Práctica 3

SID - Q2 2019/2020

Marc Monfort Grau

Marcos Gomez Vazquez

16 de Junio de 2020

Introducción **3**

Caso de uso **3**

Ontología **4**

Sistema Multiagente **4**

Agente Entorno: 6

OntologyParser: 6

Agente Vehículo: 7

Agente Semáforo: 7

Agente Cloud: 8

Investigación y implementació de extensiones **10**

Uso de mecanismos avanzados de comunicación 10

Uso de mecanismos avanzados de negociación 12

Uso de mecanismos avanzados de coordinación 15

Metodología de trabajo **17**

Bibliografía **18**

Introducción

En esta práctica implementaremos un sistema multiagente para simular la tecnología Edge Computing en un subconjunto del caso de uso detallado en la práctica anterior. Utilizaremos la misma Ontología pero añadiendo algunas propiedades más y las instancias necesarias para representar el escenario y sus agentes.

Haremos uso de la tecnología JADE para implementar el sistema multiagente y su comunicación usando protocolos FIPA y el componente DF, y también usaremos JENA para obtener la información de la ontología

Caso de uso

El caso de uso implementado en esta práctica es un subconjunto del caso de uso detallado en la anterior práctica sobre **Edge Computing**. Hemos intentado buscar un equilibrio para un caso de uso sencillo y a la vez interesante.

Representaremos al entorno por un **conjunto de calles** conexas con posibles intersecciones. Cada calle será de un único carril y sentido.

En cada intersección entre dos calles donde puedan existir conflictos (un vehículo entrando a una vía principal) tendremos un **agente semáforo** (nivel Edge) para controlar el flujo de vehículos entre las dos calles. Tendrá un comportamiento binario. Para poder abrir el paso a una calle tendrá que cerrar el paso a la otra calle. Los semáforos permanecerán estáticos hasta que algún vehículo que llegue a él por la calle cerrada le solicite ponerse en verde.

En el escenario también aparecerán **agentes vehículos** (nivel Usuario) que podrán moverse por las calles siguiendo la dirección correcta y respetando los semáforos. También evitarán los accidentes con otros vehiculos y mantendrán una distancia de seguridad. Los vehículos podrán comunicarse con los semáforos que les cierren el paso para pedir que lo abran y les dejen circular. Además cada vehículo tiene como objetivo llegar a su destino.

Finalmente tendremos un único **agente Cloud** (nivel Cloud) que servirá para comunicar y coordinar los semáforos.

Ontología

Hemos reutilizado la ontología ya diseñada en la práctica anterior. Pero para adaptarla al caso de uso hemos añadido más propiedades que nos han facilitado obtener la información del entorno. Por ejemplo hemos añadido la relación **cierraPasoA** para saber la calle que cierra un determinado semáforo, o el atributo **tieneDireccion** para saber la dirección de una calle. Además mediante la propiedad ya definida anteriormente **ocurreEn** hemos indicado en qué calle se encuentra cada agente semáforo y vehículo. También hemos añadido una instancia para cada calle y agente.

(Adjuntamos la ontología modificada a la entrega)

Sistema Multiagente

El **escenario planteado** consiste en cuatro manzanas con cuatro calles alrededor y dos calles que interseccionan dibujando una cruz. Tendremos tres semáforos, uno para cada intersección conflictiva (donde un vehículo puede incorporarse en mitad de otra calle). Habran X vehiculos repartidos por el escenario, cada uno con su objetivo. El agente cloud se encargará de coordinar los tres semáforos mencionados, actuando como intermediario.

El escenario se representa por coordenadas enteras de un plano. Cada punto se considera como una posición que puede ocupar un vehículo. Los vehículos se van moviendo de punto en punto a una velocidad constante.

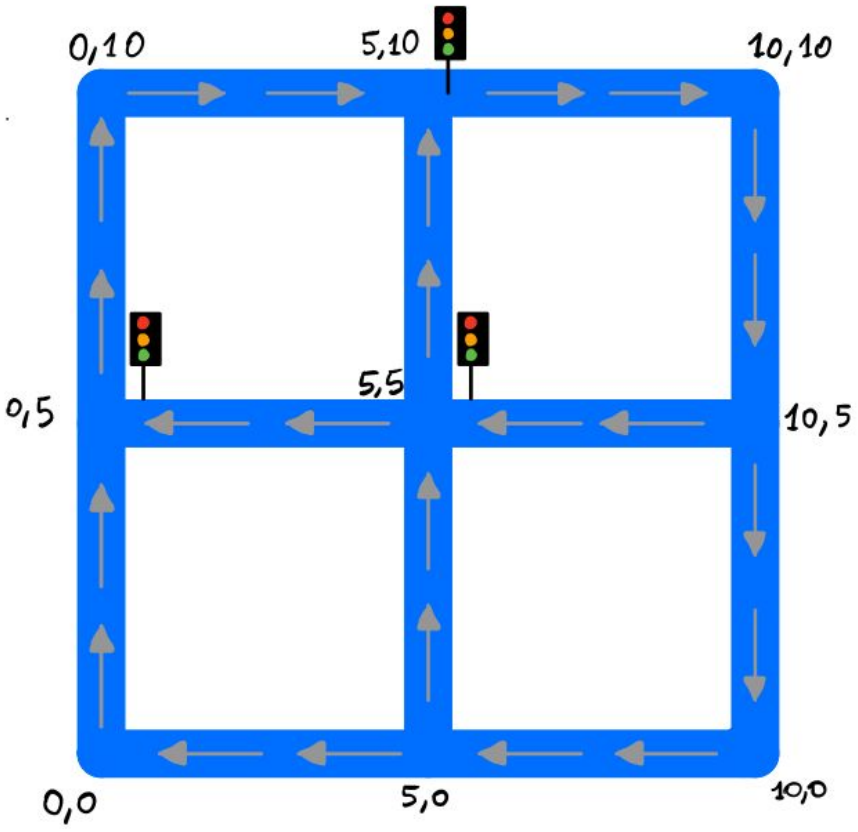

Figura 1: Escenario planteado para el caso de uso

Agente Entorno:

El agente entorno será encargado de mantener la información actualizada del propio entorno. Empezará inicializando el escenario obteniendo las instancias de las calles, los semáforos, los vehículos y del cloud. Utilizamos un clase auxiliar llamada **OntologyParser** donde están implementadas todas las funciones para extraer las instancias necesarias de la ontología usando JENA mediante consultas SPARQL.

A continuación creará cada uno de los agentes y los inicializará en la plataforma JADE. Para mantener el entorno de forma más realista, hemos pasado como argumento a cada agente una referencia al objeto java donde se mantiene su información en la clase Entorno. De esta forma, cada agente puede modificar su estado en el entorno de forma instantánea desde su propia clase y puede consultar el estado del entorno también de forma instantánea. El entorno es capaz de proporcionar el estado de la ciudad a los agentes de forma reactiva.

Finalmente el agente entorno incorpora un **TickerBehaviour** que muestra cada segundo por terminal la representación del estado del entorno.

OntologyParser:

La clase OntologyParser contiene métodos de consulta de instancias de la ontología. Estos métodos son utilizados por el agente Entorno para inicializarse. Los valores iniciales del entorno vienen dados por los datos de las instancias de la ontología.
Para obtener las instancias se realizan **consultas SPARQL** que obtienen todas las instancias de una clase en concreto. Una vez obtenidas, se extraen los valores de los **object properties** y **data properties** utilizados en este caso de uso (hay propiedades que pese a estar presentes en la ontología no utilizamos). Cuando finalmente se obtienen todos los datos de todos los objetos de un mismo tipo (semáforo / calle / vehículo), se devuelven al entorno en un HashMap de Java. Es de esos diccionarios de donde los agentes obtendrán la información del entorno, basándose en la técnica de

blackboard. Podrán acceder a un HashMap que contiene las calles, otro que contiene los vehículos y otro los semáforos.

Agente Vehículo:

Cada agente vehículo representa uno de los vehículos autónomos del escenario. Cuando son creados se inicializan en el DF con su nombre y con su tipo "Vehículo".

Incorporan un **TickerBehaviour** que cada segundo le permite actualizar su estado en el entorno y interactuar con el. En cada *tick* comprueba si ha llegado a su objetivo. Si es así, se le asigna aleatoriamente un nuevo objetivo (como si fuera un taxi).

También calcula la distancia al vehículo de delante más cercano (como si tuviera un cámara o radar delantero). Sí la distancia es inferior a la mínima el vehículo se parará hasta tener de nuevo suficiente distancia de seguridad. Sí dos vehículos están en el mismo punto se representa como un accidente (lo cual no debería suceder).

Cuando un vehículo llega al final de una calle, este continuará circulando por la calle sucesora actualizando su estado en el entorno (la calle en la que está). Si un vehículo se encuentra en una intersección entre dos calles, este decidirá sí incorporarse a la nueva calle si en esta se encuentra su objetivo, si no, continuara por la misma calle.

En el caso que un vehículo se encuentre delante de un semáforo en rojo, el vehículo mandará un mensaje al semáforo usando el **AchieveREInitiator** para pedir que le deje pasar. El vehículo esperará hasta que el semáforo se ponga de nuevo en verde.

Agente Semáforo:

Cada semáforo del entorno es manejado por un SemaforoAgent. Este agente se mantiene a la espera de posibles peticiones de un vehículo mediante el protocolo **FIPA Request.** Cuando un vehículo le pide abrir una calle, el semáforo le pide al agente Cloud, también mediante FIPA Request, que le pregunte a los otros semáforos cuál es el tiempo que ellos deben esperar para cambiar, teniendo en cuenta el tiempo que éste necesita. Ahí se produce una negociación. Los semáforos deberán ponerse de

acuerdo en qué tiempo esperan (ya que todos esperan el mismo tiempo, están sincronizados), teniendo en cuenta el tiempo que pide el semáforo que hace la petición y el de los demás, con el fin de llegar a un valor óptimo para todos los semáforos. Cuando acabe la negociación el agente cloud responderá a todos los semáforos con el tiempo de espera decidido. Una vez hecho esto, los agentes semáforo activarán un **Waker Behaviour** con la cuenta atrás qué decidieron entre todos. Cuando se activa el Waker Behaviour, antes de invertir las calles el semáforo bloquea las 2 calles durante 1 segundo representando el estado 'ámbar', en el cual los vehículos no pueden cruzar. Si no fuera por este estado, podría haber colisiones entre vehículos que cruzan a la vez. Después, se invierte la calle cerrada y la abierta.

El tiempo de espera de un semáforo aumenta por cada vehículo que se encuentra circulando por la calle abierta, y disminuye por cada uno que está esperando en la calle cerrada. Así, cuantos más vehículos haya esperando el semáforo podrá cambiar antes, y cuantos más vehículos haya circulando por la calle abierta el semáforo esperará más para cambiar, y estos dos conceptos se contrarrestan entre sí.

Agente Cloud:

El agente Cloud se hace el papel de mediador entre los semáforos. Cuando recibe una petición de negociación por parte de un semáforo, inicia un protocolo **FIPA Contract-net** con los otros semáforos, y bloquea posibles peticiones por parte de otros semáforos. Gracias a este bloqueo el cloud se asegura que cuando ya se ha hecho una petición de cambio de semáforos no se puedan realizar más hasta que los semáforos se inviertan (lo cual podría pasar si llega un segundo coche a otro semáforo en rojo).

Los semáforos proponen al cloud su tiempo de espera, y el cloud les responde con el valor más alto de todos. El hecho de que un semáforo tenga que esperar más tiempo del que debería no perjudica a la ejecución global, ya que garantiza un mejor funcionamiento de todo el sistema al dejar más tiempo para que otro semáforo sirva a los vehículos que lo cruzan. En la negociación a veces es necesario ceder para que el objetivo común se alcance.

Mediante el protocolo **FIPA Propose,** el cloud informa al semáforo que hizo la petición del tiempo decidido, y cuando este semáforo realiza el cambio de calles, envía un mensaje al cloud para que se abra nuevo a posibles peticiones de cualquier semáforo.

Investigación y implementació de extensiones

Uso de mecanismos avanzados de comunicación

Para un correcto desarrollo de nuestro caso de uso y de la mayoría de sistemas multiagente es esencia la comunicación entre los agentes. Los agentes se pueden comunican entre ellos para informar sobre su estado, compartir tareas, compartir objetivos, etc.

Los agentes adoptan un rol activo si inician la comunicación para obtener respuestas o afirmar hechos. Un rol pasivo si actúan de forma reactiva ante consultas de otros agentes respondiendo (o no) a estas. O pueden adoptar ambos roles.

Hemos considerado el estudio y implementación de la técnica ***blackboard*** y ***message passing***.

- **Blackboard:** La técnica *blackboard* [1] consiste en un método de comunicación que hace uso de una base de conocimiento compartida entre varios o todos los agentes. La base de conocimiento es llamada *blackboard* y permite la intercomunicación entre agentes con el objetivo de resolver un problema.

  Cada agente podrá consultar y analizar el *blackboard*. Con la información obtenida, el agente podrá extraer las conclusiones y hechos y escribir el resultado de su cambios de nuevo en la pizarra, donde será compartido con los demás agentes. Los otros agentes, de nuevo podrán usar los resultados y extraer y aportar más información. El proceso continúa hasta encontrar una solución al problema inicial.

  Es necesario utilizar un lenguaje común entre los agentes para poder entender los resultados y la información del *blackboard*.

[1] "Blackboard system - Wikipedia." https://en.wikipedia.org/wiki/Blackboard_system. Se consultó el 13 jun.. 2020.

La **ventaja** de este método de comunicación es que nos permite mucha abstracción del problema y de la forma de resolverlo.

Pero tiene como **desventaja** la posible ineficiencia de mantener el *blackboard* evitando problemas de concurrencia producidos por trabajar en una base de conocimiento compartida.

Implementación: En nuestra práctica hemos implementado un método *blackboard* para mantener la información de entorno de forma constante y compartida. Esta técnica permite que los agentes puedan saber al instante el estado de los otros agentes, y además pueden actualizar su propio estado de forma que los otros agentes también los pueden consultar al instante sin necesidad de comunicación entre agentes o de mandar mensajes al entorno de forma individual para consultarlo.

Para implementarlo hemos aprovechado las referencias a objetos en el lenguaje Java. De esta forma cada instancia de agente tiene la referencia a un objeto con su información que podrá modificar y además tendrá la referencia de otros objetos que representan el entorno (como calles y semáforos) y podrá consultar su estado sin la necesidad de usar al agente entorno como intermediario. Como todos los agentes irán actualizando su propia información en la referencia al objeto que les representa, todos tendrán la información del estado al instante.

- **Message passing:** El *message passing* es un método de comunicación entre agentes que consiste en enviar mensajes directamente de agente a agente. Se suele utilizar la propia plataforma que usa el sistema multiagente para transmitir los mensajes como si se tratara de un router. De nuevo hace falta un lenguaje común entre agentes para que puedan entenderse (por ejemplo FIPA-ACL).

  Este método tiene como **ventaja** su simplicidad ya que plataformas como JADE nos permite hacer uso de forma muy intuitiva.

Podríamos considerar como **desventaja** la tediosidad de algunos protocolos y la posible falta de precisión en los mensajes, pero es el mismo problema que sucede en la propia comunicación entre personas.

Implementación: Hemos aprovechado los protocolos FIPA ya implementados en JADE para que los agentes se comuniquen entre ellos. En una primera versión antes de utilizar el método *blackboard* los agentes actualizan su estado mediante mensajes *request* al entorno, y este respondía aceptando su nuevo estado. También estamos utilizando el protocolo *request* para que los vehículos manden un mensaje a los semáforos cuando éstos corten el paso. Al tener el entorno compartido no hace falta que el vehículo espere a recibir del semáforo cuando se pone verde, ya que está continuamente analizando el entorno y ya lo detectara cuando el semáforo cambie su estado en el entorno.

Uso de mecanismos avanzados de negociación

La negociación es una interacción de alto nivel entre agentes con un elevado nivel de abstracción, donde los objetivos de los agentes pueden estar en conflicto o simplemente ser objetivos opuestos. El objetivo de la negociación consiste en encontrar un acuerdo entre los agente. Para que tenga sentido la negociación, todos los agentes involucrados deben tener el objetivo compartido de llegar a algún acuerdo.

- **Contract-Net:** El *Contract-Net* es un protocolo FIPA de interacción usado en los sistemas multiagente, útil tanto en situaciones de cooperación (agentes con objetivos similares), o en situaciones de competición (agentes con objetivos distintos o opuestos). Este protocolo complejo permite negociar diferentes propuestas entre varios agentes para llevar a cabo una determinada acción.

  Consiste en varias fases, empezando por el agente iniciador del protocolo que envía al resto de agentes la información de una tarea a realizar. Los agentes que la reciben pueden escoger entre rechazar la petición o devolver una propuesta a esa petición. Entre todas las propuestas recibidas, el agente

iniciador deberá escoger una. Finalmente el agente que ha sido escogido para realizar la tarea deberá informar sobre el resultado de esta.

La **ventaja** de este protocolo respecto otros como el *request* es que facilitan mucho la negociación ya que si quisiéramos hacer lo mismo utilizando otro método nos sería muy engorroso.

Implementación: Hemos implementado el protocolo Contract-Net en el agente Cloud. Cuando un semáforo recibe la petición de un vehículo para abrirle el paso, este semáforo envía un mensaje al agente Cloud informando de la petición junto con el tiempo de espera que propone. Cuando el Cloud recibe el mensaje del semáforo, iniciara el protocolo Contract-Net informando sobre la necesidad de cambio de semáforos. Los semáforos proponen su tiempo en base a los coches que tienen en la calle abierta y la cerrada. Cuando el agente Cloud recibe las propuestas, responde a los semáforos con la propuesta más alta de todas.

- **Optimalidad de Pareto:** La *optimalidad de Pareto* [2] (o eficiencia de Pareto) es un concepto utilizado mayormente en economía pero con utilidad en la negociación de los sistemas multiagente.

  La optimalidad de Pareto es un situación que se produce cuando no se puede mejorar el objetivo o la situación de ningún agente sin por ello empeorar la de otro agente. Hay que tener en cuenta que una situación óptima de Pareto no significa que sea una situación justa para todos los agentes.

  También existen otras denominaciones a situaciones relacionadas con el concepto de optimalidad de Pareto. Por ejemplo una situación se considera *dominada* si es posible encontrar una situación mejor para algún agente sin que ningún otro quede perjudicado. Existen otros conceptos como *optimalidad*

[2] "Pareto efficiency - Wikipedia." https://en.wikipedia.org/wiki/Pareto_efficiency. Se consultó el 14 jun.. 2020.

debil de Pareto[3]*, obtimalidad restingida de Pareto*, etc. que se usan más en el ámbito de la economía.

La **ventaja** de encontrar la optimalidad de Pareto es clara, ya que conseguimos una situación inmejorable para ningún (sin empeorar la de ningún otro agente).

Pero como **desventaja** està situación de optimalidad no significa que sea una situación justa. Por ejemplo en la repartición de un pastel entre dos agentes, si un agente coge 3/4 de pastel y el otro únicamente 1/4, estamos ante una situación de optimalidad de Pareto, pero evidentemente no es una situación justa.

Implementación: Hemos intentado implementar el concepto de optimalidad de Pareto para el control de tráfico en los semáforos. La intención sería adaptar el tiempo de las luces de los semáforos para beneficiar la circulación de los vehículos sin perjudicar a otros vehículos. Aunque pueda parecer fácil, nos ha sido bastante complejo encontrar esta situación de optimalidad ya que es muy común la situación en que dos vehículos están continuamente en conflicto.

Para intentar aproximarnos lo máximo a la optimalidad de Pareto, en el protocolo *contract-net* mencionado anteriormente, cuando un semáforo solicita cambiar de color (al estar todos coordinados), los demás semáforos ofrecen un tiempo de espera en función del número de vehículos que se encuentran en la calle abierta antes del semáforo, y del número de vehículos que se encuentran esperando en la calle cerrada. Además teniendo en cuenta la distancia de estos vehículos al propio semáforo. El diseño de esta heurística permite equilibrar de alguna forma la espera de los vehículos y reducir el tiempo de espera de la mayoría de ellos, aproximándonos a esa optimalidad de Pareto deseada.

[3] "Weak Pareto efficiency - Wikipedia." https://en.wikipedia.org/wiki/Pareto_efficiency#Weak_Pareto_efficiency. Se consultó el 15 jun.. 2020.

Uso de mecanismos avanzados de coordinación

La coordinación es una característica básica de los sistemas multiagente que se basa en la interacción de alto nivel entre agentes con un elevado nivel de abstracción.

El **consenso** en un sistema distribuido de agentes (MAS) consisten en ponerse de acuerdo respecto una acción o resultado. Los **algoritmos de consenso** se basan en protocolos de interacción que permiten coordinar a varios agentes en busca de ese acuerdo (acción / resultado) mutuo.

- **Paxos:** *Paxos* es un algoritmo de consenso bastante complejo donde se diferencia tres roles: Los *proposers*, los *acceptors*, y los *learners*. El funcionamiento del protocolo consiste en varias fases. Empieza con los agentes *proposers* enviando a los agentes *acceptors* una propuesta con un valor y un identificador. A continuación los *acceptors* reciben la propuesta y se deben comprometer a no aceptar ninguna propuesta con un identificador inferior, si no lo han hecho ya. De esta forma el algoritmo va hacia delante y ya no se consideran propuestas anteriores. Si el *acceptor* sí que ha aceptado una propuesta anterior, deberá devolver el valor que aceptó al *proposer*.

  Cuando algún *proposer* recibe las suficientes promesas de un mínimo número de *acceptors* (suele ser la mitad), entonces envía la propuesta de su valor con un mensaje "accept request". Este tiene que ser más grande que cualquier valor que ya haya aceptado algún *acceptor*. Cuando el *acceptor* recibe el mensaje "accept request" entonces lo rechaza si ya se ha comprometido con otra propuesta con un identificador mayor, o lo acepta y envía el mensaje de aceptación al *proposer* y a todos los *learners*.

  Finalmente cuando los *learners* descubren que la mayoría de *accpetors* han aceptado la propuesta de un *proposer*, entonces se acaba el algoritmo y el valor propuesto de ese *proposer* será el valor de consenso entre los agentes.

  Este algoritmo tiene varias **ventajas** que lo hacen interesante para un sistema multiagente. Permite mensajes asíncronos, es tolerante a fallos en los agentes,

los mensajes se pueden perder o duplicar sin por ello afectar al funcionamiento del algoritmo, etc.

Aunque como ya hemos comentado, su **desventaja** sería la complejidad inherente que hace complicado entender su funcionamiento sin una lectura detallada.

Existe muchos otros algoritmos de consenso, entre ellos el llamado Raft que es parecido a Paxos pero menos complejo de entender[4].

Implementación: En nuestra práctica hemos implementado un algoritmo de consenso entre agentes más sencillo que el algoritmo Paxos pero que también permite a los agentes ponerse de acuerdo en un valor.

Este algoritmo está implementado para conseguir obtener un valor del tiempo de espera para los semáforos antes de cambiar de color. En nuestra implementación los semáforos se coordinan para abrir y cerrar las calles en el mismo momento. Por eso es necesario una forma de consenso para que todos los semáforos tengan el mismo valor.

El algoritmo consiste en una propuesta de cambio hecha por el agente Cloud a todos los semáforos (excepto uno que inicia el protocolo con el Cloud). Los semáforos después tienen que proponer un tiempo de espera dependiendo del número de vehículos que haya en sus calles (el tiempo aumenta por cada vehículo que se encuentra circulando por la calle abierta, y disminuye por cada uno que está esperando en la calle cerrada. Así, cuantos más vehículos haya esperando el semáforo podrá cambiar antes, y cuantos más vehículos haya circulando por la calle abierta el semáforo esperará más para cambiar, y estos dos conceptos se contrarrestan entre sí). Finalmente el agente Cloud actuando como líder se queda con el valor más grande y lo comunica a los demás semáforos que deberán aceptar ese valor. De esta forma todos los agentes obtienen el mismo valor.

[4] "Paxos & Raft Consensus Protocols: Complete ... - Blockonomi." 14 nov.. 2018, https://blockonomi.com/paxos-raft-consensus-protocols/. Se consultó el 15 jun.. 2020.

Metodología de trabajo

Para el desarrollo de esta práctica, debido al confinamiento COVID-19 nos hemos coordinado gracias a servicios de videollamada.

La parte más complicada nos ha sido determinar el subconjunto del caso de uso a implementar, ya que hemos tenido que buscar un equilibrio entre un caso de uso sencillo y a la vez interesante. Finalmente nos hemos quedado con un subconjunto que nos ha parecido acceptable.

El desarrollo del código ha sido progresivo y lo hemos ido trabajando en paralelo utilizando un repositorio en común y la herramienta *Live Share* del editor VSCode. Nos hemos dividido trabajo, cada uno implementando uno de los agentes en mayor medida pero colaborando en el desarrollo de los demás agentes, ya que estos están compartiendo el mismo entorno y se deben comunicar entre ellos.

No hemos tenido ningún conflicto importante, solo algunas diferencias en la presentación por terminal del estado del entorno (el color de los vehículos) y de permitir la posibilidad de que ocurran accidentes (finalmente es posible cuando un vehículo se incorpora en una calle principal, pero al implementar el semáforo en ámbar es altamente improbable [y puede que imposible, sin contemplar errores de concurrencia]).

Bibliografía

- "Blackboard system - Wikipedia." https://en.wikipedia.org/wiki/Blackboard_system. Se consultó el 13 jun.. 2020.
- "Pareto efficiency - Wikipedia." https://en.wikipedia.org/wiki/Pareto_efficiency. Se consultó el 14 jun.. 2020.
- "Paxos & Raft Consensus Protocols: Complete ... - Blockonomi." 14 nov.. 2018, https://blockonomi.com/paxos-raft-consensus-protocols/. Se consultó el 15 jun.. 2020.
- Apuntes de "Sistemas Inteligentes Distribuidos" Asignatura de la Facultad de Informática de Barcelona, UPC.