un alto numero di crociere che navigano di città in città trasportando turisti: i croceristi. Questo è un tipo di turismo particolare dato che i turisti visitano le città per un piccolo lasso di tempo, generalmente inferiore a una giornata.

Quando i passeggeri sbarcano in una città, essi visitano i posti più importanti partecipando in gite organizzate dalle crociere oppure indipendentemente. Quando capita che diverse crociere

sbarcano nel porto, è altamente probabile che i turisti abbiano l'intenzione di visitare gli stessi posti, generando problemi di congestione in alcune zone della città.

La Unione Europea ha finanziato il progetto *e-maritime* per *standardizzare il trasporto* marittimo attraverso l'utilizzo di nuove tecnologie con l'obiettivo di migliorare lo scambio di informazioni tra le diverse nazioni per agevolare il trasporto marittimo e renderlo più efficiente. Pertanto si auspica che in un futuro vicino si riusciranno a standardizzare i dati del trasporto marittimo e conseguentemente anche quelli delle crociere.

In questo contesto, si è pensato di affrontare alcuni dei problemi derivati dall'arrivo in massa di croceristi. Il progetto di questa tesi sarà svolto partendo dell'ipotesi che in un futuro si possa disporre dei dati relativi agli arrivi di crociere, e quindi si utilizzeranno i dati disponibili e altri saranno simulati.

1.3 Obiettivo della tesi

L'obiettivo di questa tesi è trovare un modo intelligente ed efficiente per gestire l'arrivo dei croceristi in modo che non si generino congestioni nella città. Una grande affluenza di turisti in una zona o un alto numero di autobus in circolazione sono fattori che influiscono nella creazione di congestioni. Per prevenire queste situazioni sarebbe conveniente trovare un modo per distribuire e organizzare le gite delle crociere e i croceristi.

Il problema in sé è molto complesso perché intervengono molti fattori legati alla città. Per questa ragione la persona in carico della organizzazione dovrebbe essere una autorità della città con accesso a dati molto diversi inerenti al traffico, trasporto pubblico, arrivo di crociere o punti d'interesse della città. Questo utente dovrebbe decidere itinerari per visitare la città che si adeguino ai requisiti delle crociere e mantenere una comunicazione con le agenzie di viaggio per raccomandarli. Questi itinerari sarebbero disegnati per visitare i punti d'interesse della città senza rischiare di congestionare la zona oppure evitando di incontrarsi con eventi prevedibili come un concerto o uno sciopero.

Visto che il problema è composto da fattori molto diversi si può strutturare come un insieme

di sotto-problemi da risolvere, dove per ogni sotto-problema ci sono decisioni che l'utente deve prendere.

1.4 Introduzione al problema

Per iniziare a lavorare sul problema è essenziale disporre dell'informazione inerente alle crociere come il numero di passeggeri, ora di arrivo e di partenza. Il primo problema da affrontare è gestire l'arrivo degli autobus nel porto, dato che in genere il porto dista dal centro della città o dalle stazioni centrali di trasporto pubblico e le agenzie contrattano autobus per trasportare i croceristi. Inoltre, il porto ha delle limitazioni sul numero di autobus che possono entrare senza congestionarlo.

Inoltre, gli autobus che portano i croceristi si devono distribuire fra diverse destinazioni per fare in modo che non si congestiona la zona o le fermate di trasporto. A questo punto già entra in gioco la conoscenza sullo stato della città come la presenza turistica, lo stato del trasporto o la presenza di eventi. Nella tesi si farà una gestione intelligente degli autobus che entrano nel porto e anche dello smistamento fra diverse destinazioni. Un ulteriore aspetto riguarda il numero di croceristi che prenotano una gita nella città con la crociera, e di conseguenza il numero di autobus prenotati. Il numero di passeggeri che prenotano una visita guidata è molto variabile, essendo vicino al 50% in una città come Barcellona oppure circa al 90% ad Istanbul o Napoli.

L'altra parte del problema consiste nel fornire alle agenzie raccomandazioni riguardo i punti d'interesse da visitare. Basandosi sulle previsioni dello stato della città e le informazioni del trasporto pubblico si possono organizzare gite che non subiscano alterazioni. Dato che l'organizzazione di percorsi è un problema già affrontato e risolto in molti modi, la tesi soltanto riporterà i punti d'interesse e lo stato della zona, senza organizzare le gite.

L'ultimo aspetto che affronterà la tesi è la previsione dell'evoluzione dello stato della città. In questo modo si potranno raccomandare periodi di tempo per visitare i punti turistici, distribuire l'arrivo degli autobus fra le diverse zone oppure prevedere le congestioni e raccomandare ai responsabili di trasporto di aumentare la capacità di una zona in un periodo di tempo

determinato. Per realizzare queste funzionalità, una parte dei dati saranno simulati per la indisponibilità d'informazione riguardo il traffico o il numero massimo di persone che possono trovarsi in un area senza generare congestioni.

1.5 Contesto

Il progetto appartiene all'ambito del software che offre sopporto alle persone che devono prendere decisioni, più specificamente forma parte dell'organizzazione del trasporto. La implementazione del software che verrà descritta non pretende di sostituire le persone nel ruolo di esperti, anzi ha lo scopo di aiutare gli operatori a svolgere il loro lavoro in un modo più semplice. Come si vedrà nella sezione 1.8, attualmente non esistono alternative per risolvere il problema descritto, giustificando lo svolgimento di questa tesi.

1.6 Impatto

Un DSS per migliorare il trasporto dei croceristi nella città, con il coinvolgimento delle autorità cittadine potrebbe aumentare l'efficienza delle risorse investite. Ad esempio, la congestione della rete di trasporto pubblico diminuirebbe grazie all'utilizzo più efficace degli autobus forniti alle agenzie di crociere. Si può concludere quindi che l'impatto economico sarebbe positivo.

Dall'altro lato, una diminuzione dei problemi di congestione e l'incremento dell'efficienza del servizio di trasporto porterebbe a una riduzione del consumo dei carburanti, generando un impatto positivo sull'ambiente.

1.7 Pianificazione

Il lavoro di tesi è stato svolto seguendo una serie di passi pianificati per raggiungere l'obiettivo prefisso e che sono di seguito riportati.

1. **Ricerca d'informazione:** Innanzitutto c'è bisogno di realizzare una ricerca per approfondire le diverse tematiche della tesi. Dopodiché si deve realizzare una ricerca per studiare gli articoli che riguardano gli argomenti della tesi o ad essi collegati, così da ottenere una formazione di base per poter approcciare il problema.

2. **Approccio al problema:** Studiare i casi simili al problema dei croceristi e decidere come affrontare la soluzione del problema presentato nella tesi. Poi si deve formalizzare il problema per scoprire i requisiti per portare a termine il lavoro.

3. **Raccolta dei dati:** Ricerca delle fonti d'informazione da cui estrarre i dati che verranno usati nel caso di studio, così come studiare la complessità dei dati trovati e i problemi relativi alla raccolta loro raccolta.

4. **Studio dell'architettura del software:** Studiare e disegnare l'architettura del DSS, sulla quale si baserà il prototipo da implementare.

5. **Progettare i modelli dei dati:** Decidere come verranno strutturati i dati, le relazioni tra di loro e come si possono elaborare per ottenere altre informazioni. A volte i dati provenienti da diverse fonti devono essere unificati in un unico modello.

6. **Progettare il modello del dominio:** Strutturare i dati disponibili e le loro relazioni assieme ai dati generati dal programma in tempo di esecuzione e che devono essere immagazzinati.

7. **Progettare le funzionalità del DSS:** Studiare e progettare le funzionalità più complesse prima di progettare la loro implementazione utilizzando diagrammi o altri metodi quando necessario.

8. **Implementazione del prototipo:** Portare a termine lo sviluppo parziale del progetto proposto nella tesi in modo che offra la possibilità di simulare scenari e osservare il funzionamento.

9. **Analizzare i risultati:** Tramite l'utilizzo del prototipo analizzare i risultati ottenuti dal software per valutare la sua utilità e possibili miglioramenti.

1.8 Stato dell'arte

Prima di iniziare a progettare e sviluppare un software si deve eseguire una fase di ricerca nella quale si determina l'esistenza o meno di una soluzione al problema dato, oppure una modo di adattare una soluzione già esistente.

Dopo aver iniziato una piccola ricerca si è evidenziato che non esisteva nessun software già implementato che potesse essere adattato con poche modifiche per risolvere il problema. Dopo un analisi degli articoli pubblicati su diversi editoriali online d'investigazione scientifica è emerso che nessuno aveva affrontato il problema dell'organizzazione del influsso dei croceristi nella città e la loro gestione.

Partendo della non esistenza di un software che fornisse una soluzione al problema nel suo complesso, si è condotta una ricerca per trovare soluzioni ad alcune delle diverse parti di cui è composto il problema. In particolare, la ricerca è stata orientata alla risoluzione dei diversi problemi elencati di seguito:

- Gestione del traffico, con lo scopo d'evitare congestioni.
- Organizzazione dei percorsi per i mezzi di trasporto.
- Modellazione e elaborazione delle informazioni che riguardano le crociere.

La ricerca riguardo la modellazione e il trattamento dell'informazione delle crociere non ha ottenuto risultati rilevanti, probabilmente perché finora questo tipo di informazioni non erano facilmente reperibili.

Riguardo l'uso dei DDS per la gestione del traffico, è di rilevanza il lavoro svolto di Ossowski et al. [Sascha Ossowski, 2004] [Sascha Ossowski, 2005]. La loro proposta prevede un software di gestione del traffico in cui si suddivide la rete stradale, ed ogni componente ha un supervisore umano che deve reagire quando si prevede una complicazione. I loro lavori verranno approfonditi nella sezione 3.7.

Due altri lavori rilevanti sono stati condotti nel campo della organizzazione dei percorsi per il

trasporto. Il primo è stato effettuato dal *Laboratorio delle Città del Futuro* (FCL), squadra stabilita a Singapore e formata da ricercatrici di Zurigo e Singapore.

Il progetto svolto prevede l'utilizzo del software di simulazione MATSim già dotato di modelli molto complessi in cui si simula ogni persona indipendentemente. La prima fase del lavoro consisteva nel raccogliere diversi tipi di dati di Singapore come statistiche delle abitudini delle persone o caratteristiche sociali ed economiche come il numero di posti di lavoro in una zona. Dopodiché si sono utilizzati questi dati come entrata per il simulatore e sono stati studiate diverse organizzazioni per le tratte del trasporto pubblico.

Il secondo lavoro nell'ambito dell'organizzazione del trasporto più attinente alle problematiche affrontate in questo lavoro, è stato svolto da Mohamad K. Hasan [Hasan, 2010] che hanno elaborato un modello teorico per rappresentare le persone e la loro necessità di trasporto a un livello altamente dettagliato per simulare il trasporto e prendere decisioni per migliorarlo. Il modello utilizza dati socio-economici per classificare gli utenti del trasporto in gruppi che hanno necessità diverse. Inoltre, propone una architettura globale per un DSS che permetterebbe al software di sfruttare dati molto diversi provenienti dalla gestione stradale, trasporto pubblico oppure statistiche socio-economiche.

Da un'altra parte, si sono trovati sistema per la gestione del trasporto per risolvere imprevisti, come il lavoro di Balbo e Pinson [Balbo and Pinson, 2010]. Il software propone decisioni all'utente per mantenere la pianificazione oraria decisa per gli autobus cercando d'evitare le congestioni oppure modificando il funzionamento del trasporto. Purtroppo, si cerca di evitare le congestioni, ovvero trovare soluzioni per non circolare per le strade congestionate. Invece, nella gestione dei croceristi si cerca di prevenire le congestioni, ovvero evitare che i croceristi e gli autobus le generano, motivo per il cui le loro idee non sono applicabili.

1.9 Struttura della tesi

Questa tesi è strutturata in 7 capitoli. I primi due capitoli, numerati con 2 e 3, esplorano le basi teoriche sulle quali si basa il progetto. Concretamente, nel capitolo 2 si presentano i Sistemi di Supporto alle Decisioni, il suo origine, le architetture software più popolari e le applicazioni

nel mondo reale tramite lo studio di implementazioni recenti.

Successivamente, nel capitolo 3 si presentano e approfondiscono le caratteristiche più rilevanti dei Sistemi Multi-Agente che bisogna conoscere perché sono parte del design e dell'implementazione del software.

Nel capitolo 4 si procede ad analizzare i requisiti che dovrà soddisfare il DSS da progettare per organizzare la gestione degli arrivi di crociera. In seguito, nello stesso capitolo 4, si presenta la progettazione elaborata per affrontare il problema e soddisfare i requisiti.

I dettagli dell'implementazione realizzata vengono presentati in maniera strutturata nel capitolo 5 per ottenere sia una visione globale sia una visione dettagliata delle diverse componenti.

Infine, nel capitolo 6 si presentano i risultati ottenuti tramite un caso di studio in cui si dimostrano le possibilità che offre il prototipo. Poi si procede a uno studio sugli sviluppi e future ricerche da realizzare per migliorare i risultati.

Capitolo 2

Sistemi di Supporto alle Decisioni

2.1 Definizione

In inglese viene chiamato Decision Support System (DSS). Si può definire come un gruppo di software che interattivamente aiuta gli utenti a prendere decisioni e a studiarle, la cui precisione e ottimalità giocano un ruolo cruciale.

2.2 Origine

Nel trattamento dei problemi complessi c'è un elevato numero di fattori da considerare con interdipendenze difficile di individuare. Questi possono creare una grande difficoltà nella predizione delle conseguenze derivanti da una decisione.

In molti ambiti come nel mercato azionario o la gestione del trasporto le decisioni da prendere richiedono velocità, precisione e ottimalità. Quando questo lavoro viene portato a termine da una persona, il processo risulta molto logorante e porta al limite la capacità delle persone. Inoltre, le persone fanno valutazioni soggettive [Druzdzel and Flynn, 1991], pertanto è interessante la idea di risolvere parte del problema automaticamente attraverso l'utilizzo di un sistema software che permetta alle persone di prendere decisioni con un minore sforzo e allo stesso tempo con più informazione sulle conseguenze, e servendosi di modelli matematici per confrontare le

decisioni e non basarsi solo sull'esperienza o il giudizio personale.

2.3 Come aiuta un DSS?

Un DSS fornisce supporto nel prendere decisioni in diversi modi. Innanzitutto deve modellare e analizzare il sistema, cioè, rappresentare i diversi elementi del dominio e come interagiscono.

Una volta studiato il sistema deve strutturare il problema decisionale, ovvero, per ogni decisione che si possa prendere, deve analizzare le dipendenze tra gli elementi che partecipano. Inoltre, è rilevante che il DSS abbia la capacità d'identificare e proporre decisioni alternative perché può influenzare nella qualità delle decisioni finali [Druzdzel and Flynn, 1991].

Inoltre, si deve offrire la possibilità di esplorare le conseguenze che possono derivare dalle alternative presentate all'utente. Infine, un DSS deve anche avere la capacità di scoprire le situazioni in cui c'è una decisione da prendere e comunicarle all'utente.

2.4 Modellazione dei problemi

In diversi esperimenti analizzati da Dawes [Dawes, 1988], si è dimostrato che le decisioni presse mediante modelli semplici ottengono migliori risultati che le decisioni presse con l'intuizione umana, in entrambi casi partendo dagli stessi dati. Si è osservato che un modo per migliorare le decisioni umane è quello di scomporre i problemi decisionali in componenti più semplici che sono ben definite e comprese. A partire da questi sotto-problemi strutturati possiamo utilizzare diverse tecniche disponibili per risolvere il problema globale.

Il processo di scomporre e standardizzare un problema viene conosciuto come "modeling", dove si rappresenta il mondo reale in un modo semplificato per ridurre la complessità, e anche in un modo matematico con l'intenzione di sviluppare algoritmi che ne permettano la gestione.

La modellazione di un problema si può studiare da due punti di vista diversi, il matematico e il decisionale. Dal punto di vista matematico un modello è formato dalle variabili e le sue interazioni ed è usato da coloro che devono implementare i modelli. Invece, il punto di vista

decisionale è quello che hanno gli scienziati che studiano i problemi decisionali in modo astratta e come li risolvono le persone. Perciò, dal punto di vista decisionale un modello deve essere formato da:

1. Una misura di preferenze tra i diversi obiettivi decisionali. È una delle parte più importanti nel processo decisionale perché permette di scegliere gli aspetti più rilevanti delle possibili soluzioni. Inoltre, esprimere le preferenze numericamente fornisce la possibilità di valutare le decisioni confrontando la desiderabilità ed i rischi.

2. Le diverse decisioni disponibili.

3. La misura dell'incertezza delle variabili che partecipano nelle decisioni e nelle conseguenze. L'incertezza è presente nel mondo reale, e diventa ancora maggiore nel fare semplificazioni e approssimazioni quando si rappresentano i problemi.

Questi aspetti sono i requisiti di un modello che devono essere implementati. Di conseguenza, nei modelli deve esistere una preferenza fra i diversi obiettivi dell'utente per valutare le decisioni, la possibilità di studiare tutte le decisioni possibili e l'inclusione di modelli probabilistici per rappresentare l'incertezza.

Infine, la partecipazione degli esperti del dominio nel disegno e implementazione dei modelli è indispensabile. Essi sono gli unici che hanno le conoscenze necessarie per individuare gli aspetti importanti da modellare e come incidono sul problema.

2.5 Architettura di un DSS

Sono state proposte architetture molto diverse per implementare i DSS. Una delle prime proposte fu quella di Minch e Burns [Minch and Burns, 1983] sulla quale si sono basate tante altre architetture. La loro proposta era la di strutturare il sistema in 4 componenti, le cui funzionalità sono state ampliate e approfondite in ulteriori articoli [Makowski, 1991] [Druzdzel and Flynn, 1991]:

Database Management System (DBMS): Immagazzina i dati utilizzati per il DSS e li fornisce senza processarli, ed è responsabile di controllarne l'accesso [TutorialsPoint, 2015].

Rappresenta anche un livello di astrazione rispetto alle diverse tecnologie che si possono utilizzare per immagazzinare i dati. Una caratteristica desiderabile è quella di fornire agli utenti l'informazione riguardo ai diversi dati disponibili e come accederli.

Model-base Management System (MBMS): Questa componente deve provvedere indipendenza tra l'applicazione e i modelli utilizzati per rappresentare l'informazione. Un'altra delle sue funzionalità è quella di elaborare i dati provenienti dal DBMS tramite i modelli, per far in modo che questi siano utili per il processo decisionale. Sarebbe anche desiderabile che l'utente avesse la possibilità di manipolare ed elaborare nuovi modelli.

Problem Processing System (PPS): È la parte logica che trova quale sono le decisioni da prendere, le alternative e le loro conseguenze grazie alle informazioni fornite dal MBMS e dal DBMS.

Il primo passo è ottenere i dati dai modelli e processarli, sia filtrandoli oppure unificandoli. Dopo questo processo i risultati vengono comunicati all'utente tramite un'altra componente che cercherà di presentarli in maniera comprensibile. Il modo di funzionamento di questa componente può essere molto diverso: a volte è basato su un sistema di regole, altre su metodi euristici.

Dialog Management System (DMS): Dato che generalmente la formazione degli utenti non prevede alte competenze informatiche, il DSS deve avere interfacce intuitive e usabili. Le interfacce devono aiutare a interagire con il modello e abilitare la esplorazione delle conseguenze di una decisione data o produrre raccomandazioni. Essendo il DSS uno strumento complesso la sua complessità deve essere trasparente all'utente che non deve essere sommerso da grandi moli di informazioni; pertanto il DMS deve consentire all'utente una facile modalità di accesso e consultazione dell'informazione disponibile.

Per poter soddisfare le caratteristiche di ottimalità e velocità del processo decisionale, i messaggi devono essere concisi e chiari. Infine, un fattore che può aiutare molto positivamente nell'utilizzo di un DSS è l'inclusione di grafici nella interfaccia per ottenere una visione veloce del problema.

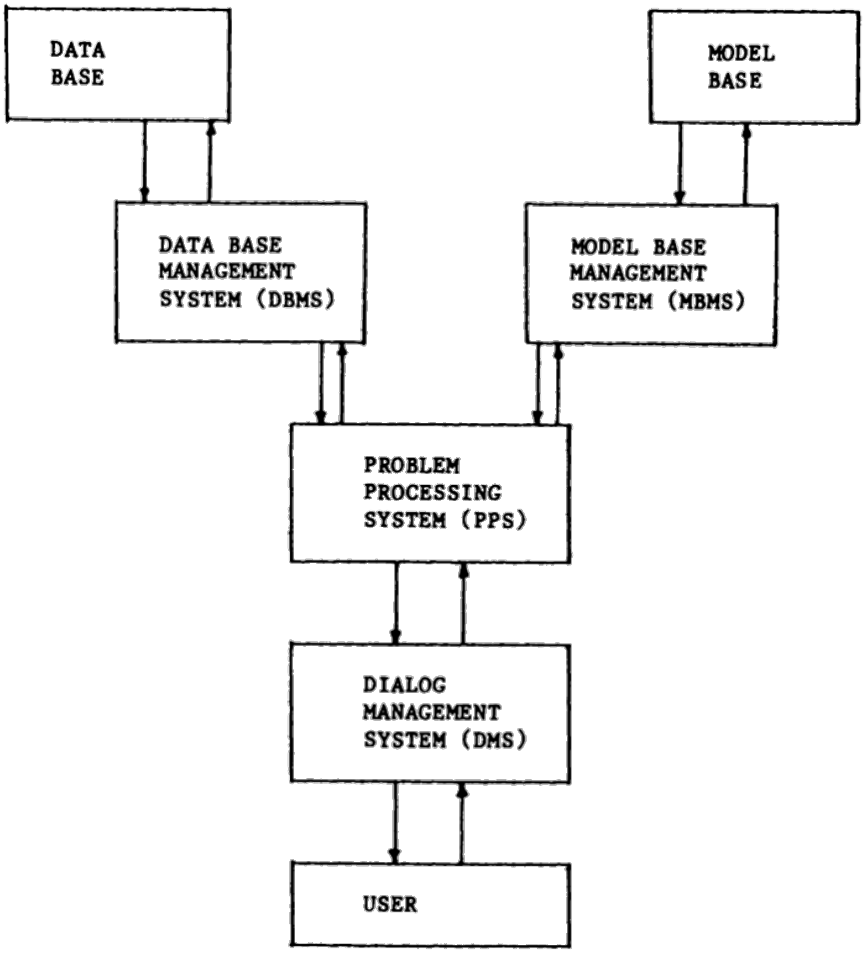


Figura 2.1: Architettura di un DSS proposta da Minch e Burns.

Successivamente sono state proposte architetture molto più complesse mantenendo i 4 blocchi come componenti centrali, anche se a volte vengono unificati il PPS con il DMS. Un'architettura più recente è quella di [Turban, 1995] mostrata in 2.2. continuazione:

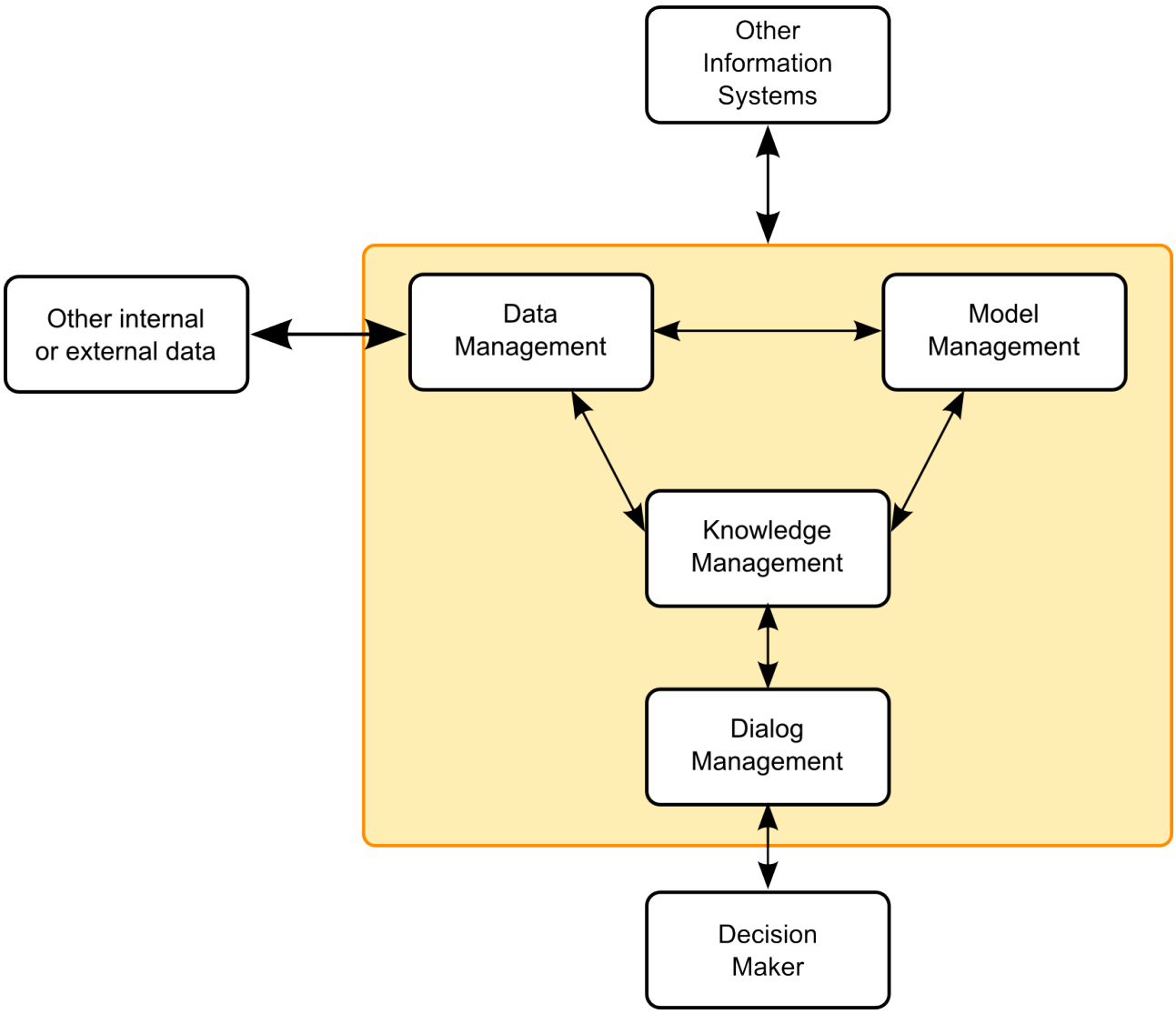


Figura 2.2: Architettura di un DSS proposta da Turban.

Il primo miglioramento della architettura di Turban riguarda l'aggiunta di un canale di comunicazione tra il MBMS e il DBMS per trasmettere i dati ai modelli direttamente.

Si può notare nello schema che la componente PPS proposta da Minch e Burns è stata sostituita con la componente di Knowledge Management. Entrambe componenti hanno comunque le stesse funzioni.

La proposta di Turban prevede la possibilità che il sistema interagisca con altri sistemi informatici che possa acquisire dati provenienti dall'esterno, attraverso le interrogazioni di servizi disponibili su Internet. Attualmente, il modello di Turban rappresenta una proposta di riferimento per gli sviluppatori.

In conclusione, è facile renderci conto che un DSS è uno strumento potenzialmente utile nel trattamento dei problemi decisionali complessi come ad esempio la gestione del trasporto che verrà studiata nella prossima sezione.

2.6 DSS per il trasporto e traffico

L'utilizzo dei DSS sia per applicazioni di gestione del traffico che del trasporto si è rilevante molto promettente. Lo scopo principale nella gestione del traffico è di evitare le congestioni, mentre che nella gestione del trasporto si cerca di reagire agli imprevisti e ottenere un'alta efficienza del servizio.

Entrambi problemi presentano similitudini con il problema dello smistamento di croceristi, che richiede la prevenzione della congestione nella città e l'organizzazione del trasporto dal porto fino a questa. Considerando le similitudini trovate si sono studiati questi sistemi software per avere un esempio di riferimento.

2.6.1 Analisi dei Sistemi per la Gestione del Trasporto

I sistemi di gestione del trasporto, detti Transport Management Systems (TMS) sono formati da elementi software e hardware. Il loro scopo principale è controllare un sistema di trasporto già organizzato, aiutando a scoprire le anomalie e proponendo soluzioni ai problemi individuati.

Inizialmente, questi strumenti non erano pensati per pianificare la organizzazione del trasporto, compito affidato ai Transport Planners (TP). Attualmente la frontiera tra TMS e TP è così sottile che spesso i loro nomi vengono scambiati oppure sono sviluppati in un unico sistema software.

Un TMS è uno strumento con molte funzionalità di diversa complessità. Ad esempio, esistono TMS che hanno una parte di pianificazione, e che sono capaci di dare supporto nella scelta delle tratte e orari, tenendo conto delle restrizioni imposte dalla autorità, città ed altri fattori. Questi di solito hanno anche la possibilità di simulare la pianificazione grazie ai dati storici dei flussi di passeggeri, e offrono la possibilità di confrontare diversi scenari.

Senza dubbio la funzionalità principale di cui sono dotati tutti i TMS è la possibilità di sorvegliare l'esecuzione degli orari e dei percorsi programmati, così come l'identificazione di alterazioni per reagire velocemente per migliorare l'efficienza del sistema di trasporto.

Altre funzionalità meno frequenti però utili e di basso carico computazionale sono il controllo della manutenzione dei veicoli o la possibilità di configurare allarmi che scattano in situazioni definite dagli utenti.

I DSS utilizzati per implementare i TMS lavorano principalmente a partire di dati raccolti tramite sensori, con lo scopo di interagire con il mondo. I sensori usati sono molto diversi e numerosi, pero senza dubbio il più utilizzato è il GPS che continuamente informa della posizione del veicolo.

Invece, per comunicare le decisioni prese sono molto usate le comunicazioni sonore per informare gli utenti, e via radio per informare l'autista.

Dovuto alle grandi dimensioni delle reti di trasporto ci saranno vari utenti che supervisioneranno il corretto funzionamento, ognuno con la responsabilità di gestire un insieme di linee.

I supervisori dovranno studiare diversi tipi di decisioni, ad esempio, aggiungere un autobus in una linea, proveniente dal deposito oppure da un'altra linea. Un'altra decisione abituale è quella di alterare il percorso per evitare una strada chiusa o congestionata, oppure di chiedere all'autobus di fermarsi per poco tempo in una fermata per rispettare l'orario.

2.6.2 Analisi dei Sistemi per la Gestione del Traffico

Le autorità che gestiscono le strade usano questi sistemi per controllare lo stato del traffico, evitare le congestioni e risolvere gli incidenti. Pertanto, gli utenti che interagiranno con il DSS saranno impiegati con formazione sulla gestione stradale, con la responsabilità di sorvegliare una parte della rete stradale.

Per analizzare lo stato del traffico i DSS prelevano dati da diversi sensori, principalmente a partire da telecamere e altri sistemi che contano il numero di veicoli. Il sistema processa i dati e informa gli utenti delle alterazioni offrendo la possibilità di studiare la situazione anche se

non è necessario un intervento. Nella Figura 2.3 si può osservare la infrastruttura minima di cui sono dotate la maggior parte delle strade attualmente.

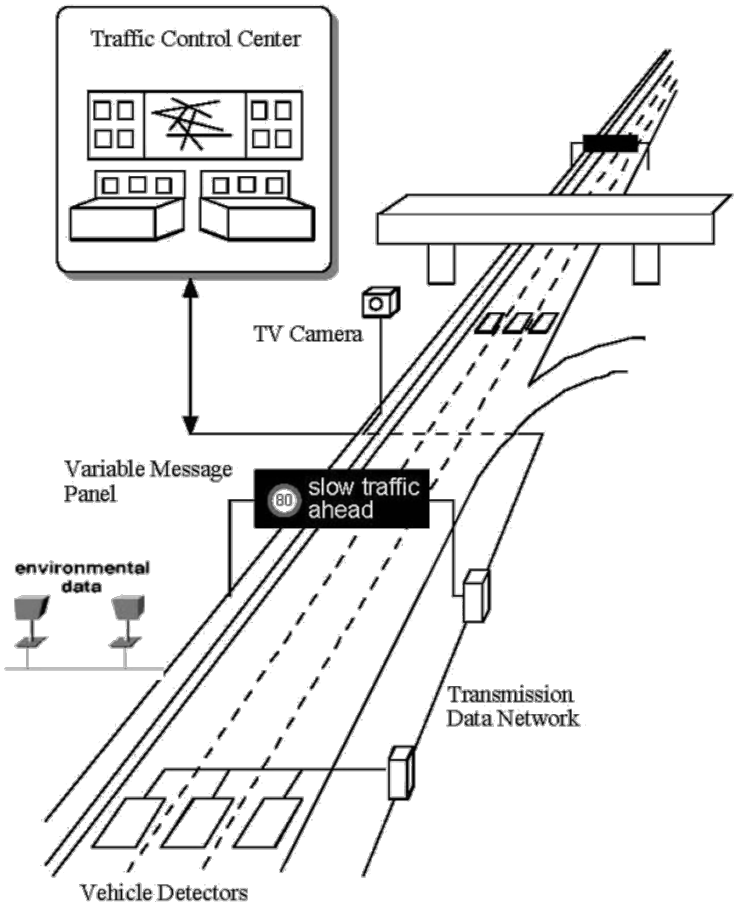


Figura 2.3: Infrastruttura stradale [Almejalli et al., 2007, p. 551].

Dal punto di vista dell'utente, le decisioni da prendere possono essere molto diverse. Una tra le più comuni è cercare una strada alternativa e raccomandarla ai guidatori per prevenire congestioni, tramite i pannelli informativi. Altre decisioni comuni sono la richiesta dell'intervento di un agente esterno come il corpo di polizia per intervenire in un incidente. Una ulteriore decisione molto ricorrente è cambiare il senso della marcia delle corsie che possono operare bi-direzionalmente a seconda delle necessità.

Come al solito, il numero di decisioni disponibile dipenderà dalla infrastruttura che fornisce i dati e della implementazione del DSS.

2.6.3 Esempi concreti

Solitamente entrambi problemi non vengono trattati insieme perché le competenze di gestione appartengono a autorità diverse. Di seguito si analizzerà prima un esempio teorico della gestione del trasporto, proposto da Mohamad K. Hasan. Poi si descriverà il software implementato per il governo di San Diego, che cerca d'evitare e risolvere le congestioni.

Approccio di Hasan per l'organizzazione del trasporto

Un buon esempio dello stato dell'arte dei TMS è la proposta di Hasan [Hasan, 2010] che prevede un Intelligent DSS, e cioè un DSS che fa un uso intensivo di tecniche d'intelligenza artificiale (IA). Il software proposto ha lo scopo principale di fare previsioni per organizzare il trasporto, per poi valutare la necessità di realizzare cambiamenti nella rete di trasporto.

Per rappresentare l'informazione, il DSS utilizza un modello chiamato Multiclass Simultaneous Transportation Equilibrium Model (MSTEM) formulato da Hasan e basato sul Simultaneous Transportation Equilibrium Model (STEM) già esistente. Il modello STEM riflette i fattori sociali ed economici della città e della popolazione. Il MSTEM è una estensione che consente di aggiungere diversi profili di utenti del trasporto basandosi su diversi parametri come il gruppo socio-economico di appartenenza.

Inoltre, il software permette di gestire diversi scenari modificando i parametri del modello, per poi confrontarli.

Un aspetto rilevante è la ricchezza di caratteristiche che rappresentano il mondo reale in un modo molto dettagliato, modellando ogni utente del trasporto individualmente.

L'architettura proposta ha la potenzialità di offrire funzionalità molto diverse, che consentono di generare una ampia gamma di scelte con una accurata previsione delle conseguenze.

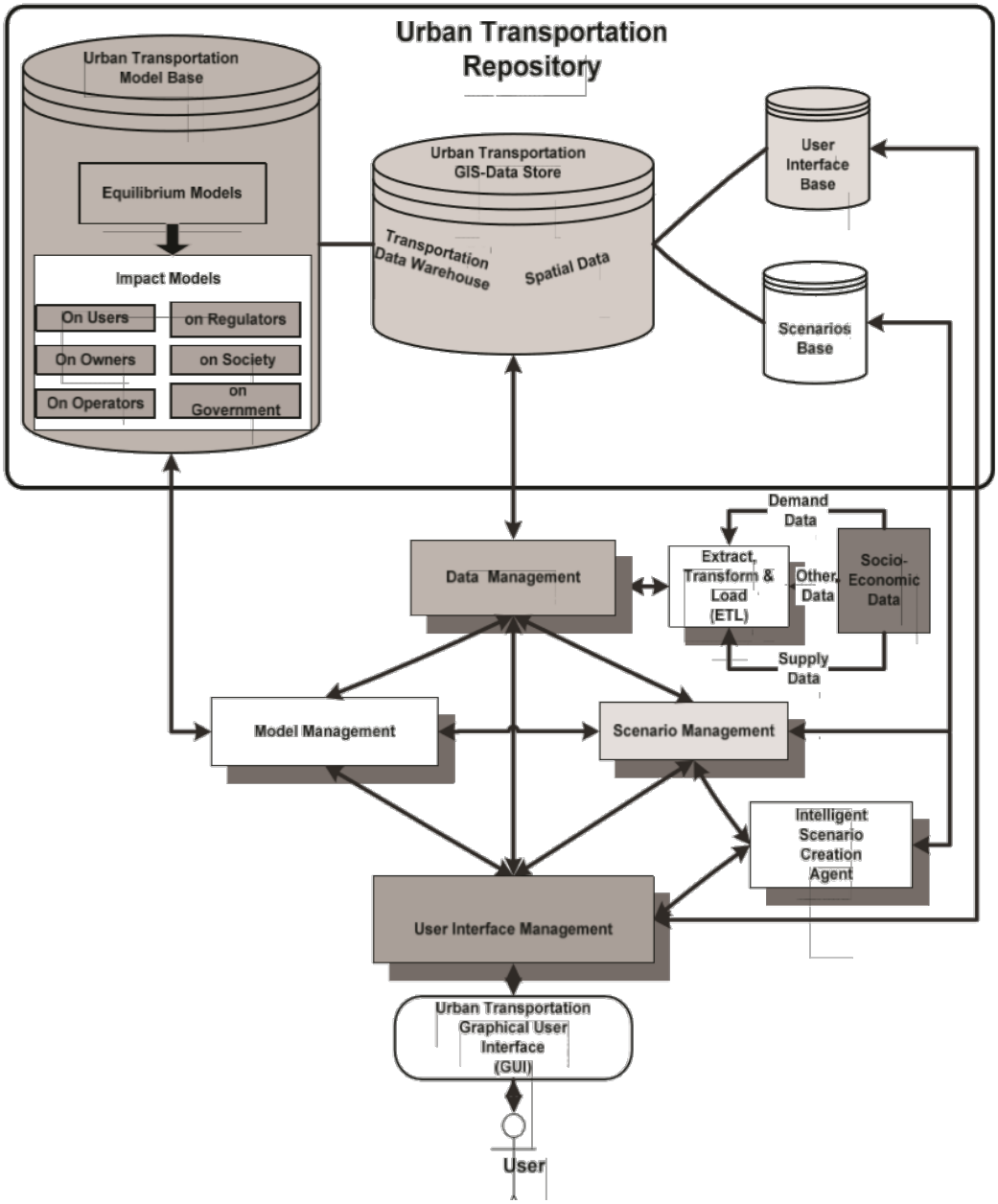


Figura 2.4: Architettura per l'IDSS proposto da Hasan.

Come si può osservare nella Figura 2.4 il sistema mantiene l'architettura generica di un DSS con il Model Management, Data Management, User Interface, ma senza la presenza della componente di computo Problem Processing System (PPS). Infatti, il sistema utilizza unicamente i modelli per calcolare le possibili decisione e le relative conseguenze.

Inoltre, c'è una comunicazione diretta tra Data Management e Model Management come visto nel modello di Turban 2.5, quindi tale comunicazione non è più gestita dal PPS. Nonostante la componente PPS sia stata rimossa è stato aggiunto il gestore di scenari, lo Scenario Management, che offre la possibilità di gestire diverse simulazioni e conservarle.

Nonostante sia una proposta teorica, il modello è molto completo e rappresenta una base per futuri sviluppi.

San Diego Integrated Corridor Management System

Un esempio pratico attuale nel campo della gestione del traffico è il sistema sviluppato dal Dipartimento di Trasporto degli USA [DOT, 2015]. L'*Integrated Corridor Management* (ICM) è una iniziativa interregionale per migliorare la gestione delle superstrade americane e ridurre gli incidenti, ancora in fase d'implementazione.

Per fare diversi test sul prototipo sono state selezionate due autostrade dove implementare completamente il sistema, una a Dallas (Texas) e l'altra a San Diego (California).

In questa sezione si studierà l'implementazione realizzata per la Inter-statale 15 (I-15) di San Diego, sotto la direzione della *San Diego Association of Governments* (SANDAG) e in collaborazione con altre entità americane. Il software è stato rilasciato nel 2014.

Il sistema è composto da due moduli, la componente di predizione della rete (NPS) e la componente per le simulazioni real-time (RTSS):

Network Prediction Subsystem (NPS): Prende come dati d'entrata l'informazione riguardo la rete e produce predizioni per la prossima ora, suddivisa in periodi di 15 minuti, per individuare possibili problemi di congestione. Dall'altro lato genera e struttura i dati per realizzare microsimulazioni. Entrambi processi si realizzano ogni 5 minuti.

Real-Time Simulation Subsystem (RTSS): Il modulo funziona sotto richiesta dall'utente. Prende i dati generati dal NPS insieme alle azioni che si possono eseguire nella zona e realizza microsimulazioni eseguendo tutte le azioni. Il modulo riporta i risultati ottenuti, ovvero l'impatto delle azioni presse e come evolve la situazione in ogni caso ipotetico e anche quale sarebbe il risultato di non applicare nessuna azione. Per confrontare le simulazioni il modulo applica una formula che valuta il nuovo scenario e riporta un numero all'utente.

Grazie all'utilizzo di entrambi moduli si possono prevedere le congestioni e dopo analizzare le conseguenze che avrebbero le diverse azioni possibili. La Inter-statale 15 è dotata di dispositivi e metodologie moderne come corsie riversibili o corsie a pagamento per veicoli con alta occupazione dove il prezzo cambia dinamicamente.

Di conseguenza, esiste anche un alto numero di decisioni possibili da intraprendere come cambiare la sincronizzazione dei semafori, regolare l'accesso alla superstrada o favorire la circolazione degli autobus rispetto agli altri veicoli [Jordi Casas, 2014] [Aimsun, 2015].

Un lavoro sull'impatto nella riduzione del numero d'incidenti non è stato ancora pubblicato pero si che si esistono dati che giustificano economicamente lo sviluppo e implementazione del sistema [SANDAG,].

Capitolo 3

Sistemi Multi-Agente

3.1 Definizioni

Il concetto di agente viene sviluppato nel campo dell'intelligenza artificiale negli anni 70 ed è rimasto invariato [Brahim Chaib-Draa and Millot, 1992]. Si intende come agente una entità capace di ottenere informazioni dal mondo circostante tramite sensori, inferenza o atti comunicativi, ed ha l'abilità d'interagire con l'ambiente tramite attuatori. Inoltre, un agente deve avere una certa autonomia, ovvero capacità per imparare e adattarsi a nuove situazioni.

All'inizio il modello prevedeva che un singolo agente fosse responsabile di tutti i compiti e l'unico ad interagire con l'ambiente, però verso il 1980 [Konolige and Nilsson, 1980] si vide che questo approccio presentava troppe limitazioni.

A questo punto nacque la idea di suddividere i compiti fra diversi agenti che avessero un obiettivo in comune, dando così origine ai Sistemi Multi-Agente (SMA).

3.2 Caratteristiche degli agenti

Alonso F. et al. [Alonso et al., 2008] hanno elaborato un elenco delle proprietà che caratterizzano gli agenti software:

Reattività: Si dice che un agente è reattivo se mantiene una continua interazione con l'ambiente e agisce ai cambiamenti.

Pro-attività: Un agente che non reagisce unicamente agli eventi ma cerca di raggiungere uno o diversi obiettivi e prende la iniziativa per farlo è pro-attivo.

Abilità sociale: Se l'agente è capace d'interagire con altri agenti per poter gestire il sistema ha l'abilità sociale. Questa caratteristica pero ha bisogno d'un canale di comunicazione fra agenti, un linguaggio comune e l'intenzione di cooperare.

Autonomia: Un agente che opera solo senza l'intervento umano o di altri elementi esterni si considera autonomo. Per farlo deve avere il controllo delle sue azioni e informazione interna.

Adattabilità: L'agente ha l'abilità di cambiare il suo stato in accordo ai cambiamenti dell'ambiente così da poter gestire gli imprevisti prodotti dall'ambiente.

Intelligenza: Perché un agente sia considerato intelligente deve possedere diverse capacità. Per primo, l'agente deve operare secondo criteri di logica e razionalità. Inoltre, deve essere dotato è la capacità di pianificare per raggiungere gli obiettivi. Infine, l'agente deve imparare dal passato. Gli agenti intelligenti aiutano a risolvere problemi di alta complessità laddove una soluzione senza la suddivisione in sotto-problemi non sia possibile.

Mobilità: Se un agente può spostarsi nell'ambiente o fino ad altri ambienti preservando la sua informazione. Deve essere capace d'interagire con il nuovo ambiente per raccogliere l'informazione necessaria per portare a termine i suoi compiti.

In uno dei primi lavori che cercava di raggruppare le conoscenze dell'epoca sugli agenti si descrisse gli agenti come sistemi che erano dotati di autonomia, abilità sociale, reattività e pro-attività [Wooldridge and Jennings, 1994]. Tuttavia lo studio degli agenti è evoluto, perciò in questa sezione si sono descritte soltanto le caratteristiche più comuni degli agenti.

3.3 Tipi di agenti

Per progettare agenti software, si deve scegliere un modo per strutturare il loro funzionamento. Tra i primi modelli proposti in letteratura, ci sono quelli definiti per prima volta da Russell e Stuart nel 1994 [Russell and Norvig, 2003] che introducono 4 tipologie di agenti:

Riflessivo semplice: Reagisce agli eventi del mondo, e per ogni evento ha un sistema a regole per scegliere la prossima azione da intraprendere.

Riflessivo basato su un modello: Ha un modello del mondo, la conoscenza delle conseguenze di ogni azione e una rappresentazione in forma di stato di com'è il mondo attualmente e anche storicamente.

Utilizza l'informazione del mondo per modificare lo stato dell'agente grazie al modello che da significato alle osservazioni e permette di sapere qual'è il prossimo stato. Una volta attualizzato lo stato, si decide l'azione da eseguire tramite le regole che relazionano ogni stato con una o diverse azioni.

Orientato ad obiettivi: Ha un modello del mondo, la conoscenza delle conseguenze di ogni azione e una rappresentazione in forma di stato di com'è il mondo attualmente e anche storicamente.

E' caratterizzato da uno o diversi stati desiderati e diversi stati non desiderati. L'agente deve elaborare un piano per raggiungere lo stato desiderato senza passare per quelli non desiderati.

Orientato alla utilità: Ha un modello del mondo, la conoscenza delle conseguenze di ogni azione e una rappresentazione in forma di stato di com'è il mondo attualmente e anche storicamente.

L'agente deve pianificare le azioni, e per decidere quale saranno le azioni da eseguire valuta le sequenze di stati che si ottengono come conseguenza delle azioni. Una volta valutate le diverse sequenze sceglie la più utile.

La scelta di quale tipologia di agente adottare dipende del compito che dovrà portare a termine l'agente. Per compiti semplici come un robot che abbia 2 o 3 azioni a disposizione può bastare

una implementazione semplice riflessiva. Invece, per affrontare un problema di pianificazione complesso può risultare meglio avere un approccio orientato ad obiettivi o basato sulla utilità.

3.4 Design degli agenti

Decidere come dovrà comportarsi un agente non è affatto banale, le metodologie sono in continua evoluzione e si basano su diverse discipline oltre all'informatica come la psicologia o la filosofia.

Nell'anno 1971 Daniel Dennett, filosofo americano, fu il primo a parlare di sistemi intenzionali [Dennett, 1980] da cui proviene la metodologia *Belief-Desire-Intention (BDI)*. Tale metodologia consente di modellare il processo di decidere, momento per momento, quali azioni l'agente deve compiere per perseguire i propri goal. Secondo il modello BDI, un agente è caratterizzato da 3 componenti che ne definiscono lo stato mentale:

Credenze: È la visione del mondo che ha l'agente, che comprende le informazioni riguardanti l'ambiente e il contesto, e le regole di inferenza utili per dedurre nuove informazioni. Tale visione può essere parziale o anche errata.

Desideri: Gli obiettivi dell'agente vengono espressi come desideri, che non sono altro che gli stati che l'agente vorrebbe raggiungere.

Intenzioni: Sono i piani che ha elaborato un agente per soddisfare i desideri, e devono essere compatibili tra di loro.

Questa strutturazione dell'informazione in categorie è particolarmente utile nei problemi complessi perché descrive le azioni degli agenti mediante analogie con il comportamento umano. Nell'attualità, il modello BDI si trova tra i modelli più diffusi per rappresentare un agente. Il primo lavoro in applicare il modello BDI nell'implementazione reale di agenti fu condotto da Rao e Georgeff [Rao and P.Georgeff, 1995].

3.5 Comunicazione fra agenti

Affinché l'atto di comunicazione sia possibile fra gli agenti, permettendo a questi di cooperare, deve esistere un canale di comunicazione e un linguaggio comune.

Il primo linguaggio di comunicazione fra agenti (ACL) fu il *Knowledge Query and Manipulation Language* (KQML), sviluppato per la DARPA [Finin et al., 1993]. Il linguaggio KQML permette d'inviare messaggi agli agenti individualmente o a gruppi, mediante operazioni di alto livello come le aste o i meccanismi di negoziazione.

Il 1995 nacque la Foundation for Intelligent Physical Agents (FIPA) con l'obiettivo di standardizzare i software utilizzati dagli agenti. Non fu fino il 1997 che la FIPA pubblicò lo standard per il linguaggio di comunicazione fra agenti chiamato FIPA-ACL, prendendo il linguaggio *N.B. ARCOL* di *France Télécom* come base invece di KQML. Nei successivi anni la FIPA ha pubblicato diversi lavori nel campo degli agenti e il 2005 è entrata a far parte dell'*IEEE* [FIPA, 2005].

Le specifiche del linguaggio di comunicazione fra agenti proposto è il FIPA-ACL che fornisce la struttura di ogni atto comunicativo.

3.6 Infrastruttura di un SMA

In genere, qualsiasi implementazione di un SMA deve affrontare una serie di difficoltà che riguardano l'organizzazione del sistema. La prima cosa a fare è identificare univocamente gli agenti. Se il sistema non prevede l'introduzione di nuovi agenti nell'interno del SMA durante il funzionamento, si possono assegnare identità univoche ad ogni agente nella implementazione. Invece, se si permette la introduzione di agenti in tempo di esecuzione il sistema deve offrire un modo per gestire l'assegnazione di nomi identificativi agli agenti.

Gli agenti hanno bisogno di un meccanismo per sapere quali sono gli agenti che convivono nel sistema e come comunicare con loro. Tale meccanismo va specificato nella fase di design del SMA.

Siccome un SMA è formato da diversi agenti, in alcuni casi risulta interessante incorporare un

meccanismo di sicurezza per validare l'autenticità degli agenti e i messaggi ricevuti. Un altro aspetto interessante riguardo i sistemi distribuiti come i SMA è la mobilità, per cui è possibile lo spostamento degli agenti o del sistema senza alterarne il funzionamento.

La FIPA ha rilasciato le specifiche dell'architettura di un SMA [FIPA, 2002a] le cui componenti principali sono descritte di seguito.

Agent Platform (AP): Formato dall'infrastruttura fisica e il sistema operativo, si trovano le altre componenti del SMA.

Agent: Processo computazionale che realizza i compiti assegnati, comunica con gli altri agenti tramite un ACL e ha un identificatore AID univoco.

Directory Facilitator (DF): Componente opzionale che può essere replicata nella stessa AP. Ha lo scopo di mantenere un registro dei servizi offerti dagli agenti, in modo che gli agenti possono scambiare servizi. Gli agenti che vogliono offrire un servizio devono registrarsi al DF, e possono registrarsi tante volte come servizi offrono.

Agent Management System (AMS): Unico in ogni AP, controlla l'accesso e supervisiona la piattaforma mantenendo un registro di tutti gli agenti presenti nella AP e assegnando a loro un nome identificativo (AID). Gli agenti non devono registrarsi all'AMS dato che è l'AMS che gestisce la creazione degli agenti, e di conseguenza ha le informazioni riferenti agli agenti della piattaforma.

Message Transport Service (MTS): Canale di comunicazione fra gli agenti descritto in profondità in un altro standard [FIPA, 2002b].

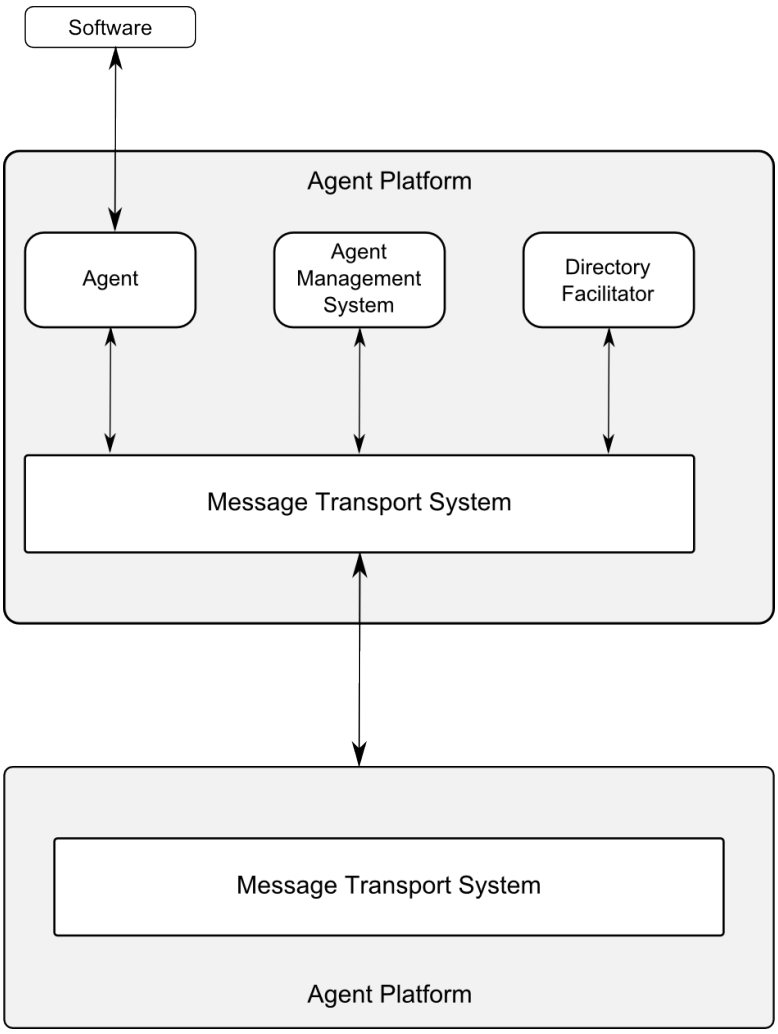


Figura 3.1: Modello di riferimento per l'organizzazione degli agenti della FIPA.

La infrastruttura divisa in AP che comunicano attraverso un canale di comunicazione permette estendere i sistemi multi-agente mediante l'aggiunta di nuovi AP, che consentono la comunicazione tra agenti che risiedono su sistemi fisicamente lontani.

3.7 Esempi di applicazioni per la gestione del traffico e degli autobus

Ossowski et al. [Sascha Ossowski, 2004] hanno realizzato due esperimenti in Spagna con l'obiettivo di studiare i vantaggi dei DSS basati su un'architettura multi-agente. Entrambi esperimenti furono simulati a partire da dati storici per divieto d'accesso ai dati real-time.

Il primo esperimento riguardava la gestione del traffico in una zona problematica di Bilbao.

L'area fu suddivisa in diverse zone, dove ogni zona era trattata come un sotto-problema.

L'architettura proposta contemplava l'utilizzo di 5 tipi di agenti, replicati a seconda del numero di zone:

Data Agent (DA): Raccoglie i dati dai sensori e completa l'informazione a partire dai database o ne filtra il contenuto. Ha il compito di trasformare i dati quantitativi in qualitativi.

Problem Detection Agent (PDA): Ogni Data Agent ha un unico PDA a cui comunica l'informazione. Il PDA ha la conoscenza per rilevare la presenza di alterazioni grazie all'utilizzo di frames, ovvero modelli che hanno parametri con valori e permettono al sistema di trovare il modello che rappresenta meglio la situazione. Una volta identificato un problema nella zona, si contattano i diversi Control Agent assegnati al PDA trasmettendo l'informazione disponibile.

Control Agent (CA): Gestisce diversi PDA con l'obiettivo di trovare soluzioni ai problemi comunicati dai PDA. Una volta trovata una soluzione questa viene comunicata agli altri CA per confermare che non entri in conflitto con le operazioni degli altri CA, e in caso di conflitto s'inizia una negoziazione. Un conflitto potrebbe sorgere nel caso un CA individui una strada alternativa da suggerire a causa di un incidente, mentre un altro CA cerca di ridurre la congestione della strada che è stata individuata.

Action Implementation Agent (AIA): Esegue le decisioni prese dagli utenti. Ad esempio, un panello stradale che mostra un messaggio ai guidatori.

User Interface Agent (UIA): Insieme di interfacce grafiche che permettono agli operatori di interagire con il sistema. I CA comunicano l'informazione alla UIA, e questa comunica le decisioni prese agli AIA.

L'assegnazione dei CA ai PDA è fatta secondo il criterio di raggruppare le zone contigue sotto lo stesso CA. Quando due zone sono adiacenti ed hanno un CA diverso, il loro PDA sarà in comunicazione con entrambi CA. Questo esperimento doveva superare la sfida di gestire diversi problemi che hanno interdipendenze dovuto alla continuità della rete di traffico. Invece, il secondo esperimento realizzato a Malaga affrontava la gestione degli autobus gestiti da un'azienda pubblica con l'intenzione di massimizzare l'efficienza del servizio offerto.

Per il caso di studio è stata selezionata una zona dove circolano 3 linee diverse fornite di dispositivi GPS che viene gestita dal DSS insieme al software di sorveglianza già esistente, delegando a quest'ultimo il compito di riconoscere le anomalie e comunicarle al DSS. Sono stati usati 5 tipi di agenti per implementare il DSS, di cui tre sono gli stessi di quelli usati nell'altro esperimento e sono i Data Agent, Action Implementation Agent e User Interface Agent.

Gli altri due tipi di agenti sono il **Line Management Agent (LMA)** e il **Coordination Facilitator (CF)**. Il primo ha un comportamento analogo al PDA nella gestione del traffico; identifica i problemi a partire dei dati trasmessi per il DA per poi facilitare l'informazione all'utente tramite l'UIA.

I problemi rilevati dal LMA a volte vengono trasmessi anche al CF, un agente unico nel sistema. Il CF ha la responsabilità di gestire tutti i LMA con l'obiettivo di farli collaborare. Ad esempio, se una linea ha bisogno di un veicolo, il LMA corrispondente fa una richiesta al CF per ottenere un veicolo, che inizia una negoziazione con gli altri LMA per ottenere il prestito. Purtroppo, il prestito di un autobus è una decisione importante ragione per la quale si chiederà sempre l'approvazione dell'operatore tramite l'User Interface Agent.

Riassumendo, il primo esperimento mostra come affrontare la gestione di una area suddividendola in zone che devono collaborare per affrontare il problema. In secondo luogo, l'esperimento sulla gestione degli autobus mostra come grazie alla negoziazione fra i diversi agenti si possono risolvere i problemi e ottenere un risultato migliore.

Capitolo 4

Analisi e design del sistema

4.1 Analisi dei requisiti e casi d'uso

La fase precedente all'implementazione in cui si cercano, elencano e studiano le funzionalità che dovrà realizzare il progetto, viene detta *analisi e studio dei requisiti*. In questa sezione si descrivono i requisiti dell'applicazione e le interazioni fra utenti e sistema, detti casi d'uso.

Per rendere lo studio dei requisiti più comprensibile, essi sono stati strutturati a partire dello schema mostrato di seguito, basato sul modello utilizzato nella metodologia Volvere [Atlantic Systems Guild Inc., 1999] per cui si descrive i passi e modelli a utilizzare nella fase di analisi di un sistema software. Infine, il modello elaborato per studiare i requisiti segue la struttura mostrata di seguito:

Requisito (Tipo di requisito) #(Numero identificativo)	
Descrizione:	Descrizione del requisito funzionale.
Motivazione:	Motivazione che ha portato a specificare questo requisito.

Tabella 4.1: Modello per strutturare i requisiti.

4.1.1 Requisiti funzionali

Questi requisiti descrivono le interazioni del software con il mondo esterno e le funzionalità che dovrà offrire.

Requisito funzionale #1	
Descrizione:	Permettere all'utente di definire il numero di autobus che possono entrare nella zona del porto e la sua capacità.
Motivazione:	Con questa caratteristica si possono studiare diverse opzioni per trasportare i croceristi dal porto al centro della città.

Requisito funzionale #2	
Descrizione:	Offrire all'utente la possibilità di modificare le destinazioni assegnate agli autobus uscenti dal porto.
Motivazione:	In questo modo si possono distribuire i croceristi fra i diversi punti della città e osservare come cambiano le previsioni.

Requisito funzionale #3	
Descrizione:	Lasciare scegliere all'utente quali crociere avranno autobus per trasportare i croceristi.
Motivazione:	Nel caso che si produca un arrivo massivo di croceristi nello stesso tempo e non sia possibile trasportare tutti i turisti allo stesso tempo per le limitazioni del porto, l'utente dovrebbe poter scegliere a chi assegnare gli autobus.

Requisito funzionale #4	
Descrizione:	L'utente deve poter consultare lo stato delle aree per poter fare predizione.
Motivazione:	Che l'utente disponga di questa informazione è necessario per molte delle decisioni che dovrà prendere, così come per prevedere le congestioni.

Requisito funzionale #5	
Descrizione:	Offrire la possibilità di modificare la data e il periodo di tempo della previsione.
Motivazione:	Requisito indispensabile per organizzare la città e fare le previsioni dettagliatamente.

Requisito funzionale #6	
Descrizione:	Permettere di aggiungere o cancellare eventi nella città.
Motivazione:	In questo modo si possono fare previsioni più realistiche.

Requisito funzionale #7	
Descrizione:	Permettere all'utente di generare percorsi con il trasporto pubblico e privato.
Motivazione:	Requisito che serve per pianificare le gite turistiche.

Requisito funzionale #8	
Descrizione:	Disporre di un metodo perchè l'utente possa generare le gite selezionando i croceristi, i punti d'interesse con l'orario e il mezzo di trasporto ad utilizzare.
Motivazione:	Grazie a questa funzionalità l'utente può generare diversi percorsi turistici da raccomandare alle agenzie delle crociere.

4.1.2 Requisiti non funzionali

I requisiti non funzionali a loro descrivono tutte le caratteristiche del software che non riguardano né il funzionamento né il modo d'interagire con il mondo reale.

Requisito non funzionale #1	
Descrizione:	Il software si deve poter estendere per aggiungere nuove funzionalità.
Motivazione:	Dato che il sistema modella parte del mondo reale rimangono fuori di questo progetto altrettanti aspetti. Di conseguenza è probabile che nel futuro si aggiungono altri servizi.

Requisito non funzionale #2	
Descrizione:	Indipendenza del software rispetto la città in cui viene utilizzato.
Motivazione:	Uno degli obiettivi del progetto è che si possa usare in qualsiasi città.

Requisito non funzionale #3	
Descrizione:	Indipendenza del software rispetto la piattaforma.
Motivazione:	Il software sarà utilizzato da diverse città che possono disporre di sistemi informatici molto diversi.

Requisito non funzionale #4	
Descrizione:	Permettere la interazione di diversi utenti con il sistema in modo concorrente e da luoghi diversi.
Motivazione:	L'informazione a gestire è voluminosa e non può essere trattata da un unico utente.

Requisito non funzionale #3	
Descrizione:	Usare soltanto formati standardizzati per i dati.
Motivazione:	Alcuni dati si possono trovare in formati diversi, come le coordinate GPS. Per evitare errori il sistema deve essere pronto a trattare diversi formati standardizzati senza imporre restrizioni.

4.1.3 Casi d'uso

La rappresentazione delle interazioni fra il sistema e gli utenti si studia e rappresenta tramite i casi d'uso, dove ogni interazione ha un obiettivo.

Il profilo dell'utente individuato durante l'analisi del sistema da progettare richiede multiple persone esperte nella gestione della città con conoscenze informatiche basiche. Essi devono far parte del personale in carico di gestire la città. I suoi compiti, già descritti nella introduzione, comprendono fra altri la prevenzione delle congestioni nella città o la comunicazione con altre entità come il gestore del trasporto pubblico. La motivazione di questa scelta radica nella decisione pressa all'inizio di progettare il sistema unicamente per l'autorità della città.

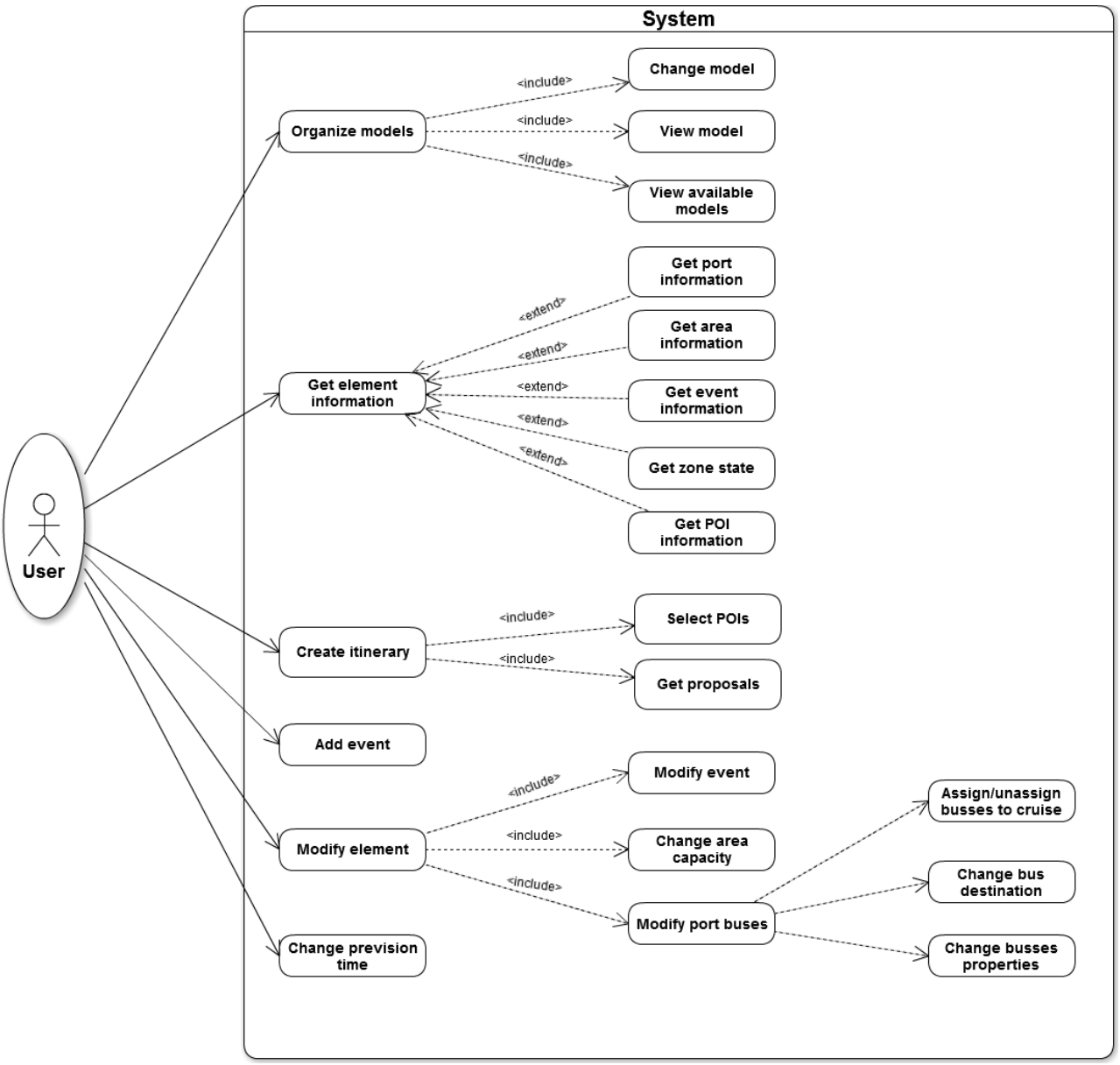


Figura 4.1: Diagramma dei casi d'uso

I casi d'uso si possono suddividere in diversi grandi gruppi. Per cominciare troviamo la gestione dei modelli sotto il caso d'uso *Organize models* per cui si possono consultare o modificare i modelli utilizzati per organizzare l'informazione. Dopodiché, un'altra interazione molto abituale sarà la richiesta d'informazione di qualsiasi elemento che interviene nel problema, tramite il caso d'uso *Get element information.*

Uno dei compiti dell'utente sarà raccomandare gli itinerari da seguire per le gite organizzate dalle crociere. Per farlo è prevista la interazione di *Create Itinerary*, suddivisa nella selezione di punti d'interesse da visitare e la richiesta per ottenere diverse proposte di gite organizzate

per visitare i posti scelti.

Aggiungere la presenza di un evento nella città può servire per studiare le conseguenze derivanti da questo, motivo per cui ci sarà la interazione *Add event*. Un'altra interazione semplice è quella di modificare la data e ora della previsione che si simula grazie a *Change prevision time*.

Per ultimo, è abituale che l'utente debba modificare dati come quelli degli autobus del porto per studiarne gli effetti. Scopo che si raggiunge con l'interazione generica *Modify element* che include anche l'azione di modificare l'informazione degli autobus.

4.2 Design del sistema

L'architettura scelta per approcciare il design del sistema è una suddivisione del sistema in componenti seguendo le idee studiate nell'ambito delle architetture dei DSS nella sezione 2.5.

Così il DSS si è strutturato in gestore di dialoghi, sistema di risoluzione del problema, gestore dei dati e in gestore di modelli. Da un punto di vista fisico le componenti si trovano in posti diversi tranne il gestore di modelli e il gestore di dati dato che entrambe interagiscono con dispositivi di immagazzinaggio fisico. Invece, il sistema di risoluzione genera processi per realizzare computi eseguiti nel processore. In questo modo si può parlare anche di una suddivisione del sistema in 3 strati (presentation, process e data layer) seguendo il modello multistrato utilizzato ampiamente per disegnare i sistemi client-server come questa.

L'architettura proposta viene esemplificata di seguito nella Figura 4.2 che offre una visione globale del sistema.

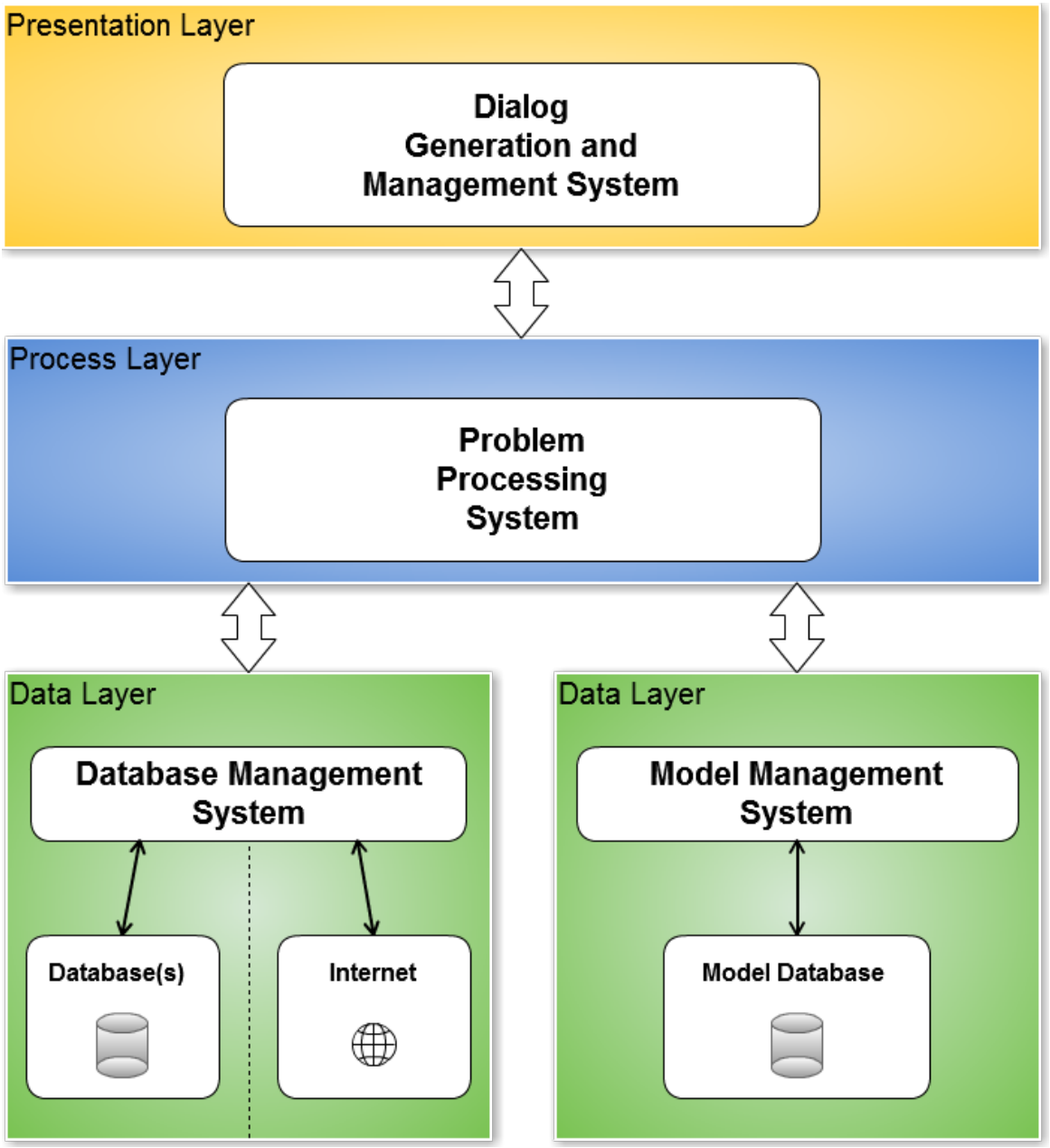


Figura 4.2: Architettura globale.

Con questa struttura basata nella divisioni in moduli si ottengono vantaggi per aggiornare, individuare o risolvere errori e si produce un codice più strutturato e intelligibile. Di seguito si studierà il design più in profondità analizzando i diversi strati separatamente.

4.2.1 Data layer

La componente Database Management System (DBMS) è in carico d'accedere al database e di comunicare tramite internet con i servizi che offrono dati real-time, come il trasporto pubblico.

Comunica direttamente con la componente per la risoluzione del problema, appartenente alla *Process layer*, per fornire i dati necessari per realizzare i calcoli.

L'altra componente che si trova in questo strato è il Model Management System (MMS), che gestisce il database di modelli e permette al *Process layer* di interagire per ottenere informazione oppure utilizzarli per strutturare i dati. Le informazioni immagazzinate nel modello sono accessibili e utilizzabili in tempo di esecuzione, e una volta che il programma termina ogni dato che non sia stato salvato nel database del DBMS viene perso.

Per il design del database del DBMS si è scelto di partire da un database relazionale e disegnare la struttura a seguire mediante un tipo de design che prende il nome di *modello dei dati* elaborato nella fase di design. L'obiettivo è di formalizzare quali dati devono conservarsi e il tipo di relazione che mantengono fra di loro, per così ottenere un modello a cui riferirsi per realizzare la implementazione nella fase successiva.

Modello dei dati

Il modello ottenuto ha 19 classi che diventeranno 19 tabelle. Inoltre, esistono le classi associative che rappresentano le relazioni fra due tabelle, e ogni relazione viene formalizzata in una tabella. Nel modello vengono rappresentate soltanto le classi associative che apportano informazione aggiuntive al database, tuttavia il modello risultante è di gran dimensione.

Per questa ragione si è suddiviso il modello in 3 parti, raggruppando le classi che avevano una tematica in comune.

Gestione delle crociere

Questa parte del database rappresenta le informazioni del porto, le crociere e i loro arrivi alla città, così come le richieste per realizzare un itinerario turistico.

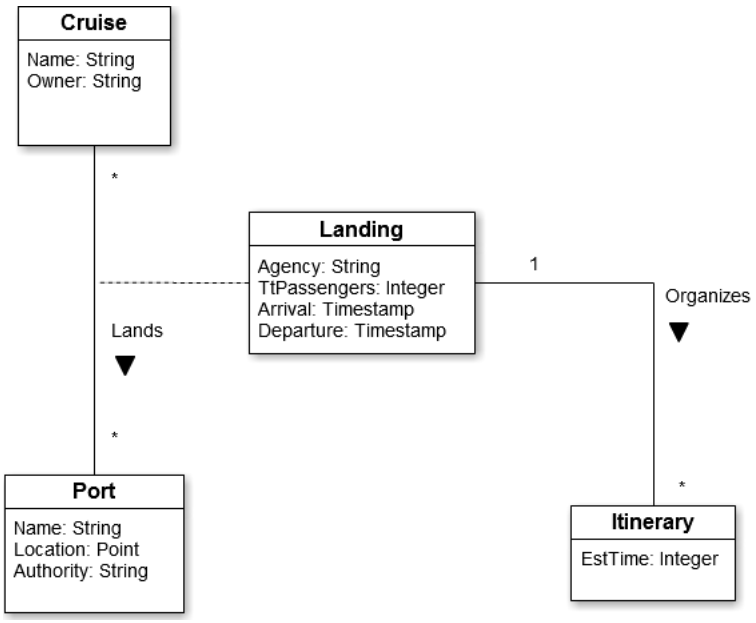


Figura 4.3: Modello dei dati - Crociere.

Nella Figura 4.3 si illustrano le classi di questa parte del database e le loro relazioni. La classe *Cruise* mantiene le informazioni sulle crociere che sono state introdotte nel sistema. Quando queste devono sbarcare in città, lo fanno al *Port* e si crea una relazione tra di loro, e di conseguenza si aggiunge l'informazione inerente all'arrivo nella classe associativa *Landing*.

Il sistema offre la possibilità che una crociera faccia richieste per visitare la città, organizzando diversi itinerari. Questi si conservano nella parte del modello che verrà spiegato in seguito ed è relazionato la gestione della crociere mediante la classe *Itinerary*.

Gestione della città e percorsi

L'obiettivo di questa sezione del modello è immagazzinare le informazioni della città come la suddivisione in aree, gli eventi che hanno luogo e le informazioni dei punti d'interesse turistici. Dall'altro lato, conserva le informazioni degli itinerari richiesti nella città cosi come i loro percorsi.

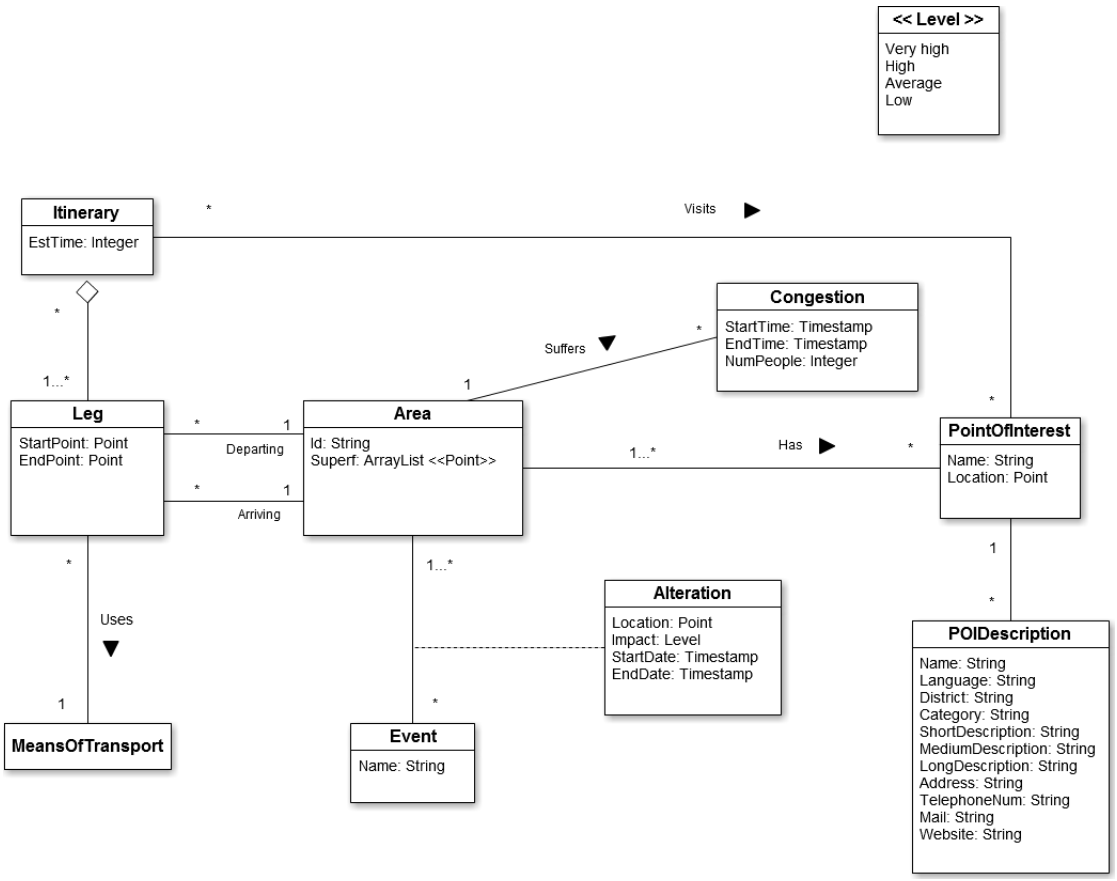


Figura 4.4: Modello dei dati - Città e percorsi.

Un itinerario richiesto da una crociera ha lo scopo di visitare alcuni punti d'interesse. Questa informazione è salvata seguendo la struttura mostrata nella Figura 4.4 in cui si può osservare la relazione fra l'itinerario e la classe *PointOfInterest*. Ogni itinerario a sua volta si può scomporre in diverse tratte, chiamate *Leg* nel design, con la restrizione che ogni tratta utilizza un unico mezzo di trasporto.

Ogni tratta parte da un'*Area* e arriva a un'*Area*, che possono anche coincidere. Questa informazione è rilevante per sapere lo stato della città (traffico, persone...) in cui si troverà ogni tratta.

Si può parlare dell'informazione sui mezzi di trasporto come un altra area del problema, motivo per il cui è rappresentata in una altra parte del modello, collegato tramite le classe *Area* e *MeansOfTransport* che viene descritta di seguito.

Gestione del trasporto

Nella ultima parte del modello si rappresenta la conoscenza che c'è del trasporto nella città, partendo dell'assunzione che ci sarà la metro e il bus, sia privato che pubblico anche se l'esistenza di questi non è una restrizione per la implementazione. Infatti, il modello contempla la possibilità che non si trovano questi mezzi o che ci siano mezzi diversi.

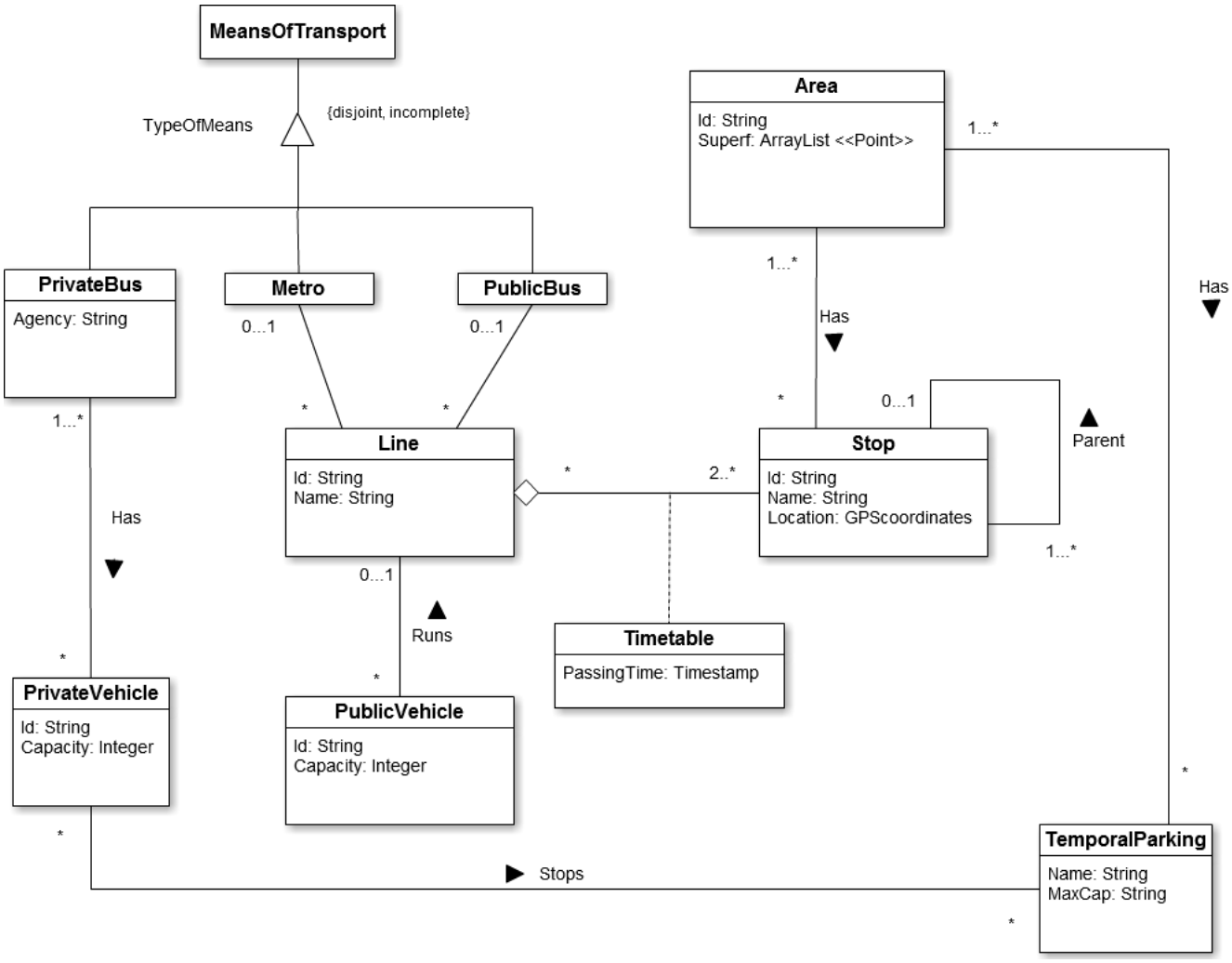


Figura 4.5: Modello dei dati - Trasporto.

Dato che ogni città può disporre di mezzi di trasporto diversi si è adottata la soluzione che permette di aggiungere o rimuovere tipi di mezzi facilmente tramite la specializzazione della classe *MeansOfTransport*, illustrata nella Figura 4.5.

Dall'altro lato, l'informazione basilare che abitualmente è disponibile nelle città e quella delle linee, delle fermate i degli orari (classe *Line*, *Stop* e *Timetable* rispettivamente). Inoltre, sarebbe

utile disporre della capacità dei veicoli che percorrono le linee per così poter calcolare la capacità della rete.

Un'altra parte del problema è l'organizzazione del trasporto dei turisti dal porto ai punti nevralgici della rete di trasporto oppure ai punti d'interesse per fare una gita. Il mezzo utilizzato in questi casi è solitamente un bus privato, perciò sono state incluse nel modello di dati le classi *PrivateBus* e *PrivateVehicle*.

In genere, la maggior parte delle città turistiche hanno parcheggi per gli autobus nelle zone più visitate, modellata dalla classe *TemporalParking*.

L'organizzazione del trasporto in questo dominio si rende necessaria al fine di spostarsi da un'area all'altra e visitare i posti turistici. Quindi si sono rappresentate le possibilità di trasporto in un'area tramite le relazioni *Area-Stop* e *Area-TemporalParking*.

4.2.2 Process layer

Da un punto di vista funzionale il sistema di calcolo del problema è la unica componente dello strato. Il suo design è soggetto alle tecnologie e paradigmi scelti per l'implementazione che possono essere molto diversi. In questa tesi si è deciso d'affrontare la implementazione basandosi su un sistema multi-agente previamente disegnato.

Gli agenti che formano il sistema multi-agente si sono classificati in 3 gruppi a seconda del tipo di compito affidato. Nel primo gruppo (*Interface Agents*) si trovano gli agenti che interagiscono con l'utente chiamati *User Interface Agent*. Hanno la funzionalità di trasmettere le richieste e dati introdotti dall'utente agli agenti e di fornire l'informazione proveniente dal sistema alla interfaccia con cui interagisce l'utente.

A continuazione ci sono i *Management Agents*, che devono organizzare l'informazione proveniente degli altri strati per soddisfare le richieste. Conseguentemente è la parte principale del sistema multi-agente.

Per ultimo troviamo gli agenti che sono in carico di comunicare con la componente *DBMS* per interrogarla e ottenere i dati che servono agli agenti.

Un ulteriore aspetto a considerare sono i *Peripherial Agents* che non formano parte del dominio del problema ma sono agenti progettati dalla FIPA (*Foundation for Intelligent Physical Agents*) per far in modo che il sistema possa funzionare. Questi agenti comunicano con tutto il sistema in modo che tutti gli agenti possano comunicare con il *MMS* per utilizzare i modelli.

Tutti questi aspetti, così come i diversi tipi d'agenti individuati, sono stati illustrati nella Figura 4.6.

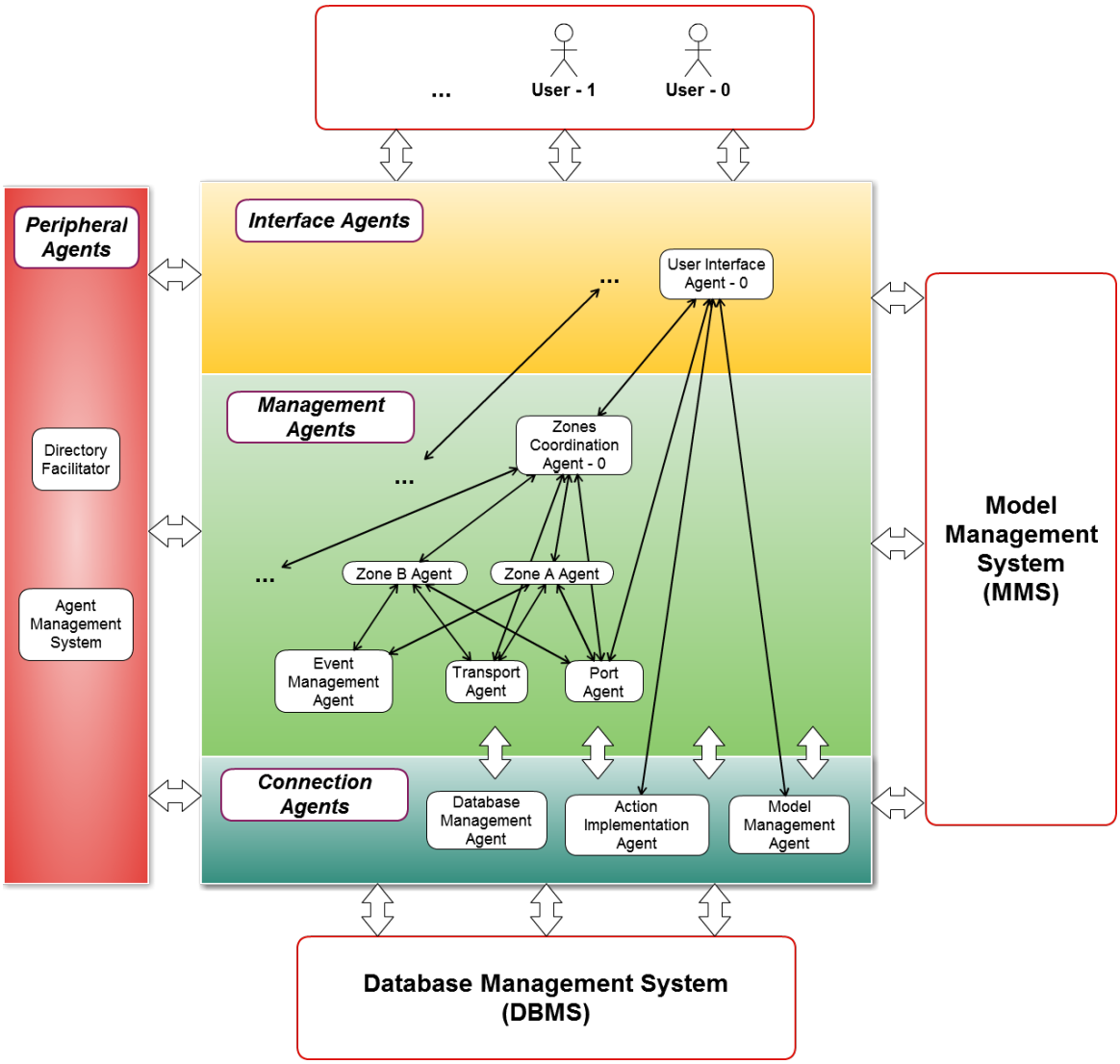


Figura 4.6: Architettura del Sistema Multi-agente.

Osservando la Figura 4.6 si riesce a differenziare le diverse parti del sistema multi-agente e le interazioni con le altre componenti del DSS. Grazie al schema si può comprovare come la maggior parte degli agenti sono concentrati nel gruppo di *management agents*, che hanno l'obiettivo di organizzare le informazioni e risolvere i problemi.

Di seguito si procede a fare un analisi più in profondità dei diversi tipi di agenti che conformano il sistema.

Peripheral Agents

Sono indipendenti dal dominio in cui lavora il DSS, non partecipano nella risoluzione del problema e hanno lo scopo di organizzare il sistema multi agente. In quest'ambito troviamo due agenti specificati seguendo le direttive della FIPA sulla strutturazione di un SMA studiate nella sezione 3.6.

Il primo agente della categoria è il **Directory Facilitator** (DF) che offre agli altri agenti un metodo per cercare e offrire servizi. Il secondo è l'**Agent Management System**, che viene implementato come un unico agente con l'obiettivo di gestire gli altri agenti.

Interface Agents

Disegnati con l'obiettivo di permettere che l'utente possa interagire con il sistema multi-agente. L'unico agente appartenente a questa categoria è l'**User Interface Agent** (UIA), dedicato ad ricevere le richieste dell'utente e mostrare le informazioni. Pertanto, l'UIA è dedicato unicamente a un utente e si dovrà replicare per ogni utente del sistema.

Per soddisfare le richieste dell'utente, l'UIA interroga gli altri agenti come il *Zones Coordination Agent* o il *Port Agent*. Inoltre, deve offrire la possibilità di modificare i dati del Database Management System mediante gli altri agenti del sistema. Inoltre può fare richieste al *Model Management Agent* per ottenere informazione dei modelli, modificarli o crearne di nuovi.

Management Agents

Lo scopo degli agenti appartenenti a questo gruppo è il trattamento dell'informazione per organizzare il problema. Tutti gli agenti elencati di seguito hanno la possibilità di comunicare con gli agenti del gruppo *data agents* per ottenere o introdurre nuove informazioni. Un ulteriore aspetto è che alcuni di questi agenti usufruiscono dei modelli proporzionati dal *Model*

Management System per calcolare e decidere le azioni da intraprendere.

Port Agent: Comunica all'*User Interface Agent* le informazioni che riguardano le crociere e le proprietà del porto. Utilizza un modello per calcolare le conseguenze per il porto dovute all'arrivo delle crociere, cosi come la organizzazione e gestione del numero di autobus che possono circolare e la loro distribuzione fra le crociere. In questo modo si può, se previsto dal modello, sfruttare il sistema per offrire alle crociere la possibilità di condividere gli autobus, incrementando la efficienza.

Transport Management Agent: Comunica al *Zone Agent* le diverse opzioni di trasporto disponibili nella zona e i loro dati elaborati, come la capacità massima di persone. Un altro dei suoi compiti è ottenere ed informare dei percorsi disponibili tra diverse aree al *Zones Coordination Agent* quando richiesto.

Zone Agent: Fornisce al *Zones Coordination Agent* le informazioni riguardanti l'area che controlla come i punti d'interesse, lo stato del traffico o il numero di persone previste nell'area in un'ora data. In caso di problemi di congestione ne informa gli agenti che sorvegliano le zone vicine.

Zones Coordination Agent: Gestisce le informazioni fornite dai diversi agenti e le elabora grazie a un modello e poi le comunica all'*User Interface Agent*. Le informazione provengono da diversi tipi d'agenti, quali i *Zone Agent*, *Port Agent* e il *Transport Agent*.

Connection Agents

Gli agenti appartenenti a questa categoria hanno lo scopo di comunicare con elementi che non formano parte dal sistema multi-agente.

Il primo esempio è la comunicazione fra il sistema multi-agente e la componente **Database Manager System** per ottenere i dati del problema, modificarli oppure inserirne di nuovi. Per svolgere questo compito si è disegnato un unico tipo di agenti, il *Database Management Agent*, che sarà l'unico agente capace di comunicare con il DBMS. Questo agente si dovrà replicare per poter processare tutte le richieste, e il numero di repliche dipende da diversi fattori come la quantità d'informazione o le caratteristiche del sistema hardware.

Un altro dei suoi compiti è aggiornare i modelli utilizzati dagli altri agenti con le informazioni fornite al DBMS. In questo modo la comunicazione sarà informativa e lo scambio di informazioni avverrà attraverso l'uso dei modelli.

Un altro agente in carico di interagire con una componente del DSS è il **Model Management Agent** che processa le richieste dell'*User Interface Agent*. Le richieste possono riguardare le informazioni riguardo i modelli disponibili, consultare i modelli utilizzati per gli agenti oppure cambiarli per altri che non sono in uso. Per soddisfare le richieste, l'UIA deve comunicare con il Model Management System che è in carico di gestire i modelli.

Infine, si è deciso di includere l'agente **Action Implementation Agent** a cui viene affidata la implementazione delle decisioni presse dall'utente. Motivo per il cui l'agente dovrà aspettare di ricevere comunicazioni dall'*User Interface Agent*. In questo contesto, la sua funzionalità sarebbe di comunicare alle altre entità che formano parte del problema come le agenzie di trasporto pubbliche, le agenzie delle crociere o il corpo di polizia, la necessità di portare a termine una azione.

4.2.3 Presentation layer

Permette all'utente di interagire con il sistema grazie alla lettura dei dati visualizzati tramite la interfaccia, che poi vengono mandati al *Process layer*. A sua volta genera i messaggi e le interfacce necessarie per mostrare all'utente le informazione elaborate dal sistema.

Il lavoro svolto da questo strato è affidato alla componente *Dialog Generation and Management System* del DSS. Per disegnare questa componente si è adottato il modello di design tipico delle applicazioni client-server noto come *Model-View-Controller*, esemplificato a continuazione nella Figura 4.7.

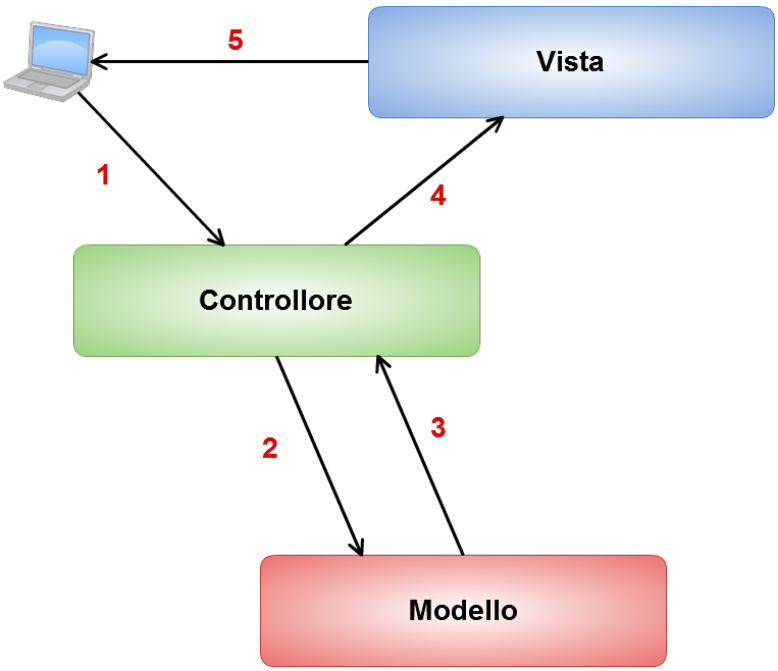


Figura 4.7: Modello Model-View-Controller.

Con questo approccio si delega al modello la elaborazione dell'informazione da mostrare per ogni richiesta dell'utente. Dopodiché le informazioni si rappresentano e organizzano in unità chiamate viste, elaborate dal controllore. È quest'ultimo a decidere per ogni richiesta dall'utente quali viste saranno mostrate all'utente.

Capitolo 5

Implementazione del prototipo

Partendo dal design proposto nella sezione 4.2 si è approcciato lo sviluppo del prototipo. Come proposto da Minch e Burns [Minch and Burns, 1983], la strutturazione globale adottata è stata quella di suddividere il sistema in 4 componenti apportando le modifiche opportune, che poi vengono raggruppate in 3 strati. In questo capitolo verranno descritte le diverse componenti e la loro implementazione in modo dettagliato.

5.1 Data layer

Questo strato è formato dal Database Management System (DBMS) e dal Model Management System (MMS), dato che le due componenti hanno funzionalità ben diverse sono state implementate indipendentemente.

Il Database Management System (DBMS) deve gestire tutti i dati del sistema, più specificamente deve fornire accesso al database, gestire l'acquisizione dei dati tramite servizi SOAP/REST e la loro inserzione nel database. Postgresql è stata scelta come la tecnologia da usare per implementare la base di dati relazionale [Group, 2015]. La scelta è stata motivata per la facilità con cui si possono aggiungere estensioni per ottenere le funzionalità desiderate.

Precedentemente è stata individuata la necessità che il database possa trattare dati geo-spaziali, per questo motivo si è aggiunta la estensione Postgis al database. Questa estensione permette

di lavorare con dati geo-spaziali in un modo molto trasparente grazie alle funzioni che apporta al sistema per immagazzinare oggetti geo-spaziali. Il database riceve gli oggetti rappresentati mediante il linguaggio *Well-known text* (WKT) e li trasforma in un formato interno per immagazzinarli.

```
'POLYGON((41.405606 2.160012, 41.409340 2.171255, 41.404769 2.182070,
41.398203 2.173401, 41.405606 2.160012))'
```

Figura 5.1: Area espressa con il linguaggio *Well-known text* (WKT).

Per strutturare il database si è proposto un modello elaborato in UML nella fase di design 4.2.1 realizzato con 31 tabelle per rappresentare i dati e le loro relazioni. Per formulare le operazioni da eseguire sul database si è implementato un gestore in Java. Il gestore comincia con generare le richieste che poi sono inviate al driver *Java Database Connectivity* (JDBC) che traduce le operazioni, rendendole indipendenti della versione e dal tipo di database.

L'altra fonte di informazione del sistema è internet, attraverso i servizi offerti dalle agenzie. Ad esempio, le agenzie di trasporto pubblico generalmente abilitano un servizio SOAP/REST per ottenere dati in tempo-reale. Questi servizi utilizzano il paradigma cliente-servitore, basandosi sul protocollo http e il linguaggio xml per fare arrivare le informazioni ai clienti. Purtroppo, nella implementazione del prototipo si è deciso di basare il sistema unicamente sui dati inseriti nel database e lasciare questa parte come un futuro sviluppo.

Per gestire sia gli ipotetici servizi SOAP/REST sia il gestore dei database si è optato per implementare un gestore DBMS in Java. In questo modo si semplifica la comunicazione con il *Process layer* che sarà implementato con lo stesso linguaggio, Java. Questo gestore chiamato *Data Manager* comunica con il gestore dei database e dovrebbe interagire con il programma che gestisce i servizi SOAP/REST in un futuro. In questo modo ogni strato dell'architettura è indipendente dello strato da cui provengono i dati ed è trasparente all'esterno del sistema. Si è sviluppata una interfaccia JAVA per gestire questa componente che in futuro dovrà interfacciare diversi DBMS. La Figura 5.2 proporziona una visione globale del modulo e delle sue parti.

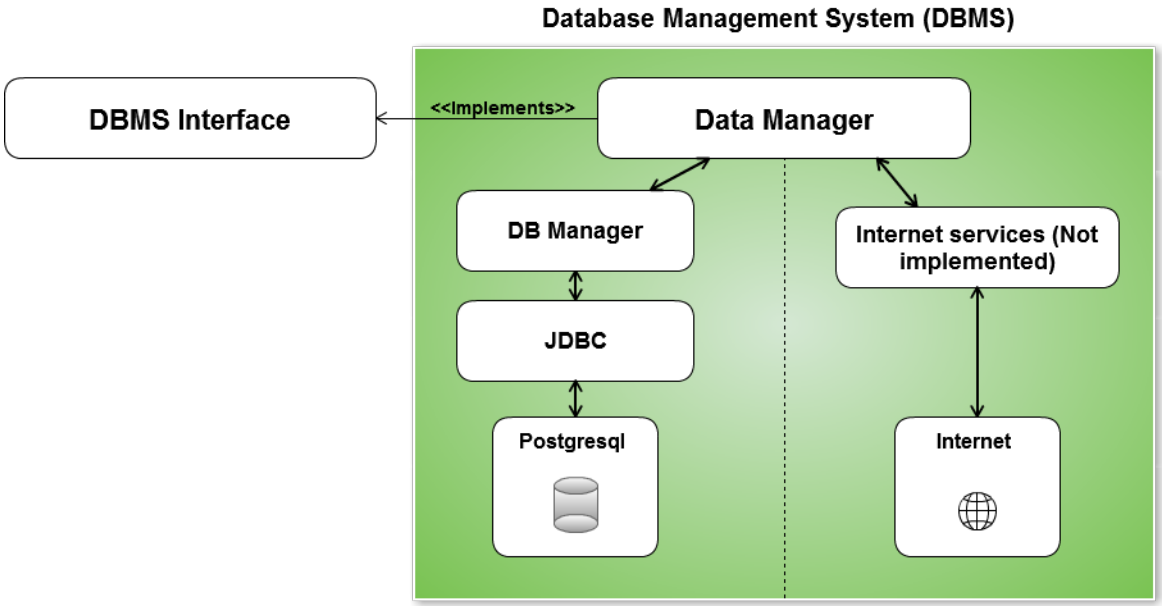


Figura 5.2: Struttura implementata del Database Management System.

Nel prototipo si è scelto di non affrontare la generazione di percorsi fra diverse zone, che si potrebbe delegare a servizi specifici esterni al sistema. Tuttavia i dati del trasporto sono stati immagazzinati nel database con l'obiettivo di sfruttarli nelle previsioni di congestione nelle zone.

Il secondo componente del DSS è il Model Management System (MMS) che gestisce l'accesso e l'uso dei modelli utilizzati per elaborare i dati. Il DSS può disporre di diversi modelli e usarli a seconda del contesto, ragione per la quale è importante avere un MMS e conservare la indipendenza dei modelli rispetto alle altre componenti del sistema.

Il gestore di modelli (MMS) è stato implementato mediante linguaggio Java e offre la possibilità d'ottenere informazione riguardo i modelli presenti oppure ottenere accesso a loro per usarli. Poiché l'applicazione implementata è un prototipo, si sono implementati 3 modelli diversi pero senza l'infrastruttura di un database per immagazzinarli.

I 3 modelli sono stati implementati con il linguaggio a regole *CLIPS* e hanno lo scopo di elaborare i dati inseriti [CLIPS, 2015]. Un primo modello è quello utilizzato per il porto, che offre diverse funzionalità:

Gestione degli arrivi di crociere: Fornisce informazione riguardo i diversi arrivi in una giornata, in una ora oppure tutta l'informazione disponibile di una crociera qualsiasi.

Gestione autobus: Permette di limitare il numero d'autobus che possono entrare nel porto e modificare la loro capacità per poi calcolare quanti autobus sono necessari per portare i croceristi fuori dal porto. Nel presupposto che non si possano portare tutti i croceristi con autobus, si fa una assegnazione automatica degli autobus fra le crociere, in modo che tutti i turisti della stessa crociera abbiano o meno il servizio degli autobus. Questa è una proposta automatica, perciò si può modificare quali crociere disporranno di autobus e calcolare di nuovo le conseguenze. Il modello permette di essere interrogato in diversi modi per ottenere queste informazioni.

Conseguenze per la città: Il modello genera le previsioni per sapere quale sarà il numero di croceristi presenti nella città ad ogni ora. Un'altra funzione è quella di permettere di modificare la percentuale di croceristi che sceglieranno di spostarsi tramite autobus dal porto.

Un altro modello è quello utilizzato da ogni zona in cui viene suddivisa la città, rappresentando l'informazione degli elementi presenti nella zona e l'informazione riguardo il suo stato di congestione. Più specificamente si possono raggruppare le informazioni modellate in diversi gruppi elencati in seguito.

Gestione degli elementi presenti nell'area: Modella diversi elementi, per primo i punti d'interesse presenti nella zona e le loro informazioni. Inoltre, mantiene le informazioni delle diverse fermate di trasporto pubblico presente nell'area e infine gestisce l'informazione degli eventi che si terranno nell'area in un periodo di tempo determinato.

Elaborazione dello stato della zona: Lo stato in cui si troverà la zona viene calcolato per ogni ora del giorno basandosi sull'informazione disponibile. Inoltre, ogni zona ha una soglia che rappresenta il numero massimo di persone che possono circolare a piedi senza causare congestione, dato che si può modificare nel sistema.

A partire da tutte le informazioni disponibili si calcolano diversi parametri come la raccomandazione di prendere o meno il trasporto pubblico, se il traffico sarà bloccato oppure quale sarà la congestione della zona basandosi sul numero di persone previste e la soglia

della zona. La presenza di congestione viene classificata in 4 livelli d'intensità, da bassa a molto alta.

Interazione con le altre aree: Il modello prevede che l'area possa ricevere più persone, proveniente delle zone vicine oppure dal porto e che di conseguenza si debbano ricalcolare gli stati. A sua volta, quando nell'area si prevede una alta affluenza di persone si allertano le aree vicine del possibile arrivo di persone.

Infine, è necessario un modello che relazioni le diverse parti della città, ovvero le aree e il porto. A questo scopo si è elaborato un ulteriore modello che prende diverse informazioni del porto come il numero di croceristi e gli autobus che escono dal porto. Con questa informazione il modello consente di distribuire automaticamente gli autobus fra le diverse zone basandosi sulle loro caratteristiche, come il livello di congestione o il numero di fermate di trasporto pubblico. Successivamente si prevede come sarà la distribuzione durante la giornata dei turisti fra le diverse aree, per prevedere la affluenza di turisti in ogni zona. Infine, il modello offre la possibilità di modificare le aree a cui saranno destinati gli autobus manualmente per osservare quali sarebbero gli effetti per la città.

```
(defrule MAIN::inistrike
  (declare (salience 30))
  (event (id ?id) (name ?name) (type ?t&:(= (str-compare ?t "Strike") 0)) (starts ?h1) (ends ?h2) (nump ?np))
  =>
  (bind ?h ?h1)
  (while (< ?h ?h2)
    do
      (assert (no-traffic (hora ?h)))
      (assert (no-transport (hora ?h)))
      (assert (increase (hora ?h) (nump ?np)))
      (bind ?h (+ ?h 1))
  )
  (assert (newevent ?id ?h1 ?h2 ?np))
)
```

Figura 5.3: Estratto dal modello di zona.

Nel frammento di codice mostrato nella Figura 5.3 si può osservare una porzione del modello che organizza l'informazione di una zona qualsiasi. Queste righe definiscono una regola mediante il linguaggio CLIPS in cui si definiscono le conseguenze che avrebbe uno sciopero nella zona in momento dato.

Lo scopo di questi modelli è dimostrare il corretto funzionamento del sistema e permettere di studiare le conseguenze delle decisioni, che è proprio uno dei requisiti del DSS. Per questa ragione alcuni parametri sono arbitrari e senza una solida base oppure i modelli non sfruttano tutta l'informazione disponibile. In una futura implementazione del sistema sarebbe necessaria una comunicazione con i diversi esperti del dominio del problema per sviluppare dei modelli più precisi.

5.2 Process layer

La parte logica del sistema, detta Problem Processing System (PPS) è costituita unicamente dal sistema multi-agente in carico d'organizzare l'informazione, di gestire la comunicazione e di soddisfare le richieste dell'utente. Il linguaggio scelto per l'implementazione di questo strato è Java, corredato librerie necessarie. La prima componente utilizzata è stata Jade, un insieme di librerie che permettono d'implementare facilmente un sistema multi-agente e tutte le particolarità come la negoziazione. Inoltre, l'uso di questo software permette di sviluppare un sistema multi-agente che soddisfa i protocolli stabiliti dalla FIPA.

La scelta alternativa a Jade era l'insieme di librerie AIMA [AIMA, 2009], focalizzate sullo sviluppo di software nel campo della intelligenza artificiale. Il supporto che offre Jade nella implementazione dei sistemi multi-agente è molto più ampia che quella d'AIMA, oltre ad essere più intuitiva. Questi argomenti hanno portato a scegliere Jade per portare a termine l'implementazione.

Jade è stato sviluppato da Telecom Italia ed è diventato open-source nel 2000. È composto da un set di librerie che permettono di sviluppare agenti semplici e anche complessi. In Jade si organizzano i compiti degli agenti in *behaviours* in cui si definiscono i diversi compiti che l'agente dovrà realizzare. I compiti possono essere vari come registrare un servizio al *Directory Facilitator*, realizzare un computo, comunicare con un altro agente oppure aspettare l'arrivo di un messaggio concreto.

Ad esempio, un comportamento comune nel prototipo è realizzare un computo che richiede informazioni che l'agente non ha. Perciò dentro il comportamento l'agente deve contattare

ulteriori agenti e completare l'informazione necessaria.

Inoltre, Jade utilizza i messaggi FIPA-ACL e opzionalmente si possono sovrascrivere i protocolli predefiniti come le aste o protocolli per la negoziazione.

Dato che l'architettura multi-agente proposta nella fase di design non si può semplificare molto senza compromettere l'operatività del sistema, si sono implementati la maggior parte degli agenti. In Figura 5.4 è riportata l'architettura del sistema multi-agente implementato specificando gli agenti e le loro relazioni:

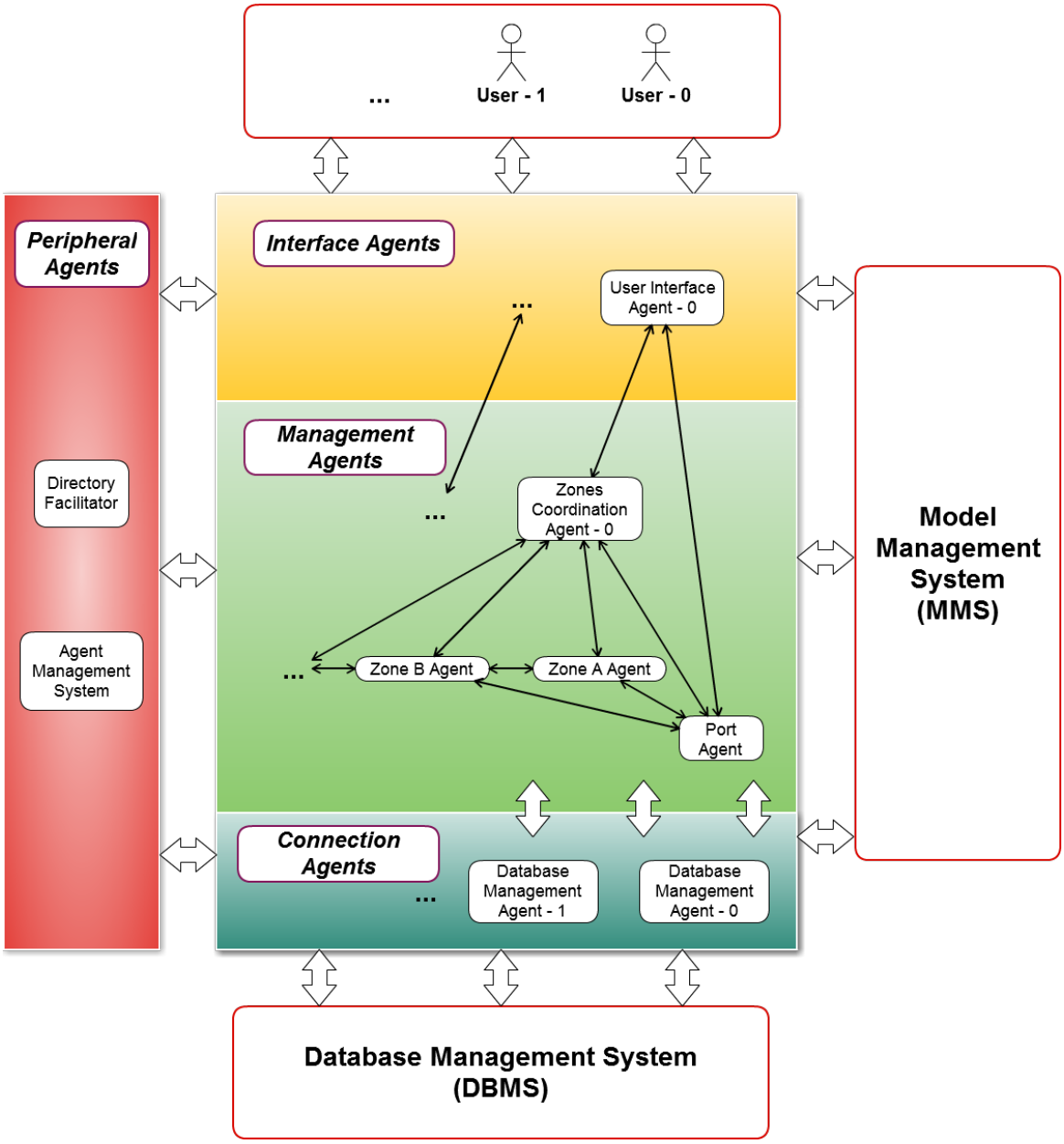


Figura 5.4: Architettura del prototipo.

La implementazione degli agenti periferici *Directory Facilitator* (DF) e *Agent Management*

System (AMS) è realizzata dalla libreria Jade che include una loro implementazione e gestione autonoma.

Ogni agente è caratterizzato dall'informazione che possiede e dai compiti che deve eseguire. Per configurare gli agenti si è implementata la funzione *setup()* che viene chiamata una volta l'agente è creato. La prima cosa a realizzare nella configurazione di un agente è riempire le informazioni private, che sono state trasmesse come parametro all'agente. La operazione più abituale è registrare i servizi offerti dal agente nel *Directory Facilitator* (DF) come mostrato di seguito nella Figura 5.5.

```
@Override
protected void setup() {
    area_id = (String) getArguments()[0];
    states = new ArrayList<>();
    neigh_areas = new ArrayList<>();
    ModelManager.getReady();
    ModelManager.addModel(area_id, ZoneModel.class);
    /**
     * Registration with the DF
     */
    DFAgentDescription dfd = new DFAgentDescription();
    ServiceDescription sd = new ServiceDescription();
    sd.setType("zone-agent");
    sd.setName(getName());
    dfd.addServices(sd);
    dfd.setName(getAID());
    try {
        DFService.register(this, dfd);
```

Figura 5.5: Registro dei servizi di un Zone Agent generico.

Per aggiungere un servizio nel DF soltanto si deve specificare il tipo di servizio offerto e il nome dell'agente, Jade automaticamente genera i messaggi fra l'agente e il DF per completare il processo.

Nella Figura 5.5 si può osservare come l'agente fa uso della componente *Model Manager System* (MMS) tramite due funzioni. La prima, *getReady()* che serve ad aspettare a che il modulo sia pronto per poter ricevere richieste. Di seguito, con la funzione *addModel()* si chiede al MMS di assegnare un modello di zona *(ZoneModel.class)* alla area identificata con il parametro *area_ id*. Questo procedimento permette di accedere al modello da qualsiasi comportamento dell'agente, soltanto con l'identificativo dell'area.

Infine, i compiti che l'agente deve portare a termine sono trattati come comportamenti dell'agente e vengono inseriti sempre attraverso la funzione *setup()*.

```
addBehaviour(new RespNeighAreas(this));
addBehaviour(new RecIncrementPpl());
addBehaviour(new RespZoneState(this));
addBehaviour(new LoadZoneInfo());
```

Figura 5.6: Configurazione dei compiti di un Zone Agent generico.

Ad esempio, nella porzione di codice 5.2 che riguarda un *Zone Agent* si osserva come sono stati aggiunti 4 compiti diversi:

RespNeighAreas: L'agente processa le richieste ricevute per sapere se l'area sorvegliata dall'agente mittente è vicina a la sua. Una volta realizzati i calcoli per sapere se la zona è vicina, l'agente risponde affermativa o negativamente. Se confermato che le aree sono vicine, l'agente che ha inviato la richiesta aggiorna sia il suo elenco privato di aree vicine sia il modello.

RecIncrementPpl: Se l'agente riceve un messaggio che lo informa di un aumento dei visitatori nell'area in una determinata ora aggiorna il modello, calcola le conseguenze aggiornando gli stati della zona e avvia le azioni necessarie come ad esempio informare un'altra area vicina che parte dei turisti potrebbero spostarsi in quell'area.

RespZoneState: Fornisce lo stato di una zona in una determinata ora in seguito a una richiesta.

LoadZoneInfo: Con questo comportamento l'agente cerca un agente con accesso al DBMS e invia una richiesta per ottenere tutti i dati della zona e introdurli nel modello. Una volta completata questa fase l'agente interroga il modello e porta a termine diverse azioni come aggiornare gli stati della zona in modo tale d'essere pronto a rispondere alle richieste sullo stato.

Per la comunicazione fra agenti si utilizza il FIPA-ACL introdotto nella sezione 3.5. Uno dei problemi rilevati nella implementazione della comunicazione è stato lo scambio di informazione

complessa come gli oggetti. La soluzione adottata è stata in alcuni casi scomporre gli oggetti in una catena di stringhe. Gli oggetti più complessi, come quelli geo-spaziali, invece sono trasformati in stringhe con il formato *JSON*.

Per implementare i protocolli di comunicazione bisogna inviare i messaggi fra gli agenti in un ordine preciso prestabilito dal protocollo. Inoltre, i messaggi sono dotati di un significato implicito che viene descritto con l'uso di *performatives*, che non sono altro che un elenco dei diversi tipi di motivi per cui l'agente invia il messaggio [Vidal, 2009].

A seconda dello scopo del mittente si impiega un protocollo o un altro, che descrive quali messaggi devono essere inviati, l'ordine e le possibile risposte. La FIPA ha standardizzato un seguito di 10 protocolli [FIPA, 2002c] che permettono gli agenti di svolgere operazioni di più alto livello come aste.

Il protocollo usato per il prototipo è quello per richiedere a un agente di eseguire un compito. Lo scambio di messaggi segue lo standard specificato da FIPA ed è mostrato in Figura 5.7.

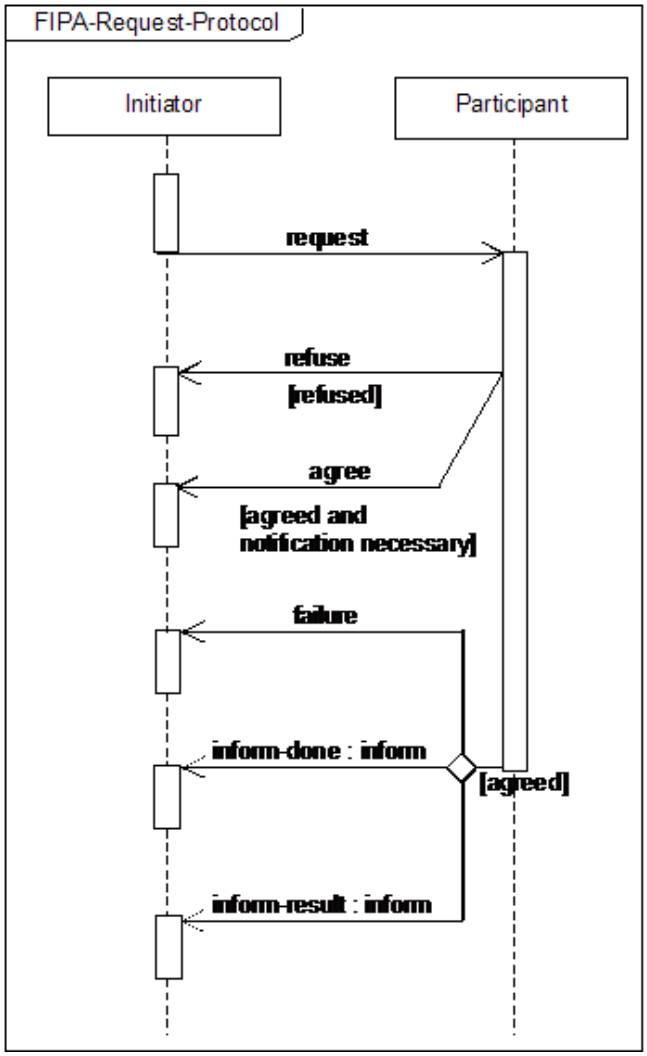


Figura 5.7: FIPA Request Protocol, Standard SC00026H.

Il protocollo permette a un agente di inviare una richiesta a un altro agente, che ha l'opzione di rifiutare. Nell'implementazione del sistema gli agenti rifiutano quando non possono soddisfare la richiesta perché non dispongono delle risorse necessarie.

Se l'agente accetta, si crea un contratto e il mittente aspetta una risposta. Il ricevente ha 3 possibilità di risposta: l'azione non è stata eseguita (*failure*), l'azione è stata eseguita come prevista (*inform-done*) l'azione è stata eseguita e il risultato dell'esecuzione viene comunicato (*inform-result*).

Inoltre, in qualsiasi momento il mittente può inviare un messaggio per cancellare la operazione che era stata richiesta di effettuare.

L'adozione di protocolli ben definiti agevola la comunicazione fra gli agenti soprattutto quando

la quantità di agenti e di messaggi da scambiare è elevata.

5.3 Interfaccia

L'applicazione è stata realizzata in modo da essere accessibile in modo concorrente da diversi dispositivi, e quindi è stata implementata come una applicazione web. Il sistema software è implementata all'interno di un server *Tomcat* [Apache, 2015] che è fornito della libreria *Java servlet* che genera e gestisce le pagine web da mostrare all'utente, utilizzando il linguaggio di programmazione Java.

Per mostrare l'informazione riguardando la città si è scelto d'includere l'applicazione *OpenStreetMap* (OSM) che contiene una mappa del mondo costruita in modo collaborativo per volontari. OSM è dotato di una *Application Program Interface* (API) che permette d'interagire con la mappa usando delle funzioni predefinite per manipolarla.

Capitolo 6

Risultati e test

Lo scopo principale del prototipo è dimostrare la utilità della proposta elaborata nella tesi per risolvere il problema della congestione derivata dai croceristi. Conseguentemente c'è stata una fase di ricerca in cui si doveva trovare una città che soddisfacesse alcune caratteristiche:

- (Mandatario) Offrire una API per consultare i dati della rete di trasporto, sia la pianificazione come i dati real-time, dati che sono essenziali per costruire la soluzione.
- (Mandatario) Offrire pubblicamente l'informazione dei punti d'interesse della città, principalmente le coordinate GPS e opzionalmente i dati statistici del numero di visitanti.
- (Pseudo-mandatario) Disporre dei dati degli arrivi delle crociere al porto agevolerebbe lo sviluppo del prototipo e permetterebbe di fare le simulazioni più precise e reali.
- (Opzionale) Avere un porto importante con molti arrivi di turisti, rende più reale la simulazione ed è più facile dimostrare l'utilità del progetto.
- (Opzionale) Essere una città molto turistica con problemi di congestione nella stagione turistica.
- (Opzionale) Avere una API per consultare lo stato del traffico.

Di tutte queste condizioni quella che restringeva di più il numero di soluzioni era la cerca di un porto che offrisse i dati delle crociere. In questo senso sono stati individuati pochi porti, e in

genere offrivano l'informazione degli arrivi delle crociere ma no il numero di passeggeri. Tutti tranne uno, il porto di Barcellona che offre un documento nel suo sito web [de Barcelona, 2015a] con le previsioni di arrivi di crociere documentate con la data di arrivo, partenza, agenzia che la gestisce e numero di passeggeri.

Una volta soddisfatto il primo requisito ci sono comprovati gli altri. La agenzia di trasporto di Barcellona, Transports Metropolitans de Barcelona (TMB), offre una API per interrogare il loro servizio e ottenere informazione della pianificazione e real-time della rete di trasporti [de Barcelona, 2015b]. Tramite la loro API si possono fare richieste di percorsi che utilizzano diversi mezzi, questo è un punto a favore perché permette delegare parte del problema a loro e centrare gli sforzi a fare che il prototipo abbia un buon funzionamento.

Un punto ad avere in mente è che la agenzia di trasporto TMB, nonostante essere privata con partecipazione del comune, è poco flessibile a modificare le frequenze delle linee. Invece le agenzie private che organizzano i tour turistici per la città sono completamente private e più flessibili a adattarsi alla demanda.

Esiste una agenzia di bus privati che offre il servizio di trasporto dal porto fino al centro, tratta conosciuta come *Moll Adossat - Mirador de Colom*. Adattano gli orari agli arrivi delle crociere al porto.

Al riguardo del traffico, non è stato possibile individuare nessuna API che garantisca accesso allo stato del traffico. L'unico software osservato in rete che aveva questa funzionalità è Google Maps che basandosi sulla localizzazione dei cellulari e la loro velocità calcola lo stato, pero l'interrogazione di questo servizio non è disponibile tramite la loro API.

Il comune di Barcellona, tramite il sito di OpenData [di Barcellona, 2015] offre diversi dati, tra di loro un file CSV in cui ci sono elencati tutti i punti turistici della città con le loro coordinate GPS e molta altra informazione. Inoltre, tramite pubblicazioni statistiche della entità che gestisce il turismo a Barcellona si possono consultare i dati statistici dei punti turistici più visitati.

Inoltre, le caratteristiche della città di Barcellona la rendono un ottimo scenario per fare delle prove. Da un lato è una città molto turistica con circa 10M di visitanti all'anno dei quali 2.5M

arrivano con crociera, dato che il porto è molto importante a livello Europeo.

L'agenzia di trasporto di Barcellona, che fornisce i dati del trasporto nel caso di studio, oltre al servizio SOAP offre i dati della rete nel formato GTFS, sviluppato da Google e Portland TriMet [Inc., 2015]. Questo formatto struttura l'informazione del trasporto in diversi file in modo che sia possibile usarsi in qualsiasi per rappresentare l'informazione di qualsiasi città indipendentemente dei mezzi che siano in disposizione. Dovuto alla struttura particolare dei dati GTFS si sono dovuti adattare i dati del trasporto alla struttura del database implementata nel progetto.

6.1 Esempi del funzionamento

Per elaborare i test si sono selezionate 3 aree del centro della città, due de le quali condividono una porzione del perimetro. A continuazione si mostrano le aree e le particolarità che sono state utilizzate nel prototipo per elaborare i dati e il modello 6.1 utilizzato per presentare l'informazione.

Nome:	Nome identificativo della zona.
Score:	Punteggio calcolato con il numero di posti d'interesse e una componente aleatoria.
Capacità massima:	Capacità massima di turisti nell'area, ottenuta in modo aleatorio con una soglia inferiore e una superiore arbitrarie [4000, 6000].
Numero di fermate:	Quantità di fermate del trasporto pubblico che si trovano nell'area.

Tabella 6.1: Modello per l'informazione dell'area.

Nome:	Zone A
Score:	97
Capacità massima:	4027
Numero di fermate:	27

Tabella 6.2: Proprietà dell'area *Zone A*.

Le aree *Zone B* e *Zone C*, come si può osservare nella Figura 6.1, condividono parte del perimetro. In questo modo gli eventi o problemi di congestione di una zona influirà sull'altra. Un altro fattore come nei diversi test è la percentuale di turisti che prenotano la gita in città, si

Nome:	Zone B
Score:	67
Capacità massima:	4498
Numero di fermate:	32

Tabella 6.3: Proprietà dell'area *Zone B*.

Nome:	Zone C
Score:	35
Capacità massima:	4916
Numero di fermate:	60

Tabella 6.4: Proprietà dell'area *Zone C*.

è assegnato il valore di 70% basandosi sulle informazioni ricavate durante la ricerca della prima fase del progetto.

Figura 6.1: Visione globale della città.

6.1.1 Gestione in caso di eventi

L'obiettivo di questo test era studiare i vantaggi che apporta il sistema nella gestione della città quando ci sono eventi e arrivano le crociere.

Per iniziare si è stata scelta una giornata sulla quale realizzare il test che rappresentasse una situazione abituale nella città. Il giorno scelto è stato il 10-08-2015 perché c'era previsto un arrivo di 12.770 croceristi, un numero ragionevole confrontando con il massimo annuale che è di 26.770 o il minimo di 580.

Di seguito, sono stati definiti due eventi nello stesso giorno, uno sciopero nella *Zone B* dalle 17:00 alle 21:00, con una affluenza di persone media, e un concerto nella *Zone C* dalle 18 fino le 2 della mattina del giorno successivo.

Poi, con questi dati si è realizzato il test per gestire la giornata con il proposito di analizzare ora per ora l'evoluzione dello stato della città, per poter raccomandare le zone da non visitare e le loro limitazioni riguardo al trasporto pubblico e traffico. Nella figura 6.2 si mostra l'evoluzione della situazione di congestione nelle zone sorvegliate.

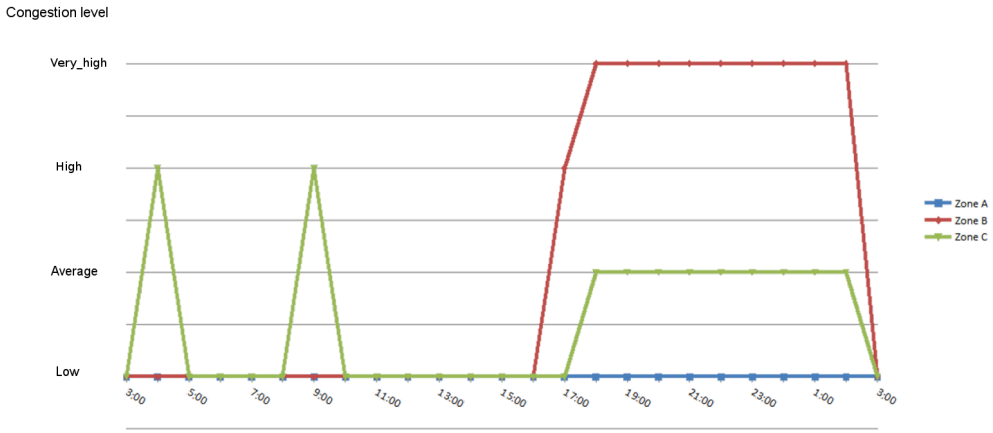


Figura 6.2: Evoluzione dei livelli di congestione nelle aree.

L'evoluzione dei livelli di congestione osservati nella Figura 6.2 si possono spiegare analizzando i diversi fattori che intervengono.

Per primo, alle 4:00 si produce l'arrivo della crociera *Disney Magic* con 5.420 passeggeri. Di seguito, il Port Agent decide di smistarli dal porto verso la *Zone C* con gli autobus, e aumenta

l'affluenza di persone in quell'area. A continuazione, si assume che i turisti si disperdano per la città e si riduce il livello di congestione, fino alle 9:00 quando arriva la crociera *Costa Diadema* con 4.980 persone. L'agente del porto propone di nuovo di spostare i croceristi verso la *Zone A* e aumenta leggermente in tutte le aree il numero di turisti, pero nella *Zone C* che ha una minore capacità ed è quella che riceve direttamente i croceristi è l'unica nella che si osserva un cambio di livello di congestione. La proposta del agente di spostare i croceristi verso la *Zone C* non è stata arbitraria, è stata scelta la zona meglio collegata con il trasporto pubblico.

Nelle ore successive la situazione si mantiene stabile fino le 17:00, momento in cui inizia uno sciopero nella *Zone B* e ne aumenta il livello di congestione. Inoltre, lo sciopero ha delle conseguenze sul traffico e trasporto che si analizzeranno più avanti.

Alle 18.00 si produce un cambio nella *Zone C* dovuto all'inizio di un concerto, che provoca un aumento della congestione nella *Zone B* dove continua lo sciopero. A questo punto, la situazione si mantiene fino che finisce il concerto alle 2:00 del giorno successivo.

Un ulteriore aspetto è la partenza durante la sera delle due crociere, che non ha influenzato la città perché il volume principale di persone era dovuto a eventi e non ai croceristi. Al largo della giornata sono arrivate altre 2 crociere pero con una quantità di passeggeri molto inferiore alle altre crociere.

L'utente può consultare per ogni zona le incidenze sulla mobilità nella zona, e quelle della *Zone B* sono ripportate nella Figura 6.3. Uno studio approfondito su questa zona è interessante dovuto alle particolarità dello sciopero. Ad esempio, viene raccomandato di non prendere il trasporto durante lo sciopero e inoltre si prevede che il traffico sia difficile fino a una ora dopo che abbia finito lo sciopero, dovuto al gran volume di persone che si troveranno nella zona.

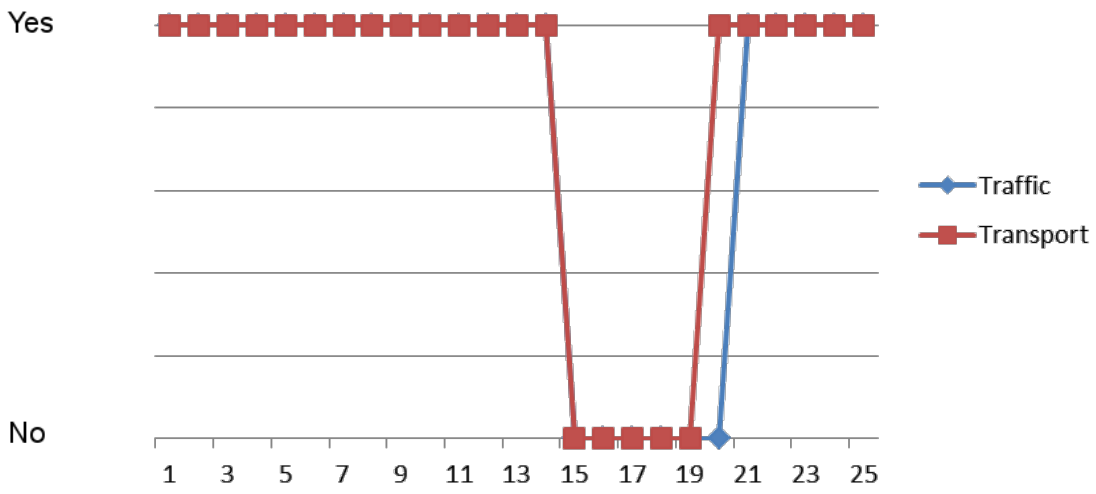


Figura 6.3: Raccomandazioni di mobilità per la *Zone B*.

Grazie a queste caratteristiche l'utente può prendere decisione riguardo le zone che sono o meno raccomandabili da visitare e gli orari per farlo. Inoltre, con questa informazione si possono prendere delle decisioni con le agenzie di trasporto per organizzare lo spostamento di turisti in un modo che non si trovino congestioni.

6.1.2 Gestione degli arrivi al porto

Uno dei principali obiettivi del progetto è spostare i croceristi dal porto a diversi punti della città mediante autobus. Per farlo, l'utente può decidere il numero di autobus che verranno usati e la loro capacità. Poi, l'utente ha la possibilità di decidere quale crociere disporranno di autobus senza superare il numero massimo d'essi, e in caso di non farsi una assegnazione manuale oppure farne una di parziale, cioè soltanto si assegnano alcuni autobus, il *Port Agent* fa una assegnazione automatica degli autobus restanti. Bisogna dire che la assegnazione si è decisa di far in modo che il numero di autobus assegnati a una crociera non sia inferiore al numero di croceristi.

Per analizzare come influisce il riparto dei croceristi fra le diverse zone si è fatto un test sul 7 di settembre di 2015. In questa giornata sono arrivate 8 crociere con un totale di 22.680 persone. Inoltre, gli arrivi si sono concentrati fra le 5:00 e le 10:00 della mattina.

Prima di tutto, il *Port Agent* fa una assegnazione degli autobus alle crociere per ogni ora, in questo caso era stata definita una limitazione di 60 autobus per il porto, con una capacità di 80 persone. Inoltre, dovuto all'alto numero di croceristi si è aumentata la soglia massima di turisti in zona al rango di [6000-7000]. Il valore è stato arbitrario basandosi sulla media di persone per zona che è vicina a 7.000. Nella Tabella 6.5 si mostrano gli arrivi programmati.

Nome crociera	**Ora arrivo**	**Num. croceristi**
CELEBRITY EQUINOX	5:00	6.300
ISLAND PRINCESS	6:00	3.940
EXPLORER OF THE SEAS	7:00	3.840
ZENITH	7:00	1.830
AZAMARA QUEST	7:00	750
COSTA DIADEMA	7:00	4.980
FUNCHAL	9:30	580
SEABOURN SOJOURN	10:00	460

Tabella 6.5: Arrivi di crociere al porto di Barcellona a 07/09/2015.

L'assegnazione di autobus si è realizzata per tutti gli arrivi tranne per la crociera *Celebrity Equinox* dato che non potevano entrare nel porto sufficienti autobus per trasportare tutti i passeggeri. Per difetto è il *Zone Coordination Agent* a decidere la destinazione degli autobus, affinché è l'unico che conosce gli stati di tutte le zone. L'azione abituale è la d'inviare gli autobus alla zona meglio collegata con il trasporto pubblico fino che questa raggiunge un livello di congestione alto ("High"). Per evitare questa situazione, nel test si è fatta una ripartizione manuale delle crociere come mostrato di seguito:

Nome crociera	**Destinazione degli autobus**	**Cambiamenti nella città**
CELEBRITY EQUINOX	-	-
ISLAND PRINCESS	Zone C	• Zone C: Aumenta il livello di congestione a High.
EXPLORER OF THE SEAS	Zone B	• Zone B: Aumenta il livello di congestione a High. • Zone C: Scende il livello di congestione a Low.
ZENITH	Zone C	• Zone C: Il livello aumenta a Average. • Zone B: Scende a Average. • Zone A: Aumenta a Average.
AZAMARA QUEST	Zone B	-
COSTA DIADEMA	Zone B	• Zone B: Aumenta il livello a Very High.
FUNCHAL	Zone A	• Zone A: Aumenta il livello a High.
SEABOURN SOJOURN	Zone C	• Zone B: Scende il livello a Low.

Tabella 6.6: Evoluzione dei livelli di congestioni nelle aree a 07/09/2015.

Dove lo stato finale in cui si trovano le aree è:

Zone A: High

Zone B: High

Zone C: Low

Gli incrementi prodotti nelle aree quando non ricevevano i croceristi direttamente sono dovuti a che il coordinatore di Zone ad ogni ora fa una distribuzione del numero totale di croceristi in

città fra le diverse aree, a seconda del suo punteggio o score. Questo si fa parallelamente alla assegnazione degli autobus dato che il *Port Agent* comunica per ogni ora il numero di croceristi che si trova in città.

Nel caso di farsi una assegnazione automatica si sarebbero collassate alcune aree mentre le altre rimanevano senza turisti. Le zone più attraenti per i turisti sono le *Zone A* e *Zone B* dovuto alla presenza di un numero più alto di punti d'interesse rispetto la *Zone C*. In questa ultima Figura 6.4 si può osservare i messaggi inviati dal *Port Agent* dopo di processare i dati degli arrivi tramite il modello.

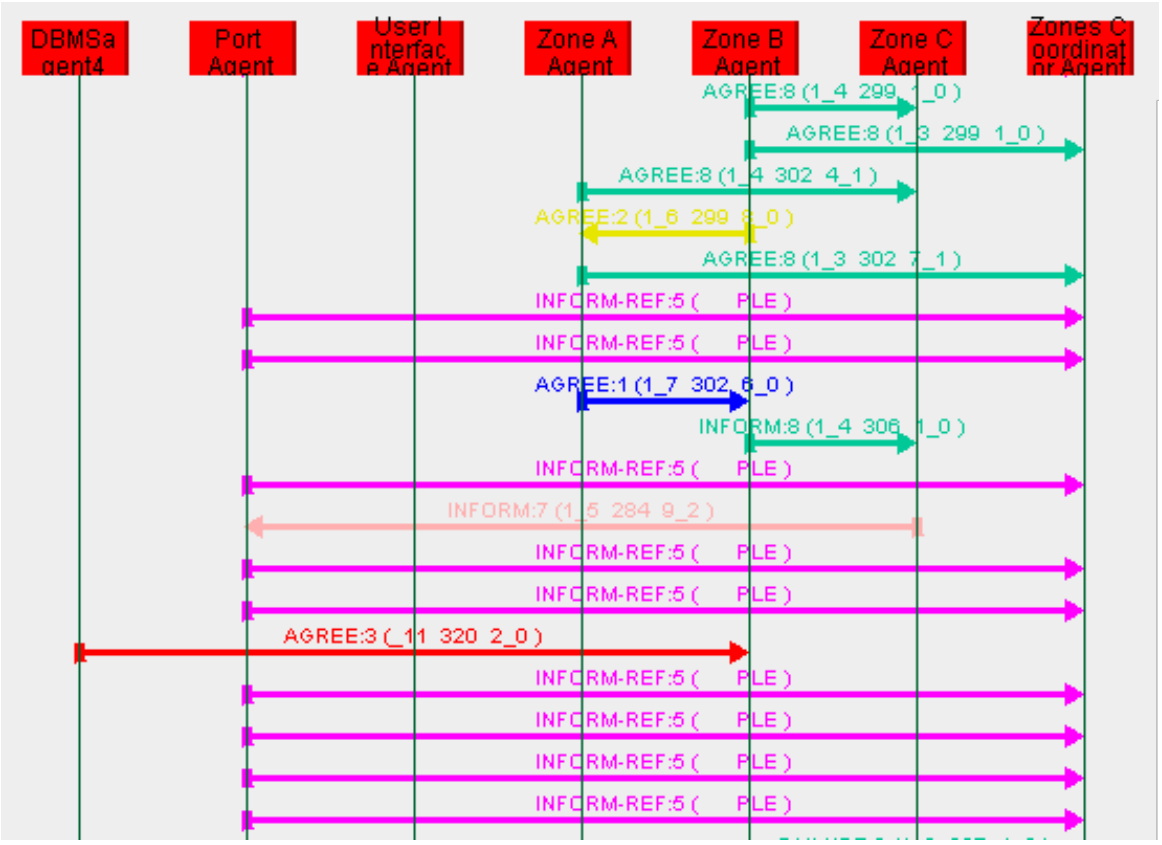


Figura 6.4: Il *Port Agent* informa degli arrivi di turisti.

Una volta il *Zones Coordination Agent* riceve i messaggi degli arrivi, gli processa grazie al modello e ai dati che possiede delle aree. per ottenere la distribuzione di croceristi fra le diverse zone. In seguito comunica ad ogni zona la quantità di turisti che arriveranno e anche a che ora. Cosi, le zone aggiornano lo stato in cui si trovano e se il modello di zona considera che si deve intraprendere qualche azione come comunicare con un'area vicina si procede.

Capitolo 7

Conclusioni e sviluppi futuri

In questo lavoro si è realizzato un primo approccio a un problema molto complesso, affrontando alcuni dei sotto-problemi che lo conformano. Durante l'approccio di alcuni aspetti si è rilevato che la complessità e il numero di fattori che entrano in gioco sono elevati, e che per elaborare una soluzione veramente efficace c'è bisogno di un lavoro costante con i diversi esperti del problema.

La predizione dell'impatto dell'arrivo dei croceristi nella città è risultato non essere difficile ma complesso, giacché intervengono molti fattori, dei cui soltanto uno esperto del campo può modellare accuratamente come nel caso del trasporto o dello spostamento delle persone.

Per questi motivi il lavoro svolto se è dimostrato utile per individuare quali sono le difficoltà, pero sopratutto per vedere che una organizzazione e previsione della gestione della città può riportare un impatto positivo nella misura che riduce i problemi di congestione e le conseguenze negative d'essi.

Nei test svolti si sono esplorate alcune delle possibilità che offre il sistema. Siccome la combinazione delle operazioni per modificare le proprietà degli autobus, il loro destino o la presenza di eventi permettono creare scenari molto diversi è difficile creare e studiare gli scenari senza un software specifico.

Per quanto il progetto svolto non è andato in profondità nei diversi sotto-problemi, già si sono individuati diversi aspetti che da risolversi significherebbero un gran passo verso la progettazione

di un sistema software avanzato.

Una delle prime sfide, approcciata recentemente [Hasan, 2010] [MATSim, 2012], è la integrazione della gestione del traffico e del trasporto come un unico sistema. La maggior parte degli studi teorici o lavori pratici affrontano entrambi problemi come due problemi diversi, quando in realtà di essersi una collaborazione tra di loro entrambi otterrebbero vantaggi. Purtroppo, il problema in generale è che la gestione del trasporto viene affidata a una organizzazione diversa della che gestisce il traffico senza che si sia un intendimento.

A misura che si progettava la soluzione Multi-Agente qua presentata, e si cercava di far in modo che il problema potesse essere gestito da diversi utenti allo stesso tempo, è risalito che una ulteriore ricerca sulla applicazione delle tecniche di decisione in gruppo per affrontare questo problema sarebbe di rilevanza. Questa metodologia si basa nel affrontare lo stesso problema da diversi utenti allo stesso tempo e di arrivare a un accordo sulla soluzione ad adottare.

Infine, nel caso di affrontare il problema con le risorse e supporto necessario per portare a termine il lavoro completo si potrebbe sviluppare un sistema molto efficiente economico e socialmente, che d'essere progettato adeguatamente sarebbe sfruttato da molte città.

Bibliography

[AIMA, 2009] AIMA (2009). Java implementation of algorithms from norvig and russell's artificial intelligence - a modern approach 3rd edition.

[Aimsun, 2015] Aimsun (2015). San diego integrated corridor management system.

[Almejalli et al., 2007] Almejalli, K., Dahal, K., and Hossain, M. (2007). *Intelligent Traffic Control Decision Support System*, volume 4448 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg.

[Alonso et al., 2008] Alonso, F., Fuertes, J., Martinez, L., and Soza, H. (2008). Measuring the social ability of software agents. In *Software Engineering Research, Management and Applications, 2008. SERA '08. Sixth International Conference on*, pages 3–10.

[Apache, 2015] Apache (2015). Apache tomcat.

[Atlantic Systems Guild Inc., 1999] Atlantic Systems Guild Inc., V. (1999). Volvere: Requirements resources.

[Balbo and Pinson, 2010] Balbo, F. and Pinson, S. (2010). Using intelligent agents for transportation regulation support system design. *Transportation Research Part C: Emerging Technologies*, 18(1):140 – 156. Information/Communication Technologies and Travel BehaviourAgents in Traffic and Transportation.

[Brahim Chaib-Draa and Millot, 1992] Brahim Chaib-Draa, Bernard Moulin, R. M. and Millot, P. (1992). Trends in distributed artificial intelligence. *Artificial Intelligence Review*, 6:35–66.

[CLIPS, 2015] CLIPS (2015). Clips: A tool for building expert systems.

[Dawes, 1988] Dawes, R. M. (1988). *Rational Choice in an Uncertain World*. Harcourt Brace Jovanovich College Publishers.

[de Barcelona, 2015a] de Barcelona, P. (2015a). Cruises consultation.

[de Barcelona, 2015b] de Barcelona, T. M. (2015b). Open data.

[Dennett, 1980] Dennett, D. C. (1980). Intentional systems. *The Journal of Philosophy*, 68(4):87–106.

[di Barcellona, 2015] di Barcellona, C. (2015). Open data bcn.

[DOT, 2015] DOT, U. (2015). Us department of transportation: Integrated corridor management.

[Druzdzel and Flynn, 1991] Druzdzel, M. J. and Flynn, R. R. (1991). Decision support systems.

[Finin et al., 1993] Finin, T., Weber, J., Wiederhold, G., Genesereth, M., Fritzson, R., Mckay, D., McGuire, J., Pelavin, R., Shapiro, S., and Beck, C. (1993). Draft specification of the kqml agent-communication language.

[FIPA, 2002a] FIPA (2002a). Fipa agent management specification, identifier fipa00023.

[FIPA, 2002b] FIPA (2002b). Fipa agent message transport service specification, identifier fipa00067.

[FIPA, 2002c] FIPA (2002c). Interaction protocol specifications.

[FIPA, 2005] FIPA (2005). Foundation for intelligent physical agents: History of fipa.

[Group, 2015] Group, T. P. G. D. (2015). Postgresql.

[Hasan, 2010] Hasan, M. K. (2010). A framework for intelligent decision support system for traffic congestion management system.

[Inc., 2015] Inc., G. (2015). General transit feed specification reference.

[Jordi Casas, 2014] Jordi Casas, Alex Torday, J. P. M. B. A. R. d. V. (2014). Decision support systems (dss) for traffic management assessment: Notes on current methodology and future requirements for the implementation of a dss.

[Konolige and Nilsson, 1980] Konolige, K. and Nilsson, N. J. (1980). Multiple-agent planning systems. 80.

[Makowski, 1991] Makowski, M. (1991). Selected issues of design and implementat ion of decision support systems.

[MATSim, 2012] MATSim (2012). Matsim: Singapore.

[Minch and Burns, 1983] Minch, R. P. and Burns, J. R. (1983). Conceptual design of decision support systems utilizing management science models. *Science, Mand and Cybernetics*, 13(4):549–557.

[Rao and P.Georgeff, 1995] Rao, A. S. and P.Georgeff, M. (1995). Bdi agents: From theory to practice.

[Russell and Norvig, 2003] Russell, S. J. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition.

[SANDAG,] SANDAG. Integrated corridor management: Analysis, modeling and simulation.

[Sascha Ossowski, 2005] Sascha Ossowski, Josefa Z. Hernández, M.-V. B. A. F. A. G.-S. J.-L. P.-d.-l.-C. J.-M. S. F. T. (2005). Decision support for traffic management based on organisational and communicative multiagent abstractions. *Transportation Research Part C: Emerging Technologies*, 13(4):272 – 298. Agents in Traffic and Transportation: Exploring Autonomy in Logistics, Management, Simulation, and Cooperative Driving.

[Sascha Ossowski, 2004] Sascha Ossowski, Josefa Z. Hernández, M.-V. B. A. F. A. G.-S. J. M. (2004). Multi-agent systems for decision support: A case study in the transportation management domain. *Applied Artificial Intelligence*, 18:779 – 795.

[Turban, 1995] Turban, E. (1995). *Decision Support and Expert Systems: Management Support Systems*. Prentice-Hall International, Reading, Massachusetts".

[TutorialsPoint, 2015] TutorialsPoint (2015). Tutorials point: Database management system tutorial.

[Vidal, 2009] Vidal, J. M. (2009). Fipa performatives.

[Wooldridge and Jennings, 1994] Wooldridge, M. and Jennings, N. R. (1994). Intelligent agents: Theory and practice. *Knowledge Engineering Review*.