Servei de transport de paquets amb vehicles intel·ligents, optimitzat mitjançant un sistema multiagent

Pedro Gonzalvez Martínez
Grau en Enginyeria Informàtica
Intel·ligència Artificial

Consultor/a : David Isern Alarcón
Professor/a responsable de l'assignatura : Carles Ventura Royo

Data Lliurament : 2/1/2018

FITXA DEL TREBALL FINAL

Títol del treball:	*Servei de transport de paquets amb vehicles intel·ligents, optimitzat mitjançant un sistema multiagent.*
Nom de l'autor:	*Pedro Gonzalvez Martínez*
Nom del consultor/a:	*David Isern Alarcón*
Nom del PRA:	*Carles Ventura Royo*
Data de lliurament (mm/aaaa):	*2/1/2018*
Titulació o programa:	*Grau en Enginyeria Informàtica*
Àrea del Treball Final:	*Intel·ligència Artificial*
Idioma del treball:	*Català*
Paraules clau	*Sistema multiagent, Transport, Agents.*

Resum del Treball (màxim 250 paraules): *Amb la finalitat, context d'aplicació, metodologia, resultats i conclusions del treball*

L'actual sistema de transport de paquets, és un sistema que depèn, en gran mesura, dels humans. Des de l'atenció de clients, fins la recollida i lliurament de paquets, el factor humà és vital, tant en la gestió dels recursos, com per la ma d'obra necessària. Aspectes com els accidents laborals, una mediocre gestió dels operaris i la picaresca, incideixen negativament en l'eficiència d'un servei d'aquestes característiques.

En aquest treball es proposa delegar, la majoria de tasques d'aquest servei a un sistema informàtic, basat en agents intel·ligents. La base de disseny és un SMA que incorpora tots els elements necessaris, per a dur a terme la implementació i estudi de l'escenari proposat (un model de ciutat, vehicles autònoms i agents de control). Per a coordinar-los, s'han utilitzat les característiques que ens proporcionen els propis agents, tècniques de IA (com els algoritmes de distància mínima, Dijkstra) i models de predicció amb arbres de decisió, pels càlculs de prioritats.

Amb l'objectiu de simular escenaris reals, s'ha implementat un seguit de test de proves, que s'asseguraren el bon funcionament del sistema (transport de paquets d'una posició a un altra, atenció de més d'un client, assignació del vehicle adequat, incidents, cua d'espera, entre d'altres). També s'ha instaurat un sistema de selecció predictiu de clients, per tal de realitzar l'estudi d'un possible augment global de l'eficiència.

Finalment, s'ha arribat a la conclusió, que és possible un nivell d'autonomia elevat, que permet instaurar un servei de transport de paquets, sense dependència humana.

Abstract (in English, 250 words or less):

The current system of transport of packages, is a system that depends, to a large extent, on the humans. From the customer service, to the collection and delivery of packages, the human factor is vital, both in the management of resources, as well as the necessary work force. Aspects such as accidents at work, mediocre management of operators and the picaresque, negatively affect the efficiency of a service of these characteristics.

In this work, we propose to delegate, most of the tasks of this service to a computer system, based on intelligent agents. The design base is an SMA that incorporates all the necessary elements, to carry out the implementation and study of the proposed scenario (a city model, autonomous vehicles and control agents). To coordinate them, the characteristics provided by the agents themselves, IA techniques (such as the minimal distance algorithms, Dijkstra) and prediction models with decision trees, have been used for the calculation of priorities.

With the aim of simulating real scenarios, a series of test tests have been implemented, which ensure the proper functioning of the system (transport of packages from one position to another, attention of more than one customer, allocation of the appropriate vehicle, incidents, waiting tail, among others). A system of predictive selection of clients has also been established, in order to carry out the study of a possible overall increase in efficiency.

Finally, it has come to the conclusion that a high level of autonomy is possible, which allows the establishment of a package transport service, without human dependence.

Índex

Llista de figures

Llista de Taules

1. Introducció

1.1 Context i justificació del Treball

El món demana contínuament, que les tasques del dia a dia, responguin a tres principis bàsics, que siguin senzilles, ràpides i eficients. Les noves tecnologies intenten cercar solucions, que incorporin aquest tres principis bàsics. Un exemple pràctic d'aquest fet, és la implementació de sistemes intel·ligents que optimitzen tasques.

Una tasca molt habitual és l'enviament i recepció de paquets entre punts geogràfics. És un servei on intervenen molts factors, tals com: vehicles, comunicació, tracte humà i tecnologia.

Coordinar tots aquest requisits, per tal de donar una bona experiència d'usuari, no és una tasca senzilla. Existeixen factors que intervenen directament, com per exemple, el temps de recollida i enviament d'un paquet. El preu del servei és un altre factor, que ve determinat en moltes vegades, pels costos afegits d'algunes ineficiències del sistema.

Actualment, les empreses de transport tenen al seu abast, diferents tecnologies que ajuden a reduir ineficiències, però la majoria de les vegades, és necessària la supervisió del seu funcionament mitjançant persones.
En aquest treball final de grau es vol optimitzar aquest servei, creant un escenari autònom que no necessiti aquesta dependència humana. Un sistema capaç de reconèixer les necessitats i actuar en conseqüència, davant cada situació. Així, ens centrarem en l'anàlisi, disseny i implementació d'un sistema multiagent, que tingui com a tasca l'optimització del transport de paquets, mitjançant vehicles intel·ligents dintre d'una determinada ciutat.

Per tal de dur a terme els nostres objectius, plantejarem un petit negoci de transport de paquets per a una determinada ciutat. El negoci estarà constituït per un agent central, alguns vehicles intel·ligents, un agent de gestió geogràfica, un agent de control de la informació de tot el sistema i agents clients. L'agent central recollirà les sol·licituds dels clients, negociarà amb els vehicles les millors opcions de transport i informarà al client de la seva decisió. Els vehicles que estiguin disponibles, rebran la informació del servei que han de realitzar, demanaran al sistema de gestió geogràfica un plànol general i mitjançant aquesta informació, calcularan el cost del servei. Una vegada l'agent central obtingui la informació dels costos de cada vehicle, escollirà el cost més assequible pel client i li farà arribar la informació final del servei.
Partirem de la base que s'utilitzen vehicles autònoms per transportar els paquets. Aquests vehicles no necessiten conductors humans per circular i funcionen mitjançant l'energia elèctrica.

1.2 Objectius del Treball

L'objectiu principal d'aquest treball final de grau és l'estudi i implementació d'un Sistema multiagent (SMA[1]) que optimitzi el servei de transport de paquets mitjançant vehicles intel·ligents, en una determinada ciutat. Ens centrarem en el transport de mercaderies d'una posició a una altra, com pot ser la distribució sota comanda de paquets.

Mitjançant aquest treball és vol aprofundir en dos aspectes fonamentals:

- Estudi dels agents i els sistemes multiagents per a resoldre problemes complexos.
- Dissenyar i avaluar una aplicació pràctica en un domini concret, en aquest cas el transport de mercaderies sense actuació humana.

Per realitzar aquest treball, primer de tot s'haurà de repassar i documentar bona part dels coneixements previs sobre intel·ligència artificial, aprenentatge computacional i estadística. També ens haurem de situar en el món tecnològic actual i la possible tendència de futur, on els vehicles intel·ligents tenen un factor determinant en aquest àmbit.
La comunicació entre tecnologies també serà un punt a tindre en compte, ja que el nostre disseny ha de comunicar-se entre ell, per tal d'oferir i recavar informació.

Bona part del nostre treball es centrarà en treure conclusions, això ens obliga a configurar un ambient de proves simulades, que ens facilitarà l'extracció de dades rellevants. En el nostre cas, serà la plataforma JADE la que ens aportarà l'ajuda necessària en aquest sentit.
No hi ha experiència prèvia amb aquesta plataforma, per tant s'haurà d'invertir cert temps a entendre el seu funcionament.

1.3 Enfocament i mètode seguit

Ens centrarem en el disseny i implementació d'un producte nou, que vol optimitzar un producte ja existent. Per aconseguir-ho, utilitzarem la potència que ens ofereixen els agents[2] intel·ligents.

Arribats aquest punt, és essencial justificar el fet d'utilitzar agents i no un sistema habitual de programació amb objectes (OOP[3]). Mes endavant, profunditzarem en les característiques pròpies dels agents, però ara donarem unes petites pinzellades, per tal de justificar la seva implicació en aquest projecte.

Quan parlem de similituds entre Objectes i agents, es pot dir que tots dos:

---

[1] Un sistema multiagent (SMA) és un sistema composat per varis agents intel·ligents que interactuen entre ells. Poden utilitzar-se per resoldre problemes complexes per un sol agent o sistema monolític.

[2] "Los objetos lo hacen gratis, los agentes porque quieren y por dinero" (M.Woolridge, Intro. To Multiagent System).

[3] OOP (Programació orientada a objectes). És un sistema de programació on els objectes manipulen les dades d'entrada per obtenir les dades de sortida específica, on cada objecte proporciona una funcionalitat especial. https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos . [22/09/2017]

- Realitzen un encapsulament del seu estat.
- Es comuniquen mitjançant pas de missatges.
- Tenen mètodes que es corresponen amb les accions, que es poden realitzar segons el seu estat.

Però, a diferència dels objectes, els agents tenen certes característiques que els fan interessants, per aconseguir l'objectiu d'aquest treball :

- Són autònoms, es a dir, tenen la capacitat de decidir per si mateixos si han d'actuar o no, quan reben peticions d'altres agents.
- Són intel·ligents, es adir, capaços de realitzar comportaments flexibles. Aquesta característica els fa reactius, proactius i socials.
- Són actius.

En el nostre cas, on volem modelar una petita empresa de transport d'objectes dintre d'una ciutat, sense dependència humana, es fa imprescindible un sistema reactiu, es a dir, que s'adapti contínuament a l'entorn i que respongui als canvis que es produeixin. Evidentment, aquestes respostes s'han de realitzar a temps, per a que tinguin una utilitat real.

Un altre factor important en el nostre projecte, és que necessitem que es realitzin tasques sense la nostra presència, es a dir el sistema ha de ser capaç de realitzar accions de forma autònoma, per arribar a un objectiu comú. Mitjançant l'habilitat social dels agents (capacitat d'interactuar amb altres agents) es pretén aconseguir una coordinació conjunta.

Aquesta coordinació només és possible, en un escenari on interactuïn diferents agents i que conjuntament, resolguin un problema en concret. Aquest fet, ens obliga a definir una arquitectura, on dintre d'un mateix escenari, puguin conviure diferents agents.

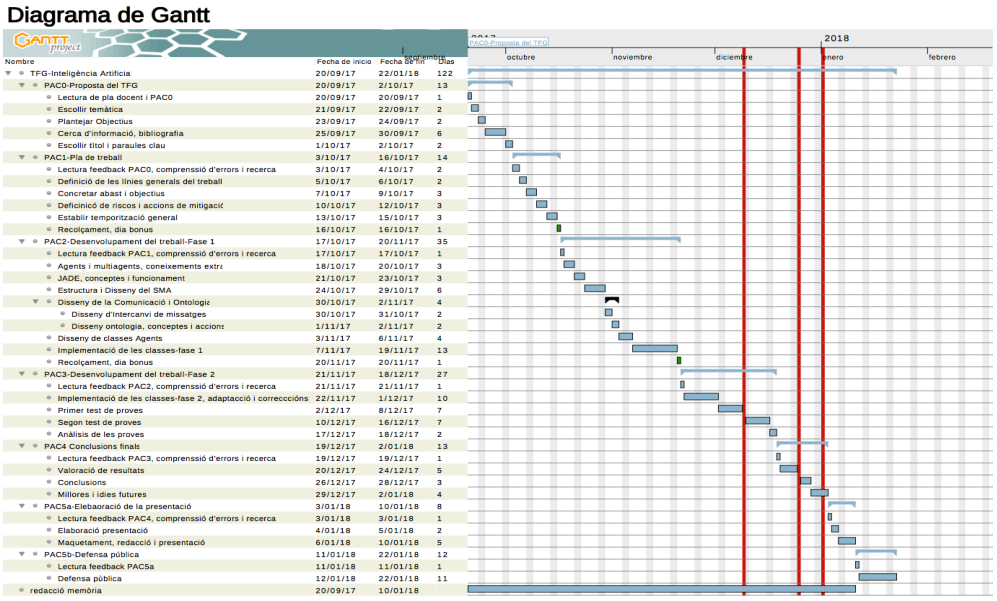
Un SMA, permet que els agents estiguin connectats, organitzats i pugin comunicar-se, per tal d'aconseguir un objectiu concret.

1.4 Planificació del Treball

Per tal de plantejar la planificació del treball, es tindrà en compte els principis de gestió de projectes que ens ofereix el PMBOK.

Il·lustració 1. Cicle de vida d'un projecte segons el PMBOK.

Un cop presentada la proposta (que seria la fase de *Initiating*[4]) es procedeix a l'anàlisi del temps de treball (fase de *Planning*[5]). El projecte s'ha dividit principalment en la temporalitat que marquen les diferents activitats de seguiment, proposades per la direcció de projecte. Dintre de cada activitat s'ha distribuït el treball segons les tasques assenyalades en cada fase. També s'han marcat tots els dies de la setmana com hàbils, excepte el dia 9 i 25 de Desembre i el dia 1 de Gener. S'ha considerat un mínim de 2.5 hores de treball al dia, per tal s'assolí l'objectiu de 300 hores lectives. El dia 16 d'Octubre i el 20 de Novembre s'han considerat dies en format de "Bonus" (si existeix retard es dedicarà per reforçar el treball o dia de descans si existeix una cronologia prou avançada).



Il·lustració 2. Diagrama de Gantt, Pla de treball

A continuació s'han considerat els riscos segons el nivell de gravetat i es donen possibles accions de mitigació. Nivell 1: Alt, Nivell 2: Mig, Nivell 3: Baix

Nivell	Risc	Temps/dies	Acció
1	Malaltia greu	Mes de 15	Retard considerable, reducció de les funcionalitats del projecte.
1	Malaltia Lleu	Entre 3 - 5	Retard considerable, reducció, d'alguna funcionalitat. Descartar la mes complexa.
1	Averia d'equip	Mes de 3	Retard considerable, Cercar altre equip (actualment només 1 disponible)
2	Retard en les dates de lliurament	Mes de 3	Augment de les hores de treball diàries, passar a 3.5h
2	Pèrdua de dades	-	Backup diari en PEN USB (memòria i codi)
3	Dubtes implementació	-	API de JAVA API de JADE

Taula 1:Riscos i mitigacions

[4] Començament del projecte i definició global. Es defineix el domini, l'abast, els recursos i el compromís que es vol assolir.
[5] Planificació detallada de les tasques que intervenen en tota la vida del projecte.

1.5 Breu sumari de productes obtinguts

Una vegada finalitzat el projecte, s'obtindrà un paquet JAVA on s'implementarà un SMA que simularà una empresa de transport, amb vehicles intel·ligents, control i clients que interactuen entre ells mitjançant agents.

L'execució d'aquest paquet ens ha de permetre veure el bon funcionament del model i extraure conclusions. Es a dir, que es compleixi la finalitat del servei (que un client pugui demanar el servei de transport i que aquest es dugui a terme amb la major eficiència possible) i extraure dades rellevants per valorar el producte.

1.6 Breu descripció dels altres capítols de la memòria

En principi i amb el condicionant que pot canviar, l'estructura de capítols i sense entrar en molts detalls, pot ser aquesta:

- Món digital : En aquest capítol abordem la situació actual i els canvis tecnològics que tenim pressents i que tenen cert impacte en el nostre estudi. Plantejarem la necessitat de la IA com a model de futur i donarem una pinzellada a les ciutats i vehicles intel·ligents.

- Agents : En aquest capítol, parlarem dels agents i els SMA (sistema multiagent). És essencial entendre què són, com funcionen i les seves característiques.

- Jade : En aquest capítol ens submergim en el software JADE. Estudiarem els seus components principals, com interactua amb els agents i el seu llenguatge de programació (Java en el nostre cas).

- Descripció del projecte : En aquest capítol es detalla de forma profunda tot el projecte. Ens centrarem en l'estructura general i específica dels agents i del problema que es vol resoldre. Resoldrem la part comunicativa entre agents i justificarem la seva aplicació. Detallarem casos d'ús i el funcionament general, tant dels agents com la lògica d'interconnexió.

- Implementació : En aquest capítol es detallen les classes .java que s'hagin desenvolupat en tot el projecte. S'explica el seu funcionament i el perquè de la seva estructura. Per exemple, explicarem els mètodes i les seves tasques. També es presentaran els test de proves i el seu funcionament.

- Simulació : En aquest capítol ens centrem en l'execució dels test de proves. L'objectiu principal es comprovar el bon funcionament de la implementació i emmagatzemar informació rellevant.

- Valoracions i conclusions : En aquest capítol analitzem els resultats generals. Haurem de ser molt crítics i extraure conclusions amb una base sòlida. Per exemple, les dades obtingudes en la fase anterior, ens han de permetre comparar el nostre sistema amb un sistema actual de transport de paquets.

- Propostes de futur : En aquest capítol ens plantegem les millores que poden sorgir després de l'estudi.

- Bibliografia : En aquest capítol trobem les fonts d'informació consultades.

- Annexos : Parts del document que siguin molt extenses i que farem referència des de el treball.

2. Món digital

Vivim en un món, que avança tecnològicament a una velocitat vertiginosa. Succeeix en gairebé tots els recons del planeta i afecta a molts àmbits, com per exemple: els aparells quotidians, la medicina, l'economia, les comunicacions, els vehicles, les ciutats i molts més.

Tasques que fa un temps enrere, consumien molt de temps i recursos en la seva execució, actualment es minimitzen gràcies a les noves tecnologies. Només cal fer una ullada als últims avenços en informàtica, per donar un petit exemple.

I és que, gràcies a la informàtica, tasques habituals del dia a dia, s'han tornat més còmodes per l'usuari. Per exemple, el fet d'enviar un paquet, d'un lloc geogràfic a un altre, ja no és una tasca gaire complicada. Normalment, l'usuari es posa en contacte amb una empresa de transport i li fa arribar les dades necessàries. Seguidament, l'empresa de transport, mitjançant una logística pròpia, realitza el servei i obté un benefici econòmic per aquest fet.
Però, és capaç la tecnologia actual de polir aquest servei ?. Es pot interconnectar sistemes tecnològics, que permetin realitzar un servei de transport d'una forma més eficient ?. Anem a respondre en aquest treball, a algunes d'aquestes qüestions.

2.1 Tendència actual en el àmbit del transport

Per tal d'avançar en el nostre estudi, es fa necessari entendre com funciona el sistema actual de transport de paquets i quins són els seus problemes més generals. També ens interessa extraure, d'aquest estudi, possibles necessitats de futur.

Actualment, el transport de paquets funciona mitjançant vehicles conduits per humans. Aquest fet, ens condiciona a complir certa normativa de salut laboral, per tal d'assegurar que el conductor estigui en òptimes condicions de treball (temps de descans). També ens condiciona, els canvis agressius de demanda (pics de treball, a l'alta o a la baixa) on es fa necessari un augment o reducció del nostre personal.

Un conductor humà, pot ser susceptible a la picaresca (conduir per rutes més llargues, aturar-se en diferents llocs sense necessitat, mentir en la seva activitat diària, entre d'altres). Trobaríem la mateixa situació en un treballador d'oficina, que tingués una actitud passiva, planificant i organitzant el treball diari.

Encara que sempre s'ha de pensar, que una persona efectuarà el seu treball amb professionalitat, no sempre és així i com a conseqüència l'empresa redueix la seva eficiència.

I si tinguéssim la possibilitat de delegar la gran part d'aquestes tasques a sistemes tecnològics, que raonin en situacions reals ?. Seria possible gestionar un servei de transport de paquets, sense intervenció humana de forma directe ?. Es a dir, és possible construir un sistema capaç d'adaptar-se a fluctuacions i que reaccionin davant de múltiples situacions ?.

2.2 La IA com a model de futur

Si volem que els nostres sistemes s'adaptin i reaccionin al seu entorn, es fa necessari implementar una certa intel·ligència. És en aquest punt, on entra en joc la intel·ligència artificial (IA).[6] No és el nostre objectiu entrar en detalls profunds en el concepte de IA, però si donarem una idea molt general.

La IA es pot definir, com la forma de simular al cervell humà, però atenció, només la part lògica i no física o emotiva d'aquest. Per dur a terme aquesta simulació, es fa necessari un sistema tecnològic amb una memòria que duri en el temps, tingui certs objectius i un sistema d'assignació de prioritats i presa de decisions. Això és molt interesant, perquè un sistema intel·ligent amb les característiques anteriors i sense elements emocionals, permet que no s'oblidi del seu objectiu i a més, que es consolidi com un sistema capaç d'executar comportaments complexes.
Cada vegada més, per tal de realitzar les tasques diàries, els dispositius adopten certa IA en el seu interior. Això, allibera a l'usuari de treballs habituals i li permet utilitzar el temps o recursos extra, per altres tasques.

Si traslladem aquest concepte al món del transport de paquets, ens obliga a qüestionar-nos si és possible incloure IA en el nostre servei i per tant, extraure un benefici, ja sigui de caràcter eficient o econòmic.

Però, com fer que el servei sigui intel·ligent ?. És necessari un únic punt de gestió intel·ligent o per contra, és millor que cada part actuï de forma intel·ligent ?.

2.3 Ciutats i vehicles intel·ligents

Si es pretén optimitzar el servei de transport de paquets, de forma que tingui autonomia pròpia, és evident que fer-ho des de un lloc o dispositiu, no és el més adequat. Analitzant amb cura el problema, es pot observar que tenim un abast massa gran, ja que necessitem un gestor de clients, una estructura central que coordini les peticions, un servei de vehicles que siguin capaços d'avaluar diferents escenaris, un punt d'accés al servei geogràfic i un control general.

De fet, el propi escenari ja és gran, perquè estem parlant, com a mínim d' una ciutat. Certament, ens podem aprofitar de les ciutats intel·ligents per realitzar el nostre estudi, ja que són llocs geogràfics que implementen una certa IA. Estem

[6] (IA) També anomenada intel·ligència computacional, que en poques paraules és la intel·ligència exhibida per màquines. https://es.wikipedia.org/wiki/Inteligencia_artificial

parlant de ciutats que permeten interactuar amb dispositius, com per exemple vehicles.

De fet, el nostre estudi es basa en vehicles intel·ligents sense conductor, que circularan per la ciutat. La interacció vehicle-ciutat és important, ja que permetrà al vehicle comunicar-se amb el sistema central, estigui on estigui geogràficament. En el nostre cas, considerarem que estem en una ciutat on simplement tenim al nostre abast, punts d'accés de comunicació.

Per altre banda, pressuposem que utilitzarem vehicles que tenen una IA consolidada (actualment ja existeixen vehicles que són capaços de viatjar d'un lloc a un altre, sense conductor) i que hauran de prendre decisions. (nosaltres ens limitarem a dissenyar algunes de les decisions que haurà de prendre, en cas de que hagi de realitzar un servei, com per exemple el càlcul de recorregut).
Però, com implementar un sistema d'aquest tipus i extraure conclusions?.

3. Agents

En la part introductòria d'aquest document (punt 1.3), es va parlar de la necessitat d'utilitzar agents en aquest treball, però no es va aprofundir gaire en el tema. Ara és el moment de fer-ho, perquè d'aquesta forma, el lector entendrà la necessitat dels agents en aquest projecte i l'obrirà el camí als coneixements necessaris, per entendre la metodologia de disseny que s'ha utilitzat en els propers capítols.

3.1 Què és un agent

Un agent és un procés de computació, capaç de realitzar tasques de forma autònoma i que te la capacitat de comunicar-se amb altres agents, per resoldre problemes mes complexes. Normalment, per dur a terme la resolució d'un problema complexa, l'agent utilitza coordinació, cooperació i negociació amb els altres agents.

En el nostre projecte, aquesta característica és vital, ja que necessitarem de la col·laboració entre agents, per tal de dur a terme correctament el servei que ens proposem.

3.2 Propietats dels agent

Són moltes, però descriurem algunes de les mes importants i que són importants en el nostre projecte :

- Autonomia: Tenen control sobre les seves accions i el seu estat intern. A més poden operar sense intervenció humana.
- Reactivitat: Percepció del seu entorn i reacció en conseqüència en temps adequat.
- Intel·ligència: Utilitzen la IA per resoldre problemes.
- Proactius: En certs moments tenen que demostrar que són capaços de prendre la iniciativa.

- Cooperatius: Mitjançant algun llenguatge de comunicació, són capaços de comunicar-se amb altres agents i interactuar.
- Aprenentatge: Milloren el seu comportament amb el temps.
- Veracitat: Mai comunicarà informació falsa de forma premeditada.

3.3 Arquitectura dels agents

Una vegada enteses les propietats dels agents, ens interessa conèixer les diferents formes que es poden adoptar, a l'hora d'implementar un agent. Es a dir, les estructures bàsiques on es poden trobar agents i que són, majoritàriament 3:

Deliberatives: Els agents raonen segons el model BDI (Beliefs Desires and Intentions[7]) , donat una representació del món real.
Reactives: Els agents reaccionen a estímuls simples i combinant-se amb altres agents reactius. Això forma un comportament intel·ligent.
Híbrides: Són estructures on hi ha una combinació d'agents deliberatius i reactius.

En el nostre cas, utilitzarem una arquitectura deliberativa, ja que representarem un escenari del món real i cada agent tindrà un o més objectius concrets a resoldre. Per conseqüència, cada objectiu serà abordat per unes determinades accions.

3.4 tipus d'agents

Existeixen diferents tipus d'agents: d'informació (gestionen i manipulen dades), d'interfície (els agents col·laboren amb l'usuari per resoldre a un problema), de col·laboració (agents que es comuniquen i col·laboren amb altres agents, per resoldre un problema en comú), mòbils (agents que es poden moure per la xarxa per recollir dades o interactuar amb altres dispositius), reactius (agents simples que reaccionen a estímuls), entre d'altres.
Però nosaltres ens centrarem en els agents del tipus de col·laboració, ja que ens interessen les característiques de comunicació i cooperació per resoldre un problema comú.
Per altra banda, queda evident per la naturalesa del nostre treball, que amb un únic agent, no es resol el problema proposat. Llavors, quina solució ens ofereixen els agents, que permeti crear un escenari on es puguin comunicar i col·laborar entre ells ?.

3.5 Sistemes multiagents (SMA)

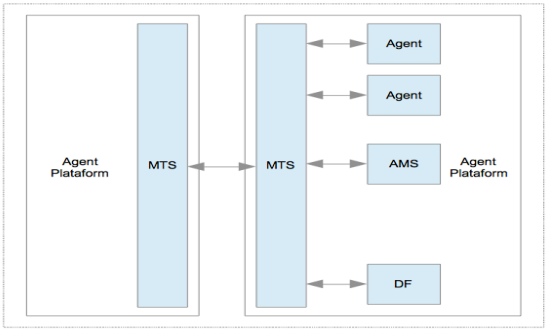
Un SMA[8] és un escenari on múltiples agents interactuen entre ells. Aquest sistema pot ser utilitzat per resoldre problemes complicats o molt difícils de resoldre, per una agent individualment.

[7] BDI, si es vol aprofundir en aquesta arquitectura d'implementació d'agents es pot consultar el següent enllaç: https://en.wikipedia.org/wiki/Belief–desire–intention_software_model
[8] SMA (sistema multiagent), si es vol mes informació es pot consultar en el següent enllaç: https://es.wikipedia.org/wiki/Sistema_multiagente

3.5.1 Avantatges dels SMA

Utilitzar un SMA en el nostre projecte, ens proporciona unes característiques, que s'adapten perfectament al nostre projecte.
Flexibilitat: En un SMA es poden afegir i esborrar agents de forma dinàmica.
Eficiència: Un SMA permet paral·lelisme entre tasques, ja que cada agent, de forma distribuïda, assoleix la resolució d'una petita part del problema general.
Modularitat: Permet una programació més estructurada.
Fiabilitat: Tolerància a fallades, si falla un agent, no vol dir que la resta falli també.
Una vegada vistes les característiques, anem a donar una ullada ràpida al funcionament general del SMA.

3.5.2 Esquemàtic d'un SMA



Il·lustració 3: Esquemàtic d'un SMA

Com es pot veure en l'esquemàtic anterior, existeix un sistema de transport comú anomenat *Message Transport System* (MTS), que gestiona la comunicació interna entre agents i entre plataformes. Existeixen dos agents més, que són indispensables pel bon funcionament del sistema. L'agent que gestiona els serveis que ofereixen els diferents agents, una mena de servei de pàgines grogues anomenat *Directory Facilitator* (DF) i l'agent que gestiona l'accés i us de la plataforma, anomenat *Agent Management System* (AMS).
En l'esquemàtic es pot veure l'intercanvi de missatges entre agents, però de quina forma s'entenen ?.

3.5.3 Comunicació entre agents en un SMA

Els agents es poden comunicar mitjançant un determinat llenguatge comú. Parlem del ACL[9] (Agent Comunication Language), on la forma i utilització del mateix, ve determinada pel protocol FIPA[10].

De forma resumida, un missatge ACL està constituït per uns determinats camps, anomenats "*slots*". Veurem més endavant amb més detall els "slots" utilitzats en el nostre projecte i la seva funció, però anem a resumir-los d'una forma molt breu.

[9] ACL (Agent Communication Language) , si es vol una ampliació sobre com funciona aquest llenguatge, es pot consultar el següent ennlaç: https://en.wikipedia.org/wiki/Agent_Communications_Language
[10] FIPA, mes informació sobre aquest protocol en el següent enllaç: http://www.fipa.org/repository/ips.php3

Definició	*Slot*
Tipus de missatge	*Performative*
Participants de la comunicació	*Sender, Receiver, Reply-to*
Contingut	*Content*
Descripció del contingut	*Ontology, Encoding, Language*
Control de convers	*Protocol, Reply-with, Conversation-id, In-reply-to, Reply-by*

Taula 2:Camps missatge ACL

Seguidament, es resumeixen els *Slots* i els possibles arguments que poden adquirir. També ho farem sobre taules, per tal de resumir breument els conceptes que es veuran amb més detall, més endavant.

Tipus de missatge	Objectiu
agree	Acceptació d'una acció
cancel	Cancel·lació d'una acció
Accept-proposal	S'accepta una proposta que s'havia rebut amb anterioritat
confirm	L'emissor confirma al receptor que una proposta és certa
disconfirm	L'emissor confirma al receptor que una proposta és falsa
cfp	Es proposa una acció
failure	El receptor informa que una proposta a fallat
inform	L'emissor informa al receptor que s'ha realitzat una proposta
propose	Es demana la realització d'una acció
Not-understood	L'emissor informa al receptor que no ha entès la seva petició
Query-if	Es demana a un altre agent si una acció és certa o no
Refuse	Es desestima una acció
Request	L'emissor demana al receptor que realitzi una acció
Subscribe	L'emissor vol ser informat quan succeeixi una determinada acció

Taula 3: Arguments dels Slots

En aquesta taula es representen els identificadors dels agents.

Participants de la comunicació	Objectiu
receiver	Agent o agents que reben el missatge
sender	Agent que envia el missatge
reply-to	Agent que rebrà el següent missatge de la conversa

Taula 4: Identificadors

Seguidament, els possibles continguts.

Contingut	Objectiu
FIPA-CCL (Constant Choice Language)	Semàntica que permet especificar, amb restriccions, predicats.
FIPA-SL (Semantic Language)	És el llenguatge més utilitzat i permet formar expressions lògiques, intercanviar coneixement i accions.
FIPA-KIF (Knowledge Interchange Format)	En aquest cas els objectes són expressats com a termes i les proposicions com a sentències.
FIPA-RDF (Resource Description Framework)	Ofereix l'opció d'expressar objectes i accions entre els agents.

Taula 5: Continguts dels identificadors

A continuació, els descriptors que permeten que un agent identifiqui el format del missatge.

Descripció del contingut	Objectiu
Language	És el llenguatge que s'utilitza per escriure el contingut del missatge
Enconding	Tipus de codificació
Ontology	Estructura que dona sentit al contingut del missatge

Taula 6: Descriptors

Finalment, identificadors que permeten a un agent extraure els tipus de conversa.

Control de conversa	Objectiu
Protocol	Declaració del protocol utilitzat
Reply-with	Identificador d'un missatge
Reply-by	Data i hora de caducitat d'un missatge
In-reply-to	Resposta a una acció passada
Conversation-id	Identificador de conversa

Taula 7: Identificadors de Tipus de conversa

Cal especificar, que en taula anterior, el control de conversa "*protocol"* permet utilitzar uns passos ja definits, per tal de realitzar una conversa entre agents. Anem a resumir-los breument:

Protocol	Objectiu
FIPA Request	Un agent demana una acció i es retorna un resultat
FIPA Query	Un agent realitza una pregunta
FIPA Contract Net	Els agents negocien, es proposen i s'avaluen propostes
FIPA Iterated Contract Net	Igual que l'anterior, però permet renegociar
FIPA Propose	És una simplificació del FIPA Contract Net
FIPA Brokering	Gestiona serveis que els agents poden fer servir
FIPA Recruting	Variant del FIPA Brokering
FIPA Subscribe	Permet que els agents puguin ser notificats de fets rellevants

Taula 8: Protocols

Com era d'esperar, els agents implementen un sistema de missatges prou extens per tal d'aconseguir una comunicació fluida i dinàmica. En el nostre treball traurem molt de profit d'aquesta característica, sobre tots dels protocols FIPA-REQUEST i FIPA-CONTRACT-NET que ens permetran, que agents puguin sol·licitar accions per rebre contestes i alguns d'ells, puguin oferir propostes per ser avaluades per altres agents.

Arribats a aquest punt, ens sorgeix una qüestió. Existeix alguna eina que permeti desenvolupar un sistema multiagent ?. I si és així, és una eina necessària en el nostre treball ?.

4. JADE

La plataforma JADE (*Java Agent Developement Enviroment*) és una eina de programació, que ofereix unes llibreries construïdes sota el llenguatge de programació JAVA i que permet el desenvolupament de sistemes multiagent. Proporciona funcions per controlar agents, a un nivell baix de programació i ofereix interfícies per facilitar la programació del conjunt, dintre d'un entorn d'execució visual, on els agents poden comunicar-se.
Degut a les característiques que proporciona JADE, la considerem una eina necessària en el nostre estudi. Per altre banda, l'estudi en profunditat de JADE

queda fora del nostre objectiu, però donarem petites pinzellades per tal d'assolir i entendre certs conceptes bàsics.

4.1 Components principals

Si donem una ullada ràpida als paquets de JADE, veurem 5 llibreries importants:

Llibreria	Contingut
Jade.core	És el nucli de creació d'agents i inclou la classe *behaviour* (comportaments que inclouen els agents)
Jade.domain	És un paquet que permet representar la plataforma d'agents. Inclou models de domini, llenguatges i ontologies
Jade.content	És un paquet que permet l'especificació de la comunicació (ACL). També inclou classes per la implementació d'ontologies.
Jade.tools	Eines que ofereixen aplicacions extra pel desenvolupament de la plataforma
Jade.proto	És un paquet que inclou els protocols d'interacció entre agents. (FIPA)

Taula 9: Paquets de Jade

En la taula anterior, s'ha especificat que en la llibreria "*Jade.tools*" existeixen diferents eines d'ajuda al control i desenvolupament de la plataforma, anem a resumir-les breument:

Sniffer: És un agent que ens ofereix visualment l'intercanvi de missatges entre els agents de la plataforma. Molt útil per fer un seguiment visual.
Remote Management Agent (RMA) : És un agent encarregat de la interfície gràfica de JADE. Ofereix administració i gestió dels agents i dominis.
Dummy Agent : És una eina que permet depurar, ja que ofereix la possibilitat de crear missatges ACL i fer enviaments a altres agents de forma visual.
La plataforma JADE proporciona un entorn definit i estructurat, dividit per uns contenidors. Dintre de cada un d'aquests contenidors, s'executen els agents. Es a dir, els agents que s'executen en un mateix contenidor, pertanyen al mateix domini.

Tota plataforma de JADE incorpora dos agents imprescindibles:

Domain Facilitator (DF) : És un directori on es registren els serveis, que proporcionen els agents donats d'alta en la plataforma. Habitualment, al DF se li atorga el nom de "pàgines grogues".
AGENT MANAGEMENT SYSTEM (AMS) : És un servei, que registra les direccions dels agents que es donen d'alta en la plataforma. Vulgarment, s'anomena "pàgines blanques".

En el nostre projecte, aquests dos agents seran vitals i tornarem a parlar d'ells en el punt 5 d'aquest document, on entrarem de ple, en el disseny del projecte.

4.2 Creació i comportaments dels agents (Behaviours)

JADE ens proporciona una forma molt ràpida de crear agents des de JAVA. Només s'ha d'estendre qualsevol classe a la classe " *Agent* ". D'aquesta forma ja tindrem accés als mètodes per la propietat d'herència.
Els mètodes que s'han de tindre en compte són:

Setup() : Aquest mètode inicialitza l'agent.
Takedown() : Aquest mètode finalitza l'agent.

Com s'ha parlat reiteradament en aquest document, una de les característiques més importants dels agents, són els comportaments. Els comportaments proporcionen funcionalitats que l'agent converteix en accions. Es a dir, ofereixen les reaccions que l'agent executarà davant estímuls (missatges) de l'exterior. Veurem els comportaments amb més detall, quan entrem de ple en el disseny del nostre projecte, però d'entrada es pot avançar que existeixen dos grans grups:

Comportaments simples : Són aquells que no poden derivar amb un o més comportaments. Es poden trobar comportaments que s'executen cíclicament (*CyclicBehaviour*) o una sola vegada (*OneShotBehaviour*). Altres que esperen la recepció d'un missatge (*ReceiverBehaviour*) o d'altres que esperen l'enviament d'un missatge (*SenderBehaviour*).

Comportaments compostos : Són aquells que sí poden derivar-se en altres comportaments. Per exemple, comportaments que executen els seus subcomportaments de forma seqüencial (*SequencialBehaviour*).

4.3 Sistema de missatges ACL

En el punt 3.5.3 d'aquest document, es va realitzar una introducció al sistema de comunicació entre agents, amb l'objectiu d'introduir al lector en la part tècnica de la formació de missatges.
Cada agent té una cua de missatges entrants que pot llegir a la seva voluntat, es a dir, un agent pot llegir el primer missatge de la cua o el primer missatge que satisfaci un requisit. Aquesta cua, és única per a cada agent i a més, compartida per a tos els seus comportaments.
Sota aquesta tessitura, ja és el moment de veure com un agent pot enviar o rebre un missatge.
Bàsicament utilitza dos mètodes :

Send (ACLMessage msg) : Una vegada construït l'objecte missatge "*msg*" amb les característiques que coneixem (punt 3.5.3) ja es pot enviar mitjançant la funció "*send*".
ACLMessage receive() : Aquesta funció permet la recepció d'un missatge. Val a dir que l'agent agafarà el primer missatge de la cua de missatges, encara que es pot especificar quin tipus de missatges espera l'agent.

4.4 Creació bàsica d'ontologies

Les ontologies[11] juguen un paper molt important en la forma en que es comunica un agent, perquè mitjançant ontologies es pot implementar un llenguatge propi. Això obre les portes a una comunicació molt més acotada a les característiques del nostre projecte, perquè els missatges estaran estructurats, segons un llenguatge comú.
JADE proporciona eines per dissenyar i implementar ontologies, mitjançant classes específiques.
Existeixen altres eines en el mercat, com pot ser *protégé*[12], que també permeten el disseny d'ontologies. *Protégé*, és una eina potent que permet construir ontologies des de zero, d'una mida considerable i a més, d'una forma molt visual. Encara que es va avaluar la possibilitat d'utilitzar *protégé* per realitzar l'ontologia del nostre projecte, finalment es va optar per fer servir les eines de JADE. Aquesta decisió va ser derivada per dos motius principals: El primer, perquè es volia aprofundir en la mesura del possible, en el potencial de la plataforma JADE i el segon, perquè una vegada dissenyada l'ontologia, no es va observar que fos excessivament gran, com per utilitzar un programari específic.

Una ontologia, està basada pràcticament en 3 elements principals:

Conceptes (Objectes) : Són els elements bàsics, entitats que representen un concepte.
Predicats : Són expressions que relacionen els conceptes per dir alguna cosa.
Accions : Són accions que els agents poden dur a terme.

Els missatges que els nostres agents enviaran-rebran, estaran estructurats segons la nostra ontologia. Però hi ha un aspecte molt important que s'ha de tindre en compte. Un Concepte mai pot ser enviat o rebut per un agent. Els agents rebran o enviaran, predicats o accions, que contindran encapsulats els conceptes (informació específica que es vol comunicar).
En el nostre projecte s'ha utilitzat una ontologia creada des de JADE. Entrarem amb molt més detall en breu i es detallarà amb molta més profunditat l'estructura escollida, però ens ha semblat interessant avançar els conceptes bàsics.

4.5 Protocol per a la comunicació: FIPA

El protocol de comunicació dels agents, ens permet posar en pràctica tots els conceptes vistos fins ara. JADE permet implementar una gran varietat de protocols, però ens centrarem en els utilitzats en aquest projecte. Com s'ha fet en els anteriors punts, donarem una informació bàsica i deixarem les parts més específiques de la nostra implementació, per veure-les en la part de disseny.

FIPA-REQUEST : Permet que un agent demani realitzar una acció a un altre agent. L'emissor enviarà la petició, el receptor rebrà la petició i valorarà si la pot realitzar. Si la pot fer, enviarà un missatge de confirmació a l'emissor (tipus "*agree*"), per informar-li que la pot fer i seguidament la farà. Si no la pogués

[11] Si necessiteu mes informació sobre ontologies o voleu ampliar conceptes, podeu visitar el següent enllaç: https://es.wikipedia.org/wiki/Ontología_(informática)
[12] Protégé, lloc oficial: https://protege.stanford.edu/

realitzar, enviarà a l'emissor un missatge tipus "*refuse*". Una vegada, el receptor finalitzi l'acció encomanada, enviarà un missatge "*inform*" per indicar que ja ha finalitzat.

FIPA-CONTRACT-NET : Permet una negociació entre un o més agents. L'agent emissor envia una petició. El receptor avaluarà la petició i enviarà una proposta. L'agent emissor escollirà la proposta que s'adapti a les necessitats i acceptarà una. Informarà a l'agent receptor de que ha sigut escollit.

5. Disseny SMA

Entrem de ple en el disseny del nostre treball. A mesura que avancem, anirem recuperant parts dels punts anteriors i anirem ampliant i especificant la seva aplicació en el nostre projecte. Per això, ens assembla clarament justificable les introduccions bàsiques anteriors.

5.1 Descripció detallada del domini

El nostre objectiu principal és modular un escenari real mitjançant un sistema multiagent (d'ara en endavant SMA). L'escenari escollit, és el servei de transport de paquets en una ciutat, mitjançant vehicles intel·ligents.

Es vol optimitzar el servei de tal forma, que s'aconsegueixi la màxima autonomia i eficiència possible. Començant per les peticions de clients envers a un servei, fins a la gestió dels propis vehicles. Es a dir, aconseguir que el sistema estigui automatitzat de tal forma, que no existeixi una implicació humana generalitzada. Finalment, el sistema podria substituir l'actual sistema de transport de paquets, per un servei informàtic autònom.

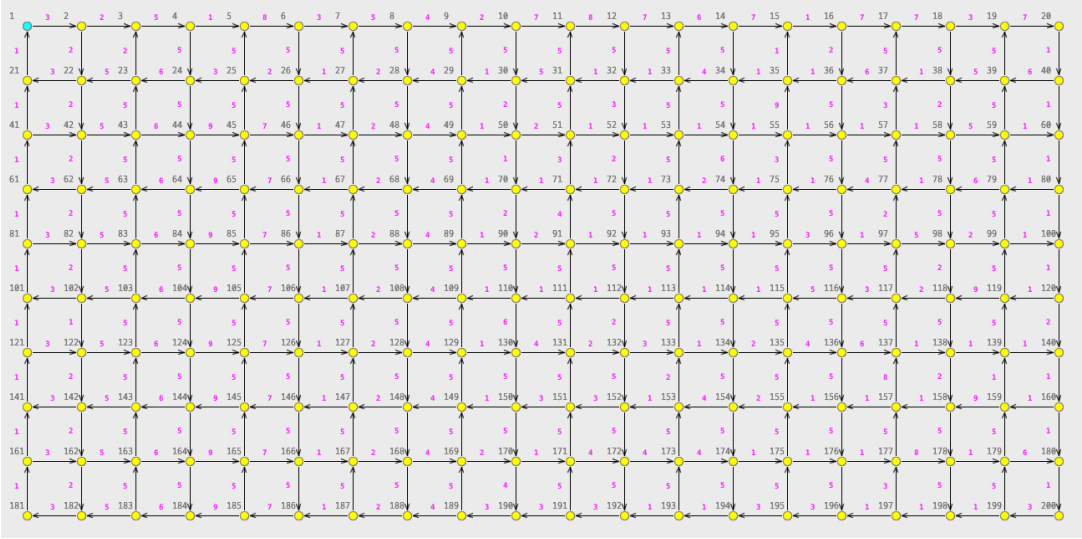
El nostre escenari es centra en una ciutat. Partim de la base que la ciutat te repartits uns punts digitals, que permeten la comunicació (accés a la xarxa). No cal que consirem una ciutat completament intel·ligent, però si una ciutat que pugui obrir les portes, als dispositius que pertanyin al seu domini.

Tenim a la nostra disposició vehicles intel·ligents. Els vehicles contindran una IA pròpia i un sistema de comunicació, es a dir, considerem que els vehicles, gràcies a la seva lògica, són capaços de circular de forma autònoma per una determinada ciutat (sense conductor i mitjançant una cartografia que rebran per part del nostre sistema). La seva autonomia vindrà donada per l'energia elèctrica.

Un client que vulgui realitzar un servei de transport, es posarà en contacte amb un agent central (*AgentCentral*), mitjançant un dispositiu digital, que contindrà un agent personal (*AgentClient*). L'agent central, rebrà la petició del client i l'avaluarà. Això vol dir, que primer mirarà si existeix la possibilitat de realitzar el servei. Si es pot fer, enviarà la petició als vehicles que estiguin disponibles (*AgentCar*). Si en aquell moment no es pot realitzar el servei, el client entrarà en una llista d'espera.

Els vehicles, quan entrin o surtin del servei, comunicaran el seu estat a l'agent central i demanaran a un servei de tràfic (*AgentTrafic*), el plànol de la ciutat. D'aquesta forma, es podran situar en l'escenari concret.

Per reduir la complexitat del model, el plànol d'una determinada ciutat serà un graf dirigit. En aquest cas, els nodes del graf seran els punts de posició dels vehicles, dels clients i naturalment, la posició de destí d'un paquet. Les arestes, simularan els carrers que en el nostre cas seran d'una única direcció. Per simular la distància d'un node a un altre, s'aplicaran pesos a les arestes.



Il·lustració 4: Plànol que s'utilitzarà en les proves

Els nodes estaran nominats per nombres naturals, que simularan noms de posicions diferents dintre d'una ciutat, es a dir, per anar d'un lloc a un altre, direm per exemple, que anem de la posició 1 a la posició 30.

Quan els vehicles rebin la petició de l'agent central, avaluaran el cost del servei i ho faran segons la situació on es trobin els vehicles, el client i el destí del paquet (calcularan el cost de tot el trajecte, mitjançant l'algoritme *Dijkstra*[13]) . És a dir, els vehicles calcularan el cost d'anar a cercar al client i el cost d'anar, d'on és el client fins on s'ha de portar el paquet, mitjançant el camí més curt. Quan tinguin el cost final, enviaran la proposta a l'agent central.
En el nostre cas, l'estratègia de transportar un paquet recau en els vehicles i ho fan, mitjançant una única estratègia. És a dir, si reben el consentiment de l'agent central, aniran a cercar el paquet i ho portaran a les posicions demanades, sense tindre en compte cap altra situació.

L'agent central, rebrà totes les propostes dels vehicles i escollirà la proposta que sigui més interessant pel client (cost mes baix, ja que el preu anirà en funció del cost). Informarà al vehicle que hagi estat seleccionat i li assignarà el servei. Per últim, una vegada el servei estigui validat, *AgentCentral*, confirmarà al client la seva execució.

[13] Explicarem la necessitat d'aquest algoritme en aquest treball, però si es vol més informació sobre aquest algoritme i profunditzar una mica més en el seu funcionament es pot visitar el següent enllaç: https://es.wikipedia.org/wiki/Algoritmo_de_Dijkstra

El sistema també tindrà un agent de control (*AgentControl*), que realitzarà la tasca d'emmagatzemar informació del sistema. Emmagatzemarà en un arxiu persistent, dades de cada servei que es realitzi amb èxit (posició inicial del vehicle, posició destí del paquet, posició del client, nombre de nodes totals recorreguts i cost). Entrarem amb molt més detall en la necessitat d'emmagatzemar aquesta informació, en la part de disseny d'aquest treball. Però, per ara avancem que serà una base de coneixement, que anirà creant el propi sistema, per tal d'implementar un nou sistema de selecció de peticions.

En primera instància, el sistema pot treballar en mode lineal. Es a dir, quan un client realitza una petició, se l'assigna un vehicle i s'executa el servei. A mesura que arriben clients, es van assignat vehicles fins que tots ells estiguin ocupats. Si en un moment determinat, arriba un client i no hi ha vehicles disponibles, en comptes de refusar al client, aquest serà emmagatzemat en una llista d'espera. Quan un vehicle quedi disponible, automàticament s'extrau el primer client que va entrar en la llista d'espera i se l'assignarà aquest vehicle (funcionament FIFO[14]).

Ara bé, aquest sistema de selecció de clients pot ser que no sigui del tot eficient. A priori, es pot pensar que és just, que el primer client que entra en la llista, és el client que ha de ser atès en primera instància. Però, analitzant aquest fet, ens podem trobar en la tessitura, que un determinat client demana un servei amb un cost que serà alt i que darrera d'ell existeix un altre client que demana un servei que serà més baix. I atenció, diem "serà", perquè a priori, el sistema no pot esbrinar el cost del servei d'uns clients situats en la llista d'espera, ja que no ha rebut les propostes dels vehicles envers a aquests nous clients.

Si bé, el disseny i implementació d'aquest projecte, ja representa un nivell alt de dificultat, es vol anar més enllà, dissenyant i analitzant un sistema de predicció de selecció de clients. Es comprovarà si és efectiu trencar l'execució lineal de la llista de clients en espera, per millorar el funcionament general del sistema, cercant l'equilibri entre eficiència i experiència d'usuari.

Per aconseguir-ho, primer ens centrarem en el disseny general (característiques dels agents, comunicació, cooperació i emmagatzematge d'informació), per tal de fer efectiu un sistema lineal. Després, plantejarem un sistema de selecció de clients, mitjançant prediccions. Es a dir, utilitzant les dades emmagatzemades del propi sistema, es llençarà una predicció per cada client en espera. Finalment, es donarà preferència als clients en espera, que tinguin una predicció "*High*" (cost baix) , seguits dels clients amb una predicció "*Middle*" (cost entre alt i baix) i finalment, els clients que tinguin una predicció "*Low*" (cost alt).

En conseqüència, el sistema haurà de treballar en mode lineal o de predicció de forma correcte i s'ha de poder extraure informació rellevant, per analitzar els avantatges i els inconvenients de cada mode.

Finalment, per aconseguir apropar-nos a escenaris reals, és necessari considerar la implementació d'alguns incidents ocasionals. Per exemple, en el

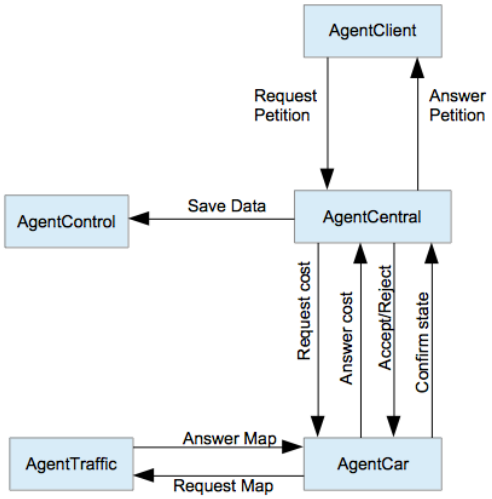
[14] FIFO més informació en: https://es.wikipedia.org/wiki/First_in,_first_out

plànol de la ciutat poden aparèixer carrers tallats (en el plànol de la ciutat, un carrer tallat s'identifica perquè no es veurà el carrer). Això provocarà que els vehicles tinguin que plantejar camins alternatius. Un altre incident que s'ha plantejat, és l'averia o imprevist d'un determinat vehicle. Si un vehicle, de forma ocasional entra en l'estat d'incident, s'enviarà un substitut per resoldre el servei.

Seguidament, anem a veure l'arquitectura del nostre SMA.

5.2 Arquitectura del sistema

A continuació es mostra l'arquitectura del SMA. En la següent figura no es mostra tot el contingut dels missatges que s'intercanvien entre els agents, només és una visió general de l'objectiu final. Recordem, que els agents han de coordinar-se per tal de realitzar un servei determinat i és això, el que es vol mostrar en la següent figura.



Il·lustració 5: Arquitectura general del SMA proposat

5.3 Descripció dels agents

Una vegada vist el domini en detall i l'arquitectura del nostre SMA, arriba l'hora de veure la descripció i funcionament de cada agent. En els següents punts, es veurà d'una forma detallada, les tasques i responsabilitats de cada agent. A més, la seva política de treball.

5.3.1 AgentCentral

Com el seu nom indica, és un dels agents més importants del nostre SMA. Cal recordar, que l'objectiu final del nostre projecte és optimitzar un servei de transport i per tant, tots els agents del nostre sistema són importants per aconseguir-ho. Ara bé, segons el nostre disseny, hi ha agents amb un pes de càrrega més important i l'*AgentCentral,* és el cas.

Les seves tasques són:

- Rebre peticions dels clients.
- Informar del resultat del servei al client.
- Confirmar al client, el final d'un servei.
- Escollir la millor oferta de servei pel client.
- Rebre informació de l'estat dels vehicles.
- Mode de funcionament de selecció de clients en espera.
- Càlcul de predicció de clients en espera.
- Enviar vehicle de suport, en cas d'incident d'un altre vehicle.
- Enviar informació a l'*AgentControl*.

Aquest agent, en primera instància es troba a l'espera de peticions de clients i actualitzacions d'estat dels vehicles.
En el cas, de les peticions dels clients, quan aquest agent rep una petició, aquesta és analitzada. Verifica que hi hagi vehicles disponibles per realitzar el servei. Si no hi ha vehicles disponibles, introdueix al client en una llista d'espera. En el cas contrari, llença la petició a cada vehicle, per rebre una oferta de cost. El cost, és la suma de la distància, d'on es troba el vehicle fins arribar al client i la distància que hi ha, des de el client fins al destí final.
Quan tots els vehicles disponibles contesten a la petició, l'*AgentCentral* escull el cost més baix de tots (distància més curta). Seguidament, informa al vehicle seleccionat, que ha de realitzar el servei.
Al cost rebut i seleccionat, aquest agent li aplica el preu, que és (cost * euros) i li fa arribar al client tota la informació.
També, està a l'espera d'actualització d'estat d'un vehicle. Quan un vehicle comença a treballar, comença un servei o torna a estar disponible després d'un servei, l'*AgentCentral* rep una actualització (situació del vehicle i estat) i l'emmagatzema en una taula.
És possible, que un vehicle faci arribar, mitjançant el seu estat que ha tingut un incident (averia). En aquest cas, l'*AgentCentral* el marcarà com a no disponible i donarà d'alta un altre vehicle, situat en una llista de vehicles de suport. Recuperarà la petició inicial (servei que havia de realitzar el vehicle accidentat) i modificarà les dades, per a que un altre vehicle la pugui realitzar-la.

AgentCentral pot treballar en dos modes diferents de selecció de clients, mode lineal i mode predictiu. El mode lineal, consisteix en introduir clients en una llista d'espera, quan no hi ha vehicles disponibles. Una vegada torna a existir un vehicle disponible, el primer client que va entrar en la llista d'espera, és el primer client en ser atès.
El mode predictiu, consisteix en seleccionar el client de la llista d'espera que tingui una predicció de cost de servei baix. És a dir, per a cada client que hi ha a la llista d'espera, es realitza una predicció envers a unes dades, que el propi sistema emmagatzema en cada servei. Si la predicció diu que el client demanarà un servei de cost baix, se li dona prioritat de sortida de la llista. Existeixen 3 tipus de prioritats: "*High*" prioritat alta, perquè es preveu un cost baix de servei. "*Middle*" prioritat mitja, perquè es preveu un cost de servei que està entre, una prioritat alta i una baixa. Finalment, "*Low*" prioritat baixa, perquè es preveu un cost de servei alt.
Sigui un mode o un altre, quan s'executa un servei correctament, les dades del servei són enviades a l'*AgentControl*.

5.3.2 AgentClient

Aquest agent és qui comença tot el procés de servei. Les seves tasques són:

- Carregar la seva configuració.
- Enviar peticions de servei a l'*AgentCentral*.
- Rebre informació del servei des de l'*AgentCentral*.
- Mostrar gràficament la informació del servei.

Quan un *AgentClient* s'inicia, el primer que fa és carregar la informació que identifica al client i elaborar una petició. En aquesta petició s'inclou: la posició on es troba el client, la posició on es vol enviar el paquet i la quantitat de paquets. El nom del client coincideix amb el nom del propi *AgentClient*, per tal de reduir la complexitat.
Seguidament, envia la petició a l'*AgentCentral*. Una vegada enviada, espera la resposta de l'*AgentCentral*. Pot rebre alguns missatges diferents.
Missatge d'espera, que indica que no hi ha vehicles disponibles i que serà inclòs en una llista d'espera.
Missatge d'informació del servei, on rebrà el vehicle assignat, el cost del servei, el preu, el recorregut que realitzarà el vehicle fins arribar al client i el recorregut que realitzarà el vehicle fins al destí.
Missatge de refús, on s'indica que la petició no es podrà atendre, perquè la posició on vol enviar el paquet és inaccessible.
Finalment, un missatge de confirmació on es mostrarà mitjançant una petita interfície gràfica, la informació del resultat de la petició.

5.3.4 AgentCar

Aquest agent és l'encarregat de transportar el paquet d'una posició a una altra. Les seves tasques són:

- Carregar la seva configuració.
- Demanar el plànol de la ciutat a l'*AgentTraffic*.
- Rebre el plànol de la ciutat des de l'*AgentTraffic*.
- Enviar a l'*AgentCentral* el seu estat.
- Enviar situació d'incident.
- Espera rebre peticions de l'*AgentCentral*.
- Elabora ofertes envers a les peticions i a la seva pròpia situació en el plànol.
- Enviar una proposta d'oferta a l'*AgentCentral*.
- Rebre l'acceptació o el refús de la proposta enviada.
- Realitzar el servei.

Quan un *AgentCar* s'inicia en el sistema, la seva primera tasca és carregar la seva configuració. Aquesta informació identifica les característiques pròpies de cada vehicle.
Seguidament, demana a l'*AgentTraffic*, el plànol de la ciutat i una vegada rebut, envia a l'*AgentCentral* el seu estat, que inicialment és de disponible per realitzar un servei.

En aquest punt es queda a l'espera de rebre una petició, que una vegada rebuda pot derivar en dues situacions: Que el vehicle rebi una petició i es trobi ocupat, per tant refusarà el servei i l'altre, que accepti la petició. Si l'accepta, s'inicia el càlcul del cost necessari.
Els nostres vehicles utilitzen l'algoritme *Dijkstra* per calcular el camí més curt per anar d'un punt a un altre, ja que els carrers (arestes) tenen assignats valors que simulen distàncies.
Una vegada elaborada la proposta, s'envia a l'*AgentCentral*, per a que l'analitzi conjuntament amb la dels altres vehicles.
Cada vehicle, rebrà una confirmació d'acceptació o de refús de la seva proposta, per part de l'*AgentCentral*. Si la resposta que rep el vehicle és un REJECT-PROPOSAL, el vehicle quedarà disponible per realitzar un altre servei. Pel contrari, si rep una confirmació ACCEPT-PROPOSAL, el vehicle queda assignat al servei i estarà un temps fora de servei (realitzant la petició del client).
Quan el vehicle hagi finalitzat el servei (en el nostre cas, serà un temps determinat en que el vehicle no estarà disponible) , el vehicle enviarà un missatge a l'*AgentCentral*, per informar-lo de la seva posició i disponibilitat.

Val a dir, que en aquest punt, el vehicle quedarà configurat amb les dades reals. Es a dir, la seva posició en el plànol serà la posició on va finalitzar el servei o la posició real en cas d'incident.

Els vehicles poden patir incidents (averia, accident, etc..) durant un servei. En aquest cas, un vehicle canviarà el seu estat a accidentat i farà saber a l'*AgentCentral*, si ha sigut en el recorregut d'anar cap al client o en el recorregut cap al destí (li farà arribar la posició exacta de l'accident).

5.3.5 AgentTraffic

Aquest agent és l'encarregat de lliurar el plànol de la ciutat. Les seves tasques són:

- Carregar la seva configuració (en aquest cas carregarà un plànol d'una determinada ciutat i una llista de possibles carrers tallats).
- Esperar peticions de l'*AgentCar*.
- Respondre les peticions de l'*AgentCar*, lliurant el plànol d'una ciutat.

Una vegada s'inicia l'*AgentTraffic* en el sistema, la seva primera tasca és la de carregar una configuració inicial. En aquesta configuració, es carreguen les dades que determinen un plànol d'una ciutat i si hi ha, una llista de carrers tallats. En cas que hi hagin carrers tallats, modifica el plànol avanç d'enviar-lo als vehicles.
Seguidament, es queda a l'espera de rebre peticions de l'*AgentCar*. Quan rep una petició, respon amb la informació del plànol.
Entrarem amb molt més detall, en la configuració del plànol i com és interpretat en el punt 6 Implementació, però ara mateix es pot avançar que estarà constituït per dues parts: una llista que contindrà objectes amb informació dels nodes i les arestes (posicions i carrers) i una segona, que proporcionarà el nombre de nodes totals (nombre total de posicions).

5.3.6 AgentControl

Aquest agent és l'encarregat de emmagatzemar informació del sistema. Les seves tasques són les següents:

- Carregar la seva configuració personal. En aquest cas, mostra una petita informació per pantalla, presentant al sistema en general.
- Crear, si és necessari, un arxiu de text persistent (arxiu de coneixement).
- Rebre peticions de l'*AgentCentral*, per emmagatzemar informació.

Quan s'inicia l'*AgentControl*, es presenta el sistema i elabora un arxiu de text persistent, per emmagatzemar dades. Aquest arxiu de text, te una extensió *.arff* i és així, perquè serà la base de coneixement necessària per ser utilitzada en el cas del mode de funcionament predictiu. Si l'arxiu ja està creat (per exemple, perquè el sistema ja havia estat en funcionament en una altre execució), l'*AgentControl* no l'esborrarà i continuarà emmagatzemant informació a continuació de la que ja hi havia.
Una vegada creat l'arxiu (si ha sigut necessari) , l'agent queda a l'espera de rebre peticions de l'*AgentCentral*. Quan rep una petició, extrau el seu contingut i escriu la informació en l'arxiu de text *.arff*.
La informació que emmagatzemarà és: El punt on es troba el vehicle que ha realitzat el servei, el punt on s'ha portat el paquet (punt de destí), el punt on es trobava el client, el nombre total de nodes recorreguts i la prioritat real del servei ("High", "Middle", "Low").

5.3.7 ConfigAgent

En aquest cas ens trobem davant d'un agent, que no te una característica activa en el sistema. És un agent base i d'ell s'estendran els altres agents. La seva funció és la de facilitar certes funcions comuns als altres agents. Amb aquest fet es pretén reduir codi de programació, ja que ens aprofitarem del concepte d'herència de les classes *Java*.
Ho veurem amb més detall en el pròxim capítol 6, on descriurem les seves característiques i el perquè del seu ús, en la implementació.

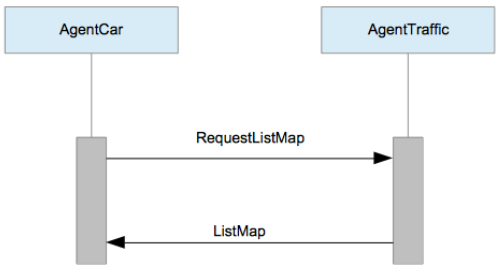
5.4 Intercanvi de missatges entre agents

Com ja s'ha parlat, en la part teòrica d'aquest treball, per tal de realitzar certes tasques de coordinació, els agents necessiten comunicar-se entre ells. En el nostre cas i per la topologia del projecte que ens proposem, els agents que constitueixen el sistema han de comunicar-se entre ells, per tal d'aconseguir l'objectiu final. Anem a detallar, les comunicacions que ens trobarem entre els diferents agents.

- Comunicació entre l'*AgentCar* i l'*AgentTraffic*.

Quan un *AgentCar* inicia la seva activitat, és necessari que tingui el plànol de la ciutat, per tal de que pugui localitzar els recorreguts on realitzarà el servei. Així, una vegada entra en el sistema, l'*AgentCar* llença una sol·licitud (

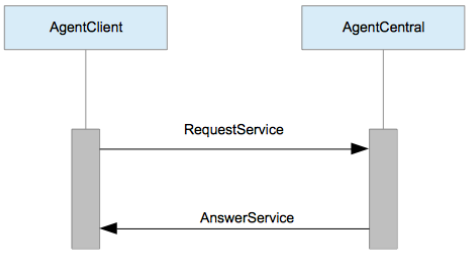
RequestListMap) a l'*AgentTraffic* i aquest li respon amb la informació demanada (*ListMap*).



Il·lustració 6: Comunicació AgentCar - AgentTraffic

- Comunicació entre l'*AgentClient* i l'*AgentCentral*.

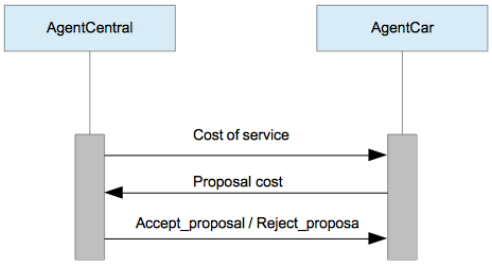
Quan un *AgentClient* realitza una petició a l'*AgentCentral*, aquest rebrà una contesta amb la informació del servei.



Il·lustració 7: Comunicació AgentClient - AgentCentral

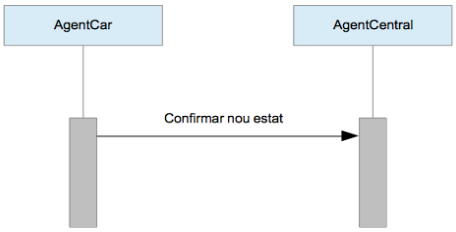
- Comunicació entre l'*AgentCentral* i *AgentCar*.

Quan l'*AgentCentral* vol obtenir les ofertes dels vehicles que estiguin disponibles, llençarà un missatge a cada vehicle. Aquests respondran amb les seves propostes i quan l'*AgentCentral* esculli una, enviarà un altre missatge per fer saber als vehicles, qui d'ells ha estat seleccionat i qui no.



Il·lustració 8: Comunicació AgentCentral - AgentCar

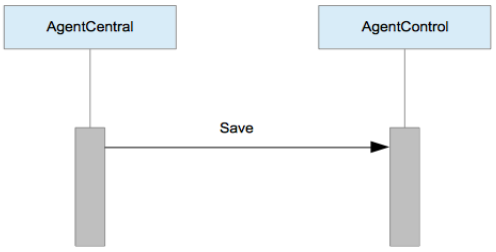
Quan l'*AgentCar* necessiti actualitzar el seu estat, ho farà saber a l'*AgentCentral*. Això passarà quan l'*AgentCar* entri en el sistema, comenci un servei o finalitzi un servei.



Il·lustració 9: Comunicació AgentCar - AgentCentral

- Comunicació entre l'*AgentCentral* i *AgentControl*.

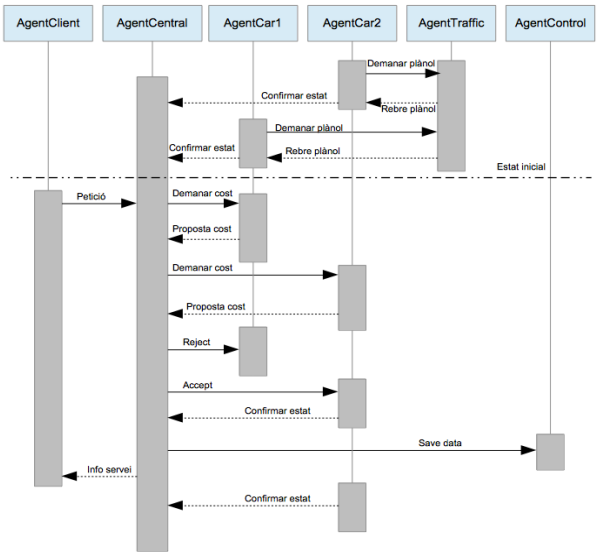
Quan es realitza un servei complert, és necessari emmagatzemar les dades del servei, per un posterior anàlisis.



Il·lustració 10: Comunicació AgentCentral - AgentControl

- Resum bàsic i general dels missatges entre tots els agents del sistema.

El funcionament bàsic comença després de l'estat inicial (línia de punts horitzontal). En aquest cas, ja es dona per suposat que els vehicles han demanat el plànol i han confirmat el seu estat a l'*AgentCentral*.
El client inicia el procés, enviant la seva petició i es pot veure, d'una forma temporal, com succeeixen els diferents intercanvis de missatges, fins arribar al final de la seva execució.



Il·lustració 11: Comunicació general entre agents

5.5 Disseny de l'ontologia

En el punt 4.4, ja es va avançar la part teòrica que determina una ontologia i es va justificar la necessitat del seu ús. Una vegada vista l'arquitectura del nostre projecte i els intercanvis de missatges que tindran lloc en el SMA, és fa evident estructurar un vocabulari comú i que permeti una comunicació exclusiva entre els diferents agents. Recordem que els missatges són accions o predicats i que el contingut intern del missatge són els conceptes de l'ontologia, que es volen transmetre.
Una tasca inicial, a l'hora de dissenyar l'ontologia, és determinar els serveis i propietats de cada un dels agents que hi participen. Aquesta informació, en cas de necessitat ens ajudarà a cercar un determinat agent o un conjunts d'ells. Això serà molt pràctic, perquè els nostres agents donaran d'alta els seus serveis en el DF i per tant, només s'haurà de consultar aquest directori, per cercar un determinat agent.
Cada agent, quan es dona d'alta el seu servei, proporciona dos paràmetres. El primer, és el nom del servei i el segon el tipus de servei. En el nostre cas, tots els agents proporcionaran el mateix tipus de servei, ara bé, cada un tindrà un nom diferent. Això ens indica, que els agents conviuen en un mateix escenari conjunt, però que cada un proporciona un servei diferent. A continuació, les propietats dels nostres agents i els conceptes més significatius:

TYPE_SERVICE = "sma-transport-services"
SERVICE_CLIENT = "client"
SERVICE_CAR = "car"
SERVICE_TRAFFIC = "tràfic"
SERVICE_CONTROL = "control"
SERVICE_CENTRAL = "Servei-general-de-gestio-del-transport"

- *AnswerPetition*: És un objecte que permet incloure informació, sobre la resposta a una determinada petició de servei. El nom del vehicle assignat, el cost (distància total) del recorregut realitzat pel vehicle en el servei, el seu preu, la llista de posicions que s'han visitat en el recorregut des de la posició on es troba un vehicle fins al client i finalment, la llista de posicions que s'han visitat en el recorregut que ha realitzat el vehicle, des de el client fins al destí del paquet.

Slot	Type
idCar	String
cost	int
price	float
routeCarToClient	List
routeClientToFinal	List

Taula 10: Objecte AnswerPetition

- *Position*: És un objecte que permet incloure informació sobre una determinada posició. Emmagatzema el valor d'un punt concret.

Slot	Type
pointX	int

Taula 11: Objecte Position

- *Petition*: És un objecte que permet incloure informació sobre la petició d'un client concret. Permet emmagatzemar, el nombre de paquets, la posició inicial (posició on es troba el client) i la posició final (posició destí del paquet).

Slot	Type
numpackets	int
positionInitial	Position
positionFinal	Position

Taula 12: Objecte Petition

- *infoNode*: És un objecte que permet incloure informació sobre un determinat node del plànol. Emmagatzema el node inicial, el node final, el pes de l'aresta que els uneix (distància entre nodes) i la direcció de l'aresta. La direcció de l'aresta, s'utilitzarà per mostrar de forma gràfica el plànol. Pot prendre 4 valors, que aniran del 0 al 3: direcció dreta, direcció esquerra, direcció a dalt i direcció a baix, respectivament.

Slot	Type
initial	int
near	int
weight	int
direction	int

Taula 13: Objecte infoNode

- *ListMap*: És un objecte que permet incloure informació general del plànol. En aquest cas, s'emmagatzema una llista de *infoNodes* i la quantitat de nodes totals.

Slot	Type
listmap	List
nVertex	int

Taula 14: Objecte ListMap

- *CarInformation*: És un objecte que permet incloure informació específica de l'estat d'un vehicle.

Slot	Type
idCar	String
positionCar	Position
state	Boolean
incident	Boolean

Taula 15: Objecte CarInformation

Una vegada dissenyats els nostres conceptes, li toca el torn als predicats o accions. En el nostre cas, configurarem accions que permetran encapsular els nostres conceptes en missatges.

- *RequestService*: Aquesta acció permet que un client demani una petició de servei. Com es veu, encapsula un objecte del tipus *Petition*.

Name	Type
RequestService	Petition

Taula 16: Acció RequestService

- *AnswerService*: Aquesta acció permet que es rebi la resposta d'una determinada petició. Com es pot observar, encapsula un objecte del tipus *AnswerPetition*.

Name	Type
AnswerService	AnswerPetition

Taula 17: Acció AnswerService

- *StateCar*: Aquesta acció permet enviar l'estat d'un vehicle. Encapsula el concepte *CarInformation*.

Name	Type
StateCar	CarInformation

Taula 18: Acció stateCar

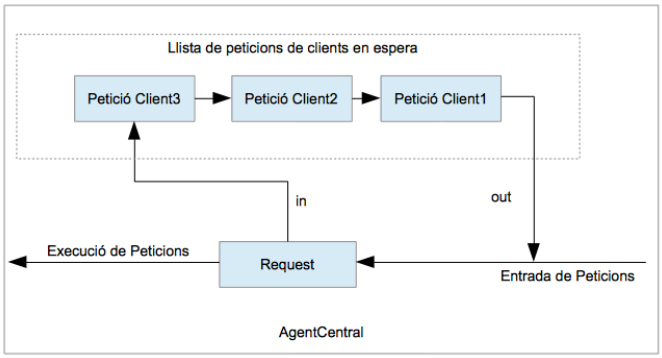
- *RequestListMap*: Aquesta acció permet encapsular informació sobre un plànol. No ens farà falta encapsular cap objecte.

5.6 Disseny del sistema de gestió de clients.

El nostre SMA, te com objectiu gestionar peticions de serveis de clients, que volen enviar un paquet d'una posició a un altra, dintre d'una determinada ciutat. Els vehicles que s'utilitzen, són autònoms i per tant no depenen de gestió humana per realitzar recorreguts. Ara bé, una vegada el sistema comença a donar serveis, ens podem trobar en diferents escenaris.

- Que existeixin prou vehicles, per assolir tots els serveis que els clients demanen.
- Que no hi hagi prou vehicles, per assolir tots els serveis que els clients plantegen.

Per respondre a la problemàtica del primer escenari, és suficient que una vegada entri un client en el sistema i realitzi una petició, aquesta sigui atesa per qualsevol vehicle que estigui disponible.
En el cas del segon escenari, es poden plantejar algunes alternatives. Per exemple, si no hi ha vehicles disponibles, es podria refusar la petició del client i que ell mateix ho torni a demanar, una estona més tard. Aquesta solució no ens gens atractiva, ja que la nostra prioritat és realitzar el servei que el client proposa i no esperar a que sigui el propi client, qui la torni a demanar.
Una altra solució, és la d'incloure al client en una llista d'espera. D'aquesta forma, ens assegurem que el sistema podrà realitzar el servei, una vegada hi hagi vehicles disponibles. Però llavors haurem d'analitzar la política de funcionament d'aquesta llista, ja que ens interessa trobar l'equilibri entre eficiència i experiència d'ús.
Sembla natural, que si un client entra primer en la llista d'espera, aquest sigui el primer en sortir. Això permet respectar l'ordre d'execució de peticions i és el que anomenarem en aquest projecte " mode d'execució lineal " o el que informàticament s'anomena llista FIFO.
Aquest mode ens assegura, que les peticions dels clients seran ateses en el mateix ordre que van ser incloses en la llista d'espera.



Il·lustració 12: Model d'execució lineal

Però, encara que no es vol renunciar a aquest mode de funcionament, volem anar més enllà i ens interessa cercar una solució que sigui més eficient o al menys, que ens doni la possibilitat de realitzar un estudi i una posterior valoració. Per tal de dur a terme aquesta tasca, plantegem el "mode de funcionament predictiu" que serà l'alternativa al " mode d'execució lineal ". El seu funcionament serà el següent:
Quan hi hagi un excés de peticions, aquestes seran incloses en la llista d'espera, de la mateixa forma que en el mode lineal. Ara bé, quan un vehicle torni ha estar disponible, l'*AgentCentral* calcularà, de forma predictiva, el cost d'execució de totes les peticions que estan a l'espera. Ho farà en base a un arxiu de coneixement (que el propi sistema anirà creant a cada servei) i utilitzant un algoritme de classificació.
La base de coneixement, serà un arxiu de text amb extensió *.arff* (anomenat *Control.arff*), que estarà constituït per 5 atributs (4 numèrics i un que serà la classe). S'ha escollit un arxiu amb extensió *.arff*, perquè és un arxiu llegible directament pels algoritmes de classificació que ens ofereixen les llibreries *Weka* (entrarem amb més detall, sobre aspectes de les llibreries, en la part d'implementació d'aquest mateix projecte).

Els atributs escollits i que constituiran aquest arxiu són: posició inicial del vehicle, posició final del vehicle, posició del client, nombre de nodes recorreguts i classe.

- Posició inicial del vehicle **(** *initial_position_car* **):** Aquest atribut ens informa sobre la posició on el vehicle a començat ha realitzar un servei determinat (node inicial).
- Posició final del vehicle **(** *Final_Position_Car* **):** Aquest atribut ens informa sobre la posició on el vehicle a finalitzat un servei determinat (node final).
- Posició del client **(** *Position_Client* **):** Aquest atribut ens informa sobre la posició del client en un determinat servei (node on està situat el client).
- Nombre de nodes recorreguts (Nodes) : Aquest atribut ens dona informació sobre el nombre total de nodes que s'han recorregut en un servei.
- Classe **(** *Class*): Aquest atribut ens informa del cost d'un servei determinat. Els seus valors són: { *High, Middle, Low* }

Quan s'executa una petició i aquesta és donada per bona, el sistema escriurà en l'arxiu *Control.arff,* els valors dels 4 atributs que s'ha descrit anteriorment i la seva classe. La classe vindrà determinada pel cost total del servei que s'hagi realitzat.

En el nostre cas, si un servei ha tingut un cost menor o igual a 30, la classe assignada serà *High*. En el cas que el cost sigui superior a 30 però inferior o igual a 80, s'assignarà la classe *Middle*. Finalment, un servei amb cost superior a 80, se li assignarà la classe *Low*.
Un exemple de servei inclòs en l'arxiu *Control.arff* seria el següent:

11, 2, 18, 31, Low

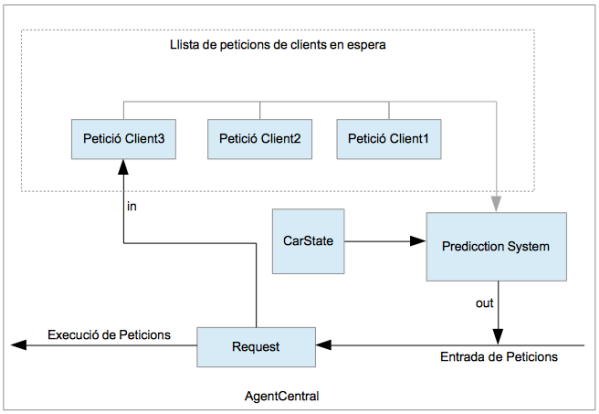
En aquest cas, es pot veure que quan es va iniciar el servei, el vehicle es trobava en la posició 11, el client en la posició 18 i el paquet es va conduir fins a la posició 2. El servei va estar considerat lent (*Low*), perquè el seu cost va ser superior a 80 i com es pot veure es van recórrer 31 nodes (diferents posicions).
S'ha inclòs el nombre real de nodes recorreguts per una raó important. Pensem que degut a les característiques del problema i la informació que es vol manipular, és molt possible que la dispersió entre els diferents valors sigui tant gran que no permeti crear un sistema prou fiable de classificació. Un valor comú i a l'hora decisiu entre les diferents instàncies, és el recorregut entre les posicions. Si volem un recorregut ràpid (poc cost, *High*) és molt probable que el nombre de posicions recorregudes sigui petita. Aquest nombre anirà creixent a mesura que el cost augmenti i per tant, la diferència entre els 3 tipus serà més destacable.
Ara bé, també pot succeir que si es vol transportar un paquet a una posició propera i el cost entre les diferents posicions és gran, el recorregut final sigui més llarg, ja que el vehicle calcularà un recorregut amb menys cost, encara que això signifiqui visitar més posicions. Això pot incloure distorsió en el model presentat i influir negativament en la seva avaluació. Però, d'entrada acceptem aquest detall i ho estudiarem més endavant, ja que l'objectiu principal de disseny d'aquest apartat, és la d'incloure un sistema de predicció de prioritat en la llista d'espera.
Un petit incís, avanç de continuar per tal d'aclarir conceptes. Quan parlem de cost, parlem de la distància total recorreguda pel vehicle en tot el servei. Així, si considerem que el cost d'anar d'una posició a una altra, és mesurat en kilòmetres, en el cas de l'exemple, el servei hauria superat els 80Km de distància. Aquesta base de coneixement anirà creixent a mesura que es realitzin serveis i quedarà emmagatzemada de forma persistent en el sistema, pel seu futur ús (arxiu *Control.arff*).

Continuant amb la descripció del mode predictiu, toca analitzar la llista d'espera de les peticions dels clients. Les peticions inclouen, la posició del propi client i la posició final del servei (destí del paquet). En el moment de realitzar una predicció, l'*AgentCentral*, proporcionarà en temps real, la posició actual del vehicle que estigui disponible en aquell moment i el sistema predictiu, haurà de preveure, a quina classe pertany la petició que es troba en espera.
Per fer-ho, *AgentCentral* calcularà una aproximació als possibles nodes recorreguts, tenint com a base, la posició del vehicle, la del client, la del paquet i el nombre de posicions màximes per línia horitzontal del plànol. La idea és posicionar amb valors equidistants, les posicions que es troben més enllà de les primeres 20 posicions. Es a dir, un vehicle, quan ha de viatjar de la posició 1 fins a la posició 22, no ha de recórrer 21 nodes, és possible que només hagi de recórrer 2. Això és perquè, cada línia del nostre plànol està constituïda per 20

posicions diferents i quan saltem de línia ho fem de forma consecutiva, per tant, si un node de la primera línia està connectat amb un node de la segona línia, automàticament avancem 20 posicions respecte a l'anterior. S'ha d'anar amb compte amb aquest fet, ja que en aquest moment no parlem de distàncies, estem parlant de posicions.
AgentCentral farà una estimació de recorregut tenint en compte les posicions (no distàncies) i llançarà una predicció tenint en compte aquests atribut conjuntament amb les posicions reals. L'objectiu final d'aquesta tasca, és la d'elaborar un instància el més propera a la realitat, amb la posició del vehicle, client, paquet i nodes que s'hauran de recórrer. Una vegada construïda, es cridarà a un model de classificació que determinarà una determinada classe i amb conseqüència, una certa prioritat dintre de la llista d'espera.

Una vegada hagi realitzat la seva predicció, per a cada una de les peticions que estan a l'espera, donarà prioritat de sortida a aquella que en aquell moment consideri de la classe *High*. Si no hi ha cap, donarà prioritat a la que consideri *Middle* i si no hi ha, donarà pas a la de classe *Low*.
Les prediccions es realitzaran cada vegada que s'assigni un nou vehicle, perquè a cada petició, els vehicles assignats modifiquen la seva posició inicial, ja que han realitzat un servei i per tant, ja no es troben en el mateix lloc.

El motiu d'implementar aquest sistema, és la d'estudiar si s'aconsegueix una millora en el funcionament global del sistema, incidint en la selecció d'entrada de peticions. Un model lineal, a priori, pot semblar més costós, ja que s'executen peticions sense tindre en compte el cost del servei (és possible que en el temps de realitzar un servei, es puguin executar dos de molt més ràpids). En canvi, el sistema predictiu, a priori pot semblar més eficient, ja que en primera instància s'assignen els serveis més ràpids. A més, com els vehicles es mouen a cada servei, és possible que una petició que havia estat considerada *Low* en un estat anterior, en un següent estat estigui considerada com a *High* o *Middle*.



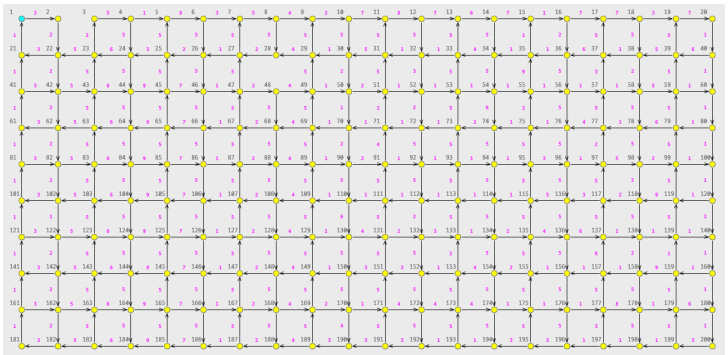
Il·lustració 13: Model d'execució predictiu

En aquest moments, es fa evident que el model predictiu tindrà dos punts febles. El primer, la base de coneixement, que haurà d'anar augmentant la seva mida per tal de que pugui ser considerada i el segon, l'algoritme utilitzat en el càlcul de la predicció.
Pel que fa a la base de coneixement i per tal de que augmenti contínuament, el sistema escriurà en l'arxiu *Control.arff*, tots els serveis que es realitzin

correctament. En primera instància, no ens preocuparem si hi ha entrades duplicades, ja que avanç d'aplicar un algoritme classificador, farem una purga de l'arxiu.
Degut a que, un dels objectius d'interès és veure l'impacte del model lineal envers al model predictiu, no utilitzarem únicament un sol algoritme classificador. Però, degut a que existeixen gran quantitat d'algoritmes i que l'ús de tot ells, donaria pas a un document molt extens, només ens centrarem en uns quants. Ara bé, inicialment utilitzarem l'arbre classificador *RandomTree* que ens ofereix les llibreries *Weka*.
La decisió d'utilitzar *RandomTree*, com a base d'estudi, és que necessitem que el nostre classificador sigui significatiu i fàcilment interpretable. De fet, l'utilitzarem per construir un arbre de decisió que, al nostre criteri, ens permetrà una millor comprensió de les dades. A més, funciona molt bé amb valors nominals i numèrics.
La funció de *RandomTree* serà predir la classe de la nostra petició, es a dir, quin cost tindrà. La petició, està formada per uns atributs (posició del vehicle, posició destí, posició del client i nombre de nodes) i en funció d'aquests valors, es classifica en un tipus de classe o un altra. *RandomTree* crea un arbre, on cada node correspon a un dels atributs de la nostra petició. Cada node pot generar altres nodes o fulles, on les fulles seran els valors de les etiquetes de cada node. Aquesta característica visual i el fet que és un algoritme, que amb valors numèrics treballa de forma eficient, el fa un bon candidat pel nostre treball.

5.7 Disseny d'incidències.

El nostre SMA te com objectiu representar el servei de transport de paquets, en una determinada ciutat. Aquesta ciutat te una tipologia bàsica, com tota ciutat (posicions i carrers) i per tal d'afegir realisme a les nostres simulacions, afegirem l'opció de tallar un nombre de carrers. Això provocarà que alguns vehicles, per tal de realitzar un mateix servei, tinguin que modificar el seu recorregut i conseqüentment el cost del servei.
Els carrers tallats, els representarem sense dibuixar el propi carrer en el plànol.



Il.lustració 14: Plànol amb carrers tallats

Com es pot veure en l'anterior il·lustració, els carrers 2→3 i el 28→ 48 estan tallats i els vehicles hauran de cercar alternatives.

També afegirem el cas, en que un vehicle tingui un incident. En aquest escenari, *AgentCentral* donarà d'alta un altre vehicle (que estarà situat en una llista de vehicles de suport) i marcarà el vehicle que ha tingut l'incident, com a no

disponible. La petició que el vehicle accidentat no ha pogut realitzar, passarà a la llista de peticions en espera, degudament configurada amb les posicions adequades al nou escenari. Es a dir, pot ser que el vehicle accidentat hagi tingut l'incident avanç d'arribar al client o després de recollir el paquet. Inclòs pot haver tingut l'averia en els punts inicials o finals de cada recorregut. Si l'averia ha tingut lloc avanç d'arribar al client, el nou vehicle anirà directament a cercar al client. Si l'averia a tingut lloc després de recollir el paquet, el nou vehicle anirà a cercar el vehicle avariat, recuperarà el paquet i després continuarà el camí fins a destí.

6. Implementació

Entrem de ple en la primera part pràctica del projecte. Tots els punts que constitueixen aquesta part, contindran aspectes relatius a l'estructura i implementació del codi del SMA proposat. És ara, quan anirem desvetllant algunes de les qüestions que ens hem anat plantejat, en els diferents punts anteriors.
Per realitzar tota aquesta part, s'ha utilitzat un maquinari i un software que passem a descriure a continuació:

- Maquinari: MacBook Pro 17", i7 a 2.66GHz amb 8Gbytes de Ram DDR3. El sistema operatiu macOS Sierra 10.12.6
- Software: Per l'elaboració del codi JAVA, s'ha utilitzat el Framework NetBeans IDE 8.2[15]. Les llibreries externes JADE 4.5.0 [16] i WEKA 3.8.1[17].

NetBeans és un *Framework* que permet crear projectes i treballar amb diferents llenguatges de programació. En el nostre cas, utilitzarem JAVA, més concretament el *JDK 1.8* per implementar totes les classes necessàries en aquest treball. No entrarem en detalls de funcionament del propi *Framework* i tampoc en la seva configuració, es dona per suposat que existeixen coneixements bàsics.
Pel que fa a les llibreries externes *JADE* i *WEKA*, no tenen més dificultat que la seva importació al projecte, una vegada descarregades. Una vegada importades, ja es poden fer servir les classes necessàries.
Val a dir, que en els dos casos, es pot tindre accés a una interfície gràfica, que permet una experiència d'usuari més agradable. En el nostre cas, utilitzarem la de *JADE* per visualitzar algun cas pràctic, ja que permet veure gràficament la comunicació entre agents. El mateix amb el cas de *WEKA*, que permet veure gràficament aspectes d'alguns dels algoritmes que utilitzarem.

Pel que fa a l'organització del SMA, es fa imprescindible descriure certes qüestions importants. Com es va veure en el punt 3.5.2, l'estructura interna d'un SMA pot estar composta per diferents plataformes i cada plataforma pot tindre diferents agents. A més, agents d'una plataforma poden comunicar-se amb agents d'una altre plataforma. En el nostre cas, els agents conviuran en una mateixa plataforma, per tant en el mateix contenidor. Això vol dir, que en les nostres implementacions i simulacions, tindrem dos agents extra, el *AMS* i el *DF*

[15] NetBeans: https://netbeans.org/
[16] Jade: http://jade.tilab.com/
[17] Weka: https://www.cs.waikato.ac.nz/ml/weka/

que tota plataforma ha de tindre. El *AMS*, ens gestionarà l'ús de la plataforma i el *DF* ens oferirà l'opció, de que els nostres agents puguin publicar els seus serveis. Aquest dos agents (el *AMS* i el *DF*), els gestiona de forma automàtica *JADE* i no ens hem de preocupar d'ells. Ara bé, degut a que en el nostre projecte utilitzarem molts dels serveis que ofereixen els nostres agents, serà vital configurar correctament l'entrada i sortida dels agents mitjançant el DF. Veurem aquesta característica amb més detall en el punt 6.2.1, on desvetllarem les funcions que ofereixen, al nostre sistema, la possibilitat de registrar o no serveis.

6.1 Estructura general del projecte

Una vegada creat el projecte en *NetBeans*, s'han constituït diferents paquets. Considerem interessant, la divisió de classes JAVA en diferents paquets, ja que permet la identificació ràpida i un millor comprensió de les dades.

La següent taula, mostra un resum general dels paquets que s'han construït.

Paquet	Descripció
actionsOntology	Classes per les accions de l'ontologia
agents	Classes que implementen els agents del SMA
conceptsOntology	Classes que implementen els conceptes de l'ontologia
init	Classe que inicia el SMA
ontology	Classe que implementa l'ontologia
test	Classes que implementen els test de prova
txt	Arxius persistents, Log.txt, test.arff i Control.arff
visual	Classes que implementen una interfície visual

Taula 19: Llista de paquets del projecte

6.2 Agents

Una vegada analitzada l'estructura general del projecte i els requeriments, anem a veure com s'han implementat les parts més importants. S'ha inclòs en el projecte el javadoc corresponent a totes les classes i funcions implementades, tot i això es mostrarà un petit resum, en format de taula, en cada agent. Començarem pels agents del sistema.

6.2.1 ConfigAgent.java

Aquesta classe, situada al paquet "agents" del nostre projecte, estén de la classe Agent de *JADE*. No l'utilitzarem com un agent funcional, és a dir, no la considerarem un agent que tingui una activitat directa en el SMA. Simplement és una classe que serveix com a base de configuració de tots els altres agents. Els altres agents, s'estendran d'aquesta i per herència tindran els mètodes i funcions de *JADE* i els que nosaltres configurem en aquesta classe.
Com que tots els agents del nostre sistema, hauran de registrar serveis en el *DF*, aquesta classe implementarà una funció que permet fer aquesta tasca. També, necessitem que els agents cerquin altres agents, per tant, també implementem en aquesta classe, funcions que permeten cercar un agent determinat o una llista d'agents que ofereixen un mateix servei.
En el nostre SMA utilitzem un llenguatge propi (ontologia) i per referenciar-nos sempre al mateix contenidor (plataforma), necessitem que els agents tinguin sempre a la vista, una referència al llenguatge i al contenidor al qual pertanyen.

Per tant, en aquesta classe també implementem variables que fan referència a un llenguatge, còdec i ontologia pròpia. Així, els agents que heretin d'aquesta classe, ja tindran les referències adequades pel que fa al contenidor i llenguatge utilitzat.
Finalment, per tal de mantenir un arxiu persistent on s'emmagatzemin totes les accions rellevants del SMA, també implementem una funció que permeti aquest fet, creant i omplint un arxiu anomenat "*Log.txt*" quan sigui necessari.

Aquest agent, no carregarà res en el seu mètode *setup(){},* però sí inicialitzarà les variables *Cmanager* (contenidor on es situen tots els agents), *codec* (codificació del llenguatge), *ontology* (ontologia utilitzada) i directori on s'emmagatzemarà les dades del *Log*.

Posa a disposició dels altres agents, les següents funcions:

Funció	Descripció
logSystem()	Emmagatzema informació general del sistema
RegisterToDF()	Registra un agent en el DF
searchAgent()	Cerca un agent en el DF
searchAgents()	Cerca un grup d'agents amb el mateix servei en el DF

Taula 20: Funcions generals ConfigAgent

6.2.2 AgentCentral.java

Anteriorment, ja s'ha comentat que aquests és un dels agents amb més càrrega de treball del nostre SMA. Això és degut a que, és un agent que gestiona les peticions dels clients i respon a les mateixes, negocia amb els vehicles per obtenir ofertes, enregistra dades dels serveis, rep l'estat dels vehicles, gestiona els incidents dels vehicles i calcula prediccions gestionant als clients (en cas del model predictiu). Tot seguit un petit resum de les parts més importants d'aquest agent (taula comportaments, classes i funcions)

Comportament	Descripció
InformCars	Comportament simple per gestionar missatges CONFIRM
AnswerCars	ContractNetInitiator per la negociació amb els vehicles
AnswerRequest	AchieveREResponder per gestionar missatges REQUEST dels clients.

Taula 21: Comportaments AgentCentral

Classes	Descripció
InfoClient	Emmagatzema informació sobre el servei
Estadistic	Emmagatzema informació sobre els resultats predictius

Taula 22: Classes AgentCentral

Funció	Descripció
getPrediction()	Retorna la predicció d'una petició
viewTable()	Mostra la taula resum de prediccions
getHashCarTable()	Mostra la taula d'estats dels vehicles

Taula 23: Funcions generals AgentCentral

6.2.3 AgentCar.java

Aquesta classe és l'encarregada de gestionar els comportaments dels vehicles. Per tal de donar cert dinamisme a tot el sistema, es poden crear diferents agents d'aquest tipus, tots ells amb les mateixes característiques i diferents valors. Per

això, una de les primers tasques d'aquest agent és carregar la configuració personal. Després demanarà informació a l'*AgentTraffic* per rebre un plànol de la ciutat. Una vegada te el plànol, enviarà una confirmació a l'*AgentCentral* per informar del seu estat. Finalment, estarà a l'espera de rebre peticions. Tot seguit els comportaments, classes i funcions més significatives.

Comportament	Descripció
timeDeregister	Simula el temps d'un determinat servei
AnswerCars	Espera missatges Contract_net
RequestMap	Demana el plànol de la ciutat

Taula 24: Comportaments AgentCar

Classe	Descripció
registerUpdate	Fil d'execució per simular el temps d'un servei
Dijkstra	Permet implementar l'algoritme Dijkstra

Taula 25: Classes AgentCar

Funció	Descripció
configCar()	Carrega les dades del vehicle
CarInformation()	Retorna les dades d'un vehicle
transformArray()	Transforma un array jade en un array java
carStat()	Envia un missatge amb les dades de l'estat del vehicle

Taula 26: Funcions generals AgentCar

6.2.4 AgentTraffic.java

En aquest cas es gestiona la situació geogràfica del nostre escenari. La primera tasca d'aquest agent és la de carregar un plànol en la seva configuració i el modifica si existeixen carrers tallats. Una vegada carregat, queda a l'espera de rebre peticions, per respondre amb aquesta informació als vehicles que la necessitin. A continuació, els comportaments i funcions més significatives:

Comportament	Descripció
AnswerMap	Respons als vehicles amb el plànol d'una ciutat

Taula 27: Comportament AgentTraffic

Funció	Descripció
getListMap()	Retorna informació relativa al plànol

Taula 28: Funció AgentTraffic

6.2.5 AgentClient.java

En aquest cas, estem davant de l'agent que inicia l'activitat de tot el procés. Si no hi ha peticions, el sistema queda a l'espera de rebre una. Per tant, és l'agent principal en el servei que es vol dur a terme. Una vegada enregistrat un client, la seva primera tasca, és carregar la seva configuració personal. Seguidament rep la informació del plànol i la llista de carrers tallats. Finalment, elabora una petició amb les dades del servei i l'envia a l'*AgentCentral*. A continuació, queda a l'espera de rebre un missatge d'informació amb dades del servei i la seva confirmació. Tot seguir, les taules de comportaments i funcions més rellevants.

Comportament	Descripció
initRequest	Inicia els missatges REQUEST cap a l'AgentCentral
ComfirmService	Espera missatges CONFIRM des de l'AgentCentral

Taula 29: Comportaments AgentClient

Funció	Descripció
CreatePetition()	Elabora una petició de servei
SendPetition()	Envia una petició
configClient()	Carrega la configuració inicial d'un client

Taula 30: Funcions generals AgentClient

6.2.6 AgentControl.java

Aquest últim agent, s'encarrega d'emmagatzemar informació dels serveis realitzats i fer una petita presentació del conjunt del sistema. Una de les tasques més importants que realitza aquest agent, és la de crear l'arxiu " *Control.arff* ". Aquest arxiu, inclou una capçalera, que és essencial per a que els algoritmes de *Weka* puguin interpretar el contingut. Una vegada creat, quan l'agent rep un missatge per emmagatzemar dades, extrau el contingut del missatge i escriu les dades en l'arxiu. Seguidament, les taules de comportaments, classes i funcions més importants.

Comportament	Descripció
SaveServices	Espera missatges per emmagatzemar informació d'un servei.

Taula 31: Comportament AgentControl

Classe	Descripció
GenerateUUID	Crea un identificador únic
Session	Emmagatzema una llista de serveis
Service	Emmagatzema informació d'un servei determinat

Taula 32: Classes AgentControl

Funció	Descripció
setLogTitle()	Informació de la capçalera del Log
initDocument()	Crea l'arxiu Control.arff si és necessari

Taula 33: Funcions generals AgentControl

6.3 Ontologia

En aquest moment, ja tenim una idea de com i què faran els nostres agents. També es van dissenyar (punt 5.5) els conceptes i accions que s'utilitzarien en el nostre sistema, per tant només queda veure, on es troben les classes de la nostra ontologia i com s'han implementat.

El paquet *Ontology*, amaga en el seu interior l'estructura bàsica de l'ontologia. *OntologyTransport.java* és la classe on es declaren totes les variables que fan referència als conceptes i a les accions necessàries. És una classe que s'estén de la classe Ontology de JADE.

La creació de l'ontologia s'ha dividit per seccions, clarament documentades, per tal que sigui fàcilment llegible. Passem a descriure, de forma resumida cada secció:

En la primera secció es troben les declaracions dels diferents objectes.

- *NAME_ONTOLOGY*
- *TYPE_SERVICE i SERVICE_AGENT*

En la segona secció es troben les declaracions dels objectes. (conceptes i accions).

Objecte	tipus
Concepte	Petition, Position, Answerpetition, Carinformation, Listmap, infonode
Acció	Answerservice, Requestservice, Requestlistmap, Statecar

Taula 34: Conceptes i accions de l'ontologia

En la tercera part, es troben els objectes que fan que l'ontologia mantingui una sola instància d'ella mateixa.

- *Ontology instance* : objecte de la instància
- *Ontology getInstance()* : Mètode que retorna la instància de l'ontologia.

En la quarta part i última, trobem el constructor que carrega tota la informació donada. En aquesta part, s'inicialitzen els conceptes i s'afegeixen els tipus a cada un. Finalment, s'afegeixen els conceptes a les accions i es configuren correctament.
Per tal de tindre correctament identificades les parts de l'ontologia, s'ha decidit crear un paquet per a cada secció. Així, en el paquet " *actionsOntology* " es troben les classes que configuren les accions de l'ontologia i en el paquet " *conceptsOntology* ", es troben les classes que configuren els conceptes.

6.4 Altres parts estructurals

Per tal de dur a terme l'estudi de tot el sistema, s'han plantejat diferents solucions.

La primera, l'elaboració d'un test de proves que asseguri un correcte funcionament bàsic.

La segona solució, permet emmagatzemar arxius persistents pel seu posterior anàlisis. Amb anterioritat, ja s'ha fet referència a que cada agent pot escriure, tota la seva activitat, en l'arxiu " *Log.txt* ". També s'ha parlat de l'arxiu "*Control.arff* " que és una base de coneixement i de l'arxiu " test.arff " que emmagatzema la última instància avaluada. Bé, doncs aquest arxius es poden trobar en el paquet " *txt* ". Val a dir, que l'arxiu *Log.txt* te una importància elevada, ja que en el mode d'execució predictiva, es mostra una taula de resum, on es poden veure les diferències i similituds entre classes predites i reals. A part, també es mostra l'ordre d'execució de les peticions en espera i l'error comés en la predicció.

La tercera i última solució, ha sigut la implementació d'una petita interfície gràfica. Aquesta interfície, és capaç de representar el plànol de la ciutat, mitjançant nodes i arestes que representen punts geogràfics amb carrers. Cada posició està identificat amb un nombre i els carrers (arestes) mostren la distància entre punts. És capaç de pintar de color blau fosc el recorregut que fa el vehicle, des de la posició on es troba el propi vehicle fins a la posició on es troba el client. Realitza la mateixa tasca amb el recorregut des de el client fins al destí, però en aquest cas, el pinta de color taronja. També identifica, mitjançant colors, els punts següents: posició inicial del vehicle (punt de color verd), posició del client (punt de color negre) i posició final o de destí (color blau fluix). En la part inferior de

la pantalla, mostra informació addicional en dos petits blocs, un amb informació relativa al client (bloc de l'esquerra) i l'altra relativa al vehicle. En la part del client es mostra: el nom del client, la seva posició, el destí del paquet, el preu en euros del servei realitzat i la distància total recorreguda. En la part del vehicle, es mostra: l'identificador del vehicle, el recorregut vehicle-client, el recorregut client-destí, la posició inicial i final del vehicle. Les classes que implementen aquesta solució, es poden trobar en el paquet " *visual* ".

S'ha considerat, que aquestes 3 solucions són suficients per començar a analitzar el sistema i extraure conclusions.

En tot cas, junt a la carpeta del projecte, s'adjunta un petit manual d'usuari.

6.5 Estructura dels test de proves

Arribats a aquest punt, la nostra primera tasca serà la de plantejar un escenari bàsic de funcionament. És a dir, iniciar el SMA i que es carreguin correctament els agents en la plataforma. En aquest cas, serà suficient amb l'*AgentCentral*, *AgentControl*, *AgentTrafic*, un *AgentCar* i un *AgentClient*.
Les classes necessàries per la configuració del client, del vehicle i del plànol, les trobarem en el paquet " *test* ". En aquest paquet hi trobarem les 3 classes que implementen les configuracions: *Car.java, Client.java, Traffic.java*.

Una vegada assegurat el funcionament bàsic, es realitzarà un segon test que tindrà com objectiu, la informació que transporten els diferents missatges implicats, en una determina petició.
Es continuarà amb un tercer test, que permetrà demostrar que s'atenen correctament, més d'una petició amb un sol vehicle.
Un quart test, permetrà comprovar que amb més d'un vehicle en el sistema, es selecciona correctament, l'opció encertada davant d'una determinada petició.
Posteriorment, provarem que el sistema es comporta correctament amb més d'un vehicle i més d'una petició.
També es posarà a prova i a estudi, el mode de funcionament predictiu, on ho compararem amb el mode lineal i extraurem certes conclusions. És en aquest cas, on farem un anàlisis més profund.
En el nostre SMA, no deixarem de banda els possibles incidents i executarem un test simulant carrers tallats i un altra on un vehicle tindrà un accident.
L'últim test demostrarà que, el SMA és capaç de funcionar correctament amb totes les opcions (més d'un vehicle, més d'un client, execució predictiva, carrers tallats i incidents).

Quan s'executi un test de proves, es podrà veure informació per pantalla (línia de comandes), també es mostrarà la interfície gràfica del client (pantalla de dades del resultat d'un servei) i també s'enregistrarà tota la mateixa informació, en l'arxiu *Log.txt*.

Per tal d'identificar certes parts dels missatges que es mostren per pantalla (línia de comandes o arxiu Log.txt), s'han incorporat etiquetes descriptives a l'inici de cada línia. En la següent taula, es mostren les diferents etiquetes que el sistema pot llançar:

Etiqueta	Descripció
INFO	Mostra informació general d'una determinada situació. Normalment, s'utilitza per descriure una determinada acció d'un determinat agent o conversa entre agents.
FINE	S'utilitza quan es vol ampliar informació derivada d'una acció.
STATE*	S'utilitza per mostrar el contingut d'alguna taula, estat de vehicles o prediccions.
PRED	Aquesta etiqueta s'utilitza per descriure la predicció que està realitzant l'*AgentCentral,* en una determinada petició.
ERROR	Aquesta etiqueta mostra la informació sobre un determinat error.
ALERT	Aquesta etiqueta es mostra un problema de servei.

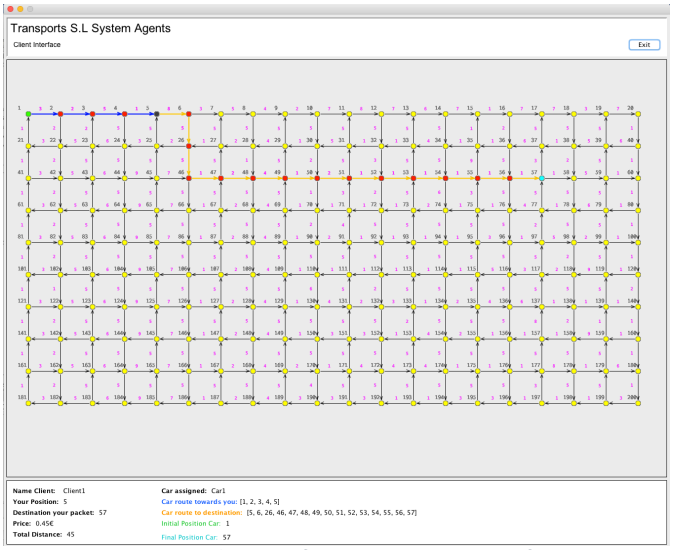
Taula 35: Etiquetes descriptives en execució

En el cas de la taula de prediccions, etiqueta STATE, es mostra la següent informació:

Atribut	Descripció
Classify_algorithm	És l'algoritme utilitzat per cercar les prediccions.
ClientX	És el client al qual pertany la petició avaluada.
Pred_class	És la predicció del cost per aquesta petició.
Real_class	És el cost real de la petició, una vegada executada.
Equal	Indica si han coincidit o no la predicció amb el cost real.
Time	És el torn en que es va executar la petició.
Incorrectly_instance	És el nombre de peticions que s'han predit malament.
Correctly_instance	És el nombre de peticions que s'han predit correctament.
Error	És l'error global comès en les prediccions.

Taula 36: Descripció etiqueta STATE

Una vegada s'ha confirmat un determinat servei, el client rep la informació. Aquesta informació, encara que es pot extraure de l'arxiu *Log.txt*, s'ha pensat que seria molt més agradable veure-la visualment. Així, per a cada petició es mostra una finestra com la següent:



Il·lustració 15: Interfície visual de dades finals

El graf mostra el plànol d'una determinada ciutat, on els punts grocs són posicions on es poden situar, clients, vehicles i paquets. Les arestes són carrers d'una única direcció amb una certa distància (nombre de color rosa). El punt de color verd informa de la posició on es troba el vehicle. El punt de color negre, la posició on es troba el client. El punt de color blau fluix, el destí del paquet. Els

carrers de color blau, mostren el recorregut del vehicle fins arribar al client. Els carrers de color taronja, mostren el recorregut del vehicle, des de el client fins al destí. En la part inferior, es veu informació del client i del vehicle.

7. Simulació

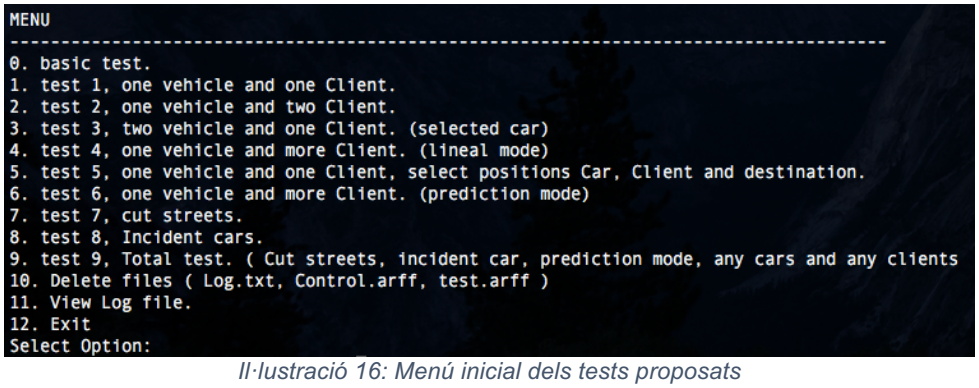
En aquest punt donarem vida al nostre projecte. El nostre objectiu serà, mitjançant els escenaris proposats en el punt 6.5, avalar el bon funcionament de les diferents parts del SMA. A més, una vegada iniciades les simulacions, aprofitarem per estudiar les dades generades.
A cada escenari de simulació, s'explicarà el què i com es vol aconseguir. Es llençarà la simulació i s'examinarà el resultat segons el següent patró:

- Teòrica d'inici (dades inicials).
- Teòrica de sortida (dades que teòricament haurien de sortir).
- Dades reals de sortida (arxius persistents i dades visuals).

Atenció: A l'inici de cada test de prova, es pregunta si es vol emmagatzemar el seu resultat en l'arxiu *Control.arff*. No es farà referència a aquest fet en cada test, es dona per suposat que el lector entén que emmagatzemar dades provocarà un canvi de resultats en algun test. En el test del model predictiu i el test total, també es demana el tipus d'algoritme de classificació que es vol utilitzar. En el nostre cas, sempre utilitzarem el RandomTree i utilitzarem els altres per un estudi comparatiu en punts posteriors.

7.1 Escenari bàsic, test 0

L'escenari bàsic pretén comprovar que els agents fan el que han de fer, davant dels estímuls esperats. Per a realitzar aquesta tasca farem servir la interfície visual de *JADE*. Com ja s'ha parlat, *JADE* disposa d'una interfície gràfica que permet veure, de forma gràfica, la comunicació entre agents.
En el nostre projecte, s'inclou un petit menú, que només te com objectiu, recrear els nostres escenaris de proves.

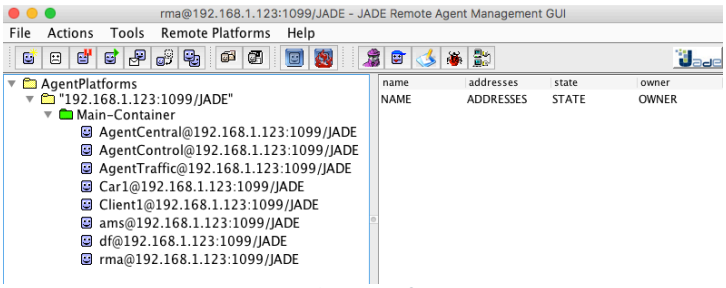


Il·lustració 16: Menú inicial dels tests proposats

Si llancem l'aplicació i seleccionem l'opció " *0. basic test.* ", s'iniciarà la interfície gràfica de *JADE* amb els següents agents: *AgentCentral, AgentTraffic, AgentControl, Car1 i Client1*.

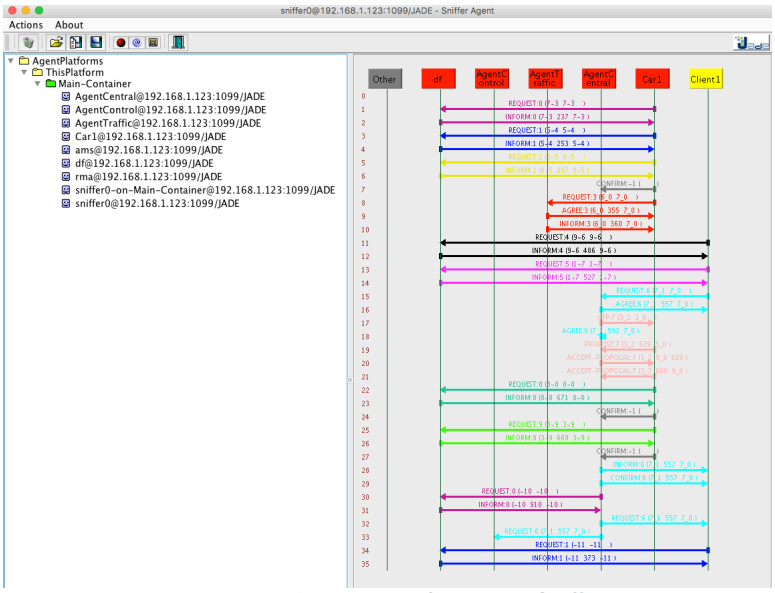
En aquest primer test i per tal de realitzar aquesta prova, s'ha inclòs un retard en l'execució, per tal de veure la comunicació entre els diferents agents. Una vegada iniciada la prova, es veurà la interfície de *JADE* amb tots els agents carregats. Però, romandrà una estona aturada per a que tinguem temps de desplegar l'eina *Sniffe*r de *JADE* i col·locar els agents correctament.

Sniffer es pot executar anant a: *Tools → Start Sniffer de JADE*. Una vegada obert, veurem en el costat esquerra, el mateix arbre d'agents que en el cas de *JADE*. Seleccionarem els agents que ens interessin i l'afegirem al costat dret mitjançant: *Actions → Do sniff this agent(s).* Una vegada finalitzi el temps d'espera, automàticament s'executarà la totalitat del test i es veuran les comunicacions entre agents.



Il·lustració 17: Plataforma JADE

Una vegada iniciat el test, es pot veure *JADE*. Expandim l'arbre d'agents, fins arribar al nostre contenidor i veurem els nostres agents en execució. En aquest moments es troben aturats i aprofitarem aquest fet per obrir *Sniffer* i col·locar-los en el visor de missatges.



Il·lustració 18: Resultat final test0 Sniffer

Una vegada s'executi el test al complert, veurem com els agents intercanvien els missatges.
Comencem la descripció, observant la gràfica en forma descendent. *AgentCar* dialoga amb el DF, això es deriva del seu registre i de la cerca de l'*AgentTraffic*. En la línia 7 es veu com *AgentCar* confirma a l'*AgentCentral* que està disponible. Seguidament (línies 8,9 i 10) demana el plànol a l'*AgentTraffic*. A continuació,

es pot veure com l'*AgentClient* entra en el sistema, es registra en el DF i cerca l'*AgentCentra*l. Després, realitza la seva petició de servei (línies 15 i 16). A continuació, l'*AgentCentral* llança el seu missatge per negociar amb *AgentCar*. Es pot veure que *AgentCar* retorna la seva proposta en la línia 19 i que finalment és acceptada, línia 20.
Una vegada acceptada, *AgentCentral* envia un missatge al client amb informació del servei, línia 28.
També es pot veure com *AgentCar* cerca a l'AgentCentral mitjançant el DF i a més li envia un missatge de confirmació per informar del seu estat, línies 24 i 27 (començar i finalitzar el servei). Seguidament, *AgentCentral* interpreta que s'ha finalitzat el servei, informa i confirma aquest fet a *AgentClient*, línia 28 i 29 respectivament. Finalment, *AgentClient* surt de la plataforma.
Queda demostrat, que el SMA actua com inicialment es va dissenyar i que tots els agents interactuen correctament en la comunicació bàsica. El següent pas, serà comprovar si la informació que es rep en els diferents escenaris, és correcte.

7.2 Test 1. Un vehicle i un client

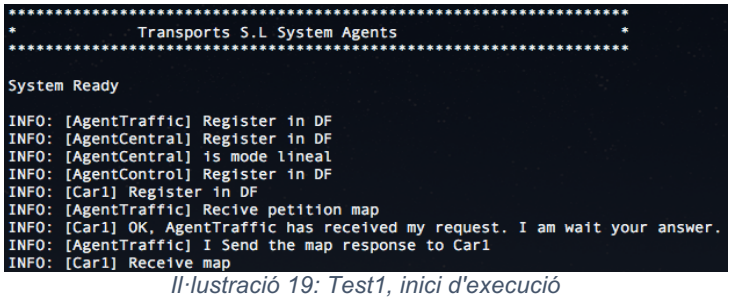
En aquest segon test, que es pot executar mitjançant la segona opció del menú d'inici de l'aplicació (*1. Test 1, one vehicle and one Client*.), tenim com objectiu, comprovar que s'ha realitzat correctament un servei. Es a dir, que es demana correctament una petició, que s'analitza la seva execució, que es negociï de forma correcte amb els vehicles disponibles i que es rebi la informació requerida en cada pas.
En el test anterior, ja es va realitzar aquesta prova, però a un nivell de missatges. En aquesta ocasió, es vol comprovar que els missatges transporten la informació correcte.
Per tal de veure si realment s'executa correctament, iniciem alguns dels agents amb una configuració inicial predisposada. Passem a descriure la configuració de cada agent.

AgentCentral	AgentTraffic	AgentControl	AgentClient	AgentCar
Funcionament en mode d'execució lineal	Càrrega del plànol de la il·lustració 4.	Emmagatzemar dades.	Nom: Client1 Posició: 200 Destí paquet :60 Num.Paquets: 1	Nom: Car1 Time: 2000 Posició: 1 Estat: true Incident: False

Taula 37: Configuració agents Test 1

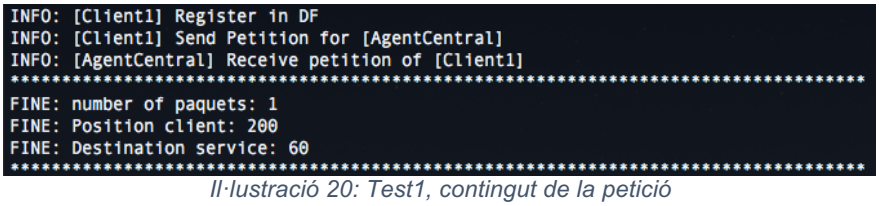
Avanç d'iniciar el test, anem a veure de forma teòrica, quin haurà de ser el resultat.
El *Client1* llançarà la seva petició, que arribarà a *AgentCentral*. Aquest, analitzarà si hi ha vehicles disponibles revisant la taula d'estats. En la taula d'estats només hi haurà *Car1*, perquè està actiu i no te incidents. Així que *AgentCentral* li demanarà una proposta. *Car1*, calcularà el recorregut tenint en compte el següent: es troba en la posició 1 i haurà de viatjar fins la posició 200. Mirant el plànol, es pot veure que el camí mes curt és [1, 2, 3, 4, 5, 6, 26, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 80, 100, 120, 140, 160, 180, 200]. Una vegada en la posició 200, el vehicle ha de viatjar fins a la posició 60. El camí més curt, és passant per les posicions [200, 199, 179, 159, 139, 119, 99, 79, 59, 60] amb un cost total, entre els dos recorreguts de 95.

Una vegada calculat el cost, li enviarà la proposta a *AgentCentral*. Aquest, avaluarà totes les propostes rebudes (només una, en aquest cas), per tant escollirà *Car1* per realitzar el servei. Per la seva banda, *Car1* quedarà fora de servei durant un cert temps (simulant el temps d'execució del servei). Una vegada finalitzada la tasca, *AgentCentral* informarà i confirmarà les dades a Client1, que sortirà de la plataforma.
Comencem executant el test 1 i justifiquem la nostra tesis. La primera informació que ens presenta el test, és el registre dels diferents agents en el sistema i la creació del document *Control.arff*. Seguidament es pot veure com *AgentTraffic* rep la petició de *Car1*, per a que li faci arribar el plànol de la ciutat. Al final de la captura, es pot apreciar com *Car1* rep el plànol.



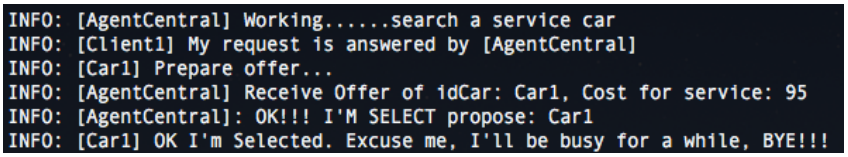
Il·lustració 19: Test1, inici d'execució

Continuem amb l'entrada de *Client1* en el sistema i la seva posterior petició. Es pot veure, mitjançant l'etiqueta FINE, que es mostra el contingut de la petició (on es pot comprovar que coincideix amb l'estat inicial predisposat).



Il·lustració 20: Test1, contingut de la petició

Ara li toca el torn a *AgentCentral*, que inicia la seva cerca de vehicles. Internament, consulta la taula d'estats de vehicles i només cerca un, *Car1*. Envia la petició a l'únic vehicle que a cercat i aquest, prepara l'oferta. Internament, *Car1* està calculant recorreguts mínims i una vegada calculats, envia el cost a *AgentCentral.* Una vegada, *AgentCentral* rep la informació del cost, selecciona el mes petit, en aquest cas la que proporciona *Car1*.



Il·lustració 21: Test1, negociació i selecció de vehicle

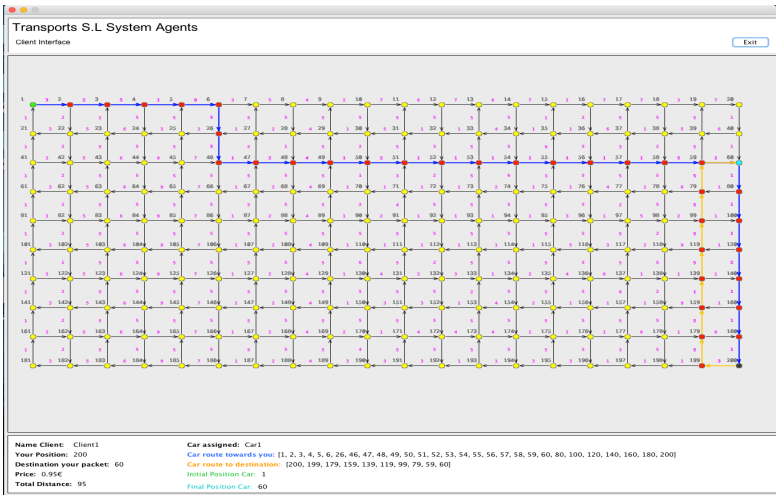
Finalment s'envia la informació al client, amb les dades finals i la confirmació que se ha realitzat el servei.

```
INFO: [Client1] Receive inform of [AgentCentral] , Your service is:
**********************************************************************************************************************************
FINE: Assing Car: Car1
FINE: Cost for your service: 95
FINE: Price for your service: 0.95€
FINE: The car will make this route to you: [1, 2, 3, 4, 5, 6, 26, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 80, 100, 120, 140, 160, 180, 200]
FINE: The car will make this route from you to the destination: [200, 199, 179, 159, 139, 119, 99, 79, 59, 60]
**********************************************************************************************************************************

##################################################################################################################################
INFO: Finaly Service for [Client1]
INFO: [AgentCentral] Thanks for your service [Client1]
INFO: [Client1] Deregister in DF
##################################################################################################################################
```

Il·lustració 22: Test1, informació i confirmació del servei

Com es pot veure, les etiquetes *FINE* tornen a mostrar el contingut del missatge rebut i es pot comprovar que coincideix amb el que es va preveure en un inici. També es mostra el preu, que és el producte del cost per 0.01€.

La confirmació del servei és un pas important, perquè permet d'assegurar que el vehicle assignat ha realitzat el servei correctament. Si per qualsevol situació, hi ha un incident una vegada escollit i assignat un vehicle, aquesta confirmació no es produirà fins que un altre vehicle prengui el relleu.
Seguidament, mostrem la interfície gràfica de *Client1*, amb la informació que s'ha rebut, una vegada finalitzat el servei. Conseqüentment, queda demostrat que el sistema ha realitzat les tasques de forma correcte i que els agents s'han coordinat correctament, per assolir un objectiu comú.



Il·lustració 23: Test1, resultat final de forma visual

7.3 Test 2. Un vehicle i dos clients

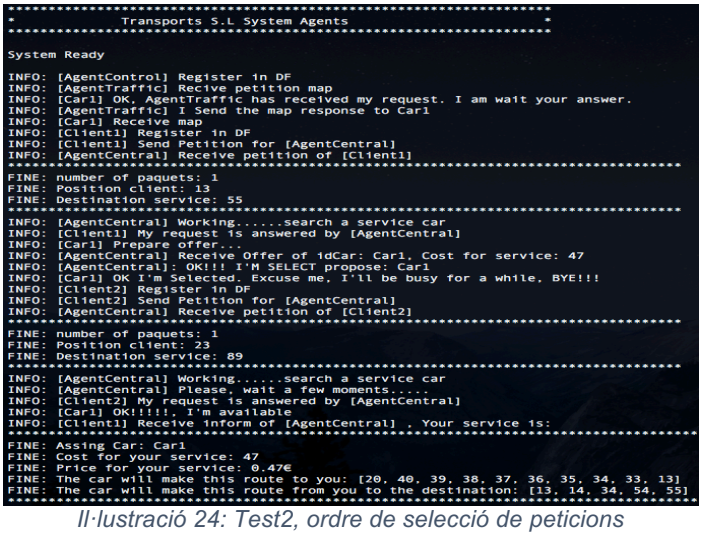
En aquest segons test, es pretén esbrinar si el sistema es capaç d'atendre més d'una petició, amb un sol vehicle. Aquest test es pot executar, mitjançant l'opció (2. *test 2, one Vehicle and Two Client*) de l'aplicació.

Quan s'executa una petició, el vehicle es mou per la ciutat d'una posició a una altra. Una vegada finalitza el servei, el vehicle haurà d'estar situat en la posició on va finalitzar el seu darrer servei. Passem a descriure la configuració inicial dels agents:

AgentCentral	AgentTraffic	AgentControl	AgentClient	AgentClient	AgentCar
Funcionament en mode d'execució lineal	Càrrega del plànol de la il·lustració 4.	Emmagatzemar dades.	Nom: Client1 Posició: 13 Destí paquet :55 Num.Paquets: 1	Nom:Client2 Posició: 23 Destí paquet: 89 Num.Paquets: 1	Nom: Car1 Time: 2000 Posició: 20 Estat: true Inicident: False

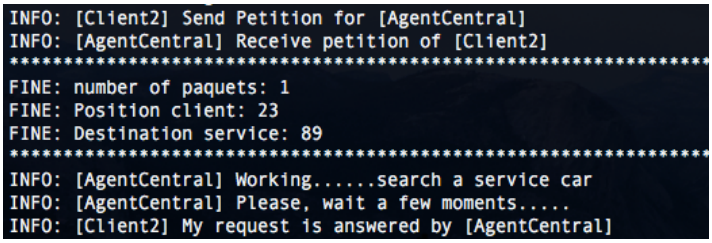
Taula 38: Configuració agents Test 2

En aquest escenari, el vehicle *Car1* atendrà la primera petició (*Client1*) i quedarà fora de servei un temps. Com que hi haurà una altre petició (*Client2*) el sistema inclourà aquesta petició en la llista d'espera (perquè no hi ha vehicles disponibles). En el moment que *Car1* torni a estar disponible, atendrà la següent petició i com que estem en el mode d'execució lineal, el primer en sortir de la llista d'espera serà, la petició de *Client2* (i en aquest cas l'única).
En aquest moments el vehicle es trobarà en la posició 55 (final de l'anterior servei) i haurà de recórrer el camí més curt fins la posició 23 (punt on es troba el *Client2*). Així, el primer recorregut serà: [55, 35, 34, 33, 32, 31, 30, 29, 28, 27, 26, 25, 24, 23]. Després, haurà de desplaçar-se fins a la posició 89, destí del paquet del *Client2*. Per tant, el segon recorregut serà: [23, 3, 4, 5, 6, 26, 46, 47, 48, 68, 88, 89].
En aquest test ens interessa veure que, a més de realitzar correctament els recorreguts demanats pels clients, el vehicle es troba en els punts finals, cada vegada que realitza un servei. Ens centrarem en aquesta inquietud i a més, en l'ordre d'execució de les peticions. Recordem que estem en mode d'execució lineal i s'han de respectar els torns d'entrada de peticions. Observem la següent captura de pantalla:



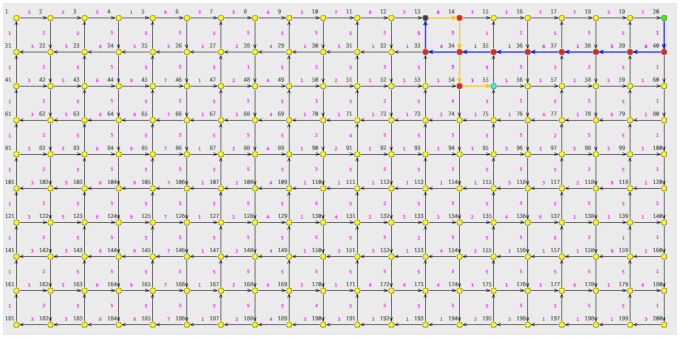
Il·lustració 24: Test2, ordre de selecció de peticions

Primerament s'executa la petició del *Client1*. Fins aquí cap novetat i com es pot veure, les dades que es reben del servei són correctes. A més, *Car1* queda fora de servei durant una estona. Quan torna del servei, es confirmen les dades.
En el moment que entra *Client2*, es pot veure que s'accepta la petició, però que es mostra un missatge d'espera. És en aquest moment, quan la petició de *Client2* passa a la llista d'espera i continuarà allà, fins que un vehicle quedi disponible.

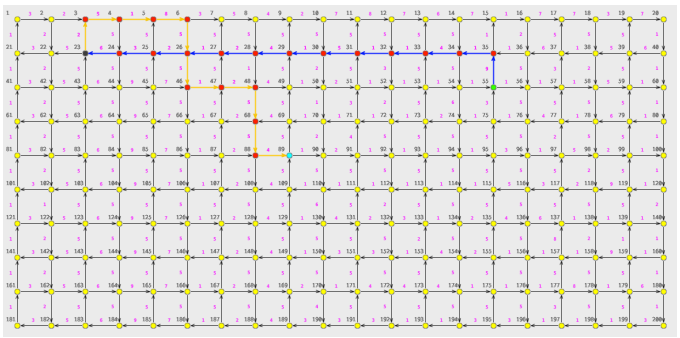


Il·lustració 25: Test2, inserció d'una petició en la llista d'espera

Una vegada Car1 torna al servei, *AgentCentral* sap que torna a estar disponible i extrau la petició de *Client2*. Com està treballant en mode d'execució lineal, extrau la primera petició que va entrar, en aquest cas la de *Client2*. A continuació, es realitza el mateix procediment que en la petició de *Client1* i una vegada confirmada la tasca, finalitza el servei.
Es pot veure el recorregut realitzat i es comprova que són correctes, coincidint amb la nostra tesis inicial.



Il·lustració 26: Test2, resultat final Client1



Il·lustració 27: Test2, resultat final Client2

7.4 Test 3. Un vehicle i dos clients, selecció de petició

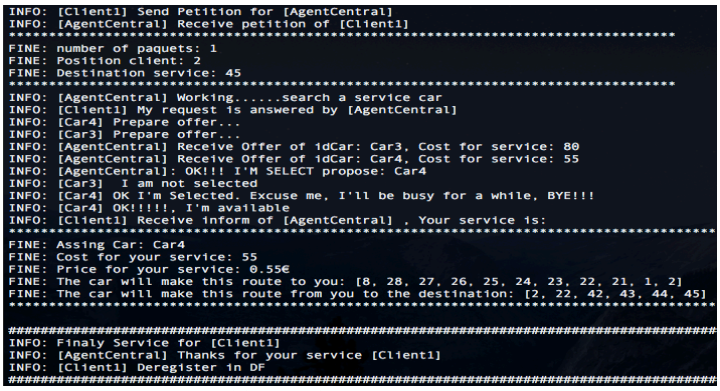
En aquest següents test, es planteja l'objectiu de corroborar que el nostre SMA és capaç de realitzar correctament la selecció d'un vehicle per una determinada petició. En el cas que s'hagi d'atendre una petició i hi hagi més d'un vehicle disponible, el sistema ha d'escollir el vehicle que proporcioni un servei amb el cost més baix. Aquest test es pot executar mitjançant l'opció (*3. test 3, two vehicle and one client*) de l'aplicació i està constituït pels següents agents:

AgentCentral	AgentTraffic	AgentControl	AgentClient	AgentCar	AgentCar
Funcionament en mode d'execució lineal	Càrrega del plànol de la il·lustració 4.	Emmagatzemar dades.	Nom: Client1 Posició: 2 Destí paquet :45 Num.Paquets: 1	Nom:Car3 Time: 2000 Posició: 18 Estat: true Incident: False	Nom: Car4 Time: 2000 Posició: 8 Estat: true Incident: False

Taula 39: Configuració agents Test 3

El client es troba en la posició 2 i quan s'accepti la petició, cada vehicle calcularà el cost total del servei. *AgentCentral* ha de seleccionar, de totes les ofertes rebudes, la que tingui un cost més baix.
Si analitzem les posicions dels vehicles i tenint en compte, els recorreguts mínims fins arribar al client, veurem que *Car4* te un cost més baix (si ho calculem

es pot veure que el cost total és 55). Per tant, haurà de ser *Car4* el seleccionat i qui realitzi el servei.
En aquest cas, no s'inclou la petició en la llista d'espera, ja que existeixen vehicles disponibles per realitzar el servei. En la següent imatge, es pot veure la selecció del vehicle amb el cost més baix.



Il·lustració 28: Test3, selecció de cost més baix

Queda demostrat que el sistema ha realitzat correctament l'elecció del vehicle amb el cost més baix.

7.5 Test 4. Correcte funcionament del mode lineal

En aquest test, ens marquem l'objectiu de comprovar el bon funcionament del mode lineal. Per veure d'una forma clara el resultat d'aquest test, proposem un vehicle i 6 clients. El mode d'execució lineal, permet incloure peticions en una llista d'espera, quan no hi ha vehicles disponibles i extraure-les en el mateix ordre d'entrada. D'aquesta forma, es preserva l'ordre d'execució de les peticions.
Per tal de disposar de suficient temps per omplir la llista d'espera amb totes les peticions, seleccionem un temps de servei alt pel vehicle (en el nostre cas 2 segons). Si no fos així, el vehicle tornaria del seu primer servei, avanç d'omplir la llista i executaria una petició avanç d'omplir-la totalment.
Aquest test és pot executar mitjançant l'opció (*4. Test 4, One Vehicle and more Client. (lineal mode*). Tot seguit, els agents que hi participaran en aquesta prova.

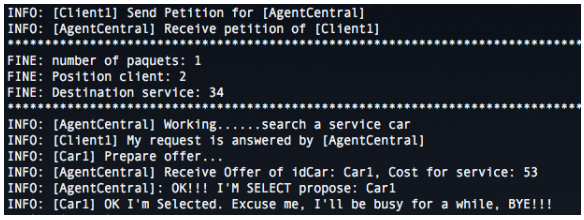
AgentCentral	AgentTraffic	AgentControl	AgentCar	AgentClient
Funcionament en mode d'execució lineal	Càrrega del plànol de la il·lustració 4.	Emmagatzemar dades.	Nom:Car1 Time: 3000 Posició: 1 Estat: true Incident: False	Nom: Client1 Posició: 2 Destí paquet: 34 Num.Paquets: 1

AgentClient	AgentClient	AgentClient	AgentClient	AgentClient
Nom: Client2 Posició: 45 Destí paquet: 89 Num.Paquets: 1	Nom: Client3 Posició: 190 Destí paquet: 45 Num.Paquets: 1	Nom: Client4 Posició: 132 Destí paquet: 56 Num.Paquets: 1	Nom: Client5 Posició: 2 Destí paquet: 200 Num.Paquets: 1	Nom: Client6 Posició: 67 Destí paquet: 109 Num.Paquets: 1

Taula 40: Configuració agents Test 4

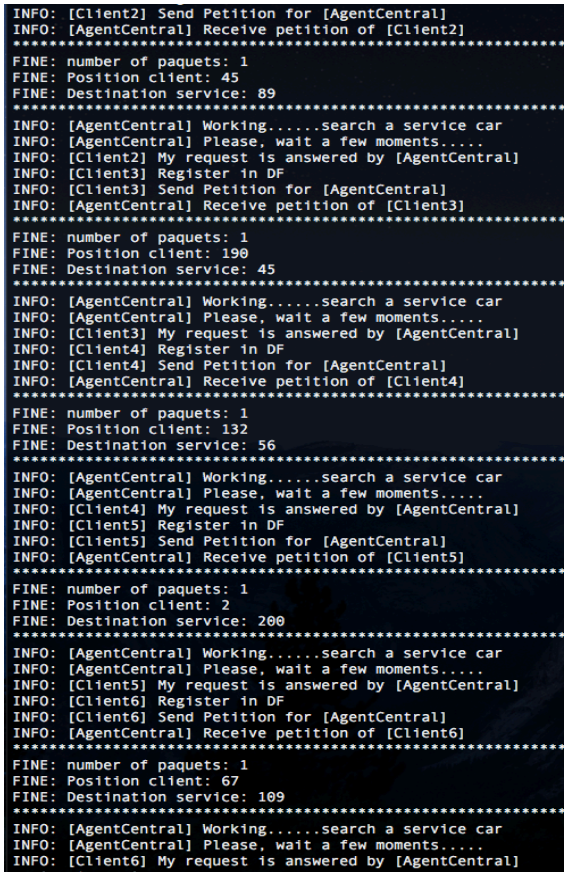
El funcionament és senzill. La primera petició, serà atesa per l'únic vehicle disponible i aquest quedarà en estat no disponible, durant un període de 2 segons. En aquest interval de temps, *AgentCentral* atendrà més peticions que s'aniran introduint en la llista d'espera. En el moment en que *Car1* entri de nou en servei, *AgentCentral* extraurà la primera petició que va entrar en la llista i

l'assignarà (perquè és l' únic vehicle que hi ha). Aquest escenari es repetirà fins al final de la llista.
Una vegada iniciada la prova, es pot veure que s'executa la petició de *Client1* i que *Car1* queda en l'estat de no disponible.



Il·lustració 29: Test4, execució de la primera petició en mode lineal

A continuació, s'introdueixen les altres peticions en la llista d'espera.



Il·lustració 30: Test4, omplir la llista d'espera en mode lineal

Cal remarcar que en aquest cas, no s'envien les dades a l'*AgentControl*, ja que el servei està pendent de realitzar-se. Una vegada torna *Car1*, comença l'execució de les peticions que hi ha en la llista d'espera (per motius d'espai no es mostrarà tota la llista de peticions), però s'executen en el mateix ordre d'entrada.

En definitiva, queda demostrat que el mode lineal funciona correctament, ja que les peticions s'han executat en l'ordre en que van introduir-se en la llista d'espera. A més, també s'ha comprovat que la informació final dels determinats serveis és correcte. Si es vol veure amb més detall aquest test, es pot consultar l'arxiu *Log.txt,* una vegada executat.

7.6 Test 5. Selecció de posicions, vehicle i client

En aquest test tenim com objectiu comprovar que l'assignació de qualsevol posició, ja sigui vehicle, client o destí d'un paquet, és correctament executada pel nostre SMA. Per tant, instarem a l'usuari a inserta qualsevol posició (entre els límits disponibles) i seguidament es realitzarà un servei amb les dades introduïdes. A priori, pot semblar que aquest test no te una gran utilitat, però de cara a l'aplicació del model predictiu, ens ajudarà a augmentar la base de coneixement.
Aquest test es pot executar mitjançant l'opció (*5. test 5, one vehicle and one Client, select position Car, Client and destination*.) i els agents que hi participaran són els següents:

AgentCentral	AgentTraffic	AgentControl	AgentClient	AgentCar
Funcionament en mode d'execució lineal	Càrrega del plànol de la il·lustració 4.	Emmagatzemar dades	Nom: Client1 Posició: X Destí paquet :X	Nom: Car1 Posició: X Estat: true

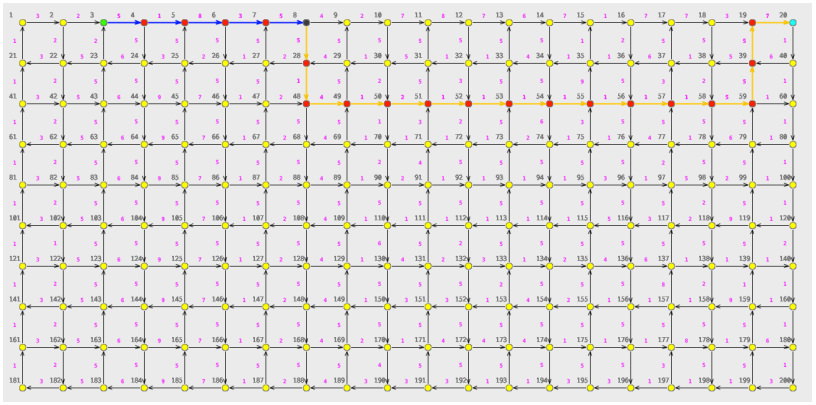
Taula 41: Configuració agents Test 5

Una vegada iniciat el test, el sistema ens preguntarà per la posició del vehicle, del client i del destí del paquet. Seguidament recrearà el servei.

```
Enter position Car (number 1 to 200): 3
Enter position Client (number 1 to 200): 8
Enter destination packet (number 1 to 200): 20
```

Il·lustració 31: Test5, entrada de posició vehicle, client i destí

En aquest cas i segons l'usuari, el vehicle es troba en la posició 3, el client en la posició 8 i el destí del paquet està situat en la posició 20.



Il·lustració 32: Test5, resultat final de forma visual

La imatge anterior demostra de forma clara, que l'execució ha sigut correcte. La virtut d'aquest petit test, va més enllà de l'execució d'un determinat servei. Com ja s'ha comentat, a cada servei realitzat de forma correcte, la base de coneixement augmentarà la seva mida i per tant, són dades que ajudaran a les prediccions que es vulguin realitzar, mitjançant el funcionament predictiu.

7.7 Test 6. Predicció d'ordre d'execució de peticions

No només ens vam comprometre a dissenyar i implementar un SMA, també volem arribar a optimitzar-lo. Realitzant un estudi més meticulós del nostre

escenari, creiem que una de les parts que més pot incidir en el funcionament del SMA, és en el procés de selecció de peticions.

En aquest test, posarem a prova un model predictiu. Com ja s'ha descrit anteriorment, tenim dos parts importants: els arxius de suport (*Control.arff* com arxiu de coneixement i *test.arff* com arxiu on s'emmagatzema l'última petició avaluada) i l'algoritme utilitzat (en aquest cas *RandomTree*).

Partirem de la base que no existeixen cap dels arxius de control o es troben buits. Per assegurar-nos, avanç de començar executarem l'opció 10 (*Delete files (log.txt, Control.arff, test.arff)*) . Aquesta opció eliminarà els arxius de control, si existeixen. Es pren aquesta decisió, perquè es vol avaluar el model predictiu, quan no existeix una base de coneixement. Es a dir, quan es vol realitzar una predicció, l'algoritme utilitzarà com a base de classificació el contingut de l'arxiu *Control.arff* (conjunt d'entrenament). La petició que es vol avaluar, agafarà el rol de conjunt de test i l'algoritme intentarà preveure, la classificació d'aquesta nova instància, dintre de les classes predisposades (recordem que existeixen 3 classes, High – Middle – Low).

És evident, que si no existeix una base de coneixement, a priori, l'algoritme no pot identificar correctament la seva classificació, però amb aquest test es vol veure si degut a un seguit d'execucions, l'algoritme pot aconseguir una millora.

Com que el model predictiu ha de funcionar en base a unes determinades peticions, és necessari que existeixi una llista d'espera. Per tant, en aquest test utilitzarem la mateixa tècnica que en el test 4 (mode lineal). Utilitzarem un vehicle, 6 Clients i forçarem l'emmagatzematge de les peticions a la llista d'espera. Aquest test és pot executar mitjançant l'opció (*6. Test 6, one vehicle and more Client. (prediction mode)*). I els agents que hi participaran són els següents:

AgentCentral	AgentTraffic	AgentControl	AgentCar	AgentClient
Funcionament en mode d'execució predictiu	Càrrega del plànol de la il·lustració 4.	Emmagatzemar dades.	Nom:Car1 Time: 2000 Posició: 1 Estat: true Incident: False	Nom: Client1 Posició: 3 Destí paquet: 45 Num.Paquets: 1

AgentClient	AgentClient	AgentClient	AgentClient	AgentClient
Nom: Client2 Posició: 34 Destí paquet: 45 Num.Paquets: 1	Nom: Client3 Posició: 1 Destí paquet: 60 Num.Paquets: 1	Nom: Client4 Posició: 2 Destí paquet: 5 Num.Paquets: 1	Nom: Client5 Posició: 40 Destí paquet: 32 Num.Paquets: 1	Nom: Client6 Posició: 23 Destí paquet: 16 Num.Paquets: 1

Taula 42: Configuració agents Test 6

Inicialment, el sistema executarà la petició que proposa *Client1*. Això és degut a que només hi ha un vehicle i s'assigna directament a la primera petició que arriba. Una vegada *Car1* quedi fora de servei, *AgentCentral* emmagatzemarà en la llista d'espera, les següents peticions. Fins aquest punt, cap diferència amb el model lineal.

Una vegada, *Car1* torni a estar disponible, comença el procés de predicció. *AgentCentral* comença a llegir la llista de peticions en espera i a cada petició, li assigna una predicció (un valor que serà *High, Middle* o *Low,* determinat per

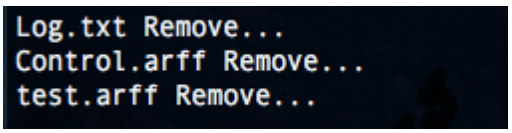
l'algoritme de classificació). Aquest algoritme construeix l'arbre de decisió en temps real, envers a les dades de l'arxiu *Control.arff*. una vegada construït l'arbre de decisió, avalua la nova entrada (petició del client) i llança la seva predicció de classificació. Aquesta predicció, és llegida i emmagatzemada per *AgentCentral* que decideix la prioritat.

El sistema està dissenyat per donar, en primera instància, prioritat a les peticions que tinguin una predicció del tipus *High* (un valor de cost no superior a 30). Si no hi ha cap, donarà prioritat a les considerades com a *Middle* (un valor entre 30 i 80) i finalment a les predites com a *Low* (un valor superior a 80).

Si totes les peticions tenen la mateixa predicció, es donarà prioritat a aquella que va ser inserida, en primera instància, en la llista d'espera. D'aquesta forma, s'intenta continuar amb un model lineal dintre del propi model predictiu, preservant l'ordre d'entrada inicial.

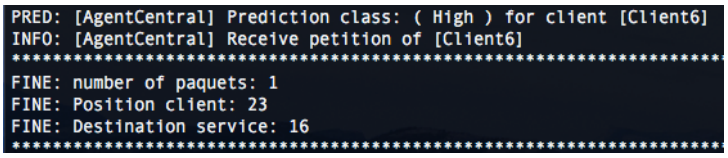
És molt important analitzar l'estat inicial d'aquest test, per avaluar correctament el resultat. S'ha de pensar que estem davant d'un procés de predicció i que aquesta predicció dependrà de l'estat inicial, de les dades generades amb anterioritat i del conjunt de dades que s'estiguin avaluant.

Deixarem en segon pla, la informació de les peticions i ens centrarem en l'ordre d'execució i els resultats de les prediccions. En aquest punt, es considera que les dades són correctes, ja que s'han superat els test anteriors.
Com ja s'ha comentat, primer esborrem, si cal, els arxius de control, amb l'opció 10 del test.



Il·lustració 33: Test6, esborrar arxius Log.txt, Control.arff i test.arff

I a continuació, executem el test 6. Veurem que la primera part, es a dir, la primera execució, coincideix amb el que ja es va veure en el test 4. S'executa la petició de *Client1* i s'emmagatzemen les peticions dels altres clients. Obviarem aquesta part per no ser redundants, ja que és exacte.
Continuem en el moment que *Car1* torna al servei, després d'executar la petició de *Client1*.



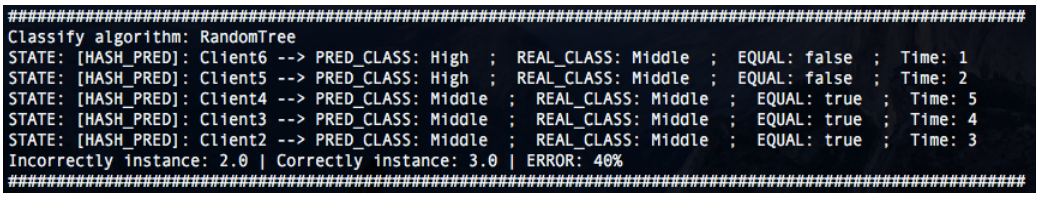
Il·lustració 34: Test6, llançament de predicció d'una petició

Com es pot veure, en el moment que *Car1* està disponible, *AgentCentral* calcula les prediccions de les peticions que es troben en la llista d'espera. En aquest cas, en el moment d'iterar sobre la llista d'espera, ha trobat una predicció High i per tant, l'executa immediatament.
AgentCentral encara que ja sap que el vehicle assignat serà *Car1*, li demana la proposta de cost. Aquest pas és necessari, perquè d'aquesta forma podrà

emmagatzemar el cost real del servei. Recordem que ha llançat una predicció i que aquesta, pot ser que no sigui correcte. El fet d'emmagatzemar el cost real, ens ajudarà a llançar millors prediccions en un futur.

Executada la petició de *Client6* i una vegada *Car1* està disponible, el sistema torna a calcular prediccions, segons les dades actuals. S'ha de pensar que *Car1* s'ha mogut i que ara, es troba en una posició diferent a la predicció anterior.

Si continuem observant la sortida de dades a cada execució, veurem les respectives prediccions i els seus costos finals a l'hora de confirmar el servei. No mostrarem totes les dades, però si mostrarem la taula final de predicció. Com ja es va descriure, després de cada test de mode predictiu, es mostra una taula on hi ha tota la informació rellevant.



Il·lustració 35: Test6, taula de prediccions en la primera execució

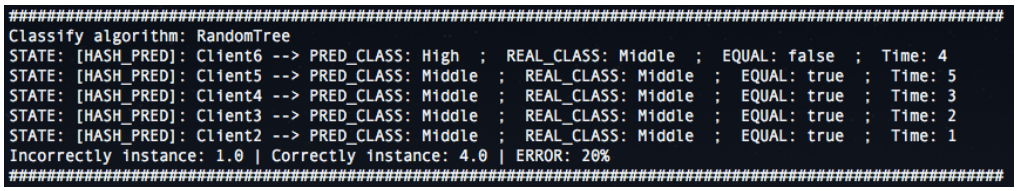
L'anterior imatge mostra la taula de prediccions, de l'execució d'aquest primer test predictiu. Com es pot veure, l'error global ha sigut del 40%, això vol dir que de 5 prediccions s'han encertat 3.
També es veu la classe predita i la classe real de cada petició. Una informació molt interessant és l'ordre d'execució (etiqueta *Time*). Es pot veure clarament, que les peticions s'han atès en aquest ordre: *Client1 – Client6 – Client5 – Client2 – Client3 – Client4*. (*Client1* no surt a la taula perquè no va ser introduït en la llista d'espera).
Tot seguit, es mostra el contingut de la base de coneixement, arxiu Control.arff.

```
@relation transport
@attribute initial_position_car NUMERIC
@attribute Final_Position_Car NUMERIC
@attribute Position_Client NUMERIC
@attribute nodes NUMERIC
@attribute Class {High,Middle,Low}
@data
1,45,3,6,High
45,16,23,21,Middle
16,32,40,13,Middle
32,45,34,18,Middle
45,60,1,27,Middle
60,5,2,27,Middle
```

Il·lustració 36: Test6, contingut de l'arxiu Control.arff en primera execució

Com es pot veure, la primera entrada (1, 45, 3, 6, *High*) és la que pertany a la petició de *Client1* (és el primer que es va executar). Les altres són les que es van avaluar en el model predictiu i coincideixen amb las classes reals de la taula de predicció, que s'ha vist anteriorment.
Realitzem una altre execució del mateix test, però quan el test ens pregunti si volem emmagatzemar els resultats, contestarem que NO. L'objectiu és veure si amb les dades que ja s'han obtingut, l'algoritme és capaç de millorar les seves prediccions. Internament, cada vegada que es vulgui llançar una predicció, es construirà, amb les dades de l'arxiu *Control.arff*, un arbre de decisió. Aquest arbre, serà el mateix sempre i quant el contingut de l'arxiu no canviï i per tant,

totes les prediccions seran en base aquest. Executem de nou el test 6 i mostrem els resultats finals.



Il·lustració 37: Test6, resultat taula de prediccions, segona execució

Aquesta segona execució s'ha obtingut un error global del 20%, és a dir, de 5 peticions s'han encertat 4 i es poden veure algunes diferències, com per exemple que els costos reals i les prediccions han canviat. Per conseqüència, l'ordre d'execució també ha canviat i ara és: *Client1 – Client2 – Client3 – Client4 – Client6 – Client5*.

En definitiva, queda demostrat que mitjançant prediccions, es pot assignar prioritats a les peticions en espera. El nostre SMA, mitjançant aquestes prioritats, pot canviar l'ordre d'execució de les peticions i per tant, escollir aquelles, que de forma predictiva tenen un cost més baix. També queda demostrat que el sistema és capaç d'ajustar-se i cercar una solució més acurada a cada execució.
Entrarem amb molt més detall, en el punt 8.1, on aprofundirem en aquest sistema per esbrinar, si degut a aquestes característiques es millora el funcionament del SMA.

7.8 Test 7. Simulació amb carrers tallats

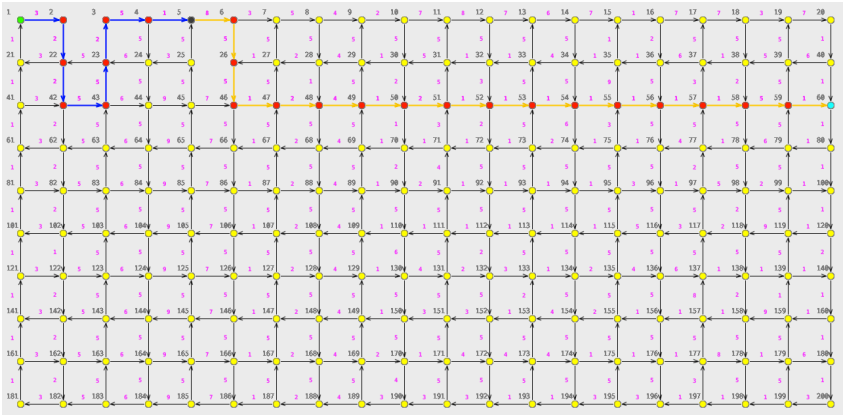
En aquest cas, les característiques de simulació són molt semblants al test 1, on un vehicle realitzarà el servei a una determinada petició. Aquest test, és pot executar mitjançant l'opció " 7. Test 7, cut streets ". Les dades d'execució, són les següents:

AgentCentral	AgentTraffic	AgentControl	AgentClient	AgentCar
Funcionament en mode d'execució lineal	Càrrega del plànol de la il·lustració 4.	Emmagatzemar dades.	Nom: Client1 Posició: 5 Destí paquet :60 Num.Paquets: 1	Nom: Car1 Time: 2000 Posició: 1 Estat: true Incident: False

Taula 43: Configuració agents Test 7

L'execució normal (és a dir, sense carrers tallats) no la detallarem, seria reiterar la descripció del test 1.

Seguidament tallarem el carrer que van des de la posició 2→3 i el carrer de la posició 26 → 25. El primer ha d'obligar al vehicle, a cercar un nou recorregut i el segon no ha d'influir en cap decisió. Per realitzar aquest test, li passem una llista de posicions a *AgentTraffic* i *AgentClient*, per a que modifiquin el plànol. El primer per a que reconegui els carrers que no es poden fer servir i el segon, per a que no els visualitzi en la interfície visual.
Una vegada executat el test, veurem que el vehicle ha rectificat el seu recorregut i el resultat final és el següent:



Il·lustració 38: Test7, execució amb carrers tallats

Queda demostrat que el sistema reconeix els carrers tallats i que els vehicles són capaços de rectificar el seu recorregut, envers a aquestes modificacions.

7.9 Test 8. Simulació d'incident en un vehicle

En aquest test anem a cercar una solució, en cas que un vehicle estigui actiu, es a dir, es trobi en la ciutat, però que hagi tingut un incident. Considerem un incident a situacions tals com: sense bateria, averia mecànica o elèctrica, accident de tràfic o aturat per motius aliens a la ciutat.
En aquest cas, *AgentCentral* detectarà aquesta situació, en el moment en que l'opció *Incident* de la configuració del vehicle, estigui activada. En el moment que es detecti aquesta situació, *AgentCentral,* marcarà el vehicle accidentat com no disponible i activarà un vehicle de suport, situat en una llista de vehicles a l'espera d'incidents. Aquest nou vehicle, prendrà el relleu i realitzarà els serveis.

Aquest test es pot executar mitjançant l'opció " 8. Test 8, Incident cars ". Tenim les següents dades d'execució:

AgentCentral	AgentTraffic	AgentControl	AgentClient	AgentClient	AgentCar
Funcionament en mode d'execució lineal. Llista de vehicles de suport: Car5	Càrrega del plànol de la il·lustració 4.	Emmagatzemar dades.	Nom: Client1 Posició: 200 Destí paquet :60 Num.Paquets: 1	Nom: Client2 Posició: 20 Destí paquet: 6 Num.Paquets: 1	Nom: Car1 Time: 2000 Posició: 20 Estat: true Incident: true

Taula 44: Configuració agents Test 8

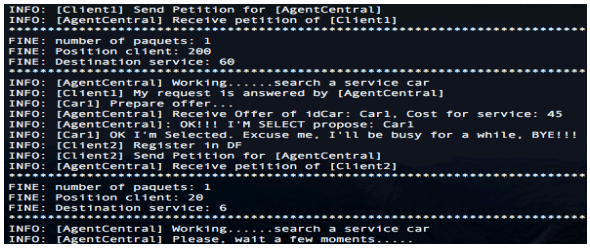
Car5 és el vehicle de suport i te les següents característiques:

Nom: Car5 ; Time:1000; Posició: 1; Estat: True; Incident: False;

Una vegada iniciat, *Client1* llançarà la seva petició que serà assignada a *Car1* (només tenim aquest vehicle). *Car1* començarà el servei, però tindrà un incident en alguna posició del recorregut. A priori no sabem en quina posició ho tindrà, perquè serà aleatori (pot ser en qualsevol posició de tot el recorregut). En el transcurs d'aquest primer servei, entra en joc *Client2*, que no trobarà vehicles disponibles i passarà a la llista d'espera.
Car1, enviarà el seu estat a *AgentCentral* i l'informarà que ha tingut un incident. Quan *AgentCentral* rebi la informació de l'estat i identifiqui l'incident, activarà un nou vehicle, d'una llista de vehicles de suport. En la informació rebuda, també s'inclou la posició actual del vehicle accidentat, per tant *AgentCentral* podrà

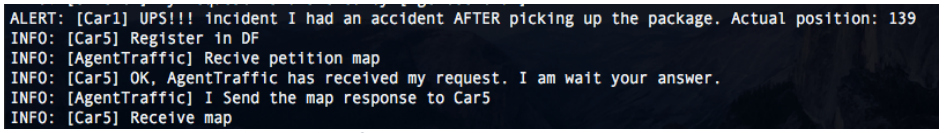
cercar *Car1* en la taula de peticions generals (aquesta taula, permet identificar el servei que el vehicle havia de realitzar). Una vegada recuperat el servei, *AgentCentral* te accés a la seva petició inicial i la modifica (identifica la posició per esbrinar si l'incident s'ha produït avanç d'arribar al client o després). Un cop modificada, la introdueix en la llista d'espera per a que sigui atesa, per un altre vehicle.
En aquest cas, l'únic vehicle que es troba en el sistema és *Car5* (vehicle de suport) que haurà de recuperar el servei de *Car1* i executar la petició de *Client1.* Finalment haurà de continuar amb la petició de Client2, que es troba en la llista d'espera.

Seguirem l'execució del test, ja que a cada execució la posició de l'incident de *Car1* pot canviar, però en tots els casos s'ha de complir el procediment.
Primerament, es pot observar com s'assigna *Car1* a la petició de *Client1*. Posarem especial atenció a la informació de la petició, ja que com es pot veure, *Client1* es troba en la posició 200 i vol enviar un paquet a la posició 60. Per la seva banda, es pot apreciar que *Car1* calcula el cost correctament i que inicia el seu servei amb normalitat. Seguidament, entra la petició de *Client2*, que és inserida en la llista d'espera.



Il·lustració 39: Test8, execució amb un vehicle accidentat

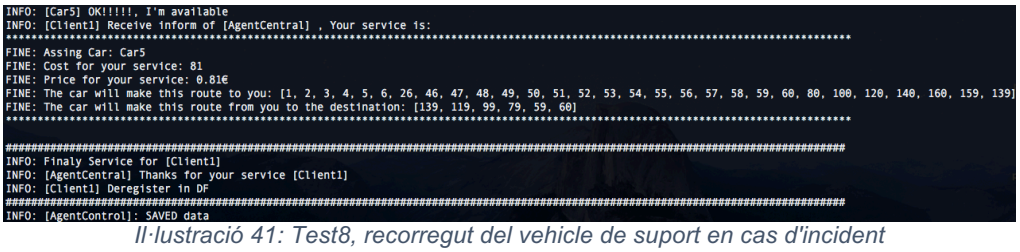
En un moment determinat del servei de *Client1*, *Car1* te un incident (es pot veure que es troba aturat en la posició 139) i ho fa saber a *AgentCentral*. En l'interior del missatge de *Car1*, viatja la informació relativa a la posició de l'incident. Es a dir, si l'incident s'ha produït en alguna posició del recorregut d'avanç d'arribar a *Client1*, *Car1* enviarà la posició de *Client1*. En cas contrari, es a dir, que l'incident s'hagi produït en alguna posició del recorregut entre client i destí, *Car1* enviarà la seva pròpia posició (en el nostre cas, els recorreguts que havia de realitzar *Car1* són: vehicle-client [20, 40, 60, 80, 100, 120, 140, 160, 180, 200] , client-destí [200, 199, 179, **139**, 119, 99, 79, 59, 60], per tant l'incident s'ha produït després de recollir el paquet). Aquest factor és determinat, perquè *AgentCentral* prendrà aquests valors per reescriure la petició. Una vegada modificada, s'inclou en la primera posició de la llista d'espera.
Automàticament, s'activa un altre vehicle (en aquest cas *Car5*, que es troba en una llista de vehicles de suport).



Il·lustració 40: Test8, informació de la posició d'un vehicle accidentat

Una vegada *Car5* es troba en servei, *AgentCentral* recupera la primera petició de la llista d'espera i l'assigna. Aquesta petició coincideix amb la que *Car1* no va

poder realitzar, ja que s'ha introduït al principi de la llista. Ara bé, *Car5* es troba inicialment en la posició 1 i com que *Car1* va tindre l'incident després de recollir el paquet, *Car5* haurà d'anar a cercar *Car1*, recollir el paquet i portar-lo a destí.



Il·lustració 41: Test8, recorregut del vehicle de suport en cas d'incident

Una vegada resolt l'incident i confirmat el servei, es continua amb *Client2*. Com que *Car1* està accidentat, *AgentCentral* l'ha marcat com a no disponible i és *Car5,* qui ha de fer-se càrrec de les altres peticions. En aquest cas, només la de *Client2*.

Algunes consideracions importants. En el cas d'un incident, el Client no te coneixement del que ha passat, és el propi sistema qui resolt el problema i només envia la informació al client, quan ja existeix un vehicle que ho pot realitzar.
El vehicle accidentat, una vegada comunica el seu estat, queda automàticament fora de servei.

Així, queda demostrat que el nostre SMA és capaç de solucionar incidents dels vehicles i recuperar els serveis que no s'han realitzat.

7.10 Test 9. Simulació General

En aquest últim test, es posa en pràctica tots els escenaris anteriors i te com objectiu justificar el bon funcionament de tot el sistema. Aquest test es pot executar mitjançant l'opció 9 " test 9. Total test. (*Cut Streets, incident car, prediction mode, many cars and many clients* ". Tenim les següents dades d'execució:

AgentCentral	AgentTraffic	AgentControl	AgentCar	AgentCar	AgentClient
Funcionament en mode d'execució predictiu. Llista de vehicles de suport: Car5	Càrrega del plànol de la il·lustració 4.	Emmagatzemar dades.	Nom:Car1 Time: 2000 Posició: 20 Estat: true Incident: true	Nom:Car2 Time: 2000 Posició: 58 Estat: true Incident: false	Nom: Client1 Posició: 130 Destí paquet: 60 Num.Paquets: 1

AgentClient	AgentClient	AgentClient
Nom: Client2 Posició: 20 Destí paquet: 6 Num.Paquets: 1	Nom: Client3 Posició: 90 Destí paquet: 180 Num.Paquets: 1	Nom: Client4 Posició: 100 Destí paquet: 200 Num.Paquets: 1

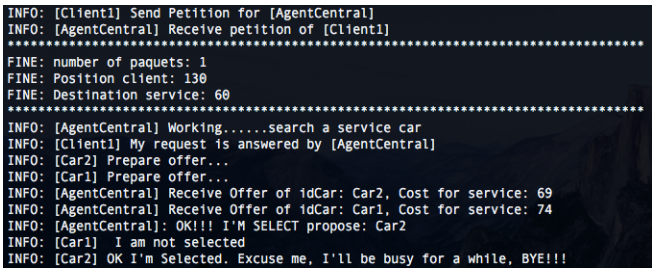
Taula 45: Configuració agents Test 9

Car5 és el vehicle de suport i te les següents característiques:

Nom: Car5 ; Time:1000; Posició: 1; Estat: True; Incident: False;

Llista de carrers tallats: 103→83 , 91→92 , 52→53 , 95→75
L'arxiu *Control.arff* només te la informació de l'execució del test 6.

Inicialment, el sistema rebrà la petició de *Client1*. En aquest cas, existeixen dos vehicles disponibles *Car1* i *Car2*. En el cas de *Car1* s'ha introduït un incident, així que quan sigui seleccionat per un servei, aquest fallarà.
En el nostre cas, *Client1* serà assignat a *Car2*, perquè és qui proporciona el cost més petit i quedarà en l'estat de no disponible, una estona.



Il·lustració 42: Test9, inici d'execució de peticions

La següent petició, *Client2* s'assignarà a *Car1*, ja que és l'únic vehicle disponible. Recordem que aquest vehicle tindrà un incident, així que *Car1* no serà capaç de finalitzar el seu servei.

Les peticions de *Client3* i *Client4* no es poden atendre, ja que els dos vehicles disponibles estan realitzant un servei. Així que aquestes peticions s'introdueixen en la llista d'espera.

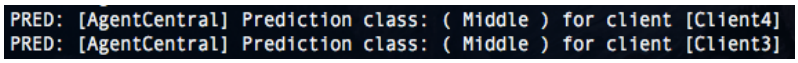
El sistema detecta que *Car1* ha tingut un incident en la posició 74. Això inicia la recuperació de la petició de *Client2* i l'activació de *Car5*.

Il·lustració 43: Test9, detecció d'un incident

Atenció: És possible que una nova execució d'aquest test no doni el mateix valor d'incident. Recordem que és un valor aleatori, d'entre els valors possibles dels recorreguts del vehicle accidentat.

Car2 assoleix el final del servei i dona per finalitzada la petició de *Client1*. *AgentCentral* comença la predicció de les peticions que es troben en aquell moment en la llista d'espera (Client3 i Client4).

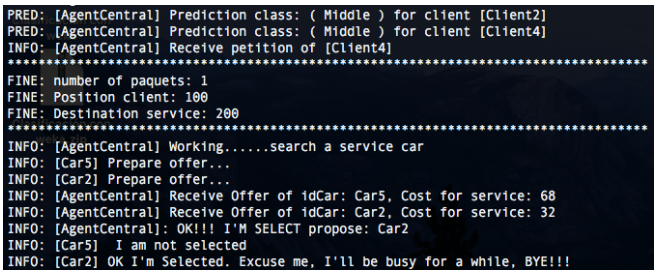
Recordem que s'està treballant en mode predictiu i per tant, s'assignaran prioritats a les peticions.



Il·lustració 44: Test9, prediccions i prioritats

En aquest cas, s'han assignat les mateixes prioritats i el sistema extraurà de la llista la primera que es va inserir, en el nostre cas *Client3*. El vehicle que es troba actiu és *Car2* i serà l'encarregat d'atendre la petició.
En el transcurs del servei de *Car2*, s'ha rebut l'estat de l'incident de *Car1*, s'ha recuperat la petició de *Client2*, s'ha inclòs la petició en la llista d'espera i s'ha activat *Car5*.
Quan arriba *Car2*, comença de nou el procés de predicció, però ara les peticions en espera són: *Clien2* i *Client4*. En execució lineal, la petició que s'hauria

d'atendre en aquest moment és la de *Client2*, però com estem en mode predictiu, és *AgentCentral* qui marca la prioritat.

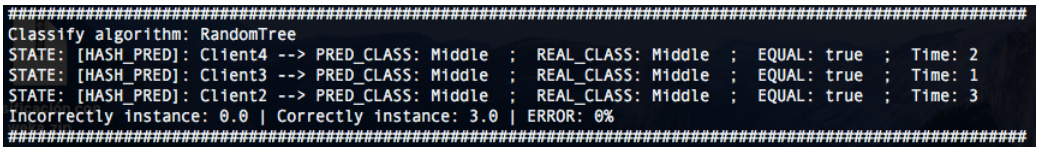


Il·lustració 45: Test9, execució segons el mode predictiu

En aquest cas, les prediccions han coincidit i seguint amb la condició de preservar (en la mesura del possible) el funcionament del mode lineal, s'extrau de la llista la primera petició que es va inserir, en el nostre cas *Client4*.
S'ha tornat a seleccionar a *Car2* per realitzar el servei, així que *Client4* queda assignat a quest vehicle.
Una vegada finalitzat el servei de *Client4*, es torna avaluar la llista d'espera. En el nostre cas, només queda la petició de *Client2* (que recordem és una petició que va patir un incident) i tenim dos vehicles disponibles *Car2* i *Car5*. Per costos, l'assignat és *Car2*. Recordem que *Car1* va patir l'incident en la posició 74. Això indica que *Car1* no va arribar a destí, però ja havia recollit el paquet. Així que, *Car2* haurà de desplaçar-se des de la posició 200 fins a la posició 74, recuperar el paquet i desar-lo en la posició 6 (destí).

Il·lustració 46: Test9, recuperació del servei accidentat

En aquests moments, ja no hi ha més peticions per atendre i finalitzen els serveis. La taula de prediccions ens mostra la informació relativa a les prioritats.



Il·lustració 47: Test9, resultat final de la taula de prediccions i ordre d'execució

Com es pot observar, s'ha comés un error del 0%, es a dir, en aquest cas s'han encertat totes les prediccions.
En definitiva, s'ha justificat el bon funcionament global del sistema, amb totes les condicions que es van proposar a l'inici.

8. Resolució final

En aquest punt, ja estem en condicions d'analitzar el nostre treball a un nivell més profund. De fet, tenim els coneixements teòrics necessaris, perquè s'han introduït en el transcurs de tot el treball i també tenim els escenaris pràctics, que s'han consolidat en punts anteriors.

Existeixen dos punts bàsics i destacables a tindre en compte a l'hora de llançar valoracions i conclusions. En primer lloc, és que ens hem proposat dissenyar i implementar un SMA que permet una solució alternativa, a una tasca existent en l'actualitat. Parlem, del transport de paquets en una determinada ciutat, mitjançant vehicles intel·ligents.
El segon punt destacable és que una vegada construït, ens hem proposat millorar la seva eficiència, optimitzant la part de selecció de clients.
En els següents punts valorarem la part teòrica i pràctica del nostre SMA i extraurem conclusions.

8.1 Anàlisis de les proves

En l'execució del test bàsic s'ha pogut comprovar, que el SMA actua com s'havia plantejat al començament d'aquest treball. Els clients poden enviar peticions amb informació en el seu interior. Un gestor central, analitza la petició i la situació actual del sistema, per tal d'escollir alternatives. Per la seva banda, els agents encarregats del transport, actuen segons les indicacions que els hi arriben en les peticions i els agents de suport, (un de tràfic i l'altre d'emmagatzematge d'informació), realitzen les tasques requerides en els moments desitjats. Davant d'aquesta tessitura, es pot dir que l'estructura general és correcte i modela d'una forma realista, la tasca de transport de paquets.

En el test de comprovació de dades, s'ha comprovat que la informació arriba i surt correctament a tots els agents. De fet, s'entenen i actuen envers a la informació que manipulen. En aquesta situació, es pot dir que també és un sistema realista, on tots els actors poden comunicar-se i entendre la informació que manipulen.

En el test de mode d'execució lineal, es comprova que es mantenen els criteris de selecció de peticions. Una de les tasques més naturals en un servei de transport, és la de mantenir una llista de clients en un ordre determinat. Les peticions s'atenen a mesura que van arribant i només en casos exclusius es trenca aquest ordre. Ara bé, aquest fet no sempre pot ser una bona solució i en aquest treball, es planteja l'alternativa mitjançant un mode d'execució predictiu.

La prova del model predictiu va demostrar que el sistema era capaç de assignar prioritats i canviar l'ordre d'execució. Però, en aquest cas és imprescindible aprofundir encara més, per esbrinar les conseqüències d'aquesta alternativa.

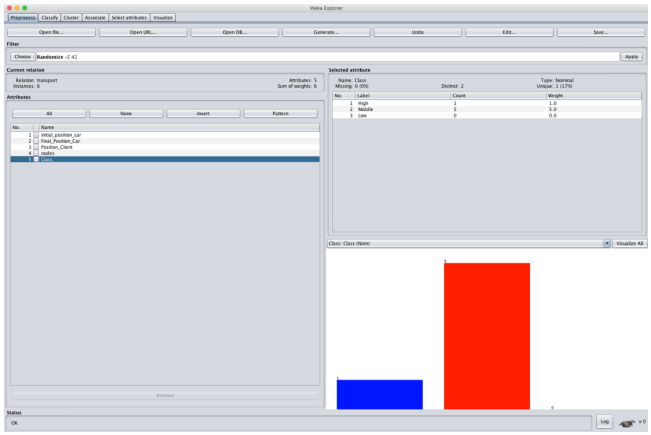
El programari *Weka*[18] és un software que permet realitzar tasques de *Data Mining*[19]. Permet preparar dades i aplicar algoritmes d'associació, clústering i classificació. És una eina molt interessant i instem al lector a aprofundir en les seves característiques, per un major coneixement.

Utilitzarem *Weka* com a eina de suport en el nostre anàlisis i treballarem en paral·lel amb la nostra aplicació.

[18] https://www.cs.waikato.ac.nz/ml/weka/
[19] https://es.wikipedia.org/wiki/Minería_de_datos

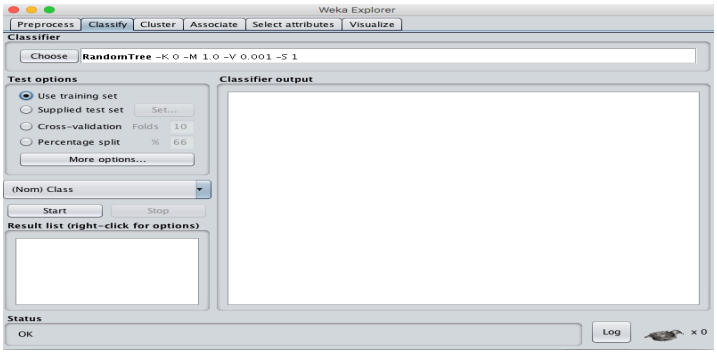
Partim de la base dels resultats obtinguts en el test 6, que es va estudiar en el punt 7.7. En aquest test es va realitzar una primera execució i es va aconseguir un 40% d'error total. Seguidament, es va realitzar una segona execució, aconseguint un 20% d'error total. Tal i com es va veure, en l'arxiu *Control.arff* es va escriure la informació real de l'execució dels serveis proposats. Aquest arxiu es considera un conjunt d'entrenament i contra ell, s'avalua la proposta que es vol predir.
També tenim un arxiu anomenat *test.arff* que emmagatzema la última petició avaluada (es pot consultar en cas necessari).

Introduïm l'arxiu *Control.arff* a *Weka*, tal qual es va finalitzar en el test 6.



Il·lustració 48: Interfície principal de Weka

Com es pot observar Weka permet carregar directament arxius .arff que estiguin configurats correctament. D'aquí el fet, de que en el nostre disseny ja tinguéssim en compte aquest factor. A part, les llibreries de *Weka* en *Java* permeten treballar directament amb aquest arxius, d'una forma molt còmode.
Ens interessa la pestanya *Classify* on es troben els algoritmes de classificació, entre ells *RandomTree*.



Il·lustració 49: Pestanya Classify de Weka

Ens centrarem en les 4 opcions que tenim a l'esquerra, que són diferents tipus de test sota el nostre algoritme.

- *Use training set* : Es mesura la qualitat del classificador per a predir la classe de les instàncies que ha sigut entrenat. És útil per a quan hi ha poques instàncies.

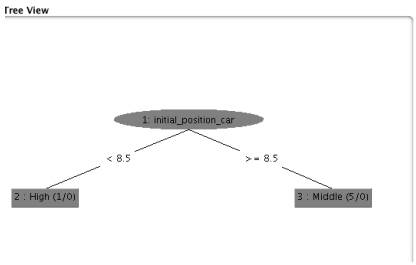
- *Supplied test set* : Es mesura la qualitat del classificador per predir la classe d'un conjunt d'instàncies que es troben en un arxiu. En el nostre cas, l'arxiu test.arff
- *Cross-validation* : Es mesura la qualitat del classificador mitjançant avaluació creuada, segons un nombre de conjunts fitxat.
- *Percentatge Split* : Es mesura la qualitat del classificador segons un tant per cent de les dades que es reserva per a test.

Percentatge Split, permet reservar un tant per cent de la mostra i mesurar la qualitat del classificador. Posarem un valor del 50% per avaluar-lo i veure l'arbre resultant.

```
=== Run information ===

Scheme:       weka.classifiers.trees.RandomTree -K 0 -M 1.0 -V 0.001 -S 1
Relation:     transport
Instances:    6
Attributes:   5
              initial_position_car
              Final_Position_Car
              Position_Client
              nodes
              Class
Test mode:    split 50.0% train, remainder test

=== Classifier model (full training set) ===


RandomTree
==========

initial_position_car < 8.5 : High (1/0)
initial_position_car >= 8.5 : Middle (5/0)

Size of the tree : 3

Time taken to build model: 0 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0 seconds

=== Summary ===

Correctly Classified Instances           3               100      %
Incorrectly Classified Instances         0                 0      %
Kappa statistic                          1
Mean absolute error                      0
Root mean squared error                  0
Relative absolute error                  0      %
Root relative squared error              0      %
Total Number of Instances                3

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0,000    0,000    0,000      0,000    0,000      0,000    ?         ?         High
               1,000    0,000    1,000      1,000    1,000      0,000    ?         1,000     Middle
               0,000    0,000    0,000      0,000    0,000      0,000    ?         ?         Low
Weighted Avg.  1,000    0,000    1,000      1,000    1,000      0,000    0,000     1,000

=== Confusion Matrix ===

 a b c   <-- classified as
 0 0 0 | a = High
 0 3 0 | b = Middle
 0 0 0 | c = Low
```

Il·lustració 50: resultats d'algoritme RandomTree a weka, del test6

De 6 instàncies, s'han reservat 3 i com es pot veure en la matriu de confusió, s'han classificat bé totes 3 (un 100% de precisió). Tot seguit, es pot veure l'arbre resultant.



Il·lustració 51: Arbre de decisió del test6

L'arbre és molt senzill i com es pot veure, només te en consideració l'atribut *Initial_position_car*. D'aquest factor ja s'indueix que no serà un bon model, encara que la seva precisió hagi sigut d'un 100%. Només cal fitxar-se en que existeixen instàncies en el nostre domini que serien mal classificades, per exemple les de classe *Low*. Ara bé, l'objectiu d'aquesta petita descripció no és la d'avaluar l'arbre, sinó més aviat la d'entendre com es planteja la seva estructura i com es pot interpretar la seva informació, de cara a futures valoracions, en aquest mateix punt.
La nostra següent tasca serà la d'augmentar la mida de les dades del fitxer *Control.arff*. Això permetrà que l'arbre tingui en consideració, més atributs de les

nostres instàncies i conseqüentment, una classificació més realista al domini presentat.
S'ha considerat inserir 30 instàncies aleatòries (10 de cada classe) que es mostren en la següent taula:

Num.instància	Atributs	Classe
1	1, 45, 3, 6	High
2	45, 16, 23, 21	Middle
3	16, 32, 40, 13	Middle
4	32, 45, 34, 18	Middle
5	45, 60, 1, 27	Middle
6	60, 5, 2, 27	Middle
7	45, 103, 89, 15	Middle
8	199, 1, 56, 31	Low
9	12, 19, 16, 7	Middle
10	10, 12, 11, 2	High
11	1, 1, 186, 28	Low
12	34, 3, 187, 32	Low
13	34, 32, 33 ,2	High
14	40, 1, 101, 28	Middle
15	112, 200, 1, 44	Low
16	35, 1, 60, 31	Middle
17	7, 27, 28, 3	High
18	95, 75, 97, 5	High
19	1, 1, 120, 48	Low
20	17, 1, 200, 40	Low
21	155, 155, 174, 4	High
22	33, 51, 28, 9	High
23	49, 176, 56, 17	Middle
24	19, 19, 1, 48	Low
25	13, 200, 1, 44	Low
26	134, 156, 136, 3	High
27	127, 127, 128, 4	High
28	41, 181, 60, 45	Low
29	66, 66, 65, 4	High
30	8, 1, 188, 25	Low

Taula 46: 30 instàncies aleatòries, 10 de cada classe

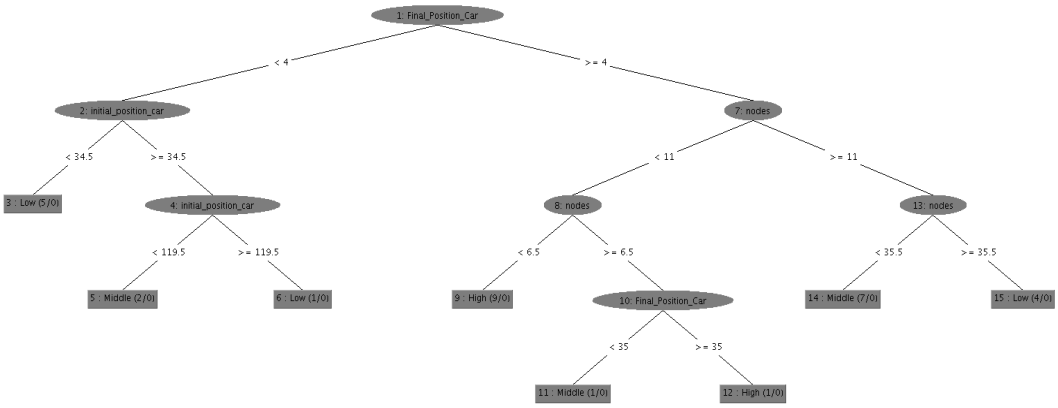
Seguidament s'ha realitzat un anàlisis a *Weka*. Per tal de augmentar l'aleatorietat entre instàncies, s'ha aplicat un filtre d'instància, que permet barrejar les mostres (filtre *Randomize* de *Weka*).
Tot seguit, la taula de resultats amb l'algoritme *RandomTree* i dos algoritmes més, del mateix estil (*RandomForest* i *J48*), amb l'objectiu de contrastar les diferències i similituds.

Algoritme	Precissió
RandomTree	73.33%
RandomForest	80.00%
J48	73.33%

Taula 47: Primers resultats amb 3 algoritmes classificadors

Les avaluacions s'han realitzat sota *Cross-validation* amb un valor de 10.
Som conscients, que encara és una mostra petita, però ens interessa veure l'impacte dels diferents atributs i aprofitarem per analitzar l'arbre resultant (mida i composició d'atributs).

Es mostra, tot seguit, l'arbre de l'anterior avaluació.



Il·lustració 52: Arbre de decisió a Weka amb 30 serveis, 10 de cada classe

L'arbre obtingut ens indica, que l'atribut amb més rellevància és *Final_position_car*. A partir d'aquesta arrel, les branques es distribueixen amb *Initial_position_car* i *nodes*.
Aquest resultat va en direcció contrària a la nostra primera hipòtesis, ja que esperàvem que l'atribut *nodes*, fos el predominant a l'hora de classificar.

Tot i això, amb aquesta informació en l'arxiu Control.arff, executarem de nou el test 6. L'objectiu d'aquesta nova prova, és observar si el mode predictiu a millorat i és capaç de llançar prediccions més ajustades. De fet, Weka ens a dit que el nostre model te una precisió del 73.3% i això voldria dir que hauríem d'encertar 4 prediccions de 5.

```
#########################################################################################################
Classify algorithm: RandomTree
STATE: [HASH_PRED]: Client6 --> PRED_CLASS: Middle  ;  REAL_CLASS: Middle  ;  EQUAL: true  ;  Time: 5
STATE: [HASH_PRED]: Client5 --> PRED_CLASS: Middle  ;  REAL_CLASS: Middle  ;  EQUAL: true  ;  Time: 4
STATE: [HASH_PRED]: Client4 --> PRED_CLASS: High  ;  REAL_CLASS: Middle  ;  EQUAL: false  ;  Time: 1
STATE: [HASH_PRED]: Client3 --> PRED_CLASS: Middle  ;  REAL_CLASS: Middle  ;  EQUAL: true  ;  Time: 3
STATE: [HASH_PRED]: Client2 --> PRED_CLASS: Middle  ;  REAL_CLASS: Middle  ;  EQUAL: true  ;  Time: 2
Incorrectly instance: 1.0 | Correctly instance: 4.0 | ERROR: 20%
#########################################################################################################
```

Il·lustració 53: taula de prediccions amb 30 serveis aleatoris, 10 de cada classe

Com es pot veure, l'error no a millorat, respecte al test 6. S'ha comprovat que l'execució del mateix test, amb els algoritmes *RandomForest* i *J48*, han donat el mateix resultat, un 20% d'error. És a dir, amb 30 mostres aleatòries no s'ha aconseguit millorar la predicció, que si observem, s'ha comés en la mateixa classe que en el test 6.

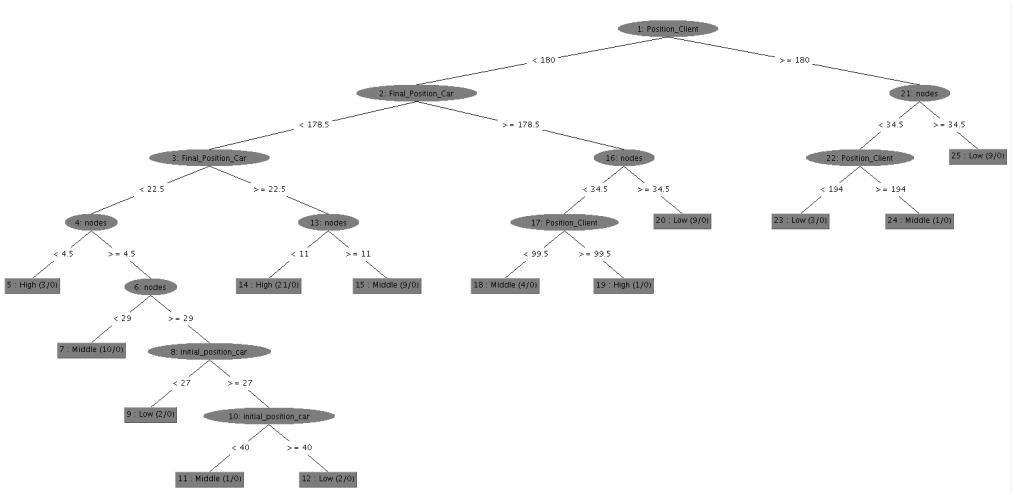
Tot i això, s'observa un augment de la mida de l'arbre, la quantitat d'atributs avaluats i que l'atribut *Position_client* no s'ha utilitzat.

Amb l'objectiu d'estudiar amb més detall el model predictiu, s'ha realitzat una altra prova amb 75 instàncies aleatòries (25 de cada classe, que es pot cercar en l'annex 2). En aquest cas, els resultats a Weka han sigut els següents:

Algoritme	Precissió
RandomTree	85.33%
RandomForest	85.33%
J48	84.00%

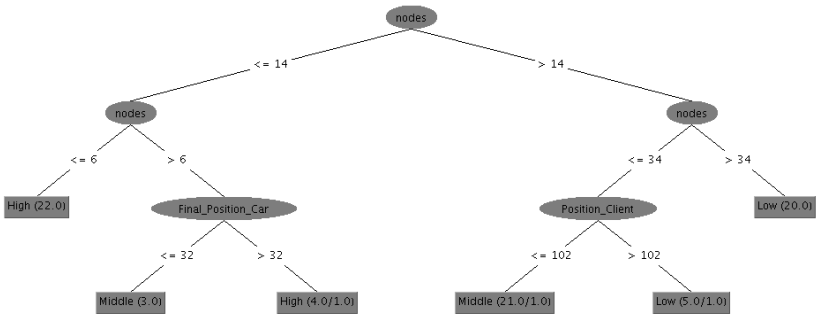
Taula 48: 75 instàncies, 25 de cada classe amb 3 algoritmes

I l'arbre resultant (en el cas de RandomTree):



Il·lustració 54: Arbre de decisió RandomTree 75 serveis aleatoris, 25 de cada classe

És interessant veure que l'arrel del model no és *nodes*. En canvi, l'arbre que construeix l'algoritme J48, sí te com arrel l'atribut *nodes*.



Il·lustració 55: Arbre de decisió J48, 75 serveis aleatoris, 25 de cada classe

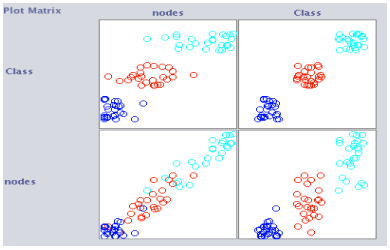
També s'observa, que *J48* te una mida inferior (degut a la seva característica de poda) a l'arbre generat per *RandomTree*, encara que ha obtingut una precisió lleugerament inferior. Però, si tenim present que el plànol utilitzat en aquest projecte, és d'una mida moderada i que en un futur, aquesta mida pot augmentar considerablement, l'opció de construir un model d'arbre de menys mida amb una precisió semblant, és molt interessant i desitjable.

El fet de no veure una millora en l'error global, ens fa pensar que les instàncies no estan equilibrades respecte al domini que es vol avaluar. De fet, pensem que els atributs de posició ens estan inserint soroll en el model i no aporten la informació de forma correcte.

Així que ens qüestionem si tots els atributs de les nostres instàncies són necessaris. Es podria construir un arbre que tingués una precisió igual o semblant, sense tindre en compte tots els atributs?. La resposta, en principi és sí.

Si recordem, les nostres instàncies tenen l'atribut *nodes*, que es va incorporar en el nostre disseny, per intentar separar peticions envers a les posicions

recorregudes en un servei. Si es fa un anàlisis gràfic d'aquest atribut es pot veure el següent:



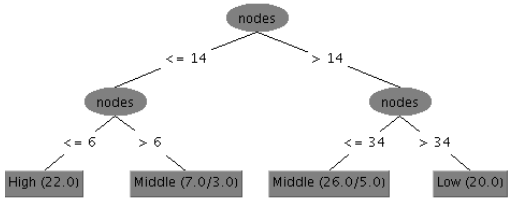
Il·lustració 56: Gràfic de distribució de l'atribut nodes

La gràfica mostra clarament l'agrupació de l'atribut *nodes* en diferents classes (blau=High, vermell=Middle, blau fluix=Low). La seva distribució ens indica, que és un atribut que classifica d'una forma bastant clara cada classe (tot i que hi ha alguna instància dispersa). Si avaluem el nostre model, només amb aquest atribut, obtenim el següent resultat:

Algoritme	Precissió
RandomTree	84.00%
RandomForest	82.66%
J48	82.66%

Taula 49: Resultats amb només l'atribut nodes

L'arbre obtingut amb l'algoritme RandomTree te una mida més elevada que la de J48, així que mostrarem aquest últim.



Il·lustració 57: Arbre de decisió J48, exclusivament amb l'atribut nodes

Les nostres inquietuds eren certes i s'observa clarament que amb l'atribut *nodes*, seriem capaços de crear un model molt similar en precisió, al model amb tots els atributs.

L'arbre és molt senzill i mostra de forma clara la classificació d'instàncies. Si el valor de *nodes* és igual o inferior a 14, es torna a avaluar el mateix atribut. Si en aquest segon cas, el valor de *nodes* és igual o inferior a 6, la instància es considera de classe *High*. Si el valor de *nodes*, és superior a 6 (recordem que serà un valor entre 14 i 6), la instància serà considerada de classe *Middle*. La branca dreta de l'arbre, s'avaluaria de la mateixa forma i queda clar que si el valor de *nodes* és superior a 34, la instància serà considerada de la classe *Low*.

Una vegada analitzat el sistema predictiu, es pot concloure que l'atribut *nodes* és el centre de la predicció en la llista d'espera. De fet, sense aquest atribut no aconseguiríem una predicció més enllà del 60% de precisió, tal i com està dissenyat el sistema. La resta d'atributs es limiten, en algun cas, a afinar la predicció d'unes instàncies determinades. De fet, la següent captura de pantalla de *Weka*, mostra una avaluació amb les mateixes instàncies (75 en total, 25 de

cada classe) sense tindre en compte l'atribut *nodes*. L'avaluació s'ha realitzat mitjançant l'algoritme RandomTree.

```
Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances          48               64      %
Incorrectly Classified Instances        27               36      %
Kappa statistic                          0.46
Mean absolute error                      0.24
Root mean squared error                  0.4899
Relative absolute error                 53.8981 %
Root relative squared error            103.7166 %
Total Number of Instances               75
```

Il·lustració 58: Avaluació del model sense l'atribut nodes

L'arbre resultant és força gran i no el mostrarem, però es pot veure clarament l'impacte que l'atribut nodes ocasiona en les instàncies que es volen avaluar.

S'ha de tindre en compte, que el càlcul del nombre de nodes visitats que realitza l'*AgentCentral*, en el moment de llançar una predicció, no és un càlcul exacte. Aquest fet també influeix en la predicció final i per tant, un error que s'ha de tindre present. Tot i això, s'han aconseguit precisions força elevades, tenint en compte la mida de l'arxiu de dades. A més, el seguit de proves demostren que a mesura que s'augmenta la mida de l'arxiu *Control.arff*, la precisió de les prediccions augmenta, degut a que l'arbre pot avaluar més casos i ajustar de forma més correcte els valors de tall.

9. Valoracions i conclusions

Hem arribat al punt àlgid del nostre treball. És ara, amb tota la informació teòrica, pràctica i amb els nous coneixements adquirits, on es donaran resposta a totes les qüestions que s'han plantejat, en el transcurs d'aquest treball.

La base principal d'aquest projecte, es centrava en el servei de transport de paquets dintre d'una ciutat, mitjançant vehicles intel·ligents. Es plantejava un escenari modern, on la ciutat i els diferents dispositius que hi participaven podien interconnectar-se i comunicar-se entre ells, per tal de realitzar un servei amb la mínima dependència humana.

Com es va descriure, en els primers apartats d'aquest treball, l'actual sistema de transport de paquets en una determinada ciutat, te actualment una gran dependència dels humans. Aquesta dependència provoca que en alguns escenaris, el servei es torni ineficient, amb les conseqüències que això comporta. Factors com la picaresca humana, els accidents laborals i la vida social dels treballadors, són alguns dels exemples del que estem parlant.

La primera qüestió que ens plantejàvem, era si l'actual tecnologia era capaç d'implementar aquest servei i a més, si seria possible optimitzar-lo.

A la primera part d'aquesta qüestió, es pot contestar que sí existeix tecnologia suficient, per implementar un servei d'aquestes característiques. L'actual avanç en vehicles i ciutats intel·ligents, amb la combinació de potents eines de

programació, permeten el disseny i implementació d'un projecte d'aquestes característiques.

Ara bé, no n'hi havia prou amb una implementació freda que modelés l'escenari descrit, ens interessava esbrinar si era possible delegar decisions i evitar així, en la mesura del possible, la intervenció humana. Com ja sabem, les situacions reals tenen tendència a reproduir escenaris imprevisibles i ens interessava obtenir un sistema, que s'adaptés a la majoria d'aquestes situacions. Amb aquesta inquietud, es va abordar el concepte de IA.

La intel·ligència artificial ens permetia dotar als nostres dispositius d'una determinada consciència es a dir, permet que un dispositiu, davant un determinat estímul, reaccionés de forma intel·ligent. Aquesta necessitat, que plantejava el nostre projecte, ens va conduir fins als agents.
Un agent permetia realitzar tasques senzilles, d'una forma ràpida i eficient. Ens proporcionava reaccions davant estímuls i un nivell alt d'autonomia. La seva programació s'adaptava perfectament als estàndards actuals i permetien un nivell alt d'escalat. Tot i això, el nostre objectiu era mes ambiciós, ja que un servei de transport de paquets no es pot reduir a una sola tasca. Necessitàvem un sistema capaç de coordinar i que mitjançant tasques senzilles, realitzés un objectiu final. En aquest punt, ens vam adonar que era necessari un escenari comú, molt semblant al real, on els agents poguessin interactuar entre ells.

Un SMA ens proporcionava tots aquest factors. Un escenari on poden conviure tots els agents necessaris (que pot ser perfectament una ciutat), on els agents es poden comunicar entre ells (comunicació mitjançant missatges) i que mitjançant tasques senzilles coordinades, s'assolís un objectiu final comú.

Una vegada resolta la filosofia de disseny, es va aprofundir en el sistema de comunicació entre diferents agents i l'elaboració d'un model de simulació (JADE com a llibreria de suport).

Assolits els coneixements previs, es va iniciar l'arquitectura general del treball. D'aquesta forma, es va simular un escenari real (una ciutat, vehicles, clients, una central, un sistema de control del tràfic i un sistema de control), tenint en compte les necessitats del servei que es volia reproduir. Per tal d'apropar-nos a situacions reals, es van incloure incidents ocasionals (talls de carrers i avaries en els vehicles) i els funcionaments típics del dia a dia (atendre un o més clients, clients que no poden ser atesos en el moment de la seva petició, deficiència en el nombre de vehicles i elecció de vehicles amb el millor servei).

Les simulacions van demostrar que és possible un nivell d'autonomia alt. Els agents, de forma conjunta, proporcionen una gran capacitat de resolució de problemes, sense oblidar el seu objectiu final i han demostrat, que la seva implementació ha estat a l'alçada en tots els escenaris proposats.

El sistema de gestió de clients és capaç de separar completament el servei, de la informació final, es a dir, el client demana i rep la informació, sense preocupar-se del que ha succeït internament. Si existeix alguna incidència, és el sistema qui la soluciona, el client no ha de participar en aquestes tasques. La gestió de

clients no depèn d'un horari, ni tampoc estarà absent per voluntat pròpia, sempre atendrà als clients que ho necessitin.

Els vehicles, encara que inicialment són capaços de circular pels carrers, s'han dotat d'intel·ligència pròpia. En aquest sentit, potser hagués sigut una bona idea, incloure diferents estratègies, a l'hora de transportar un paquet. Per exemple, establir en un inici, quadrants de posicions assignades als vehicles. No hagués sigut gaire complicat, implementar certes estratègies en els vehicles i deixar que els propis vehicles decideixin la millor alternativa.
Tot i això, en el nostre sistema, els vehicles són capaços de preveure el cost d'un servei i d'actuar en conseqüència. Mai enganyaran al client o a l'empresa i continuaran actius sempre que sigui necessari, evidentment sempre que no hi hagi qualsevol incident que ho prohibeixi.

Davant de canvis en el tràfic, un agent és capaç de reproduir la situació i transmetre la informació quan li sigui demanada. El model en aquest cas, és molt senzill, però pot escalar-se fins a punts inimaginables.

El sistema és capaç d'emmagatzemar qualsevol situació, incident o informació derivada dels dispositius, per si fos necessari una posterior valoració. La informació és vital en un sistema d'aquestes característiques, ja que permet valorar situacions i de millorar-les en cas necessari.

La coordinació central és el centre neuràlgic de tot el sistema (és l'equivalent a l'atenció d'un servei actual d'una empresa de transports de paquets). És en aquest dispositiu, on s'atenen i es verifiquen les peticions dels clients, es negocia amb els vehicles els costos mes baixos, s'assignen serveis als vehicles, es gestionen incidents, es confirmen les tasques finals i s'informa al client dels resultats. Per pal·liar les conseqüències on, temporalment no hi ha vehicles disponibles, s'implementa una llista d'espera de clients. Aquesta llista es gestiona sota dos modes d'execució, el mode lineal que permet atendre les peticions en el mateix ordre que es van inserir i el mode predictiu, que assigna prioritats de sortida.
Totes les tasques es realitzen de forma autònoma i no és necessària la intervenció humana.

Sota aquesta tessitura, es pot afirmar que és completament possible implementar un servei de transport d'aquest tipus i que a més, funcioni correctament. La similitud d'aquest disseny amb l'actual sistema de transport, permet concloure que, amb un nivell de disseny elevat, pot arribar a substituir-lo.

Encara que el disseny i implementació del SMA d'aquest projecte ja ha sigut una càrrega elevada de treball i un gran repte personal, degut al desconeixement inicial de molts dels conceptes, s'ha volgut avançar una mica més. Aquest segon objectiu ha tingut com a punt de mira, l'optimització del servei a un nivell més profund, per tal d'aconseguir un augment de l'eficiència. Soc conscient que existeixen diferents factors que poden interferir en el bon funcionament del sistema, però he volgut centrar el punt de mira en la selecció de clients, introduint un mode predictiu de selecció.

El model predictiu a demostrat que és capaç d'assignar prioritats, a les peticions que es troben en espera. Aquest mode, permet d'assignar i executar peticions amb una precisió força elevada, que en un principi s'haurien d'atendre en ordre diferent. Aquest factor influeix positivament en el sistema, perquè s'atenen peticions més ràpidament i en conseqüència, es disminueix la mida de la llista d'espera en un temps més reduït. Ara bé, de cara al client amb un servei de cost alt, s'augmenta el seu temps en la llista d'espera, degut a que s'atenen en primera instància les peticions amb una predicció de cost més baix. Aquest fet pot ser un punt negatiu, ja que en primera instància s'està castigant a un client, que vol realitzar un servei amb cost alt.

Tot i això, considerem que el mode predictiu és un sistema molt interessant, perquè cada cop que es cerca un vehicle amb disponibilitat de realitzar un servei, es calculen de nou les prediccions de prioritat tenint en compte les noves posicions i aquest fet, pot modificar l'anterior prioritat d'una determina petició. D'aquesta forma, una petició que va ser assignada amb un cost alt, en una altra iteració, pot considerar-se amb un cost més baix i canviar la seva prioritat en la llista.

Ara bé, les conclusions finals sobre els dos models de funcionament, no considero que siguin prou aclaridores. S'ha comprovat que en el model lineal s'executen les peticions en el mateix ordre i s'assignen als vehicles que estiguin disponibles en aquell moment. Això permet mantenir l'ordre, però amb les proves realitzades no és possible extraure si ens ofereix un augment d'eficiència, perquè necessitaríem esbrinar els temps de cada servei. Això ens obliga a crear un escenari, on sigui possible combinar el cost del servei amb el temps necessari per realitzar-lo.
El model predictiu dona la possibilitat de canviar l'ordre d'execució, però tampoc ens ofereix la possibilitat d'esbrinar el temps que aquests canvis representen. Weka ha demostrat que les instàncies que es plategen en aquest projecte, es centren en un únic atribut, *nodes*. Això, en certa mesura era d'esperar, ja que les classe representen costos i aquests costos venen en funció de la quantitat de nodes visitats. A més nodes visitats, més possibilitat d'augmentar el cost.
En definitiva, no es pot assegurar que en tots els escenaris, el model predictiu sigui millor que el lineal, però si es pot afirmar que el model predictiu, permet assegurar que es seleccionen correctament els costos dels clients en espera, amb un nivell de precisió força alt. Aquest fet, permet d'extraure de la llista d'espera, aquelles peticions que s'haurien d'executar en un menor temps, amb la conseqüent disminució de recursos globals. Degut a que es calculen de nou totes les prediccions a cada servei, existeix la possibilitat de canviar la prioritat de les peticions en temps real, tenint en compte les posicions dels vehicles i de la pròpia petició. És en aquesta situació, on els altres atributs de la nostres instàncies poden afinar les prediccions.
Sota aquesta última hipòtesis i sense oblidar que s'hauria d'aprofundir molt més, es pot dir que el model predictiu ens aporta cert augment d'eficiència, que el model lineal no pot, ja que el model predictiu permet assignar les peticions envers al cost i no al torn de sortida.

En el dia a dia, és inevitable que existeixin incidents, per tant és necessari implementar un protocol d'actuació en aquests casos. Un sistema d'aquestes

característiques necessitaria un estudi molt acurat i extens en aquest tema i que en aquests moments s'escapa a l'abast del nostre treball. Tot això, s'ha inclòs un parell d'incidents típics, per tal de comprovar el funcionament general. El primer d'ells és un dels més quotidians dintre d'una ciutat, els talls de carrers i el segon, l'incident d'un vehicle en el moment de realitzar un servei o durant el propi servei.

S'ha comprovat que el sistema és capaç de cercar nous recorreguts, en el cas de carrers tallats, per tal de realitzar serveis, sense que això signifiqui un canvi molt significatiu. Per altra banda, en el moment que un vehicle pateix un incident, un altre vehicle agafa el relleu i el sistema, és capaç de recuperar l'estat anterior i donar solució sense la implicació del client.
De fet, en el moment de l'incident s'activa automàticament un altre vehicle, però no amb l'objectiu de realitzar exclusivament el servei accidentat. El que es pretén, és anivellar les possibilitats de servei, es a dir, si existien 2 vehicles en el moment de l'incident, s'activa un altre per mantenir aquest nombre de vehicles en actiu. El servei que no s'ha pogut realitzar, es recupera i s'introdueix de nou en el sistema, d'aquesta forma, qualsevol dels vehicles actius pot arribar a realitzar-lo.

10. Propostes de futur

S'ha arribant a la part final del nostre treball. El recorregut ha sigut llarg i s'han assolit multitud de conceptes, que després s'han aplicat al disseny. Un disseny, que en un principi es va iniciar amb unes determinades característiques, que de forma dinàmica i gracies al model iteratiu de la gestió de projectes, s'han ampliat i polit. S'han assolit tots i cadascuns dels objectius que s'havien plantejat, tant teòrics com a pràctics, del model proposat.
Però, com tot disseny, després de la seva implementació i posterior posta en marxa, és susceptible a millores.

10.1 Millores

El sistema proposat està obert a infinitat de millores.

Es pot incloure la possibilitat de que un client pugui realitzar dos o més serveis en una mateixa petició. Amb el disseny actual només es permet una petició per client i si el mateix client vol realitzar una altra petició, ha de tornar a iniciar el procés. Seria interessant incloure la possibilitat de que un client, en una mateixa petició, pugui realitzar l'enviament de diferents paquets a diferents posicions. Aquest fet obre la porta a una millor col·laboració entre vehicles, perquè per tal de preservar el costos més baixos, els vehicles haurien de comunicar-se i intercanviar-se els paquets, segons les seves posicions geogràfiques. Es a dir, si un client necessites enviar dos paquets a diferents posicions, primer s'assignaria un vehicle al paquet amb menys cost, però el vehicle recolliria tots dos en el mateix moment. Aquest vehicle, durant el recorregut de lliurament del primer paquet, hauria de negociar amb els altres vehicles, per tal d'esbrinar quina opció és més eficient (si lliurar el segon paquet ell mateix o intercanviar-lo amb un altre vehicle que estigués per la zona i que proporcionés un cost més baix).

Seguretat en les comunicacions. La tecnologia de la informació és susceptible a la manipulació i pèrdua de dades. En un disseny amb vehicles intel·ligents, no tripulats i circulant per una ciutat, un factor de vital importància és la seguretat. Si un usuari malintencionat, aconseguís el control remot d'un vehicle podria provocar incidents a immobles de la pròpia ciutat o de forma molt més greu a vianants. Degut a aquest fet, seria prioritari un sistema de xifrat i protecció de les dades, tant dels propis dispositius com dels missatges que intercanvien.

Ampliar l'estructura del servei tan de forma terrestre, aèria i marítima. El nostre treball s'ha centrat en una ciutat i s'ha presentat com un model de servei a peu de carrer, es a dir, totes les posicions de recollida i lliurament, es suposen que es troben a un lloc accessible pel vehicle. Però, i si s'ha de creuar un riu en el recorregut o simplement s'ha de fer el lliurament a dalt d'un edifici ?. La solució pot passar per incloure alguns vehicles destinats exclusivament a serveis d'aquest tipus i que siguin assignats, quan sigui necessari.

La nostra ciutat és relativament petita, en el cas que tingués una mida molt més gran, seria interessant l'assignació de vehicles per sectors. La idea seria implementar un sistema central, però en aquest cas distribuït i que cadascun tingués assignat un o uns sectors determinats. En el nostre treball, les peticions que realitzen els clients són molt diverses i pot succeir que un client vulgui enviar un paquet a un lloc molt pròxim a ell. Sota aquesta situació, el nostre disseny assigna els vehicles que estiguin disponibles amb un mínim de cost, però no te en compte si existeixen vehicles, encara que estiguin realitzant algun servei, que per sector estiguin més a prop. La distribució dels vehicles per sectors, ens assegura que uns vehicles determinats s'ocuparan exclusivament d'aquell escenari i que els seus agents centrals, els assignaran conseqüentment.

Per últim, seria interessant anar més enllà d'una ciutat concreta. És a dir, aprofitant les característiques de connexió dels SMA, seria interessant que un client pugui enviar un paquet d'una ciutat a una altra. Cada ciutat estaria controlada per un agent central i es comunicarien entre ells, per tal d'assignar els recursos necessaris en aquest cas.

Com s'ha descrit a l'inici d'aquest punt, les millores i propostes de futur són immenses. L'objectiu final, és oferir un millor servei als usuaris amb una alta eficiència, un cost baix i la possibilitat d'eliminar possibles desviacions humanes, que poden ser perfectament implementades per dispositius intel·ligents.

11. Reflexió i valoració personal

El plantejament del projecte, en la seva fase inicial, va estar sotmesa a dubtes. De fet, centrar de forma definitiva el temari va causar un cert retard a l'inici de la planificació, degut a diferents punts de vista. Però, aquest retard va ser absorbit al poc temps i no ha tingut un impacte significatiu.

En aquest sentit, haig de donar les gracies al consultor *David Isern Alarcón,* per les seves intervencions i guia, en el plantejament d'aquest projecte.

El món de la intel·ligència artificial m'apassiona i he gaudit realitzant aquest treball. De fet, m'hagués agradat aprofundir molt més en alguns aspectes d'aquest projecte, com per exemple la seguretat de la informació, però estic convençut que en un futur no gaire llunyà, hi haurà ocasió de desenvolupar-lo.

M'ha semblant necessària, tota la formació que he rebut en el transcurs del grau. Des de la part de redacció de documents i gestió de projectes, fins a càlculs matemàtics i aplicació d'algoritmes específics. Soc conscient, que sintetitzar tota aquesta docència en un sol projecte no és fàcil, però considero que són necessàries totes elles, en el transcurs del seu desenvolupament.

He descobert moltes coses que he volgut combinar amb els coneixements que ja s'havien assolit. Els SMA i els agents eren un tema poc desenvolupat per la meva part, encara que tenia alguna petita noció, gracies a assignatures com IA i Aprenentatge computacional.

Abordar el plantejament i el disseny, va ser un esforç considerable, però vaig pensar que també era l'ocasió de posar en pràctica, els coneixements assolits en programació en JAVA. Per tant, la decisió d'implementar tot el sistema, també a tingut un pes considerable a nivell de programació. De fet, la planificació general ja predisposava un temps força alt, dedicat exclusivament a la part de codi.

Les simulacions i anàlisis dels casos estudiats m'han donat alguna sorpresa. De fet, gracies a la implementació dels test de prova, es van posar en evidència alguns errors de programació, indetectables en primera instància i alguns resultats força interessants, sobretot en el test del model predictiu.

En definitiva, un treball que no el considero definitiu, més aviat l'inici d'un camí.

12. Glossari

A continuació es defineixen els termes i acrònims més rellevants que es poden cercar dins d'aquest projecte.

Terme	Significat
SMA	Sistema multiagent
JADE	JAVA Agent Development Framework
FIPA	Foundation for Intelligent Physical Agents
ACL	Agent Comunication Language
DF	Directory Facilitator
AMS	Agent Management System
AID	Agent identificator
FIFO	First in First out

Taula 50: termes i acrònims

13. Bibliografia

- Data Mining and Analysis: fundamental Concepts and Algoritms. Mohammed J.Zaki, Wagner Meira Jr. Document .pdf

https://repo.palkeo.com/algo/information-retrieval/Data%20mining%20and%20analysis.pdf
Data de consulta [4/10/2017]

- Jade Programmer's guide.
  Fabio Bellifemine, Giovanni Caire, Tiziana Trucco, Giovanni Rimassa
  Last Update (8/04/2010)
  http://jade.cselt.it/doc/programmersguide.pdf
  Data de consulta [6/10/2017]
- The Foundation for intelligent Physical Agents
  Pàgina Web
  http://www.fipa.org
  Data de consulta [7/10/2017]

- Estudio de la implementación de agentes en dispositives móviles
  Alexandre Viejo Galicia. Document .pdf
  http://deim.urv.cat/~itaka/itaka2/PDF/Awards/Viejo.pdf
  Data de consulta [5/10/2017]

- Sistema multiagente.
  Wikipedia. Document web. (11/05/2017)
  https://es.wikipedia.org/wiki/Sistema_multiagente
  Data de consulta [4/10/2017]

- Agentes inteligentes i los sistemes multiagente. Sesion 2.
  Marin Lujak. Document .pdf
  http://www.ia.urjc.es/cms/sites/default/files/userfiles/file/MUII-IAD/2014-15/IAD_Sesion2.pdf
  Data de consulta [6/10/2017]

- Modelado de sistemes multi-agente
  Jorge J. Gómez Sanz. Document .pdf
  http://grasia.fdi.ucm.es/jorge/tesis.pdf
  Data de consulta [7/10/2017]

- Aprendizaje automàtico.
  Wikipedia. Document web. (2/08/2017).
  https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico
  Data de consulta [4/10/2017]

- Vehículo Autónomo.
  Wikipedia. Document web (24/09/2017)
  https://es.wikipedia.org/wiki/Veh%C3%ADculo_aut%C3%B3nomo
  Data de consulta [4/10/2017]
- Ciudad inteligente.
  Wikipedia. Document web (20/09/2017)
  https://es.wikipedia.org/wiki/Ciudad_inteligente
  Data de consulta [4/10/2017]

- Problemática de las actuaciones de las empreses de transporte en las empreses clientes. Document pdf
http://www.osalan.euskadi.eus/contenidos/informacion/5_foro_gipuzkoa_prl/es_jt120903/adjuntos/presentacion_2_adegi.pdf
Data de consulta [12/10/2017]

- Dificultades en las empreses de paqueteria, y no son exclusives de España. Document Web (21/12/2016)
https://cronicaglobal.elespanol.com/ecoonomia/tecnologia-ecoonomia/dificultades-en-las-empresas-de-paqueteria-y-no-son-exclusivas-de-espana_65331_102.html
Data de consulta [12/10/2017]

- El colapso en las empreses de paqueteria complica la campaña de Reyes de Amazon. Document Web (5/01/2017)
http://www.elconfidencialdigital.com/dinero/empresas-paqueteria-complica-Reyes-Amazon_0_2850914887.html
Data de consulta [12/10/2017]

- La logística se pone en pie de guerra contra Amazon por los bajos precios Document Web (3/07/2017)
http://www.eleconomista.es/empresas-finanzas/noticias/8471440/07/17/La-logistica-se-pone-en-pie-de-guerra-contra-Amazon-por-los-bajos-precios.html
Data de consulta [12/10/2017]

- Cómo organitzar el Reparto de las vacaciones entre los empleados de su empresa. Document Web (20/06/2017)
http://www.expansion.com/pymes/2017/06/20/5943c54e46163f7e6c8b45cb.html
Data de consulta [12/10/2017]

- [Il·lustració 1] Imatge Internet : http://www.torke.com.ar/images/pmbok-processes.png
Data de consulta [12/10/2017]

14. Annexos

En aquest apartat es trobaran els annexos corresponents. Contingut que es considera massa extens, com per incloure-ho en el cos del treball.

14.1 Annex 1

Contingut de l'arxiu *Control.arff* amb 75 instàncies (25 de cada classe). Arxiu original, sense l'aplicació del filtre *Random* de *Weka*.

```
@relation transport
@attribute initial_position_car NUMERIC
@attribute Final_Position_Car NUMERIC
@attribute Position_Client NUMERIC
@attribute nodes NUMERIC
@attribute Class {High,Middle,Low}
```

@data
1,45,3,6,High
45,16,23,21,Middle
16,32,40,13,Middle
32,45,34,18,Middle
45,60,1,27,Middle
60,5,2,27,Middle
45,103,89,15,Middle
199,1,56,31,Low
12,19,16,7,Middle
10,12,11,2,High
1,1,186,28,Low
34,3,187,32,Low
34,32,33,2,High
40,1,101,28,Middle
112,200,1,44,Low
35,1,60,31,Middle
7,27,28,3,High
95,75,97,5,High
1,1,120,48,Low
17,1,200,40,Low
155,155,174,4,High
33,51,28,9,High
49,176,56,17,Middle
19,19,1,48,Low
13,200,1,44,Low
134,156,136,3,High
127,127,128,4,High
41,181,60,45,Low
66,66,65,4,High
8,1,188,25,Low
90,125,101,17,Middle
2,200,78,27,Middle
12,200,18,21,Middle
45,1,133,30,Low
22,24,23,6,High
90,92,91,2,High
1,20,9,23,Middle
46,199,99,22,Middle
10,185,60,34,Middle
87,85,86,6,High
12,54,55,8,High
60,21,61,22,Middle
44,45,64,5,High
23,100,49,22,Middle
9,17,18,16,Middle
120,1,200,32,Middle
70,200,1,40,Low
180,120,160,13,Middle
15,200,100,14,High
3,199,101,35,Low
14,103,41,24,Middle
55,2,55,17,Middle
12,12,14,8,Middle
67,65,66,2,High
101,103,102,6,High
133,131,132,6,High
27,18,12,16,Middle
2,1,200,55,Low
4,1,200,53,Low
5,1,200,52,Low
6,1,200,51,Low
7,1,200,50,Low
8,1,200,49,Low
9,1,200,48,Low
200,200,1,56,Low
200,200,2,56,Low
200,200,4,52,Low
9,11,10,2,High
22,1,21,2,High
18,58,58,2,High
172,1,200,37,Low
120,200,1,52,Low
129,131,131,2,High
166,168,168,2,High
164,144,165,3,High