# Conducción autónoma y calibración de dispositivos en un entorno Smart City/F2C

Ilyas el Idrissi Achahbar y Adrián Rivas López

Escola Politècnica Superior d'Enginyeria de Vilanova i la Geltrú

## Resumen

Las ciudades son el lugar donde más han impactado las tecnologías, ya que es en estas donde la mayoría de sus ciudadanos realizan sus actividades diarias, como por ejemplo ir al trabajo, hacer la compra, etc.

De este gran incremento tecnológico surge la necesidad de que los servicios sean cada vez más eficientes con la finalidad de poder servir al mayor número de personas posibles. Para hacer esto posible, se requiere de un gran número de recursos, que gestionados de la manera conveniente faciliten el desarrollo de tales servicios.

Basándonos en estas necesidades, aparece el concepto de Smart City. Una ciudad inteligente (Smart City en inglés) tiene como objetivo principal poder utilizar la tecnología de la información y comunicación (TIC) con la finalidad de mejorar el nivel de vida de los ciudadanos que habitan en esta.

Para el correcto desarrollo de estos servicios que una ciudad inteligente debe proporcionar, se ha de introducir el concepto de Fog to Cloud (F2C). El F2C pretende acercar las capacidades de las cuales dispone el Cloud a los usuarios, con la finalidad de obtener un tiempo de comunicación mucho más reducido que el que habría en una comunicación directa con el Cloud.

En este proyecto, se pretende llevar a cabo el desarrollo de una conducción autónoma la cual utilice las diferentes características proporcionadas por F2C en un entorno de ciudad inteligente y desarrollar un sistema de calibración de los diferentes dispositivos de esta ciudad inteligente, con la finalidad de cerciorarnos de que el funcionamiento de estos es el correcto, de una manera sencilla y eficaz.

*Palabras clave:* Smart City, Fog to Cloud, Cloud, servicio, dispositivo, ciudad inteligente.

## 1. Introducción

Actualmente, nos encontramos en un continuo avance de las nuevas tecnologías en el ámbito de la conducción, así como un gran crecimiento de los dispositivos interconectados que hay en las ciudades inteligentes. Gracias a este avance, se presenta la posibilidad de crear un sistema de conducción autónoma el cual introduzca ciertos beneficios para el conjunto de personas que dispongan de ella, así como un uso óptimo de las diferentes variables inteligentes que podemos encontrar en una ciudad inteligente. La idea del proyecto surge a raíz de uno del grupo de investigación multidisciplinario CRAAX (Centre d'Arquitectures Avançades de Xarxa) [1], el cual se encuentra trabajando activamente en el desarrollo de tecnologías relacionadas con el transporte inteligente, la salud electrónica y las ciudades, entre las cuales se encuentra un proyecto llamado mF2C [2].

Este equipo dispone de un laboratorio en el cual se ha diseñado e implementado un banco de pruebas (Testbed), el cual es utilizado para realizar ensayos de diferentes servicios que podrían ser emulados en una ciudad inteligente real. Para poder hacer estas emulaciones, este Testbed dispone de diferentes dispositivos como semáforos, farolas, vehículos, etc.

El objetivo principal de este proyecto es el diseño y desarrollo de un sistema de conducción autónoma capaz de utilizar la información proporcionada por diferentes dispositivos inteligentes de una ciudad inteligente para de esta manera añadir inteligencia a la propia conducción, inteligencia que hoy en día no existe en los sistemas de conducción autónoma actuales. También se pretende diseñar y desarrollar un sistema de calibración de los diferentes dispositivos inteligentes de una ciudad inteligente con la finalidad de poder tener controlado en todo momento el estado de estos y asegurar su correcto funcionamiento dentro de la ciudad inteligente.

Como objetivo adicional de este proyecto, se introduce el diseño e implementación de un agent a nivel software, así como cada uno de sus módulos con la finalidad de poder crear una topología F2C sobre la cual se puedan ejecutar diferentes servicios.

## 2. Escenario del proyecto

A raíz de las diferentes actividades de desarrollo en las que se encuentra actualmente el equipo CRAAX, se vieron con la necesidad de desarrollar el Testbed, para testear todas sus innovaciones en los ámbitos anteriormente nombrados.

El Testbed tiene unas dimensiones de 15 metros cuadrados. Dispone de 3 calles verticales y 2 horizontales. A parte cuenta con un puente levadizo y una serie de edificios que representan, un hospital, una estación de bomberos y el centro de tecnología *Neàpolis*, situado en Vilanova i la Geltrú, lugar donde se encuentra el laboratorio del CRAAX. El Testbed ha sido diseñado e implementado por otro equipo que ha realizado un TFG con nombre "Diseño e implementación de un Testbed para una SmartCity" [3].

*Figura 1: Visión general del Testbed*

El Front-End es la parte gráfica de una plataforma desarrollada e implementada por un estudiante que ha realizado el TFG nombrado "Panel Front-End para el control de una Smart City" [4]. En el contexto de este proyecto, el Front-End será utilizado para visualizar el estado de diferentes componentes del Testbed en la interfaz que este nos proporciona en tiempo real.

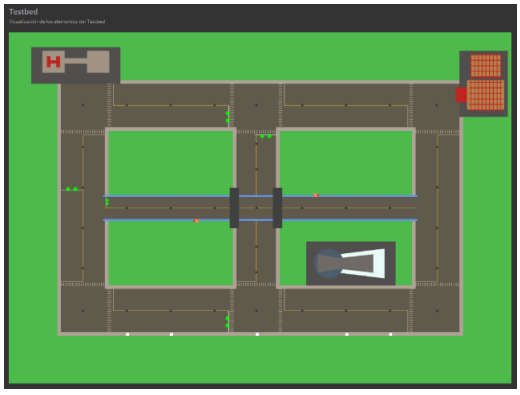


*Figura 2: Interfaz con el mapa del Testbed*

Dentro de este escenario también es necesario introducir un agent. Un agent es un componente que permite añadir inteligencia a ciertos elementos que formen parte de una ciudad inteligente. A parte, también aporta una capacidad de computación, que puede ser utilizada para ejecutar servicios de la ciudad inteligente. Se ha decidido esta estructura de un agent basándonos en el TFG realizado por un alumno con nombre "Multi Inteligent Agent Unit per a una Smart City" [4].

## 3. Requisitos funcionales

Este apartado tiene como objetivo identificar los diferentes requisitos necesarios para poder llevar a cabo el desarrollo del proyecto. Estos requisitos son identificados a partir de diferentes necesidades del cliente que harán que la solución obtenida se adapte completamente a estas. El listado de requisitos funcionales de cada uno de los diferentes apartados a desarrollar en este proyecto son los siguientes:

Agent:

- Tiene que ser capaz de comunicarse con otros agents de la misma topología de red.
- Ha de ser capaz de poder comunicarse con otros elementos que se encuentren dentro de la misma topología de red.
- La estructura de un agent tiene que estar formada por diferentes módulos, definidos por el cliente, cada uno de ellos independiente de los demás.
- Debe ser capaz de gestionar servicios, es decir, poder recibir, ejecutar y solicitar estos.

Panel de administración (Front-End):

- Ha de permitir la visualización de todos los nodos que se encuentren registrados en la base de datos topológica.
- Debe de poder visualizar todos los servicios del catálogo de servicios de la ciudad inteligente.
- Ha de contemplar la visualización detallada de un nodo específico de la topología. A demás de ver esta información detallada, también debe de poder mostrar el conjunto de servicios que el nodo seleccionado puede solicitar.
- Ha de permitir añadir nuevos servicios al catálogo de servicios, siguiendo el formato definido.
- Debe de poder solicitar la ejecución de cualquier servicio del catálogo de servicios al nodo cloud agent, para que posteriormente este gestione el servicio de la manera correspondiente.

Aplicación cliente de un agent:

- Debe de poder configurar un agent desde cero, es decir, asignar a este todos los parámetros necesarios para que pueda ser registrado correctamente en la base de datos topológica.
- Ha de poder mostrar un listado de los servicios que el agent puede solicitar.
- Ha de permitir la solicitud de cualquier servicio que se encuentre en el listado de servicios mostrado.
- Debe de poder mostrar el resultado de la ejecución de cualquier servicio que haya sido solicitado.

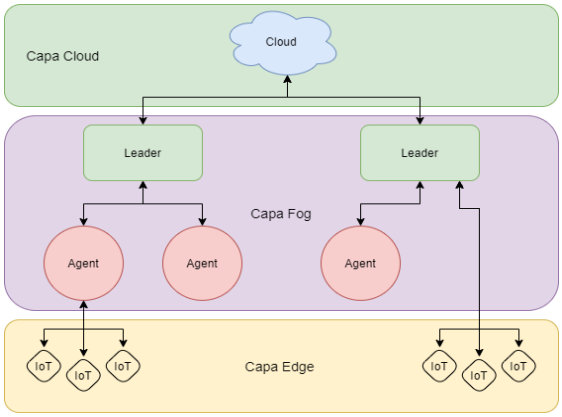


*Figura 3: Topología de la red*

## 4. Diseño

Una vez identificados los requisitos, en este apartado se detalla de forma precisa el conjunto de características que ha de tener cada uno de los elementos a implementar para que estos cumplan sus funcionalidades correctamente. A continuación, se detalla el diseño de cada uno de los elementos:

- Topología de la red: El diseño de la topología de la red está basado en la arquitectura Fog-to-Cloud. Dicha topología está compuesta por varios nodos repartidos en diferentes niveles o capas. Estas capas son las siguientes: capa Cloud, capa Fog y capa Edge.

  La capa Cloud es la capa más alta de la topología. Está formada por uno o más servidores que ofrecen diferentes servicios a los dispositivos que se encuentran en las capas situadas bajo esta. En este proyecto se ha utilizado para almacenar el Front-End, la base de datos topológica, el catálogo de servicios y los códigos de dichos servicios.

  La capa Fog es una capa en la cual podemos encontrar nodos, los cuales pueden tener como rol el de agent o leader. La funcionalidad principal de esta capa es la de poder intercambiar información entre estos nodos de forma directa y casi inmediata.

  La capa Edge es una capa en la cual se encuentran todos los dispositivos IoT que podemos encontrar en nuestra topología. Estos están conectados directamente con su respectivo agent/leader y sirven para generar o recopilar información que puede ser útil para su agent/leader.

  En la figura 3 se muestra la topología de red con los respectivos niveles nombrados anteriormente.

- Conducción autónoma del vehículo: El sistema de conducción autónoma de un vehículo está dividido en los siguientes módulos: car_movement.py, sensors.py, line_follower.py y decisión_maker.py.

  El módulo car_movement.py es el encargado de controlar la parte mecánica del vehículo, es decir, el motor y el direccionamiento de las ruedas. Dispone de un conjunto de funciones capaces de hacer girar las ruedas y de poner el motor en movimiento.

  El módulo sensors.py dispone de diferentes funciones capaces de proporcionar información sobre cada uno de los diferentes sensores que están instalados en nuestros vehículos. La información que podemos obtener de estos sensores es la siguiente:

  - Lector de línea: Este sensor capta una serie de valores los cuales indican si el vehículo se encuentra sobre una línea blanca, o no. Esta línea es utilizada para que el vehículo sea capaz de moverse dentro de los carriles de la ciudad.
  - Lector de RFID: Este sensor es utilizado para tener constancia de cual es la posición de un vehículo en la ciudad en todo momento. Esto se hace mediante la lectura de tags RFID situados bajo la ciudad, con la finalidad de que los vehículos puedan leer estos y actualizar su localización en tiempo real.
  - Sensor de ultrasonido: Este sensor está situado en la parte frontal del vehículo. Su función es la de proporcionar la distancia de cualquier objeto situado delante del propio vehículo para poder detectar obstáculos y realizar las acciones correspondientes en el caso de detectar alguno.

  El módulo line_follower.py es el encargado de recoger información del lector de línea y con la

información obtenida realizar cambios en el ángulo de giro de las ruedas del vehículo, para que de esta manera este pueda circular siempre por su carril correspondiente.

El módulo decisión_maker.py es el más importante de todos. Este recoge información de los dos sensores restantes y realiza ciertas acciones dependiendo de esta. Estas acciones pueden ser detener el vehículo por completo, detenerlo para ponerlo posteriormente en circulación, o ponerlo a circular directamente. Pero la funcionalidad más importante que tiene este módulo es la de poder comunicarse con otros agents que tienen el control de otros dispositivos necesarios para poder realizar la conducción autónoma óptima. Estos agents con los que se ha de comunicar el vehículo, generalmente son agents que controlan semáforos, farolas y el estado del tráfico de la ciudad.

En la figura 4 se muestra el diagrama de los diferentes módulos de la conducción autónoma.

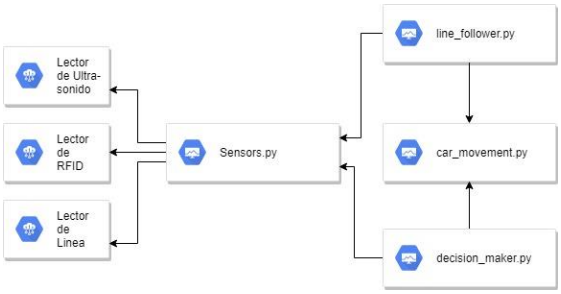


*Figura 4: Diagrama de los módulos de la conducción autónoma*

- Agent: En los requisitos funcionales de un agent se ha definido que este debe de tener sus módulos bien definidos y cada uno independiente de los demás. Los módulos que podemos encontrar en un agent son los siguientes: API, Runtime, Service Execution y Topology Resource Management.

  Módulo API: Este módulo tiene como funcionalidad principal poder recibir peticiones HTTP de otros elementos que sean visibles a nivel de red, así como también poder realizar peticiones HTTP a otros elementos de la misma red. Este ha de ser capaz de resolver un conjunto de peticiones necesarias para la correcta comunicación entre agents. Dichas peticiones son las siguientes:

  - GET: Las peticiones GET son utilizadas para recuperar recursos.
  - POST: Las peticiones POST son utilizadas para crear nuevos recursos.
  - PUT: Las peticiones PUT son utilizadas para modificar valores de recursos ya existentes.
  - DELETE: Las peticiones DELETE son utilizadas para eliminar recursos.

  Módulo Runtime: Este módulo se encarga de la ejecución de servicios. Para hacerlo debe descargar los códigos a ejecutar, analizar los parámetros de entrada para poder recogerlos desde el código, y una vez se ha hecho todo esto, ejecuta el código del servicio para finalmente devolver el resultado de este.

  Módulo Service Execution: Este módulo tiene la función de gestionar los servicios solicitados al propio agent. Es el encargado de comprobar las diferentes características de los servicios y dependiendo de estas realiza unas acciones u otras. Para los servicios complejos, este módulo se encarga de resolver dichas dependencias para que se ejecute correctamente el servicio solicitado.

  Módulo Topology Resource Management: Este módulo es el encargado de gestionar conexiones entre agents de la topología, así como de mantener actualizado el estado de esta en tiempo real.

  En la figura 5 se puede observar el diagrama de la estructura de un agent.

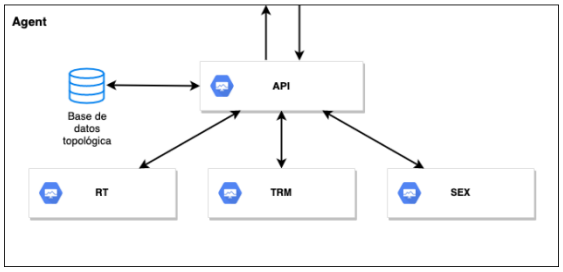


*Figura 5: Diagrama de la estructura de un agent*

- Servicio: En cuanto al diseño de los servicios, se ha definido un conjunto de parámetros para tener en cuenta. Estos parámetros tienen un papel muy importante a la hora de solicitar cualquier la ejecución de cualquier servicio. En este proyecto, el catálogo de servicios es representado como una base de datos no relacional que ha sido creada utilizando la herramienta MongoDB [4]. Entre los parámetros más importantes del diseño de un servicio podemos encontrar los siguientes:

  - Identificador: Nombre del servicio. Este campo es utilizado para identificar un servicio.
  - Descripción: Descripción del servicio utilizada para ayudar al usuario a entender el funcionamiento de este.
  - Versión de Python: Versión de Python necesaria para ejecutar el servicio. Puede ser

"python2" o "python3".

- Permanente: Campo que indica si el servicio se ejecutará de forma permanente o si por el contrario tiene fin.
- Servicios de los que depende: Listado de identificadores de servicios de los cuales depende.
- IoT: Conjunto de IoT necesarios para poder ejecutar un servicio.
- Parámetros de entrada: Conjunto de parámetros de entrada necesarios para poder ejecutar el servicio.
- Programa del servicio: Fichero que contiene el código del servicio a ejecutar.
- Programas de los que depende: Fichero o ficheros de los cuales depende el servicio para poder ser ejecutado correctamente.

- Nuevas funcionalidades del Front-End: El panel Front-End proporcionado es muy importante, pero este no dispone de todas las funcionalidades necesarias. Por eso se han diseñado las cuatro siguientes pantallas:

  Pantalla de visualización de agents: Esta pantalla contiene un listado con todos los agents que se encuentran dentro de la topología de red de la ciudad inteligente. La pantalla cuenta con un filtro y en la tabla se muestra, para cada agent los parámetros más importantes.

  Pantalla de visualización del catálogo de servicios: Pantalla que contiene un listado con el conjunto de servicios que se encuentran disponibles dentro de la ciudad inteligente. En esta pantalla hay un botón que sirve para añadir nuevos servicios al catálogo de servicios.

  Pantalla de visualización de la información de un agent: En esta pantalla se pueden ver todos los campos que tiene un agent cuando este es creado. Estos son mostrados en forma de listado y se encuentran situados en la parte izquierda de la pantalla. También se pueden ver el conjunto de servicios que el agent en cuestión puede ejecutar, repartidos en servicios de conducción autónoma y servicios de calibración.

  Pantalla para añadir servicios al catálogo de servicios: Esta última pantalla tiene como finalidad poder añadir nuevos servicios al catálogo de servicios. Para añadir nuevos servicios es necesario que el administrador del Front-End, rellene todos los campos necesarios para crear un servicio, los cuales han sido mencionados anteriormente.

- Aplicación cliente de un agent: Como se ha mencionado anteriormente, la aplicación cliente de un agent tiene como objetivo principal poder solicitar la ejecución de diferentes servicios disponibles para ese agent. Esta aplicación dispone de un primer proceso donde, si esta es ejecutada por primera vez, permite al cliente configurar el agent. En esta configuración se asigna al agent un tipo de "device" y un tipo de "role". El parámetro "device" sirve para identificar fácilmente que elemento de la ciudad inteligente será el agent. El parámetro "role" sirve para indicar si el nuevo agent hará la función de agent o leader. Una vez se ha configurado el agent, se pasa al segundo proceso, donde se muestra un listado con todos los servicios que puede solicitar el propio agent. Una vez solicitado un servicio, este se ejecuta y finalmente se muestra en la pantalla el resultado obtenido.

## 5. Implementación

Una vez se ha acabado con el diseño de cada uno de los elementos necesarios, pasamos a la implementación de estos. Esta implementación se ha realizado sobre los mismos elementos nombrados anteriormente en el diseño:

- Conducción autónoma del vehículo: Disponemos de 4 módulos los cuales cada uno cumple con una función específica del vehículo:

- Módulo car_movement.py: este módulo se encarga de gestionar los movimientos de los vehículos, es decir, del movimiento de las ruedas. Para ello tiene principalmente dos atributos, el ángulo (*set_angle(angle)*) y la velocidad (*set_speed(speed*), que al modificarlos, la velocidad y el ángulo del vehículo variaran.

- Módulo sensors.py: este módulo es el encargado de recoger la distinta información recogida por los diferentes sensores:
  - Sensor de Línea: Se utiliza la función *read_digital_line(referencia)* a la cual le pasamos una referencia obtenida en el proceso de calibración del sensor de línea. Esta función, nos devolverá una lista de 5 elementos, una por cada lector que tiene el vehículo. A partir de la referencia pasada, que es el límite miraremos por cada lector si es blanco o negro. En el caso de que el valor del lector sea mayor a la referencia, significará un color más cercano al blanco y si está por debajo, significara que el color más cercano al negro.
  - Sensor de ultrasonido: Se utiliza la función *read_distance()* con la cual obtenemos el valor numérico de la distancia expresada en cm.

- Lector de RFID: Se utiliza la función *read_RFID()* con la cual obtenemos el TAG del RFID obtenido en formato String

- Módulo line_follower: el cual tiene el método *follow_line(),* que es un bucle infinito que recoge información del sensor de línea (*read_digital_line* del módulo sensors*)*, de la dirección del vehículo (dirección de la ruta) y si el vehículo está parado(por un semáforo, obstáculo…). Con esta información, si el vehículo está parado, no hace nada, pero si no lo está, mira si hay alguna dirección en el vehículo, si la dirección es izquierda, solo seguirá los dos sensores de la izquierda, si es ir recto, solo hará caso de los 3 sensores centrales y si es derecha solo hará caso a los dos de la derecha. También hay el caso de que no tenga ninguna dirección por lo que hará caso de los 5 sensores.

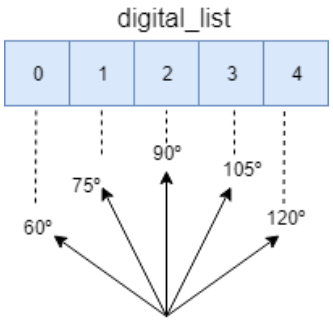


*Figura 6: Relación entre "digital_list" y los ángulos de giros correspondiente*

Como podemos observar en la figura anterior, cada elemento del lector de línea tiene asociado un ángulo de rotación de los neumáticos por lo que cuando alguno de esos nos llega a 1, significa que el vehículo debe rotar en esos grados, usando la función set_angle del car movement, aparte de actualizar el ángulo, también se actualiza la velocidad para que se adapte a la curva.

- Módulo decisión_maker: En este módulo se encuentra la inteligencia mediante la conexión a varios elementos de la ciudad (gestor del tráfico, gestor de semáforos y gestor de farolas). La ejecución comienza guardándose los datos de la ruta que le llega por parámetro y conectándose a los gestores mencionados, pidiéndole al gestor semáforos, la posición de estos y a la gestión farolas, la posición de estas también. Y después en bucle ejecuta el siguiente:

- Check_final(): Se encarga de comprobar que la ruta ha finalizado y por lo tanto terminar la ejecución del programa. Devolviendo el mensaje de que todo ha ido OK.

- Check_next_rfid(): A partir del RFID actual, manda a comprobar si el siguiente está libre y por lo tanto que puede moverse a través del gestión tráfico.

- Check_traffic_lights: En esta función, tenemos dos casuísticas: la primera es el caso de conducción normal, que el vehículo le pide a la gestión semáforos el estado de este y por lo tanto a partir del estado de este continua o se para. Después tenemos el caso de que realizas una emergencia por lo que pides al semáforo que se ponga en estado de emergencia.

- Check_distance(): Aquí leemos del sensor de ultrasonidos la distancia con el obstáculo de delante. A partir del nivel de velocidad, el vehículo se detiene cuando la distancia es menor a la que tiene en relación a la velocidad. A continuación tenemos la tabla que relacióna los niveles de velocidad con la distancia minima que debe respetar con el obstaculo.

```
STOP_DISTANCES = {
        0: 20,
        1: 10,
        2: 12,
        3: 13,
        4: 14,
        5: 15,
        6: 16,
        7: 17,
        8: 18,
        9: 19,
        10: 20
    }
```

*Figura 7: Relación entre la velocidad y la distancia*

- Check_route(): Función que comprueba si debe o no girar dependiendo del último RFID leído. Las cuales pueden ser, go_straight, turn_left, turn_right, stop y emergency_stop.
- Check_streetlight(): Función encargada de pedir que se enciendan las farolas cuando este lea el RFID relacionado con la farola.

- El Agent: Disponemos de 4 módulos los cuales cada uno cumple con una función específica del vehículo:

- El módulo API, tiene las siguientes funciones para gestionar las peticiones:
  - Start(): encargado de arrancar la API.
  - GET(): si llega una petición GET, entra aquí para ver si queremos obtener los agents, los servicios o bien saber si un agent está vivo.
  - POST(): Si llega una petición POST, entra aquí indicando si quieres registrar el agent, pedirle un servicio, pedirle que ejecute un servicio.
  - PUT(): Si llega una petición PUT, entra aquí indicando si quieres modificar un agent.
  - DELETE(): Si llega una petición DELETE, entra aquí indicando si quieres eliminar un agent.
- El módulo Runtime, se encarga de primero ir a descargar el código del servicio donde este le indica (*get_code()*), después analiza el input con el método *prepare_params()* para que los pueda leer desde dentro del programa a ejecutar, después lo ejecuta con el método *exeute_code()* y devuelve el resultado con el método *subprocess.getoutput()*.
- *El módulo ServiceExecution:* para llevar a cabo la ejecución de los servicios, se debe hacer el siguiente:
  1. El agent que recibe la solicitud tiene como rol asignado "agent":
     a. El agent añade su IP a los datos del servicio y solicita este a su leader.
  2. El agent que recibe la solicitud tiene como rol asignado "leader":
     a. Si el servicio tiene dependencias, el leader recupera de la base de datos la información de cada servicio dependiente y se lo solicita a si mismo.
     b. Comprueba si quien ha solicitado el servicio puede ejecutarlo. Si es el caso, le solicita le ejecución de este.
     c. Si no puede ejecutar el servicio quien lo ha solicitado, el leader comprueba si el mismo puede solicitar, si puede se manda un mensaje de ejecución a sí mismo.
     d. Si el propio leader tampoco puede ejecutar el servicio busca alguno de sus agents que sí que pueda. En caso de encontrar uno le solicita le ejecución del servicio a este, si por el contrario no encuentra ninguno, le solicita el servicio a su leader, es decir al cloud agent.
  3. El agent que recibe la solicitud tiene como rol asignado "cloud_leader":
     a. El cloud agent se encarga de buscar alguien de la topología que pueda ejecutar el servicio.
     b. Si encuentra a alguien:
        i. Si es agent, le solicita al leader del agent el servicio, para que este gestione la solicitud y finalmente esta llegue al agent destinado.
        ii. Si es leader, le solicita directamente la ejecución del servicio a este.
  4. Finalmente, si nadie puede ejecutar el servicio en el momento en que este es solicitado, se devuelve como resultado de la ejecución del servicio "unattended", indicando que este no ha podido ser atendido.
- El módulo Topology Resource Management, se encarga de mantener actualizada la base de datos de la topología, para ello, tenemos estas dos funciones:
  - Check_alive_agents(): Recoge los agents que descienden de este y mira si están vivos, mandándoles una petición get de check_alive, si les contesta, significa que están activos sino, modifican el estado a inactivo en la base de datos.
  - Change_agent_status(): Se encarga de cambiar el estado de un agent.

## 6. Pruebas de funcionamiento

La primera prueba realizada consiste en verificar la correcta comunicación entre agents. Los objetivos de esta son: verificar el correcto envío de información y verificar la correcta recepción de información. Estas verificaciones se han hecho mediante el registro de un agent a su leader, donde se puede observar como este envía un mensaje de registro, el leader lo recibe y finalmente el leader le devuelve el identificador dentro de la topología al agent.

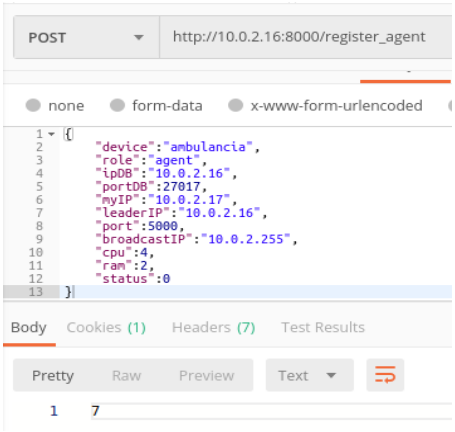


*Figura 8: Solicitud de registro y devolución del identificador*

La siguiente prueba consiste en solicitar un servicio complejo y ver cómo, teniendo activos todos los agents que han de formar parte de este, la ejecución del servicio se realiza correctamente. Para hacer la solicitud de este se puede hacer directamente desde el Front-End, o desde la aplicación cliente de un agent. Una vez solicitado, cuando finaliza el servicio se obtiene el resultado de la ejecución de este.

```
RESULTADO DEL SERVICIO GET_DUMPSTERS:

Estado: success
dumpsters: {'N6': 'DM2', 'S2': 'DM1', 'N2': 'DM3'}

Presiona ENTER para solicitar otro servicio
```

*Figura 9: Resultado de la ejecución de un servicio complejo*

De esta manera se prueba el correcto funcionamiento de la ejecución de cualquier servicio del catálogo de servicios de la ciudad inteligente, pudiendo ser estos, servicios de conducción autónoma y servicios de calibración de diferentes elementos de la ciudad inteligente.

## 7. Conclusiones

Una vez finalizado el proyecto, desde el punto de vista de los objetivos propuestos al inicio de este, podemos afirmar que todos han sido cumplidos satisfactoriamente. Se ha realizado con éxito tanto el diseño como la implementación de la conducción autónoma, dentro de un entorno de ciudad inteligente, así como la calibración de los diferentes dispositivos que podemos encontrar dentro de esta. Todo esto, teniendo en cuenta el gran papel que tiene F2C, el cual hace que la conducción autónoma desarrollada en este proyecto disponga de grandes ventajas respecto a la conducción autónoma existente hoy en día.

Para concluir, no solo se ha conseguido desarrollar un gran proyecto, cumpliendo de manera exitosa cada uno de los objetivos propuestos, sino que también se han ido mejorando y ampliando para llegar a un gran producto final.

## Agradecimientos

Agradecer a nuestro tutor Sergi Sánchez y nuestro cliente Alejandro Jurnet, por la ayuda que nos han proporcionado durante la creación de este proyecto. También agradecer a nuestras familias por su apoyo durante el desarrollo de este.

## Referencias

[1] CRAAX (Centre d'Arquitectures Avançades de Xarxes):

https://www.craax.upc.edu/index.php/about-us

[2] MF2C: https://www.mf2c-project.eu/objectives/

[3] Marlon Díaz, Roger Figueras, Marc Medrano, Carlos Padial (2018). Diseño e implementación de un Testbed para una Smart City.

[4] Adrián Roldán (2018). Panel Front-End para el control de una Smart City.

[5] MongoDB: https://www.mongodb.com

## Bibliografía

- Python. *Documentation.* Disponible en: https://www.python.org/doc/
- Fog-to-cloud Computing (F2C). Disponible en: https://ieeexplore.ieee.org/abstract/document/7528425/