GESTIÓN DINÁMICA DE PLAZAS DE APARCAMIENTO PARA VEHÍCULOS PESADOS A TRAVÉS DE SISTEMA MULTIAGENTES

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Realizado por:

Raquel García Arévalo - ragarar2@posgrado.upv.es

Tutor: ***Vicent Botti***

TABLA DE CONTENIDOS

Introducción 4
Objetivos del Trabajo 5
Conceptos Básicos 6
Definición de Agente Inteligente 6
Sistema Multi-Agente 7
Simulación Basada en Agentes 7
Comunicación entre Agentes: Estándar FIPA 7
Ontología 8
Diseño del Sistema 8
Agentes del Sistema 9
Gestor de Trayectos - AgJourneyManager 9
Áreas de Servicio - AgRestAreaManager 10
Plaza - AgParkingSpot 10
Agentes de Apoyo 10
Ontología 11
Conceptos 11
Acciones 12
Ampliaciones de la Ontología propuestas para futuras versiones 13
Restricciones 16
Tiempos de Conducción 16
Hechos sancionables 16
Otras Restricciones 17
Restricciones aplicadas en el sistema 17
Negociación de la Reubicación 17
Plataforma: JADE 19

Protégé 20
Diseño de Interfaz 20
Agent.GUI 20
WindowBuilder 21
Desarrollo 21
Fase 1 21
Fase 2 23
Interfaz 23
AmpliAción de la Ontología 23
Desarrollo de los Agentes 24
FASE 3 27
Mejoras de Interfaz 27
Gestión del Tiempo 28
Heurística Modo Libre 29
Formato Definitivo de los Ficheros 30
Pruebas Realizadas 31
Pruebas Básicas 31
Pruebas Finales 34
Resultados 34
Conclusiones 35
Visión de futuro 36
Trabajo futuro 37
Agradecimientos 39
Glosario 40
Glosario del dominio del problema 40
Bibliografía 43
Anexo I – Código Fuente del Proyecto 44

Ontología 44

GUI 60

Agents 72

INTRODUCCIÓN

En la actualidad gran parte del transporte de mercancías en Europa se realiza a través de la red de carreteras. Dado el impacto de este tipo de transporte en la economía, en la actualidad existen ya muchas iniciativas que optimizan estos tiempos de transporte, y parece ser un campo con espacio para la mejora.

Una de las restricciones que tienen los transportistas en sus trayectos son los tiempos de conducción máximo permitidos que pueden realizar antes de un descanso. Estos tiempos de conducción y descanso están regulados por la legislación europea [1]. Estas restricciones tienen como objetivo que un transportista realice la conducción en las condiciones físicas y mentales adecuadas, evitando así los riesgos potenciales.

Dado que los descansos se deben realizar en áreas de servicios aptas para ello y puede existir una limitación física al espacio de aparcamiento, es necesario un sistema que distribuya de un modo eficiente a los transportistas en las distintas áreas de servicio. Se quiere evitar que una mala gestión del espacio fuerce a un conductor a infringir la ley al no poder realizar su descanso reglamentario o un aparcamiento en un lugar inadecuado.

El Parlamento Europeo en la Directiva ITS: Sistemas de Transporte Inteligentes [2] plantea la importancia de crear sistemas de gestión inteligente de las plazas de aparcamiento para tratar de resolver estos problemas, y se están desarrollando y aplicando distintas soluciones, pero usualmente centradas en la monitorización individual de las áreas de servicio, por lo que se plantea que una coordinación entre los distintos elementos implicados podría llevar a una mejor solución.

Este proyecto realizado en colaboración con la universidad Jaume I, en el contexto de una serie de trabajos relacionados, propone como posible solución crear un sistema multiagente basado en la propuesta de negociación de Capdevila M. [3] que permita la gestión dinámica de ese espacio por medio de la interacción de los agentes.

Este documento explica el trabajo realizado en el proyecto de la forma siguiente:

- Un apartado de diseño donde se explica el planteamiento general del proyecto a realizar, con la explicación de los agentes, datos e interacciones entre agentes a desarrollar.
- En el apartado de desarrollo se explican las fases de desarrollo incremental realizadas, con los resultados para cada una de ellas.
- En el apartado de conclusiones se evalúan los resultados obtenidos y se plantean las conclusiones generales del proyecto.

OBJETIVOS DEL TRABAJO

La finalidad de este proyecto es comprobar que se pueden maximizar los tiempos de conducción de los transportistas entre descansos mediante la gestión dinámica de las plazas de aparcamiento disponibles en las áreas de servicio y reducir el número de conflictos que puedan surgir durante las reservas (y de este modo las posibles multas por exceso de tiempo de conducción o aparcamiento en lugares inadecuados) mediante el uso de la negociación basada en el protocolo de votación Borda.

Para ello se va a implementar una simulación utilizando un sistema multiagente, que estará compuesto por los agentes principales: Gestor de Trayectos (agente que realiza las reservas), Área de servicio (agente que recibe las solicitudes de reserva) y Plazas (agente que en que se realiza la reserva). Adicionalmente se pretende que este sistema pueda servir como base para futuros trabajos, y mediante ampliaciones ir acercándose a lo que pudiera llegar a ser un sistema real.

Dentro de los objetivos de la Gestión Inteligente de aparcamiento de vehículos pesados, se pretende conseguir mejoras en los siguientes aspectos:

- Mejorar la capacidad de aparcamiento de la red de carreteras, al optimizar el uso de las plazas.
- Ayudar a los conductores a maximizar su tiempo de conducción permitido legalmente y a la vez cumplir con sus periodos de descanso.
- Evitar el aparcamiento ilegal, mejorando por tanto la seguridad en las carreteras y el acceso a las áreas de servicio.

CONCEPTOS BÁSICOS

En este apartado se explicarán los conceptos básicos utilizados en el documento.

DEFINICIÓN DE AGENTE INTELIGENTE

El término agente viene del latín *agere* que significa hacer. Expresa la capacidad de acción o actuación de una entidad.

En el contexto de la informática, un agente es un sistema computador capaz de realizar acciones autónomas en algún entorno con el propósito de conseguir sus objetivos de diseño.

En este ámbito, se considera que un agente es una aplicación capaz de realizar acciones de forma **inteligente** y **autónoma** dentro de un entorno concreto, con el que se comunica, para conseguir sus objetivos.

Las propiedades básicas de un agente para ser considerado como tal, sugeridas por Wooldridge y Jennings (1995) [4], son las siguientes:

- **Reactividad**: los agentes perciben su entorno y responden a los cambios en el mismo en un tiempo aceptable.
- **Proactividad**: los agentes pueden tomar la iniciativa para tratar de cumplir sus objetivos.
- **Habilidad social**: los agentes deben ser capaces de interactuar con otros agentes o con usuarios humanos mediante algún método de comunicación.

Otras posibles propiedades que pueden tener los agentes:

- Movilidad: La habilitad del agente de moverse dentro de una red.
- Veracidad: Un agente no va a comunicar información falsa.
- Benevolencia: Los agentes no tienen objetivos contradictorios, y cada agente siempre intentará realizar lo que se le solicite.
- Racionalidad: El agente actuará para conseguir sus objetivos, y no realizará acciones que vayan en contra de los mismos.
- Aprendizaje / Adaptación: El rendimiento de los agentes mejora con el tiempo.

SISTEMA MULTI-AGENTE

Un sistema multiagente (SMA) es un sistema distribuido que se compone de varios agentes que interactúan entre ellos, y que se utilizan para resolver problemas complejos.

Las características de un agente dentro del sistema son:

- **Autonomía**: Los agentes actúan de forma autónoma e independiente al resto de agentes.
- **Visión local**: Cada agente tiene conocimiento sólo de la información que le afecta, ningún agente tiene visión global de todo el sistema, o el sistema es demasiado complejo para que un agente pueda hacer uso práctico de disponer de toda la información.
- **Descentralización**: No hay un agente de control designado (de no ser así, sería una aplicación monolítica)

Los agentes dentro del sistema, para interactuar correctamente, tienen que ser capaces de cooperar, coordinarse y negociar, considerando que no todos los agentes van a tener los mismos intereses. Estos sistemas multiagentes tienen la ventaja de que si el objetivo o tarea a alcanzar puede descomponerse en objetivos o tareas más simples, se puede distribuir la carga de trabajo, aplicando la heurística “divide y vencerás”.

SIMULACIÓN BASADA EN AGENTES

Los SMA pueden utilizarse también para la simulación de un entorno determinado, emulando por medio de agentes los comportamientos e interacciones de las diferentes entidades del entorno, pudiendo así determinar qué efectos se producen en el conjunto del sistema, a este tipo de sistemas se les conoce como Modelos basados en agentes o Simulaciones basadas en agentes.

En estos modelos, los agentes que los componen actúan de forma independiente en base a sus propios intereses, pero se utilizan para observar comportamientos globales en el entorno simulado por el sistema.

COMUNICACIÓN ENTRE AGENTES: ESTÁNDAR FIPA

FIPA (Foundation for Intelligent Physical Agents) es una organización cuyo objetivo es estandarizar los modelos y tecnologías de agentes. La estandarización es necesaria para lograr la interoperabilidad de los sistemas basados en agentes (facilidad de interconexión e integración) y su apertura (posibilidad de extensión).

En las especificaciones de FIPA se definen las características que deben cumplir las plataformas de gestión de sistemas multiagentes. Ejemplos de implementaciones creadas a partir de este estándar son FIPA OS y JADE

En relación con la plataforma de agentes, FIPA especifica el comportamiento y la interfaz que se debe respetar con el fin de permitir la interoperabilidad entre plataformas, quedando a cargo del desarrollador las decisiones de diseño de los componentes.

ONTOLOGÍA

El término **ontología** en informática hace referencia a la especificación formal y explícita de unos conceptos compartidos referentes a un dominio concreto, legibles por una computadora, con la finalidad de facilitar la comunicación y compartición de la información entre diferentes sistemas.

Esto aplicado a los agentes permite la comunicación entre aquellos que tengan definida una ontología común para poder entender los conceptos que se intercambian en sus mensajes.

DISEÑO DEL SISTEMA

En este apartado se explicará el planteamiento general del sistema a implementar.

El sistema es un simulador que cuenta con tres agentes principales, Gestor de Trayectos (AgJourneyManager), Área de Servicio (AgRestAreaManager) y Plaza (AgParkingSpot). Se asume que cada vehículo está representado con un agente Gestor de Trayectos, que se encargará de planificar la ruta a partir de la información facilitada por el conductor (en la simulación estos datos serán generados aleatoriamente) y otras fuentes de información, obteniendo una posible ruta.

Las áreas de servicio estarán representadas también por un agente Área de Servicio, y adicionalmente existirá un agente Plaza por cada plaza del área de servicio, de forma que se libere de carga de trabajo al agente Área de servicio y el sistema sea más modular y ampliable.

Una vez obtenida una posible ruta, el Gestor de Trayectos determinará intentando maximizar el tiempo de conducción, en qué área de servicio se debería hacer la primera parada, y negociará con las áreas de servicio del tramo que puede realizar hasta encontrar una plaza libre. Cuando realice la reserva, quedará pendiente de una posible reubicación en caso de que otro vehículo requiera su plaza y tenga más preferencia. Si un vehículo no pudiera realizar una reserva, saldrá del sistema indicando que ha causado una infracción.

Esquema de la comunicación entre los agentes del sistema:

AGENTES DEL SISTEMA

A continuación se plantea una explicación general de cada uno de los agentes que forman parte del sistema.

Se ha planteado para la simulación a realizar el desarrollo de los siguientes agentes:

- Gestor de Trayectos
- Área de Servicio
- Plaza
- Agentes de apoyo

GESTOR DE TRAYECTOS - AgJourneyManager

Se plantea que cada conductor y/o vehículo tendría su propio Gestor de Trayectos, que se encargará de planificar las paradas y negociar la reserva de las plazas, considerando los Datos del Conductor y los Datos del Vehículo.

Cuando un agente Gestor de Trayectos tiene que planificar una parada, recopilará una lista de Áreas de servicio preferidas, y solicitará una plaza en la más deseada indicando la hora prevista y la duración.

- Si la Área de servicio tiene plaza disponible, se realiza la reserva.
- Si no tuviera plaza disponible, el Área de Servicio iniciará una negociación para tratar de reubicar los vehículos que tienen reserva en esa franja horaria, incluyendo al vehículo solicitante.
- Si el Área de servicio indicase que no hay reubicación posible, se intentará reservar en la siguiente de la lista.

Ver explicación detallada en el apartado de Negociación de la reubicación.

Si al agente Gestor de Trayectos se le solicita una reubicación (tanto si es él quien está solicitando actualmente la plaza o si tenía una reserva previa y otro vehículo requiere una plaza), consultará una posible reserva de plaza a la siguiente Área de servicio preferida.

- Si encuentra plaza en la siguiente Área de Servicio deseada, informará del tiempo perdido al Área de servicio que solicitó la reubicación, que le informará si debe confirmar la nueva reserva y cancelar la inicial.
- Si la siguiente Área de servicio no tuviera plaza, se pasará a la siguiente de la lista de Áreas de Servicio preferidas.

ÁREAS DE SERVICIO - AgRestAreaManager

El agente Área de Servicio se encargará de coordinar la ocupación de sus plazas, actuando como intermediario entre los agentes Gestor de Trayectos y Plazas.

También gestionará junto con los agentes Gestores de Trayectos la reubicación de vehículos en caso de que se soliciten más plazas de las disponibles para una misma franja horaria, proponiendo a todos los Gestores de Trayecto que tengan reserva de plaza en esa franja horaria y al nuevo solicitante que re-planifiquen sus paradas omitiendo el Área en conflicto. Los Gestores de Trayectos a los que se haya solicitado la reubicación informarán si es o no posible, y en caso de ser posible, indicarán el tiempo de conducción ininterrumpida que perderían con la reubicación. Una vez obtenidas las respuestas, la Área solicitará la reubicación al vehículo con una pérdida menor.

Ver explicación detallada en el apartado de Negociación de la reubicación.

PLAZA - AgParkingSpot

Gestionará la información del Histórico de ocupación, registrando las reservas o anulaciones de las mismas y facilitará su información al área de servicio a la que pertenece cuando se la soliciten.

AGENTES DE APOYO

Se han creado adicionalmente unos agentes de apoyo para la simulación.

LOADER - AgLoader

El agente Loader se encarga del control de la simulación, realizando las siguientes tareas:

- La generación de los agentes necesarios para realizar la simulación a partir de información de ficheros de texto o de datos aleatorios.
- Control del tiempo de la simulación.
- Mostrar y conectar con la Interfaz de usuario para recuperar la información de configuración de la simulación.

GESTOR DEL REGISTRO - AgLogManager

El agente de registro LogManager se encarga de gestionar los ficheros de log, recibir la información del resto de agentes, y registrarla.

ONTOLOGÍA

En este apartado se van a detallar los conceptos que disponemos dentro del sistema representados con una ontología, con todos los atributos que podrían llegar a utilizarse y las acciones que pueden realizar. En la fase actual del sistema nos hemos limitado sólo a los datos más necesarios para la simulación, posteriormente podría añadirse más información.

CONCEPTOS

CARRETERA - ROAD

Información sobre las carreteras contempladas en el sistema:

- Identificador de carretera – roadId
- Nombre de la carretera – Name
- Indicador de tráfico – traffic: indica el nivel de congestión de la carretera.
- Longitud – lenght: Es la longitud de la carretera (km máximo), utilizado para la creación de los datos aleatorios y para limitar las búsquedas.

DATOS DEL CONDUCTOR – DRIVER

Detalla la información referente a un conductor.

- ID del conductor – driverId.
- Nombre – name
- Tiempo de conducción – drivingTime: Se ha simplificado la simulación limitándonos a registrar el tiempo ininterrumpido que el conductor ya lleva conducido.

DATOS DEL VEHÍCULO – VEHICLE

Información sobre cada uno de los vehículos que dispondrán de un Agente gestor de trayectos en el sistema.

- ID del Vehículo - idVehicle
- Velocidad máxima - maxSpeed
- Km en que se encuentra actualmente - km
- Carretera en que se encuentra actualmente - road
- Conductor para el trayecto - driver

ÁREA DE SERVICIO - RESTAREA

Información sobre las áreas de servicio del sistema.

- Identificador del área de servicio - idRestArea
- Nombre - name
- Número de plazas – spots
- Carretera en que se encuentra – road: Se considera la carretera principal desde la que se tiene acceso al área de servicio.
- Kilómetro de la carretera en que se encuentra el acceso al área de servicio – km.

RESERVA – RESERVATION

Información sobre una reserva concreta, que se utilizará para almacenar los registros de ocupación de las plazas de aparcamiento, y también para intercambiar información sobre las reservas entre los agentes.

- ID del Gestor de trayectos que ha realizado la reserve – JourneyManagerID
- Hora inicio – timeStart
- Hora fin – timeEnd

ACCIONES

A continuación se detallan las distintas acciones que realizan los agentes.

Nombre - Clase	Origen	Destino	Descripción	Atributos
Solicitar plaza - ReservationRequest	Agente Gestor de Trayectos	Área de Servicio	Un Gestor de Trayectos solicita reservar una plaza en el Área de Servicio.	Reservation
Solicitar ocupación de plaza - OccupationRequest	Agente Gestor de Trayectos	Área de Servicio	Un gestor de trayectos realiza una consulta para saber si existe una plaza libre en el Área de Servicio en la hora en que está prevista su llegada a ella.	Reservation
Solicitar ocupación de plaza - OccupationRequest	Área de Servicio	Plaza	El Área de Servicio solicita saber la ocupación de la Plaza en la franja horaria requerida.	Reservation
Informar de liberación de plaza – CancelRequest	Área de Servicio	Plaza	El Área de Servicio indica a la Plaza que libere la franja horaria de un vehículo en su ocupación.	Reservation
Registro de Reubcación - LogRelocation	Área de Servicio	Gestor del Registro	Cada vez que se produce una reubicación se registra información sobre la misma.	time, affectedVehicles, relocableVehicles, success, relocatedVehicleOrder, relocatedVehicleLostTime
Registro de Trayectos - LogJourneyData	Gestor de Trayectos	Gestor del Registro	Cuando un vehículo sale del sistema o finaliza la simulación, se registran información sobre su trayecto.	drivedTime, lostTime, restTime, km, kmIn, kmOut, TimeIn,TimeOut Infraction (0/1)

INFORMACIÓN A REGISTRAR

Dentro de la ontología se han definido las siguientes acciones, para informar al agente LogManager de la información que debe registrar para luego sacar indicadores y estadísticas de las simulaciones realizadas.

DATOS DE REUBICACIÓN – LOGRELOCATION

Registro realizado por las Áreas de Servicio con la información de cada reubicación que se produce en el sistema.

- Hora de la reubicación
- Número de vehículos afectados
- Número de vehículos que pueden reubicarse
- Éxito: Si/No
- Orden del vehículo que se ha reubicado
- Tiempo perdido por el vehículo reubicado

INFORMACIÓN SOBRE EL TRAYECTO – LOGJOURNEYDATA

Información que el agente Gestor de Trayectos recopila sobre el trayecto realizado en cada carretera:

- Tiempo total de conducción
- Tiempo perdido (Suma de todos los periodos desde el tiempo de conducción continuada realizado hasta cada parada y el tiempo máximo de conducción continuada).
- Tiempo total de descanso
- Kilómetros conducidos
- Kilómetro de entrada a la carretera
- Kilómetro de salida
- Hora de entrada
- Hora de salida
- Infracción: Salida sin infracción (0), Salida por infracción (1)

Actualmente se registra una vez en el sistema, si se introdujese un sistema de trayectos por varias carreteras habría que replantearlo o enviar un registro por cada carretera.

AMPLIACIONES DE LA ONTOLOGÍA PROPUESTAS PARA FUTURAS VERSIONES

DATOS ADICIONALES PARA CONDUCTOR

En vez de registrar el tiempo de conducción de cada conductor, sería preciso disponer de su registro de tiempos completo con la Información de todos los trayectos realizados que se necesitan para poder aplicar todas las posibles restricciones en la planificación de las nuevas rutas y paradas.

- Fecha – Hora inicio

- Fecha – Hora Fin
- Estado: Trayecto, Descanso

Esta información habitualmente se registra en el tacógrafo que los vehículos de transporte de mercancías y pasajeros deben llevar por ley.

DATOS ADICIONALES PARA VEHÍCULO

Datos adicionales que podrían registrarse sobre los vehículos para su uso en el sistema:

- Tipo de Mercancía: Existen restricciones de aparcamiento de ciertos tipos de mercancías que requieren de condiciones especiales.
- Carga, Altura y Longitud máxima: Algunas carreteras tienen restricciones según la carga, altura o longitud del vehículo debido a las restricciones físicas de la misma, como puedan ser pavimentos poco resistentes, puentes, túneles o giros cerrados.
- Conductor(es) para el trayecto: Podrían registrarse más de un conductor por vehículo, ya que en ocasiones se utiliza este método para realizar menos descansos y poder realizar trayectos largos en menos tiempo.

TIPOS DE MERCANCIA

Basándonos en la clasificación habitual de mercancías peligrosas del Acuerdo Europeo relativo al Transporte Internacional de Mercancías Peligrosas por Carretera [5], planteamos la siguiente codificación para registrar el Tipo de Mercancía transportada por el vehículo, esto será útil para trabajos futuros, donde se puede añadir una condición a la heurística del sistema según la cual ciertas mercancías no pueden estar próximas a otras:

Código	Clase	Descripción
00	-	Mercancías no peligrosas
01	-	Alimentos
10	Clase 1	Materias y objetos explosivos
20	Clase 2	Gases
30	Clase 3	Materias líquidas inflamables
41	Clase 4.1	Materias sólidas inflamables
42	Clase 4.2	Materias susceptible de inflamación espontánea
43	Clase 4.3	Materias que, al contacto con el agua, desprenden gases inflamables
51	Clase 5.1	Materias comburentes
52	Clase 5.2	Peróxidos orgánicos
61	Clase 6.1	Materias tóxicas
62	Clase 6.2	Materias infecciosas
70	Clase 7	Materias radiactivas
80	Clase 8	Materias corrosivas
90	Clase 9	Materias y objetos peligrosos diversos

Se asume que un vehículo sólo puede transportar un Tipo de mercancía.

DATOS ADICIONALES PARA ÁREA DE SERVICIO

Información adicional que se podría disponer de un área de Servicio:

- Dirección: Sería interesante para la vinculación del sistema con aplicaciones tipo mapa.
- Accesible desde: Sentido A / Sentido B /Ambos sentidos…
- Coordenadas: Otra alternativa para la localización en mapas.
- Servicios: Gasolinera / Parking /Restaurante /Alojamiento…
- Horario: Se podrían aplicar filtros según el horario del área de servicio.
    - Día de la semana
    - Desde hora
    - Hasta hora
- Valoración Media: Se podría añadir un sistema de valoraciones de las áreas de servicio e incluir esta valoración en la heurística.

Ejemplo de datos completos:

Nombre	Ctra Ppal	Km	Nº Plazas	Dirección	Accesible desde	Coordenadas	Horario		Servicios
L'horta	V-31	11		Camino Puente de Piedra, 3 46910 Sedaví	Dirección Valencia		Lunes Martes Miércoles Jueves Viernes Sábado Domingo	0:00–23:30 0:00–23:30 0:00–23:30 0:00–23:30 0:00–23:30 0:00–23:30 0:00–23:30	

DATOS ADICIONALES PARA HISTÓRICO DE OCUPACIÓN

Podría guardarse además el Tipo de Carga para su uso en restricciones de aparcamiento según la carga transportada.

RESTRICCIONES

Existen las siguientes restricciones respecto a la conducción, descanso y aparcamiento en el transporte de mercancías y pasajeros:

TIEMPOS DE CONDUCCIÓN

Según la legislación vigente sobre tiempos de conducción:

- **Conducción ininterrumpida**
  Tras un período de conducción de cuatro horas y media, el conductor hará una pausa ininterrumpida de al menos 45 minutos, a menos que tome un período de descanso.
  Podrá sustituirse dicha pausa por una pausa de al menos 15 minutos seguida de una pausa de al menos 30 minutos, ambas intercaladas en el período de conducción de 4 horas y media.
- **Conducción diaria**
  El tiempo máximo de conducción diario no puede exceder de 9 horas, salvo dos veces a la semana que puede llegar a las 10 horas.
- **Conducción semanal**
  El tiempo de conducción semanal no superará las 56 horas (se entenderá por semana el período de tiempo comprendido entre las 00.00 del lunes y las 24.00 del domingo).
- **Conducción bisemanal**
  El tiempo de conducción en dos semanas consecutivas no puede exceder de 90 horas.

  Así, si en una semana se conduce durante 56 horas (máximo permitido), en la siguiente sólo podrá conducirse durante 34 horas, puesto ambas suman el máximo de 90 horas.

HECHOS SANCIONABLES

Cualquier exceso en tiempo de conducción se considera sancionable, por lo que el sistema tratará de evitar siempre los excesos.

- Faltas leves (Sanciones desde 301 a 400 €): El exceso en los tiempos máximos de conducción o de la conducción ininterrumpida, salvo que deba ser considerado infracción grave o muy grave.
- Faltas graves (Sanciones desde 1.501 a 2.000 €): El exceso superior al 20 por ciento en los tiempos máximos de conducción o de conducción ininterrumpida, salvo que dicho exceso deba ser considerado infracción muy grave, de conformidad con lo previsto en el artículo 140.20.
- Faltas muy graves (Sanciones desde 3.301 a 4.600 €):El exceso superior al 50 por ciento en los tiempos máximos de conducción o de conducción ininterrumpida.

Para la simulación, si se determina que un vehículo no podrá disponer de una plaza con la que no se salte los tiempos de conducción, no se contemplará y su Gestor de trayectos dejará el sistema, registrándose una infracción. No se determinará si la infracción se produce por aparcar fuera de las plazas establecidas o por exceso de tiempo de conducción permitido, ya que sería una suposición aleatoria y simplemente se contemplará que se ha producido una infracción.

OTRAS RESTRICCIONES

Proximidad de Mercancías: Según el tipo de carga se podría restringir el aparcamiento a un vehículo si hay otro vehículo incompatible en el área de servicio se denegará la reserva de plaza y el agente deberá buscar otra Área de servicio en la que parar [7].

Disponibilidad de plazas: El sistema debe controlar que no se sobrepase el número de plazas disponibles de una Área de servicio.

RESTRICCIONES APLICADAS EN EL SISTEMA

Para simplificar la simulación y centrarnos en la heurística en sí, vamos a limitarnos a controlar la restricción de **conducción ininterrumpida máxima de 4,5 horas**, de forma que los vehículos intentarán maximizar el tiempo de conducción para quedar lo más cerca posible del máximo, pero sin superarlo.

NEGOCIACIÓN DE LA REUBICACIÓN

La negociación para la reubicación está basado en el método planteado por el artículo de Capdevila M.[1], pero se ha decidido delegar la negociación entre los distintos agentes involucrados en vez de disponer de un agente *Manager* que se encargue de gestionar la negociación de manera centralizada.

Se plantea que sea cada Área de Servicio la que plantee la reubicación de los Gestores de Trayectos involucrados en una negociación y determine qué vehículo debe reubicarse.

Se ha tomado la decisión de eliminar la figura del *Manager* para evitar que un solo agente tenga acceso a los datos de todos los vehículos y de la información de ocupación de todas las áreas de servicio (pues se considera que esta es información sensible a ser sustraída). Además de este modo se reduce la magnitud del problema, y por tanto se mejora el tratamiento del mismo de cara a que el sistema llegase a gestionar un gran número de vehículos y áreas de servicio, pues no habría que gestionar y planificar la reubicación con todos los vehículos que transiten una carretera o una zona, sino sólo con los que tuviesen previsto aparcar en las áreas de servicio saturadas en determinado horario.

El funcionamiento de la negociación será el siguiente:

- Se inicia cuando un Gestor de Trayectos solicita la reserva de una plaza a un Área de Servicio y no hay ninguna plaza disponible en el horario requerido en dicha área.
- El Área de Servicio solicitará a las Plazas ocupadas en esa franja horaria los identificadores de todos Gestores de Trayectos de los vehículos que ocupan las plazas.
- Enviará un mensaje a todos los Gestores de Trayectos de dichos vehículos, solicitando que verifiquen si pueden reubicar su reserva a otra Área de Servicio.

    - Los vehículos que vayan a ocupar alguna plaza del Área de servicio en los próximos quince minutos rechazarán la negociación, ya que en ese plazo consideramos la reserva inamovible.
- Cada Gestor de Trayectos buscará si encuentra una plaza libre donde poder reubicarse revisando su lista de Áreas de Servicio preferidas por orden desde el principio (excluyendo el área en conflicto), consultando a cada área si dispone de una plaza libre en la franja que tiene previsto parar, continuando con la siguiente de la lista mientras le respondan negativamente, hasta que encuentre una respuesta afirmativa. Es muy probable que varios vehículos consulten una única plaza, pero como sólo es una consulta y sólo uno se moverá, esto no supone un problema.
- El Área de servicio que está gestionando la reubicación recibirá las respuestas de los Gestores de Trayectos que le indicarán una de las siguientes respuestas:
    - Que no pueden reubicar su parada.
    - Que pueden reubicar su parada, junto con el tiempo de conducción ininterrumpida que perderían (expresada en minutos).
- Ordenando a los que han respondido que pueden reubicarse de menor a mayor pérdida de tiempo de conducción:
    - Se asignará al primero como el designado para reubicarse, al que se le indicará que realice la reserva en el Área de servicio consultada.
    - El Gestor de Trayectos realizará la nueva reserva, teniendo en este instante dos reservas activas, ya que a modo de seguridad no se le elimina la reserva inicial hasta que no ha confirmado la nueva.
    - Si era un vehículo que tenía una reserva previa:
        - Una vez confirma la nueva reserva, se cancela la reserva inicial.
        - Tras realizarse la cancelación, se concede la reserva al vehículo que originó la negociación.
    - Si era el vehículo que originó la negociación ya no es preciso hacer nada más.
    - Si el Gestor de Trayectos no consiguiera realizar la nueva reserva, como designado para reubicarse al siguiente de la lista. Así se hará hasta que alguno pueda reubicarse o la lista esté vacía (momento en el cual se indica al vehículo que ha originado el conflicto que es imposible que realice la reserva en el Área de Servicio).

PLATAFORMA: JADE

Se ha decidido utilizar la plataforma Jade [6] para el desarrollo del proyecto.

JADE (Java Agent DEvelopment Framework) es un entorno de programación completamente implementado en el lenguaje de programación Java, distribuido por Telecom Italia. Es un software de código abierto bajo licencia LGPL (Lesser General Public License Version 2).

Este entorno de programación simplifica la implementación de sistemas multiagente por medio de un software intermedio que cumple con las especificaciones FIPA, y que ofrece una serie de herramientas que facilitan la depuración y despliegue del sistema.

La plataforma de agentes puede implementarse de forma distribuida, con distintos sistemas operativos, y puede gestionarse la configuración por medio de una interfaz gráfica remota.

La configuración puede modificarse en tiempo de ejecución, incluso moviendo agentes de una máquina a otra si se requiere.

Algunos de los motivos para la elección de esta plataforma son:

- Los autores ya estaban familiarizados con esta plataforma.
- Es un entorno que continúa en proceso de ampliación y mejora, lo que da una mayor robustez, fiabilidad y continuidad a este proyecto y al trabajo futuro.
- Al estar basado en el lenguaje de programación Java cuyo uso está muy extendido el aprendizaje de la plataforma es más sencillo.
- Esta plataforma puede operar en un gran sistema operativo, incluidos sistemas para dispositivos móviles.
- La plataforma es de licencia abierta.
- Soporta arquitectura distribuida.
- Se adapta al estándar FIPA como lenguaje de comunicación de agentes.

La última versión de la plataforma, hasta el momento de confección de este documento, es JADE 4.3.2 y publicada en marzo del 2014. Para este trabajo estamos utilizando la versión 4.3.

JADE define dos lenguajes de contenidos predefinidos. En el paquete jade.content se incluyen dos códec para dos lenguajes de contenido, los tipos de lenguaje son los siguientes:

- Lenguaje SL: legible para las personas y codifica las expresiones como string (jade.content.lang.sl).
- Lenguaje LEAP: no legible y codifica en byte-encoded (jade.content.lang.leap).

Se ha optado por el lenguaje SL considerando que es mejor para la simulación porque es comprensible, y facilita la depuración del código al poder visualizar la información que se está transmitiendo.

PROTÉGÉ

La aplicación Protégé es un Editor de Ontologías para sistemas inteligentes gratuito, que ofrece una plataforma en la que modelar ontologías que pueden ser exportadas y utilizadas en los sistemas de agentes de Jade por medio de la extensión BeanGenerator.

Actualmente Protégé dispone de dos versiones de su aplicación:

- *WebProtégé*: Aplicación web para el modelado de ontologías, por lo que es posible el trabajo con ontologías sin necesidad de instalar software en local. Además dispone herramientas para la interacción entre los participantes del proyecto que la hacen muy interesante, con funcionalidades para trabajo colaborativo sobre la ontología, comunicación, comentarios, y revisión.
- *Protégé Desktop*: Es una aplicación de escritorio con una funcionalidad mayor, con soporte para el lenguaje OWL 2 Web de ontologías y posibilidad de conectar con otras aplicaciones como HermiT y Pellet.

Se evaluó al inicio del desarrollo de la ontología el uso de esta herramienta, pero se descartó porque parecía que la ontología sería bastante sencilla y parecía que llevaría un tiempo mayor modelarla en Protégé y tenerla que exportar.

Posteriormente se planteó que los inconvenientes de la creación manual de la ontología eran bastantes, sobre todo al requerirse total exactitud de términos y uso de mayúsculas entre la definición de los datos en la ontología y los nombres de las variables de los atributos de las clases, pero ya se había realizado la ontología casi al completo, por lo que se finalizó de forma manual.

DISEÑO DE INTERFAZ

AGENT.GUI

Se ha evaluado la herramienta Agent.GUI como una posibilidad para implementar la simulación.

Esta herramienta consiste en una plataforma desarrollada íntegramente en Java que ofrece una interfaz gráfica de usuario predefinida que cuyo código puede extenderse para añadir funcionalidades adicionales.

El objetivo que buscaba la creación de esta plataforma, es acercar el uso de los sistemas multiagente y las simulaciones de agentes a usuarios finales de otros dominios fuera de la informática, como por ejemplo ingenieros de otras ramas o economistas.

Se ha descartado su uso por los siguientes motivos:

- Dispone de poca documentación, los tutoriales avanzados no están disponibles aún.
- No parece permitir carga ni generación de datos masivos, es preciso definir agente por agente los datos que tendrán, lo que dificultaría una simulación de las dimensiones que se pretende, y no sirve de base para la idea que llevábamos de generación de agentes con datos aleatorios.

- Parecía que disponía de un sistema de visualización de los agentes, pero no se ha encontrado esta funcionalidad o información sobre la misma, puede que los ejemplos vistos se hayan realizado extendiendo la aplicación por código.

WINDOWBUILDER

Finalmente, se ha optado por el diseño desde cero de una interfaz más sencilla que nos permita configurar la simulación, y que con menor esfuerzo cubrirá lo necesario para la realización del proyecto.

Para ello se ha utilizado la extensión de Eclipse WindowBuilder, que ofrece un diseñador de interfaces gráficas para Java.

DESARROLLO

Para el desarrollo del sistema se ha intentado aplicar una metodología de desarrollo iterativo, empezando por lo más básico y ampliando la funcionalidad progresivamente hasta conseguir llegar a los objetivos planteados para este proyecto.

FASE 1

Para la fase 1 se ha implementado en JADE la base del sistema con los tres agentes y una funcionalidad inicial básica que se detalla a continuación.

AGENTES E INTERACCIONES:

- Agente Gestor de Trayectos
  - Para la primera fase simplificaremos asumiendo periodos de conducción de 4,5 horas y descansos de 1 hora, y ajustando los parámetros iniciales para evitar que más de un vehículo ocupen una plaza durante la franja deseada en una negociación.
  - Generar una lista de Áreas de Servicio deseadas.
  - Consulta la disponibilidad de plazas en la Área de servicio más propicia y solicita una reserva de plaza
  - Atiende solicitudes de reubicación, consultando un Área de Servicio de su lista de áreas deseadas.

- Agente Área de Servicio
  - Atiende solicitudes de disponibilidad.
  - Atiende solicitudes de reserva.

  - Deriva la reserva de plaza a la plaza disponible con mayor ocupación[1]
  - Inicia el proceso de negociación de reubicación si se solicita una reserva y todas las plazas están ocupadas en la franja horaria deseada.

- Agente Plaza:
  - Realiza la reserva de la plaza registrándola en su histórico de ocupación.
  - Informa de la ocupación de la plaza a la Área de servicio en determinado horario.
  - Informa del vehículo que tiene una reserva en determinado horario.
  - Gestiona cancelaciones de reservas.

ONTOLOGÍA FASE 1

Los elementos de la ontología incluidos para la fase 1 son los siguientes:

- Vehicles:
  - IDVehículo - IDVehicle
  - Velocidad Máxima – maxSpeed
- Datos conductor: Se ha optado por omitir la información de registros de tiempos y simplificarla registrando directamente los tiempos de conducción acumulados en los datos del agente Gestor de Trayectos.
  - Tiempo Conducción (minutos)– drivingTime
- Datos del Área de Servicio – RestArea
  - Nombre – Name
  - Carretera Principal – road
  - Kilometro – Position
  - Número de Plazas – Parking Space
- Histórico de ocupación de las plazas – HistoricalOccupancy: Todos los datos excepto Tipo Carga.

CARGA DE DATOS

Para la definición de los datos se han utilizado unos ficheros de texto separados por comas con los siguientes formatos:

- **Roads**: IDRoad, Name, Length (KM)
- **RestAreas**: IDRestArea, IDRoad,Pos (Km), Name, Parking Space
- **Vehicles**: IDVehicle, MaxSpeed

[1] Se ha decidido de este modo, ya que si se consigue ocupar un mayor número de huecos en la misma plaza, dejando otras plazas con mayor tiempo disponible, y por lo tanto más posibilidad de poder alojar otros vehículos.

Como se puede observar, por simplicidad se han introducido los datos de las carreteras directamente dentro de los ficheros RestAreas y Vehicles.

Para cargar los datos en el sistema y crear los agentes, se ha preparado un agente auxiliar llamado AgLoader cuyo objetivo es leer los ficheros y lanzar los agentes correspondientes a partir de los datos.

FASE 2

Durante el desarrollo del sistema, en el paso de la fase 1 a la fase 2 se han ido realizando las siguientes ampliaciones:

INTERFAZ

Se ha añadido una interfaz que permita configurar los parámetros de la simulación, y que permite:

- Elegir la ubicación donde leer los ficheros de configuración de los agentes.
- Decidir si se quiere realizar la carga de datos de los vehículos para la creación de los Gestores de Trayectos a partir de fichero, o si se desea que el sistema vaya creando agentes con datos aleatorios cada cierto intervalo de tiempo.
    - En caso de generar vehículos aleatorios, decidir si se quiere guardar un fichero con los datos de los vehículos creados.

AMPLIACIÓN DE LA ONTOLOGÍA

Conceptos:

- **Driver**: ID, name, drivingTime.
- **Road**: ID, name, traffic, lenght.
- **Vehicle**: id, maxSpeed, road, KM, driver.
- **RestArea**: id, name, spots, road, km.
- **Reservation**: JourneyManagerID, timeStart, timeEnd.

Acciones:

- **ReservationRequest**: Reservation.
- **OccupationRequest**: Reservation.
- **CancelRequest**: Reservation.
- **InformOccupation**: occupation (entero que indica la ocupación).
- **LogJourneyData**: drivedTime, lostTime, restTime, km, kmIn, kmOut, numberOfRelocations.
- **LogRelocation**: time, affectedVehicles, relocableVehicles, success, relocatedVehicleOrder, relocatedVehicleLostTime.

DESARROLLO DE LOS AGENTES

Se ha ido ampliando la funcionalidad de los agentes y se ha añadido el nuevo agente LogManager para registrar los resultados de la simulación.

En esta fase se realiza una simulación estática, realizando en el momento de la creación de cada Agente Gestor de trayectos una única reserva para su siguiente parada respecto a la posición y hora actuales.

El funcionamiento de los agentes con sus comportamientos en esta fase es el siguiente:

AGLOADER

El agente "Loader" muestra una interfaz de usuario y se encarga de generar los agentes necesarios para realizar la simulación. Este agente también puede generar dinámicamente agentes JourneyManager (configurable con la opción de interfaz).

Los comportamientos del agente "loader" son los siguientes:

- CreateJourneyManager --> Crea un vehículo con parámetros aleatorios.

- CreateJourneyManagerAgent --> Añade tantos comportamientos CreateJourneyManager como vehículos se quieran crear en cada iteración de la simulación (definido en interfaz)

AGPARKINGSPOT

Los agentes "plaza de parking" almacenan un registro de sus reservas. Estos agentes pueden recibir una solicitud de reserva (para realizar una reserva) o una solicitud de consulta (para saber si durante un momento determinado tienen o no tienen espacio disponible). Es importante tener en cuenta que sólo almacenamos reservas del "día actual" y del "día siguiente", se ha realizado de este modo para no almacenar una cantidad innecesaria de datos, y para agilizar la búsqueda de reservas.

Los comportamientos del agente "plaza de parking" son los siguientes:

- BParkingSpotMessageReceiver --> Este comportamiento queda a la espera de mensajes, dependiendo el tipo de mensaje que lleve (reserva, consulta de ocupación o cancelación de reserva), se lanza uno de los siguientes comportamientos.

- BParkingSpotManagerReservationRequest --> Comprueba si dispone de espacio para una solicitud determinada, si dispone de tiempo, responde al mensaje indicando su nivel de ocupación (es decir, cuantas horas estará ocupado el día en el que se está realizando la ocupación). La plaza de parking que tenga una ocupación mayor, "ganará" la negociación y se le asignará la reserva del

vehículo solicitante. Si la plaza no tuviera espacio disponible, rechazará la participación en la negociación.

- BParkingSpotManagerOccupationRequest --> Consulta si el agente plaza tiene hueco disponible en un determinado espacio de tiempo, tanto si lo tiene como si no, responde al REQUEST con un mensaje AGREE o REFUSE (dependiendo la disponibilidad).

- BParkingSpotManagerCancelRequest --> Cancela las reservas realizadas por un vehículo en la plaza de parking (nunca se debería dar el caso, pero en el caso de que un vehículo tenga varias reservas en una plaza de aparcamiento, se anularán todas).

AGRESTAREA

El agente "área de servicio" genera durante su fase "setup" tantos agentes "plaza de parking" como se indica en el fichero de configuración (este dato se le pasa al constructor del área de servicio). El área de servicio gestiona solicitudes de reservas (sólo se puede gestionar una reserva cada vez, hasta que no se termina de gestionar la reserva de un vehículo, el agente no puede gestionar una nueva reserva). También gestiona consultas de ocupación (es decir, para saber si el área tiene espacio disponible a una hora determinada), se pueden tener abiertas varias consultas de ocupación (pues varios vehículos pueden querer consultar simultáneamente la disponibilidad de plaza a la vez).

Si una reserva requiere de negociación por encontrarse todas las plazas ocupadas, el área de servicio iniciará y gestionará la negociación, iniciando el CFP a todos los vehículos involucrados para solicitar una reubicación y liberar la plaza requerida.

Los comportamientos del agente "Área de servicio" son:

- BRestAreaMessageReceiver --> Recibe mensajes de ocupación y crea tantos comportamientos de consulta de ocupación como sean requeridos.

- BRestAreaManagerOccupationRequest --> Consulta a sus plazas de aparcamiento si disponen de hueco disponible para una hora determinada. Responde al gestor de trayectos que le realizó la consulta con la respuesta.

- BRestAreaManagerReservationRequest --> Al recibir una petición de reserva de un gestor de trayectos, el área de servicio inicia un CFP con todas las plazas de las que dispone, solicitando una reserva (las plazas libres responden con su índice de ocupación para ese día y las ocupadas indican qué vehículos están impidiendo la reserva).
    - Si hay hueco disponible, el área de servicio selecciona la plaza de parking con mayor índice de ocupación y le manda un mensaje para que realice la reserva, se manda otro mensaje al vehículo indicando que la reserva se ha realizado.

    - Si ninguna plaza tiene hueco, se genera un comportamiento de reubicación (ver a continuación).

- BRestAreaManagerNegotiateForRelocation --> Este comportamiento recibe del comportamiento de reservas un listado con todos los gestores de trayectos involucrados en la negociación, y comienza un CFP en el que recibe respuesta de los implicados indicando el tiempo que "pierden" por realizar la reubicación (o un REFUSE por parte de aquellos que no puedan realizar el cambio), aquel vehículo que "pierda" menos tiempo por reubicarse, será escogido para reubicarse.
    - Cuando el gestor de trayectos confirme que ha podido reubicarse con éxito, se cancelará la reserva que tuviera este gestor de trayectos y se creará una reserva para el vehículo que solicitó plaza (a no ser que el que se reubique sea el vehículo que la solicitó).
    - Si por algún motivo el gestor de trayectos no pudiera realizar la reserva, se consulta al siguiente que menos tiempo "pierda" por la reubicación, y así secuencialmente. Si ningún vehículo pudiera reubicarse, la reserva no podría realizarse.

AGJOURNEYMANAGER

El agente "Gestor de Trayectos", correspondiente a un vehículo, realiza solicitudes de reserva al Área de servicio que más le interese.

Cada vez que se va a realizar una reserva, se genera una tabla de áreas de servicio a las que puede acceder; la que más le interesa será dentro del margen de tiempo de conducción restante hasta el máximo, aquella que esté a más distancia del vehículo (es decir, aquella en la que al detenerse haya podido realizar la mayor cantidad posible de tiempo de conducción sin realizar infracción alguna).

También dispone de un comportamiento para gestionar solicitudes de negociación por parte de las áreas de servicio, cada vez que le llega una de estas solicitudes, el vehículo consulta a todas las áreas de servicio a las que puede acceder (sin incluir la que ha iniciado la negociación) si tienen disponibilidad, si es así, responden al CFP con la diferencia en kilómetros del área de servicio en la que tenían reserva hasta la nueva área de servicio donde podrían ubicarse (esta distancia puede ser negativa, ya que un vehículo puede encontrar hueco en una área de servicio que antes no tuviera espacio libre).

Los comportamientos del agente vehículo son:

- BJourneyManagerReservation --> Añade un comportamiento en el agente que genera una tabla de áreas de servicios disponibles, una vez creada la tabla, intenta realizar una solicitud de reserva en la área de servicio más beneficiosa para el vehículo (aquella que esté a mayor distancia de su posición de las que puede acceder). Si no consigue realizar la reserva en dicha área de servicio, lo intentará con la siguiente (y así hasta quedarse sin áreas de servicio posible, en cuyo caso el agente mandará un log indicando el fracaso y saldrá del sistema).

- BJourneyManagerCreatingPreferencesTable --> Selecciona todas las áreas de servicio de la carretera en la que se encuentra el vehículo, selecciona aquellas a las que el vehículo puede acceder (es decir, aquellas que no estén "detrás" del vehículo y aquellas a las que puede llegar con el tiempo legal de conducción restante), en una tabla paralela, almacenamos la distancia que hay entre el vehículo y el área de servicio (para consultar posteriormente).

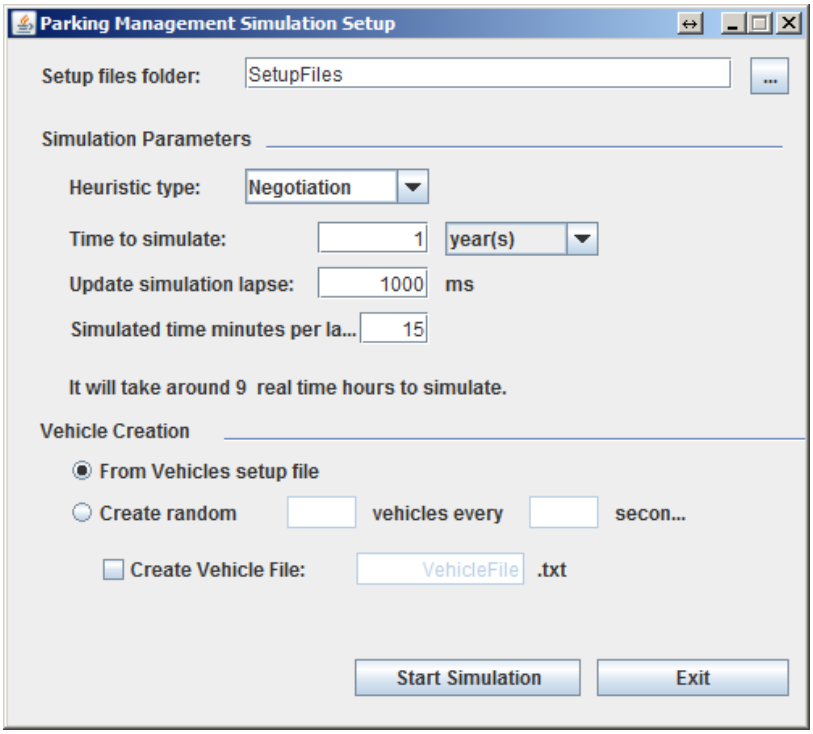
- BJourneyManagerNegotiation --> Cuando recibe un CFP del agente "área de servicio", consulta a las áreas de servicio de su tabla si tienen disponibilidad. Si la tienen, responde al CFP con el tiempo "que pierde" al realizar la reubicación. Cuando el área de servicio confirma que se reubique, realiza una solicitud de reserva, e informa al área de servicio que ha iniciado el CFP del éxito o fracaso de dicha solicitud.

FASE 3

Se ha añadido la gestión del tiempo a la simulación, de forma que la simulación avanzará la fecha y hora y se realizarán nuevas reservas una vez los vehículos se pongan de nuevo en trayecto.

MEJORAS DE INTERFAZ

Se ha mejorado la interfaz para incluir opciones que nos permitan controlar los parámetros de los tiempos de la simulación:



- **Setup files folder**: Permite seleccionar la carpeta de la que se leerán los ficheros de configuración de los agentes del sistema.

Simulation Parameters:

- **Heuristic type:** Permite seleccionar si se quiere lanzar la simulación:
  - Free Mode: En modo libre (los camiones irán siempre a última área de servicio posible dentro del tiempo de conducción permitido)
  - Negotiation: El modo de negociación aplica las heurísticas planteadas en este proyecto.

- **Time to simúlate**: Es el tiempo total que vamos a simular, es decir, el tiempo que consideraremos que transcurre durante nuestra simulación. Se puede seleccionar si queremos indicar este tiempo en minutos, horas, días o años.
  - **Update simulation lapse**: Es el lapso de actualización de la simulación, dónde se indica cada cuantos milisegundos se avanzará el tiempo en la simulación, de esta forma podemos "acelerar" el tiempo y simular largos periodos en menos tiempo.
  - **Simulated time minutes per lapse**: Es el número de minutos de tiempo de simulación que avanzaremos en cada lapso de actualización.

Vehicle Creation:

- **Método de creación:**
  - **From vehicles setup file:** Con este modo los vehículos se generarán al iniciar la simulación a partir de los datos del fichero Vehicles. Esto permite realizar pruebas pequeñas concretas y controladas al poder determinar los valores de los parámetros de los vehículos.
  - **Create Random:** Permite definir que se vayan generando datos de vehículos aleatorios y creando su correspondiente agente Gestor de Trayectos, pudiendo terminar una frecuencia de tiempo determinada y el número de vehículos que se generarán cada vez.
    - También es posible determinar si queremos que se almacenen los datos de estos vehículos aleatorios generados para luego poder realizar verificaciones.

GESTIÓN DEL TIEMPO

Se ha introducido la lógica de simulación para hacer que el tiempo avance y poder simular periodos de tiempo.

- El agente "Loader" se encarga de gestionar el "calendario" que se usa en la simulación (es decir, este agente hace avanzar el tiempo de simulación, en base a los parámetros recogidos de la interfaz e informa de ello a todos los agentes).

Se ha incluido el comportamiento UpdateTime --> Ticket Behaviour que aumenta el tiempo de la simulación cada X segundos en Y segundos (por ejemplo, podemos definir que cada X=1 segundo la simulación avance Y=1 hora). Esta información se manda a todos los agentes "vehículo" y "plaza de parking".

- El agente ParkingSpot, cuando avanza el día en el simulador, "día actual" recoge los datos de "día siguiente" y "día siguiente" se limpia. Para ello se ha incluido el comportamiento:

  BParkingSpotManagerUpdate --> Recoje del agente AgLoader el tiempo del sistema simulado en forma de mensaje. Si avanza el día, las reservas del día actual desaparecen y las del día siguiente se convierten en las del día actual.

- El agente Gestor de Trayectos (JourneyManager) ahora deberá considerar el tiempo transcurrido en cada lapso y actualizar sus datos en consecuencia, para ello, se ha añadido el comportamiento:

  BJourneyManagerUpdate --> Recibe las actualizaciones del tiempo del simulador a través del agente loader. Cada vez que avanza el tiempo, modifica las variables "kilómetro" (posición del vehículo en una carretera) y "tiempo de conducción" (tiempo que lleva conduciendo el conductor del vehículo) con la diferencia entre el anterior tiempo del simulador y el recibido (en minutos). También se actualiza la tabla de áreas de servicios deseadas (pues el vehículo puede haber "pasado de largo" alguna).

HEURÍSTICA MODO LIBRE

Se ha añadido el modo libre para comparar las estadísticas de la heurística implementada con un modo que intenta simular el comportamiento actual del transporte. En este modo se considerará que los vehículos pararán en el área de servicio que les interesa sin consulta previa, sin importarles si hay o no plazas. En caso de no haber plazas, se considerará una infracción.

FORMATO DEFINITIVO DE LOS FICHEROS

Los ficheros utilizados para la carga de datos son ficheros sin extensión, editados desde la misma plataforma eclipse (Si se editan desde Notepad de windows se producen problemas debido a la diferente codificación de los saltos de línea).

Los ficheros de Setup se componen de los siguientes datos, separados por comas:

Roads: IDRoad, Name, Length (KM), Traffic

RestAreas: IDRestArea, RoadID, Km, Name, Parking Space

Vehicles: IDVehicle, MaxSpeed, RoadID, Km, DriverID, DriverName, DriverDrivingTime (mins)

El formato de los ficheros de Log generados durante la simulación, es el siguiente:

RelocationsLog: Time, AffectedVehicles, RelocableVehicles, RelocatedVehicleOrder, Success

JourneyDataLog: IdVehicle, DrivedTime (mins), KM, KmIn, KMOut, lostTime, restTime, timeIn, timeOut, infraction (0/1)

PRUEBAS REALIZADAS

A continuación se detallarán algunas de las pruebas de simulación que se han realizado.

PRUEBAS BÁSICAS

Se formalizan una serie de pruebas básicas para validar que el sistema funciona de la forma esperada.

- Tiempo de simulación realizado: 1 día.
- Trabajamos siempre sobre la carretera:
  1,A7,1100, 1

PRUEBA 1 SIN CONFLICTOS

Condiciones:

- 2 camiones
  1,90,1,**180**,1,Ramona,120
  2,90,1,**170**,1,Pepe,100

- 1 área de servicio con 2 plazas
  1,1,**350**,A,**2**

MODO NEGOCIADO

Resultado esperado: Correcto

- No se producen infracciones (InfractionLog queda vacío)
- No se producen reubicaciones (RelocationsLog vacío)
- Datos de los vehículos:
  2,1465,374,170,544,50,60
  1,1485,197,180,377,45,60

MODO LIBRE

Resultado esperado: Correcto

- En modo libre no hay reubicaciones.
- No se producen infracciones, hay sitio suficiente.
- Datos de los vehículos:
  2,940,779,170,949,0,0
  1,960,602,180,782,0,0

PRUEBA 2: MODO NEGOCIADO, REUBICACIÓN FALLIDA E INFRACCIÓN

Condiciones:

- 3 camiones (Se añade un tercer camión)
  1,90,1,**180**,1,Ramona,120
  2,90,1,**170**,1,Pepe,100
  3,80,1,**150**,1,Juan,110

- 1 área de servicio con 2 plazas (Misma que en Prueba 1)
  1,1,**350**,A,**2**

MODO NEGOCIADO

Resultado esperado: Correcto

- El tercer camión en entrar al sistema generará una infracción, porque no queda plaza y no hay más estaciones de servicio disponibles.
  InfractionsLog: 2,Sat Jul 12 09:02:27 CEST 2014,170

En el momento de entrar al sistema e intentar realizar la reserva, ya se registra la infracción, porque no le es posible realizar una reserva.

- Se produce una reubicación fallida:
  RelocationsLog: Sat Jul 12 09:02:37 CEST 2014,2,0,0,false

Los dos coches han participado en la reubicación, pero ninguno podía reubicarse.

- Datos de los vehículos:
  2,100,148,170,318,**0,0**
  1,1485,158,180,338,45,60
  3,1475,876,150,1026,10,60

El último camión en entrar al sistema y solicitar la reserva ha sido el 2, por lo que no ha acumulado tiempo de conducción ni de descanso adicionales.

MODO LIBRE

Resultado esperado: Correcto

- En modo libre no hay reubicaciones.
- Se produce una infracción porque no hay lugar para los 3 vehículos: TBD
- Datos de los vehículos:
  2,565,654,170,824,0,0
  1,585,190,180,370,0,0
  3,575,263,150,413,0,0

PRUEBA 3: MODO NEGOCIADO, REUBICACIÓN CON ÉXITO

Condiciones:

- 3 camiones (Se añade un tercer camión)
  1,90,1,**180**,1,Ramona,120
  2,90,1,**170**,1,Pepe,100
  3,80,1,**150**,1,Juan,110

- 2 áreas de servicio con 2 y 1 plazas (Se añade un área de servicio respecto a la Prueba 2, en un kilometraje anterior a la existente, donde se deberá reubicar uno de los 3 vehículos.)
  1,1,350,A,2
  2,1,300,B,1

MODO NEGOCIADO

Resultado esperado: Correcto

- No se producen infracciones.

- Se produce una reubicación con éxito: Participan los 2 camiones en conflicto más el que ha realizado la solicitud.
  ReubicationsLog: Sat Jul 12 09:04:41 CEST 2014**,2,3,1**,**true**

- Datos de los vehículos:
  2,1465,158,170,328,50,60
  3,1490,523,150,673,-5,45
  1,1485,441,180,621,45,60

MODO LIBRE

Resultado esperado: Correcto

- En modo libre no hay reubicaciones.
- Se produce una infracción porque los 3 vehículos intentarán aparcar en la estación de servicio más lejana, y no hay lugar para los 3 vehículos.
  3,Mon Jul 14 00:09:33 CEST 2014,330

- Datos de los vehículos:
  3,135,849,150,999,135,0
  1,225,575,180,755,165,600
  2,210,425,170,595,165,600

PRUEBAS FINALES

Una vez validado el funcionamiento del sistema con las pruebas básicas, planteamos la ejecución de una simulación con mayor carga de datos que nos permita extraer información.

La idea inicial era poder realizar una simulación de 1 año, pero se ha detectado que no es factible debido a la carga en CPU y memoria que se producen a causa de la carga masiva de agentes y no es posible realizar simulaciones de tanta duración.

Se ha optado por realizar las simulaciones con las siguientes propiedades, reduciendo el problema para facilitar la interpretación de los datos finales, pero siendo suficientes como para ser representativos.

- Carreteras, 1 carretera: con 200 km de longitud.
- Áreas de servicio pre-generadas:
  - 10 áreas de servicio a lo largo de la carretera
  - Distancia entre áreas de 20 km.
  - Número de plazas fijo a 2.
- Vehículos generados aleatoriamente en cada ejecución de las pruebas:
  - 8 vehículos cada segundo (15 minutos de simulación).
  - En total son del orden de unos 800 vehículos.
- Parámetros de la simulación:
  - Duración: 1 día
  - Ciclos de 1000 ms
  - Aumento en cada ciclo: 15 minutos
- Se han realizado 5 ejecuciones para las pruebas en cada modo: Negociado y Libre.

En el siguiente apartado se detallan los resultados de las mismas.

RESULTADOS

Modo Negociado:

	RUN 1	RUN 2	RUN 3	RUN 4	RUN 5	MEDIAS
Nº Infracciones	23,00	31,00	30,00	35,00	33,00	**30,4**
Tiempo de conducción medio	141,26	138,28	141,64	139,98	144,68	**141,17**
Tiempo de conducción perdido medio (minutos)	9,78	8,99	9,06	9,09	8,73	**9,13**

Modo Libre:

	RUN 1	RUN 2	RUN 3	RUN 4	RUN 5	MEDIAS
Nº Infracciones	187,00	152,00	191,00	180,00	179,00	**177,80**
Tiempo de conducción medio	135,44	136,64	137,08	135,11	134,75	**135,80**
Tiempo de conducción perdido medio (minutos)	52,29	51,56	55,54	57,43	56,90	**54,74**

Se ha observado lo siguiente a partir del estudio de los resultados:

- El orden en el fichero de JourneyDataLog es relevante, ya que nos indica el orden de salida del sistema de los camiones, en especial en caso de infracciones.

Comparativa:

	NEGOCIADO	LIBRE
Nº Infracciones	**30,4**	**177,80**
Tiempo de conducción medio	**141,17**	**135,80**
Tiempo de conducción perdido medio	**9,13**	**54,74**

CONCLUSIONES

- Se ha visto recomendable el uso de herramientas de modelado de las ontologías (Como por ejemplo Protégé) para la definición de la ontología para reducir el tiempo de corrección de errores en la misma, incluso en ontologías sencillas.

- Se ha logrado el objetivo del trabajo, que era desarrollar un simulador que permita la gestión de las plazas de aparcamiento mediante distintas heurísticas, utilizando para ello una plataforma de agentes inteligentes realizada en Jade.

- En los resultados se puede apreciar que efectivamente se observa notablemente un menor número de infracciones en el modo negociado que intenta optimizar la ocupación de las plazas, ya que se fuerza a los vehículos a parar en estaciones de servicio anteriores en que haya espacio, y con esto se consigue evitar la congestión de las áreas de servicio.

- En cuanto al tiempo de conducción perdido (el tiempo restante que el conductor podría continuar, dentro del límite establecido legalmente, y que no aprovecha al parar con anterioridad en un área de servicio) se observa que en el modo libre se produce un aumento notable de este tiempo perdido, cuando se esperaba un tiempo similar.

- Se ha observado que intentando una simulación mucho más amplia que la utilizada en las pruebas la carga en CPU y en Memoria se vuelve insostenible.

- Por lo anterior se plantea que sería necesario revisar la extracción de datos y el uso de recursos físicos para poder realizar simulaciones más extensas que permitan un análisis más preciso de las heurísticas simuladas.

VISIÓN DE FUTURO

Una visión de hasta donde pensamos que puede llegar este proyecto:

Imaginemos la situación, un conductor tiene que salir a realizar un transporte en unos días desde Madrid a Paris, así que entra en su ordenador y solicita a su Agente Gestor de Trayectos la planificación de la ruta indicando sus preferencias para el viaje (Por ejemplo modo de máxima conducción ininterrumpida 4,5h continuas o parada cada 2h/2,5h], si quiere alojamiento, …).

Su agente consulta con el sistema de Google Maps la ruta. El agente recibe la ruta, que será de unas 18 horas según la velocidad máxima del vehículo de 90km / h. Según el histórico de trayectos realizados por el conductor durante la semana o semanas que abarcará el viaje, y del resto de rutas planificadas para esas semanas, podrá realizar [x horas de trayecto en X días, y necesitará X paradas]. El agente irá planificando la ruta, considerando para cada una de las paradas que debe realizar un tramo del trayecto en que el conductor debería parar para cumplir las restricciones de tiempos de conducción indicadas por la legislación, y consulta las Áreas de servicio existentes en esos tramos. Entre las disponibles, por orden de valoración (relación de confianza entre los agentes) solicitará reserva de plaza a cada Área hasta que acuerde con una de ellas la reserva de una plaza libre. En base a ello, seguirá planificando el resto del viaje. Si el agente Gestor de Trayectos considera que se debe hacer noche en algún punto del trayecto, se seleccionará para esa parada una Área de Servicio que disponga de Alojamiento. Se podrá también verificar con el agente del Hostal la disponibilidad de habitación para esa noche y el precio.

Una vez finalizado el proceso, le mostrará al Conductor el resultado indicándole el trayecto y las áreas de servicio donde se ha planificado parar y cuanto tiempo. El conductor podrá hacer alguna modificación si prefiere ir por otro camino, o parar en alguna Área de servicio diferente de las propuestas, con lo que el Gestor de Trayectos replanificará la ruta y negociará las paradas de nuevo, hasta que el Conductor le convenza el resultado y confirme la ruta. En ese momento el Gestor de Trayectos confirmará las reservas con las Áreas de servicio, e incluso con los Hostales si aplica, informando al conductor.

Cuando llega el día y el Conductor debe empezar el trayecto, el Gestor de Trayectos confirmará el inicio de la ruta (en caso contrario cancelaría todas las reservas), y durante la ruta también llevará un control del trayecto en conexión con la información de la ubicación por GPS y de la retroalimentación del conductor desde dispositivos móviles como su Smartphone o Tablet, de forma que si se produjese una parada no planificada, o un imprevisto como un atasco que retrasase el trayecto habría que replanificar de nuevo la ruta, cancelando las reservas que ya no se podrán realizar, negociando las nuevas reservas y notificando al conductor.

Cuando el conductor llegue a una Área de Servicio podría realizar un Check In manual, o podría confirmarse la ocupación de la plaza utilizando el sistema de cámaras de la Área de Servicio y un sistema OCR que reconociese la matrícula.

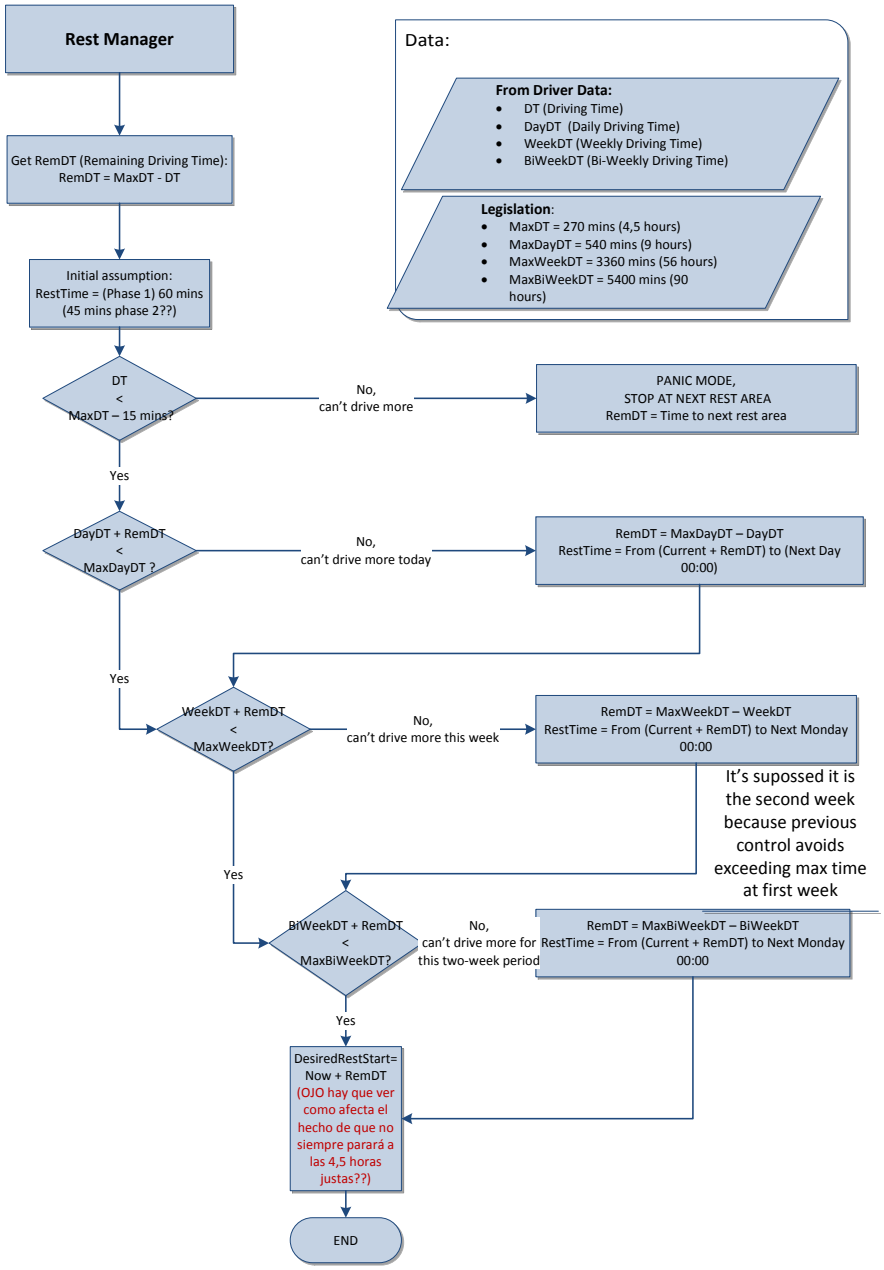
TRABAJO FUTURO

MEJORAS EN EL CÁLCULO DE TIEMPOS

En el sistema presentado, se consideran únicamente tiempos de conducción de 4,5 horas seguidos de 1 hora de descanso.

Para darle un mayor realismo al sistema, se podrían mejorar los cálculos de tiempos y la planificación de las paradas en base al cálculo de tiempos de conducción y descanso aplicando las restricciones.

Un posible algoritmo inicial para el cálculo de tiempos de conducción podría ser:



- Considerar las excepciones: Días que se pueden realizar 10 horas en vez de 9.

- Incluir limitaciones de tiempos de descanso.
- Considerar la posibilidad de tener varios conductores.
- Incluir cálculo de los tiempos de conducción a partir de registros de tiempo de los conductores. Esto podría plantearse como un enlace con el tacógrafo que deben llevar por ley los vehículos de transportes.

RESTRICCIONES PARA MERCANCÍAS PELIGROSAS

Entre el 10% y el 15% de las mercancías que se transportan por carretera en España son peligrosas.

Existen distintas legislaciones que afectan al transporte y aparcamiento de mercancías peligrosas, y que podrían considerarse en el sistema, por ejemplo:

- Existen limitaciones del paso de determinadas mercancías peligrosas por ciertas carreteras o en determinados horarios.
- Normalmente no se puede transportar mercancías peligrosas atravesando núcleos urbanos.
- Existen restricciones respecto al aparcamiento de mercancías peligrosas cerca de otros vehículos que transporten alimentos.
- Existen unas recomendaciones para el aparcamiento de mercancías peligrosas (vigilancia en el área de servicio, plazas con mayor separación…)

ENLACE CON SISTEMA DE CÁLCULO DE RUTAS

Para este proyecto se ha simplificado considerando la ruta por una única carretera, pero para darle mayor viabilidad lo ideal sería vincularlo con un sistema de cálculo de rutas que nos diese la ruta a seguir por diversas carreteras.

También sería interesante aprovechar que estos sistemas disponen de información sobre las áreas de servicio en dichas carreteras para realizar la búsqueda de las áreas de servicio preferidas.

Por ejemplo: Google maps, que además dispone de una API de conexión.

INFORMACIÓN DE CONGESTIÓN DE CARRETERA

En el proyecto actual se ha considerado que el Gestor de Trayectos conoce a priori toda la información de la carretera incluyendo el parámetro de congestión de la carretera (Road.traffic) que además se mantiene fijo.

Una posible ampliación sería convertir las carreteras en agentes que nos facilitasen información sobre la misma, y el parámetro de tráfico podrían conseguirlo consultando información real del tráfico de alguna fuente de datos externa (DGT, Google Maps…)

NUEVAS HEURÍSTICAS

- Cuando un vehículo no encuentra plaza, "ignora" al sistema y ocupa la plaza que mejor le convenga, cuando esto ocurra el área de servicio negociará con el vehículo al que le han ocupado la plaza (que será el que tuviera menos preferencia de todos los vehículos con reserva para dicha hora…¿o tal vez aquellos que apenas tendrán solapamiento en sus tiempos de descanso?)

MEJORAS DEL CÓDIGO

TIPOS DE DATOS

Revisar los tipos de datos: posiblemente haya que cambiar el tipo de datos utilizado en algunas variables, ya que se producen en casos aislados desbordamientos que dan valores incorrectos.

Por ejemplo:

- Los tiempos de conducción al estar en minutos pueden dar valores muy altos.
- En el cálculo de la estimación del tiempo de la simulación en la interfaz.

OPTIMIZACIÓN DEL USO DE LA MEMORIA

Debido a las dimensiones de las simulaciones que se plantean poder llegar a realizar con este simulación y vistas las limitaciones, sería interesante profundizar en la mejora del uso de la memoria para

AGRADECIMIENTOS

Agradecemos a los profesores Vicent J. Botti Navarro de la Universidad Politécnica de Valencia y a Luis Amable García Fernández de la Universidad Jaume I por su apoyo, sin el cual no habríamos podido realizar este proyecto.

GLOSARIO

GLOSARIO DEL DOMINIO DEL PROBLEMA

CONDUCTOR / DRIVER:

Toda persona que conduzca el vehículo, incluso durante un corto período, o que esté a bordo de un vehículo como parte de sus obligaciones para conducirlo en caso de necesidad.

DESCANSO / REST PERIOD:

Cualquier período ininterrumpido durante el cual un conductor pueda disponer libremente de su tiempo.

PAUSA:

Cualquier período durante el cual un conductor no pueda llevar a cabo ninguna actividad de conducción u otro trabajo y que sirva exclusivamente para su reposo. Es decir, las pausas son las paradas cortas que deben realizarse entre tiempos de conducción y no superar el tiempo máximo de conducción ininterrumpida.

PERÍODO DE DESCANSO DIARIO / DAILY REST TIME:

El período diario durante el cual un conductor puede disponer libremente de su tiempo, ya sea un «período de descanso diario normal» o un «período de descanso diario reducido»:

- «Período de descanso diario normal»: cualquier período de descanso de al menos 11 horas. Alternativamente, el período de descanso diario normal se podrá tomar en dos períodos, el primero de ellos de al menos tres horas ininterrumpidas y el segundo de al menos 9 horas ininterrumpidas.
- «Período de descanso diario reducido»: cualquier período de descanso de al menos 9 horas, pero inferior a 11 horas.

SEMANA / WEEK:

El período de tiempo comprendido entre las 00.00 del lunes y las 24.00 del domingo.

PERÍODO DE DESCANSO SEMANAL / WEEKLY REST TIME:

El período semanal durante el cual un conductor puede disponer libremente de su tiempo, ya sea un «período de descanso semanal normal» o un «período de descanso semanal reducido»:

- «Período de descanso semanal normal»: cualquier período de descanso de al menos 45 horas,
- «Período de descanso semanal reducido»: cualquier período de descanso inferior a 45 horas que, sujeto a las condiciones establecidas en el artículo 8, apartado 6, se puede reducir hasta un mínimo de 24 horas consecutivas.

TIEMPO DE CONDUCCIÓN / DRIVING TIME:

El tiempo que dura la actividad de conducción registrada o planificada.

TIEMPO DIARIO DE CONDUCCIÓN / DAILY DRIVING PERIOD:

El tiempo acumulado total de conducción entre el final de un período de descanso diario y el principio del siguiente período de descanso diario o entre un período de descanso diario y un período de descanso semanal.

TIEMPO SEMANAL DE CONDUCCIÓN / TOTAL WEEKLY DRIVING TIME:

El tiempo acumulado total de conducción durante una semana.

MASA MÁXIMA AUTORIZADA / MAXIMUM AUTHORIZED MASS:

La masa máxima admisible del vehículo dispuesto para la marcha, incluida la carga útil.

RUTA / ROUTE:

Es un conjunto de carreteras determinado que permite llegar de un origen a un destino.

TRAYECTO / JOURNEY:

Es la planificación del viaje que va a realizar el conductor, incluyendo toda la información de tiempos de conducción y descanso previstos, y las áreas de servicio en que se prevén las paradas.

TACÓGRAFO / TACHOGRAPH:

Un tacógrafo es un dispositivo electrónico que registra diversos sucesos originados en los vehículos de transporte terrestre durante su conducción. Los sucesos que registra un tacógrafo normalmente son: velocidad (promedio y máxima), RPM, kilómetros recorridos, aceleraciones y frenadas bruscas, tiempo de ralentí (periodo durante el cual el vehículo permanece detenido con el motor en marcha), entre otros. Estos datos, dependiendo del tacógrafo, pueden ser recopilados por una computadora y almacenados en una base de datos o imprimirse en forma de gráficos para su posterior análisis.

Por medio de estos dispositivos se registran los tiempos de parada y descanso efectuados por los conductores, sirviendo como dispositivo de control requerido por la legislación.

BIBLIOGRAFÍA

[1] European Commission. Directive 2002/15/EC of the European parliament and of the council of 11 March 2002 on the organization of the working time of persons performing mobile road transport activities. 2012.

[2] European Commission. Directive 2010/40/EU of the european parliament and of the council of 7 July 2010 on the framework for the deployment of Intelligent Transport Systems in the field of road transport and for interfaces with other modes of transport. 2010.

[3] Capdevila, M.; Tomas, V.R.; Garca, L.A.; Prades Farron, M., "Dynamic Management of Parking Spaces in Road Rest Areas with Automatic Negotiation," Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on , vol., no., pp.3609,3614, 13-16 Oct. 2013

[4] Wooldridge, M. (1995) This is MYWORLD: the logic of an agent-oriented testbed for DAI. In Intelligent Agents: Theories, Architectures and Languages (eds M. Wooldridge and N. R. Jennings), LNAI Volume 890, pp. 1 60-1 78- Springer, Berlin.

[5] Ministerio de Fomento, *Acuerdo Europeo relativo al Transporte Internacional de Mercancías Peligrosas por Carretera,* 2011, http://www.fomento.gob.es/AZ.BBMF.Web/documentacion/pdf/R16959.pdf [Consulta: domingo, 29 de junio del 2014]

[6] F. Bellifemine, G. Caire, A. Poggi, G. Rimassa, *JADE A White Paper*, 2003, http://jade.cselt.it/papers/2003/WhitePaperJADEEXP.pdf [Consulta: domingo, 29 de junio del 2014]

[7] Consejo de la Unión Europea, *Directiva 94/55/CE del Consejo, de 21 de noviembre de 1994, sobre la aproximación de las legislaciones de los Estados miembros con respecto al transporte de mercancías peligrosas por carretera*, 1996, http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31994L0055:ES:HTML [Consulta: 29 de junio del 2014]

ANEXO I – CÓDIGO FUENTE DEL PROYECTO

ONTOLOGÍA

CANCELREQUEST

```
package ontology;

import jade.content.AgentAction;

public class CancelRequest implements AgentAction {

	private Reservation reservation;

	public Reservation getReservation() {
		return reservation;
	}

	public void setReservation(Reservation reservation) {
		this.reservation = reservation;
	}

}
```

DRIVER

```
package ontology;

import jade.content.Concept;

/**
 * @author Raquel García Arévalo
 * @code: ontology - Driver Concept
 *
 */

public class Driver implements Concept {

	private int driverId;
	private String name;
	private int drivingTime;

	public int getDriverId() {
		return driverId;
	}

	public void setDriverId(int id) {
		this.driverId = id;
	}

	public String getName() {
		return name;
	}
```

```
	public void setName(String name) {
		this.name = name;
	}

	public int getDrivingTime() {
		return drivingTime;
	}

	public void setDrivingTime(int drivingTime) {
		this.drivingTime = drivingTime;
	}
}
```

LOGJOURNEYDATA

```
package ontology;

import java.util.Date;

import jade.content.AgentAction;

/**
 * @author Raquel García Arévalo
 * @code: ontology - Reubication Data to be Logged
 *
 */

public class LogJourneyData implements AgentAction
{
	private int idVehicle;
	private int drivedTime;
	private int lostTime;
	private int restTime;
	private int km;
	private int kmIn;
	private int kmOut;
	private Date TimeIn;
	private Date TimeOut;
	private int infractionType; //0: No infraction, 1: Unable to park


	public int getdrivedTime() {
		return drivedTime;
	}

	public void setdrivedTime(int drivedTime) {
		this.drivedTime = drivedTime;
	}
```

```
public int getlostTime() {
	return lostTime;
}

public void setlostTime(int lostTime) {
	this.lostTime = lostTime;
}

public int getrestTime() {
	return restTime;
}

public void setrestTime(int restTime) {
	this.restTime = restTime;
}

public int getKm() {
	return km;
}

public void setKm(int Km) {
	this.km = Km;
}

public int getKmIn() {
	return kmIn;
}

public void setKmIn(int kmIn) {
	this.kmIn = kmIn;
}

public int getKmOut() {
	return kmOut;
}

public void setKmOut(int kmOut) {
	this.kmOut = kmOut;
}

public int getIdVehicle() {
	return idVehicle;
}

public void setIdVehicle(int idVehicle) {
	this.idVehicle = idVehicle;
}

public int getInfractionType() {
```

```
		return infractionType;
	}

	public void setInfractionType(int infractionType) {
		this.infractionType = infractionType;
	}

	public Date getTimeIn() {
		return TimeIn;
	}

	public void setTimeIn(Date timeIn) {
		TimeIn = timeIn;
	}

	public Date getTimeOut() {
		return TimeOut;
	}

	public void setTimeOut(Date timeOut) {
		TimeOut = timeOut;
	}
}
```

LOGRELOCATION

```
package ontology;

import jade.content.AgentAction;

import java.util.Date;

/**
 * @author Raquel García Arévalo
 * @code: ontology - Reubication Data to be Logged
 *
 */

public class LogRelocation implements AgentAction {

	private Date relocationTime;
	private int affectedVehicles;
	private int relocableVehicles;
	private boolean success;
	private int relocatedVehicleOrder;
	private Float relocatedVehicleLostTime;


	public Date getRelocationTime() {
```

```
		return relocationTime;
	}

	public void setRelocationTime(Date relocationTime) {
		this.relocationTime = relocationTime;
	}

	public int getAffectedVehicles() {
		return affectedVehicles;
	}

	public void setAffectedVehicles(int affectedVehicles) {
		this.affectedVehicles = affectedVehicles;
	}

	public int getRelocableVehicles() {
		return relocableVehicles;
	}

	public void setRelocableVehicles(int relocableVehicles) {
		this.relocableVehicles = relocableVehicles;
	}

	public boolean getSuccess() {
		return success;
	}

	public void setSuccess(boolean success) {
		this.success = success;
	}

	public int getRelocatedVehicleOrder() {
		return relocatedVehicleOrder;
	}

	public void setRelocatedVehicleOrder(int relocatedVehicleOrder) {
		this.relocatedVehicleOrder = relocatedVehicleOrder;
	}

	public Float getRelocatedVehicleLostTime() {
		return relocatedVehicleLostTime;
	}

	public void setRelocatedVehicleLostTime(Float relocatedVehicleLostTime) {
		this.relocatedVehicleLostTime = relocatedVehicleLostTime;
	}


}
```

OCCUPATIONREQUEST

```
package ontology;

import jade.content.AgentAction;

/**
 * @author Raquel García Arévalo
 * @code: ontology - Occupation Request Action
 *
 */

public class OccupationRequest implements AgentAction {

	private Reservation reservation;

	public Reservation getReservation() {
		return reservation;
	}

	public void setReservation(Reservation reservation) {
		this.reservation = reservation;
	}
}
```

PARKINGMANAGEMENTONTOLOGY

```
package ontology;

import jade.content.onto.BasicOntology;
import jade.content.onto.Ontology;
import jade.content.onto.OntologyException;
import jade.content.schema.AgentActionSchema;
import jade.content.schema.ConceptSchema;
import jade.content.schema.PrimitiveSchema;

import java.sql.Driver;

/**
 * @author Raquel García Arévalo
 * @code: ontology definition
 *
 */

public class ParkingManagementOntology extends Ontology {

	public static final String ONTOLOGY_NAME = "Parking Management Ontology";

	// Vocabulary
	// Road
	public static final String ROAD = "Road";
	public static final String ROAD_ID = "roadId";
```

```
public static final String ROAD_NAME = "name";
public static final String ROAD_TRAFICC = "traffic";
public static final String ROAD_LENGHT = "lenght";

// Driver
public static final String DRIVER = "Driver";
public static final String DRIVER_ID = "driverId";
public static final String DRIVER_NAME = "name";
public static final String DRIVER_DRIVING_TIME = "drivingTime";

// Vehicle
public static final String VEHICLE = "Vehicle";
public static final String VEHICLE_ID = "vehicleId";
public static final String VEHICLE_MAXSPEED = "maxSpeed";
public static final String VEHICLE_ROAD = "road";
public static final String VEHICLE_KM = "KM";
public static final String VEHICLE_DRIVER = "driver";
// Potential improvement: Consider multiple drivers

// RestArea
public static final String REST_AREA = "Rest Area";
public static final String REST_AREA_ID = "restAreaId";
public static final String REST_AREA_NAME = "name";
public static final String REST_AREA_SPOTS = "areaSpots";
public static final String REST_AREA_ROAD = "road";
public static final String REST_AREA_KM = "KM";

// Reservation
public static final String RESERVATION = "Reservation";
public static final String RESERVATION_TIME_START = "timeStart";
public static final String RESERVATION_TIME_END = "timeEnd";
public static final String RESERVATION_JOURNEY_MANAGER_ID = "journeyManagerId";

// Reservation Request
public static final String RESERVATION_REQUEST = "ReservationRequest";
public static final String RESERVATION_REQUEST_RESERVATION = "reservation";

// Occupation request
public static final String OCCUPATION_REQUEST = "OccupationRequest";
public static final String OCCUPATION_REQUEST_RESERVATION = "reservation";

// Cancel request
public static final String CANCEL_REQUEST = "CancelRequest";
public static final String CANCEL_REQUEST_RESERVATION = "reservation";

//Log Journey Data
public static final String LOG_JOURNEY_DATA = "LogJourneyData";
public static final String LOG_JOURNEY_ID_VEHICLE = "idVehicle";
public static final String LOG_JOURNEY_DATA_DRIVED_TIME = "drivedTime";
public static final String LOG_JOURNEY_DATA_LOST_TIME = "lostTime";
```

```
	public static final String LOG_JOURNEY_DATA_REST_TIME = "restTime";
	public static final String LOG_JOURNEY_DATA_KM = "km";
	public static final String LOG_JOURNEY_DATA_KM_IN = "kmIn";
	public static final String LOG_JOURNEY_DATA_KM_OUT = "kmOut";
	public static final String LOG_JOURNEY_DATA_TIME_IN = "timeIn";
	public static final String LOG_JOURNEY_DATA_TIME_OUT = "timeOut";
	public static final String LOG_JOURNEY_INFRACTION_TYPE = "infractionType";

	//Log Reubication
	public static final String LOG_RELOCATION = "LogRelocation";
	public static final String LOG_RELOCATION_RELOCATION_TIME = "relocationTime";
	public static final String LOG_RELOCATION_AFFECTED_VEHICLES = "affectedVehicles";
	public static final String LOG_RELOCATION_RELOCABLE_VEHICLES= "relocableVehicles";
	public static final String LOG_RELOCATION_SUCCES = "success";
	public static final String LOG_RELOCATION_RELOCATED_VEHICLE_ORDER =
"relocatedVehicleOrder";
	public static final String LOG_RELOCATION_RELOCATED_VEHICLE_LOST_TIME =
"relocatedVehicleLostTime";

	//Update Time
	public static final String UPDATE_TIME = "UpdateTime";
	public static final String UPDATE_TIME_TIME_UPDATED = "timeUpdated";
	public static final String UPDATE_TIME_MINUTES_UPDATED = "minutesUpdated";


	// Ontology Instance
	private static final Ontology instance = new ParkingManagementOntology();

	// Function to access the ontology
	public static Ontology getInstance() {
		return instance;
	}

	// Constructor
	private ParkingManagementOntology() {
		// ParkingManagementOntology extends basic Ontology
		super(ONTOLOGY_NAME, BasicOntology.getInstance());

		try {

			// Add the elements
			add(new ConceptSchema(DRIVER), Driver.class);
			add(new ConceptSchema(ROAD), Road.class);
			add(new ConceptSchema(VEHICLE), Vehicle.class);
			add(new ConceptSchema(REST_AREA), Vehicle.class);
			add(new ConceptSchema(RESERVATION), Reservation.class);
			add(new
AgentActionSchema(RESERVATION_REQUEST),ReservationRequest.class);
			add(new AgentActionSchema(OCCUPATION_REQUEST),OccupationRequest.class);
			add(new AgentActionSchema(CANCEL_REQUEST), CancelRequest.class);
```

```
                    add(new AgentActionSchema(LOG_JOURNEY_DATA), LogJourneyData.class);
                    add(new AgentActionSchema(LOG_RELOCATION), LogRelocation.class);
                    add(new AgentActionSchema(UPDATE_TIME), UpdateTime.class);


                    // Road Scheme
                    ConceptSchema cs = (ConceptSchema) getSchema(ROAD);
                    cs.add(ROAD_ID, (PrimitiveSchema) getSchema(BasicOntology.INTEGER));
                    cs.add(ROAD_NAME, (PrimitiveSchema) getSchema(BasicOntology.STRING));
                    cs.add(ROAD_TRAFICC,(PrimitiveSchema) getSchema(BasicOntology.FLOAT));
                    cs.add(ROAD_LENGHT,(PrimitiveSchema) getSchema(BasicOntology.INTEGER));

                    // Driver Scheme
                    cs = (ConceptSchema) getSchema(DRIVER);
                    cs.add(DRIVER_ID,(PrimitiveSchema) getSchema(BasicOntology.INTEGER));
                    cs.add(DRIVER_NAME,(PrimitiveSchema) getSchema(BasicOntology.STRING));
                    cs.add(DRIVER_DRIVING_TIME,(PrimitiveSchema)
getSchema(BasicOntology.INTEGER));

                    // Vehicle Scheme
                    cs = (ConceptSchema) getSchema(VEHICLE);
                    cs.add(VEHICLE_ID,(PrimitiveSchema) getSchema(BasicOntology.INTEGER));
                    cs.add(VEHICLE_MAXSPEED,(PrimitiveSchema)
getSchema(BasicOntology.INTEGER));
                    cs.add(VEHICLE_ROAD, (ConceptSchema) getSchema(ROAD));
                    cs.add(VEHICLE_KM,(PrimitiveSchema) getSchema(BasicOntology.INTEGER));
                    cs.add(VEHICLE_DRIVER, (ConceptSchema) getSchema(DRIVER));

                    // Rest Area Scheme
                    cs = (ConceptSchema) getSchema(REST_AREA);
                    cs.add(REST_AREA_ID,(PrimitiveSchema) getSchema(BasicOntology.INTEGER));
                    cs.add(REST_AREA_NAME,(PrimitiveSchema) getSchema(BasicOntology.STRING));
                    cs.add(REST_AREA_SPOTS,(PrimitiveSchema)
getSchema(BasicOntology.INTEGER));
                    cs.add(REST_AREA_ROAD, (ConceptSchema) getSchema(ROAD));
                    cs.add(REST_AREA_KM,(PrimitiveSchema) getSchema(BasicOntology.INTEGER));

                    // Reservation Scheme
                    cs = (ConceptSchema) getSchema(RESERVATION);
                    cs.add(RESERVATION_JOURNEY_MANAGER_ID,(PrimitiveSchema)
getSchema(BasicOntology.STRING));
                    cs.add(RESERVATION_TIME_START,(PrimitiveSchema)
getSchema(BasicOntology.DATE));
                    cs.add(RESERVATION_TIME_END,(PrimitiveSchema)
getSchema(BasicOntology.DATE));

                    // Reservation Action
                    AgentActionSchema as = (AgentActionSchema)
getSchema(RESERVATION_REQUEST);
```

```
                        as.add(RESERVATION_REQUEST_RESERVATION,(ConceptSchema)
getSchema(RESERVATION));

                        // Occupation Action
                        as = (AgentActionSchema) getSchema(OCCUPATION_REQUEST);
                        as.add(OCCUPATION_REQUEST_RESERVATION,(ConceptSchema)
getSchema(RESERVATION));

                        // Cancel Action
                        as = (AgentActionSchema) getSchema(CANCEL_REQUEST);
                        as.add(CANCEL_REQUEST_RESERVATION,(ConceptSchema)
getSchema(RESERVATION));


                        //Log Journey Data
                        as = (AgentActionSchema) getSchema(LOG_JOURNEY_DATA);
                        as.add(LOG_JOURNEY_DATA_DRIVED_TIME,(PrimitiveSchema)
getSchema(BasicOntology.INTEGER));
                        as.add(LOG_JOURNEY_ID_VEHICLE,(PrimitiveSchema)
getSchema(BasicOntology.INTEGER));
                        as.add(LOG_JOURNEY_DATA_LOST_TIME,(PrimitiveSchema)
getSchema(BasicOntology.INTEGER));
                        as.add(LOG_JOURNEY_DATA_REST_TIME,(PrimitiveSchema)
getSchema(BasicOntology.INTEGER));
                        as.add(LOG_JOURNEY_DATA_KM,(PrimitiveSchema)
getSchema(BasicOntology.INTEGER));
                        as.add(LOG_JOURNEY_DATA_KM_IN,(PrimitiveSchema)
getSchema(BasicOntology.INTEGER));
                        as.add(LOG_JOURNEY_DATA_KM_OUT,(PrimitiveSchema)
getSchema(BasicOntology.INTEGER));
                        as.add(LOG_JOURNEY_DATA_TIME_IN,(PrimitiveSchema)
getSchema(BasicOntology.DATE));
                        as.add(LOG_JOURNEY_DATA_TIME_OUT,(PrimitiveSchema)
getSchema(BasicOntology.DATE));
                        as.add(LOG_JOURNEY_INFRACTION_TYPE,(PrimitiveSchema)
getSchema(BasicOntology.INTEGER));


                        //Log Relocation
                        as = (AgentActionSchema) getSchema(LOG_RELOCATION);
                        as.add(LOG_RELOCATION_RELOCATION_TIME,(PrimitiveSchema)
getSchema(BasicOntology.DATE));
                        as.add(LOG_RELOCATION_AFFECTED_VEHICLES,(PrimitiveSchema)
getSchema(BasicOntology.INTEGER));
                        as.add(LOG_RELOCATION_RELOCABLE_VEHICLES,(PrimitiveSchema)
getSchema(BasicOntology.INTEGER));
                        as.add(LOG_RELOCATION_SUCCES,(PrimitiveSchema)
getSchema(BasicOntology.BOOLEAN));
                        as.add(LOG_RELOCATION_RELOCATED_VEHICLE_ORDER,(PrimitiveSchema)
getSchema(BasicOntology.INTEGER));
```

```
                        as.add(LOG_RELOCATION_RELOCATED_VEHICLE_LOST_TIME,(PrimitiveSchema)
getSchema(BasicOntology.FLOAT));

                        //Update Time
                        as = (AgentActionSchema) getSchema(UPDATE_TIME);
                        as.add(UPDATE_TIME_TIME_UPDATED,(PrimitiveSchema)
getSchema(BasicOntology.DATE));
                        as.add(UPDATE_TIME_MINUTES_UPDATED,(PrimitiveSchema)
getSchema(BasicOntology.INTEGER));



                } catch (OntologyException oe) {
                        oe.printStackTrace();
                }

        }

}
```

RESERVATION

```
package ontology;

import jade.content.Concept;

import java.util.Date;

public class Reservation implements Concept {

        private Date timeStart, timeEnd;
        private String journeyManagerId;

        public Date getTimeStart() {
                return timeStart;
        }

        public void setTimeStart(Date timeStart) {
                this.timeStart = timeStart;
        }

        public Date getTimeEnd() {
                return timeEnd;
        }

        public void setTimeEnd(Date timeEnd) {
                this.timeEnd = timeEnd;
        }

        public String getjourneyManagerId() {
```

```
		return journeyManagerId;
	}

	public void setjourneyManagerId(String journeyManagerId) {
		this.journeyManagerId = journeyManagerId;
	}

}
```

RESERVATIONREQUEST

```
package ontology;

import jade.content.AgentAction;

/**
 * @author Raquel García Arévalo
 * @code: ontology - Reservation Request Action
 *
 */

public class ReservationRequest implements AgentAction {

	private Reservation reservation;

	public Reservation getReservation() {
		return reservation;
	}

	public void setReservation(Reservation reservation) {
		this.reservation = reservation;
	}

}
```

RESTAREA

```
package ontology;
import jade.content.Concept;

/**
 * @author Raquel García Arévalo
 * @code: ontology - Rest Area Concept
 *
 */
public class RestArea implements Concept {
	private int restAreaId;
	private int areaSpots;
	private Road road;
	private int KM;
	private String name;

	public int getRestAreaId() {
		return restAreaId;
	}

	public void setRestAreaId(int id) {
		this.restAreaId = id;
	}

	public int getAreaSpots() {
		return areaSpots;
	}

	public void setAreaSpots(int areaSpots) {
		this.areaSpots = areaSpots;
	}

	public Road getRoad() {
		return road;
	}

	public void setRoad(Road road) {
		this.road = road;
	}

	public int getKM() {
		return KM;
	}

	public void setKM(int kM) {
		KM = kM;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

}
```

ROAD

```
package ontology;

import jade.content.Concept;

/**
 * @author Raquel García Arévalo
 * @code: ontology - Road Concept
 *
 */

public class Road implements Concept {
	private String name;
	private int roadId;
	private float traffic;
	private int lenght;

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public int getRoadId() {
		return roadId;
	}

	public void setRoadId(int id) {
		this.roadId = id;
	}

	public float getTraffic() {
		return traffic;
	}

	public void setTraffic(float traffic) {
		this.traffic = traffic;
	}

	public int getLenght() {
		return lenght;
	}

	public void setLenght(int lenght) {
		this.lenght = lenght;
	}

}
```

UPDATETIME

```
package ontology;

import java.util.Date;

import jade.content.AgentAction;

public class UpdateTime implements AgentAction {

	private Date timeUpdated;
	private int minutesUpdated;

	public Date getTimeUpdated() {
		return timeUpdated;
	}

	public void setTimeUpdated(Date timeUpdated) {
		this.timeUpdated = timeUpdated;
	}

	public int getMinutesUpdated() {
		return minutesUpdated;
	}

	public void setMinutesUpdated(int minutesUpdated) {
		this.minutesUpdated = minutesUpdated;
	}

}
```

VEHICLE

```
package ontology;

import jade.content.Concept;

/**
 * @author Raquel García Arévalo
 * @code: ontology - Vehicle Concept
 **/
public class Vehicle implements Concept {
	private int vehicleId;
	private int maxSpeed;
	private Road road;
	private int KM;
	private Driver driver;

	public int getVehicleId() {
		return vehicleId;
	}

	public void setVehicleId(int id) {
		vehicleId = id;
	}

	public int getMaxSpeed() {
		return maxSpeed;
	}

	public void setMaxSpeed(int maxSpeed) {
		this.maxSpeed = maxSpeed;
	}

	public Road getRoad() {
		return road;
	}

	public void setRoad(Road road) {
		this.road = road;
	}

	public int getKM() {
		return KM;
	}

	public void setKM(int kM) {
		KM = kM;
	}

	public Driver getDriver() {
		return driver;
	}

	public void setDriver(Driver driver) {
		this.driver = driver;
	}

}
```

GUI

GUIDATA

```
package gui;

/**
 * @author Raquel García Arévalo
 * @code: GUI Data Auxiliar class
 *
 */

public class GUIData {

	public String i_setupFilesFolder;

	public int i_simulationTime;
	public int i_simulationUpdate;
	public int i_simMinutesPerLapse;
	public int i_simulationType; //0: Negotiation, 1: free mode

	public int i_numberOfVehicles;
	public int i_vehicleCreationLapse;

	public Boolean i_createRandomVehicles = false;
	public String i_vehicleLogName;
	public Boolean i_CreateLog;


}
```

LOADERGUI

```
package gui;

import jade.gui.GuiEvent;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JRadioButton;
import javax.swing.JSeparator;
import javax.swing.JTextField;
import javax.swing.SwingConstants;

import agents.AgLoader;

import javax.swing.JComboBox;
import javax.swing.DefaultComboBoxModel;

import com.ibm.icu.impl.duration.TimeUnit;

import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.util.Date;

import javax.swing.JProgressBar;

/**
 * @author Raquel García Arévalo
 * @code: Loader Graphic User Interface
 *
 */

public class LoaderGUI extends JFrame implements ActionListener, KeyListener {

	protected GUIData result;
	private agents.AgLoader myAgent;

	private JTextField setupFilesFolder;
```

```
private JLabel lblSetupFolder;
private JButton btnSelectFile;

private JSeparator sepSimulation;
private JLabel lblRealTimeMinutes;
private JLabel lblSimulationTime;
private JLabel lblRealTimeTo;
private JLabel lblUpdateSimulationEach;
private JLabel lblVehicles;
private JLabel lblSeconds;
private JLabel lblHeuristicType;
private JLabel lblSeconds_1;
private JLabel lbltxt;
private JSeparator sepVehicles;
private JLabel lblVehicleCreation;

private JComboBox simulationType;
private JTextField simulationTime;
private JComboBox simTimeUnit;
private JTextField simulationUpdate;
private JTextField minutesPerLapse;
private JLabel lbEstimation;

private JTextField numberOfVehicles;
private JTextField vehicleCreationLapse;

private JRadioButton rdbtnFromRandomData, rdbtnFromFile;
private JCheckBox chckbxCreateLog;
private JTextField txLogName;

private JButton btnStart, btnExit;

private JProgressBar progressBar;
private JLabel lblStatus;

public void startSimulation() {
        simulationType.setVisible(false);
        setupFilesFolder.setVisible(false);
        simulationUpdate.setVisible(false);
        minutesPerLapse.setVisible(false);
        simulationTime.setVisible(false);
        simTimeUnit.setVisible(false);
        numberOfVehicles.setVisible(false);
        vehicleCreationLapse.setVisible(false);
        rdbtnFromFile.setVisible(false);
        rdbtnFromRandomData.setVisible(false);
        chckbxCreateLog.setVisible(false);
        txLogName.setVisible(false);
        lblSeconds_1.setVisible(false);
        lbltxt.setVisible(false);
```

```
                sepVehicles.setVisible(false);
                lblVehicleCreation.setVisible(false);

                lblSetupFolder.setVisible(false);
                setupFilesFolder.setVisible(false);

                btnSelectFile.setVisible(false);

                sepSimulation.setVisible(false);
                lblRealTimeMinutes.setVisible(false);
                lblSimulationTime.setVisible(false);
                lblRealTimeTo.setVisible(false);
                lblUpdateSimulationEach.setVisible(false);
                lblVehicles.setVisible(false);
                lblSeconds.setVisible(false);
                lblHeuristicType.setVisible(false);

                btnStart.setVisible(false);

                getContentPane().add(progressBar);
                getContentPane().add(lblStatus);

        }

        public LoaderGUI(AgLoader loader) {
                setBackground(Color.LIGHT_GRAY);
                setForeground(Color.BLACK);

                myAgent = loader;
                result = new GUIData();

                setTitle("Parking Management Simulation Setup");
                getContentPane().setLayout(null);
                setResizable(false);
                setSize(new Dimension(500, 450));

                // SELECT SETUP FILES FOLDER
                lblSetupFolder = new JLabel("Setup files folder:");
                lblSetupFolder.setToolTipText("Agent data will be loaded from files at this location. The
name and fields for the files are explained at documentation.");
                lblSetupFolder.setBounds(20, 15, 124, 14);
                getContentPane().add(lblSetupFolder);

                setupFilesFolder = new JTextField();
                setupFilesFolder.setToolTipText("Agent data will be loaded from files at this location. The
name and fields for the files are explained at documentation.");
                setupFilesFolder.setText("SetupFiles");
                setupFilesFolder.setBounds(147, 11, 302, 20);
                getContentPane().add(setupFilesFolder);
                setupFilesFolder.setColumns(10);
```

```
btnSelectFile = new JButton("...");
btnSelectFile.addMouseListener(new MouseAdapter() {
	@Override
	public void mouseClicked(MouseEvent arg0) {

		JFileChooser j = new JFileChooser();
		j.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
		Integer option = j.showOpenDialog(null);

		if (option == JFileChooser.APPROVE_OPTION) {
			setupFilesFolder.setText(j.getSelectedFile().toString());

		}

	}
});
btnSelectFile.addActionListener(new ActionListener() {
	@Override
	public void actionPerformed(ActionEvent arg0) {
	}
});
btnSelectFile.setBounds(460, 11, 24, 23);
getContentPane().add(btnSelectFile);

// SIMULATION SETUP

lblSimulationTime = new JLabel("Simulation Parameters");
lblSimulationTime.setBounds(20, 54, 144, 14);
getContentPane().add(lblSimulationTime);
simulationTime = new JTextField();


simulationTime.setHorizontalAlignment(SwingConstants.RIGHT);
simulationTime.setText("1");
simulationTime.setBounds(192, 113, 69, 20);
getContentPane().add(simulationTime);
simulationTime.setColumns(10);

simulationTime.addKeyListener(this);

sepSimulation = new JSeparator();
sepSimulation.setBounds(134, 247, 360, 2);
getContentPane().add(sepSimulation);

lblRealTimeMinutes = new JLabel("Simulated minutes per lapse:");
lblRealTimeMinutes.setBounds(38, 172, 180, 14);
getContentPane().add(lblRealTimeMinutes);

lblRealTimeTo = new JLabel("Time to simulate:");
```

```
lblRealTimeTo.setBounds(38, 113, 152, 20);
getContentPane().add(lblRealTimeTo);

simulationUpdate = new JTextField();
simulationUpdate.setHorizontalAlignment(SwingConstants.RIGHT);
simulationUpdate.setText("1000");
simulationUpdate.setBounds(192, 141, 69, 20);
getContentPane().add(simulationUpdate);
simulationUpdate.setColumns(10);

simulationUpdate.addKeyListener(this);

lblUpdateSimulationEach = new JLabel("Update simulation lapse:");
lblUpdateSimulationEach.setBounds(38, 144, 152, 14);
getContentPane().add(lblUpdateSimulationEach);



minutesPerLapse = new JTextField();
minutesPerLapse.setHorizontalAlignment(SwingConstants.RIGHT);
minutesPerLapse.setText("15");
minutesPerLapse.setBounds(218, 169, 43, 20);
getContentPane().add(minutesPerLapse);
minutesPerLapse.setColumns(10);

minutesPerLapse.addKeyListener(this);

lblSeconds_1 = new JLabel("ms");
lblSeconds_1.setBounds(271, 141, 43, 20);
getContentPane().add(lblSeconds_1);

lbltxt = new JLabel(".txt");
lbltxt.setBounds(328, 321, 29, 14);
getContentPane().add(lbltxt);

// VEHICLES CREATION SETUP

sepVehicles = new JSeparator();
sepVehicles.setBounds(160, 66, 320, 2);
getContentPane().add(sepVehicles);

lblVehicleCreation = new JLabel("Vehicle Creation");
lblVehicleCreation.setBounds(20, 235, 104, 14);
getContentPane().add(lblVehicleCreation);

rdbtnFromFile = new JRadioButton("From Vehicles setup file");
rdbtnFromFile.setSelected(true);
rdbtnFromFile.setBounds(36, 256, 386, 23);
getContentPane().add(rdbtnFromFile);
rdbtnFromFile.addActionListener(this);
```

```
rdbtnFromRandomData = new JRadioButton("Create random");
rdbtnFromRandomData.setBounds(36, 282, 128, 23);
getContentPane().add(rdbtnFromRandomData);
rdbtnFromRandomData.addActionListener(this);

ButtonGroup bG = new ButtonGroup();
bG.add(rdbtnFromFile);
bG.add(rdbtnFromRandomData);

numberOfVehicles = new JTextField();
numberOfVehicles.setHorizontalAlignment(SwingConstants.RIGHT);
numberOfVehicles.setEnabled(false);
numberOfVehicles.setBounds(173, 283, 43, 20);
getContentPane().add(numberOfVehicles);
numberOfVehicles.setColumns(10);

lblVehicles = new JLabel("vehicles every");
lblVehicles.setBounds(226, 286, 94, 14);
getContentPane().add(lblVehicles);

vehicleCreationLapse = new JTextField();
vehicleCreationLapse.setHorizontalAlignment(SwingConstants.RIGHT);
vehicleCreationLapse.setEnabled(false);
vehicleCreationLapse.setBounds(323, 283, 43, 20);
getContentPane().add(vehicleCreationLapse);
vehicleCreationLapse.setColumns(10);

lblSeconds = new JLabel("seconds");
lblSeconds.setBounds(376, 286, 100, 14);
getContentPane().add(lblSeconds);

chckbxCreateLog = new JCheckBox("Create Vehicle File:");
chckbxCreateLog.setBounds(55, 317, 144, 23);
chckbxCreateLog.setEnabled(false);
getContentPane().add(chckbxCreateLog);
chckbxCreateLog.addActionListener(this);

txLogName = new JTextField();
txLogName.setText("VehicleFile");
txLogName.setHorizontalAlignment(SwingConstants.RIGHT);
txLogName.setEnabled(false);
txLogName.setBounds(216, 318, 104, 20);
txLogName.setEnabled(false);
getContentPane().add(txLogName);
txLogName.setColumns(10);

lblHeuristicType = new JLabel("Heuristic type:");
lblHeuristicType.setBounds(38, 84, 106, 14);
getContentPane().add(lblHeuristicType);
```

```
		simulationType = new JComboBox();
		simulationType.setModel(new DefaultComboBoxModel(new String[] {"Negotiation", "Free
Mode"}));
		simulationType.setSelectedIndex(0);
		simulationType.setEditable(true);
		simulationType.setBounds(147, 80, 114, 22);
		getContentPane().add(simulationType);


		simTimeUnit = new JComboBox();
		simTimeUnit.setModel(new DefaultComboBoxModel(new String[] {"minute(s)", "hour(s)",
"day(s)", "year(s)"}));
		simTimeUnit.setSelectedIndex(3);
		simTimeUnit.setBounds(271, 113, 95, 21);
		getContentPane().add(simTimeUnit);
		simTimeUnit.addActionListener(this);


		//Calculate how much time it will take to simulate

		lbEstimation = new JLabel("");
		lbEstimation.setBounds(38, 208, 446, 14);
		getContentPane().add(lbEstimation);
		updateEstimation();



		// BUTTONS
		// Start Simulation
		btnStart = new JButton("Start Simulation");
		btnStart.setBounds(140, 384, 160, 25);
		getContentPane().add(btnStart);
		btnStart.addActionListener(this);

		// Exit Simulation
		btnExit = new JButton("Exit");
		btnExit.setBounds(320, 384, 160, 25);
		getContentPane().add(btnExit);

		btnExit.addActionListener(this);


	}

	public void setStatusProgress(int value, Date date)
	{
		progressBar.setValue(value);
		lblStatus.setText("Simulation in progress. Simulation Time: "+date.toString());
	}
```

```
private int simulationMinutes()
{
	int minutes = 0;
	int totalTime = 0;
	try {
		totalTime = Integer.parseInt(simulationTime.getText());
	}
	catch (NumberFormatException nfe) {
		totalTime = 0;
	}

	switch(simTimeUnit.getSelectedIndex())
	{
	//minutes
	case (0):
		minutes = totalTime;
		break;

	//hours
	case(1):
		minutes = totalTime * 60;
		break;

	//days
	case(2):
		minutes = totalTime * 24 * 60;
		break;

	//years
	case(3):
		minutes = totalTime * 24 * 60 * 365;
		break;

	}

	return minutes;
}


private void updateEstimation() {
	int estimation = 0;
	int update = Integer.parseInt(simulationUpdate.getText());
	int lapseTime = Integer.parseInt(minutesPerLapse.getText());

	try {
	//Calculate estimated simulation time
	estimation =  (update * simulationMinutes()) / lapseTime;

	//estimation now is ms, convert to minutes
```

```
                estimation = (estimation / 1000) / 60;

                if (estimation < 0)
                        lbEstimation.setText("It will take less than a minute to simulate.");
                if (estimation < 60)
                        lbEstimation.setText("It will take around " + String.valueOf(estimation) + "  real
time minutes to simulate.");
                else if (estimation >= 60)
                        lbEstimation.setText("It will take around " + String.valueOf(estimation/60) + "  real
time hours to simulate.");

                }
                catch (Error e)
                {
                        lbEstimation.setText("It wasn't possible to estimate how long the simulation will
take.");
                }

        }

        public void setSetupdata() {
                result.i_setupFilesFolder = setupFilesFolder.getText();

                result.i_simulationType = simulationType.getSelectedIndex();

                result.i_simulationTime = simulationMinutes();

                try {
                        result.i_simMinutesPerLapse = Integer.parseInt(minutesPerLapse
                                        .getText());

                } catch (NumberFormatException nfe) {
                        result.i_simMinutesPerLapse = 1;
                }

                try {
                        result.i_simulationUpdate = Integer.parseInt(simulationUpdate
                                        .getText());

                } catch (NumberFormatException nfe) {
                        result.i_simulationUpdate = 1000;
                }
                try {
                        result.i_numberOfVehicles = Integer.parseInt(numberOfVehicles
                                        .getText());

                } catch (NumberFormatException nfe) {
                        result.i_numberOfVehicles = 0;
                }
```

```
		try {
			result.i_vehicleCreationLapse = Integer
					.parseInt(vehicleCreationLapse.getText());

		} catch (NumberFormatException nfe) {
			result.i_vehicleCreationLapse = 1;
		}

		result.i_vehicleLogName = txLogName.getText();

		result.i_CreateLog = chckbxCreateLog.isSelected();

		if (rdbtnFromRandomData.isSelected())
			result.i_createRandomVehicles = true;
	}

	@Override
	public void actionPerformed(ActionEvent ae) {
		if (ae.getSource() == btnExit) {
			GuiEvent ge = new GuiEvent(this, 0);
			myAgent.postGuiEvent(ge);

			System.out.println("Closing...");
			System.exit(0);
		} else if (ae.getSource() == rdbtnFromRandomData
				|| ae.getSource() == rdbtnFromFile) {

			vehicleCreationLapse.setEnabled(rdbtnFromRandomData.isSelected());
			numberOfVehicles.setEnabled(rdbtnFromRandomData.isSelected());
			chckbxCreateLog.setEnabled(rdbtnFromRandomData.isSelected());
			txLogName.setEnabled(chckbxCreateLog.isSelected()
					&& chckbxCreateLog.isEnabled());

		} else if (ae.getSource() == chckbxCreateLog) {
			txLogName.setEnabled(chckbxCreateLog.isSelected());
		} else if (ae.getSource() == btnStart) {
			progressBar = new JProgressBar();
			progressBar.setBounds(20, 54, 464, 31);
			progressBar.setMaximum(simulationMinutes());


			lblStatus = new JLabel("Simulation in progress.");
			lblStatus.setBounds(20, 15, 464, 14);

			startSimulation();
			setSetupdata();
			btnExit.setText("Stop Simulation");

			GuiEvent ge = new GuiEvent(this, 1);
			ge.addParameter(result);
```

```
                        myAgent.postGuiEvent(ge);
                } else if (ae.getSource() == simTimeUnit)
                {
                        updateEstimation();
                }



        }

        @Override
         public void keyTyped(KeyEvent evt)
         {
                 if (evt.getSource() == simulationTime || evt.getSource() == simulationUpdate ||
evt.getSource() == minutesPerLapse)
                 {
                        if (simulationTime.getText() == null && simulationTime.getText() == "")
                                lbEstimation.setText("");
                        else

                                updateEstimation();



                 }
         }

        @Override
        public void keyPressed(KeyEvent arg0) {
                // TODO Auto-generated method stub

        }

        @Override
        public void keyReleased(KeyEvent arg0) {

                updateEstimation();
        }
}
```

AGENTS

AGJOURNEYDATA

```
package agents;

import java.util.Calendar;
import java.util.ArrayList;
import java.util.Random;
import java.util.UUID;

import ontology.Driver;
import ontology.LogJourneyData;
import ontology.OccupationRequest;
import ontology.ParkingManagementOntology;
import ontology.Reservation;
import ontology.ReservationRequest;
import ontology.Road;
import ontology.UpdateTime;
import ontology.Vehicle;
import jade.content.ContentElement;
import jade.content.lang.Codec;
import jade.content.lang.Codec.CodecException;
import jade.content.lang.sl.SLCodec;
import jade.content.onto.Ontology;
import jade.content.onto.OntologyException;
import jade.content.onto.basic.Action;
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.Behaviour;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

/**
 * @author Francisco Palau Romero
 * @code: Journey Manager agent code
 *
 */

public class AgJourneyManager extends Agent {
	//Vehicle parameters
	private Road road;
	private int km, km_in, km_out; //Kms
	private int drivingSpeed = 90; //Km/h
	private String journeyManagerId;
	private int vehicleId;
	private int restTime = 60; //Minutes
	private int maximunDriveTime = 270; //Minutes
	private int drivedTime = 0; //Minutes
	private int drivedTimeBeforeRest = 0; //Minutes
	private int lostTime = 0; //Minutes
	private int restedTime = 0; //Minutes
	//Driver parameters
```

```
	private Driver driver;
	//General parameters
	private AID ID;
	private Calendar actualTime = Calendar.getInstance();
	private Calendar estimatedArrivalTime = Calendar.getInstance();
	private Calendar estimatedEndingTime = Calendar.getInstance();
	private int infractionCommited = 0;
	private Calendar timeIn = Calendar.getInstance();
	private Calendar timeOut = Calendar.getInstance();
	//Simulation type
	private int simulationType;
	private static final int BORDA_NEGOTIATION_SIMULATION_TYPE = 0;
	private static final int FREE_SIMULATION_TYPE = 1;
	//Desired rest areas list
	private ArrayList<AID> RestAreaList = new ArrayList<AID>();
	private ArrayList<Float> RestAreaDistanceList = new ArrayList<Float>();
	private ArrayList<Float> RestAreaKMList = new ArrayList<Float>();
	boolean listCompleted = false;
	boolean creatingList = false;
	private AID nextRestArea;
	private boolean reachRestArea = false; //For free reservation mode only
	private boolean resting = false;
	//Ontology
	private Codec codec = new SLCodec();
	private Ontology ontoloy = ParkingManagementOntology.getInstance();
	private Action action;
	//Log AID
	private AID logManagerAID;

	//*************************************************************************
*************//
	//*************************************************************************
*************//
	// Setup
	                                                                    //
	//*************************************************************************
*************//
	//*************************************************************************
*************//
	protected void setup() {
		//Register codec and ontology
	    getContentManager().registerLanguage(codec);
	    getContentManager().registerOntology(ontoloy);
	    //

	    //Extract parameters from Loader
	      Object[] args = getArguments();
	      Vehicle vehicleInfo = (Vehicle) args[0];

	      road = vehicleInfo.getRoad();
	      km = vehicleInfo.getKM();
	      km_in = vehicleInfo.getKM();
	      //Calculating the km out (value between the km in which the truck is and
the maximum length of the road)
	      int km_max = road.getLenght();
	      Random myRandomizer = new Random();
	      km_out = myRandomizer.nextInt((km_max - km) + 1) + km;
```

```
            //
            drivingSpeed = vehicleInfo.getMaxSpeed();
            vehicleId = vehicleInfo.getVehicleId();
            journeyManagerId = getLocalName();
            driver = vehicleInfo.getDriver();
            drivedTime = vehicleInfo.getDriver().getDrivingTime();
            drivedTimeBeforeRest = drivedTime;

            simulationType = (Integer) args[1];
            actualTime = (Calendar) args[2];
            timeIn.setTime(actualTime.getTime());
            //


            //Register the offered service
            DFAgentDescription dfd = new DFAgentDescription();
            ID = getAID();
            dfd.setName(ID);
            ServiceDescription sd = new ServiceDescription();
            sd.setType("JourneyManager");
            sd.setName("truck"+vehicleId);
            dfd.addServices(sd);
            try{
                  DFService.register(this, dfd);
            } catch (Exception e) {e.printStackTrace();}
            //

            //Get the AIDs of the log
            DFAgentDescription[] result = null;
            ServiceDescription sdLog = new ServiceDescription();
            DFAgentDescription dfdLog = new DFAgentDescription();

            sdLog.setType("LogManager");
            sdLog.setName("logManager");
            dfdLog.addServices(sdLog);

            while (result == null) {
                try {
                  //Search for 100 ms
                  result = DFService.searchUntilFound(this, getDefaultDF(), dfdLog,
null, 100);
                  } catch (FIPAException e) {
                        e.printStackTrace();
                        System.out.println("FIPA Error while searching for agents");
                  }
            }
          if (result.length >= 1) {
            for (DFAgentDescription logManagerResult : result){
                  logManagerAID = logManagerResult.getName();
            }
          }
          //

          //The type of simulation it will depend of simulationType
          switch (simulationType) {
                case FREE_SIMULATION_TYPE:
                        addBehaviour(new BJourneyManagerReservationFree());
```

```
                        addBehaviour(new BJourneyManagerUpdateFree());
                    break;
                case BORDA_NEGOTIATION_SIMULATION_TYPE:
                        addBehaviour(new BJourneyManagerReservation());
                        addBehaviour(new BJourneyManagerNegotiation());
                        addBehaviour(new BJourneyManagerUpdate());
                    break;
            }
            //

              //Shut down behaviour
              addBehaviour(new BJourneyManagerShutDown());
              //
        }
        //**************************************************************************
*************//
        //**************************************************************************
*************//


        //**************************************************************************
*************//
        //**************************************************************************
*************//
        // takeDown: Sending logs
                                        //
        //**************************************************************************
*************//
        //**************************************************************************
*************//
        @Override
        protected void takeDown(){
              //TimeOut
              timeOut.setTime(actualTime.getTime());
              //Send the log with the journey data to logManager
              ACLMessage msgLogJourneyData = new ACLMessage(ACLMessage.INFORM);
              msgLogJourneyData.setLanguage(codec.getName());
              msgLogJourneyData.setOntology(ontoloy.getName());
              LogJourneyData logJourneyData = new LogJourneyData();
              logJourneyData.setIdVehicle(vehicleId);
              logJourneyData.setdrivedTime(drivedTime);
              logJourneyData.setKm(km_out - km_in);
              logJourneyData.setKmIn(km_in);
              logJourneyData.setKmOut(km_out);
              logJourneyData.setlostTime(lostTime);
              logJourneyData.setrestTime(restedTime);
              logJourneyData.setTimeIn(timeIn.getTime());
              logJourneyData.setTimeOut(timeOut.getTime());
              logJourneyData.setInfractionType(infractionCommited);
              //The action must be "stored" in an standard "Action" object, instead of
a "custom" AgentAction
              action = new Action(getAID(),logJourneyData);
              try {
                    getContentManager().fillContent(msgLogJourneyData, action);
              } catch (CodecException | OntologyException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
```

```
			}
			msgLogJourneyData.setConversationId("log");
			msgLogJourneyData.addReceiver(logManagerAID);
			send(msgLogJourneyData);
			//

			while (receive() != null) {};
		}
		//**************************************************************************
*************//
		//**************************************************************************
*************//


		//**************************************************************************
*************//
		//**************************************************************************
*************//
		// Borda Negotiation Behavious: BJourneyManagerReservation,
BJourneyManagerNegotiation,		//
		//                                                    BJourneyUpdate
                                                                //
		//**************************************************************************
*************//
		//**************************************************************************
*************//
		private class BJourneyManagerReservation extends Behaviour {

			int step = 0;
			MessageTemplate mtRequestReply;
			AID bestRestArea;
			String reservationCode;
			int bestIndex = 0;
			int index = 0;
			float bestDistance;
			ArrayList<AID> newRestAreaList = new ArrayList<AID>();
			ArrayList<Float> newRestAreaDistanceList = new ArrayList<Float>();

			@Override
			public void action() {
				switch (step){
				//Creating the Preference List
				case 0:
					if (!creatingList) {
						addBehaviour(new
BJourneyManagerCreatingPreferencesTable());
						creatingList = true;
					}
					if (listCompleted) {
						step++;
						creatingList = false;
						newRestAreaList.addAll(RestAreaList);
						newRestAreaDistanceList.addAll(RestAreaDistanceList);
					}
					break;
				//Choosing the best rest area to park (it is the rest area that
allows the truck drive the maximun distance
```

```
                    case 1:
                          //If the journey manager runs out of options, it must exit
of the system
                          if (newRestAreaList.isEmpty()){
                                infractionCommited = 1;
                                myAgent.doDelete();
                                step = 3;
                                break;
                          }
                          bestIndex = 0;
                          index = 0;
                          bestDistance = 0;
                          //The best will be the one most far away from my position
(the one which is value is greater)
                          for (float distance : newRestAreaDistanceList){
                                if (distance > bestDistance) {
                                      bestIndex = index;
                                      bestDistance = distance;
                                }
                                index++;
                          }
                          bestRestArea = newRestAreaList.get(bestIndex);
                          //For the moment, we will assume that we got a reservation
there, just in case there is a negotiation
                          nextRestArea = bestRestArea;
                          //

                          ACLMessage msgRequestParkingSpot = new
ACLMessage(ACLMessage.REQUEST);
                          reservationCode = UUID.randomUUID().toString();

                          msgRequestParkingSpot.addReceiver(bestRestArea);
                          msgRequestParkingSpot.setConversationId("request");

                          //Creating the reservation request message using ontology
                          msgRequestParkingSpot.setLanguage(codec.getName());
                          msgRequestParkingSpot.setOntology(ontoloy.getName());
                          Reservation reservation = new Reservation();
                          Calendar timeStart, timeEnd;
                          timeStart = Calendar.getInstance();
                          timeEnd = Calendar.getInstance();
                          timeStart.add(Calendar.MINUTE,(int)
(bestDistance*60/drivingSpeed));
                          estimatedArrivalTime.setTime(timeStart.getTime());
                          timeEnd.add(Calendar.MINUTE, (int)
(bestDistance*60/drivingSpeed)+restTime);
                          estimatedEndingTime.setTime(timeEnd.getTime());
                          reservation.setTimeStart(timeStart.getTime());
                          reservation.setTimeEnd(timeEnd.getTime());
                          reservation.setjourneyManagerId(journeyManagerId);
                          ReservationRequest reservationRequest = new
ReservationRequest();
                          reservationRequest.setReservation(reservation);
                          //The action must be "stored" in an standard "Action"
object, instead of a "custom" AgentAction
                          action = new Action(getAID(),reservationRequest);
                          try {
```

```
                                    getContentManager().fillContent(msgRequestParkingSpot,
action);
                              } catch (CodecException | OntologyException e) {
                                    // TODO Auto-generated catch block
                                    e.printStackTrace();
                              }
                              msgRequestParkingSpot.setInReplyTo(reservationCode);

                              //Send the reservation
                              myAgent.send(msgRequestParkingSpot);
                              step++;
                              break;
                        case 2:

                              //Waiting for response
                              mtRequestReply =
MessageTemplate.MatchInReplyTo(reservationCode);

                              ACLMessage msgCFPReply = myAgent.receive(mtRequestReply);

                              if (msgCFPReply != null) {

                                    //If everything is OK, the behavior ends
                                    if (msgCFPReply.getPerformative()==ACLMessage.AGREE) {
                                          //The Journey Manager stored the reservation
start/end and target rest area, so this behaviour ends
                                          step++;
                                          break;
                                    }
                                    //If there was a problem, the next best rest area will
be selected
                                    if (msgCFPReply.getPerformative()==ACLMessage.REFUSE)
{
                                          nextRestArea = null;
                                          newRestAreaList.remove(bestIndex);
                                          newRestAreaDistanceList.remove(bestIndex);
                                          step--;
                                          break;
                                    }
                              } else block();
                              break;
                        }
                  }

                  @Override
                  public boolean done() {
                        if (step == 3) return true;
                        return false;
                  }
            }



            //This behavior begins with an CFP message, and the journey manager must offer
another option (rest area)
            private class BJourneyManagerNegotiation extends Behaviour {
                  int step = 0;
```

```
		MessageTemplate mtCFP = 
MessageTemplate.MatchPerformative(ACLMessage.CFP);
		MessageTemplate mtCFPAccept;
		String replyCode;
		AID restAreaManager;
		int index, bestIndex;
		String reservationCode;
		ArrayList<AID> newRestAreaList = new ArrayList<AID>();
		ArrayList<Float> newRestAreaDistanceList = new ArrayList<Float>();
		ArrayList<Float> newRestAreaKMList = new ArrayList<Float>();
		String ocupationCode;
		Reservation reservation;
		OccupationRequest occupationRequest;
		ReservationRequest reservationRequest;

		@Override
		public void action() {
			switch (step){
			case 0:
				//We recieve a calling for proposal
				ACLMessage msg = myAgent.receive(mtCFP);
				if (msg!=null){
					replyCode = msg.getInReplyTo();
					restAreaManager = msg.getSender();
					newRestAreaList.addAll(RestAreaList);
					newRestAreaDistanceList.addAll(RestAreaDistanceList);
					newRestAreaKMList.addAll(RestAreaKMList);

					step++;
				} else block();
				break;
			case 1:
				//If the vehicle is 15 minutes away from the source of
negotiation, the negotiation ends
				float distance = 
newRestAreaDistanceList.get(newRestAreaList.indexOf(restAreaManager));
				float timeUntilArrival = (distance/drivingSpeed)*60;
//Minutes until the vehicle arrives to the rest area
				if (timeUntilArrival <= 15) {
					ACLMessage msgCFPRefuse = new 
ACLMessage(ACLMessage.REFUSE);
					msgCFPRefuse.addReceiver(restAreaManager);
					msgCFPRefuse.setInReplyTo(replyCode);
					myAgent.send(msgCFPRefuse);
					step = 6;
					break;
				}
				step++;
				break;
			case 2:
				//Is the list is empty, refuse the negotiation
				if (newRestAreaList.isEmpty()) {
					ACLMessage msgCFPRefuse = new 
ACLMessage(ACLMessage.REFUSE);
					msgCFPRefuse.addReceiver(restAreaManager);
					msgCFPRefuse.setInReplyTo(replyCode);
					myAgent.send(msgCFPRefuse);
```

```
                                step = 6;
                                break;
                        }
                        //Searching for the best rest area (besides the one which
initiate the negotiation)
                        distance = 0;
                        index = 0;
                        bestIndex = -1;
                        for (AID restArea : newRestAreaList) {
                                if (restArea != restAreaManager &&
newRestAreaKMList.get(index) >= km && newRestAreaDistanceList.get(index) > distance) {
                                        distance = newRestAreaDistanceList.get(index);
                                        bestIndex = index;
                                }
                                index++;
                        }

                        //If there is no one to negotiate, refuse the negotiation
                        if (bestIndex == -1) {
                                ACLMessage msgCFPRefuse = new
ACLMessage(ACLMessage.REFUSE);
                                msgCFPRefuse.addReceiver(restAreaManager);
                                msgCFPRefuse.setInReplyTo(replyCode);
                                myAgent.send(msgCFPRefuse);
                                step = 6;
                                break;
                        }

                        //Ask to the selected rest area if a spot is avaliable
                        ocupationCode = UUID.randomUUID().toString();

                        ACLMessage msgRequestOcupation = new
ACLMessage(ACLMessage.REQUEST);

      msgRequestOcupation.addReceiver(newRestAreaList.get(bestIndex));
                        msgRequestOcupation.setInReplyTo(ocupationCode);
                        msgRequestOcupation.setConversationId("occupation");

                        //Creating the occupation request message using ontology
                        msgRequestOcupation.setLanguage(codec.getName());
                        msgRequestOcupation.setOntology(ontoloy.getName());
                        reservation = new Reservation();
                        Calendar timeStart, timeEnd;
                        timeStart = Calendar.getInstance();
                        timeEnd = Calendar.getInstance();
                        timeStart.add(Calendar.MINUTE,(int)
(distance*60/drivingSpeed));
                        timeEnd.add(Calendar.MINUTE,(int)
(distance*60/drivingSpeed)+restTime);
                        reservation.setTimeStart(timeStart.getTime());
                        reservation.setTimeEnd(timeEnd.getTime());
                        reservation.setjourneyManagerId(journeyManagerId);
                        occupationRequest = new OccupationRequest();
                        occupationRequest.setReservation(reservation);
                        //The action must be "stored" in an standard "Action"
object, instead of a "custom" AgentAction
                        action = new Action(getAID(),occupationRequest);
```

```
                        try {
                                getContentManager().fillContent(msgRequestOcupation,
action);
                        } catch (CodecException | OntologyException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }

                        myAgent.send(msgRequestOcupation);
                        step++;
                        break;
                case 3:
                        //Check if there is free space in the selected rest area
                        MessageTemplate mtReplyOccupation =
MessageTemplate.MatchInReplyTo(ocupationCode);

                        ACLMessage msgReplyOccupation =
myAgent.receive(mtReplyOccupation);

                        if (msgReplyOccupation != null) {
                                if
(msgReplyOccupation.getPerformative()==ACLMessage.AGREE) {
                                        ACLMessage msgCFPPropose = new
ACLMessage(ACLMessage.PROPOSE);
                                        msgCFPPropose.addReceiver(restAreaManager);
                                        msgCFPPropose.setInReplyTo(replyCode);
                                        //The journey manager sends the remaining time
from the new rest area to the maximun driving time
                                        float timeUntilMaxDriving =
(newRestAreaDistanceList.get(bestIndex))/drivingSpeed * 60; //Time to arrive to the
new rest area
                                        timeUntilMaxDriving = maximunDriveTime -
(driver.getDrivingTime() + timeUntilMaxDriving);

      msgCFPPropose.setContent(String.valueOf(timeUntilMaxDriving));
                                        myAgent.send(msgCFPPropose);
                                        step++;
                                        break;

                                }
                                else {
                                        newRestAreaDistanceList.remove(bestIndex);
                                        newRestAreaList.remove(bestIndex);
                                        step--;
                                        break;

                                }
                        } else block();

                        break;
                case 4:
                        //Wait for acceptation or rejection
                        mtCFPAccept = MessageTemplate.MatchInReplyTo(replyCode);

                        ACLMessage msgCFPResponse = myAgent.receive(mtCFPAccept);

                        if (msgCFPResponse != null) {
```

```
                                    //If the proposal is accepted, the new reservation
will start
                                    if
(msgCFPResponse.getPerformative()==ACLMessage.ACCEPT_PROPOSAL) {
                                          ACLMessage msgRequestParkingSpot = new
ACLMessage(ACLMessage.REQUEST);

      msgRequestParkingSpot.setLanguage(codec.getName());

      msgRequestParkingSpot.setOntology(ontoloy.getName());

      msgRequestParkingSpot.addReceiver(newRestAreaList.get(bestIndex));
                                          reservationCode = UUID.randomUUID().toString();

                                          reservationRequest = new ReservationRequest();
                                          reservationRequest.setReservation(reservation);
                                          //The action must be "stored" in an standard
"Action" object, instead of a "custom" AgentAction
                                          action = new
Action(getAID(),reservationRequest);

                                          try {

      getContentManager().fillContent(msgRequestParkingSpot, action);
                                          } catch (CodecException | OntologyException e)
{
                                                // TODO Auto-generated catch block
                                                e.printStackTrace();
                                          }


      msgRequestParkingSpot.setInReplyTo(reservationCode);

      msgRequestParkingSpot.setConversationId("request");
                                          myAgent.send(msgRequestParkingSpot);
                                          step++;
                                          break;
                                    }

                                    //If our proposal is rejected, the behavior is over
                                    if
(msgCFPResponse.getPerformative()==ACLMessage.REJECT_PROPOSAL){

                                          step = 6;
                                          break;
                                    }
                              } else block();
                              break;
                        case 5:
                              //Send a msg to restAreaManager confirming that the
reservation has been done
                              MessageTemplate mtReservationReply =
MessageTemplate.MatchInReplyTo(reservationCode);

                              ACLMessage msgReservationReply =
myAgent.receive(mtReservationReply);
```

```
                        if (msgReservationReply != null) {

                                //Everything is OK, the agent answers the original
rest area that we have done a reservation
                                if
(msgReservationReply.getPerformative()==ACLMessage.AGREE) {
                                        nextRestArea = msgReservationReply.getSender();
                                        ACLMessage msgInform = new
ACLMessage(ACLMessage.INFORM);
                                        msgInform.addReceiver(restAreaManager);
                                        msgInform.setInReplyTo(replyCode);
                                        myAgent.send(msgInform);
                                        nextRestArea = newRestAreaList.get(bestIndex);

        estimatedArrivalTime.setTime(reservation.getTimeStart());

        estimatedArrivalTime.setTime(reservation.getTimeEnd());
                                        step++;
                                        break;
                                }

                                //If there is a problem, the agent answers the
original rest area that we have not done a reservation
                                if
(msgReservationReply.getPerformative()==ACLMessage.REFUSE) {
                                        ACLMessage msgInform = new
ACLMessage(ACLMessage.FAILURE);
                                        msgInform.addReceiver(restAreaManager);
                                        msgInform.setInReplyTo(replyCode);
                                        myAgent.send(msgInform);
                                        step++;
                                        break;
                                }
                        } else block();
                        break;
                }
        }

        @Override
        public boolean done() {
                if (step == 6) {
                        addBehaviour(new BJourneyManagerNegotiation());
                        return true;
                }
                // TODO Auto-generated method stub
                return false;
        }
    }

    private class BJourneyManagerUpdate extends Behaviour {
        int minutes = 0;
        int km_aux = 0;

        @Override
        public void action() {
                // TODO Auto-generated method stub
```

```
                    MessageTemplate mtInformUpdate =
MessageTemplate.and(MessageTemplate.MatchPerformative(ACLMessage.INFORM),
MessageTemplate.MatchConversationId("timeUpdate"));

                    ACLMessage msgInformUpdate = myAgent.receive(mtInformUpdate);
                    if (msgInformUpdate != null) {
                          try {
                                ContentElement content =
getContentManager().extractContent(msgInformUpdate);
                                UpdateTime updateTime = (UpdateTime) ((Action)
content).getAction();
                                actualTime.setTime(updateTime.getTimeUpdated());
                                minutes = updateTime.getMinutesUpdated();
                                km_aux = Math.round(drivingSpeed * minutes / 60);
                                //The truck is in the rest area
                                if ((actualTime.before(estimatedEndingTime) ||
actualTime.equals(estimatedEndingTime)) && (actualTime.after(estimatedArrivalTime) ||
actualTime.equals(estimatedArrivalTime))) {
                                      resting = true;
                                      if (drivedTimeBeforeRest != 0)
                                            lostTime = lostTime + (maximunDriveTime -
drivedTimeBeforeRest);
                                      drivedTimeBeforeRest = 0;
                                }
                                //The truck advances
                                else {
                                      drivedTimeBeforeRest = drivedTimeBeforeRest +
minutes;
                                      drivedTime = drivedTime + minutes;
                                      km = km + km_aux;
                                      //If the truck reaches certain point of the
road, it creates a log and disapear from the system
                                      if (km >= km_out) {
                                            myAgent.doDelete();
                                      }
                                      //If the journey manager leaves the rest area,
it must do another reservation
                                      if (resting) {
                                            long restTime =
(estimatedEndingTime.getTimeInMillis() -
estimatedArrivalTime.getTimeInMillis())/60000; //from ms to min
                                            restedTime = restedTime + (int)restTime;
                                            resting = false;
                                            myAgent.addBehaviour(new
BJourneyManagerReservation());
                                      }
                                }
                          } catch (CodecException | OntologyException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                          }
                    } else block();

              }

              @Override
              public boolean done() {
```

```
                    // TODO Auto-generated method stub
                    return false;
              }
       }
       //**************************************************************************
*************//
       //**************************************************************************
*************//


       //**************************************************************************
*************//
       //**************************************************************************
*************//
       // Simulation without negotiation behaviours: BJourneyManagerReservationFree,
                    //
       //
BJourneyManagerUpdateFree                                    //
       //**************************************************************************
*************//
       //**************************************************************************
*************//
       private class BJourneyManagerReservationFree extends Behaviour {

              int step = 0;
              int bestIndex = 0;
              int index = 0;
              float bestDistance;
              String reservationCode;

              @Override
              public void action() {
                    switch (step){
                    //Creating the Preference List
                    case 0:
                          if (!creatingList) {
                                addBehaviour(new
BJourneyManagerCreatingPreferencesTable());
                                creatingList = true;
                          }
                          if (listCompleted) {
                                step++;
                                creatingList = false;
                          }
                          break;
                    //Choosing the best rest area to park (it is the rest area that
allows the truck drive the maximun distance
                    case 1:
                          //If the journey manager runs out of options, it must exit
of the system
                          if (RestAreaList.isEmpty()){
                                infractionCommited = 1;
                                myAgent.doDelete();
                                step = 5;
                                break;
                          }
                          bestIndex = 0;
```

```
                        index = 0;
                        bestDistance = 0;
                        //The best will be the one most far away from my position
(the one which is value is greater)
                        for (float distance : RestAreaDistanceList){
                              if (distance > bestDistance) {
                                    bestIndex = index;
                                    bestDistance = distance;
                              }
                              index++;
                        }
                        //This will be the target rest area
                        nextRestArea = RestAreaList.get(bestIndex);
                        Calendar timeStart = Calendar.getInstance(), timeEnd =
Calendar.getInstance();
                        timeStart.setTime(actualTime.getTime());
                        timeEnd.setTime(actualTime.getTime());
                        timeStart.add(Calendar.MINUTE,(int)
(bestDistance*60/drivingSpeed));
                        timeEnd.add(Calendar.MINUTE,(int)
(bestDistance*60/drivingSpeed)+restTime);
                        estimatedArrivalTime.setTime(timeStart.getTime());
                        estimatedEndingTime.setTime(timeEnd.getTime());
                        //
                        step++;
                        break;
                  case 2:
                        if (reachRestArea) {
                              reservationCode = UUID.randomUUID().toString();
                              ACLMessage msgRequestParkingSpot = new
ACLMessage(ACLMessage.REQUEST);
                              msgRequestParkingSpot.addReceiver(nextRestArea);

      msgRequestParkingSpot.setConversationId("requestFree");
                              msgRequestParkingSpot.setInReplyTo(reservationCode);
                              //Send the reservation
                              myAgent.send(msgRequestParkingSpot);
                              step++;
                              break;
                        }
                        break;
                  case 3:
                        MessageTemplate mtReservationReply =
MessageTemplate.MatchInReplyTo(reservationCode);

                        ACLMessage msgReservationReply =
myAgent.receive(mtReservationReply);

                        if (msgReservationReply != null) {
                              //The journey manager dida reservation
                              if (msgReservationReply.getPerformative() ==
ACLMessage.AGREE){
                                    resting = true;
                                    step++;
                                    break;
                              }
```

```
                                //The journey manager couldn't rest in the desired
area
                                else {
                                        infractionCommited = 1;
                                        myAgent.doDelete();
                                        step = 5;
                                        break;
                                }
                        }
                        break;
                case 4:
                        if (actualTime.after(estimatedEndingTime)) {
                                //The Journey Manager requests to the Rest Area
Manager to free the parking spot
                                ACLMessage msgRequestFreeParkingSpot = new
ACLMessage(ACLMessage.REQUEST);
                                msgRequestFreeParkingSpot.addReceiver(nextRestArea);

        msgRequestFreeParkingSpot.setConversationId("requestSetSpotFree");
                                //Send the reservation
                                myAgent.send(msgRequestFreeParkingSpot);
                                step++;
                                break;
                        }
                        break;
                }
        }

        @Override
        public boolean done() {
                if (step == 5) return true;
                return false;
        }
    }
    private class BJourneyManagerUpdateFree extends Behaviour {
        int minutes = 0;
        int km_aux = 0;

        @Override
        public void action() {
                // TODO Auto-generated method stub
                MessageTemplate mtInformUpdate =
MessageTemplate.and(MessageTemplate.MatchPerformative(ACLMessage.INFORM),
MessageTemplate.MatchConversationId("timeUpdate"));

                ACLMessage msgInformUpdate = myAgent.receive(mtInformUpdate);
                if (msgInformUpdate != null) {
                        try {
                                ContentElement content =
getContentManager().extractContent(msgInformUpdate);
                                UpdateTime updateTime = (UpdateTime) ((Action)
content).getAction();
                                actualTime.setTime(updateTime.getTimeUpdated());
                                minutes = updateTime.getMinutesUpdated();
                                km_aux = Math.round(drivingSpeed * minutes / 60);
                                //The truck is in the rest area
```

```
                                if ((actualTime.before(estimatedEndingTime) ||
actualTime.equals(estimatedEndingTime)) && (actualTime.after(estimatedArrivalTime) ||
actualTime.equals(estimatedArrivalTime))) {
                                        reachRestArea = true;
                                        if (drivedTimeBeforeRest != 0)
                                                lostTime = lostTime + (maximunDriveTime -
drivedTimeBeforeRest);
                                        drivedTimeBeforeRest = 0;
                                }
                                //The truck advances
                                else {
                                        drivedTimeBeforeRest = drivedTimeBeforeRest +
minutes;
                                        drivedTime = drivedTime + minutes;
                                        km = km + km_aux;
                                        //If the truck reaches certain point of the
road, it creates a log and disapear from the system
                                        if (km >= km_out) {
                                                myAgent.doDelete();
                                        }
                                        if (resting) {
                                                reachRestArea = false;
                                                resting = false;
                                                long restTime =
(estimatedEndingTime.getTimeInMillis() -
estimatedArrivalTime.getTimeInMillis())/60000; //from ms to min
                                                restedTime = restedTime + (int)restTime;
                                                myAgent.addBehaviour(new
BJourneyManagerReservationFree());
                                        }
                                }
                        } catch (CodecException | OntologyException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                } else block();

        }

        @Override
        public boolean done() {
                // TODO Auto-generated method stub
                return false;
        }
    }
    //**************************************************************************
*************//
    //**************************************************************************
*************//


    //**************************************************************************
*************//
    //**************************************************************************
*************//
    // General behavious: BJourneyManagerCreatingPreferencesTable,
BJourneyManagerShutDown          //
```

```
	//**************************************************************************
*************//
	//**************************************************************************
*************//
	private class BJourneyManagerCreatingPreferencesTable extends Behaviour {
		int step = 0;
		DFAgentDescription dfd = null;
		DFAgentDescription[] result = null;
		int numParkingSpots = 0;

		@Override
		public void action() {
			// TODO Auto-generated method stub

			switch(step) {
			case 0:
				//Send a message to all Rest Area agents in the same road of
the truck to ask AIDs
				ServiceDescription sd = new ServiceDescription();
				sd.setType("RestArea");
				sd.setName(road.getName());
				dfd = new DFAgentDescription();
				dfd.addServices(sd);
				step++;
				break;
			case 1:
				//Search for 0.1 sec
				while (result == null) {
				    try {
					result = DFService.searchUntilFound(
							myAgent, getDefaultDF(), dfd, null,
100);
					} catch (FIPAException e) { e.printStackTrace();}
				}
			    if (result.length >= 1) step++;
			    break;
			case 2:
				String localName;
				float distance;

				for (DFAgentDescription restArea : result) {
					localName = restArea.getName().getLocalName();
					distance =
Float.parseFloat(localName.substring(localName.indexOf(road.getName())+road.getName().
length()));
					//The only rest areas that could be of use, are those
which are at not to far away or behind of the truck
					if ((distance  > km) && (distance < km +
((maximunDriveTime - (driver.getDrivingTime())*drivingSpeed/60)))) {
						RestAreaList.add(restArea.getName());
						RestAreaDistanceList.add((distance-km));
						RestAreaKMList.add(distance);
					}
				}
				step++;
				break;
			}
```

```
			}

		@Override
		public boolean done() {
			if (step == 3) {
				listCompleted = true;
				return true;
			}
			return false;
		}
	}

	private class BJourneyManagerShutDown extends Behaviour {

		@Override
		public void action() {
			// TODO Auto-generated method stub
			MessageTemplate mtInformShutDown =
MessageTemplate.and(MessageTemplate.MatchPerformative(ACLMessage.INFORM),
MessageTemplate.MatchConversationId("shutDown"));

			ACLMessage msgInformUpdate = myAgent.receive(mtInformShutDown);
			if (msgInformUpdate != null) {
				myAgent.doDelete();
			}
			}

		@Override
		public boolean done() {
			// TODO Auto-generated method stub
			return false;
		}
	}
	//**************************************************************************
*************//
	//**************************************************************************
*************//


}
```

AGLOADER

```
package agents;

import gui.LoaderGUI;
import jade.content.lang.Codec;
import jade.content.lang.Codec.CodecException;
import jade.content.lang.sl.SLCodec;
import jade.content.onto.Ontology;
import jade.content.onto.OntologyException;
import jade.content.onto.basic.Action;
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.Behaviour;
```

```
import jade.core.behaviours.CyclicBehaviour;
import jade.core.behaviours.OneShotBehaviour;
import jade.core.behaviours.TickerBehaviour;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.domain.JADEAgentManagement.JADEManagementOntology;
import jade.domain.JADEAgentManagement.ShutdownPlatform;
import jade.gui.GuiAgent;
import jade.gui.GuiEvent;
import jade.wrapper.AgentController;
import jade.wrapper.ContainerController;
import jade.wrapper.StaleProxyException;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import java.nio.charset.Charset;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Random;
import java.util.StringTokenizer;

import ontology.Driver;
import ontology.ParkingManagementOntology;
import ontology.Reservation;
import ontology.ReservationRequest;
import ontology.RestArea;
import ontology.Road;
import ontology.UpdateTime;
import ontology.Vehicle;


/**
 * @author Raquel García Arévalo
 * @code: Loader agent code
 *
 */

public class AgLoader extends GuiAgent {
	private Codec codec = new SLCodec();
```

```
private Ontology ontologia = ParkingManagementOntology.getInstance();
private AID ID;
private gui.GUIData SetupData;
private int createdVehicles = 0;
private AgentController ac;
private Calendar SimulationStart = Calendar.getInstance();
private Calendar SimulatedNow = Calendar.getInstance();
private LoaderGUI myGUI;
private ArrayList<Road> roadList = new ArrayList<Road>();
private PrintWriter wrVehicleFile;

@Override
protected void setup() {
	// Register codec and ontology
	getContentManager().registerLanguage(codec);
	getContentManager().registerOntology(ontologia);

	// Register the offered service
	DFAgentDescription dfd = new DFAgentDescription();
	ID = getAID();
	dfd.setName(ID);
	ServiceDescription sd = new ServiceDescription();
	sd.setType("Loader");
	sd.setName("loader");
	dfd.addServices(sd);
	try {
		DFService.register(this, dfd);
	} catch (Exception e) {
		e.printStackTrace();
	}


	// Open the GUI
	myGUI = new gui.LoaderGUI(this);
	myGUI.setVisible(true);

	//Set time
	SimulationStart.setTime(Calendar.getInstance().getTime());
	SimulatedNow.setTime(Calendar.getInstance().getTime());
}

@Override
protected void takeDown()
{
	//Warn  the agents of the system of the shut down of the system
	DFAgentDescription[] resultJM = null;
	ACLMessage msgShutDownJM = new ACLMessage(ACLMessage.INFORM);
	msgShutDownJM.setConversationId("shutDown");

	//Add all the journey managers
```

```
                ServiceDescription sdJourneyManager = new ServiceDescription();
                sdJourneyManager.setType("JourneyManager");
                DFAgentDescription dfdJourneyManager = new DFAgentDescription();
                dfdJourneyManager.addServices(sdJourneyManager);

                while (resultJM == null) {
                  try {
                        resultJM = DFService.searchUntilFound(this, getDefaultDF(), dfdJourneyManager,
null, 500);
                        } catch (FIPAException e) {
                                e.printStackTrace();
                                System.out.println("FIPA Error while searching for agents");
                        }
                }

        if (resultJM.length > 0)
                for (DFAgentDescription agent : resultJM) {
                        msgShutDownJM.addReceiver(agent.getName());
                }

        //Send the shut down msg and wait 0.5 sec to allow journey managers to send their
information
        send(msgShutDownJM);

        System.out.println("Closing journey managers, this could take some time...");
                try {
                        Thread.sleep(500);
                } catch (InterruptedException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

                //Warn  the agents of the system of the shut down of the system
                DFAgentDescription[] resultRA = null;
                ACLMessage msgShutDownRA = new ACLMessage(ACLMessage.INFORM);
                msgShutDownRA.setConversationId("shutDown");

                //Add all the rest areas
                ServiceDescription sdRestAreaManager = new ServiceDescription();
                sdRestAreaManager.setType("RestArea");
                DFAgentDescription dfdRestAreaManager = new DFAgentDescription();
                dfdRestAreaManager.addServices(sdRestAreaManager);

                while (resultRA == null) {
                  try {
                        resultRA = DFService.searchUntilFound(this, getDefaultDF(), dfdRestAreaManager,
null, 500);
                        } catch (FIPAException e) {
                                e.printStackTrace();
                                System.out.println("FIPA Error while searching for agents");
```

```
                        }
                }

        if (resultRA.length > 0)
                for (DFAgentDescription agent : resultRA) {
                        msgShutDownRA.addReceiver(agent.getName());
                }

        //Send the shut down msg and wait 0.5 sec to allow journey managers to send their
information
        send(msgShutDownRA);

        System.out.println("Closing rest area managers, this could take some time...");
                try {
                        Thread.sleep(500);
                } catch (InterruptedException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

                //Warn  the agents of the system of the shut down of the system
                DFAgentDescription[] resultPS = null;
                ACLMessage msgShutDownPS = new ACLMessage(ACLMessage.INFORM);
                msgShutDownPS.setConversationId("shutDown");

                //Add all the parking spots
                ServiceDescription sdParkingSpots = new ServiceDescription();
                sdRestAreaManager.setType("ParkingSpot");
                DFAgentDescription dfdParkingSpots = new DFAgentDescription();
                dfdParkingSpots.addServices(sdParkingSpots);

                while (resultPS == null) {
                   try {
                        resultPS = DFService.searchUntilFound(this, getDefaultDF(), dfdParkingSpots, null,
500);
                        } catch (FIPAException e) {
                                e.printStackTrace();
                                System.out.println("FIPA Error while searching for agents");
                        }
                }

        if (resultPS.length > 0)
                for (DFAgentDescription agent : resultPS) {
                        msgShutDownPS.addReceiver(agent.getName());
                }

        send(msgShutDownPS);

        System.out.println("Closing parking spot managers, this could take some time...");
                try {
```

```
                Thread.sleep(500);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }

            //Warn  the agents of the system of the shut down of the system
            DFAgentDescription[] resultLog = null;
            ACLMessage msgShutDownLog = new ACLMessage(ACLMessage.INFORM);
            msgShutDownLog.setConversationId("shutDown");

            //Add the log agent
            ServiceDescription sdLogManager = new ServiceDescription();
            sdLogManager.setType("LogManager");
            sdLogManager.setName("logManager");
            DFAgentDescription dfdLogManager = new DFAgentDescription();
            dfdLogManager.addServices(sdLogManager);

            while (resultLog == null) {
              try {
                resultLog = DFService.searchUntilFound(this, getDefaultDF(), dfdLogManager, null,
500);
                } catch (FIPAException e) {
                    e.printStackTrace();
                    System.out.println("FIPA Error while searching for agents");
                }
            }

        if (resultLog.length > 0)
            for (DFAgentDescription agent : resultLog) {
                msgShutDownLog.addReceiver(agent.getName());
            }

            //Send the message
            send(msgShutDownLog);

        System.out.println("Closing log manager, this could take some time...");
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }

            System.out.println("Closing...");
            //If we didn't start the simulation, SetupData is null.
            if (SetupData != null && SetupData.i_CreateLog) {
                wrVehicleFile.close();
            }
```

```
		if (myGUI != null)
		{
			myGUI.setVisible(false);

			//Asking the AMS to shut down the platform
			Codec codec = new SLCodec();
			Ontology jmo = JADEManagementOntology.getInstance();
			getContentManager().registerLanguage(codec);
			getContentManager().registerOntology(jmo);
			ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
			msg.addReceiver(getAMS());
			msg.setLanguage(codec.getName());
			msg.setOntology(jmo.getName());
			try {
			   getContentManager().fillContent(msg, new Action(getAID(), new
ShutdownPlatform()));
			   send(msg);
			}
			catch (Exception e) {}
		}
	}

	Road findRoad(int id) {
		for (Road road : roadList) {
			if (road.getRoadId() == id) {
				return road;
			}
		}
		return null;
	}

	@Override
	protected void onGuiEvent(GuiEvent ev) {

		int command = ev.getType();

		if (command == 0)
		{
			this.doDelete();
		}
		else if (command == 1) {
			SetupData = (gui.GUIData) ev.getParameter(0);

			File FilesPath;
			FilesPath = new File(SetupData.i_setupFilesFolder);

			ContainerController cc = getContainerController();

			try {
				Object[] args = new Object[1];
```

```
            args[0] = FilesPath.toString();

            ac = cc.createNewAgent("LogManager",
                                   "agents.AgLogManager", args);
            ac.start();
        } catch (StaleProxyException e) {
            System.out.println("Error creating LogManager Agent");
            e.printStackTrace();
        }

        Charset charset = Charset.forName("US-ASCII");

        //The created agents need some time to finish their setup
        System.out.println("Creating log manager, this could take some seconds...");
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        //

        // ***************** Road ********************************
        Path RoadsFile = Paths.get(FilesPath.getAbsolutePath() + "\\Roads");
        try (BufferedReader reader1 = Files.newBufferedReader(RoadsFile,
                    charset)) {
            String line = null;

            while ((line = reader1.readLine()) != null) {

                StringTokenizer fields = new StringTokenizer(line, ",");

                Road r = new Road();
                r.setRoadId(Integer.parseInt(fields.nextToken().trim()));
                r.setName(fields.nextToken().trim());
                r.setLenght(Integer.parseInt(fields.nextToken().trim()));
                r.setTraffic(Float.parseFloat(fields.nextToken().trim()));
                roadList.add(r);

            }
        } catch (IOException x) {
            System.out
                    .println("Couldn't read file:" + RoadsFile.toString());
            System.err.format("IOException: %s%n", x);
        }


        // ***************** REST AREAS CREATION ************************
        // Search for the RestAreas file
        Path RestAreasFile = Paths.get(FilesPath.getAbsolutePath()
```

```
                + "\\RestAreas");
try (BufferedReader reader2 = Files.newBufferedReader(
                RestAreasFile, charset)) {
        String line = null;
        while ((line = reader2.readLine()) != null) {
                StringTokenizer fields = new StringTokenizer(line, ",");

                RestArea ra = new RestArea();

                ra.setRestAreaId(Integer.parseInt(fields.nextToken().trim()));

                int roadID = Integer.parseInt(fields.nextToken().trim());

                try {
                        ra.setRoad(findRoad(roadID));
                } catch (NullPointerException e) {
                        System.out.println("Road for the Rest Area not found");
                        e.printStackTrace();
                }

                ra.setKM(Integer.parseInt(fields.nextToken().trim()));
                ra.setName(fields.nextToken().trim());
                ra.setAreaSpots(Integer.parseInt(fields.nextToken().trim()));

                // Create agent
                try {
                        Object[] args = new Object[2];
                        args[0] = ra;
                        args[1] = SetupData.i_simulationType;

                        ac = cc.createNewAgent("" + ra.getName() + ""
                                        + ra.getRoad().getName() + ra.getKM(),
                                        "agents.AgRestAreaManager", args);
                        ac.start();
                } catch (StaleProxyException e) {
                        System.out.println("Error creating AgRestAreaManager");
                        e.printStackTrace();
                }

        }
} catch (IOException x) {
        System.out.println("Couldn't read file:"
                        + RestAreasFile.toString());
        System.err.format("IOException: %s%n", x);
}


//The created agents need some time to finish their setup
System.out.println("Creating rest areas, this could take some seconds...");
try {
```

```
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    }
                    //

                    if (SetupData.i_createRandomVehicles) {

                        if (SetupData.i_CreateLog) {
                            try {
                                wrVehicleFile = new PrintWriter(
                                        SetupData.i_setupFilesFolder.toString() +
"\\"
                                                                        +
SetupData.i_vehicleLogName, "UTF-8");
                            } catch (FileNotFoundException
                                    | UnsupportedEncodingException e) {

                                e.printStackTrace();
                            }

                        }


                        addBehaviour(new CreateJourneyManagerAgent(this,
SetupData.i_vehicleCreationLapse * 1000));

                    } else {
                        // ***************** JOURNEY MANAGERS CREATION
                        // ************************
                        // Search for the Vehicles file
                        Path VehiclesFile = Paths.get(FilesPath.getAbsolutePath()
                                + "\\Vehicles");
                        try (BufferedReader reader3 = Files.newBufferedReader(
                                VehiclesFile, charset)) {
                            String line = null;
                            while ((line = reader3.readLine()) != null) {
                                StringTokenizer fields = new StringTokenizer(line, ",");

                                Vehicle v = new Vehicle();

                                v.setVehicleId(Integer.parseInt(fields.nextToken()
                                        .trim()));
                                v.setMaxSpeed(Integer.parseInt(fields.nextToken()
                                        .trim()));

                                int roadID = Integer
                                        .parseInt(fields.nextToken().trim());
                                v.setRoad(findRoad(roadID));
```

```
                                        v.setKM(Integer.parseInt(fields.nextToken().trim()));

                                        Driver vehicleDriver = new Driver();

        vehicleDriver.setDriverId(Integer.parseInt(fields.nextToken().trim()));
                                        vehicleDriver.setName(fields.nextToken().trim());
                                        vehicleDriver.setDrivingTime(Integer.parseInt(fields
                                                .nextToken().trim()));
                                        v.setDriver(vehicleDriver);

                                        // Create agent
                                        try {
                                                Object[] args = new Object[3];
                                                args[0] = v;
                                                args[1] = SetupData.i_simulationType;
                                                args[2] = SimulatedNow;

                                                ac = cc.createNewAgent("JM" + v.getVehicleId(),
                                                                "agents.AgJourneyManager",
args);
                                                ac.start();
                                        } catch (StaleProxyException e) {
                                                e.printStackTrace();
                                        }

                                }
                        } catch (IOException x) {
                                System.err.format("IOException: %s%n", x);
                        }
                        // END OF AGENTS CREATION
                }

                //The created agents need some time to finish their setup
                System.out.println("Creating journey managers, this could take some seconds...");
                try {
                        Thread.sleep(1000);
                } catch (InterruptedException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                //

                addBehaviour(new BUpdateTime(this, SetupData.i_simulationUpdate));
        }

}

private class CreateJourneyManager extends OneShotBehaviour {
```

```
		private Vehicle CreateRandomVehicle() {
			Random myRandomizer = new Random();
			createdVehicles++;
			Vehicle v = new Vehicle();
			Driver d = new Driver();
			d.setDriverId(createdVehicles);
			d.setName("Name" + createdVehicles);
			d.setDrivingTime(myRandomizer.nextInt(20) * 10); //Random drived time from 0
up to 200 minutes
			v.setDriver(d);
			v.setVehicleId(createdVehicles);
			v.setMaxSpeed((myRandomizer.nextInt(4) + 6) * 10); //Random speed from 60 up
to 100 km/h
			v.setRoad(roadList.get(myRandomizer.nextInt(roadList.size())));
			v.setKM(myRandomizer.nextInt(v.getRoad().getLenght()));
			return v;
		}

		@Override
		public void action() {
			try {
				Vehicle rv = CreateRandomVehicle();
				Object[] args = new Object[3];
				args[0] = rv;
				args[1] = SetupData.i_simulationType;
				args[2] = SimulatedNow;

				ContainerController cc = getContainerController();
				ac = cc.createNewAgent("JM" + rv.getVehicleId(),
						"agents.AgJourneyManager", args);

				if (SetupData.i_CreateLog) {
					String vdata = rv.getVehicleId() + "," + rv.getMaxSpeed() + ","
							+ rv.getRoad().getRoadId() + "," + rv.getKM() + ","
							+ rv.getDriver().getDriverId() + ","
							+ rv.getDriver().getName() + ","
							+ rv.getDriver().getDrivingTime();
					wrVehicleFile.println(vdata);
				}

				ac.start();
			} catch (StaleProxyException e) {
				e.printStackTrace();
			}

		}

	}
```

```
	private class CreateJourneyManagerAgent extends TickerBehaviour {

		public CreateJourneyManagerAgent(Agent a, long period) {
			super(a, period);
		}

		@Override
		protected void onTick() {
			for (int i = 0; i < SetupData.i_numberOfVehicles; i++) {
				addBehaviour(new CreateJourneyManager());
			}
		}

	}

	private class BUpdateTime extends TickerBehaviour {
		//Ontology
		private Codec codec = new SLCodec();
		private Ontology ontoloy = ParkingManagementOntology.getInstance();
		Action action;

		public BUpdateTime(Agent a, long period) {
			super(a, period);
			// TODO Auto-generated constructor stub
		}

		@Override
		protected void onTick() {
			DFAgentDescription[] result = null;
			ACLMessage msgTimeUpdate = new ACLMessage(ACLMessage.INFORM);
			msgTimeUpdate.setConversationId("timeUpdate");

			//Hacer una búsqueda de todos los journeyManager que están "vivos"
			ServiceDescription sd = new ServiceDescription();
			sd.setType("JourneyManager");
			DFAgentDescription dfd = new DFAgentDescription();
			dfd.addServices(sd);

			//Search for a tenth of the waiting time of the tick behaviour
			   try {
				result = DFService.searchUntilFound(myAgent, getDefaultDF(), dfd, null,
getPeriod()/10);
				} catch (FIPAException e) {
					e.printStackTrace();
					System.out.println("FIPA Error while searching for agents");
				}

			//Update time adding the lapse
			SimulatedNow.add(Calendar.MINUTE, SetupData.i_simMinutesPerLapse);
```

```
                        //Creating the reservation request message using ontology
                        msgTimeUpdate.setLanguage(codec.getName());
                        msgTimeUpdate.setOntology(ontoloy.getName());
                        UpdateTime updateTime = new UpdateTime();
                        updateTime.setTimeUpdated(SimulatedNow.getTime());
                        updateTime.setMinutesUpdated(SetupData.i_simMinutesPerLapse);
                        //The action must be "stored" in an standard "Action" object, instead of a
"custom" AgentAction
                        action = new Action(getAID(),updateTime);
                        try {
                                getContentManager().fillContent(msgTimeUpdate, action);
                        } catch (CodecException | OntologyException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                        long diference = (SimulatedNow.getTimeInMillis() -
SimulationStart.getTimeInMillis()) /60000;

                        myGUI.setStatusProgress((int)diference,SimulatedNow.getTime());

                        if (diference < SetupData.i_simulationTime) {
                                //msgTimeUpdate.setContent(SimulatedNow.getTime().toString());

                                //If we found at least one Journey Manager, we send them the new time
                                if (result != null && result.length >= 1) {
                                        //Add all journey manager to the msg
                                        for (DFAgentDescription journeyManager : result) {

        msgTimeUpdate.addReceiver(journeyManager.getName());
                                        }
                                        //Send the update time msg
                                        myAgent.send(msgTimeUpdate);
                                }
                        }
                        else {
                                //If we have reached the end time, we close.
                                doDelete();
                        }
                }

        }
}
```

AGLOGMANAGER

```
package agents;

import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;

import jade.content.Concept;
import jade.content.ContentElement;
import jade.content.lang.Codec;
import jade.content.lang.sl.SLCodec;
import jade.content.onto.Ontology;
import jade.content.onto.basic.Action;
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.Behaviour;
import jade.core.behaviours.CyclicBehaviour;
import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import ontology.LogJourneyData;
import ontology.LogRelocation;
import ontology.ParkingManagementOntology;

/**
 * @author Raquel García Arévalo
 * @code: Log Manager Agent code
 *
 */

public class AgLogManager extends Agent {
	private Codec codec = new SLCodec();
	private Ontology ontologia = ParkingManagementOntology.getInstance();
	private AID ID;

	private String filesPath;
	private PrintWriter wrInfractionsLog;
	private PrintWriter wrRelocationsLog;
	private PrintWriter wrJourneyDataLog;

	@Override
	protected void setup() {
		// Register codec and ontology
		getContentManager().registerLanguage(codec);
		getContentManager().registerOntology(ontologia);

		// Register the offered service
		DFAgentDescription dfd = new DFAgentDescription();
```

```
		ID = getAID();
		dfd.setName(ID);
		ServiceDescription sd = new ServiceDescription();
		sd.setType("LogManager");
		sd.setName("logManager");
		dfd.addServices(sd);
		try {
			DFService.register(this, dfd);
		} catch (Exception e) {
			e.printStackTrace();
		}

		//Load files path
		Object[] args = getArguments();
		filesPath = (String) args[0];



		//Create files
		try {
			wrJourneyDataLog = new PrintWriter(
					filesPath + "\\JourneyDataLog.txt", "UTF-8");
		} catch (FileNotFoundException
				| UnsupportedEncodingException e) {

			e.printStackTrace();
		}

		try {
			wrRelocationsLog = new PrintWriter(
					filesPath + "\\RelocationsLog.txt", "UTF-8");
		} catch (FileNotFoundException
				| UnsupportedEncodingException e) {

			e.printStackTrace();
		}


		addBehaviour(new BLogMessageReceiver());
		addBehaviour(new BLogManagerShutDown());

	}


	@Override
	protected void takeDown()
	{
		//Close files
		wrRelocationsLog.close();
		wrJourneyDataLog.close();
```

```
}

private class BLogMessageReceiver extends CyclicBehaviour {

	@Override
	public void action() {
		ACLMessage msg = receive(MessageTemplate.MatchConversationId("log"));
		if (msg == null) {
			block();
			return;
		}

		try {
			ContentElement content = getContentManager()
					.extractContent(msg);
			Concept action = ((Action) content).getAction();
			switch (msg.getPerformative()) {
			case (ACLMessage.INFORM):

				if (action instanceof LogJourneyData)
				{
					String journeyDataLine = "";
					LogJourneyData j = (LogJourneyData) action;
					journeyDataLine = j.getIdVehicle()+ ",";
					journeyDataLine += j.getdrivedTime()+ ",";
					journeyDataLine += j.getKm()+ ",";
					journeyDataLine += j.getKmIn()+ ",";
					journeyDataLine += j.getKmOut()+ ",";
					journeyDataLine += j.getTimeIn()+ ",";
					journeyDataLine += j.getTimeOut()+ ",";
					journeyDataLine += j.getlostTime()+ ",";
					journeyDataLine += j.getrestTime()+ ",";
					journeyDataLine += j.getInfractionType();

					wrJourneyDataLog.println(journeyDataLine);


				}

				if (action instanceof LogRelocation)
				{

					String relocationLine = "";
					LogRelocation r = (LogRelocation) action;
					relocationLine = r.getRelocationTime().toString()+ ",";
					relocationLine += r.getAffectedVehicles()+ ",";
					relocationLine += r.getRelocableVehicles()+ ",";
					relocationLine += r.getRelocatedVehicleOrder()+ ",";
					relocationLine += r.getSuccess();
```

```
                                        wrRelocationsLog.println(relocationLine);

                                    }
                                    break;

                            }
                        } catch (Exception ex) {
                            ex.printStackTrace();
                        }
                }
        }

        private class BLogManagerShutDown extends Behaviour {

                @Override
                public void action() {
                        // TODO Auto-generated method stub
                        MessageTemplate mtInformShutDown =
MessageTemplate.and(MessageTemplate.MatchPerformative(ACLMessage.INFORM),
MessageTemplate.MatchConversationId("shutDown"));

                        ACLMessage msgInformUpdate = myAgent.receive(mtInformShutDown);
                        if (msgInformUpdate != null) {
                                myAgent.doDelete();
                            }
                        }

                @Override
                public boolean done() {
                        // TODO Auto-generated method stub
                        return false;
                }
        }

}
```

AGPARKINGSPOT

```
package agents;

import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.Random;
import java.util.UUID;

import ontology.CancelRequest;
import ontology.OccupationRequest;
import ontology.ParkingManagementOntology;
import ontology.Reservation;
import ontology.ReservationRequest;
import jade.content.Concept;
import jade.content.ContentElement;
import jade.content.lang.Codec;
import jade.content.lang.Codec.CodecException;
import jade.content.lang.sl.SLCodec;
import jade.content.onto.Ontology;
import jade.content.onto.OntologyException;
import jade.content.onto.UngroundedException;
import jade.content.onto.basic.Action;
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.Behaviour;
import jade.core.behaviours.CyclicBehaviour;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;


/**
 * @author Francisco Palau Romero
 * @code: Parking Spot agent code
 *
 */

public class AgParkingSpot extends Agent {
	private AID ID;
	private class reservationList {
		Calendar day;
		ArrayList<Reservation> list = new ArrayList<Reservation>();
	}
	private reservationList todayReservations = new reservationList();
	private reservationList tomorrowReservations = new reservationList();
	//Ontology
```

```
	private Codec codec = new SLCodec();
	private Ontology ontoloy = ParkingManagementOntology.getInstance();


	//**************************************************************************************
************//
	//**************************************************************************************
************//
	// Setup
	                                                                                //
	//**************************************************************************************
************//
	//**************************************************************************************
************//
	protected void setup() {
		//Register codec and ontology
	  getContentManager().registerLanguage(codec);
	  getContentManager().registerOntology(ontoloy);

	  //Register the offered service
		DFAgentDescription dfd = new DFAgentDescription();
		ID = getAID();
		dfd.setName(ID);
		ServiceDescription sd = new ServiceDescription();
		sd.setType("ParkingSpot");
		sd.setName("Spot");
		dfd.addServices(sd);
		try{
			DFService.register(this, dfd);
		} catch (Exception e) {e.printStackTrace();}


		//When it is registered, the rest area agent begins his behaviour
		addBehaviour(new BParkingSpotMessageReceiver());
		addBehaviour(new BParkingSpotShutDown());
	}
	//**************************************************************************************
************//
	//**************************************************************************************
************//


	//**************************************************************************************
************//
	//**************************************************************************************
************//
	// Override
	                                                                                //
	//**************************************************************************************
************//
```

```
	//**********************************************************************************//
	@Override
	protected void takeDown() {
		while (receive() != null) {};
	}
	//**********************************************************************************//
	//**********************************************************************************//


	//**********************************************************************************//
	//**********************************************************************************//
	// Reservation behaviours:	BParkingSpotMessageReceiver, BParkingSpotManagerCancelRequest	//
	//	BParkingSpotManagerReservationRequest	//
	//	BParkingSpotManagerOccupationRequest	//
	//**********************************************************************************//
	//**********************************************************************************//
	private class BParkingSpotMessageReceiver extends CyclicBehaviour {
		public void action(){
			ACLMessage msg = receive(MessageTemplate.MatchConversationId("request"));
			if (msg==null){
				block();
				return;
			}

		try {
				ContentElement content = getContentManager().extractContent(msg);
				Concept action = ((Action) content).getAction();
				switch (msg.getPerformative()) {
				case (ACLMessage.REQUEST):
					if (action instanceof OccupationRequest) {
						addBehaviour(new BParkingSpotManagerOccupationRequest(msg));
					}
					else if (action instanceof CancelRequest){
						addBehaviour(new BParkingSpotManagerCancelRequest(msg));
					}
					break;
				case (ACLMessage.CFP):
					if (action instanceof ReservationRequest) {
```

```
                                        addBehaviour(new
BParkingSpotManagerReservationRequest(msg));
                                }
                                break;
                        }
                }
                catch(Exception ex) { ex.printStackTrace(); }
                }
        }


        private class BParkingSpotManagerReservationRequest extends Behaviour {

                int step = 0;
                ACLMessage msg;
                Calendar timeStart, timeEnd;
                Calendar today = Calendar.getInstance();;
                String journeyManagerId;
                AID restArea;
                String reservationCode;
                Boolean startToday;
                int index = 0;
                Action action;

                public BParkingSpotManagerReservationRequest(ACLMessage msg) {
                        // TODO Auto-generated constructor stub
                        this.msg = msg;
                }

                @Override
                public void action() {
                        switch (step) {
                        case 0:
                                //Reading the message
                                restArea = msg.getSender();
                                reservationCode = msg.getInReplyTo();
                                try {
                                        ContentElement content =
getContentManager().extractContent(msg);
                                        ReservationRequest reservationRequest = (ReservationRequest)
((Action) content).getAction();
                                        Reservation reservation = reservationRequest.getReservation();
                                        timeStart = Calendar.getInstance();
                                        timeEnd = Calendar.getInstance();
                                        timeStart.setTime(reservation.getTimeStart());
                                        timeEnd.setTime(reservation.getTimeEnd());
                                        journeyManagerId = reservation.getjourneyManagerId();
                                } catch (CodecException | OntologyException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
```

```
                        }
                        step++;
                        break;
                case 1:
                        //Check for free space
                        startToday = (timeStart.get(Calendar.DAY_OF_YEAR) ==
today.get(Calendar.DAY_OF_YEAR));
                        //The reservation will start today
                        if (startToday) {
                                //If the list is empty...
                                if (todayReservations.list.isEmpty()) {
                                        step++;
                                        break;
                                }
                                //Check for all reservations for this day
                                for ( Reservation reservation : todayReservations.list) {
                                        Calendar timeStartReservation = Calendar.getInstance();
                                        timeStartReservation.setTime(reservation.getTimeStart());
                                        Calendar timeEndReservation = Calendar.getInstance();
                                        timeEndReservation.setTime(reservation.getTimeEnd());
                                        //The asked time reservation will end in an occupied block
of time
                                        if (timeStart.before(timeStartReservation) &&
timeEnd.after(timeStartReservation)){
                                                ACLMessage msgReplyRefuse = new
ACLMessage(ACLMessage.REFUSE);
                                                msgReplyRefuse.addReceiver(restArea);
                                                msgReplyRefuse.setInReplyTo(reservationCode);

        msgReplyRefuse.setContent(reservation.getjourneyManagerId());
                                                myAgent.send(msgReplyRefuse);
                                                step = 4;
                                                break;
                                        }
                                        //The asked time reservation will start between an
occupied block of time
                                        if ((timeStart.after(timeStartReservation) &&
timeStart.before(timeEndReservation)) || timeStart.equals(timeStartReservation)){
                                                ACLMessage msgReplyRefuse = new
ACLMessage(ACLMessage.REFUSE);
                                                msgReplyRefuse.addReceiver(restArea);
                                                msgReplyRefuse.setInReplyTo(reservationCode);

        msgReplyRefuse.setContent(reservation.getjourneyManagerId());
                                                myAgent.send(msgReplyRefuse);
                                                step = 4;
                                                break;
                                        }
                                }
                        }
```

```
//The reservation will begin tomorrow
else {
	//If the list is empty...
	if (tomorrowReservations.list.isEmpty()) {
		step++;
		break;
	}
	//Check for all reservations for today
	for ( Reservation reservation : todayReservations.list ) {
		Calendar timeStartReservation = Calendar.getInstance();
		timeStartReservation.setTime(reservation.getTimeStart());
		Calendar timeEndReservation = Calendar.getInstance();
		timeEndReservation.setTime(reservation.getTimeEnd());
		//The asked time reservation will end in an occupied block
of time
		if (timeStart.before(timeStartReservation) &&
timeEnd.after(timeStartReservation)){
			ACLMessage msgReplyRefuse = new
ACLMessage(ACLMessage.REFUSE);
			msgReplyRefuse.addReceiver(restArea);
			msgReplyRefuse.setInReplyTo(reservationCode);

	msgReplyRefuse.setContent(reservation.getjourneyManagerId());
			myAgent.send(msgReplyRefuse);
			step = 4;
			break;
		}
		//The asked time reservation will start between an
occupied block of time
		if ((timeStart.after(timeStartReservation) &&
timeStart.before(timeEndReservation)) || timeStart.equals(timeStartReservation)){
			ACLMessage msgReplyRefuse = new
ACLMessage(ACLMessage.REFUSE);
			msgReplyRefuse.addReceiver(restArea);
			msgReplyRefuse.setInReplyTo(reservationCode);

	msgReplyRefuse.setContent(reservation.getjourneyManagerId());
			myAgent.send(msgReplyRefuse);
			step = 4;
			break;
		}
	}
	//Check for all reservations for tomorrow
	for ( Reservation reservation : tomorrowReservations.list ) {
		Calendar timeStartReservation = Calendar.getInstance();
		timeStartReservation.setTime(reservation.getTimeStart());
		Calendar timeEndReservation = Calendar.getInstance();
		timeEndReservation.setTime(reservation.getTimeEnd());
		//The asked time reservation will end in an occupied block
of time
```

```
                                        if (timeStart.before(timeStartReservation) &&
timeEnd.after(timeStartReservation)){
                                                ACLMessage msgReplyRefuse = new
ACLMessage(ACLMessage.REFUSE);
                                                msgReplyRefuse.addReceiver(restArea);
                                                msgReplyRefuse.setInReplyTo(reservationCode);

        msgReplyRefuse.setContent(reservation.getjourneyManagerId());
                                                myAgent.send(msgReplyRefuse);
                                                step = 4;
                                                break;
                                        }
                                        //The asked time reservation will start between an
occupied block of time
                                        if ((timeStart.after(timeStartReservation) &&
timeStart.before(timeEndReservation)) || timeStart.equals(timeStartReservation)){
                                                ACLMessage msgReplyRefuse = new
ACLMessage(ACLMessage.REFUSE);
                                                msgReplyRefuse.addReceiver(restArea);
                                                msgReplyRefuse.setInReplyTo(reservationCode);

        msgReplyRefuse.setContent(reservation.getjourneyManagerId());
                                                myAgent.send(msgReplyRefuse);
                                                step = 4;
                                                break;
                                        }
                                }
                        }
                        step++;
                        break;
                case 2:
                        //fraparo: Propongo mi reserva (por el momento, envío un valor igual al
conjunto de horas de ocupación que hay en el día)
                        int minutes = 0;
                        Calendar calendarStart  = Calendar.getInstance(), calendarEnd  =
Calendar.getInstance();

                        ACLMessage msgCFPPropose = new ACLMessage(ACLMessage.PROPOSE);
                        if (startToday) {
                                for ( Reservation reservation : todayReservations.list) {
                                        calendarEnd.setTime(reservation.getTimeEnd());
                                        calendarStart.setTime(reservation.getTimeStart());
                                        minutes = calendarEnd.get(Calendar.MINUTE) -
calendarStart.get(Calendar.MINUTE);
                                }
                        }

                        msgCFPPropose.addReceiver(restArea);
                        msgCFPPropose.setContent(String.valueOf(minutes));
                        msgCFPPropose.setInReplyTo(reservationCode);
```

```
                        myAgent.send(msgCFPPropose);
                        step++;
                        break;
                case 3:
                        //Check if our propose is accepted
                        MessageTemplate mtCFPReply =
MessageTemplate.MatchInReplyTo(reservationCode);

                        ACLMessage msgCFPReply = myAgent.receive(mtCFPReply);

                        if (msgCFPReply != null) {

                                //If our proposal is accepted, we make the reservation
                                if
(msgCFPReply.getPerformative()==ACLMessage.ACCEPT_PROPOSAL) {

                                        //The reservation will start and finish the same day
                                        Reservation r = new Reservation();
                                        r.setTimeStart(timeStart.getTime());
                                        r.setTimeEnd(timeEnd.getTime());
                                        r.setjourneyManagerId(journeyManagerId);
                                        if (startToday) todayReservations.list.add(r);
                                        else tomorrowReservations.list.add(r);
                                }
                                step++;
                                break;
                        } else block();
                }
                }

        @Override
        public boolean done() {
                if (step == 4) {
                        return true;
                }
                return false;
        }

    }


private class BParkingSpotManagerOccupationRequest extends Behaviour {

        int step = 0;
        ACLMessage msg;
        Calendar today = Calendar.getInstance();
        AID restArea;
        String occupationCode;
        Calendar timeStart, timeEnd;
        String journeyManagerId;
```

```
		Boolean occupiedSlot = false;
		Boolean startToday;

		public BParkingSpotManagerOccupationRequest(ACLMessage msg) {
			// TODO Auto-generated constructor stub
			this.msg = msg;
		}

		@Override
		public void action() {
			switch (step){
			case 0:
				today = Calendar.getInstance();
				//Reading the message
				restArea = msg.getSender();
				occupationCode = msg.getInReplyTo();
				try {
					ContentElement content =
getContentManager().extractContent(msg);
					OccupationRequest occupationRequest = (OccupationRequest)
((Action) content).getAction();
					Reservation reservation = occupationRequest.getReservation();
					timeStart = Calendar.getInstance();
					timeEnd = Calendar.getInstance();
					timeStart.setTime(reservation.getTimeStart());
					timeEnd.setTime(reservation.getTimeEnd());
					journeyManagerId = reservation.getjourneyManagerId();
				} catch (CodecException | OntologyException e) {
					e.printStackTrace();
				}
				step++;
				break;
			case 1:
				//Check for free space
				startToday = (timeStart.get(Calendar.DAY_OF_YEAR) ==
today.get(Calendar.DAY_OF_YEAR));
				//The reservation will start today
				if (startToday) {
					//If the list is empty...
					if (todayReservations.list.isEmpty()) {
						step++;
						break;
					}
					//Check for all reservations for this day
					for ( Reservation reservation : todayReservations.list) {
						Calendar timeStartReservation = Calendar.getInstance();
						timeStartReservation.setTime(reservation.getTimeStart());
						Calendar timeEndReservation = Calendar.getInstance();
						timeEndReservation.setTime(reservation.getTimeEnd());
```

```
                                        //The asked time reservation will end in an occupied block
of time
                                        if (timeStart.before(timeStartReservation) &&
timeEnd.after(timeStartReservation)){
                                                occupiedSlot = true;
                                                ACLMessage msgReplyRefuse = new
ACLMessage(ACLMessage.REFUSE);
                                                msgReplyRefuse.addReceiver(restArea);
                                                msgReplyRefuse.setInReplyTo(occupationCode);
                                                myAgent.send(msgReplyRefuse);
                                                step++;
                                                break;
                                        }
                                        //The asked time reservation will start between an
occupied block of time
                                        if ((timeStart.after(timeStartReservation) &&
timeStart.before(timeEndReservation)) || timeStart.equals(timeStartReservation)){
                                                occupiedSlot = true;
                                                ACLMessage msgReplyRefuse = new
ACLMessage(ACLMessage.REFUSE);
                                                msgReplyRefuse.addReceiver(restArea);
                                                msgReplyRefuse.setInReplyTo(occupationCode);
                                                myAgent.send(msgReplyRefuse);
                                                step++;
                                                break;
                                        }
                                }
                        }

                        //The reservation will begin tomorrow
                        else {
                                //Check for all reservations for today
                                for ( Reservation reservation : todayReservations.list ) {
                                        Calendar timeStartReservation = Calendar.getInstance();
                                        timeStartReservation.setTime(reservation.getTimeStart());
                                        Calendar timeEndReservation = Calendar.getInstance();
                                        timeEndReservation.setTime(reservation.getTimeEnd());
                                        //The asked time reservation will end in an occupied block
of time
                                        if (timeStart.before(timeStartReservation) &&
timeEnd.after(timeStartReservation)){
                                                occupiedSlot= true;
                                                ACLMessage msgReplyRefuse = new
ACLMessage(ACLMessage.REFUSE);
                                                msgReplyRefuse.addReceiver(restArea);
                                                msgReplyRefuse.setInReplyTo(occupationCode);
                                                myAgent.send(msgReplyRefuse);
                                                step++;
                                                break;
                                        }
```

```
                                        //The asked time reservation will start between an
occupied block of time
                                        if ((timeStart.after(timeStartReservation) &&
timeStart.before(timeEndReservation)) || timeStart.equals(timeStartReservation)){
                                                occupiedSlot= true;
                                                ACLMessage msgReplyRefuse = new
ACLMessage(ACLMessage.REFUSE);
                                                msgReplyRefuse.addReceiver(restArea);
                                                msgReplyRefuse.setInReplyTo(occupationCode);
                                                myAgent.send(msgReplyRefuse);
                                                step++;
                                                break;
                                        }
                                }
                                //Check for all reservations for tomorrow
                                for ( Reservation reservation : tomorrowReservations.list ) {
                                        Calendar timeStartReservation = Calendar.getInstance();
                                        timeStartReservation.setTime(reservation.getTimeStart());
                                        Calendar timeEndReservation = Calendar.getInstance();
                                        timeEndReservation.setTime(reservation.getTimeEnd());
                                        //The asked time reservation will end in an occupied block
of time
                                        if (timeStart.before(timeStartReservation) &&
timeEnd.after(timeStartReservation)){
                                                occupiedSlot= true;
                                                ACLMessage msgReplyRefuse = new
ACLMessage(ACLMessage.REFUSE);
                                                msgReplyRefuse.addReceiver(restArea);
                                                msgReplyRefuse.setInReplyTo(occupationCode);
                                                myAgent.send(msgReplyRefuse);
                                                step++;
                                                break;
                                        }
                                        //The asked time reservation will start between an
occupied block of time
                                        if ((timeStart.after(timeStartReservation) &&
timeStart.before(timeEndReservation)) || (timeStart.equals(timeStartReservation))){
                                                occupiedSlot= true;
                                                ACLMessage msgReplyRefuse = new
ACLMessage(ACLMessage.REFUSE);
                                                msgReplyRefuse.addReceiver(restArea);
                                                msgReplyRefuse.setInReplyTo(occupationCode);
                                                myAgent.send(msgReplyRefuse);
                                                step++;
                                                break;
                                        }
                                }
                        }
                case 2:
                        if (!occupiedSlot) {
```

```
                                        ACLMessage msgReplyAgree = new
ACLMessage(ACLMessage.AGREE);
                                        msgReplyAgree.addReceiver(restArea);
                                        msgReplyAgree.setInReplyTo(occupationCode);
                                        myAgent.send(msgReplyAgree);
                                        step++;
                                        break;
                                }
                        }
                }

                @Override
                public boolean done() {
                        if (step == 3) return true;
                        return false;
                }
        }


        private class BParkingSpotManagerCancelRequest extends Behaviour {

                int step = 0;
                ACLMessage msg;
                String journeyManagerId;
                AID restArea;

                public BParkingSpotManagerCancelRequest(ACLMessage msg) {
                        // TODO Auto-generated constructor stub
                        this.msg = msg;
                }

                @Override
                public void action() {
                        switch (step){
                        case 0:
                                //Reading the message
                                restArea = msg.getSender();
                                try {
                                        ContentElement content =
getContentManager().extractContent(msg);
                                        CancelRequest cancelRequest = (CancelRequest) ((Action)
content).getAction();
                                        Reservation reservation = cancelRequest.getReservation();
                                        journeyManagerId = reservation.getjourneyManagerId();
                                } catch (CodecException | OntologyException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                }
                                step++;
                                break;
```

```
                        case 1:
                                for ( Reservation reservation : todayReservations.list) {
                                        if (reservation.getjourneyManagerId() == journeyManagerId) {
                                                todayReservations.list.remove(reservation);
                                                step++;
                                                break;
                                                }
                                        }
                                for ( Reservation reservation : tomorrowReservations.list ) {
                                        if (reservation.getjourneyManagerId() == journeyManagerId) {
                                                todayReservations.list.remove(reservation);
                                                step++;
                                                break;
                                                }
                                        }
                                //The reservation doesn't exist
                                step++;
                                break;
                        }
                }

                @Override
                public boolean done() {
                        if (step == 2) return true;
                        return false;
                }
        }
        //**************************************************************************************
************//
        //**************************************************************************************
************//


        //**************************************************************************************
************//
        //**************************************************************************************
************//
        // General behaviours: BParkingSpotShutDown
                                                //
        //**************************************************************************************
************//
        //**************************************************************************************
************//
        private class BParkingSpotShutDown extends Behaviour {

                @Override
                public void action() {
                        // TODO Auto-generated method stub
```

```
                        MessageTemplate mtInformShutDown =
MessageTemplate.and(MessageTemplate.MatchPerformative(ACLMessage.INFORM),
MessageTemplate.MatchConversationId("shutDown"));

                        ACLMessage msgInformUpdate = myAgent.receive(mtInformShutDown);
                        if (msgInformUpdate != null) {
                                myAgent.doDelete();
                        }
                }

                @Override
                public boolean done() {
                        // TODO Auto-generated method stub
                        return false;
                }
        }
        //**************************************************************************************
************//
        //**************************************************************************************
************//

}
```

AGRESTAREAMANAGER

```
package agents;

import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.Random;
import java.util.UUID;

import ontology.CancelRequest;
import ontology.LogJourneyData;
import ontology.LogRelocation;
import ontology.OccupationRequest;
import ontology.ParkingManagementOntology;
import ontology.Reservation;
import ontology.ReservationRequest;
import ontology.RestArea;
import ontology.Road;
import jade.content.Concept;
import jade.content.ContentElement;
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.Behaviour;
import jade.core.behaviours.CyclicBehaviour;
import jade.domain.AMSService;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.wrapper.AgentController;
import jade.wrapper.ContainerController;
import jade.wrapper.StaleProxyException;
import jade.content.lang.Codec;
import jade.content.lang.Codec.CodecException;
import jade.content.lang.sl.SLCodec;
import jade.content.onto.Ontology;
import jade.content.onto.OntologyException;
//
import jade.content.onto.basic.*;


/**
 * @author Francisco Palau Romero
 * @code: Rest Area agent code
 *
**/

public class AgRestAreaManager extends Agent {
```

```
	private Road road;
	private int km;
	private int numParkingSpots;
	private int numOccupiedParkingSpots;
	private int idRestArea;
	private AID ID;
	private ArrayList<AID> parkingSpotList = new ArrayList<AID>();
	//Simulation type
	private int simulationType;
	private static final int BORDA_NEGOTIATION_SIMULATION_TYPE = 0;
	private static final int FREE_SIMULATION_TYPE = 1;
	//Log AID
	private AID logManagerAID;
	//Ontology
	private Codec codec = new SLCodec();
	private Ontology ontoloy = ParkingManagementOntology.getInstance();
	Action action;

	//**************************************************************************************
************//
	//**************************************************************************************
************//
	// Setup
	                                                                        //
	//**************************************************************************************
************//
	//**************************************************************************************
************//
	protected void setup() {
		//Register codec and ontology
	  getContentManager().registerLanguage(codec);
	  getContentManager().registerOntology(ontoloy);
	  //

	  //Extract parameters from Loader
		Object[] args = getArguments();
		RestArea restAreaInfo = (RestArea) args[0];
		simulationType = (Integer) args[1];
		road = restAreaInfo.getRoad();
		km = restAreaInfo.getKM();
		numParkingSpots = restAreaInfo.getAreaSpots();
		idRestArea = restAreaInfo.getRestAreaId();
		//

		//Register the offered service
		DFAgentDescription dfd = new DFAgentDescription();
		ID = getAID();
		dfd.setName(ID);
		ServiceDescription sd = new ServiceDescription();
		sd.setType("RestArea");
```

```
                sd.setName(road.getName());
                dfd.addServices(sd);
                try{
                        DFService.register(this, dfd);
                } catch (Exception e) {e.printStackTrace();}
                //

                //Register the AID of the log manager
                DFAgentDescription[] result = null;
                ServiceDescription sdLog = new ServiceDescription();
                DFAgentDescription dfdLog = new DFAgentDescription();

                sdLog.setType("LogManager");
                sdLog.setName("logManager");
                dfdLog.addServices(sdLog);

                while (result == null) {
                  try {
                        //Search for 100 ms
                        result = DFService.searchUntilFound(this, getDefaultDF(), dfdLog, null, 100);
                        } catch (FIPAException e) {
                                e.printStackTrace();
                                System.out.println("FIPA Error while searching for agents");
                        }
                }

        if (result.length >= 1) {
                for (DFAgentDescription logManagerResult : result){
                        logManagerAID = logManagerResult.getName();
                }
        }
        //


        //The type of simulation it will depend of simulationType
        switch (simulationType) {
                case FREE_SIMULATION_TYPE:
                                numOccupiedParkingSpots = 0;
                                addBehaviour(new BRestAreaManagerSetSpotFree());
                                addBehaviour(new BRestAreaManagerReservationFree());
                        break;
                case BORDA_NEGOTIATION_SIMULATION_TYPE:
                                //Create parking spots
                                ContainerController cc = getContainerController();
                                AgentController ac;
                                for (int i=1;i<=numParkingSpots;i++) {
                                        try {
                                                String spotName = getAID().getLocalName()+"spot"+i;
                                                ac = cc.createNewAgent(spotName,
"agents.AgParkingSpot", null);
```

```
                                        ac.start();
                                        parkingSpotList.add(new AID(spotName,false));
                                } catch (StaleProxyException e) {
                                        e.printStackTrace();
                                }
                        }

                        addBehaviour(new BRestAreaMessageReceiver());
                        addBehaviour(new BRestAreaManagerReservationRequest());
                break;
          }
          //

          //Shut down behaviour
                addBehaviour(new BRestAreaManagerShutDown());
                //
        }
        //**************************************************************************************
************//
        //**************************************************************************************
************//


        //**************************************************************************************
************//
        //**************************************************************************************
************//
        // Override
                                                                        //
        //**************************************************************************************
************//
        //**************************************************************************************
************//
        @Override
        protected void takeDown() {
                while (receive() != null) {};
        }
        //**************************************************************************************
************//
        //**************************************************************************************
************//


        //**************************************************************************************
************//
        //**************************************************************************************
************//
        // Borda Negotiation Behavious: BRestAreaMessageReceiver,
BRestAreaManagerOccupationRequest,//
```

```
	//
	BRestAreaManagerReservationRequest,                                    //
	//
	BRestAreaManagerNegotiateForRelocation                                 //
	//*************************************************************************
***********//
	//*************************************************************************
***********//
	//This behaviour allows to create the behaviours needed
	private class BRestAreaMessageReceiver extends CyclicBehaviour {

		public void action(){
			ACLMessage msg =
receive(MessageTemplate.MatchConversationId("occupation"));
			if (msg==null){
				block();
				return;
			}

	    try {
		ContentElement content = getContentManager().extractContent(msg);
		   Concept action = ((Action) content).getAction();
		   switch (msg.getPerformative()) {
			case (ACLMessage.REQUEST):
				if (action instanceof OccupationRequest) {
					addBehaviour(new BRestAreaManagerOccupationRequest(msg));
				}
			break;
		   }
		}
		catch(Exception ex) { ex.printStackTrace(); }
		}
	}

	//This behaviour allos to create a reservation
	private class BRestAreaManagerReservationRequest extends Behaviour {

		int step = 0;
		MessageTemplate mtRequestReservation =
MessageTemplate.MatchPerformative(ACLMessage.REQUEST);
		AID journeyManager;
		String replyCode;
		String msgContent;
		int numResp = 0;
		ArrayList<AID> freeParkingSpots = new ArrayList<AID>();
		ArrayList<AID> occupiedParkingSpots = new ArrayList<AID>();
		ArrayList<AID> driversToNegotiate = new ArrayList<AID>();
		int time = 0, bestTime = -999999;
		AID bestParkingSpot;
		Calendar timeStart, timeEnd;
```

```
		String journeyManagerId;


		@Override
		public void action() {
			switch (step){
			case 0:
				//Reading the message
				ACLMessage msg =
receive(MessageTemplate.and(MessageTemplate.MatchPerformative(ACLMessage.REQUEST),MessageTe
mplate.MatchConversationId("request")));
				if (msg != null) {
					replyCode = msg.getInReplyTo();
					journeyManager = msg.getSender();
					try {
						ContentElement content =
getContentManager().extractContent(msg);
						ReservationRequest reservationRequest =
(ReservationRequest) ((Action) content).getAction();
						Reservation reservation =
reservationRequest.getReservation();
						journeyManagerId = reservation.getjourneyManagerId();
						timeStart = Calendar.getInstance();
						timeEnd = Calendar.getInstance();
						timeStart.setTime(reservation.getTimeStart());
						timeEnd.setTime(reservation.getTimeEnd());
					} catch (CodecException | OntologyException e) {
						// TODO Auto-generated catch block
						e.printStackTrace();
					}
					step++;
				}
				else block();
				break;
			case 1:
				//Asking to parking spots if they're available
				ACLMessage msgCFPParkingSpot = new ACLMessage(ACLMessage.CFP);
				msgCFPParkingSpot.setLanguage(codec.getName());
				msgCFPParkingSpot.setOntology(ontoloy.getName());

				//Content of the msg
				Reservation reservation = new Reservation();
				reservation.setTimeStart(timeStart.getTime());
				reservation.setTimeEnd(timeEnd.getTime());
				reservation.setjourneyManagerId(journeyManagerId);
				ReservationRequest reservationRequest = new ReservationRequest();
				reservationRequest.setReservation(reservation);
				//The action must be "stored" in an standard "Action" object, instead of a
"custom" AgentAction
				action = new Action(getAID(),reservationRequest);
```

```
                        try {
                                getContentManager().fillContent(msgCFPParkingSpot, action);
                        } catch (CodecException | OntologyException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }

                        for (AID parkingSpot : parkingSpotList) {
                                msgCFPParkingSpot.addReceiver(parkingSpot);
                        }
                        msgCFPParkingSpot.setInReplyTo(replyCode);
                        msgCFPParkingSpot.setConversationId("request");
                        myAgent.send(msgCFPParkingSpot);
                        step++;
                        break;
                case 2:
                        //Checking if some parking spot is available
                        MessageTemplate mtCFPReply =
MessageTemplate.MatchInReplyTo(replyCode);

                        ACLMessage msgCFPReply = myAgent.receive(mtCFPReply);

                        if (msgCFPReply != null) {
                                if (msgCFPReply.getPerformative()==ACLMessage.PROPOSE) {
                                        freeParkingSpots.add(msgCFPReply.getSender());
                                        time = Integer.parseInt(msgCFPReply.getContent());

                                        if (time > bestTime) {
                                                bestTime = time;
                                                bestParkingSpot = msgCFPReply.getSender();
                                        }
                                }
                                //The ones who doesn't have free spot, would reply with the local
name of the journey manager agent involved
                                else {
                                        driversToNegotiate.add(new
AID(msgCFPReply.getContent(),AID.ISLOCALNAME));
                                        occupiedParkingSpots.add(msgCFPReply.getSender());
                                }
                                numResp++;
                                if (numResp>=parkingSpotList.size()) step++;
                        } else block();
                        break;
                case 3:
                        //If there is a parking spot available, accept the proposal
                        if (bestParkingSpot!= null) {
                                ACLMessage response = new
ACLMessage(ACLMessage.REJECT_PROPOSAL);
                                response.setInReplyTo(replyCode);
```

```
                                        //Send reject messages
                                        for (AID parkingSpot : parkingSpotList)
                                                if
(!parkingSpot.getLocalName().equals(bestParkingSpot.getLocalName()))
                                                        response.addReceiver(parkingSpot);
                                        myAgent.send(response);

                                        //Send accept message
                                        response = new ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
                                        response.setInReplyTo(replyCode);
                                        response.addReceiver(bestParkingSpot);
                                        response.setContent(journeyManager.toString());
                                        myAgent.send(response);

                                        //Tell to the Journey Manager that his reservation has been done
                                        response = new ACLMessage(ACLMessage.AGREE);
                                        response.setInReplyTo(replyCode);
                                        response.addReceiver(journeyManager);
                                        myAgent.send(response);
                                }
                                else {
                                        //If there is not an available parking spot, the negotiation
behavior will start
                                        addBehaviour(new
BRestAreaManagerNegotiateForRelocation(journeyManager, replyCode, msgContent,
driversToNegotiate, occupiedParkingSpots));
                                }
                        step++;
                        break;
                        }
                }

                @Override
                public boolean done() {
                        if (step == 4) {
                                addBehaviour(new BRestAreaManagerReservationRequest());
                                return true;
                        }
                        return false;
                }

        }

        //This behaviour return if has or hasen't any free parking spot
        private class BRestAreaManagerOccupationRequest extends Behaviour {

                int step = 0;
                MessageTemplate mtRequestOccupation =
MessageTemplate.MatchPerformative(ACLMessage.REQUEST);
                AID journeyManager;
```

```
		String replyCode;
		String journeyManagerId;
		Calendar timeStart, timeEnd;
		int numResp = 0;
		ACLMessage msg;
		boolean freeSpot;


		public BRestAreaManagerOccupationRequest(ACLMessage msg) {
			// TODO Auto-generated constructor stub
			this.msg = msg;
		}

		@Override
		public void action() {
			switch (step){
			case 0:
				replyCode = msg.getInReplyTo();
				journeyManager = msg.getSender();
				try {
					ContentElement content =
getContentManager().extractContent(msg);
					OccupationRequest occupationRequest = (OccupationRequest)
((Action) content).getAction();
					Reservation reservation = occupationRequest.getReservation();
					journeyManagerId = reservation.getjourneyManagerId();
					timeStart = Calendar.getInstance();
					timeEnd = Calendar.getInstance();
					timeStart.setTime(reservation.getTimeStart());
					timeEnd.setTime(reservation.getTimeEnd());
				} catch (CodecException | OntologyException e) {
					// TODO Auto-generated catch block
					e.printStackTrace();
				}
				step++;
				break;
			case 1:
				//Asking to parking spots if they're available
				ACLMessage msgCFPParkingSpot = new
ACLMessage(ACLMessage.REQUEST);
				for (AID parkingSpot : parkingSpotList) {
					msgCFPParkingSpot.addReceiver(parkingSpot);
				}

				msgCFPParkingSpot.setLanguage(codec.getName());
				msgCFPParkingSpot.setOntology(ontoloy.getName());

				//Content of the msg
				Reservation reservation = new Reservation();
				reservation.setTimeStart(timeStart.getTime());
```

```
                        reservation.setTimeEnd(timeEnd.getTime());
                        reservation.setjourneyManagerId(journeyManagerId);
                        OccupationRequest occupationRequest = new OccupationRequest();
                        occupationRequest.setReservation(reservation);
                        //The action must be "stored" in an standard "Action" object, instead of a
"custom" AgentAction
                        action = new Action(getAID(),occupationRequest);
                        try {
                                getContentManager().fillContent(msgCFPParkingSpot, action);
                        } catch (CodecException | OntologyException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }

                        msgCFPParkingSpot.setInReplyTo("occupation");
                        msgCFPParkingSpot.setConversationId("request");
                        myAgent.send(msgCFPParkingSpot);
                        step++;
                        break;
                case 2:
                        //Checking if some parking spot is available
                        MessageTemplate mtCFPReply =
MessageTemplate.MatchInReplyTo("occupation");
                        ACLMessage msg_cfp_reply = myAgent.receive(mtCFPReply);
                        freeSpot = false;

                        if (msg_cfp_reply != null) {
                                if (msg_cfp_reply.getPerformative()==ACLMessage.AGREE) {
                                        freeSpot = true;
                                }
                                numResp++;
                                if (numResp>=parkingSpotList.size()) step++;
                        } else block();
                        break;
                case 3:
                        //If there is a parking spot available, accept the proposal
                        ACLMessage response;
                        if (freeSpot) response = new ACLMessage(ACLMessage.AGREE);
                        else response = new ACLMessage(ACLMessage.REFUSE);
                        //Send accept message
                        response.setInReplyTo(replyCode);
                        response.addReceiver(journeyManager);
                        myAgent.send(response);
                        step++;
                        break;
                }
        }

        @Override
        public boolean done() {
```

```
                if (step == 4) return true;
                return false;
            }

        }


        private class BRestAreaManagerNegotiateForRelocation extends Behaviour {

            int step = 0;
            int numResp = 0;
            MessageTemplate mtCFPReply;
            MessageTemplate mtConfirmReservation;
            ArrayList<AID> driversToNegotiate = new ArrayList<AID>();
            ArrayList<AID> occupiedParkingSpots = new ArrayList<AID>();
            ArrayList<AID> driversToChange = new ArrayList<AID>();
            ArrayList<Float> proposedTimes = new ArrayList<Float>();
            AID bestDriver;
            float bestChange = 99999;
            AID journeyManager;
            String replyCode;
            String msgContent;
            boolean newReservationCompleted = false;
            boolean waitingForResponse = false;
            //Data for the log
            int affectedVehicles = 0;
            int relocableVehicles = 0;
            float relocatedVehiclesLostTime = 0;
            int relocatedVehicleOrder = 0;
            Calendar relocationTime = Calendar.getInstance();
            boolean success = false;


            public BRestAreaManagerNegotiateForRelocation(AID journeyManager1, String
replyCode1, String msgContent1, ArrayList<AID> driversToNegotiate1, ArrayList<AID>
occupiedParkingSpots1) {
                journeyManager = journeyManager1;
                replyCode = replyCode1;
                msgContent = msgContent1;
                driversToNegotiate = driversToNegotiate1;
                affectedVehicles = driversToNegotiate.size();
                occupiedParkingSpots = occupiedParkingSpots1;
            }

            @Override
            public void action() {
                switch (step){
                case 0:
                    ACLMessage msgCFPDriverToNegotiate = new
ACLMessage(ACLMessage.CFP);
```

```
//Including journeyManager to the negotiation
driversToNegotiate.add(journeyManager);
occupiedParkingSpots.add(null); //The journey manager who ask for a
place, hasn't parking spot yet
for (AID driver : driversToNegotiate) {
	msgCFPDriverToNegotiate.addReceiver(driver);
}
msgCFPDriverToNegotiate.setContent(getAID().toString());
msgCFPDriverToNegotiate.setInReplyTo(replyCode);
myAgent.send(msgCFPDriverToNegotiate);

step++;
break;
case 1:
//Checking the drivers avaliable
mtCFPReply = MessageTemplate.MatchInReplyTo(replyCode);

ACLMessage msgCFPDriverReply = myAgent.receive(mtCFPReply);

if (msgCFPDriverReply != null) {
	if (msgCFPDriverReply.getPerformative()==ACLMessage.PROPOSE)
{
		relocableVehicles++;
		driversToChange.add(msgCFPDriverReply.getSender());
		float time =
Float.parseFloat(msgCFPDriverReply.getContent());
		proposedTimes.add(time);
		if (time < bestChange) {
			bestChange = time;
			bestDriver = msgCFPDriverReply.getSender();
		}
	}
	numResp++;
	if (numResp>=driversToNegotiate.size()) step++;
} else block();
break;
case 2:
//We got at least 1 driver willing to change their occupation
if (bestDriver!= null) {
	//If the message for the new reservation hasn't been sent...
	if (!waitingForResponse && !newReservationCompleted) {
		ACLMessage response = new
ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
		//Send accept message
		response.setInReplyTo(replyCode);
		response.addReceiver(bestDriver);
		myAgent.send(response);
		relocatedVehicleOrder++;
		waitingForResponse = true;
	}
```

```
                                        //If the message for the new reservation has been sent
                                        if (waitingForResponse && !newReservationCompleted)  {
                                                mtConfirmReservation =
MessageTemplate.MatchInReplyTo(replyCode);

                                                ACLMessage msgCFPDriverReservation =
myAgent.receive(mtConfirmReservation);

                                                if (msgCFPDriverReservation != null) {
                                                        //If everything is ok, the parking spot must be
freed
                                                        if (msgCFPDriverReservation.getPerformative() ==
ACLMessage.INFORM){
                                                                newReservationCompleted = true;
                                                                int index =
driversToNegotiate.indexOf(bestDriver);
                                                                ACLMessage msgCancellation = new
ACLMessage(ACLMessage.REQUEST);

        msgCancellation.addReceiver(occupiedParkingSpots.get(index));

        msgCancellation.setLanguage(codec.getName());

        msgCancellation.setOntology(ontoloy.getName());

                                                                //Content of the msg
                                                                Reservation reservation = new
Reservation();

        reservation.setjourneyManagerId(bestDriver.getLocalName());

                                                                Calendar foo = Calendar.getInstance();
                                                                reservation.setTimeStart(foo.getTime());
                                                                reservation.setTimeEnd(foo.getTime());
                                                                CancelRequest cancelRequest = new
CancelRequest();

        cancelRequest.setReservation(reservation);
                                                                //The action must be "stored" in an
standard "Action" object, instead of a "custom" AgentAction
                                                                action = new
Action(getAID(),cancelRequest);
                                                                try {

        getContentManager().fillContent(msgCancellation, action);
                                                                } catch (CodecException |
OntologyException e) {
                                                                        // TODO Auto-generated catch
block
                                                                        e.printStackTrace();
```

```
                                        }

                                        myAgent.send(msgCancellation);
                                    }
                                    //If not, we repeat the process
                                    else {
                                        //If this was the last driver, the
negotiation fails
                                        if (driversToChange.isEmpty()) {
                                            break;
                                        }
                                        //Remove the driver that couldn't make
the reservation
                                        int index =
driversToChange.indexOf(bestDriver);
                                        proposedTimes.remove(index);
                                        driversToChange.remove(index);

                                        //Checking for the new best time
                                        float time = 99999;
                                        int bestIndex = 0;
                                        index = 0;

                                        for (float newBestTime : proposedTimes) {
                                            if (time > newBestTime) {
                                                time = newBestTime;
                                                bestIndex = index;
                                            }
                                            index++;
                                        }
                                        bestChange = time;

                                        bestDriver =
driversToChange.get(bestIndex);
                                        waitingForResponse = false;
                                    }
                                } else block();
                            }

                            //Send reject message
                            if (newReservationCompleted) {
                                ACLMessage rejectDrivers = new
ACLMessage(ACLMessage.REJECT_PROPOSAL);
                                rejectDrivers.setInReplyTo(replyCode);
                                for (AID parkingSpot : driversToChange)
                                    if
(!parkingSpot.getLocalName().equals(bestDriver.getLocalName()))
                                        rejectDrivers.addReceiver(parkingSpot);
                                myAgent.send(rejectDrivers);
                                relocatedVehiclesLostTime = bestChange;
```

```
                                        success = true;
                                        step++;
                                }
                        }
                        else {
                                //If there is not an available parking spot, the negotiation is over
and the driver that initiated the negotiation must seek for another rest area
                                ACLMessage response = new ACLMessage(ACLMessage.REFUSE);
                                response.setInReplyTo(replyCode);
                                response.addReceiver(journeyManager);
                                myAgent.send(response);
                                success = false;
                                step++;
                                break;
                        }
                }
        }

        @Override
        public boolean done() {
                if (step == 3){
                        ACLMessage msgLogRelocation = new
ACLMessage(ACLMessage.INFORM);
                        msgLogRelocation.setLanguage(codec.getName());
                        msgLogRelocation.setOntology(ontoloy.getName());

                        LogRelocation logRelocation = new LogRelocation();

                        logRelocation.setAffectedVehicles(affectedVehicles);
                        logRelocation.setRelocableVehicles(relocableVehicles);
                        logRelocation.setRelocatedVehicleLostTime(relocatedVehiclesLostTime);
                        logRelocation.setRelocatedVehicleOrder(relocatedVehicleOrder);
                        logRelocation.setRelocationTime(relocationTime.getTime());
                        logRelocation.setSuccess(success);

                        //The action must be "stored" in an standard "Action" object, instead of a
"custom" AgentAction
                        action = new Action(getAID(),logRelocation);
                        try {
                                getContentManager().fillContent(msgLogRelocation, action);
                        } catch (CodecException | OntologyException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                        msgLogRelocation.setConversationId("log");
                        msgLogRelocation.addReceiver(logManagerAID);
                        send(msgLogRelocation);

                        return true;
                }
```

```
                        return false;
                }
        }
        //**************************************************************************************
**********//
        //**************************************************************************************
**********//


        //**************************************************************************************
**********//
        //**************************************************************************************
**********//
        // Simulation without negotiation behaviour: BRestAreaManagerReservationFree
                //
        //
BRestAreaManagerSetSpotFree                                     //
        //**************************************************************************************
**********//
        //**************************************************************************************
**********//
        private class BRestAreaManagerReservationFree extends Behaviour {

                int step = 0;
                AID journeyManager;
                String replyCode;

                @Override
                public void action() {
                        switch (step){
                        case 0:
                                //Reading the message
                                ACLMessage msg =
receive(MessageTemplate.and(MessageTemplate.MatchPerformative(ACLMessage.REQUEST),MessageTe
mplate.MatchConversationId("requestFree")));
                                if (msg != null) {
                                        replyCode = msg.getInReplyTo();
                                        journeyManager = msg.getSender();
                                        step++;
                                }
                                else block();
                                break;
                        case 1:
                                //Checking if some parking spot is available
                                if (numOccupiedParkingSpots < numParkingSpots) {
                                        ACLMessage msgResponse = new
ACLMessage(ACLMessage.AGREE);
                                        msgResponse.setInReplyTo(replyCode);
                                        msgResponse.addReceiver(journeyManager);
                                        numOccupiedParkingSpots++;
```

```
                                        myAgent.send(msgResponse);
                                }
                                else {
                                        ACLMessage msgResponse = new
ACLMessage(ACLMessage.REFUSE);
                                        msgResponse.setInReplyTo(replyCode);
                                        msgResponse.addReceiver(journeyManager);
                                        myAgent.send(msgResponse);
                                }
                                step++;
                                break;
                        }
                }

                @Override
                public boolean done() {
                        if (step == 2) {
                                addBehaviour(new BRestAreaManagerReservationFree());
                                return true;
                        }
                        return false;
                }
        }

        private class BRestAreaManagerSetSpotFree extends Behaviour {
                int step = 0;

                @Override
                public void action() {
                        switch (step){
                        case 0:
                                //Reading the message
                                ACLMessage msg =
receive(MessageTemplate.and(MessageTemplate.MatchPerformative(ACLMessage.REQUEST),MessageTe
mplate.MatchConversationId("requestSetSpotFree")));
                                if (msg != null) {
                                        numOccupiedParkingSpots--;
                                        step++;
                                        break;
                                }
                        }
                }

                @Override
                public boolean done() {
                        if (step == 1) {
                                addBehaviour(new BRestAreaManagerSetSpotFree());
                                return true;
                        }
                        return false;
```

```
		}
	}
	//**************************************************************************************
**********//
	//**************************************************************************************
**********//


	//**************************************************************************************
**********//
	//**************************************************************************************
**********//
	// General behaviours: BRestAreaManagerShutDown
	                                        //
	//**************************************************************************************
**********//
	//**************************************************************************************
**********//
	private class BRestAreaManagerShutDown extends Behaviour {

		@Override
		public void action() {
			// TODO Auto-generated method stub
			MessageTemplate mtInformShutDown =
MessageTemplate.and(MessageTemplate.MatchPerformative(ACLMessage.INFORM),
MessageTemplate.MatchConversationId("shutDown"));

			ACLMessage msgInformUpdate = myAgent.receive(mtInformShutDown);
			if (msgInformUpdate != null) {
				myAgent.doDelete();
			}
			}

		@Override
		public boolean done() {
			// TODO Auto-generated method stub
			return false;
		}
	}
	//**************************************************************************************
**********//
	//**************************************************************************************
**********//

}
```