Universitat de les Illes Balears

Escola Politècnica Superior

Memòria del Treball de Fi de Grau

Sistema Central De Gestión De Puntos De Carga

Javier Moyá García

Grau de Enginyeria Informàtica

Any acadèmic 2016-17

DNI de l'alumne: 43462675N

Treball tutelat per Dr. Bartomeu Jaume Serra Cifre
Departament de

S'autoritza la Universitat a incloure aquest treball en el Repositori Institucional per a la seva consulta en accés obert i difusió en línia, amb finalitats exclusivament acadèmiques i d'investigació	Autor		Tutor	
	Sí	No	Sí	No
	X		X	

Paraules clau del treball:
OCPP, estándar, carga de vehículos eléctricos, sistema central de gestión, movilidad sostenible, *Smart*UIB

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

Acrónimos y Términos

UIB Universitat de les Illes Balears

OCPP Open Charge Point Protocol

CO2 Dióxido de Carbono

TIB Transport de les Illes Balears

SOAP Simple Object Access Protocol

SQL Structured Query Language

CP Punto de Carga

IoT Internet of Things

CSS Central System Service

JSON JavaScript Object Notation

BBDD Bases de Datos

REST Representational State Transfer

3G Tercera Generación de telefonía móvil

4G Cuarta Generación de telefonía móvil

5G Quinta Generación de telefonía móvil

URI Uniform Resource Identifier

RFID Radio Frequency Identification

RESUMEN

El presente proyecto consiste en el diseño y desarrollo de un sistema de comunicación central para la red de puntos de carga de vehículos eléctricos de Mallorca, permitiendo a todos los puntos de carga formar parte de una misma red, siendo este mismo sistema capaz de ofrecer a otros sistemas externos de gestión y sensorización, la información referente a los puntos de carga que conforman la red.

El sistema desarrollado permitirá la comunicación con los puntos de carga, independientemente del fabricante de los mismos, para lo que se utilizará un protocolo de comunicaciones específico para este tipo de sistemas llamado OCPP. De este modo se elimina la necesidad de conocer los protocolos internos desarrollados por cada fabricante y se permite la inclusión de futuros fabricantes sin necesidad de modificar el sistema para garantizar su compatibilidad.

El presente proyecto forma parte del proyecto de campus sostenible ***Smart*UIB**, tratándose de un sistema para la capa de aplicación de IoT, además de ser pensado para dar soporte a un sistema de gestión de puntos de carga de ámbito autonómico desarrollado por el *Govern de les Illes Balears*, que actuaría en la capa de negocio.

CAPÍTULO 1

INTRODUCCIÓN

A lo largo de los años, desde que en 1886 Karl Benz fabricara los primeros automóviles con motor de combustión interna, la fabricación y utilización de automóviles ha aumentado de manera exponencial llegando a superar los 1.000.000 vehículos en el año 2010[1][2]. Debido a la gran cantidad de vehículos en funcionamiento cuya principal fuente de energía son los combustibles fósiles, y teniendo en cuenta que el coche con menor emisión de CO_2, fabricado el año 2007, [3] emite 104 gramos de CO_2 cada kilómetro, los automóviles se han convertido en una de las principales fuentes de contaminación mundial.

Como consecuencia, se han llevado a cabo diversos acuerdos para reducir la contaminación producida, entre las que se encuentran el Protocolo de Kioto o la Cumbre del Clima realizada en Paris el año pasado. Algunas de las propuestas pasan por reducir el nivel de emisión de los vehículos mediante diferentes legislaciones, mas una de ellas supone reducir el número de vehículos emisores, sustituyéndolos por vehículos eléctricos.

Los vehículos eléctricos suponen una de las principales alternativas a los vehículos convencionales para reducir la emisión de gases contaminantes, llegando a cero siempre que la energía eléctrica necesaria para su funcionamiento haya sido obtenida por medio de energías renovables. Por ello, no es suficiente con reemplazar los vehículos convencionales, sino que se debe realizar el cambio dentro de un entorno destinado a la sostenibilidad. En este punto surge la necesidad de conocer y gestionar el consumo producido por lo vehículos eléctricos. Esta gestión, junto con la de infinidad de dispositivos, se realiza en las *Smart Cities*, entre cuyos principales objetivos se encuentra la eficiencia, gracias a disponer de información proporcionada por sensores repartidos a lo largo de toda la ciudad que comunican constantemente el estado de los dispositivos, para satisfacer las demandas globales en términos de reducción de emisiones de carbono y consumo de energía para un futuro más sólido y sostenible [4].

En el ámbito universitario, la UIB lleva a cabo un proyecto llamado ***Smart*UIB**, que tiene como objetivo transformar el campus universitario en, tal como se indica en su definición [5], un agente activo que mejore y potencie aspectos fundamentales que faciliten un metabolismo de la universidad tales la sostenibilidad, la transversalidad, la innovación y la transferencia. El presente proyecto se enmarca en el ámbito de ***Smart*UIB** con el objetivo de permitir la gestión de las cargas realizadas en el campus de la UIB.

1.1. Objetivos del proyecto

Este proyecto consiste en el desarrollo de un sistema que permita la comunicación en una red de puntos de carga de vehículos eléctricos por medio del protocolo abierto OCPP. Además, será necesario implementar una lógica de gestión para los puntos de carga, incluyendo las bases de datos con la información de los puntos de carga y los sistemas de autorización. El sistema desarrollado deberá ser capaz de gestionar los puntos de carga, las acciones que estos realizan y conocer el estado de cada uno de ellos, incluyendo un registro de todos los mensajes intercambiados entre el CSS y los diferentes puntos de carga. Con este sistema se espera poder proveer a sistemas de gestión de mayor nivel, entre los que se incluyen la plataforma Sentilo en la que la UIB gestiona los datos de todos los sensores instalados en el Campus para ***Smart*UIB**, y una plataforma central desarrollada por el Govern De le Illes Balears para gestionar todos los puntos de carga de la comunidad autónoma como si estos fueran de un propietario único, de igual manera que ya sucede con los autocares en el *Consorci de Transports de Mallorca* (TIB).

1.2. Estructura del documento

El presente documento se encuentra estructurado de la siguiente manera:

- En el capítulo 2 se analizará el estado del arte referente a las comunicaciones y gestión de puntos de carga de vehículos eléctricos y cuáles son los retos que esto plantea.

- En el capítulo 3 se expondrá en contexto tecnológico en el que se enmarca el presente proyecto, explicando las tecnologías y herramientas utilizadas para su desarrollo. Principalmente se ha hecho uso de Java como lenguaje de programación, utilizando tecnología de Web Services con el protocolo OCPP y bases de datos MySQL.

- En el capítulo 4 se definirá el proyecto **Sistema central de gestión de puntos de carga**, los requisitos, tanto funcionales como no funcionales, del proyecto, la arquitectura del sistema desarrollado y cómo este ofrecerá solución a los retos mencionados en el capítulo 2.

- En el capítulo 5 se presentará la metodología de trabajo seguida para el desarrollo del proyecto y la planificación seguida durante el mismo.

- En el capítulo 6 se mostrará el desarrollo del sistema, incluyendo los principales problemas surgidos y las soluciones que se han seguido para subsanarlos.

- En el capítulo 7 se ofrecerá una visión de las pruebas realizadas durante el desarrollo del proyecto y cómo han influido en el mismo.

- En el capítulo 8 se recogerán las conclusiones del proyecto, incluida la resolución de objetivos, mostrando además las líneas futuras que surgen del desarrollo del proyecto **Sistema central de gestión de puntos de carga**.

CAPÍTULO 2

ESTADO DEL ARTE

Este capítulo del documento tiene como objetivo poner en conocimiento del lector el contexto en el que se encuentra el proyecto que describe, ofreciendo una visión general del tipo de sistema desarrollado y las soluciones que hasta ahora se han llevado a cabo.

Para ello, se analizará en primer lugar la estructura de IoT y el estado actual en el que se encuentran los sistemas de carga de vehículos eléctricos. Posteriormente se analizará la plataforma Sentilo, ya que se trata de un sistema frecuentemente utilizado en el ámbito del IoT; y se estudiará el protocolo OCPP para comunicación de puntos de carga de vehículos eléctricos. Finalmente, se mostrarán diferentes soluciones desarrolladas en el mismo ámbito.

2.1. IoT

Para poder gestionar los puntos de carga tal y como plantea este proyecto, primero es necesario disponer de una red en la que se encuentren conectados dichos puntos y mediante la cual sea posible obtener información referente a ellos. Este sistema se correspondería con un sistema de Internet of Things.

Internet de las cosas (IoT por su abreviatura en inglés de Internet of Things) es un concepto que se refiere a la interconexión digital de objetos cotidianos a internet. Consiste en un conjunto de objetos inteligentes con capacidad para obtener, procesar y enviar información sobre su entorno, ya sea un sensor que detecta el ritmo cardíaco de una persona, un sensor capaz de reconocer señales de tráfico o un sensor para detectar la temperatura ambiental de un domicilio.

A partir de este concepto, se espera que en un futuro el Internet de las cosas evolucione a ser un sistema no determinista [6] (hablando en términos matemáticos) de red abierta, donde entidades inteligentes auto-organizadas serán interoperables y capaces de actuar de forma independiente, en función del contexto, las circunstancias o el ambiente.

Este futuro IoT seguiría una arquitectura orientada a eventos, construida de abajo hacia arriba (basada en el contexto de procesos y operaciones en tiempo real) teniendo en consideración cualquier nivel adicional. Como se puede ver en la Ilustración 1 [7], la base del IoT serán los dispositivos inteligentes (o nodos) conectados a internet y accesibles para los usuarios a través de dicha conexión. Por lo tanto, el modelo orientado a eventos y el enfoque funcional coexistirían con nuevos modelos capaces de tratar excepciones.

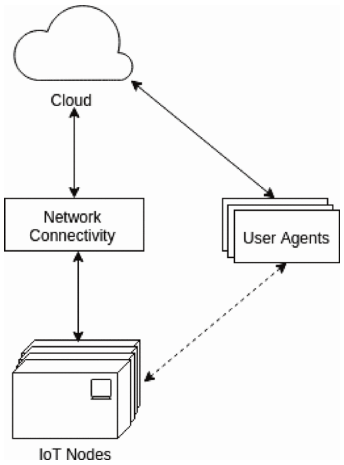


Ilustración 1: Estructura IoT

Tal y como se puede observar en la Ilustración 2 [8], el IoT puede ser representado mediante un modelo de cinco capas. En el caso concreto de la gestión de los puntos de carga, los nodos serían dichos puntos, y formarían parte de la capa de percepción.

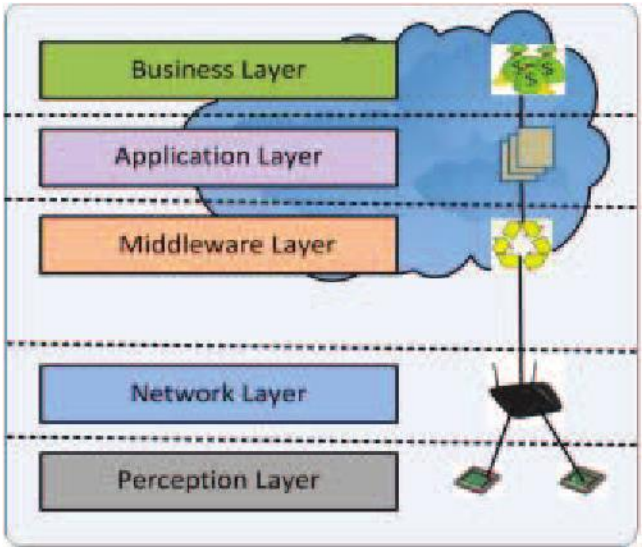


Ilustración 2: Arquitectura de capas IoT

Por el momento éste IoT no es más que una promesa de futuro, más conocerlo permite contextualizar el desarrollo del presente proyecto y las posibilidades que el mismo plantea.

2.2. Solución actual

Actualmente, los diferentes puntos de carga repartidos por Mallorca son gestionados por un sistema del proveedor eléctrico, del fabricante, o en algunos casos, de la empresa propietaria del punto de carga si ésta ha desarrollado un sistema de gestión propio. Por lo tanto, los puntos de carga quedan divididos en diversos sistemas independientes e incomunicados, impidiendo además a los propietarios de esos puntos a gestionar sus propios puntos de carga y requiriendo

acceder al sistema del proveedor para obtener información de los mismos. Éste es el caso de la UIB, que para obtener la información de sus puntos de carga debe ejecutar un script que periódicamente acceda al sistema de la empresa proveedora, descargue una hoja de cálculo con los datos de consumo de los puntos de carga y posteriormente envíe esos mismos datos a su sistema Sentilo de sensorización.

En este punto surge la necesidad de tener un sistema de fácil acceso que permita la comunicación entre la plataforma del propietario y los puntos de carga. Además, por parte del Govern de les Illes Balears, se propone integrar todos los puntos de carga en un sistema único, gestionado por el Transport de les Illes Balears, de modo que de cara al usuario los puntos de carga aparenten ser de un mismo gestor.

2.3. Sentilo

El proyecto ***Smart*UIB** es un proyecto de campus sostenible que afecta a varios ámbitos de la comunidad universitaria, incluyendo un sistema de IoT que conecta los sensores tanto de temperatura, uso energético de los edificios, como los puntos de carga de vehículos eléctricos. Toda la información recogida por estos sensores es almacenada en la plataforma de código abierto Sentilo [9], que tal como muestra la Ilustración 3, recibiría la infoemación de las redes en las que se encuntran los sensores, siendo el sistema diseñado por el presente proyecto en el caso de los puntos de carga de vehículos eléctricos.

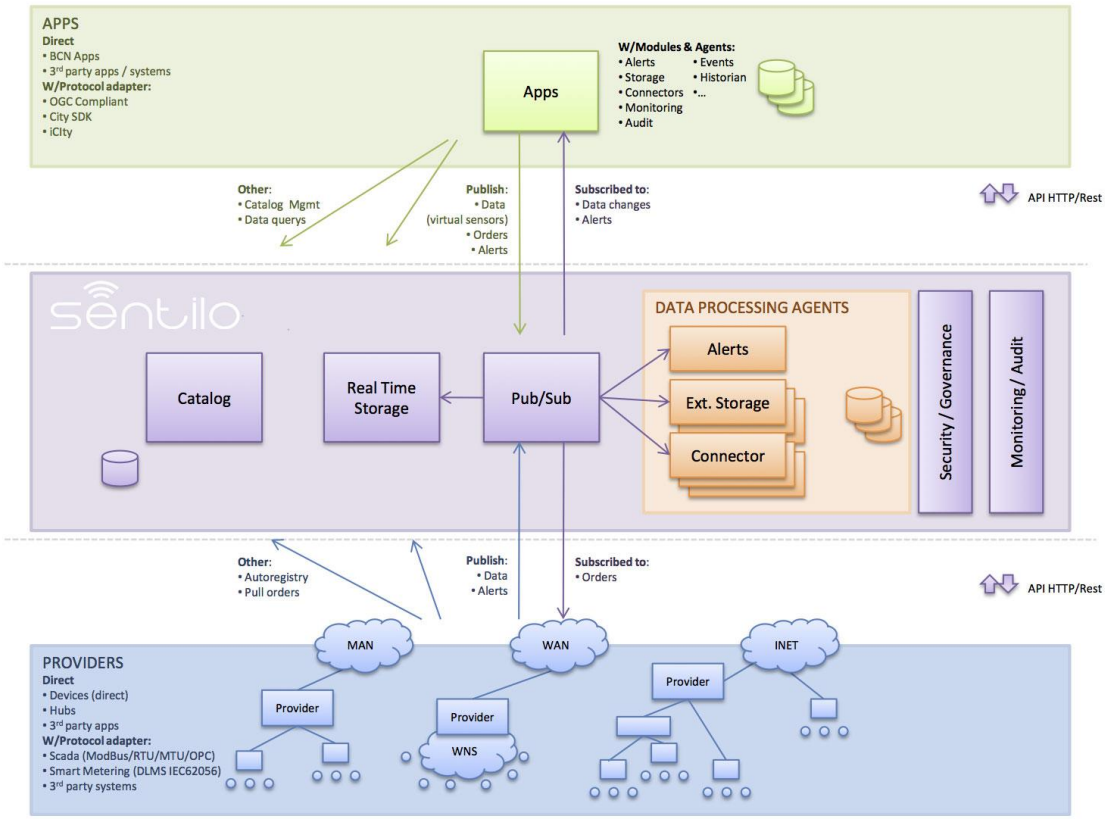


Ilustración 3: Estructura IoT de la plataforma Sentilo

Para la obtención de esos datos, la plataforma Sentilo requiere que le sean enviados en el formato JSON, permitiendo así un gran flujo de datos, necesario al soportar una gran cantidad de sensores conectados a la plataforma. Para enviar estos mensajes JSON se hace uso de la arquitectura REST.

Para hacer posibles las propuestas del *Govern de les Illes Balears* y ***Smart*UIB**, y para garantizar la posibilidad de integración de cualquier fabricante de puntos de carga es necesario establecer un protocolo de comunicaciones común para los diferentes puntos de carga del sistema. A continuación, se analizará el protocolo abierto OCPP.

2.4. OCPP

OCPP (Open Charge Point Protocol) es, como su propio nombre indica, un protocolo abierto de comunicación para los puntos de carga de vehículos eléctricos, desarrollado por la OCA (Open Charge Alliance) e iniciado por la fundación neerlandesa E-Laad (ahora llamada ElaadNL) para, tal y como explican en su página web [10], ofrecer una solución uniforme para el método de comunicación entre los puntos de carga y la estación central, permitiendo la conexión entre cualquier punto de carga y cualquier estación central.

Gracias a implementar OCPP, los puntos de carga adquieren un nivel de flexibilidad que les permite integrarse en cualquier red de carga, simplemente modificando el destinatario de los mensajes que producen. Gracias a esto, todos los puntos de carga pueden formar parte de una única red, compartiendo el mismo sistema central, independientemente tanto del fabricante y el propietario del punto de carga, como de su suministrador eléctrico.

También resulta interesante para, en el caso de existir más de una red, poder migrar los puntos de carga en caso de cambiar de propietario, o en el caso de que el proveedor eléctrico ofrezca precios muy elevados, favoreciendo la competitividad entre suministradores sobre precio y servicios.

OCPP está basado en un sistema de WebService cuyas llamadas siguen el protocolo SOAP. Tanto los diferentes puntos de carga como la estación central, a la que de aquí en adelante llamaremos CSS, funcionan como servidor y como cliente, pudiendo ambos iniciar peticiones para realizar una acción, como recibir una solicitud desde el otro lado del sistema.

Recientemente la OCA ha finalizado el desarrollo de la versión 1.6 de OCPP, que incluye compatibilidad con el formato JSON, disminuyendo así la sobrecarga que introduce al sistema la utilización del formato SOAP, ya que reduce el tamaño del flujo de datos de forma considerable; además de la revisión de algunas de sus funcionalidades y se prevé un primer lanzamiento de la versión 2.0 para finales del año 2017, que añadiría una gran variedad de funcionalidades, siendo la más destacable la inclusión de *Smart Charging* [11].

2.5. Otras soluciones desarrolladas

Finalmente, antes de analizar qué opciones se han escogido para el desarrollo del presente proyecto, se ofrecerá una visión de otras soluciones llevadas a cabo en el ámbito de la gestión de puntos de carga.

2.5.1. iGSEGeS

La empresa Innova Estudi Soft ha desarrollado un sistema de gestión de redes de carga de vehículos eléctricos llamado iGSEGeS [12]. El sistema iGSEGeS es una aplicación que abarca todas las capas del modelo IoT, incluyendo la gestión técnica, la gestión de clientes, la gestión administrativa, y la gestión con los otros agentes del mercado eléctrico, dando lugar a un sistema llamado *Gestor de Carga*. Para establecer comunicación con los puntos de carga iGSEGeS utiliza el protocolo OCPP. Al diseñar una aplicación única para todas las capas de IoT, no permite la integración de la red de puntos de carga con otros sistemas ni permite la gestión de la red en diferentes niveles, como sería una gestión por parte de la UIB y una gestión a mayor nivel por parte del *Govern de les Illes Balears*.

2.5.2. NOC

Otra solución llevada a cabo es el proyecto NOC desarrollada por la empresa Etra. Tal como indican en su página web [13] El proyecto NOC (Network Operation Center) provee la tecnología necesaria para la gestión de los postes de recarga de la ciudad de Barcelona.

Al igual que iGSEGeS, hace uso del protocolo OCPP para la comunicación entre el sistema y los puntos de carga, pero en este caso, el sistema diseñado correspondería a las capas de aplicación y negocio del modelo IoT, permitiendo la inclusión de puntos de carga de otros sistemas. El sistema NOC se encuentra actualmente instalado en el Ayuntamiento de Barcelona, en concreto en el Centro de Gestión de Tráfico Urbano, y es el que utiliza la Oficina LIVE (Logística para la Implementación del Vehículo Eléctrico) para la gestión de los puntos de recarga de la ciudad. Actualmente esta red cuenta con 12 puntos de recarga equipadas con dos tomas de corriente cada una e interconectadas a través de la red de comunicaciones municipal con el Centro de Control de tráfico [13].

Una vez analizados los sistemas y soluciones presentes en el ámbito de la comunicación y gestión de puntos de carga de vehículos eléctricos, en el capítulo 3 se ofrecerá una explicación sobre los sistemas y tecnologías seleccionados para el desarrollo del proyecto **Sistema central de gestión de puntos de carga.**

CAPÍTULO 3

MARCO TECNOLÓGICO

Ahora que se ha estudiado el estado actual de los sistemas de gestión de puntos de carga de vehículos eléctricos y las posibilidades que en él residen, se procederá a analizar la tecnología seleccionada para el desarrollo del proyecto. Cabe destacar que queda fuera del alcance de este proyecto la integración con otros sistemas de gestión o sensorización externos, como podría ser la plataforma Sentilo o el sistema de gestión propuesto por el Govern de le Illes Balears. Otro aspecto importante a destacar es que el sistema de almacenamiento de datos utilizado en el proyecto **Sistema central de gestión de puntos de carga** es una solución temporal, ya que, al realizar la integración con otros sistemas de gestión, el almacenamiento de la información dejaría de ser competencia del proyecto en cuestión y sería necesario desarrollar una plataforma que permita realizar el intercambio de los datos almacenados en el sistema externo.

El presente proyecto consiste en el desarrollo de un sistema que pueda dar soporte a sistemas de capa de negocio, entre los que se encontrarían ***Smart*UIB** y el sistema para el TIB, capa en la que se analiza la información proporcionada y se llevan a cabo acciones de gestión en consecuencia. Por lo que a diferencia de las soluciones analizadas en el capítulo 2, el sistema diseñado sería de capa de aplicación, permitiendo la integración con sistemas de las diferentes capas de IoT.

El núcleo del proyecto **Sistema central de gestión de puntos de carga** radica en el desarrollo de un CSS (Central System Service) que permita la comunicación con los puntos de carga de vehículos eléctricos. Para ello es necesario tanto poder recibir mensajes de los puntos de carga con peticiones para iniciar una acción, como enviar dichas peticiones en sentido contrario.

El protocolo utilizado para establecer estas comunicaciones es el previamente explicado OCPP. No obstante, debido a la reciente estabilidad de la versión 1.6, la mayoría de fabricantes optan por utilizar la versión 1.5 en sus puntos de carga, por lo que, a pesar de requerir mayor utilización del ancho de banda, para desarrollar este proyecto y garantizar la compatibilidad con los diferentes fabricantes ha sido necesario utilizar la versión 1.5 del protocolo.

Una vez aclarados estos aspectos, esta sección explicará las tecnologías seleccionadas para este proyecto.

3.1. Tecnologías de CSS

En este punto es necesario aclarar que, según el formato de comunicaciones definido para el protocolo OCPP, es necesario que tanto el CSS como los CP tengan un comportamiento dual, pudiendo funcionar ambos como servidor y como cliente. Para ello se hará uso de la tecnología de web services, quedando definidos un web service para el CSS y uno para cada

uno de los puntos de carga, además de un cliente para el CSS en cada punto de carga y un cliente en el CSS que permita la comunicación con los web services de los CP. La creación de este cliente dinámico será explicada en el apartado 6.

Para el desarrollo del CSS, tanto del servidor, el cliente y la lógica interna se ha optado por el lenguaje de programación Java, ya que además de incluir una potente librería de web services, el sistema de gestión del TIB con el que se aspira a integrar el proyecto **Sistema central de gestión de puntos de carga** sería sido desarrollado en Java.

3.2. Tecnologías de almacenamiento de la información

Como ya se ha explicado al inicio de este capítulo, se espera que en próximos proyectos de integración la información pase a ser almacenada por un sistema externo de gestión, más por el momento se contará con dos bases de datos internas, una para los datos de autorización y otro para la información de los puntos de carga conectados al sistema. Dichos datos deberán ser accesibles de manera estructurada, por lo que la opción más adecuada en una base de datos tipo SQL.

Estas bases de datos no serán sometidas a una gran carga, y la mayoría de los accesos serán de lectura. Para su implementación se ha optado por una solución MySQL, al ser esta la opción más utilizada en el entorno de aplicaciones web. Además, el lenguaje Java tiene una muy buena compatibilidad con las llamadas a bases de datos con peticiones tipo SQL.

3.3. Otras posibles tecnologías

Tal y como se ha expuesto anteriormente, este proyecto pretende dar soporte a la capa de negocio de un sistema de IoT, por lo que entran en juego las tecnologías utilizadas en esta capa a la hora de valorar este proyecto. Originalmente en esta capa quedan contempladas la gestión realizada por la UIB y la realizada por el *Govern de les Illes Balears*, por lo que únicamente se procederá al análisis de las mismas.

En primer lugar, al tratarse de un proyecto universitario, la prioridad es servir al proyecto ***Smart*UIB**, de modo que a continuación se analizará la tecnología utilizada para la gestión de sus sensores a lo largo del campus. A pesar de quedar fuera del alcance de este proyecto, al haber seleccionado Java como lenguaje de programación para desarrollar el CSS no habrá problema en integrar el envío de mensajes a la plataforma mediante llamadas al web service RESTful de Sentilo.

En lo referente al sistema desarrollado por el TIB, por el momento solo se conoce que será implementado con el lenguaje de programación Java, y desarrollar de igual manera el CSS facilitará tanto su integración como su mantenimiento.

CAPÍTULO 4

ANÁLISIS DEL SISTEMA

A lo largo de la siguiente sección, una vez ha sido explicada la tecnología seleccionada para la implementación del proyecto, se expondrán de igual manera la ingeniería de requisitos del proyecto y la arquitectura del sistema, comenzando por ofrecer una definición más específica del sistema.

4.1. Definición del sistema

El proyecto **Sistema central de gestión de puntos de carga** tiene como finalidad facilitar la gestión de la carga de vehículos eléctricos tanto en el ámbito del campus universitario como en toda la isla de Mallorca, promoviendo así la movilidad sostenible en las Islas Baleares. Para ello, se desarrollará un sistema central con el que los puntos de carga puedan establecer conexión, y que sea capaz de gestionar los mismo mediante el protocolo abierto OCPP. El CSS debe permitir su fácil integración con sistemas de gestión externos, como la plataforma Sentilo o la aplicación de la capa de negocio del Govern de les Illes Balears.

El sistema diseñado constará principalmente de tres partes, siendo dos de ellas para el establecimiento de las comunicaciones OCPP y la otra para la gestión de las bases de datos. Las partes propias de OCPP del CSS son una parte de servidor, en la que los CP pueden acceder al web service del CSS para iniciar una conexión; y la parte de cliente, en la que el CSS actúa como cliente del web service de los diferentes CP de la red. De aquí en adelante, el documento se referirá a las partes del CSS utilizando los nombres *"servidor"*, *"cliente"* y *"gestión de las* BBDD*"*.

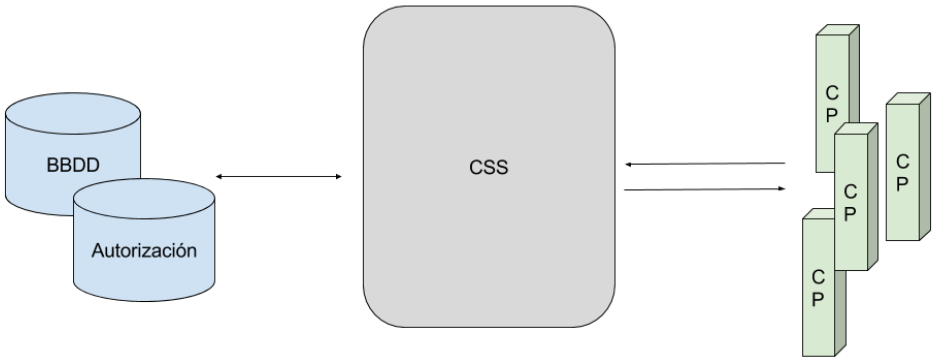


Ilustración 4 Esquema del sistema CSS

- La parte *"servidor"* implementa todas las llamadas iniciadas por los puntos de carga con destino al CSS, y gracias a la *"gestión de las* BBDD*"* permitirá realizar un

registro de los mensajes intercambiados y mantener información relevante sobre los puntos de carga.

- La parte *"cliente"* implementa todas las llamadas que el CSS inicia con destino a alguno de los puntos de carga, incorporando de igual manera la *"gestión de las* BBDD*"* para mantener actualizada la información de los CP.

- La *"gestión de las* BBDD*"*, como ya se ha expuesto anteriormente, permite mantener un registro de los mensajes intercambiados entre CSS y los CP, conocer el estado, tanto de los CP como de las transacciones que ellos realizan¸ y obtener las credenciales de autorización mediante el acceso a una base de datos con las tarjetas de identificación.

La arquitectura establecida para desarrollar el sistema CSS con las partes mencionadas será explicado próximamente, una vez que hayan sido establecidos formalmente los requisitos del sistema desarrollado en el presente proyecto, tanto funcionales como no funcionales.

4.2. Ingeniería de requisitos

4.2.1. Requisitos funcionales

RF01: El sistema debe permitir enviar mensajes a cualquier CP de la red

RF02: Cualquier CP de la red debe poder enviar mensajes al sistema.

RF03: El sistema debe mantener un registro de todos los mensajes intercambiados entre el CSS y los CP.

RF04: El sistema debe permitir conocer el estado en el que se encuentra un CP determinado.

RF05: El sistema debe disponer de un sistema para autorizar a los usuarios de los CP.

RF06: El sistema debe permitir a los CP iniciar una transacción.

RF07: El sistema debe permitir a los CP finalizar una transacción.

RF08: El sistema debe permitir conocer el consumo energético asociado a una determinada transacción.

RF09: El sistema debe permitir la incorporación de un nuevo CP a la red.

RF10: El sistema no debe permitir dos transacciones simultáneas en el mismo conector de un CP.

RF11: El sistema podrá modificar la disponibilidad de un CP para impedir su utilización.

RF12: El sistema podrá modificar la disponibilidad de un CP para permitir su utilización.

RF13: El sistema debe permitir iniciar una transacción de forma remota por petición del sistema de gestión.

RF14: El sistema debe permitir finalizar una transacción de forma remota por petición del sistema de gestión.

RF15: El sistema debe permitir realizar una reserva sobre un CP durante 30 minutos.

RF16: El sistema debe permitir cancelar una reserva realizada.

4.2.2. Requisitos no funcionales

RNF01: El sistema debe ser fácilmente integrable en otros sistemas de gestión

4.3. Arquitectura del sistema

La arquitectura del sistema CSS, teniendo en cuenta las partes establecidas en el apartado 4.1 de esta sección, correspondería a la que muestra la Ilustración 5 a continuación. La arquitectura de clases necesaria para desarrollar el sistema será mostrada en el capítulo 6 del presente documento.

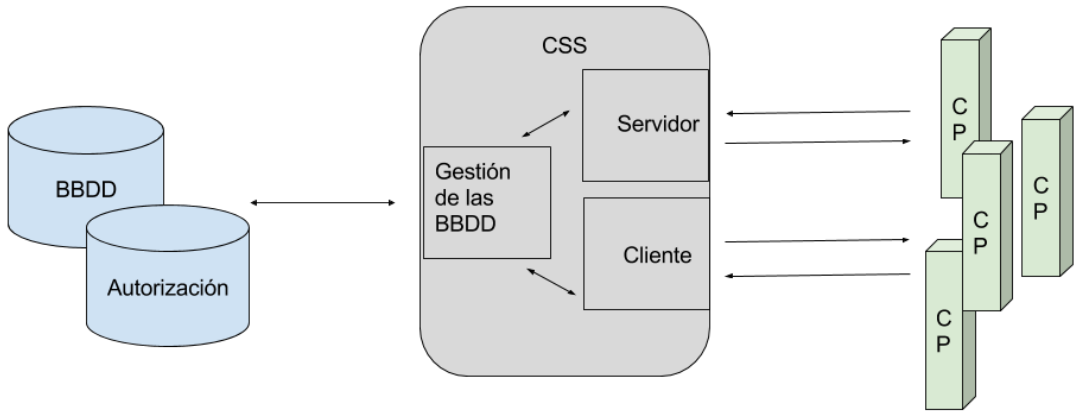


Ilustración 5 Arquitectura del sistema

A continuación, en el capítulo 5 se mostrará cómo se ha llevado a cabo este proyecto, indicando tanto la metodología seguida como la palfinifación del mismo.

CAPÍTULO 5

DEFINICIÓN DEL PROYECTO

Una vez conocidos los requisitos del proyecto, este capítulo mostrará la metodología de trabajo y la planificación que se ha utilizado durante el desarrollo de este proyecto.

5.1. Metodología de trabajo

Debido a las características del sistema, en un primer momento se optó por seguir un modelo de desarrollo incremental [14], en el que a cada incremento se analizaría, diseñaría, implementaría y probaría una nueva funcionalidad del protocolo OCPP, estableciendo así en las primeras fases la base a seguir para el desarrollo de cada una de dichas funcionalidades, y permitiendo conocer las casuísticas que estas generan antes de comenzar con la siguiente.

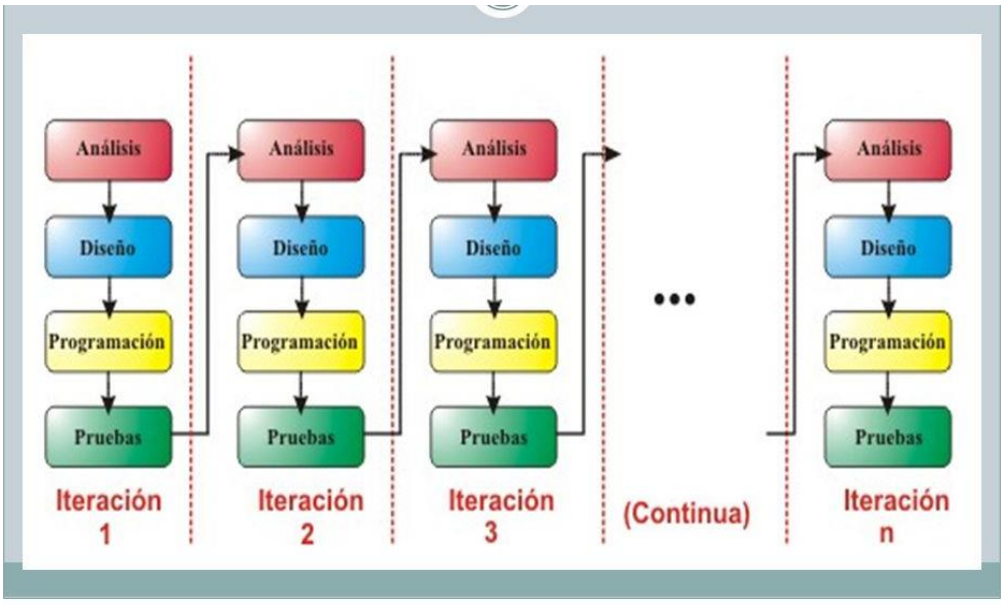


Ilustración 6 Modelo incremental

Desgraciadamente, debido a la dificultad que suponía disponer de un entorno de laboratorio con un punto de carga, la fase de pruebas de cada incremento no podía realizarse al final del mismo, por lo que se decidió continuar con una metodología similar, agrupando las fases de pruebas de 4 iteraciones para realizarlas de manera conjunta, por lo que el modelo seguido sería similar al mostrado en la ilustración 7.

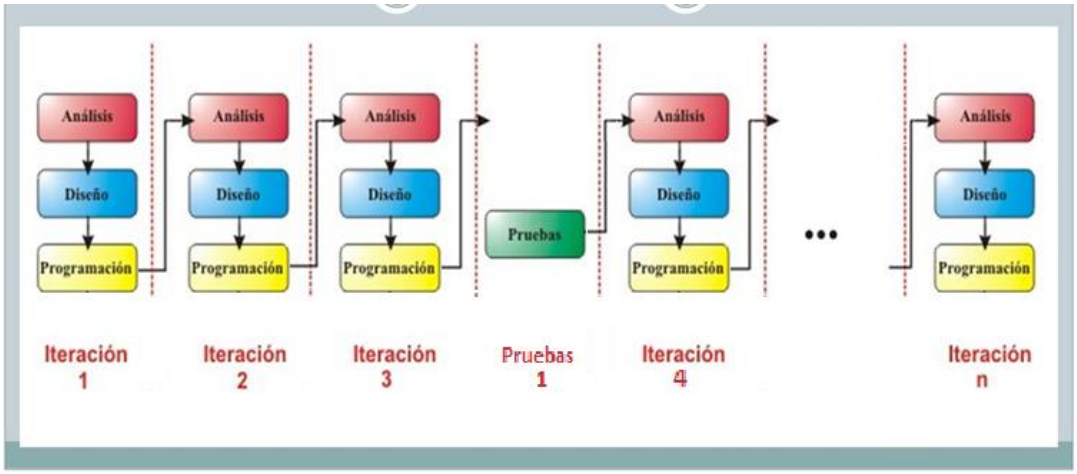


Ilustración 7 Modelo de trabajo seguido

Siguiendo un modelo así, es posible incluso desarrollar dos incrementos de manera simultánea, evitando arrastrar los errores encontrados en las fases de pruebas previas y desarrollando todas las funcionalidades siguiendo un mismo formato.

5.2. Planificación

Siguiendo la metodología establecida, el desarrollo del proyecto se divide en 21 incrementos, de los cuales 5 son fases de pruebas. Los incrementos más relevantes son el primero, cuya duración es de dos semanas, en el que se desarrollaría la funcionalidad BootNotification, método en el cual el CSS actúa como servidor, y se establecen las bases de la arquitectura de clases, la gestión de las bases de datos y los logs; junto con el incremento cinco, en el que se desarrolla el método Reset, ya que es el primer incremento en el que se desarrolla un método en el que el CSS actúa como cliente, definiendo con él como serán las conexiones del CSS hacia los puntos de carga y como se creará el cliente de forma dinámica para establecer comunicación con los web services de los CP.

A la hora de decidir el orden de desarrollo de las funcionalidades OCPP, se ha establecido un orden en base al orden lógico de actuación de las mismas, siendo la primera BootNotification, utilizada al poner en funcionamiento un punto de carga; y llegando hasta RemoteStopTransaction, utilizada, como su propio nombre indica, cuando se solicita a un CP poner fin a una transacción. Para algunas de las funcionalidades, como podrían ser Reset o ClearCache, se ha asignado un orden decidido de manera arbitraria, ya que no forman parte del flujo de comunicaciones habitual de un punto de carga, y se ha decidido desarrollarlos antes que las llamadas correspondientes a las transacciones por su menor complejidad y para definir la parte *"cliente"* del CSS en las primeras fases del proyecto.

Cada incremento tendría una duración de una semana, realizando dos incrementos de manera simultánea, a excepción de los dos incrementos mencionados al inicio de esta sección, por ser más complejos al definir cómo se actuará de cara a los próximos, y de los incrementos de pruebas, ya que en ellos se agruparán las pruebas y correcciones de 4 incrementos.

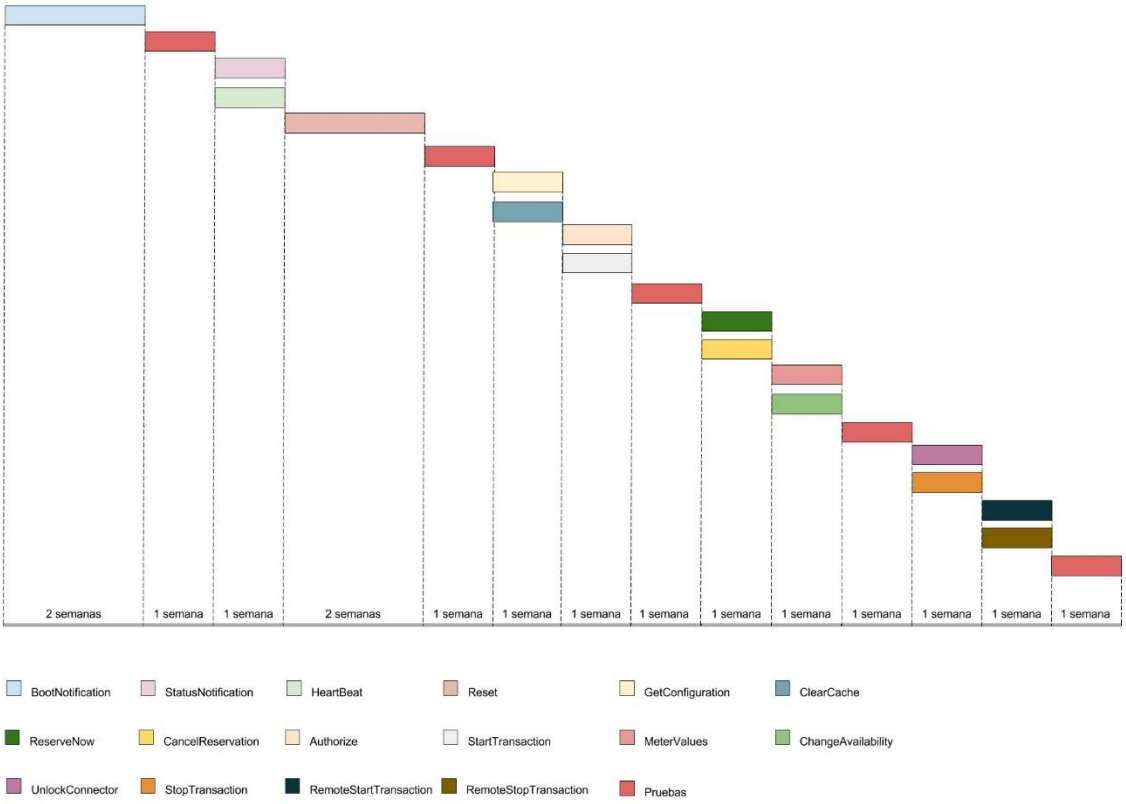


Ilustración 8 Planificación

Con esta planificación, el desarrollo del proyecto queda en una duración de 16 semanas, a las que se han añadido otras 4 para disponer de margen de contingencia.

Una vez conocidas las características tanto del sistema como del proyecto, en el próximo capítulo se analizará detalladamente todo el desarrollo llevado a cabo.

CAPÍTULO 6

DESARROLLO DEL PROYECTO

A lo largo del presente capítulo se analizará el desarrollo de las funcionalidades mencionadas en los incrementos del pasado capítulo, incluyendo los problemas correspondientes a cada uno de ellos y las soluciones llevadas a cabo para enmendarlos.

No obstante, antes de ello, se mostrará, en primer lugar, en la Ilustración 9 la arquitectura de clases establecida para el sistema diseñado, y a continuación, se analizarán las bases de datos utilizadas en el proyecto. Para ilustrar esta arquitectura se han obviado algunas clases, ya que su función es puramente proporcionar utilidades para la programación y no añaden valor para el diseño.

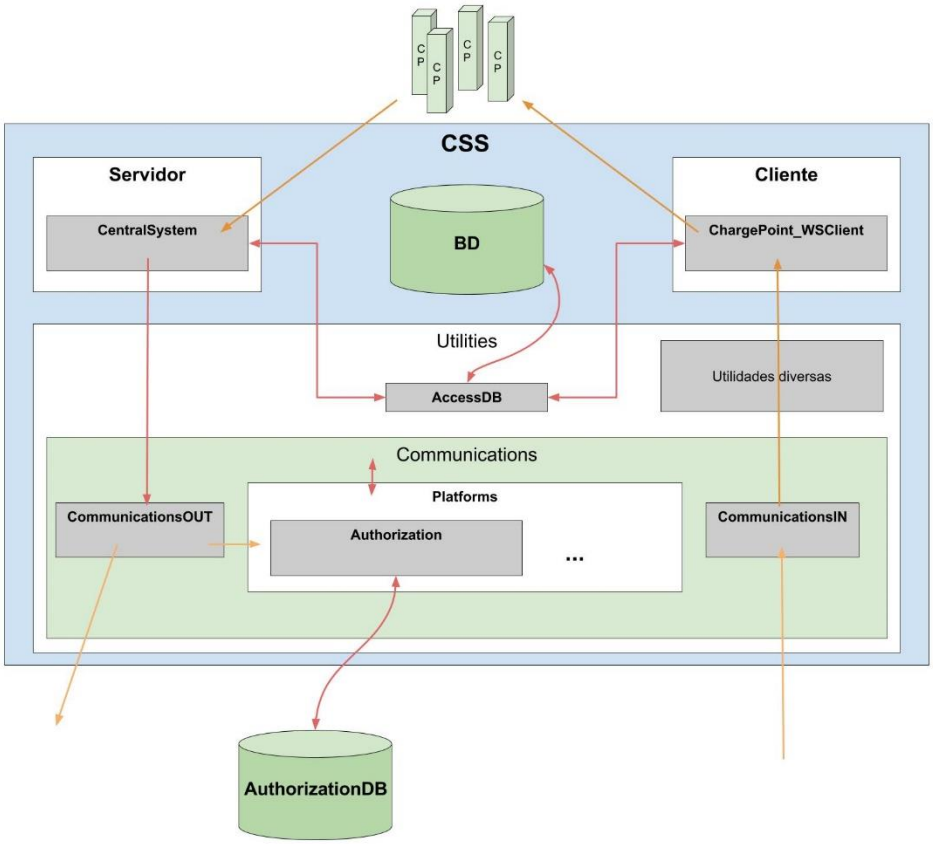


Ilustración 9 Arquitectura de clases

Para una mayor comprensión de la arquitectura de clases y de cómo esta ofrece una solución a la arquitectura propuesta en el capítulo 4, a continuación, se ofrecerá una explicación de las clases y paquetes más importantes.

- *CentralSystem*, del paquete *WS_Central*, es el web service que representa la parte *"servidor"* del CSS, y a ella acceden los diferentes puntos de carga para iniciar sus peticiones. El construcctor recibe un URI que es utilizado para la creación de un cliente de web service de manera dinámica, permitiendo así tener un cliente para cada CP en el momento que es necesitado.

- *ChargePoint_WSClient* es la clase encargada de acceder al web service de los puntos de carga para realizar las peticiones iniciadas por el CSS, por lo que representa la parte *"cliente"* del mismo.

- *AccessDB* es la clase encargada de realizar la *"gestión de las* BBDD*"*. A ella acceden las dos clases anteriores para registrar en la base de datos interna el estado de los puntos de carga y sus acciones. No obstante, como ya ha sido expuesto anteriormente en el presente documento, es posible que, en un futuro, con la integración del sistema **Sistema central de gestión de puntos de carga** en otros sistemas de gestión, tanto la base de datos interna como esta misma clase desaparezcan o sufran modificaciones.

- *Communications* es el paquete encargado de establecer la integración del sistema **Sistema central de gestión de puntos de carga** con el resto de sistemas.

- La clase *CommunicationsOUT* servirá para comunicar las peticiones realizadas por los puntos de carga incluyendo llamadas a las diferentes plataformas del paquete *Platforms*. Esto, junto al paquete que se explica a continuación es esencial para la consecución del **RNF01**.

- *CommunicationsIN* consiste en un web service mediante el cual los sistemas de gestión pueden solicitar iniciar una petición a los puntos de carga, indicando el CP sobre el que se desea ejecutar la acción. Con el identificador de CP indicado, se obtiene el URI correspondiente y se crea un objeto de la clase *ChargePoint_WSClient*.

- *Platforms* es el paquete que contiene las clases relativas a las diferentes plataformas con las que se integra el sistema **Sistema central de gestión de puntos de carga**, disponiendo inicialmente de *Authorization*, para obtener la información relativa a las tarjetas de identificación. En un futuro podrían incluirse más clases, ya sean una correspondiente a cada plataforma con las que se realiza la integración del sistema **Sistema central de gestión de puntos de carga**, por ejemplo, para la plataforma Sentilo, una clase que realice llamadas a su web service REST; o una clase única a la que pueda acceder cualquier sistema para obtener la información.

6.1. Bases de Datos

Principalmente, este proyecto requiere de una base de datos en la que almacenar la información referente a los puntos de carga y sus estados, más para su desarrollo ha sido necesario incluir una pequeña base de datos que cumpla el papel de sistema externo para la gestión de la

autorización de las tarjetas de identificación. Esta BBDD, como ya ha sido explicado anteriormente, será sustituida en un futuro por la integración con otro sistema de la capa de negocio de IoT.

En este apartado se analizarán ambas bases de datos, mostrando sus tablas y campos para permitir una mayor comprensión de la explicación ofrecida sobre la implementación de las diferentes funcionalidades del sistema.

En primer lugar, se realiza el análisis de la base de datos interna, llamada *basedatos*ocpp, que consta de 6 tablas: *chargepoint_list*, *connector_list*, *meter_values*, *transaction*, *reservas* y *log*.

Nombre	Tipo
URI	varchar(255)
chargePointID	varchar(255)
lastStatusUpdate	datetime
chargeBoxSerialNumber	varchar(255)
chargePointModel	varchar(255)
chargePointSerialNumber	varchar(255)
chargePointVendor	varchar(255)
firmwareVersion	varchar(255)
iccid	varchar(255)
imsi	varchar(255)
meterSerialNumber	varchar(255)
meterType	varchar(255)
HeartBeatInterval	int(11)
status	varchar(255)

Ilustración 10 chargepoint_list

La tabla *chargepoint_list* contiene los diferentes puntos de carga del sistema y su información. Algunos de estos campos referentes a la información del punto de carga físico, como se verá más adelante al analizar el método Boot Notification, no serán conocidos para todos los CP ya que es información que no todos los fabricantes incluyen. Los campos más relevantes de esta tabla son:

- *chargePointID*, el identificador de los puntos de carga, que es único y se utiliza como clave primaria. Este identificador es de tipo *String*, tal y como indica la especificación de OCPP.

- *URI*, con la dirección IP del CP, que debido al gran número de accesos que hacen uso de este campo se ha decidido utilizar como índice

- *HeartBeatInterval*, que registra cada cuanto tiempo el punto de carga debe enviar el mensaje Heartbeat.

- *status* contiene el último estado conocido del punto de carga.

- *lastStatusUpdate* contiene la marca de tiempo del último estado conocido del punto de carga, permitiendo saber si la información sobre el estado es fiable.

Nombre	Tipo
connectorID	int(11)
chargePointID	varchar(255)
lastStatusUpdate	datetime
status	varchar(255)

Ilustración 11 connector_list

La tabla *connector_list* registra la información relativa a los diferentes conectores incluidos en los puntos de carga del sistema. Los campos *lastStatusUpdate* y *status* cumnplen con la misma función explicada anteriormente en la tabla *chargepoint_list*. En este caso, la clave primaria es doble, ya que el identificador del conector puede coincidir en varios conectores, pero es único para un CP dado.

Nombre	Tipo
connectorID	int(11)
chargePointID	varchar(255)
meterValues	varchar(255)
transactionID	int(11)
hora	datetime

Ilustración 12 meter_values

La función de la taba *meter_values* es mantener un registro con todas las muestras de valores del medidor eléctrico del punto de carga. Estos mensajes van asociados a un conector en un momento específico, por ello en la clave se han incluido las claves de la tabla

connector_list. Además, pueden ser valores referentes a la realización de una carga, por lo que se incluye el campo *transactionID*.

Nombre	Tipo
meterSTART	int(11)
transactionID	int(11)
connectorID	int(11)
CP	varchar(255)
hora	datetime
meterSTOP	int(11)

Ilustración 13 transaction

La tabla *transaction* mantiene un registro de las transacciones (cargas) que se realizan en los conectores de los diferentes puntos de carga. Estas transacciones disponen de un identificador y van asociadas a un punto de carga.

- *meterSTART* es el valor del medidor al iniciar la transacción.

- *meterSTOP* hace referencia al valor del medidor una vez finalizada la carga, por lo que es posible conocer la energía consumida en esa transacción. Al iniciar la transacción, el valor de este campo es nulo.

Nombre	Tipo
reservationID	int(11)
connectorID	int(11)
expiryDate	datetime
idTag	varchar(255)
CP	varchar(255)
status	varchar(255)

Ilustración 14 reservas

La taba *reservas* contiene todas las reservas de CP efectuadas, registrando el identificador de la reserva, el CP sobre el que se ha efectuado, el conector específico reservado (si se ha indicado alguno), la fecha en que deja de ser efectiva, la tarjeta que se utilizó para

realizarla y el estado en el que se encuentra, que puede ser “pendiente”, “cancelada” o “finalizada”.

Nombre	Tipo
emisor	varchar(255)
receptor	varchar(255)
parametros	varchar(255)
hora	datetime
metodo	varchar(255)

Ilustración 15 log

Con tal de mantener un registro con los mensajes intercambiados entre los puntos de carga y el CSS, se utiliza la tabla de *log*. En esta tabla se identifican los registros por el emisor (CSS o el URI del CP) y la marca temporal del mensaje. Además, el campo *parámetros* contiene la información del mensaje y el *método* hace referencia a la funcionalidad OCPP por la que se ha establecido la comunicación.

En segundo lugar, se procede al análisis de la base de datos para las autorizaciones (*basedatosauth*), que se compone únicamente de la tabla *auth_list*, donde se recogen las tarjetas de identificación válidas.

Nombre	Tipo
idTag	varchar(255)
status	varchar(255)
parentIdTag	varchar(255)
expiryDate	datetime

Ilustración 16 auth_list

El campo *idTag* contiene el código identificador de la tarjeta, y el *parentIdTag* es un campo opcional en el que se indica si existe una tarjeta “padre” de la que herede privilegios la tarjeta en cuestión. El campo *status* hace referencia al estado de autorización de ese identificador y *expiryDate* marca hasta cuando es válido ese estado.

Una vez explicada la arquitectura de clases seguida y las bases de datos utilizadas, se procederá al análisis de cada una de las funcionalidades contempladas en el protocolo OCPP, siguiendo el orden establecido en el capítulo 5.

6.2. Boot Notification

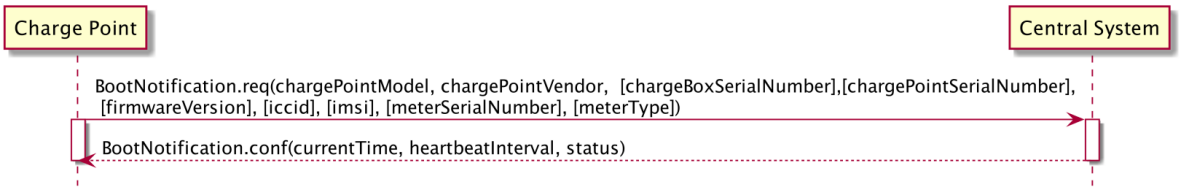


Ilustración 17 OCPP: Boot Notification

Al poner en marcha un CP conectado a la red, este envía una notificación al CSS con información sobre su configuración. El CSS decidirá si acepta el punto de carga, y en caso afirmativo, enviará una marca de tiempo para mantener la sincronía junto con el intervalo con el que el CP deberá ejecutar la llamada HeartBeat.

En este caso el CSS actúa como servidor, por lo que el CP se conectaría al web service de la clase *CentralSystem* y llamaría al método bootNotification( ), que recibe como parámetro un objeto BootNotificationRequest y devuelve un objeto BootNotificationResponse.

Los campos indicados entre corchetes [ ] son campos voluntarios, por lo que no todos los fabricantes incluyen esa información en sus mensajes Boot Notification, de modo que en la base de datos no se dispondrá de la misma información para todos los CP.

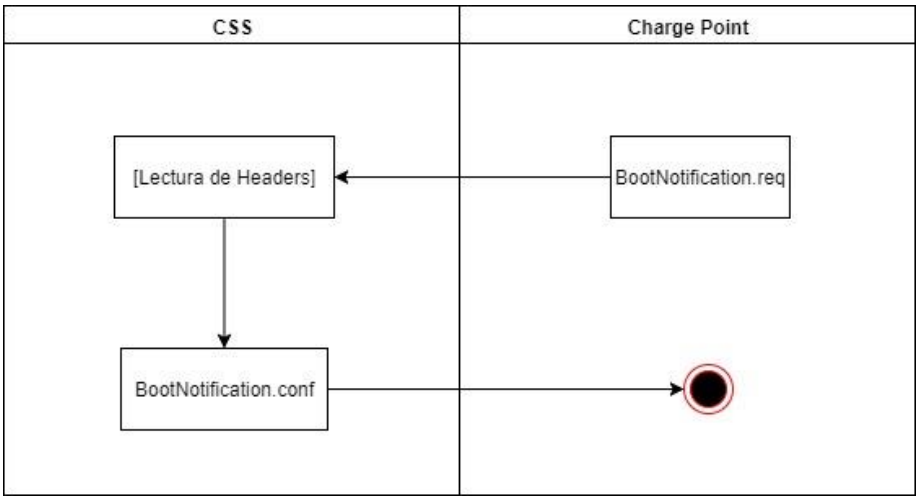


Ilustración 18 Boot Notification

En el caso del proyecto **Sistema central de gestión de puntos de carga** se ha optado por aceptar todos los puntos de carga que sean introducidos en la red, registrando en nuestra base de datos los CP desconocidos, tal y como se puede ver en la ilustración 18.

Una vez registrado un nuevo punto de carga en la tabla *chargepoint_list*, será necesario introducir en la base de datos de forma manual la información sobre los conectores del CP en la tabla *connector_list*, ya que para realizar el resto de funciones el CSS necesita conocer cuántos conectores tiene un CP y cuál es su identificador, pero dicha información no se incluye en la petición BootNotification.req.

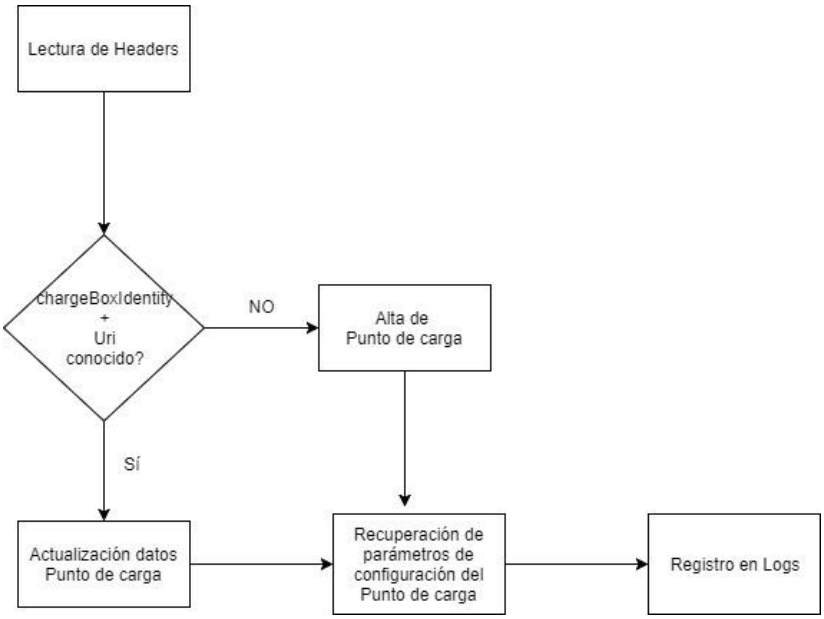


Ilustración 19 Boot Notification: Lectura de Headers

Para determinar si un CP es conocido, es necesario realizar una lectura de la cabecera del mensaje SOAP y comprobar que la combinación del identificador y URI indicados resida en la base de datos. Este mismo proceso se realizará de igual manera en el resto de llamadas iniciadas por los CP, por lo que su explicación no será repetida.

El alta y la actualización del punto de carga consisten en realizar un *insert* y un *update* respectivamente, en la tabla de puntos de carga de la BBDD con la información recibida en el mensaje BootNotification.req.

Para asignar el intervalo de *heartbeat* se obtendrá el registrado para ese CP en la base de datos si es un punto conocido, o se asignará el valor por defecto establecido en caso contrario.

El método bootNotification( ), antes de devolver la confirmación al CP, realiza una llamada al método registerBootNotification( ) de la clase *CommunicationsOUT*, que ofrece al resto de sistemas la información referente a la puesta en marcha del punto de carga que inició la petición; y registra en la base de datos de Logs la recepción del mensaje Boot Notification para mantener un registro con los mensajes intercambiados entre el CSS y los CP, incluyendo los parámetros indicados en el mensaje.

6.3. Status Notification

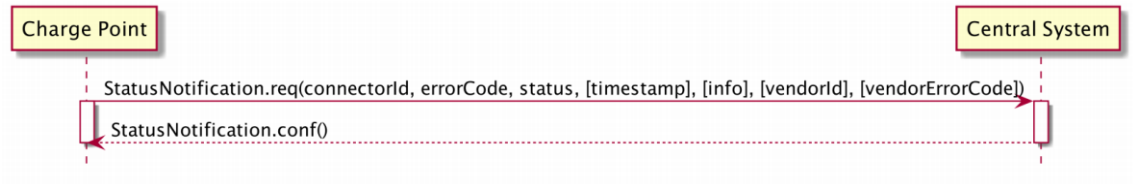


Ilustración 20 OCPP: Status Notification

El mensaje Status Notification es enviado al CSS por el punto de carga para informar cuando en el CP se ha producido una condición de error o se haya ejecutado un comando *Change Availability* programado.

Igual que en el caso anterior, Status Notification es una funcionalidad iniciada por los CP, por lo que la parte del CSS que actúa es la parte "*servidor*", conectándose los CP mediante la clase web service *CentralSystem* al método statusNotification( ). Este método recibe un objeto StatusNotificationRequest con la información relativa al error y devuelve un objeto StatusNotificationResponse.

En este caso, los campos voluntarios serán obviados, ya que además de depender del fabricante, no aportan información necesaria para analizar el error producido.

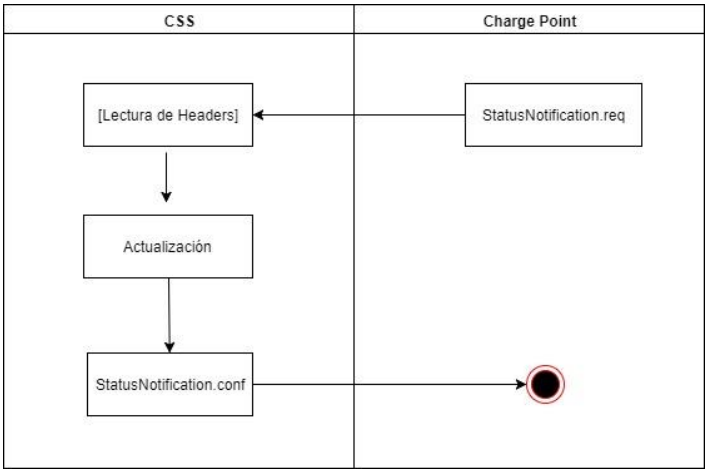


Ilustración 21 Status Notification

A diferencia de lo que ocurría en Boot Notification, tanto en Status Notification como en los próximos métodos iniciados por los CP, si la comunicación la ha realizado un punto de carga no registrado en la base de datos, el mensaje será descartado lanzando una excepción de "*ChargePointIDDesconocido*", ya que se trataría de un CP que se ha unido a la red sin pasar por el flujo de mensajes correcto al no haber ofrecido la información de Boot Notification, y se registrará la recepción del mensaje en los Logs indicando que el punto de carga no es conocido por el sistema.

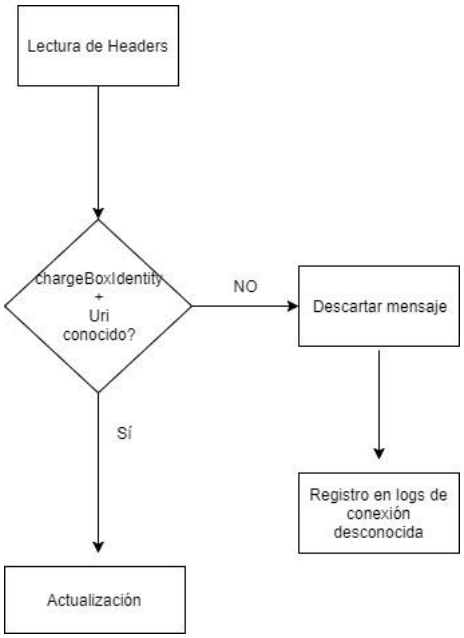


Ilustración 22 Status Notification: Lectura de Headers

Si el CP es conocido, se procederá a la actualización del estado en las bases de datos. Tal y como se muestra en la ilustración 22, dependiendo del conector indicado en los parámetros, se realizará la actualización de un único conector o del punto de carga al completo, incluyendo la actualización de todos sus conectores en la tabla de conectores y la actualización del estado en la tabla de puntos de carga.

Dicha actualización consiste en actualizar la columna *status* de la tabla de conectores para el conector indicado o para todos los del CP indicado y realizar la misma acción para la tabla de CP en caso de ser necesario, y modificar el parámetro *lastStatusUpdate*.

Una vez realizadas las actualizaciones en la base de datos, se registra en los Logs la recepción del mensaje y sus parámetos y se realiza una llamada al método registerStatusNotification( ) de la clase *CommunicationsOUT* para ofrecer la información sobre el error al resto de sistemas.

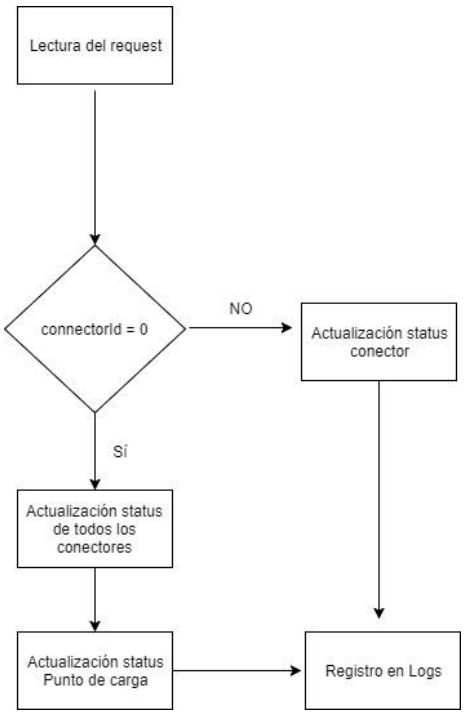


Ilustración 23 Status Notification: Actualización

6.4. Heartbeat

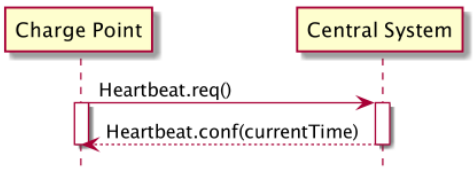


Ilustración 24 OCPP: Hertbeat

Heartbeat es un mensaje enviado por los CP periódicamente, con la frecuencia indicada por el CSS en la confirmación del mensaje Boot Notification, para informar al CSS de que el punto de carga emisor permanece operativo, a lo que el CSS responde con una confirmación del mensaje que incluye la marca de tiempo para mantener la sincronía.

Igual que en los métodos anteriores, en este caso el CSS actúa como servidor y los CP acceden al método heartbeat( ) de la clase *CentralSystem*.

En este punto es necesario remarcar la problemática de asignar incorrectamente el intervalo de Heartbeat, ya que como se ha explicado en el capítulo 3 del presente documento, debido a la reciente estabilidad de la versión 1.6 de OCPP que incluye la posibilidad de enviar mensaje en formato JSON, este proyecto ha optado por centrarse en la versión 1.5, que únicamente permite mensajes SOAP, cuyos requisitos de red son mucho mayores. Por lo tanto, es necesario conocer si los CP estarán conectados a la red por tecnologías Ethernet, en cuyo caso el flujo de datos no supondría un problema, o si lo hará mediante tecnología móvil (3G, 4G o 5G), requiriendo espaciar más en el tiempo los mensajes Heartbeat.

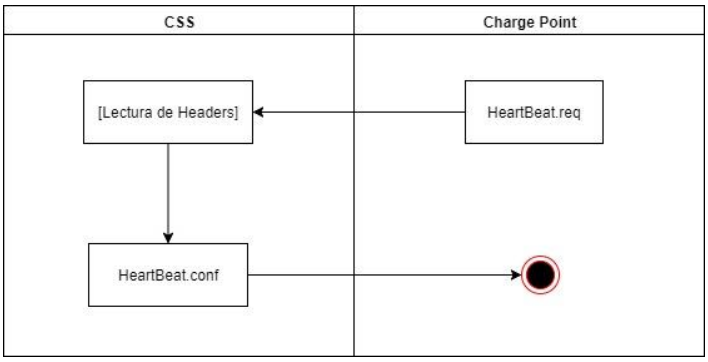


Ilustración 25 Heartbeat

El funcionamiento del método heartbeat( ) es bastante simple, ya que en caso de no conocer el CP realiza las mismas acciones que el método anterior, y si el punto es conocido marca el *status* de la tabla *chargepoint_list* a "operativo" y actualiza la fecha del *lastStatusUpdate*.

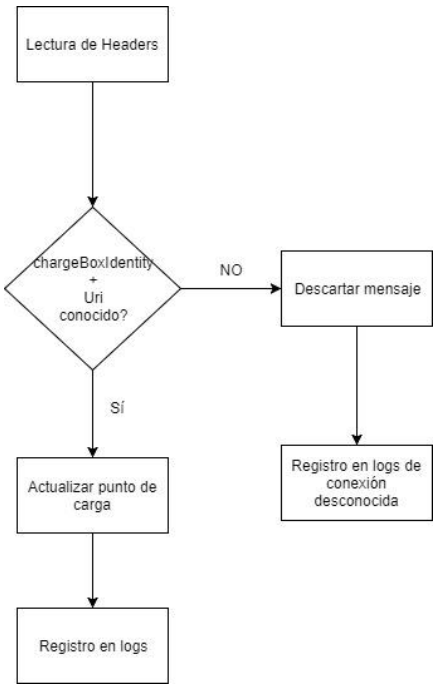


Ilustración 26 Heartbeat: Lectura de Headers

Al finalizar las gestiones del método, se registra la recepción del mensaje en la tabla *log* y se realiza una llamada a la clase *CommunicationsOUT* para informar a los sistemas necesarios que el CP se encuentra operativo mediante el método registerHearthBeat( ).

6.5. Reset

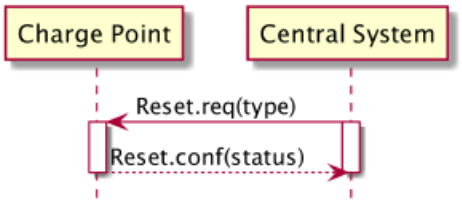


Ilustración 27 OCPP: Reset

Reset es el primer método implementado iniciado por el CSS. En este caso, el CSS actúa como cliente del web service del punto de carga, por lo que el método reset( ) está contenido en la clase ChargePoint_WSClient. El CSS puede solicitar un reinicio de software o hardware en el CP, que indica en el parámetro *type* de la petición. Si en el CP hay una transacción en curso, esta se finalizará de manera normal antes de proseguir al reinicio.

La decisión de reiniciar un punto de carga no la toma el CSS, sino el sistema de gestión en el que esté integrado, por lo que se comunicará la necesidad de realizar dicha acción mediante el web service de la clase *CommunicationsIN*. En esta clase se recibiría el tipo de reinicio como una variable de tipo String, que se enviaría al método reset( ) y sería transformado en este en un parámetro propio del protocolo OCPP que se incluiría en la construcción del ResetRequest.

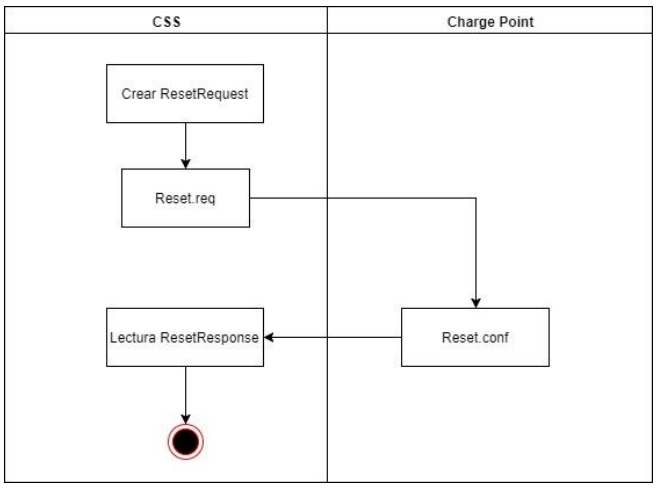


Ilustración 28 Reset

Una vez enviada la petición al web service del CP, el CSS comprueba el parámetro *status* indicado por el punto de carga en la respuesta. Dicho parámetro indica si el CP se encuentra en la capacidad de proceder al reinicio.

En caso de no recibir respuesta, se registra en la tabla de *log* el error en la recepción y se comunica el mismo al sistema que originó la petición.

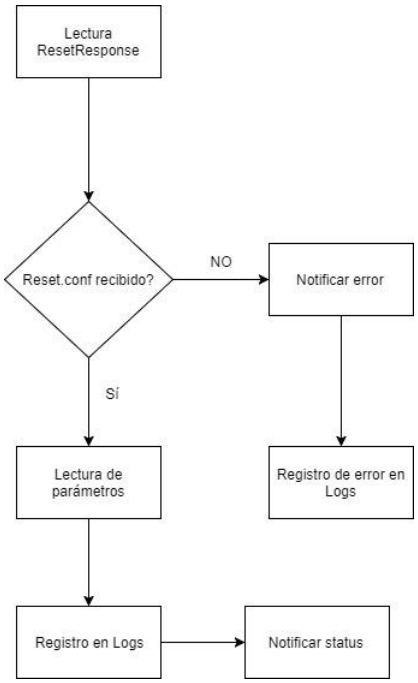


Ilustración 29 Reset: Lectura de response

Una vez analizada la respuesta, el método registra en la base de datos la recepción del mensaje y retorna a la clase *CommunicationsIN* el estado indicado por el CP, en formato de String para comunicarlo al sistema que originó la petición.

6.6. GetConfiguration

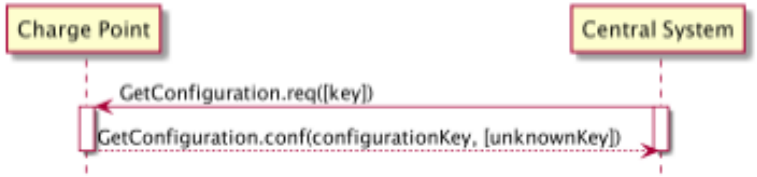


Ilustración 30 OCPP: GetConfiguration

GetConfiguration es un método iniciado por el CSS, por lo que la parte del CSS que actúa es la parte "*cliente*". Mediante el web service de la clase *CommunicationsIN*, los sistemas externos solicitan al CSS desean recuperar información de configuración del CP y esto se transforma en el método getConfiguration( ) de la clase ChargePoint_WSClient en un parámetro propio de OCPP para su envío al CP.

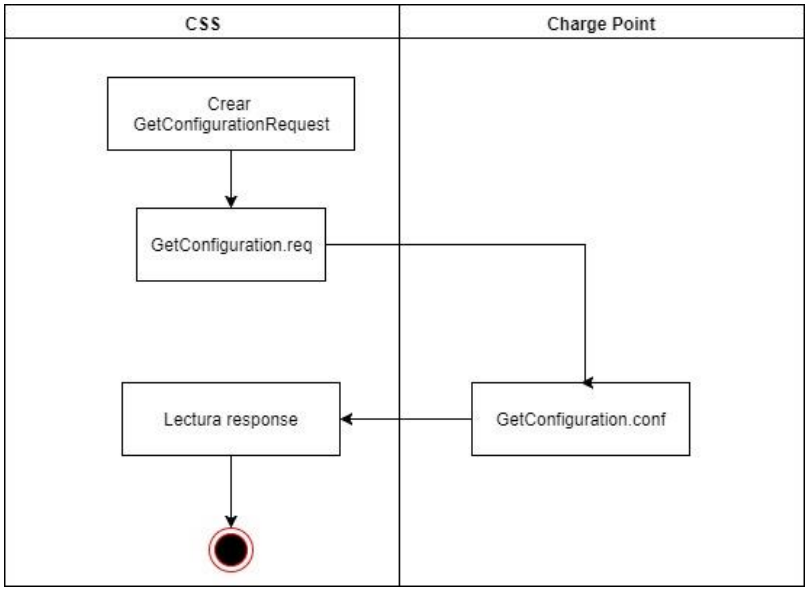


Ilustración 31 GetConfiguration

El primer paso consiste en la creación de la de una petición OCPP. En ella se solicita toda la información de configuración a través del web service del CP. En caso de no recibir contestación por parte del CP una vez agorado el time out, el CSS registra en la tabla de *log* la no recepción del mensaje i comunica el error al sistema que originó la petición.

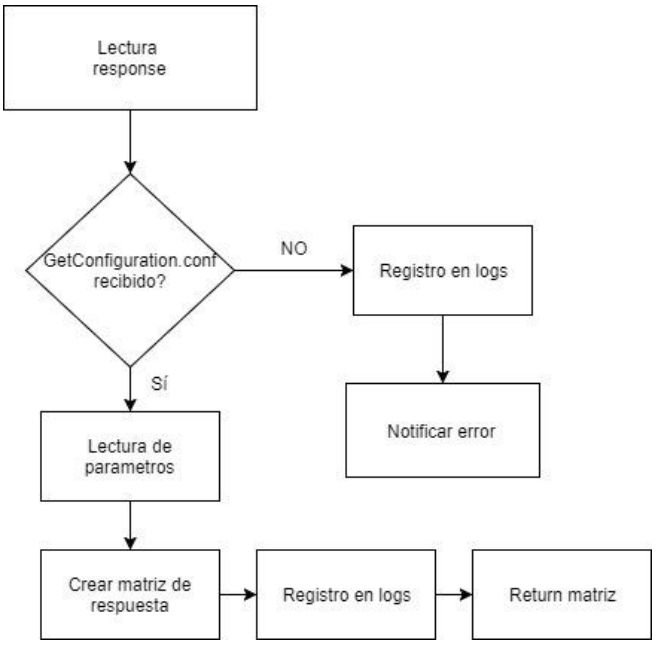


Ilustración 32 GetConfiguration: Lectura de response

Si se ha recibido una respuesta del CP, se crea una matriz de 2x50 en la que se incluyen las relaciones [clave de configuración, valor de configuración] indicadas en la respuesta. A continuación, se registra la recepción del mensaje en la tabla *log* y se envía la matriz creada al sistema que originó la petición en la clase *CommunicationsIN*.

6.7. ClearCache

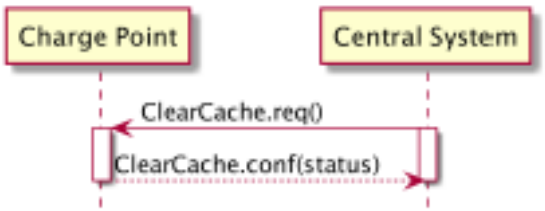


Ilustración 33 OCPP: ClearCache

Los puntos de carga mantienen en cache un registro de autorización local con los identificadores que han sido autorizados previamente. Cuando uno de los sistemas externos desea que un CP elimine su caché de autorizaciones, accede al CSS mediante el web service de la clase *CommunicationsIN*. Este método no requiere de parámetros en su petición, por lo que lo único que debe indicar el sistema es el CP en el que se debe vaciar la caché. En este método toma parte de nuevo el CSS como "*cliente*", al ser un método iniciado por el mismo.

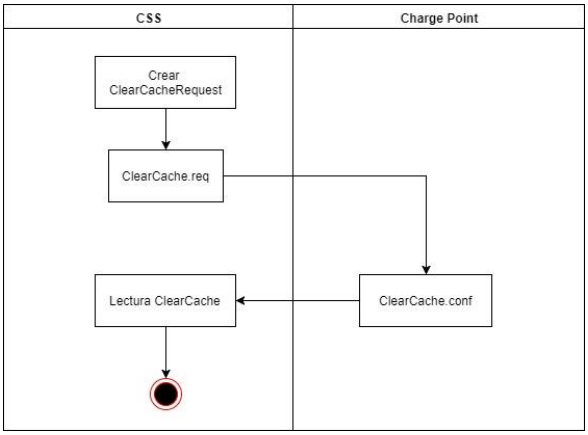


Ilustración 34 ClearCache

Una vez enviada la petición al web service del CP, el CSS comprueba el parámetro *status* indicado por el punto de carga en la respuesta. Dicho parámetro indica si el CP se encuentra en la capacidad de proceder al vaciado de caché.

En caso de no recibir respuesta, se registra en la tabla de *log* el error en la recepción y se comunica el mismo al sistema que originó la petición.

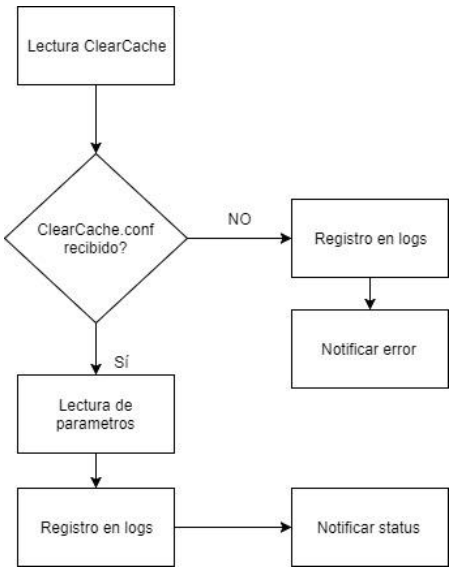


Ilustración 35 ClearCache: Lectura de response

Una vez analizada la respuesta, el método registra en la tabla de *log* la recepción del mensaje y retorna a la clase *CommunicationsIN* el estado indicado por el CP, en formato de String, para comunicarlo al sistema que originó la petición.

6.8. Authorize

Ilustración 36 OCPP: Authorize

Antes de poder iniciar una transacción, el propietario del vehículo debe identificarse en el punto de carga con su tarjeta. De igual manera, para finalizar la transacción y poder retirar el vehículo del conector, es necesario identificarse y recibir autorización si la tarjeta utilizada no es la misma que se empleó para el inicio. Es posible indicar a los CP que hagan uso de su cache para realizar la autorización, evitando así este proceso, más para el proyecto **Sistema central de gestión de puntos de carga** se ha decidido que únicamente harán uso de su lista de autorización local cuando no sea posible establecer comunicación con el CSS, favoreciendo la disponibilidad de los CP.

Este método es iniciado por el CP y el CSS actúa como "*servidor*". El punto de carga indica el identificador de la tarjeta, de tipo String, y el CSS comprueba en el sistema de autorización si dicho identificador tiene permitido realizar la acción.

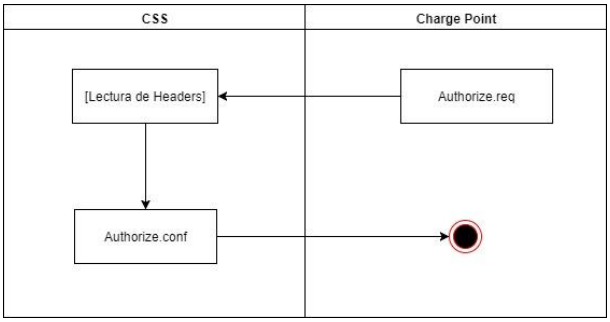


Ilustración 37 Authorize

El método authorize( ) comprueba que la combinación de URI e identificador de CP indicados en la cabecera del mensaje sea conocida por el sistema. En caso de no conocer el punto de carga, el CSS descarta el mensaje y eleva la excepción de CP desconocido y registra el mensaje en la base de datos indicando en los parámetros dicho error.

Si el CP es conocido, el CSS establece comunicación mediante la clase *CommunicationsOUT* con el sistema de autorización. Como ya se ha dicho anteriormente, para el desarrollo del proyecto se ha utilizado una base de datos como sistema de autorización, por lo que la clase *Authorization* del paquete *platforms* contiene consultas a la base de datos.

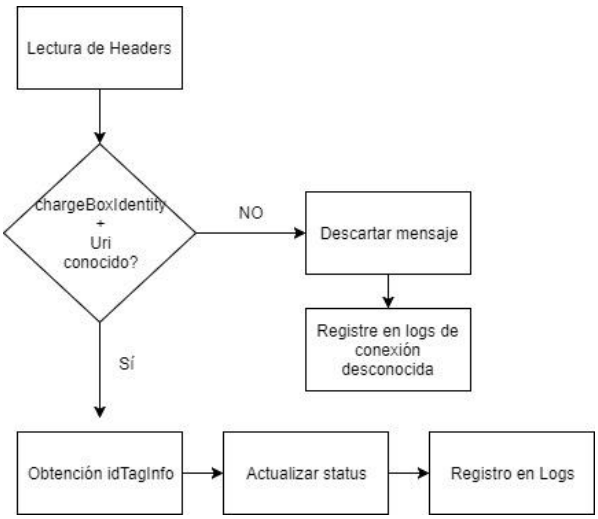


Ilustración 38 Authorize: Lectura de headers

La información obtenida en el método getIdTagInfo( ) de la clase *CommunicationsOUT* contiene el estado de autorización del identificador, fecha de expiración y la tarjeta "padre" de la que depende.

Al haber recibido este mensaje del CP, podemos determinar que el CP está operativo, por lo que actualizamos en la tabla *chargepoint_list* de la base de datos, los parámetros *status* y *lastStausUpdate*.

Antes de enviar la respuesta con la información de la tarjeta al CP, el CSS registra el mensaje en la tabla *log* y comunica mediante la clase *CommunicationsOUT* a los sistemas externos sobre la petición del CP.

6.9. ReserveNow

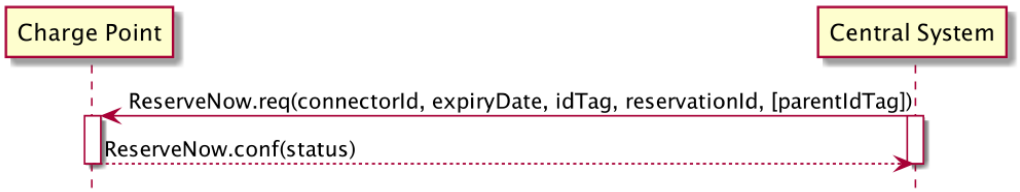


Ilustración 39 OCPP: ReserveNow

Un usuario puede solicitar la reserva de un CP durante los 30 minutos a próximos a realizar la petición. Si finalizado este tiempo no se ha hecho uso de ella, la reserva se cancela y se libera el CP.

El usuario puede acceder al CSS mediante el web service de la clase *CommunicationsIN* para solicitar realizar una reserva indicando su tarjeta de identificación y si desea reservar un conector específico. El CSS transforma esta información en parámetros de OCPP y envía la petición al CP.

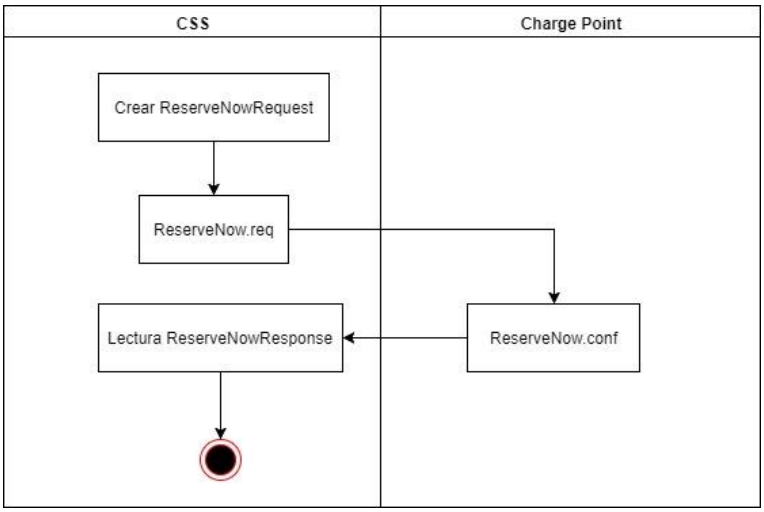


Ilustración 40 ReserveNow

Si el CP no envía una respuesta antes de superar el time out, el CSS registra ese error en la tabla *log* y notifica la falta de respuesta al sistema que originó la petición.

La respuesta del CP contiene el estado de la petición, que puede ser "aceptada", "ocupado" (si el CP o el conector se encuentran en una transacción o han sido reservados), "indisponible" (si el CP se encuentra en estado de indisponibilidad, estado que se verá en el apartado 6.13), "rechazado" (si el CP no está configurado para aceptar reservas) o "fallido" (si el CP se encuentra en estado de fallo).

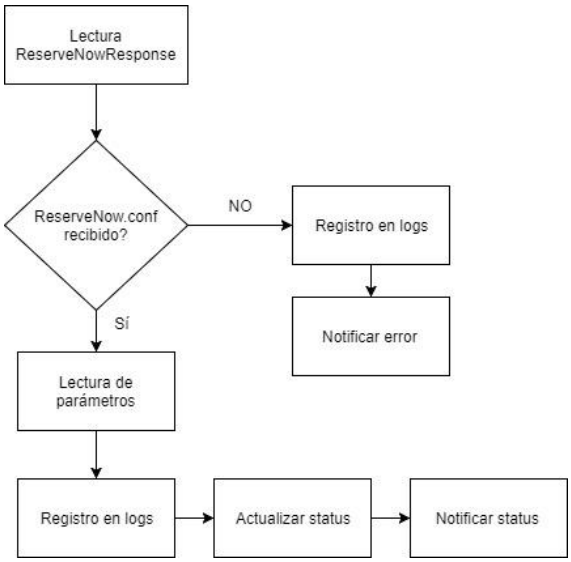


Ilustración 41 ReserveNow: Lectura de ResetResponse

Una vez analizada la respuesta, el método registra en la tabla de *log* la recepción del mensaje y retorna a la clase *CommunicationsIN* el estado indicado por el CP, en formato de String, para comunicarlo al sistema que originó la petición. Si la reserva ha sido aceptada, se registra en la tabla *reservas* de la base de datos interna. El estado indicado es el mismo que se utilizará para actualizar en la tabla *chargepoint_list* el *status* y el *lastStatusUpdate*. En caso de

que el conector indicado sea mayor que 0, significa que se desea reservar un conector específico, por lo que se actualizan los mismos parámetros para la tabla *connector_list*.

Para facilitar la gestión en el sistema externo, el identificador de reserva es asignado en el CSS y se retorna al finalizar la solicitud de reserva.

6.10. CancelReservation

Ilustración 42 OCPP: CancelReservation

Si un usuario desea eliminar una reserva realizada, puede acceder al CSS mediante el web service *CommunicationsIN* para iniciar el método cancelReservation( ), indicando la reserva que desea eliminar.

El CSS incluye el identificador de reserva indicado en un mensaje OCPP de tipo CancelReservationRequest y lo envía al CP correspondiente.

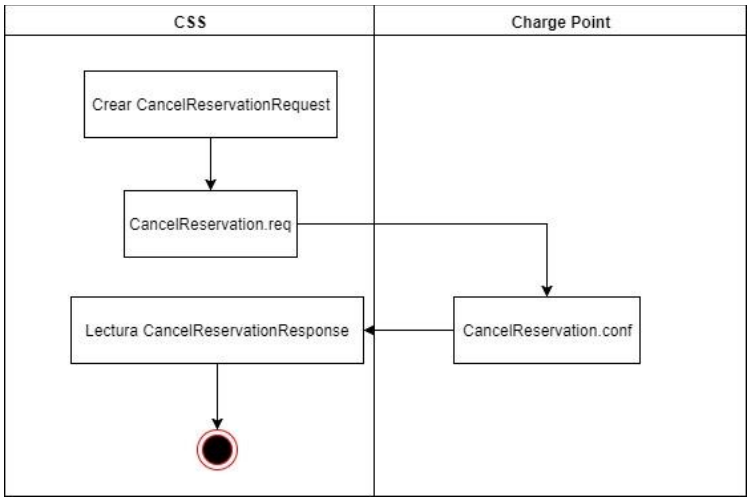


Ilustración 43 CancelReservation

La respuesta del CP incluye un estado que indica si se ha encontrado la reserva indicada. En ese caso, el estado es “aceptado” y es “rechazado” en caso contrario. Si ha sido aceptada a cancelación, se actualiza e *status* de la tabla *reservas* a “cancelada” y se actualiza el estado del CP en la tabla *chargepoint_list* a disponible. Además, si la reserva hacía referencia a un conector, se actualiza también el estado del mismo en la tabla *connector_list*. Los identificadores del CP y el conector se obtienen consultando la reserva en las BBDD.

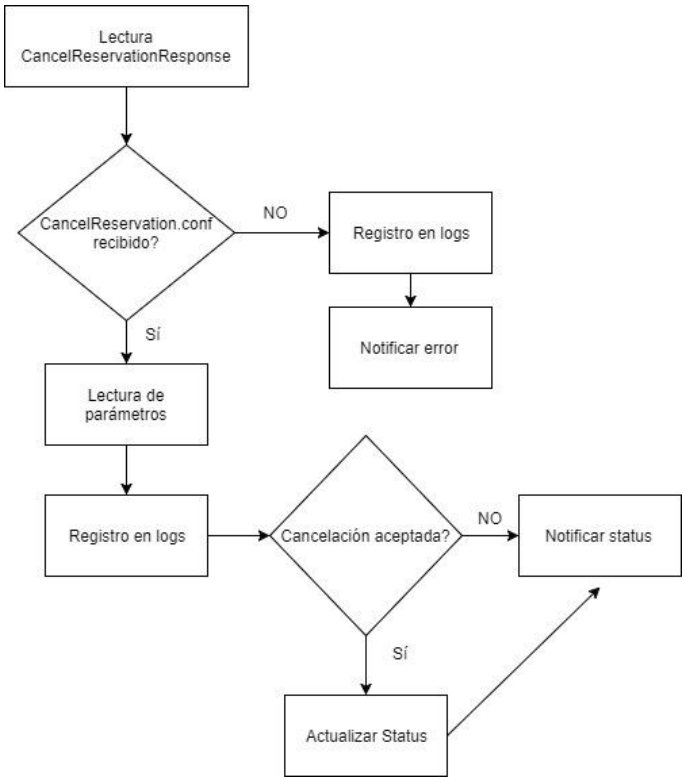


Ilustración 44 CancelReservation

Si el CP no ha dado contestación al mensaje, se registra el error en la table *log* y se comunica al sistema que originó la petición la imposibilidad de establecer comunicación.

6.11. StartTransaction

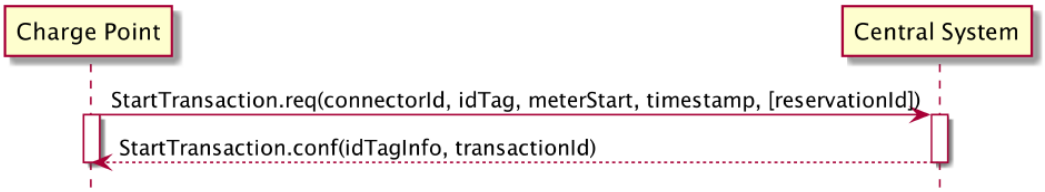


Ilustración 45 OCPP: StartTransaction

Un punto de carga envía una petición del método StartTransaction para informar al CSS de que se va a producir una operación de carga. En el caso de que dicha transacción finalice una reserva, el identificador de la misma se incluirá en la petición.

La parte del CSS que toma parte en este proceso es la parte "*servidor*", ya que el CP se debe conectar mediante el web service de la clase *CentralSystem*.

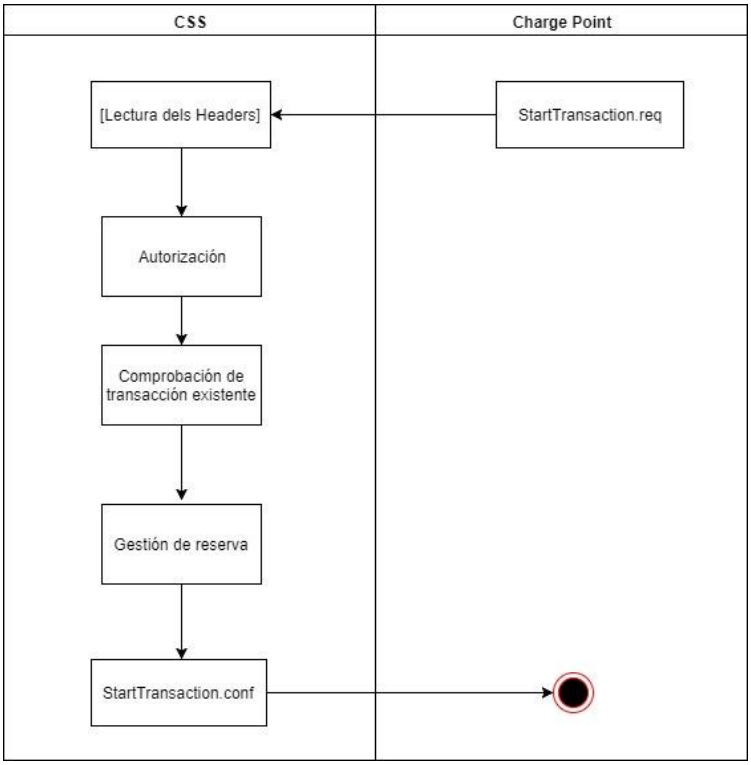


Ilustración 46 StartTransaction

Al inicio del método se comprueba que el CP indicado en la cabecera del mensaje SOAP sea conocido, descartando el mensaje y registrando el error en la tabla de logs en caso contrario. Si el CP es conocido, se obtienen los parámetros de autorización de manera similar al método anterior.

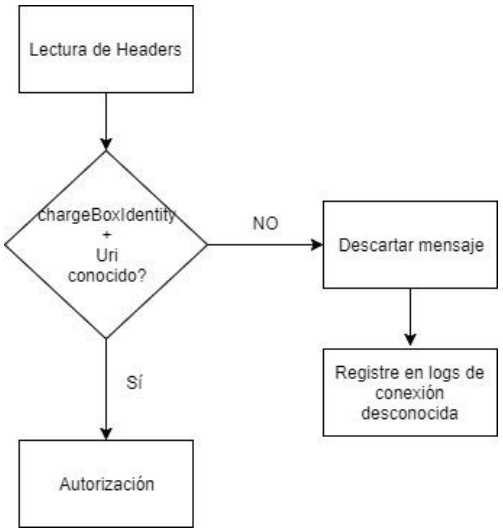


Ilustración 47 StartTransaction: Lectura de headers

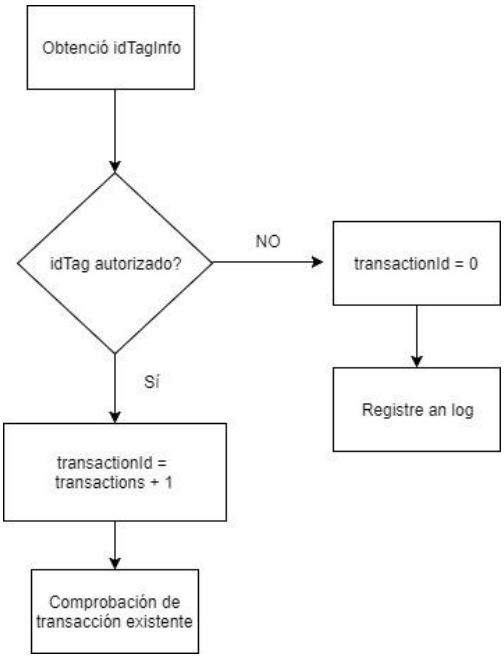


Ilustración 48 StartTransaction: Autorización

El siguiente paso realizado por el método es comprobar en la base de datos si existe una transacción existente (con *meterSTOP* = NULL) para el conector indicado. Si se da este caso, significa que el mensaje ha sido erróneo, por lo que el mensaje es descartado elevando una excepción en el CSS.

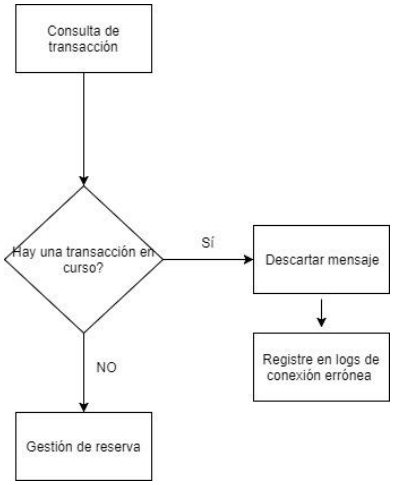


Ilustración 49 StartTransaction: Comprobación de transacción existente

Una vez comprobado que el mensaje no se ha producido por error, se analiza si se ha indicado una reserva en los parámetros, indicando que se hace efectiva la misma. Si la reserva indicada existe en la base de datos como una reserva en curso, se actualiza el *status* a finalizada.

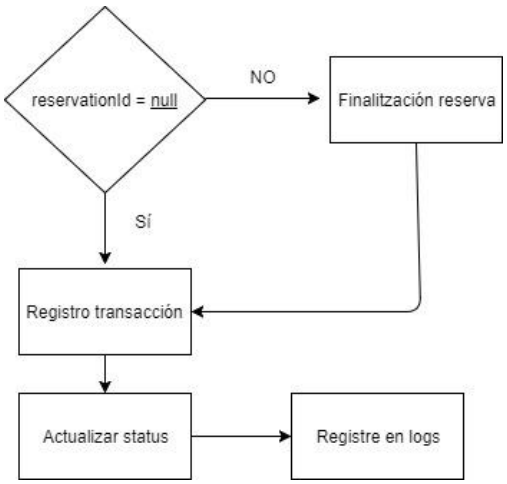


Ilustración 50 StartTransaction: Gestión de reserva

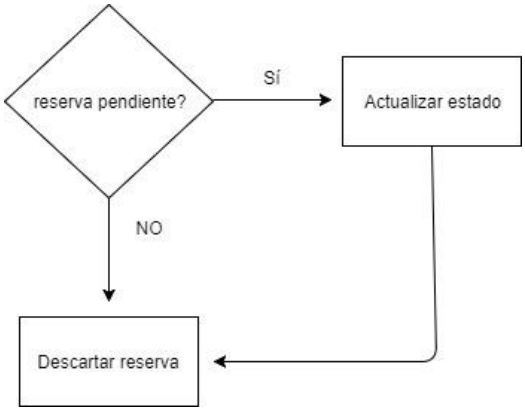


Ilustración 51 StartTransaction: Finalización de reserva

Al haber recibido este mensaje del CP, podemos determinar que el CP está operativo, y que está o va a estar ocupado, haya una transacción en curso o no, ya que va a iniciarse una, por lo que actualizamos en la tabla *chargepoint_list* de la base de datos, los parámetros *status* y *lastStausUpdate*.

Si las condiciones permiten el inicio de la transacción, se asigna como identificador un número que representa la cantidad de transacciones realizadas, incluyendo la misma. La transacción se registra en la base de datos, se comunica a los sistemas de gestión externos el inicio de la operación y se procede al envío de la confirmación.

6.12. MeterValues

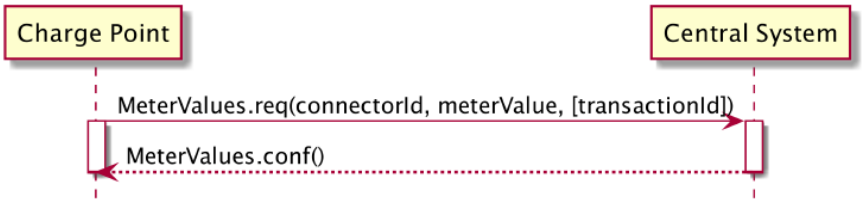


Ilustración 52 OCPP: MeterValues

Los puntos de carga envían al CSS cada cierto tiempo información sobre los valores de sus medidores eléctricos. Estos valores van relacionados con un conector en específico y pueden hacer referencia a los valores de carga de una transacción. El CSS actúa como servidor al ser un proceso iniciado por el CP. Cada elemento meterValue contiene una marca de tiempo y un conjunto de muestras, tomadas todas en el mismo momento.

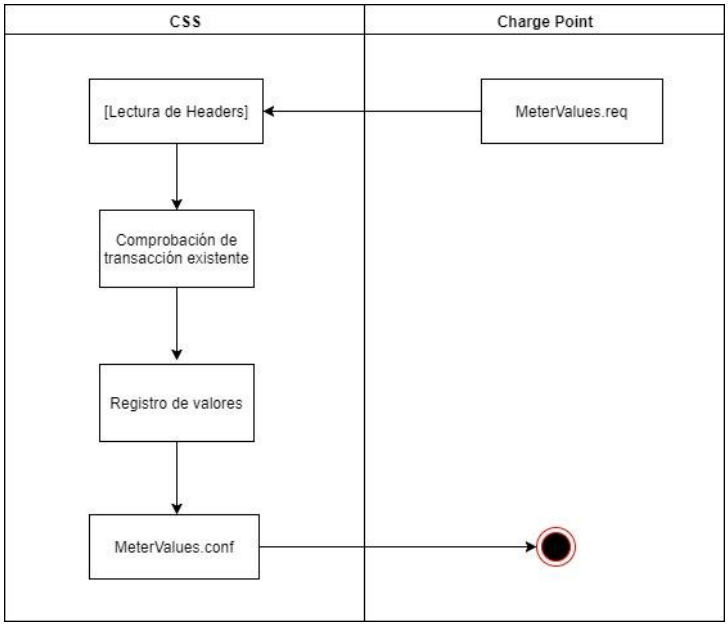


Ilustración 53 MeterValues

Al recibir el mensaje, el CSS comprueba si el CP es conocido y descarta el mensaje en caso contrario. A continuación, se analiza si se ha indicado un identificador de una transacción en marcha. En caso afirmativo, se registran todas las muestras recibidas en la tabla *meterValues* indicando el parámetro *transactionID*.

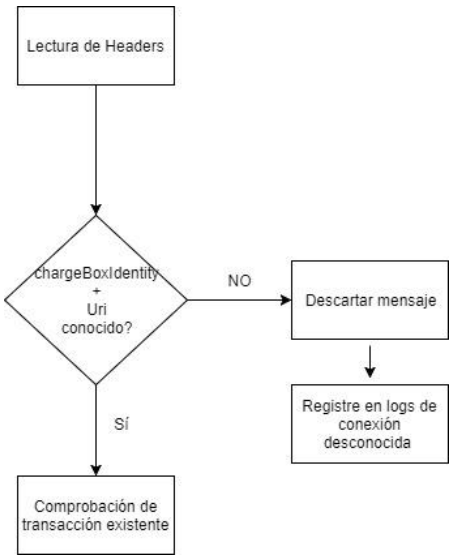


Ilustración 54 MeterValues: Lectura de headers

Si no se ha indicado una transacción o la transacción indicada ya ha finalizado, las muestras se registran dejando el parámetro *transactionID* como NULL.

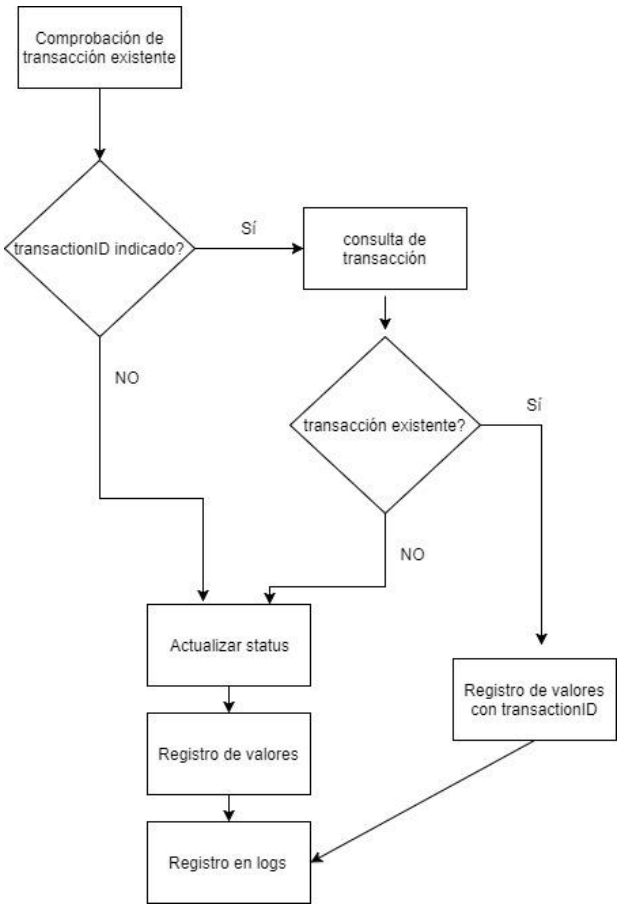


Ilustración 55 MeterValues: Comprobación de transacción

Al finalizar el registro de valores, se registra el mensaje en la tabla de *log* y se comunica a los sistemas externos los valores indicados por el CP mediante la clase *CommunicationsOUT*. Finalmente, se envía la confirmación al punto de carga.

Al haber recibido este mensaje del CP, podemos determinar que el CP está operativo, por lo que actualizamos en la tabla *chargepoint_list* de la base de datos, los parámetros *status* y *lastStausUpdate*.

La comunicación de estos valores deberá ser estudiada en proyectos posteriores, cuando se desarrolle la integración del proyecto **Sistema central de gestión de puntos de carga** con los diferentes sistemas externos de gestión, ya que es posible que, al ser todos los valores muestras tomadas en el mismo instante de tiempo, no sea necesario comunicarlos todos.

6.13. ChangeAvailability

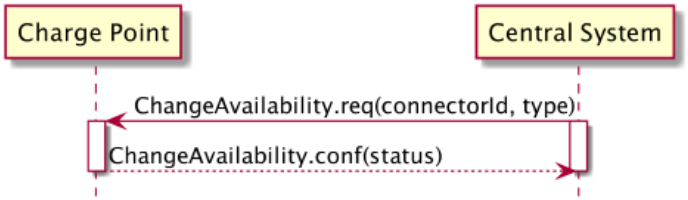


Ilustración 56 OCPP: ChangeAvailability

El CSS puede solicitar a un punto de carga que modifique su disponibilidad mediante este procedimiento. Un CP se considera "operativo" o "disponible" cuando está cargando o preparado para realizar una carga. En caso de no permitir una carga, el CP se encuentra "indisponible". Este método permite cambiar el CP de "disponible" a "indisponible" y viceversa.

Uno de los sistemas externos se conecta al CSS mediante el web service de la clase *CommunicationsIN* indicando el CP al que se desea modificar la disponibilidad, el CP sobre el que se desea aplicar el cambio y el estado al que se desea cambiar. El CSS construye la petición y se conecta al web service del CP mediante el cliente de la clase *ChargePoint_WSClient*.

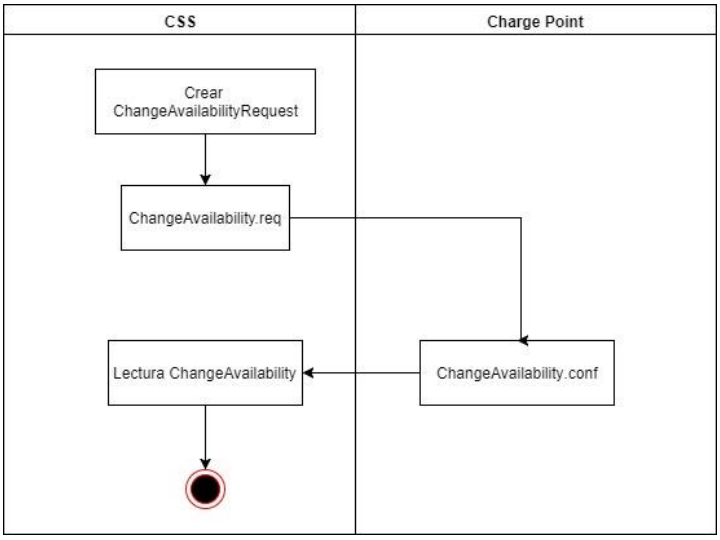


Ilustración 57 ChangeAvailability

Una vez enviada la petición al CP, se espera la confirmación enviada por el mismo. En caso de no recibirse una vez superado el *timeout*, el CSS registra la falta de mensaje en la tabla *log* y notifica al sistema que originó la petición que no se ha producido respuesta por parte del CP.

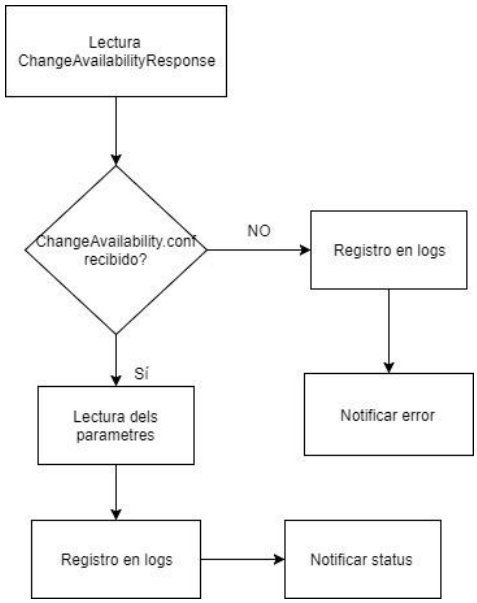


Ilustración 58 ChangeAvailability: Lectura de response

Al recibir la confirmación del mensaje, el CSS comprueba el estado indicado por el CP en el mismo, registra su recepción en la tabla *log* y comunica al sistema original si el CP ha cambiado su disponibilidad o si se encontraba en la realización de una transacción, por lo que ha programado el cambio para el momento en que la misma finalice.

6.14. UnlockConnector

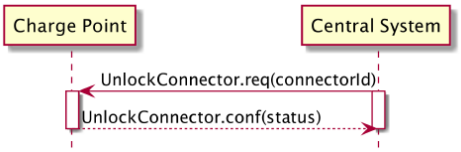


Ilustración 59 OCPP: UnlockConnector

Este método está pensado especialmente para tareas de mantenimiento o para evitar problemas de funcionamiento del CP al iniciar una transacción. De esta manera, si un usuario desea iniciar una carga pero el CP no desbloquea el cargador para iniciarla, se puede solicitar al CSS que fuerce el desbloqueo del conector. La petición al CSS se realiza mediante la clase *CommunicationsIN*.

UnlockConnector es un método iniciado por el CSS, por lo que la parte que actúa es el "*cliente*", que se conecta al web service del CP.

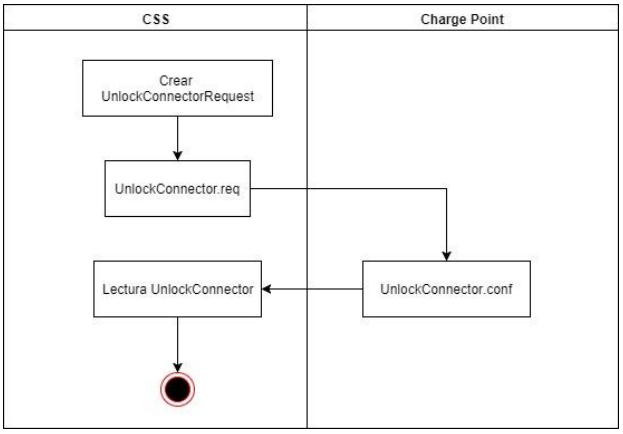


Ilustración 60 UnlockConnector

Al enviar la petición, el CSS espera la confirmación del CP con el estado de desbloqueo. Este estado indica si es posible desbloquear el conector, o si hay una transacción en curso, siendo necesario finalizar primero la misma.

Si el CP no envía una confirmación, el CSS registra el error en la tabla de *log* de la base de datos y comunica al sistema que originó la petición la falta de respuesta.

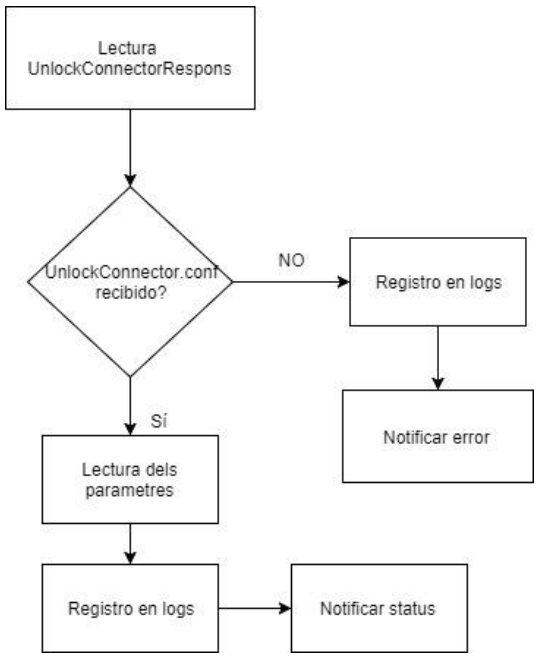


Ilustración 61 UnlockConnector: Lectura de response

Al recibir la confirmación del CP, el CSS analiza el estado indicado, registra el mensaje en la tabla *log* y comunica al sistema si el CP ha desbloqueado el conector o si hay una transacción en curso, por lo que se deberá esperar a que finalice o forzar su fin mediante el método RemoteStopTransaction.

6.15. StopTransaction

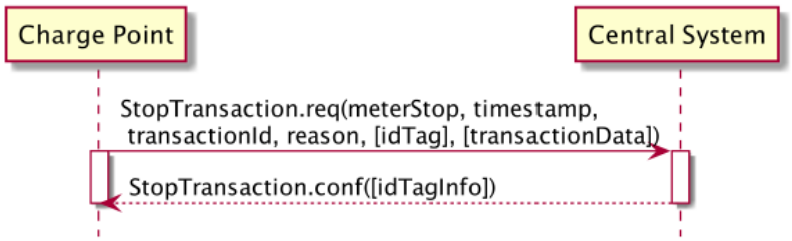


Ilustración 62 OCPP: StopTransaction

El método StopTransaction es iniciado por un punto de carga que ha finalizado una operación de carga para informar de ese hecho. En el mensaje se incluye el momento en que ha finalizado la carga, el valor del medidos del conector que ha realizado la operación y el identificador de la misma. De manera opcional, se puede comunicar la razón por la que ha finalizado la carga (Local si ha sido de manera natural), el identificador utilizado para finalizar la carga (si se ha utilizado) y detalles de consumo eléctrico durante la transacción.

A pesar de que los campos opcionales permitidos por OCPP resultarían de gran utilidad para la gestión de la red, debido a la libre interpretación que se hace de su uso por los diferentes fabricantes, en el proyecto **Sistema central de gestión de puntos de carga** se ha decidido

obviarlos, ganando de esta manera compatibilidad con todos los fabricantes que incorporen OCPP 1.5 en sus puntos de carga.

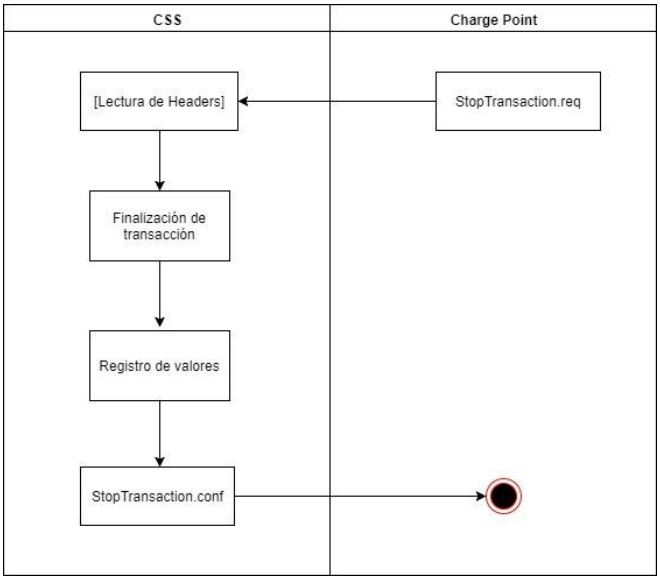


Ilustración 63 StopTransaction

Al ser iniciado por el CP, es un método en el que actúa la parte "*servidor*" del CSS a través de la clase *CentralSystem*. El método stopTransaction( ) analiza la cabecera del mensaje para comprobar que haya sido enviado desde un CP conocido por el sistema, descartando el mensaje en caso contrario.

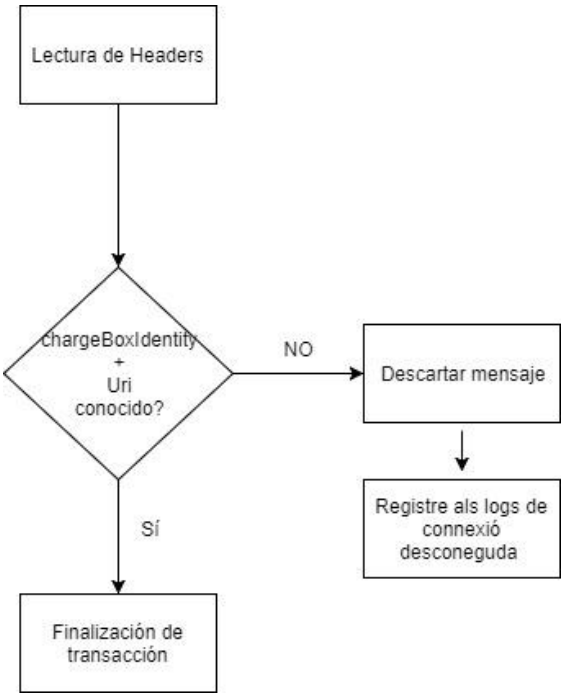


Ilustración 64 StopTransaction: Lectura de Headers

Si el CP es conocido, se comprueba que la transacción indicada en el mensaje exista y esté en curso. En caso negativo, se registra la incidencia y se descarta el mensaje.

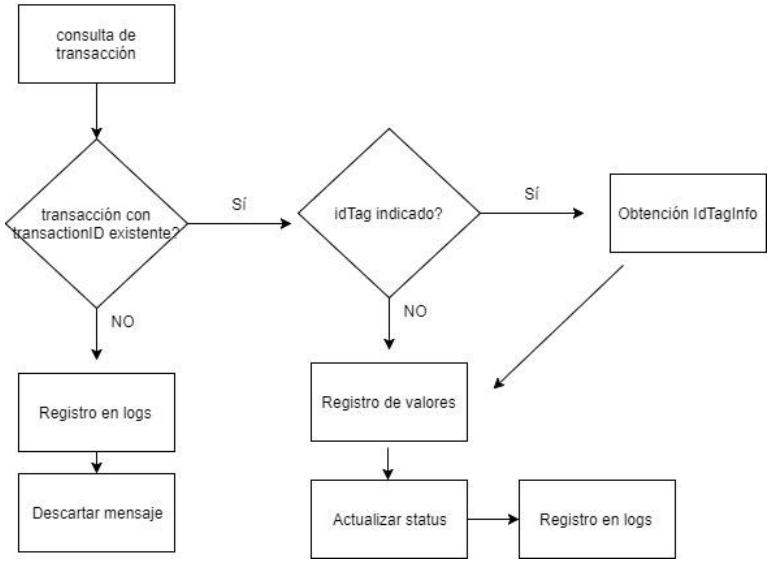


Ilustración 65 StopTransaction: finalización de transacción

Una vez comprobada la existencia de la transacción, el CSS establece comunicación con el sistema de autorización para obtener la información relativa al identificador de usuario, en caso de haberse indicado. Este parámetro opcional se decidido tener en cuenta ya que es usado por el CP para actualizar su lista local de autorizaciones, que necesitaría en caso de perder la conexión con el CSS.

Si existía una transacción, el CP pasa a estar disponible, por lo que actualizamos en la tabla *chargepoint_list* de la base de datos, los parámetros *status* y *lastStausUpdate*.

Tras obtener la información, o si no se ha indicado un identificador, se registran los valores de medición ofrecidos en meterStop como parámetro *meterSTOP* de la transacción correspondiente en la tabla *transaction*, se registra el mensaje en la tabla de *log* y se comunica a los sistemas externos que la transacción ha finalizado.

6.16. RemoteStartTransaction

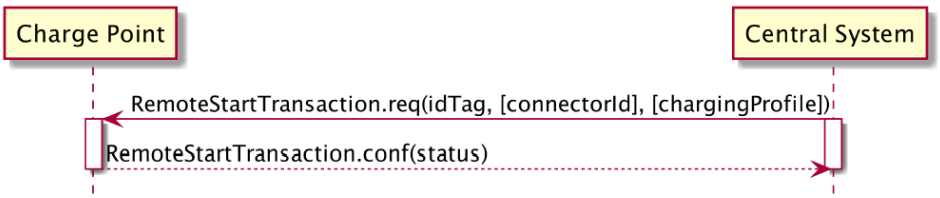


Ilustración 66 OCPP: RemoteStartTransaction

RemoteStartTransaction es una petición originada por el CSS o un sistema de gestión externo, por lo que en ella el CSS actúa en la parte "*cliente*". Este método permite indicar a un CP que debe iniciar una nueva transacción con el idTag indicado. Si el CP está disponible para iniciar

una transacción, lo indica en el status y envía posteriormente una petición StartTransaction al CSS.

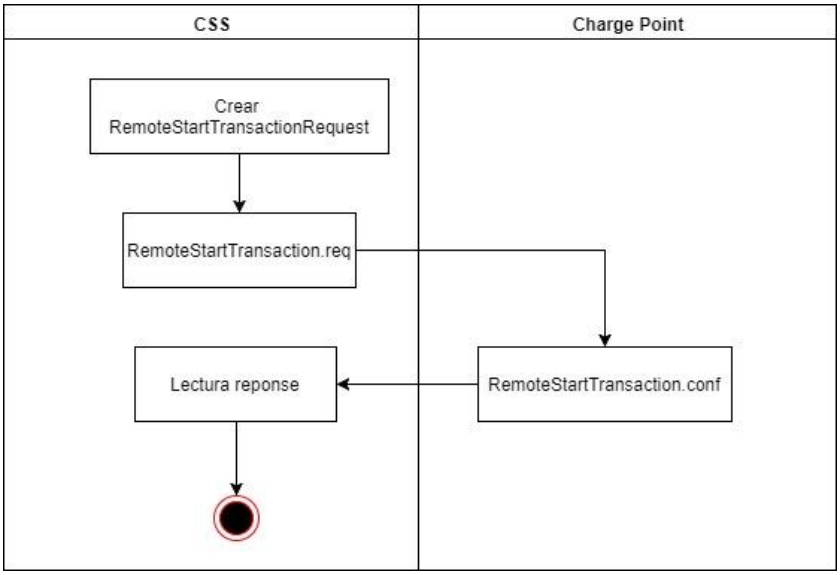


Ilustración 67 RemoteStartTransaction

Una vez enviada la petición al CP, si éste no ofrece una confirmación una vez alcanzado el time out, el CSS registra en la tabla de *log* la falta de mensaje por parte del CP y comunica lo mismo al sistema que originó la petición del web service *CommunicationsIN*.

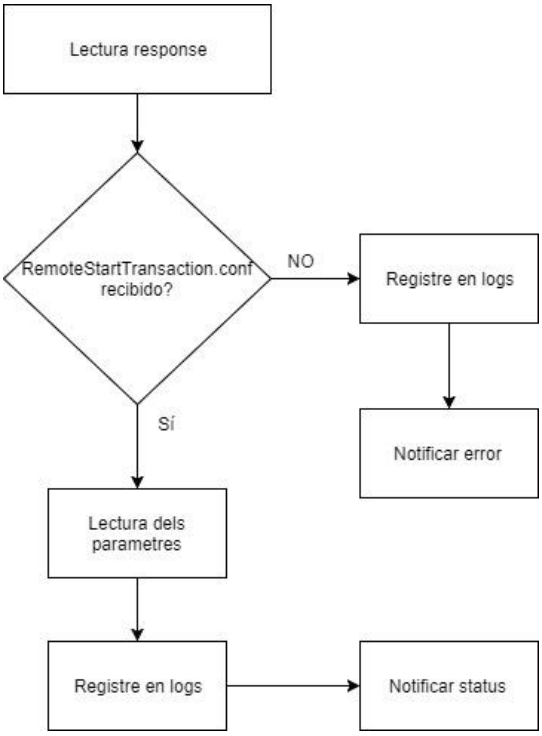


Ilustración 68 RemoteStartTransaction: Lectura de response

En caso de recibir una contestación, se registra la recepción en la tabla de *log* y se envía al sistema que originó la petición en formato de String el *status* indicado en el mensaje.

En último lugar, será explicado el método RemoteStopTransaction, mediante el cual, de manera similar a lo explicado en el presente método, se puede solicitar a un CP que finalice una transacción en curso.

6.17. RemoteStopTransaction

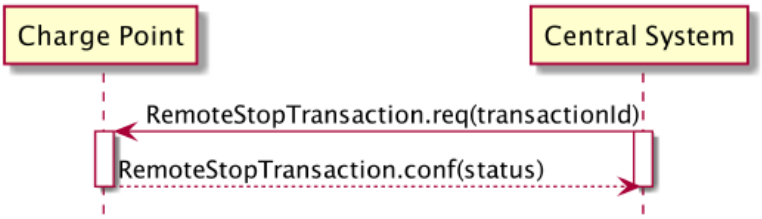


Ilustración 69 OCPP: RemoteStopTransaction

RemoteStopTransaction es un método iniciado por el CSS que permite finalizar una transacción en curso de un CP de manera remota, funcionalidad especialmente útil para tareas de mantenimiento. En este método la parte actuante del CSS es la parte "*cliente*", que se conecta al web service de los CP mediante la clase *ChargePoint_WSClient*.

De igual manera que en el método anterior, un sistema externo puede acceder al CSS mediante el web service de la clase *CommunicationsIN* y solicitar finalizar una transacción de un CP indicando el identificador de la misma y del CP en cuestión.

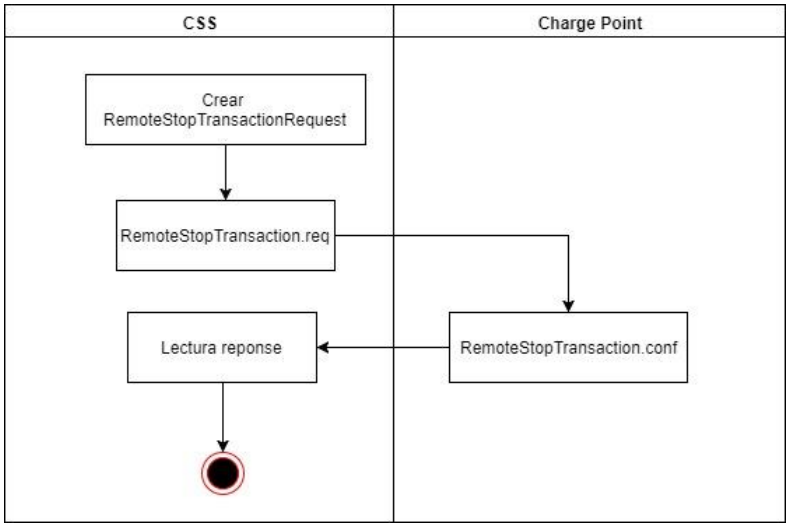


Ilustración 70 RemoteStopTransaction

El CSS crea un mensaje OCPP que es enviado al CP mediante la clase *ChargePoint_WSClient*. Si no se recibe contestación una vez superado el time out, el CSS registra el error en la tabla de *log* y lo comunica al sistema que originó la petición.

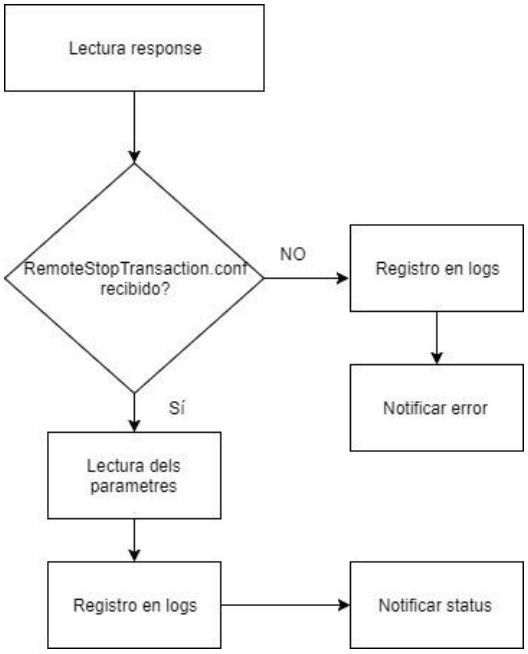


Ilustración 71 RemoteStopTransaction: Lectura de response

En caso de correcto funcionamiento, el CP indica en el mensaje de respuesta si es posible finalizar la transacción o no. El CSS registra ese mensaje en la tabla *log* y comunica el estado indicado en formato de String.

CAPÍTULO 7

PRUEBAS

Una vez que se ha explicado el desarrollo del proyecto y todos los métodos que en él se incluyen, se procederá a mostrar las pruebas realizadas para garantizar su correcto funcionamiento.

Para la realización de las pruebas del protocolo OCPP, se ha dispuesto de dos punto de carga, uno Circutor RVE-WB-MIX-Smart-Tri [15] y otro Ingeteam INGEREV CITY CW332 [16]. Estos modelos cuentan con dos tomas eléctricas cada uno, por lo que es necesario registrar dos conectores en la tabla *connector_list* de la BBDD para cada CP. Además, estos CP cuentan con un sistema lector de tarjetas RFID para poder realizar la autorización de usuarios mediante este tipo de tarjeta. En lo referente a las conexiones, ambos CP disponen de un puerto serie RS-485 y de conexión tanto Ethernet como 3G.

Ilustración 72 Circutor RVE-WB-MIX-Smart-Tri

Ilustración 73 Ingeteam INGEREV CITY CW332

Estos CP fueron instalados en el garaje subterráneo del bloque B del edificio Guillem Cifre de Colonya, en el campus de la UIB, para facilitar el acceso a ellos al realizar las tareas de testeo.

Durante las pruebas realizadas sobre estos CP, estos han sido conectado mediante Ethernet para permitir un gran tráfico de datos, ya que es necesario enviar una gran cantidad de mensajes para su correcto testeo.

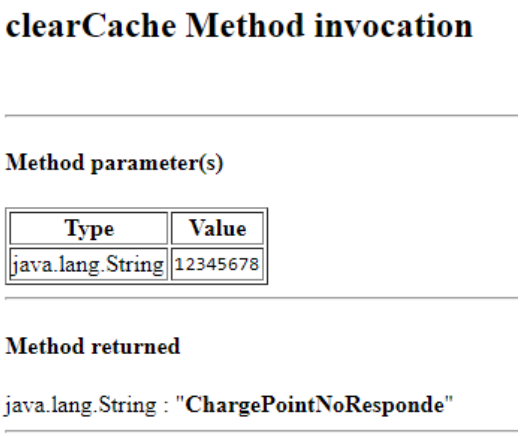
Una de las pruebas más importantes llevadas a cabo fue la comprobación del funcionamiento de Boot Notification, ya que con este método se establecieron las bases de desarrollo del sistema CSS, no únicamente de la parte "*servidor*" del mismo, sino también de la utilización del protocolo OCPP.

Durante el desarrollo del sistema, debido a la dificultad que suponía disponer permanentemente de un CP real para realizar pruebas y al retraso producido en la recepción de

los mismos, puesto que el CSS consiste principalmente en un sistema de web services, un conjunto de las fases de pruebas fue realizado mediante la herramienta de test de los mismos, alojando el sistema en un servidor de aplicaciones Glassfish.

Ilustración 74 CommunicationsIN Web Service Tester

En estas pruebas, se emularon los mensajes enviados por los CP al CSS, mas para el caso de los mensajes iniciados por este último, los que eran testeados a través del web service *CommunicationsIN*, únicamente fue posible comprobar el funcionamiento del código en caso de no recibir mensaje antes del time out o de indicar un CP no existente.



clearCache Method invocation

Method parameter(s)

Type	Value
java.lang.String	12345678

Method returned

java.lang.String : "**ChargePointNoResponde**"

Ilustración 75 Prueba de clerCache

emisor	receptor	parametros	hora ▾ 1	metodo
null	CSS	URI null, ChargePointIDDesconocido	2017-07-13 09:51:37	heartBeat
CSS	192.168.1.7		2017-07-13 09:39:43	clearCache
192.168.1.7	CSS	ChargePointNoResponde	2017-07-13 09:39:43	clearCache

Ilustración 76 Registro tras pruebas de CP incomunicado/desconocido

Este tipo de pruebas también fue especialmente útil para comprobar, al inicio del proyecto, que la lectura de las cabeceras y la estructura de los mensajes se realizaba correctamente, sin necesitar disponer de un punto de carga físico para ello. Este punto resultó especialmente conflictivo, ya que las cabeceras del mensaje SOAP contenían atributos con la etiqueta *MustUnderstand*, por lo que era necesario pre-procesarlos para confirmar al CP que se ha establecido comunicación y después volver a leerlos para obtener los valores de *ChargeBoxIdentity* y *URI* del CP.

CAPÍTULO 8

CONCLUSIONES

Para finalizar este documento, se ha considerado oportuno analizar las líneas futuras que del proyecto **Sistema central de gestión de puntos de carga** surgen, algunas de las cuales han sido introducidas a lo largo de todo el documento.

8.1. Líneas de futuro

El primer derivado de este proyecto, y probablemente el más importante, es el desarrollo de la clase *CommunicationsOUT* que servirá como interfaz de comunicación entre el CSS y sistemas de gestión externos. Esta clase podría incluir llamadas a web services externos, comunicando los mensajes en el momento de ser recibidos, o ser un web service en el que los sistemas externos realicen consultas.

Al realizar esta integración, también será necesario diseñar una interfaz diferente en la clase *Authorizations*, al dejar de utilizar una base de datos propia y requerir de la comunicación con un sistema externo.

Otro posible proyecto consistiría en la actualización del CSS diseñado en el proyecto **Sistema central de gestión de puntos de carga** a una versión posterior de OCPP, preferiblemente la 2.0, que propone realizar una gran cantidad de cambios, dotando al estándar de mayor robustez, siendo así los mensajes independientes del fabricante, y mayor eficiencia en el intercambio de mensajes, al incluir tecnologías como JSON. No obstante, la realización de dicha actualización vendrá marcada por la evolución de los fabricantes de puntos de carga, ya que el CSS debe estar diseñado en la versión que ellos decidan implementar en sus CP.

Relacionado con la integración explicada previamente, es posible que en un futuro la base de datos *basedatos*ocpp desaparezca o deje de ser de naturaleza interna, siendo necesario modificar la parte "*gestión de las* BBDD" del CSS para acceder al sistema que incluya los datos.

Además de los mencionados, es posible desarrollar una gran variedad de proyectos relacionados con el proyecto **Sistema central de gestión de puntos de carga** debido a las diversas posibilidades que ofrece gestionar una red de puntos de carga de manera unificada y uniforme, permitiendo mejorar el sistema con aplicaciones de la capa de negocio de IoT para gestionar que las cargas se realicen de manera sostenible.

8.2. Conclusión

A la finalización del proyecto **Sistema central de gestión de puntos de carga**, se han asumido todos los requisitos funcionales establecido en el capítulo 4 del presente documento, permitiendo a los puntos de carga comunicarse con el CSS para desempeñar todas las funciones

necesarias para la carga de un vehículo eléctrico y la gestión del propio CP, siendo las funciones más importantes StartTransaction, StopTransaction y Heartbeat, ya que con ellas es posible realizar transacciones y conocer el estado de los puntos de carga.

Respecto al requisito no funcional **RNF01**, se puede tomar como alcanzado debido a que las clases de *Communications* permiten integrar el proyecto **Sistema central de gestión de puntos de carga** junto con otros sistemas externos de gestión realizando únicamente un pequeño desarrollo que no afecta a la implementación del CSS.

El desarrollo de este proyecto pone de manifiesto la necesidad de disponer de un estándar robusto para la comunicación entre CP y CSS, cosa que OCPP no ofrece en su versión 1.5 debido a la gran cantidad de parámetros de carácter opcional que se incluyen en sus mensajes. No obstante, a pesar de que hasta la versión 2.0 OCPP no supondrá un estándar lo suficientemente robusto y la versión 1.5 requiere mayor tráfico de datos al no disponer de compatibilidad con mensajes JSON, su existencia permite desarrollar una gran variedad de aplicaciones como la desarrollada en este proyecto, facilitando la comunicación y la gestión de grandes redes de carga de vehículos eléctricos, por lo que OCPP es una clara apuesta de futuro en la que este proyecto cree firmemente.

BIBLIOGRAFÍA

[1] Statista.com, ”Number of vehicles in use worldwide 2006-2014”, 2017. [Online]. Available: https://www.statista.com/statistics/281134/number-of-vehicles-in-use-worldwide/ 1

[2] J. Sousanis, “World Vehicle Population Tops 1 Billion Units”, WardsAuto, august 2011. [Online]. Available: http://wardsauto.com/news-analysis/world-vehicle-population-tops-1-billion-units 1

[3] DGT, “Los coches que menos contaminan“, august 2007. [Online]. Available: http://www.dgt.es/revista/archivo/pdf/num185-2007-contaminacion.pdf 1

[4] Schenider Electric, “Villes Dourables”. [Online]. Available: http://www.schneider-electric.fr/fr/work/campaign/climate-change/cities.jsp 1

[5] UIB, “SmartUIB: Misión y Visión”. [Online]. Available: http://smart.uib.es/Visio-i-missio/ 1

[6] Wikipedia, “Sistema determinista”. [Online]. Available: https://es.wikipedia.org/wiki/Sistema_determinista 2.1

[7] G. Banda, K. Chaitanya, H. Mohan, “An IoT Protocol and Framework for OEMs to Make IoT-Enabled Devices Forward Compatible”, IEEE xplore, November 2015. [Online]. Available: http://ieeexplore.ieee.org/xpls/icp.jsp?arnumber=7400658 2.1

[8] M. Aazam, P. Hung, E. Huh, “Smart gateway based communication for cloud of things”, IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), april 2014. [Online]. Available: https://www.researchgate.net/publication/269303095_Smart_gateway_based_communication_for_cloud_of_things 2.1

[9] Sentilo, “What Is”. [Online]. Available: http://www.sentilo.io/xwiki/bin/view/Sentilo.About.Product/Whatis 2.3

[10] Open Charge Alliance, “Open Charge Alliance: Home page.” [Online]. Available: http://www.openchargealliance.org/ 2.4

[11] Etecnic “OCPP 2.0. Un indicio a la comunicación del futuro”, july 2015. [Online]. Available: http://www.etecnic.es/noticias/ocpp-2-0-un-indicio-a-la-comunicacion-del-futuro/ 2.4

[12] Innova Software Developers, “iGSEGeS software de tecnología avanzada para la gestión eficiente de toda la actividad del Gestor de Carga”. [Online]. Available: http://www.innova-soft.com/igseges.html 2.5.1

[13] Etra, “NOC. Sistema de Control de postes de recarga de vehículos eléctricos de la ciudad de Barcelona” [Online]. Available: http://www.etra.es/casos-de-exito/noc-sistema-de-control-de-postes-de-recarga-de-vehiculos-electricos-de-la-ciudad-de-barcelona.aspx 2.5.2

[14] Mauricio Chubasco, “Metodologías de Desarrollo de proyectos”, Apuntes DUOC. [Online]. Available: http://apuntesduoc.pbworks.com/w/page/49020559/Metodolog%C3%ADas%20de%20Desarrollo%20de%20proyectos 5.1

[15] Circutor, “Circutor RVE-WB-MIX-Smart-Tri ficha técnica”. [Online]. Available: http://www.ingeteam.com/Portals/0/Catalogo/Producto/Documento/PRD_1055_Archivo_ingerev-city-es.pdf 7

[16] Ingeteam, “Ingeteam INGEREV CITY CW332 ficha técnica”. [Online]. Available: http://circutor.es/docs/FT_RVE-WB_SP.pdf 7