Trabajo Fin de Grado

Grado en Ingeniería de las Tecnologías de Telecomunicación

Sistema distribuido para gestión de infraestructuras de recarga con OCPP y SOAP

Autor: Carlos Rodríguez Reyes

Tutores: Manuel Ruiz Arahal, José María Maestre Torreblanca

Dep. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2014

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Sistema distribuido para gestión de infraestructuras de recarga con OCPP y SOAP

Autor:
Carlos Rodríguez Reyes

Tutores:
Manuel Ruiz Arahal
José María Maestre Torreblanca

Dep. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2014

Trabajo Fin de Grado: Sistema distribuido para gestión de infraestructuras de recarga con OCPP y SOAP

Autor: Carlos Rodríguez Reyes

Tutores: Manuel Ruiz Arahal, José María Maestre Torreblanca

El tribunal nombrado para juzgar el Trabajo arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2014

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Quisiera mostrar mis más profundos agradecimientos a mis tutores José María Maestre Torreblanca y Manuel Ruiz Arahal, por brindarme la posibilidad de trabajar en este proyecto y por la ayuda ofrecida a lo largo del desarrollo del mismo.

También me gustaría mostrar mis agradecimientos a Daniel Gutiérrez y Javier Carmona, por la implicación en el proyecto y el gran trabajo en equipo desempeñado.

A su vez, mil gracias a mi compañero Eugenio Pérez, por la ayuda prestada durante todo el desarrollo del proyecto, así como a mis compañeros de Eneo Tecnología, por darme el apoyo y la flexibilidad necesaria para la realización del mismo.

Finalmente, solo me queda agradecer a mi familia y amigos por todo el apoyo recibido no sólo durante la elaboración de este proyecto, si no durante toda mi vida.

Resumen

El presente trabajo fin de grado presenta una solución para la gestión automática de infraestructuras de recarga para vehículos eléctricos.

Con este fin, se han desarrollado cuatro aplicaciones:

- Un poste de recarga, donde se podría conectar un único vehículo eléctrico para ser cargado.
- Un controlador de infraestructura, que gestiona los puntos de recarga
- Un sistema de información externo, que organiza las cargas comunicándose con el controlador
- Un entorno de usuario, que permite que un operario maneje la infraestructura.

El poste de recarga incorpora una serie de comunicaciones con el dispositivo electrónico que le permite leer y escribir salidas digitales.

Para la comunicación de estas aplicaciones se han desarrollado servicios web basados en SOAP. Además, se utiliza el protocolo OCPP (Open Charge Point Protocol, también basado en SOAP) para comunicar los postes de recarga con el controlador de infraestructura.

Las aplicaciones han sido programadas en el lenguaje C, utilizando las librerías gSOAP para la implementación de los servicios web SOAP y la librería SMC para la implementación de máquinas de estados.

Este documento incluye la descripción de esta solución, una guía de uso, una descripción de los protocolos y herramientas utilizadas y finalmente las conclusiones obtenidas de la realización del mismo.

En los anexos extiendo la información sobre las máquinas de estado que se han implementado, así como una panorámica del código desarrollado en cada una de las cuatro aplicaciones desarrolladas.

Índice

Índice de tablas y figuras

Notación

IEA	International Energy Agency
EVI	Electric Vehicle Initiative
OCA	Open Charge Alliance
OCPP	Open Charge Point Protocol
CIR.	Controlador de infraestructura de recarga
PR	Poste de recarga
SI	Sistema de información
EU	Entorno de usuario
SM	State Machine (Máquina de estados)
ME	Máquina de estados
SOAP	Simple Object Access Protocol
SMC	State Machine Compiler
EV	Electric Vehicle
VE	Vehículo eléctrico

1 INTRODUCCIÓN

Electric cars are going to be very important for urban transportation.

- Carlos Ghosn (CEO Nissan Motor) -

El mundo de los vehículos eléctricos es uno de los que más cambios está experimentando en los últimos años. A pesar de que el vehículo eléctrico fue uno de los primeros automóviles en desarrollarse (incluso antes que los motores Diésel), los vehículos de combustión han sido los principales vehículos de pasajeros en el pasado, debido principalmente a que, hasta ahora, los vehículos eléctricos no poseían suficiente autonomía, y solo podían mantener velocidades bajas.

Sin embargo, en el presente estamos viviendo una revolución en este campo. Los vehículos eléctricos cada vez son más adoptados. A día de hoy es fácil ver circulando vehículos híbridos (como el Toyota Prius o el Chevrolet Volt), o vehículos puramente eléctricos (como el Tesla Model S). Según los estudios realizados por la IEA, esta tendencia continuará creciendo en el futuro.

1.1 Presente y futuro de los vehículos eléctricos

En Abril del 2013, la IEA (International Energy Agency) publicó un estudio titulado "Global EV Outlook: Understanding the Electric Vehicle Landscape to 2020" [1] donde analiza el mercado de los vehículos eléctricos hasta el año 2012 en los 15 países que en ese momento forman parte de la EVI (Electric Vehicle Initiative).

La EVI [2] es una iniciativa formada por Estados Unidos, Reino Unido, Francia, España, Portugal, Dinamarca, Países Bajos, Suecia, Finlandia, Alemania, Italia, Sudáfrica, China, India y Japón destinada a impulsar el uso de los vehículos eléctricos en todo el mundo. Posteriormente, en 2013 y 2014, Canadá y Noruega se unen a la EVI dejando un total de 17 miembros en la iniciativa.

En este estudio, la IEA muestra que a finales de 2012 se encontraban en circulación aproximadamente 180.000 vehículos eléctricos, lo que suponía un 0,02% del total de vehículos mundiales.

El objetivo de la EVI es alcanzar la cifra de 20 millones de vehículos eléctricos para el año 2020, lo que supondría un 2% del total de vehículos de pasajeros.

Respecto a las ventas, el informe muestra un aumento de un 151% entre los años 2011 y 2012.

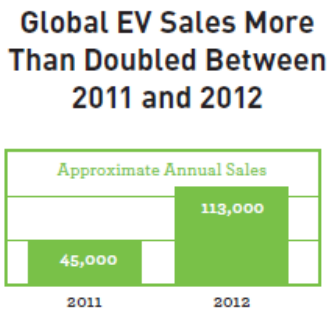


Ilustración 1: Ventas de vehículos eléctricos en 2011/2012

Como se puede observar en las siguientes figuras, la adopción de vehículos eléctricos ha crecido exponencialmente desde el año 2010, y la

Ilustración 2: Objeticos de ventas y existencias de vehículos eléctricos hasta 2020

EVI espera que esa tendencia continúe al menos hasta el año 2020.

Figure 2. EV Sales Targets [select EVI members]
Source: EVI. Note: A 20% compound annual growth rate is assumed for countries without a specific sales target (i.e., only a stock target) or with targets that end before 2020.

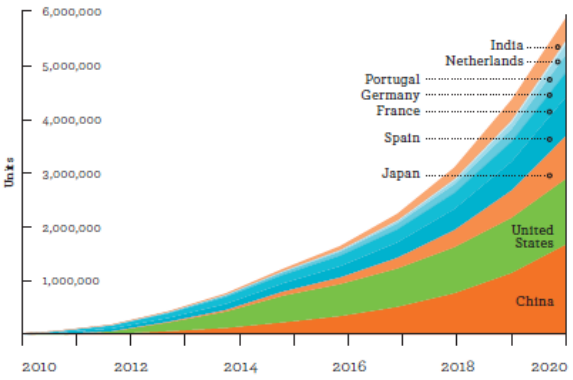
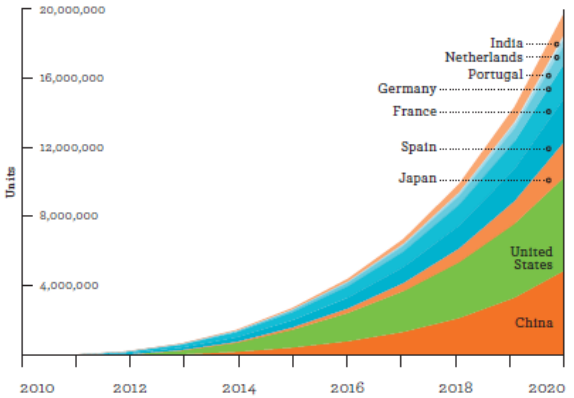


Figure 3. EV Stock Targets [select EVI members]
Source: EVI. Note: A 20% compound annual growth rate is assumed for countries without a specific stock target (i.e., only a sales target) or with targets that end before 2020.



Para apoyar el desarrollo y a la acogida del vehículo eléctrico, es importante que los ciudadanos y las empresas dispongan de la infraestructura necesaria para abastecer eléctricamente a sus vehículos. Esto implica un despliegue de diversos puntos de recarga a lo largo de toda la geografía.

Hay varios tipos de puntos de recarga. Los puntos de recarga públicos se podrían encontrar en las llamadas Electrolinera, Electrinera o Estaciones de carga, que son el equivalente eléctrico a las actuales gasolineras para vehículos de combustión. Por otro lado, también existen puntos de recarga particulares, de forma que el usuario de un vehículo eléctrico pueda cargar el vehículo en su propia casa.

En sitios web como 'Plugshare' [3] es posible encontrar los puntos de recarga para vehículos eléctricos disponibles en cada zona.

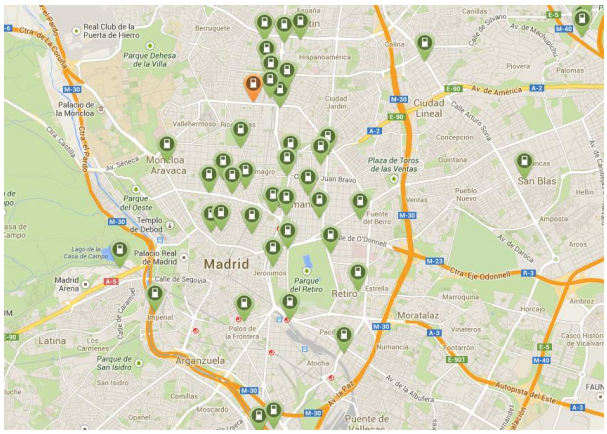


Ilustración 3: Puntos de recarga disponibles en Madrid y punto de recarga rápida de Tesla Motors.

1.2 Open Charge Alliance y OCPP

Para facilitar el desarrollo de soluciones de infraestructuras de recarga para vehículos eléctricos, se crea en el año 2009 la Open Charge Alliance [4], una unión de empresas dedicada a la impulsión de protocolos de comunicación abiertos para permitir la creación de dispositivos de infraestructura de recarga que permitan intercomunicación entre ellos, aunque no pertenezcan al mismo fabricante. Con este objetivo en mente, aparece OCPP.

OCPP (Open Charge Point Protocol) [5] es un protocolo abierto que permite que varios puntos de carga sean gestionados y controlados por un dispositivo centralizado. El protocolo es completamente abierto, y su especificación completa puede encontrarse en el sitio web de OCA [6].

Sin embargo, OCPP no está completamente adaptado para el caso de flotas de vehículos eléctricos. En estos casos, tenemos una flota de vehículos conectada de forma continua a los puntos de recarga, y queremos que

estos vehículos se recarguen de forma automática cuando ocurren ciertas condiciones, teniendo en cuenta que la flota puede estar distribuida por varios lugares geográficos.

1.3 Objetivos del trabajo

El objetivo de este trabajo es el diseño de métodos de gestión automática de una infraestructura de recarga para el abastecimiento eléctrico de flotas de vehículos eléctricos. En un escenario como este, una entidad que dispone de una flota de vehículos en varios lugares geográficos desea centralizar y controlar todas las cargas de sus vehículos. De esta manera, los vehículos se dispondrían en uno o varios puntos geográficos, y la empresa gestionaría cuándo y cómo se realizan las cargas de éstos vehículos.

Esta infraestructura está compuesta por varios elementos:

- Una serie de puntos de recarga, donde se podría conectar un único vehículo eléctrico.
- Un sistema centralizado, que gestiona los puntos de recarga
- Un sistema de información externo, que organiza las cargas comunicándose con el sistema centralizado
- Un entorno de usuario, que permite que un operario maneje la infraestructura.

Para la realización del sistema, se va a utilizar el lenguaje C para crear aplicaciones que funcionarán en sistema embebidos, ubicados tanto en los puntos de carga como en el sistema central. Además, se utilizará el protocolo OCPP para comunicar los puntos de recarga con el sistema centralizado, así como una serie de protocolos basados en SOAP para comunicar el sistema centralizado con el sistema de información externo y el entorno de usuario.

1.4 Estructura del documento

El capítulo 2 presenta los diferentes elementos que forman parte del sistema. En él se explica la función de cada elemento, cómo funciona y de qué manera se comunica con el resto de elementos.

El capítulo 3 es una guía de uso del sistema. En ella puede encontrar información sobre cómo configurar los distintos elementos, cómo iniciarlos y pararlos, y como operar sobre los puntos de recarga.

El capítulo 4 describe los protocolos que comunican lo diferentes elementos. En éste puede encontrar información sobre qué mensajes se comparten y qué tipo de información se transmite en cada mensaje.

El capítulo 5 es una explicación sobre las herramientas externas que he utilizado para construir el sistema. Concretamente, puede encontrar información sobre cómo se instala y cómo se usa gSOAP, la librería para crear servicios web SOAP, y SMC, la librería para implementar máquinas de estados.

Finalmente, en el capítulo 6 se encuentran las conclusiones del trabajo y las vías de futuro sobre el sistema.

2 MEMORIA DESCRIPTIVA

El siguiente diagrama muestra los elementos que intervienen en la comunicación y en la operación de la infraestructura de recarga, los protocolos de aplicación utilizados para la comunicación entre ellos y el ámbito de las redes en las que se encuentran.

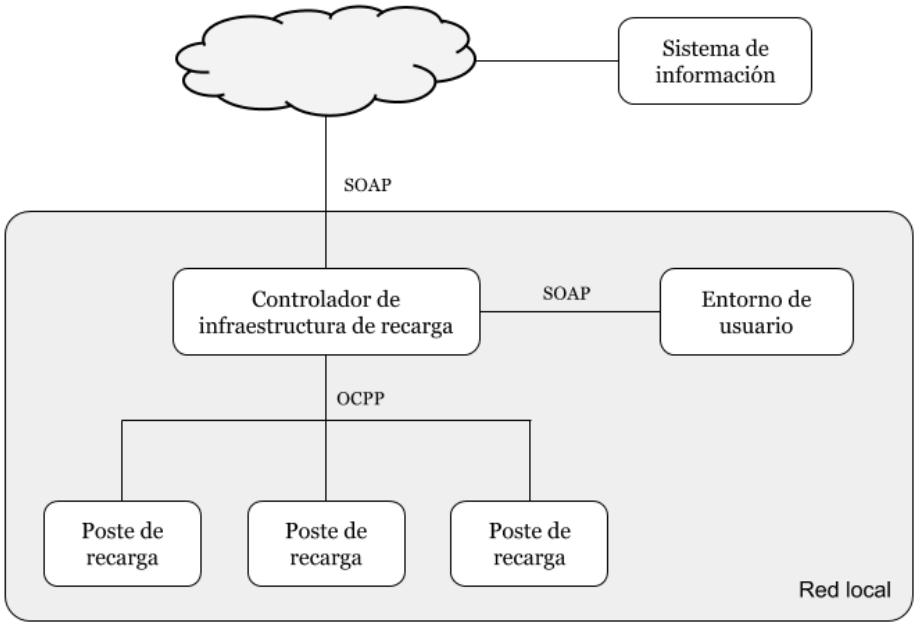


Ilustración 4: Diagrama de elementos

El **poste de recarga** es un dispositivo físico donde el vehículo eléctrico debe conectarse para realizar una recarga. Estos postes se comunican con el **controlador de infraestructura de recarga**, que es el encargado de ordenar distintas operaciones sobre ellos, denominadas "maniobras". Una operación de recarga de un vehículo eléctrico es un ejemplo de maniobra.

Para que el controlador realice una maniobra, primero es necesario programarla. El **sistema de información** es el elemento encargado de programar maniobras en el controlador de infraestructura.

El **entorno de usuario** se podría identificar con el terminal que lleva el operario de la infraestructura de recarga.

Es importante destacar que todos los elementos se encuentran en una red local, excepto el sistema de información, que es un elemento externo que accede al controlador a través de internet.

A nivel jerárquico, los postes de recarga son subordinados del controlador de infraestructura, que a su vez es subordinado del sistema de información, mientras que el entorno de usuario exclusivamente recoge información y es incapaz de ordenar o realizar acciones sobre el sistema.

En un escenario final, habría un único sistema de información que se comunicaría a través de Internet con varios controladores de infraestructura desplegados por diferentes puntos geográficos. Estos controladores a su vez se comunican mediante una red local con los postes de recarga situados en ese punto.

También es importante identificar a los actores del sistema:

1. **Operario de la infraestructura de recarga:** Es el encargado de comprobar el estado de las maniobras que se están ejecutando, ver las maniobras pendientes de ejecución, etc. El operario tiene acceso físico al controlador y a los postes. Sin embargo, su principal herramienta para utilizar el sistema es el entorno de usuario.
2. **Usuario de la infraestructura de recarga:** El usuario es la persona que utiliza la infraestructura para recargar su vehículo eléctrico. El usuario tiene acceso físico a los postes de recarga, donde conecta el vehículo eléctrico. Sin embargo, es incapaz de realizar maniobras (como operaciones de carga) por sí mismos.

2.1 Controlador de la infraestructura de recarga (CIR)

Es el encargado de dirigir los postes de recarga mediante el protocolo OCPP. Además, el CIR se comunica con el entorno de usuario y el sistema de información, permitiendo obtener datos de su funcionamiento y realizar acciones sobre el sistema.

Físicamente, el CIR no es más que un servidor que ejecuta una aplicación que permanece a la espera de recibir mensajes SOAP, ya sean originados por los PR, por el SI o por el EU.

```
[INFO] Press 'R' key to execute a remote charge operation on every client
[INFO] Press 'p' key to display the waiting manoeuvres list
[INFO] Press 'x' key to display the executing manoeuvres list
[INFO] Press 'e' key to display the executed manoeuvres list
[INFO] Started SI server on localhost:8081
[INFO] Started EU server on localhost:8082
[INFO] Started PR server on localhost:8080 with 3 threads
<-- BootNotification Request from 127.0.0.1:48087 with ID 111.
<-- Heartbeat Request from PR ID 111 at 127.0.0.1:48088.
[NEW MANOUEVER] NEW MANOUEVER ID 1 ON PR ID 111 ]
[INFO] Executing manoeuvre -- ID: 1 -- ID PR: 111
[INFO] Charge Maniouvre on PR ID 111
[STATE PR 111] SPR
--> RemoteStartTransaction Request to PR 111 at http://127.0.0.1:8083
<-- RemoteStartTransaction Response from PR http://127.0.0.1:8083.
[STATE PR 111] EFC
<-- Heartbeat Request from PR ID 111 at 127.0.0.1:48091.
```

Ilustración 5: Programa del CIR

2.1.1 Módulos

A nivel organizativo, el controlador está dividido en varios módulos, que se encargan de realizar diferentes tareas.

1. Gestor de cola de maniobras

   Encargado de controlar y generar los registros de las maniobras pendientes y en ejecución. Este módulo debe ejecutar las maniobras cuando llegue el instante adecuado, además de comprobar el estado de finalización de la maniobra para dejar constancia en los registros.

2. Gestor de postes de recarga

Este módulo se encarga de llevar constancia de los puntos de recarga que hay conectados y registrados en cada momento. Es el encargado de comprobar que todos los postes están operativos realizando el proceso de Heartbeat.

3. Gestor de mensajes de los PR

   Se encarga de recibir y procesar los mensajes OCPP que provienen de los postes de recarga. La mayoría de los mensajes intercambiados por este módulo son producidos por maniobras que previamente han sido encoladas y ejecutadas por el gestor de cola de maniobras.

4. Gestor de mensajes SI

   Recibe y procesa los mensajes que provienen del SI. Estos mensajes incluyen tareas como encolamiento de maniobras, recogida de estadísticas, recogida de diagnósticos de los postes de recarga, actualización de firmware, etc.

5. Gestor de mensajes EU

   Es el módulo encargado de recibir y procesar mensajes que provienen del EU. Estos mensajes incluyen tareas como obtener una lista de maniobras pendientes, ejecutadas o en ejecución, obtener un histórico de los registros, hacer peticiones de maniobras al SI, etc.

2.1.2 Gestión de postes de recarga

Una de las responsabilidades del CIR es llevar un registro de los postes de recarga disponibles en cada momento.

Para implementar esta funcionalidad, se realiza lo que se conoce como mecanismo de 'Heartbeat' (en inglés, latido de corazón). Este mecanismo está definido en la especificación del protocolo OCPP, y permite que el controlador de infraestructura tenga plena constancia en todo momento de qué postes de recarga están funcionando.

Para lograr esto, los postes de recarga envían un mensaje denominado 'BootNotification' (notificación de encendido) al controlador de infraestructura cuando los postes son iniciados. Al recibirlo, el CIR marca al PR como disponible.

Mientras un PR esté en estado disponible, el CIR contará el tiempo que transcurre entre cada mensaje recibido por ese PR. Si este tiempo supera en algún momento una cantidad de segundos configurable, ese PR se marca como no disponible, y no vuelve a utilizarse hasta que vuelve a enviar un nuevo mensaje de notificación de encendido.

2.1.3 Máquinas de estados

El CIR mantiene información actualizada en todo momento sobre el estado de cada uno de los PRs disponibles en el CIR. Para ello, el CIR implementa una máquina de estados por cada uno de los PRs disponibles.

Respecto a las máquinas de estados del CIR, es importante tener en cuenta lo siguiente:

1. Existen una máquina de estado por cada PR disponible en el CIR, y cada una de estas es completamente independiente de las demás.
2. Un cambio en una máquina de estados del CIR debe producir un cambio en el PR asociado a esa máquina y viceversa. Para ello se utiliza el protocolo OCPP, que comunica al CIR con los PR.

En la ilustración 7 pueden observarse los posibles estados de la máquina de estados, así como las transiciones entre ellos.

2.1.4 Gestión de maniobras

Las maniobras son las ordenes que el sistema de información realiza en los postes de recarga.

Algunos ejemplos de maniobras son: maniobra de carga de vehículo, maniobra de pausa o aborto de una carga, maniobra de actualización de un PR, etc.

Las maniobras contienen información como: el tipo de maniobra a realizar, el PR en el que debe ejecutarse la maniobra, el instante en el que debe ejecutarse la maniobra, y otros parámetros opcionales.

El CIR almacena tres listas diferentes de maniobra: la lista de maniobras pendientes de ejecución, la lista de maniobras en ejecución, y la lista de maniobras ejecutadas.

Cuando el SI ordena una nueva maniobra, el CIR agrega la maniobra a la lista de maniobras pendientes de ejecución. Cuando llega el instante de ejecución de esa maniobra, el CIR procede a su ejecución y mueve la maniobra a la lista de maniobras en ejecución. Cuando la maniobra finaliza, finalmente el gestor de colas de maniobra mueve la maniobra a la lista de maniobras ejecutadas.

Tenga en cuenta que, puesto que todas las maniobras afectan a un determinado PR, la ejecución de una maniobra implica un cambio en la máquina de estados del CIR asociado a ese PR, y eso implica una comunicación entre el CIR y el PR afectado.

En la ilustración 6, puede encontrar un diagrama de acciones que describe el procedimiento que se sigue y las acciones que se realizan en los distintos elementos del sistema cuando una maniobra es programada en el CIR.

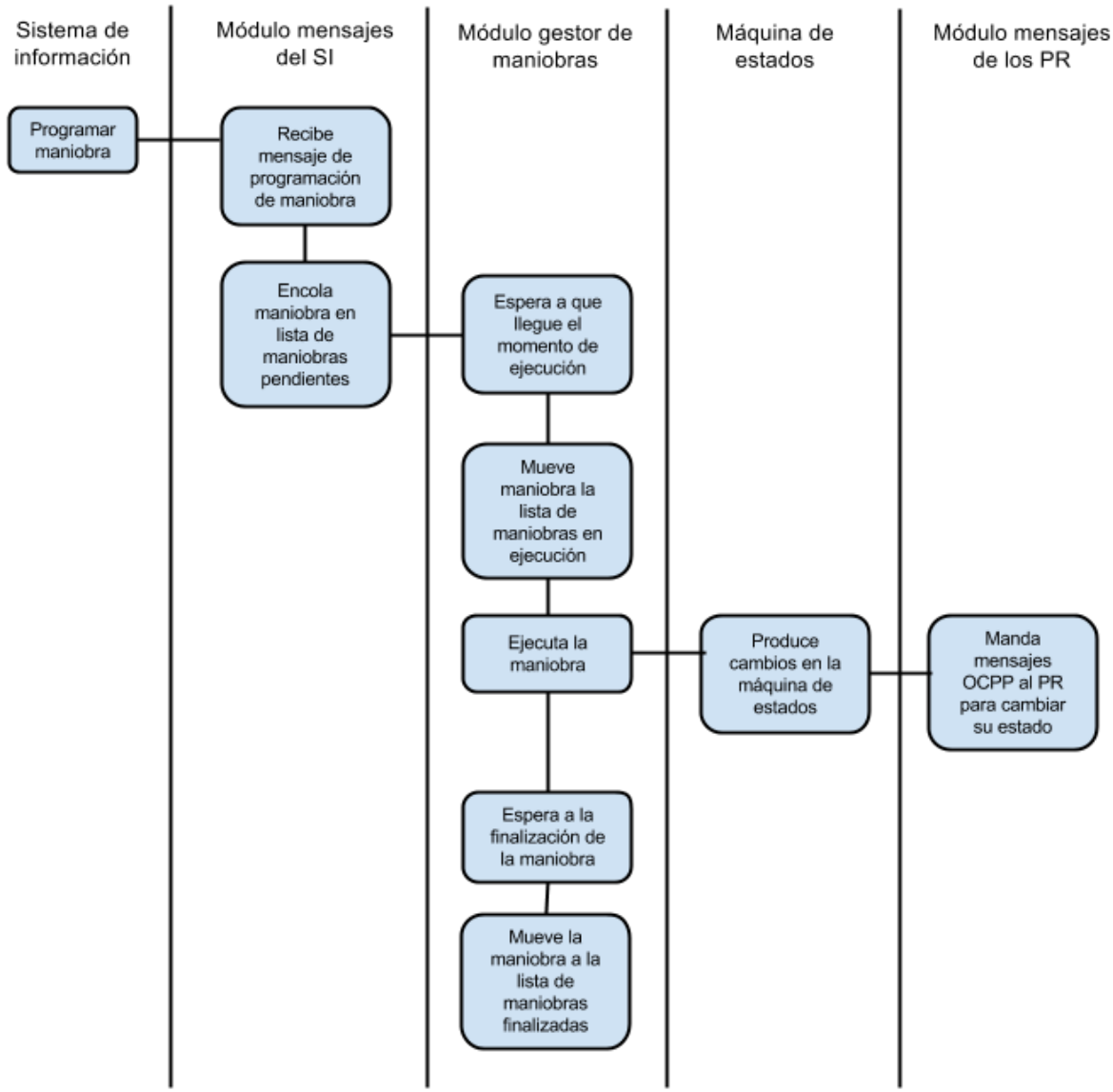


Ilustración 6: Diagrama de acciones al programar una maniobra

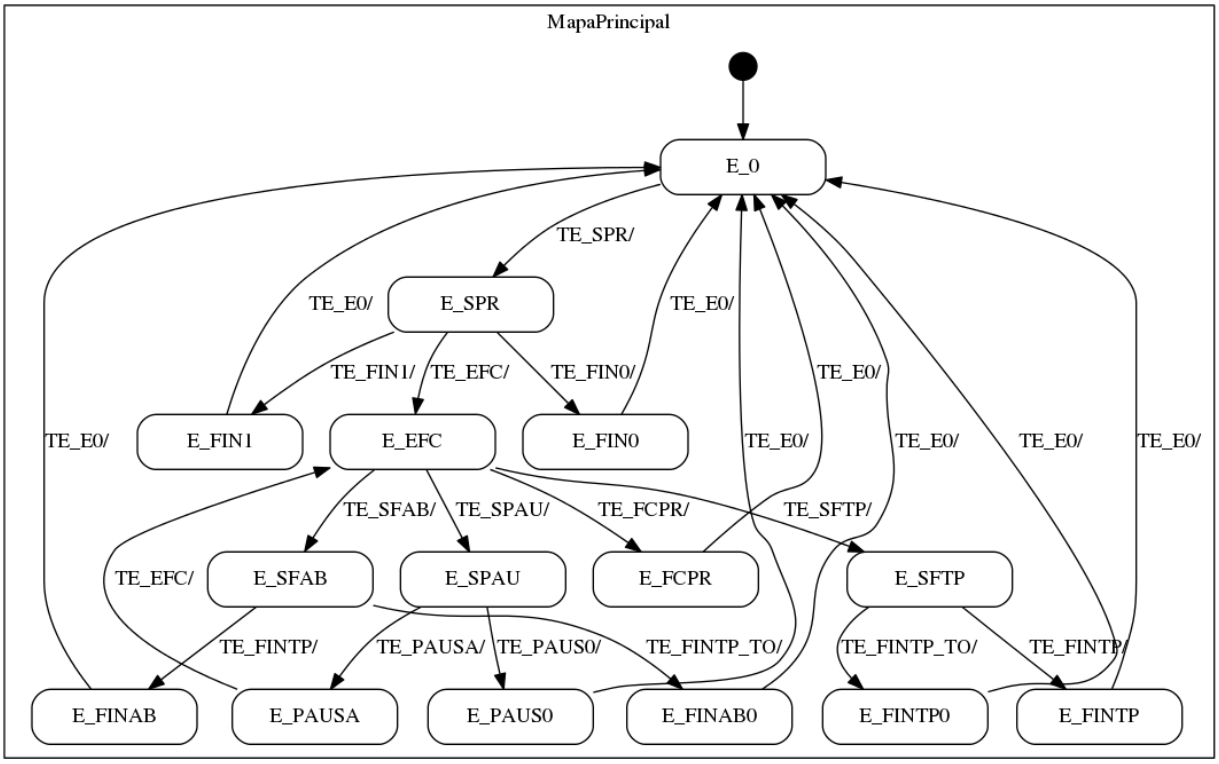


Ilustración 7: Máquina de estados del CIR

Ilustración 8: Poste de recarga físico

2.2 Poste de recarga (PR)

Los postes de recarga son los elementos finales de la infraestructura. Son los dispositivos que el usuario utiliza para conectar el vehículo eléctrico. Sin embargo, estos dispositivos son incapaces de comenzar operaciones por ellos mismos.

Para que el PR realice acciones, es necesario que intercambie mensajes OCPP con el CIR. La comunicación con el CIR puede ser en ambos sentidos. El protocolo OCPP y los mensajes que pueden intercambiarse (originados en el CIR, o en el PR) serán detallados en apartados posteriores.

```
[INFO] Server started on port 8083
--> BootNotification Request to CIR at http://localhost:8080 calling with PR ID 111
<-- BootNotification Response from CIR. Heartbeat Interval: 5 seconds
--> Heartbeat Request to CIR
C
[DBG] Cable conectado
H
[DBG] CP Cesa entrega de potencia
[WARN] Received non permitted transition
--> Heartbeat Request to CIR
<-- RemoteStartTransaction Request from CIR
[NEW STATE] EFC
--> Heartbeat Request to CIR
```

Ilustración 9: Programa del PR

2.2.1 Máquina de estados

El PR, de la misma forma que el CIR, implementa una máquina de estados que describe el estado actual del PR. La máquina de estados y la del CIR se sincronizan mediante mensajes OCPP, de forma que una de las máquinas no avanza sin que la otra no avance de manera paralela.

De esta forma, es posible saber en cada momento en qué estado se encuentra el PR, tanto en el CIR como en el propio PR. Sin embargo, los estados no son iguales y no hay una equivalencia completa entre una máquina de estados y la otra.

El siguiente esquema describe los estados y las transiciones presentes en la máquina de estados del PR.

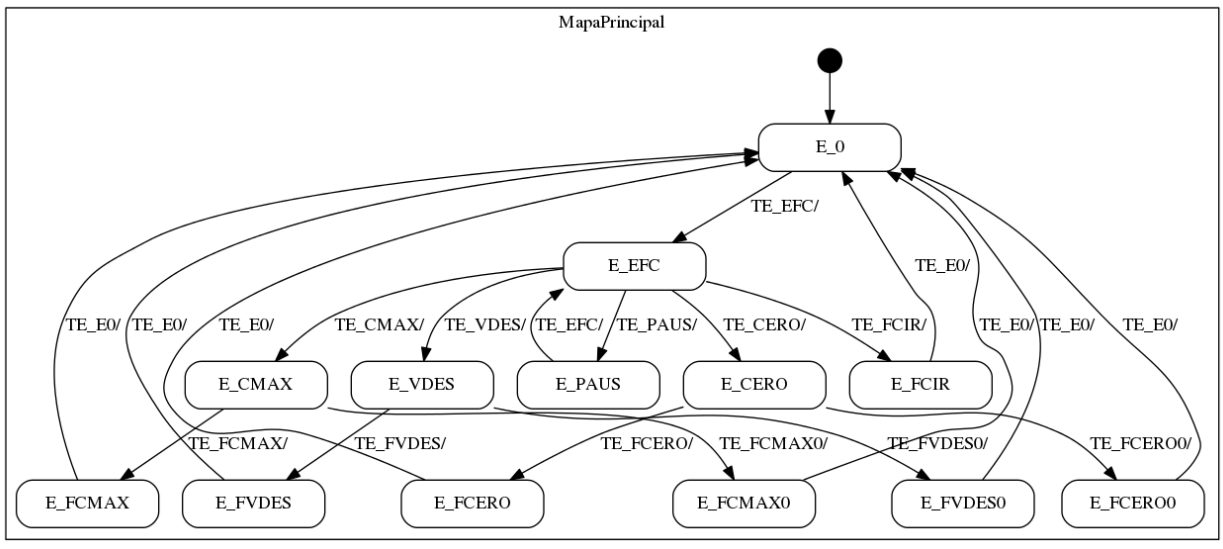


Ilustración 10: Máquina de estados del PR

2.2.2 Elementos físicos y electrónica

Físicamente, el poste es un dispositivo que incorpora varios elementos dentro de una estructura de plástico que el usuario puede manipular. En su interior se encuentra: un rectificador, un diferencial, un termomagnético, un circuito impreso que aporta una interfaz para comunicarnos con las entradas y salidas de la estructura y por último un PC-104, que no es más que un estándar de ordenador embebido.

Ilustración 11: PC-104

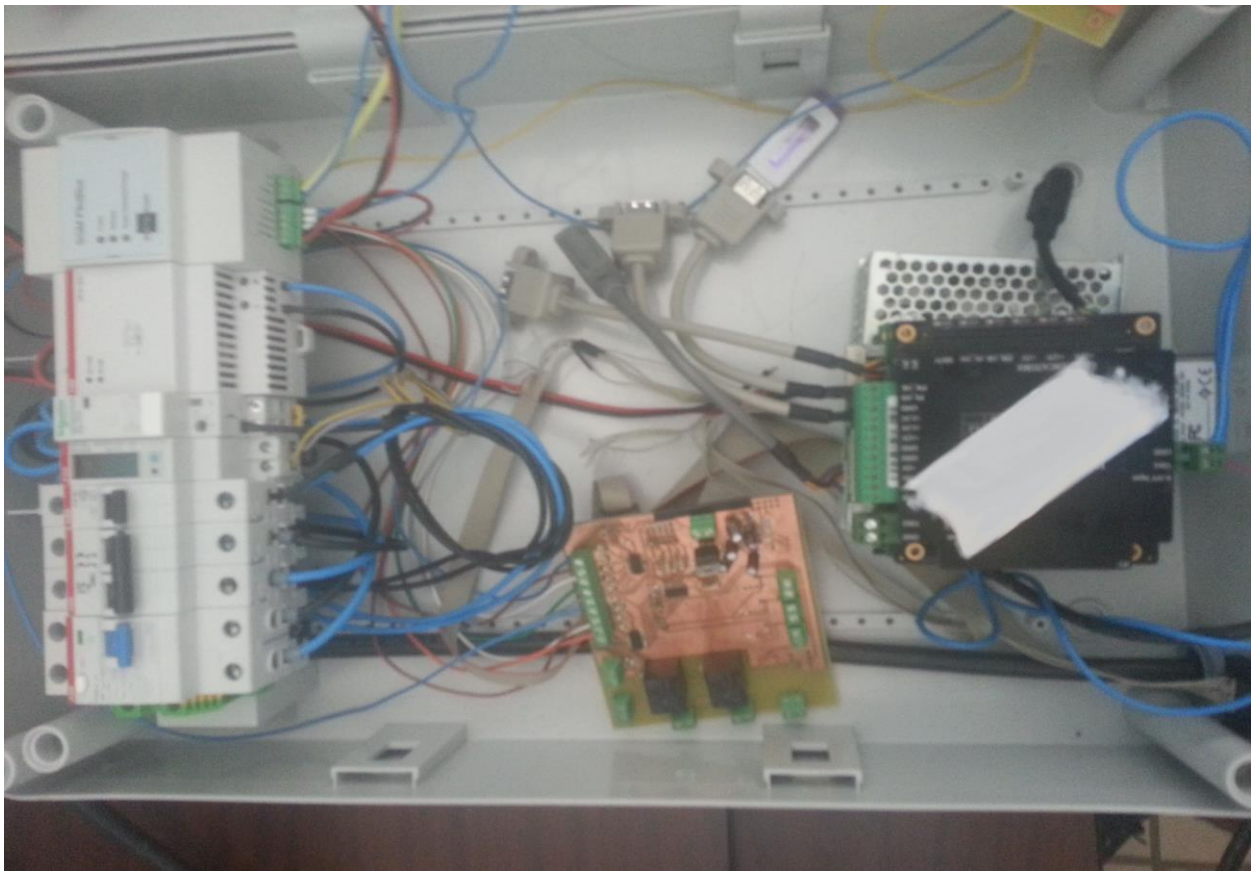


Ilustración 12: Interior del PR

El PC-104 es el elemento encargado de ejecutar el programa del PR, que se encarga de atender las peticiones SOAP y de comunicarse con la electrónica para leer y escribir las señales digitales de la placa de circuito impreso.

Ejemplos de estas señales son: el cable de alimentación, botones de emergencia, detectores de corriente, LEDs, conmutadores eléctricos (para dar corriente al cable de alimentación del vehículo eléctrico), etc.

Todos estos elementos se utilizan en la ejecución de maniobras para, por ejemplo, pausar una maniobra de carga por emergencias, o para detectar averías o situaciones no esperadas en el poste.

2.3 Sistema de información externo a la infraestructura de recarga (SI)

El sistema de información es una entidad que regula toda la instalación de manera externa al sistema a través de Internet. Su objetivo es operar varios CIR ubicados en diferentes puntos geográficos.

El SI es el encargado de ordenar maniobras en los CIR, y es el único elemento que puede realizar acciones sobre el controlador o los puntos de carga.

La lista de maniobras que el SI puede realizar sobre la infraestructura es la siguiente:

- Programar una recarga
- Abortar una recarga en ejecución
- Pausar una recarga en ejecución
- Apagado y re-arranque de un PR
- Actualización del software de un PR
- Actualización de configuración de un PR

Para realizar una maniobra, el SI debe enviar un mensaje ‘Nueva Maniobra’ al CIR, donde especifica los siguientes campos:

- Maniobra a realizar
- Momento en el que debe realizarse la maniobra
- Otros parámetros opcionales (Vehículo sobre el que se debe ejecutar la maniobra, URL para descargar el nuevo software de actualización, parámetros de configuración a modificar, etc.)

Una vez que la maniobra ha sido enviada y aceptada por el CIR, el módulo de cola de maniobras del CIR se encargará de ejecutar esa maniobra en el momento adecuado, intercambiando mensajes OCPP con los PR en caso de que fuera necesario para la ejecución de la maniobra.

```
************************************************
* Efleet SI | CIR @ http://localhost:8081
************************************************
a: Envía nueva maniobra
h: Maniobras disponibles
q: Salir
************************************************
> a

Introduzca datos de la maniobra:
Id maniobra: 1
Id poste de recarga (PR): 111
Introduzca opciones (opcional):

Enviando Maniobra
- Maniobra ID: 1
- PR ID: 111
- Opcional:
- Actual: 1408989571 Comienzo: 1408989574 Diferencia: 3

Respuesta: ok

************************************************
* Efleet SI | CIR @ http://localhost:8081
************************************************
a: Envía nueva maniobra
h: Maniobras disponibles
q: Salir
************************************************
```

Ilustración 13: Programa del SI

2.4 Entorno de usuario (EU)

El entorno de usuario es el elemento encargado de mostrar información relacionada con el estado del sistema al operario de la infraestructura de recarga.

En este entorno, el operario puede ver en una interfaz de usuario amigable información sobre las maniobras pendientes, en ejecución o las maniobras ya finalizadas.

El EU también dispone de un sistema que permite al operario solicitar una nueva maniobra.

Para solicitar una maniobra, el EU envía un mensaje de solicitud al CIR y este, a su vez, envía un mensaje de solicitud al SI. Sin embargo, es el SI el que finalmente tiene la potestad de decidir si realizar esa maniobra o ignorar la solicitud.

Si el SI decide tomar en cuenta la solicitud y ejecutarla, enviará un mensaje de nueva maniobra con los datos adecuados, tal y como se describió en el apartado 2.3. En caso contrario, el SI simplemente ignorará el mensaje de solicitud.

3 GUÍA DE USO

La siguiente guía pretende demostrar el funcionamiento y la utilización del sistema. Para ello, se hará uso del programa del CIR y del PR, y de dos programas que simulan un EU y un SI. Estos programas pueden ejecutarse en un entorno local a modo de pruebas. Solo hay que tener en cuenta los puertos que se utilizan en cada uno de ellos, de forma que no haya ningún programa que intente usar el mismo puerto que otro.

En nuestro caso, hemos utilizado un ordenador con el sistema operativo Fedora, y los puertos 8080, 8081, 8082 y 8083 para los distintos servidores. En esa misma máquina se encuentra funcionando un servidor web que sirve los archivos que se encuentran en el directorio /var/www/html, y que se utilizará para algunas de las maniobras.

3.1 Archivo de configuración del CIR

El CIR dispone de un archivo de configuración denominado "config" donde se especifican distintas opciones para su ejecución. Este archivo de configuración se lee cuando comienza el programa, de forma que es necesario reiniciar el programa del CIR si queremos que el programa lea los cambios que hacemos en este fichero.

Este archivo de configuración tiene este aspecto:

```
VERSION=1.0.0
VENDOR=TestVendor
PR_LISTEN_PORT=8080
SI_LISTEN_PORT=8081
EU_LISTEN_PORT=8082
PR_LISTEN_IP=localhost
SI_LISTEN_IP=localhost
EU_LISTEN_IP=localhost
CLIENT_PORT=8083
HEARTBEAT_INTERVAL=5
LOG_DIR=/var/www/html/pr_logs
LOG_URL_PREFIX=http://www.domain.com/logs/
SILOCATION=http://localhost:8084
```

El archivo de configuración está compuesto por pares nombre – valor, separados por un salto de línea. Cada uno de estas líneas indica lo siguiente:

- `VERSION` Versión del software. Se utiliza en algunos mensajes.
- `VENDOR` Fabricante del software

- `PR_LISTEN_PORT` Puerto donde escuchará el servidor de los mensajes que provienen de los PR
- `SI_LISTEN_PORT` Puerto donde escuchará el servidor de los mensajes que provienen del SI
- `EU_LISTEN_PORT` Puerto donde escuchará el servidor de los mensajes que provienen del EU
- `PR_LISTEN_IP` Dirección IP donde escuchará el servidor de los mensajes que provienen de los PR
- `SI_LISTEN_IP` Dirección IP donde escuchará el servidor de los mensajes que provienen del SI
- `EU_LISTEN_IP` Dirección IP donde escuchará el servidor de los mensajes que provienen del EU
- `CLIENT_PORT` Puerto donde está escuchando el servidor de los PR
- `HEARTBEAT_INTERVAL` Tiempo en segundos que indica los segundos máximos que pueden pasar entre mensajes provenientes de un PR. Si un PR tarda más de los segundos indicados en este campo en enviar un mensaje cualquiera, se considera que el PR está desconectado y se elimina de la lista de PRs activos
- `LOG_DIR` Directorio donde se guardaran los logs. Debe ser un directorio del servidor web.
- `LOG_URL_PREFIX` Es la URL que corresponde al directorio de los logs, de forma que un visitante a través de esta dirección pueda acceder al directorio de logs.
- `SILOCATION` URL que especifica la dirección IP y el puerto donde está escuchando el servidor del SI

3.2 Archivo de configuración del PR

Es un archivo con el mismo nombre y formato que el archivo de configuración del CIR, pero con valores de configuración diferentes. Este archivo tiene este aspecto:

```
VERSION=1.0.0
VENDOR=TestVendor
LISTEN_PORT=8083
CIR_URL=http://localhost:8080
PR_ID=111
```

En este archivo, las configuraciones son las siguientes:

- `VERSION` Versión del software. Se utiliza en algunos mensajes.
- `VENDOR` Fabricante del software.
- `LISTEN_PORT` Puerto donde escuchará el servidor de los mensajes OCPP que provienen del CIR. Esta configuración debe coincidir con el valor `CLIENT_PORT` del archivo de configuración del CIR.
- `CIR_URL` URL que especifica la dirección IP y el puerto donde está escuchando el servidor del CIR. Debe coincidir con los valores especificados en `PR_LISTEN_IP` y `PR_LISTEN_PORT` del archivo de configuración del CIR.
- `PR_ID` Identifica al PR de forma inequívoca en el sistema. Se utiliza para ordenar maniobras sobre los PR.

3.3 Inicio y parada del sistema

Para iniciar el CIR, solo es necesario ejecutar el archivo “cir”. Al hacerlo, si todo va correctamente, veremos que aparece algo como lo siguiente.

```
[carlos@localhost codigo_cir]$ ./cir
[INFO] Press 'R' key to execute a remote charge operation on every client
[INFO] Press 'p' key to display the waiting manoeuvres list
[INFO] Press 'x' key to display the executing manoeuvres list
```

```
[INFO] Press 'e' key to display the executed manoeuvres list
[INFO] Started SI server on localhost:8081
[INFO] Started EU server on localhost:8082
[INFO] Started PR server on localhost:8080 with 3 threads
```

Los mensajes precedidos de [INFO] son mensajes de información que le muestran las últimas acciones que el CIR ha realizado.

En este momento, el CIR habrá creado un servidor que recibe peticiones del protocolo OCPP en la dirección y el puerto configurados en las directivas PR_LISTEN_IP y PR_LISTEN_PORT de su archivo de configuración (en este ejemplo, en la dirección localhost y el puerto 8080). De igual forma, se habrán creado los servidores para el SI y el EU siguiendo las directivas del archivo de configuración.

El servidor de los PR es un servidor especial donde las peticiones se atienden por varios hilos al mismo tiempo (3 hilos en este caso), de forma que es capaz de procesar más peticiones que los otros servidores.

Llegado a este punto, el sistema permanece a la espera de órdenes provenientes del SI o EU, o de mensajes OCPP originados por los PR.

Una vez que el CIR está iniciado, podemos proceder a asociar un PR al sistema. Para ello solo debemos colocarnos en la carpeta del PR y ejecutar el archivo binario "pr", de forma que veremos la siguiente salida por pantalla:

```
[carlos@localhost codigo_pr]$ ./pr
[INFO] Server started on port 8083
--> BootNotification Request to CIR at http://localhost:8080 calling with PR ID
111
<-- BootNotification Response from CIR. Heartbeat Interval: 5 seconds
```

Los mensajes precedidos de [INFO] son mensajes de información que le muestran las últimas acciones que el PR ha realizado, mientras que los mensajes precedidos con flechas son mensajes que indican que se ha intercambiado un mensaje OCPP con el CIR.

Los mensajes con flechas hacia la derecha son mensajes OCPP que se originan en el PR, y los mensajes con flechas hacia la izquierda son mensajes que se originan en el CIR.

En este momento, el PR habrá creado un servidor que recibe peticiones del protocolo OCPP en el puerto configurado en la directiva LISTEN_PORT de su archivo de configuración (en este ejemplo, el puerto 8083).

En cuanto el PR es iniciado, comienza un procedimiento por el cual el PR intenta asociarse con el CIR enviando un mensaje BootNotification. Tal y como podemos observar en la salida por pantalla del programa PR, el PR ha enviado un mensaje BootNotification donde indica al CIR que su PR_ID es el 111. El CIR responde indicando que el `HEARTBEAT_INTERVAL` es de 5 segundos.

Si observamos la salida del programa del CIR, veremos que ha aparecido un nuevo mensaje:

```
<-- BootNotification Request from 127.0.0.1:51871 with ID 111.
```

Este mensaje indica que se ha asociado un nuevo PR al CIR con ID 111 y que se encuentra en la dirección IP 127.0.0.1. A partir de este momento el CIR y el PR se comunicarán mediante mensajes OCPP, y el PR deberá preocuparse de cumplir el mecanismo de Heartbeats.

El mecanismo de Heartbeats es un procedimiento "keep alive" por el cual envía un mensaje OCPP al CIR cada `HEARTBEAT_INTERVAL` segundos para notificar que sigue operando correctamente. Si el PR no envía un mensaje OCPP en el plazo establecido, el CIR entiende que ese PR ya no está operativo, y lo elimina de su lista de PRs activos.

Cuando el PR envíe un mensaje de HeartBeat, se producirá el siguiente mensaje en el programa del PR:

```
--> Heartbeat Request to CIR
```

Lo que produce el siguiente mensaje en el programa del CIR:

```
<-- Heartbeat Request from PR ID 111 at 127.0.0.1:51872.
```

Estos mensajes aparecerán de forma periódica, dejando entre ellos tantos segundos como se especificara en la propiedad `HEARTBEAT_INTERVAL` del archivo de configuración del CIR. En este ejemplo, estos mensajes se producen cada 5 segundos.

Cuando queramos detener el PR, solo debemos pulsar la combinación de teclas CTRL+C, que enviará una señal SIGINT que detendrá el proceso del PR, de forma que producirá la siguiente salida por pantalla:

```
^C -> Received SIGINT. Exiting...
[INFO] Server finished
```

Esto provocará que no se siga realizando el procedimiento de heartbeats, de forma que pasados unos instantes podremos ver el siguiente mensaje en el programa del CIR:

```
[INFO] Removing client http://127.0.0.1:8083 for inactivity
```

Ese mensaje indica que el PR ha sido eliminado de la lista de PRs activos debido a que no ha enviado un mensaje en la cantidad de segundos especificada en `HEARTBEAT_INTERVAL`

Para detener el CIR, solo debemos pulsar la combinación de teclas CTRL+C. Esto enviará una señal SIGINT al proceso que detendrá los servidores y finalizará el proceso del CIR.

```
^C -> Received SIGINT. Exiting...
[INFO] SI server finished
[INFO] PR server finished
[INFO] EU server finished
```

3.4 Programar maniobras

Junto al CIR y al PR se incluye el programa del SI. Es un programa sencillo con pocas funcionalidades, pero se espera que su comportamiento se extienda en el futuro. El SI está programado para actuar sobre el CIR ubicado en localhost, en el puerto 8081. En esta sección suponemos que el CIR está funcionando con el servidor del SI configurado correctamente, y que hay un PR asociado al sistema con el ID 111.

Al iniciar el SI, veremos algo como lo siguiente:

```
[carlos@localhost codigo_si]$ ./si
*********************************************
* SI | CIR @ http://localhost:8081
* Server started on localhost:8084
*********************************************
a: Envía nueva maniobra
h: Maniobras disponibles
q: Salir
*********************************************
>
```

Esto nos indica que el SI está listo para enviar nuevas maniobras, y para recibir peticiones de maniobras por parte del CIR.

Si pulsamos ‘h’, podemos ver las maniobras que podemos realizar sobre el CIR:

```
1: Carga
2: Aborto
3: Pausa (Opciones: num de segundos)
6: Reinicio
7: Actualizacion (Opciones: url con el numero firmware
8: Cambiar configuracion (Opciones: en la forma PR_ID=23)
```

Todas las maniobras requieren que introduzcamos dos parámetros obligatorios y uno opcional (aunque el protocolo soporta más parámetros opcionales que no se usan en estos ejemplos), estos son: ID de la maniobra, ID del PR donde se ejecutará la maniobra, y unos parámetros opcionales (que pueden dejarse en blanco si no se usan). Con este programa de ejemplo, la maniobra se ejecutará siempre tres segundos después del instante en el que se programa la maniobra, aunque el protocolo permite especificar el instante que se desee.

Por ejemplo, para programar una maniobra de carga en el PR con identificador 2345, pulsaríamos la letra ‘a’ e introduciríamos los siguientes valores:

```
Introduzca datos de la maniobra:
Id maniobra: 1
Id poste de recarga (PR): 2345
Introduzca opciones (opcional):
```

Al hacerlo, el programa del SI enviará un mensaje ‘NuevaManiobra’ al CIR, y si la maniobra es aceptada veremos lo siguiente representado en la salida por pantalla:

```
Enviando Maniobra
- Maniobra ID: 1
- PR ID: 2345
- Opcional:
- Actual: 1409331003 Comienzo: 1409331005 Diferencia: 2


Respuesta: ok
```

En el CIR, al recibir la nueva maniobra, veremos que aparecen los siguientes mensajes:

```
[NEW MANOUEVER] NEW MANOUEVER ID 1 ON PR ID 2345 ]
[ERROR] Couldn't find PR with ID 2345 to execute manoeuvre with ID 1. Removed
from the list.
```

La primera línea nos indica que se ha recibido la maniobra correctamente, sin embargo, en la segunda línea podemos ver que no existe el PR con ID 2345 en el sistema, y por lo tanto la maniobra es descartada.

3.5 Maniobra de carga

Vamos a programar una maniobra de carga sobre el PR con ID 111 (que debería estar asociado al CIR siguiendo los pasos indicados en apartados anteriores) para ver el funcionamiento de la maniobra.

Primero, programamos la maniobra con el programa del SI:

```
Introduzca datos de la maniobra:
Id maniobra: 1
Id poste de recarga (PR): 111
Introduzca opciones (opcional):
```

Al hacerlo, veremos que en el CIR aparece el siguiente mensaje, indicando que la maniobra ha sido aceptada:

```
[NEW MANOUEVER] NEW MANOUEVER ID 1 ON PR ID 111 ]
```

Cuando llega el momento de ejecución de la maniobra (dentro de tres segundos), el CIR comienza el procedimiento para ordenar la carga en el PR 111, y lo indica de esta manera:

```
[INFO] Executing manoeuvre -- ID: 1 -- ID PR: 111
[INFO] Charge Maniouvre on PR ID 111
[STATE PR 111] SPR
--> RemoteStartTransaction Request to PR 111 at http://127.0.0.1:8083
[ERROR] The PR 111 at http://127.0.0.1:8083 didnt accept the request
[STATE PR 111] FIN0
[INFO] Error. Finalizada carga con anomalias en PR http://127.0.0.1:8083.
[INFO] Finalizada maniobra ID 1.
```

Lo primero que podemos observar es la etiqueta `[STATE PR 111]`, esto indica que ha habido un cambio en la máquina de estados asociada al PR con ID 111. En este caso, puesto que comienza una maniobra de carga, el CIR coloca al PR 111 en el estado SPR (Solicitud poste recarga). Posteriormente, el CIR procede a enviar un mensaje RemoteStartTransaction para comenzar el proceso de carga, sin embargo, el PR parece no haber aceptado la petición, por lo que la máquina de estados del PR 111 pasa al estado FIN0 (Estado de finalización por anomalías), por lo que el CIR informa de que el PR no ha podido realizar la carga debido a anomalías, y declara la maniobra como finalizada.

Si observamos la salida generada en el programa del PR con ID 111, veremos lo siguiente:

```
<-- RemoteStartTransaction Request from CIR
[ERROR] Couldn’t start charge manoeuvre cause the cable is not connected.
```

La maniobra de carga no se ha podido realizar porque el cable del PR está desconectado. Esto indica que no hay ningún vehículo conectado al poste de recarga. Para solucionar esto en un sistema real en producción, bastaría con conectar el cable de carga al poste. El programa del PR se encargaría de detectar un cambio en su conector, y modificar la variable adecuada para que se pueda realizar la carga.

Sin embargo, en un entorno de pruebas como este en el que no disponemos de un poste de recarga físico, podemos utilizar el teclado para simular esas modificaciones de variables. La letra ‘C’ se encarga de simular la conexión del cable, y la letra ‘c’ de la desconexión. Por lo tanto, pulsamos la letra ‘C’ en el PR, lo que produce lo siguiente:

```
C
[DBG] Cable conectado
```

Volvemos al SI, programamos la maniobra de carga de nuevo de la misma forma, y observamos de nuevo la salida que se produce en el CIR.

```
[INFO] Executing manoeuvre -- ID: 1 -- ID PR: 111
[INFO] Charge Maniouvre on PR ID 111
[STATE PR 111] SPR
--> RemoteStartTransaction Request to PR 111 at http://127.0.0.1:8083
<-- RemoteStartTransaction Response from PR http://127.0.0.1:8083.
[STATE PR 111] EFC
```

Esta vez el PR responde al mensaje RemoteStartTransaction correctamente, y por lo tanto el CIR transiciona la máquina de estados del PR 111 al estado EFC (Espera fin de carga). En este estado, el CIR espera a que

transcurra el número de segundos apropiado para que la carga se completa, o bien a que el PR avise de un cambio de estado mediante un mensaje OCPP.

En el programa del PR vemos algo parecido:

```
<-- RemoteStartTransaction Request from CIR
[NEW STATE] EFC
```

En el PR, la etiqueta `[NEW STATE]` indica un cambio de estado en la propia máquina del PR. Esta máquina no es la misma que la que hay en el CIR, cada una son independientes la una de la otra. Sin embargo, los mensajes OCPP se encargan de sincronizar sus estados. En este caso, el PR coloca su máquina de estados en el estado EFC (Espera fin de carga) debido a que recibe el mensaje RemoteStartTransaction, lo que efectivamente nos indica que está realizando una operación de carga en este momento.

Hay varias maneras de que una maniobra de carga en un PR finalice. La primera es que se dispare un temporizador en el CIR, de forma que se considere que el PR ya ha cargado al vehículo durante el tiempo apropiado, y por lo tanto detenga la carga mediante un mensaje RemoteStopTransaction. Si dejamos pasar diez segundos, podremos ver este comportamiento, que se refleja en el programa del CIR de la siguiente manera:

```
[STATE PR 111] SFTP
--> RemoteStopTransaction Request to PR 111 at http://127.0.0.1:8083
<-- RemoteStopTransaction Response from PR http://127.0.0.1:8083.
[STATE PR 111] FINTP
[INFO] Fin de carga por expiracion en PR http://127.0.0.1:8083 correcto.
[INFO] Finalizada maniobra ID 1.
```

En este momento, el CIR pasa al PR 111 al estado SFTP (Solicitud de finalización por tiempo planificado) y envía un mensaje RemoteStopTransaction al PR, para notificarle que debe finalizar la operación de carga. Cuando el PR responde, el CIR coloca al PR en el estado FINTP (Finalización por tiempo planificado) y marca la maniobra como finalizada correctamente.

En el PR ocurre la situación paralela a esto:

```
<-- RemoteStopTransaction Request from CIR
[NEW STATE] FCIR
```

El PR recibe un mensaje RemoteStopTransaction del CIR, por lo que pasa al estado FCIR y finaliza la operación de carga, permaneciendo a la espera de nuevas órdenes.

Otra manera de parar una maniobra de carga es, por ejemplo, desconectar el cable de carga del vehículo eléctrico mientras se está cargando el vehículo. En nuestro caso, en este entorno de desarrollo, eso lo haríamos introduciendo la letra ‘c’ mientras se carga el vehículo (es decir, mientras el PR y el CIR se encuentran en el estado EFC), lo que produciría lo siguiente en el PR:

```
[DBG] Cable desconectado
[NEW STATE] VDES
--> StopTransaction Request to CIR
[NEW STATE] FVDES
```

Como vemos, el PR detecta que el cable se ha desconectado y pasa al estado VDES (Vehículo desconectado), lo que provoca que se envíe un mensaje StopTransaction al CIR, tras el cual se pasa al estado FVDES (Finalización por vehículo desconectado) y el PR pasa a permanecer a la espera de nuevas órdenes.

En el CIR veríamos lo siguiente:

```
<-- StopTransaction Request from 127.0.0.1:52037
[STATE PR 111] FCPR
```

```
[INFO] Fin de carga en el PR http://127.0.0.1:8083 es correcto.
```

Lo que indica que se ha recibido un mensaje StopTransaction del PR, por lo que se pasa al estado FCPR (Fin de carga ordenado por PR), y por lo tanto finaliza la maniobra de carga.

3.6 Maniobra de aborto

La maniobra de aborto (id de maniobra 2) se encarga de finalizar de forma instantánea la carga que se está produciendo en el PR indicado. Para probarla, programamos una maniobra de carga tal y como se indicó en apartados anteriores, y programamos la maniobra de aborto para que se ejecute cuando el PR se encuentre en el estado EFC.

En el SI, introduciríamos los siguientes datos:

```
Introduzca datos de la maniobra:
Id maniobra: 2
Id poste de recarga (PR): 111
Introduzca opciones (opcional):
```

Llegado el momento de ejecución de la maniobra, y suponiendo que el PR se encuentra en el estado EFC, el CIR procedería de la siguiente manera:

```
[INFO] Executing manoeuvre -- ID: 2 -- ID PR: 111
[INFO] Aborting charge manoeuvre on PR 111.
[STATE PR 111] SFAB
--> RemoteStopTransaction Request to PR 111 at http://127.0.0.1:8083
<-- RemoteStopTransaction Response from PR http://127.0.0.1:8083.
[STATE PR 111] FINAB
[INFO] Fin de carga por aborto en PR http://127.0.0.1:8083 correcto.
[INFO] Finalizada maniobra ID 1.
```

Básicamente, el CIR pasa al estado SFAB (Solicitud de fin por aborto) y envía un mensaje RemoteStopTransaction al PR, de la misma manera que lo haría si finalizase la carga por que el temporizador haya terminado. Cuando recibe la respuesta del PR, pasa al estado FINAB (Finalización por aborto). Esto provoca en el PR la siguiente salida:

```
<-- RemoteStopTransaction Request from CIR
[NEW STATE] FCIR
```

Si observamos la salida que produce el PR, podemos observar que para él no hay diferencia con una finalización de carga normal, puesto que ambas se producen recibiendo un mensaje RemoteStopTransaction, y ambas hacen que el PR pase al estado FCIR. Sin embargo, el CIR si es capaz de detectar si la finalización se ha producido por un aborto o por una finalización por tiempo de expiración, ya que el estado de finalización de su máquina de estados es distinto.

3.7 Maniobra de pausa

Si queremos que el PR pause una carga durante una cantidad de tiempo y posteriormente prosiga, podemos utilizar la maniobra de pausa. Esta maniobra (con identificador de maniobra 3), acepta un parámetro opcional que indica el número de segundos que se pausará la maniobra de carga. Es necesario que haya una operación de carga en funcionamiento en el PR para que se ejecute esta maniobra. Para probarla, lanzamos esta maniobra con un tiempo de, por ejemplo, 15 segundos, cuando el PR se encuentre en el estado EFC.

En el SI, introducimos los siguientes datos:

```
Introduzca datos de la maniobra:
Id maniobra: 3
Id poste de recarga (PR): 111
Introduzca opciones (opcional): 15
```

Cuando la maniobra se ejecute, en el CIR veremos lo siguiente:

```
[INFO] Executing manoeuvre -- ID: 3 -- ID PR: 111
[INFO] Pause Maniouvre on PR ID 111
[STATE PR 111] SPAU
[INFO] Charge operation on PR 111 paused 15 seconds.
--> DataTransfer PAUSE Request to PR 111 at http://127.0.0.1:8083
<-- DataTransfer PAUSE Response from PR http://127.0.0.1:8083.
[STATE PR 111] PAUSA
```

Como vemos, el CIR se coloca en el estado SPAU (Solicitud de pausa), donde envía un mensaje DataTransfer al PR con ID 111. Cuando el PR responde al mensaje, indicando que acepta la solicitud de pausa, el CIR pasa al estado de PAUSA, donde permanece durante 15 segundos.

El programa del PR indica este proceso de esta manera:

```
<-- DataTransfer Request from CIR
[NEW STATE] PAUS
```

Donde se puede observar que ha recibido un mensaje DataTransfer del CIR, y que pasa al estado PAUS (Pausa), donde el cable está conectado y está realizando una operación de carga, pero no cede potencia eléctrica al vehículo.

Cuando finalizan los 15 segundos de pausa, ambos vuelven al estado EFC, a la espera de que finalice la carga.

En el CIR:

```
[STATE PR 111] EFC
```

En el PR:

```
[INFO] Pause finished
[NEW STATE] EFC
```

3.8 Maniobra de cambio de configuración

Es posible realizar un cambio sobre el archivo de configuración de un PR desde el SI, realizando una maniobra de cambio de configuración (identificador de maniobra 8). Para ello, solo debemos indicar el PR sobre el cual queremos cambiar la configuración, y el nombre y el valor de la directiva que queremos modificar en el campo opcional. Por ejemplo, para cambiar el PR_ID del PR 111 de forma que sea identificado como el PR 25, programaríamos la siguiente maniobra en el SI.

```
Introduzca datos de la maniobra:
Id maniobra: 8
Id poste de recarga (PR): 111
Introduzca opciones (opcional): PR_ID=25
```

Cuando llegue el momento de la ejecución, veremos lo siguiente en el CIR:

```
[INFO] Executing manoeuvre -- ID: 8 -- ID PR: 111
[INFO] Change Config Maniouvre on PR ID 111
--> ChangeConfiguration Request to PR 111 at http://127.0.0.1:8083 with params
PR_ID=25
<-- ChangeConfiguration Response from PR http://127.0.0.1:8083.
```

Estos mensajes indican que se ha enviado un mensaje ChangeConfiguration al PR con ID 111 para colocar en su directiva PR_ID del archivo de configuración el valor 25.

El PR reacciona de la siguiente manera:

```
[INFO] ChangeConfiguration for param PR_ID. Current 111 -> New 25
```

En esta línea el PR indica el valor actual de la directiva que se ha cambiado, y el valor nuevo que toma la directiva. Si miramos ahora el archivo de configuración del PR, veremos que el PR_ID ya indica el nuevo valor, sin embargo el PR sigue funcionando con el valor antiguo. Para que el PR tenga en cuenta los cambios, es necesario reiniciar el PR de forma manual, o mediante la maniobra de reinicio.

3.9 Maniobra de reinicio

Corresponde al identificador de maniobra 6 y nos permite reiniciar un PR de forma remota. Para ello, lanzamos la maniobra en el PR que queramos reiniciar. No es necesario especificar ningún parámetro opcional.

```
Introduzca datos de la maniobra:
Id maniobra: 6
Id poste de recarga (PR): 111
Introduzca opciones (opcional):
```

Cuando llegue el momento de la ejecución de la maniobra, en el CIR veremos lo siguiente:

```
[INFO] Executing manoeuvre -- ID: 6 -- ID PR: 111
[INFO] Restart Maniouvre on PR ID 111
--> DataTransfer RESTART Request to PR 111 at http://127.0.0.1:8083
<-- DataTransfer RESTART Response from PR http://127.0.0.1:8083.
[INFO] Removing PR with ID 111 because is about to restart.
```

Como podemos ver, esta maniobra se realiza también con un mensaje DataTransfer. Además, el CIR elimina el PR de la lista de PRs activos, ya que va a ser reiniciado próximamente.

En el PR, aparece lo siguiente:

```
<-- DataTransfer Request from CIR
[INFO] Restarting PR...
```

En un sistema real en producción, en este momento el PR se reiniciaría de forma automática. Sin embargo, en este entorno de pruebas, es necesario apagar el PR pulsando la combinación de teclas CTRL+C para que el PR proceda a reiniciarse.

3.10 Maniobra de actualización

Es posible actualizar el software de un PR desde el SI de manera remota. Para ello, solo es necesario generar el archivo binario que el PR ejecutará y subirlo a un servidor web. Después, programamos la maniobra de actualización (con identificador de maniobra 7, indicando en los parámetros opcionales la URL desde donde el PR puede descargar el nuevo binario).

Para estas pruebas, he generado un binario igual que el anterior, solo que este imprime el texto "NUEVO FIRMWARE!!" en cuanto se ejecuta, y luego procede normalmente. He subido este archivo a un servidor web, y puede descargarse desde la url http://localhost/firmware.bin.

Para realizar el proceso de carga, en el SI programamos la maniobra de la siguiente manera:

```
Introduzca datos de la maniobra:
Id maniobra: 7
Id poste de recarga (PR): 25
Introduzca opciones (opcional): http://localhost/firmware.bin
```

En el momento en el que se cumpla el tiempo de ejecución de la maniobra, podremos ver lo siguiente en el CIR:

```
[INFO] Executing manoeuvre -- ID: 7 -- ID PR: 25
--> UpdateFirmware Request to PR 25 at http://127.0.0.1:8083 with URL
http://localhost/firmware.bin
<-- UpdateFirmware Response from PR http://127.0.0.1:8083.
--> DataTransfer RESTART Request to PR 25 at http://127.0.0.1:8083
<-- DataTransfer RESTART Response from PR http://127.0.0.1:8083.
[INFO] Update Maniouvre on PR ID 25
[INFO] Removing PR with ID 25 because is about to restart.
```

Como vemos, el CIR primero envía un mensaje UpdateFirmware, que indica la URL desde donde debe bajarse el nuevo firmware. El PR se descarga el firmware automáticamente desde esa URL, y sustituye su archivo binario por el que acaba de descargar. Justo después, el CIR ordena una maniobra de reinicio de manera automática.

En el PR, vemos la siguiente salida:

```
[INFO] Received UpdateFirmware with URL http://localhost/firmware.bin
<-- DataTransfer Request from CIR
[INFO] Restarting PR...


NUEVO FIRMWARE!!
[INFO] Server started on port 8083
--> BootNotification Request to CIR at http://localhost:8080 calling with PR ID
25
<-- BootNotification Response from CIR. Heartbeat Interval: 5 seconds
```

Donde vemos cómo se han recibido los mensajes indicados anteriormente, y cómo comienza a funcionar el nuevo firmware, que imprime el texto "NUEVO FIRMWARE!!" en cuanto se inicia.

3.11 Ejemplo de entorno de usuario

También se incluye un programa como ejemplo de entorno de usuario. Este programa hace peticiones al CIR, lo que le permite pedir maniobras y obtener información sobre las maniobras pendientes, ejecutadas y en ejecución.

Al iniciar el programa del EU, veremos lo siguiente:

```
[carlos@localhost codigo_si]$ ./si
*********************************************
* EU | CIR @ http://localhost:8082
```

```
*********************************************
a: Hacer peticion de maniobra
p: Obtener maniobras pendientes
e: Obtener maniobras en ejecucion
f: Obtener maniobras finalizadas
q: Salir
*********************************************
>
```

Al pulsar la letra ‘a’ podemos introducir los datos para pedir una maniobra al SI, de la misma forma que en el programa del SI. Sin embargo, en el caso del EU, la petición de maniobra no implica que se vaya a producir la maniobra en el sistema. Es el SI el encargado de decidir si decide realizar la maniobra o ignorarla.

Si usamos alguna de las otras opciones (letras ‘p’, ‘e’ o ‘f’) obtenemos un listado de las maniobras pendientes, ejecutadas o en ejecución. Al hacerlo, obtenemos en pantalla algo como lo siguiente:

```
Respuesta
===============
1*0*25*0*1409508190*
1*0*25*0*1409508211*
```

En la respuesta, cada línea es una maniobra con 6 campos separados por asteriscos (*). Estos campos son, en orden de aparición:

- ID de la maniobra
- ID de la infraestructura de recarga
- ID del poste de recarga
- ID del vehículo eléctrico
- Timestamp con el tiempo en el que se ejecutó la maniobra
- Parámetros opcionales

Sin embargo, la aplicación del EU no es más que un ejemplo de uso del servicio web que ofrece el CIR para el EU. En el futuro, el EU tendrá una interfaz gráfica que permitirá ver todos los tipos de maniobra de una forma amigable para el usuario. Para implementar esta interfaz, el EU se comunicará con el CIR compartiendo los mismos mensajes que este programa de demostración, parseará las respuestas que el CIR devuelve y las mostrará en la interfaz.

4 PROTOCOLOS

The Open Charge Point Protocol (OCPP) is the accepted protocol of choice in 50 countries and over 10,000 charging stations.

- Open Charge Alliance -

La infraestructura descrita en los anteriores apartados hace un uso intensivo de diferentes protocolos de aplicación para permitir la comunicación entre los distintos elementos. Todos los protocolos involucrados se implementan sobre el protocolo SOAP, por lo que es interesante realizar una introducción previa a este protocolo antes de describir los de capas superiores.

4.1 Simple Object Access Protocol (SOAP)

Este protocolo [7] esta desarrollado por un conjunto de empresas (entre las que se encuentran Microsoft e IBM, por ejemplo) a partir de una versión primitiva creada por David Winerenen en 1998. Está destinado a realizar RPC o "Remote Procedure Calls" (Llamadas a procedimientos remotos). Esto consiste en, partiendo de una maquina cliente, ejecutar procedimientos en un servidor remoto y recibir el resultado de ese procedimiento en la maquina cliente. SOAP podría considerarse el predecesor de sistemas como CORBA, RMI u ORPC, que realizaban estas mismas funciones.

Tradicionalmente SOAP se ha comparado con REST en los debates técnicos, aunque no son exactamente iguales. La diferencia clave es que SOAP es un protocolo que define su propio mensaje, mientras que REST es una arquitectura de software (no es protocolo, ya que no tiene mensaje propio). Sin embargo, ambas tecnologías suelen utilizarse para casos de uso similares.

Sin embargo, SOAP cuenta con una serie de características que han provocado una gran adopción por parte del sector empresarial. Entre estas, podríamos destacar las siguientes:

- Es un estándar del World Wide Web Consortium.
- Está basado puramente en XML, que no es más que texto plano. Esto le otorga gran interoperabilidad, ya que puede implementarse sobre cualquier lenguaje y plataforma.
- Puede transportarse sobre cualquier protocolo que transmita texto, aunque normalmente se suele transportar sobre HTTP. Esto permite que los servicios SOAP sean accesible incluso a través de la mayoría de los firewalls.
- Se puede aplicar autenticación y cifrado al protocolo. En el caso más común, esto se consigue utilizando HTTPS en lugar de HTTP [8].
- Soporta extensiones para aumentar las funciones del protocolo. Un ejemplo de extensión es la dedicada a la seguridad del protocolo.

Un mensaje SOAP no es más que un documento XML que contiene los siguientes elementos:

- Envelope (sobre): Es el elemento raíz del documento que incluye todos los demás. Indica que el documento es un mensaje SOAP, lo que permite al receptor identificarlo.
- Header (cabeceras): Es un elemento no obligatorio que contiene información sobre el mensaje relevante para la aplicación. Las extensiones de SOAP suelen hacer uso de este elemento cuando necesitan

comunicar con el otro extremo algún dato relevante. Un ejemplo de esto es el uso de autenticación.

- Body (cuerpo): Contiene la carga del mensaje, los datos que finalmente llegarán a la aplicación.
- Fault (error): Es un elemento opcional que se encuentra cuando ha habido un error en el otro extremo de la comunicación.

A continuación se incluye un mensaje SOAP que sirve como ejemplo.

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```

SOAP también incluye una serie de reglas para codificar los tipos de datos utilizados por la aplicación. Estos tipos de datos están divididos en tipos de datos básicos y compuestos.

La siguiente tabla muestra los tipos de datos básicos que define SOAP.

Tabla 4–1. Tipos de datos básicos de SOAP

Tipo de dato	Ejemplos
string	Ejemplo de cadena de texto
boolean	true, false, 1, 0
float	-INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN
double	-INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN
decimal	-1.23, 0, 123.4, 1000.00
binary	100010
integer	-126789, -1, 0, 1, 126789
nonPositiveInteger	-126789, -1, 0
negativeInteger	-126789, -1
long	-1, 12678967543233
int	-1, 126789675
short	-1, 12678
byte	-1, 126
nonNegativeInteger	0, 1, 126789
unsignedLong	0, 12678967543233
unsignedInt	0, 1267896754
unsignedShort	0, 12678
unsignedByte	0, 126
positiveInteger	1, 126789
date	1999-05-31, ---05
time	13:20:00.000, 13:20:00.000-05:00

Los tipos de datos complejos se dividen en listas y estructuras, y parten de los tipos de datos básicos. Las listas nos permiten definir un conjunto de un tipo, mientras que las estructuras definen un conjunto de diferentes tipos. Esto nos permite, por ejemplo, crear una lista de un tipo complejo, o una estructura que contiene tipos básicos y tipos complejos que hayan sido definidos anteriormente.

Junto a SOAP, es común utilizar un sistema de descripción del servicio como WSDL (Web Services Description Language). WSDL [9] es un lenguaje que permite describir en un archivo XML un servicio web. En el archivo WSDL se describen los siguientes aspectos del servicio web:

- Tipos de datos: Definen los tipos de datos aceptados en el servicio web. Se pueden definir tipos de datos básicos o complejos, como ya se ha explicado anteriormente.
- Mensajes: indican los mensajes intercambiados entre interlocutores. Cada mensaje está compuesto por una serie de datos de tipo básico o complejo (y por lo tanto definidos anteriormente).
- Tipos de puerto: especifican las operaciones permitidas junto con los mensajes que se usan en esas operaciones. Normalmente cada operación está formada por dos mensajes (Uno de petición, y uno de respuesta), aunque no siempre es así.
- Bindings: indican que protocolos se usan para los tipos de puerto
- Servicio: localización del servicio (dirección HTTP, por ejemplo) y conjunto de puertos que el servicio implementa (hace referencia al binding y tipo de puerto asociado)

Cuando un servicio web se implementa con SOAP y HTTP, es común dedicar una dirección concreta para servir el WSDL del servicio. De esta forma, un cliente que desconozca las operaciones que permite el servidor puede descargar el WSDL y obtener una descripción del servicio y las operaciones que implementa, así como de los mensajes que tiene que enviar e interpretar para poder comunicarse con el servidor.

Por último, solo falta mencionar cómo implementar servicios web con SOAP. Generalmente se suelen utilizar librerías o frameworks junto con un archivo WSDL para desarrollar esta clase de servicios. PHP, por ejemplo, dispone de una serie de funciones incluidas en el lenguaje para hacer peticiones a servicios SOAP. En Ruby, la librería Savon es la más extendida para programar clientes SOAP. Respecto a servidores, Java es uno de los lenguajes más extendidos para desarrollarlos. Apache Axis y Spring son ejemplos de librerías (o frameworks, en el caso de Spring) para crear servicios web en Java.

En el caso concreto de este proyecto, hemos utilizado la librería gSOAP para desarrollar nuestros clientes y servidores en el lenguaje C a partir de archivos WSDL que describen los servicios implementados. Puesto que el software de los postes de recarga son ejecutados en máquinas embebidas, consideramos que era preferible evitar la instalación de la máquina virtual de Java y desarrollar las aplicaciones en código que pudiera ser ejecutado sin necesidad de un intérprete.

El uso de gSOAP será descrito en capítulos posteriores, mientras que los WSDL pueden encontrarse en los anexos que se encuentran al final del documento.

4.2 Open Charge Point Protocol (OCPP)

OCPP es un protocolo de aplicación que funciona sobre SOAP destinado a la comunicación entre postes de recarga y un sistema centralizado de gestión. El desarrollo del protocolo fue iniciado por la fundación E-laad [10] en el año 2009. En ese mismo año nace la Open Charge Alliance (OCA), organismo dedicado fundamentalmente a promover OCPP como estándar para la comunicación en las infraestructuras de recarga. Desde entonces, OCPP se ha convertido en el estándar de facto para este tipo de comunicaciones. En la actualidad, OCPP es utilizado en 50 países y unas 10.000 estaciones de carga.

El protocolo OCPP se puede encontrar documentado en la web de la OCA, donde se puede encontrar la especificación del protocolo, además de los archivos WSDL para generar los servicios web SOAP que implementan el protocolo. Actualmente OCPP se encuentra en la versión 1.5, que es la versión que he utilizado para desarrollar el sistema descrito en el presente documento.

4.2.1 Elementos del protocolo

En el protocolo OCPP intervienen dos elementos. El primero, denominado “Central System”, es el sistema centralizado de gestión (corresponde al CIR, como antes fue denominado en los elementos del sistema desarrollado). El elemento restante se denomina “Charge Point” y se refiere al poste de recarga (PR).

Además, nos encontramos con los “Connector”, que no son más que los espacios disponibles en un “Charge Point” para cargar un vehículo eléctrico.

4.2.2 Características

4.2.2.1 Autorización

La mayoría de las acciones que un usuario puede realizar usando OCPP (como por ejemplo, pedir el inicio de una carga, o la parada de una carga) necesitan ser autorizadas. La autorización en OCPP se basa en un código, denominado idTag, que el usuario introduce en el poste de recarga en el momento de hacer una petición. Este código se puede introducir en el poste de múltiples maneras, ya sea con un código de barras, con un dispositivo con NFC, con una tarjeta, etc.

Además, el mecanismo de grupos permite agrupar idTags para autorizar a varios usuarios al mismo tiempo. De esta forma, una autorización completa constaría de un idTag (para autorizar al usuario) y un parentIdTag opcional, que identificaría al grupo. Si el idTag del grupo está autorizado para realizar cierta acción, el usuario automáticamente también lo estará.

El sistema central es el encargado de autorizar a los diferentes idTag. Los postes de recarga se pondrán en contacto con el sistema central para comprobar si un idTag (o su parentIdTag) está autorizado para realizar acciones sobre el sistema.

4.2.2.2 Lista local de autorización

Los postes de recarga pueden tener una lista local de autorización, de forma que no necesiten al sistema central para autorizar a los usuarios. Los postes pueden construir esta lista cacheando resultados del proceso de autorización anteriores. El sistema central también puede enviar una lista de autorización a los postes de recarga, con las entradas que él encuentre oportunas.

4.2.2.3 Modo fuera de linea

OCPP está pensado para trabajar fuera de línea en caso de que se pierda la conectividad entre los postes de recarga y el sistema central. En este modo, los postes de recarga pueden autorizar a los usuarios únicamente con su lista local de autorización. Además, mantendrían en una cola de mensajes aquellos mensajes que fueran destinados al sistema central. En el momento en el que vuelva la conectividad, los postes enviarán los mensajes que tenían almacenados al sistema central.

4.2.2.4 Reservas

Con OCPP es posible reservar un poste de recarga (o un conector de éste) para un determinado idTag o parentIdTag. Las reservas vienen acompañadas de un tiempo de expiración, a partir del cual se considera que el poste de recarga ya no está reservado.

4.2.2.5 Data Transfer dependiente del fabricante

OCPP soporta el envío de mensajes sin formato ni longitud especificada, que permite que los fabricantes de postes de recarga y sistemas centrales puedan utilizar sus propios mensajes para realizar tareas específicas de su implementación. Hay que tener en cuenta, por lo tanto, que el mensaje DataTransfer utiliza unos campos cuyos formatos no están estandarizados, y cuyas implementaciónes dependen enteramente del fabricante.

4.2.3 Mensajes

A continuación describo los mensajes que forman parte de OCPP. Estos mensajes constan de una petición (Request) y una respuesta (Response). Se describe brevemente la situación en la que la petición es enviada para cada pareja de mensajes, así como los campos que contienen tanto el mensaje de petición como el de respuesta.

4.2.3.1 Authorize

Situación	El poste de recarga intenta autorizar una entidad, por lo que envía este mensaje al sistema central para comprobar si el identificador es permitido.
Petición	idTag: Identificador a autorizar

Respuesta	idTagInfo: Información sobre la autorización que se envió (si se permite o no).

4.2.3.2 BootNotification

Situación	El poste de recarga acaba de iniciarse y quiere notificar al sistema central de ello.
Petición	chargeBoxSerialNumber: Opcional. Número de serie del poste de recarga. chargePointModel: Modelo del poste de recarga. chargePointVendor: Fabricante del poste de recarga. firmwareVersion: Opcional. Versión del software del poste de recarga. iccid: Opcional. ICCID de la SIM del poste en caso de que tuviera. imsi: Opcional. IMSI de la SIM del poste en caso de que tuviera. meterSerialNumber: Opcional. Número de serie del dispositivo que mide la potencia que el poste es capaz de aportar. meterType: Opcional. Tipo de dispositivo que realiza la medida de potencia.
Respuesta	currentTime: Hora que el sistema central tiene en ese momento. heartbeatInterval: Numero de segundos que el poste de recarga dispondrá para enviar un mensaje OCPP antes de que el sistema central la elimine por inactividad. status: Identifica si el poste ha sido aceptado por el sistema central o no.

4.2.3.3 ReserveNow

Situación	El sistema central reserva un conector de un poste de recarga.
Petición	connectorId: Conector que se va a reservar en el poste de recarga. expiryDate: Instante en el que la reserva finaliza. idTag: Identificador para el cual se va a reservar el conector. parentIdTag: Opcional. Identificador padre para el cual se va a reservar el conector. reservationId: Identificador único para esta reserva.
Respuesta	status: Indica si la reserva se llevó a cabo o no satisfactoriamente.

4.2.3.4 CancelReservation

Situación	El sistema central cancela una reserva realizada previamente
Petición	reservationId: Identificador de la reserva a cancelar.
Respuesta	status: Indica si la reserva se canceló o no satisfactoriamente.

4.2.3.5 ChangeAvailability

Situación	El sistema central cambia el estado de uno de los conectores del poste de recarga. Estos estados pueden ser “Operativo” o “No operativo”. No se permiten cargas en conectores en estado “No operativo”.

Petición	connectorId: Conector al cual se le cambiará el estado. type: “Operativo” o “No operativo”.
Respuesta	status: Indica si el cambio se llevó a cabo o no satisfactoriamente.

4.2.3.6 ChangeConfiguration

Situación	El sistema central cambia un parámetro de configuración del poste de recarga.
Petición	key: Parámetro a modificar value: Nuevo valor que el parámetro debe tomar.
Respuesta	status: Indica si el cambio se llevó a cabo o no satisfactoriamente.

4.2.3.7 ClearCache

Situación	El sistema central limpia la cache de autorización de un poste de recarga.
Petición	Sin campos.
Respuesta	status: Indica si la caché se limpió o no satisfactoriamente.

4.2.3.8 DataTransfer

Situación	El sistema central envía un mensaje de carácter general y dependiente del fabricante al poste de recarga, o viceversa.
Petición	vendorId: Identificador del fabricante messageId: Opcional. Campo adicional para identificar el mensaje. data: Opcional. Datos sin longitud ni formato específico.
Respuesta	status: Indica si la petición se procesó correctamente. data: Opcional. Datos sin longitud ni formato específico.

4.2.3.9 DiagnosticsStatusNotification

Situación	El poste de recarga envía información de estado al sistema central.
Petición	status: Contiene información de estado del poste de recarga.
Respuesta	Sin campos.

4.2.3.10 FirmwareStatusNotification

Situación	El poste de recarga ha terminado de instalar un nuevo firmware.
Petición	status: Contiene información sobre el estado del proceso de instalación
Respuesta	Sin campos.

4.2.3.11 GetConfiguration

Situación	El sistema central quiere obtener uno o varios valores de configuración de un poste de recarga.
Petición	key: Lista de valores de configuración a obtener.
Respuesta	configurationKey: Pares clave y valor de los valores de configuración que se han pedido. unknownKey: Lista de valores de configuración que se desconocen.

4.2.3.12 GetDiagnostics

Situación	El sistema central pide información de diagnóstico a un poste de recarga.
Petición	location: Directorio donde el archivo con los diagnósticos debe ser colocado. retries: Opcional. Máximo número de reintentos que el poste debe realizar para subir un archivo de diagnósticos a la carpeta especificada. retryInterval: Opcional. Intervalo en segundos entre reintentos. startTime: Opcional. Fecha y hora del evento más antiguo que debe incluirse en el informe. stopTime: Opcional. Fecha y hora del evento más nuevo que debe incluirse en el informe.
Respuesta	fileName: Nombre del archivo que el informe tomará en el directorio que se especificó en la petición.

4.2.3.13 GetLocalListVersion

Situación	El sistema central quiere conocer la versión de la lista local de autorización que el poste de recarga posee.
Petición	Sin campos.
Respuesta	listVersion: Versión de la lista que el poste posee.

4.2.3.14 Heartbeat

Situación	El poste de recarga envía este mensaje cuando no ha enviado un mensaje en los últimos X segundos, donde X es el tiempo especificado en la respuesta del mensaje BootNotification.
Petición	Sin campos.
Respuesta	currentTime: Hora actual del sistema central.

4.2.3.15 MeterValues

Situación	El poste de recarga envía valores medibles como potencia, voltajes o intensidades que considere oportunos al sistema central cuando está cediendo potencia eléctrica a un vehículo.

Petición	connectorId: Conector en el cual se tomaron las medidas. transactionId: Opcional. Transacción relacionada con estos valores. values: Valores medidos con sus respectivos timestamps.
Respuesta	Sin campos.

4.2.3.16 RemoteStartTransaction

Situación	El sistema central pide al poste de recarga que comience a entregar potencia en uno de sus conectores.
Petición	connectorId: Conector al que aplica. idTag: Identificador que el poste debe usar para autorizar.
Respuesta	status: Indica si el poste de recarga aceptó o no la petición.

4.2.3.17 RemoteStopTransaction

Situación	El sistema central pide al poste de recarga que detenga el cese de potencia que se pidió anteriormente.
Petición	transactionId: Identificador de la transacción para que el poste de recarga pueda identificar el proceso que debe detener.
Respuesta	status: Indica si el poste de recarga aceptó o no la petición.

4.2.3.18 Reset

Situación	El sistema central solicita un reinició de un poste de recarga.
Petición	type: Tipo de reinicio: soft o hard.
Respuesta	status: Indica si el poste realizará el reinicio.

4.2.3.19 SendLocalList

Situación	El sistema central envía una lista de autorización local que debe utilizarse en caso de que el poste de recarga no pueda comunicarse con el sistema central.
Petición	hash: Opcional. Un hash de la lista, para comprobar si se recibió correctamente en el destino. listVersion: Version de la lista que se envía. localAuthorisationList: Opcional. Puede contener todos los valores si es una lista completa, o solo las diferencias con la lista actual del poste si es una lista diferencial. updateType: Indica si la lista es completa o diferencial.
Respuesta	hash: Opcional. Un hash calculado sobre la nueva lista de autorización, para que el sistema central compruebe si es la adecuada. status: Indica si el poste aceptó o no la lista.

4.2.3.20 StartTransaction

Situación	El poste de recarga notifica al sistema central del comienzo de una entrega de potencia en uno de sus conectores.
Petición	connectorId: Conector al que aplica. idTag: Identificador para el que se comienza la transacción. meterStart: Valor en W/h que el conector tiene al comienzo de la transacción. reservationId: Opcional. Identificador de la reserva que el poste satisface al comenzar el proceso de carga. timestamp: Momento en el que la transacción comienza.
Respuesta	idTagInfo: Información sobre el identificador que se envió en la petición. transactionId: Identificador de la transacción que proporciona el sistema central.

4.2.3.21 StopTransaction

Situación	El poste de recarga notifica al sistema central de que una transacción de potencia finalizó.
Petición	idTag: Opcional. Identificador de la entidad que pidió que la transacción finalizara. meterStop: Valor en W/h para el conectar en el momento de finalización de la transacción. timestamp: Momento en el que la transacción finaliza. transactionId: Identificador de la transacción que se recibió en la respuesta del mensaje StartTransaction. transactionData: Datos opcionales detallados sobre la transacción destinados a ayudar a la facturación.
Respuesta	idTagInfo: Información sobre el identificador que se envió en la petición.

4.2.3.22 StatusNotification

Situación	Un poste de recarga envía información de estado al sistema central.
Petición	connectorId: Identificador del conector sobre el que se informa. Si el identificador es 0 significa que el estado aplica a todo el poste de recarga. errorCode: Codigo de error reportado por el poste. info: Opcional. Campo sin formato ni longitud especifica para información adicional. status: Estado actual del poste. timestamp: Opcional. Tiempo para el cual se describe el estado. vendorId: Opcional. Identificador del fabricante. vendorErrorCode: Opcional. Código de error especifico del fabricante.
Respuesta	Sin campos.

4.2.3.23 UnlockConnector

Situación	El sistema central quiere desbloquear el conector de un poste de recarga.
Petición	connectorId: Conector que se va a desbloquear
Respuesta	status: Indica si el poste desbloqueó el conector.

4.2.3.24 UpdateFirmware

Situación	El sistema central quiere actualizar el software de un poste de recarga.
Petición	location: URI desde donde debe descargarse la actualización. retries: Opcional. Número de intentos máximos para obtener la actualización. retrieveDate: Momento en el que el poste debe descargar la actualización. retryInterval: Opcional. Numero de segundos que deben pasar entre reintentos.
Respuesta	Sin campos.

4.3 Protocolos de comunicación entre CIR, SI y EU

Estos protocolos están desarrollados sobre SOAP para permitir conectar a un controlador de infraestructura de recarga (CIR) con un sistema de información (SI) y un entorno de usuario (EU). Son tres protocolos básicos, compuestos de pocos mensajes, que realizan unas tareas concretas del proyecto que se presenta en este documento.

A continuación presentamos los mensajes de estos protocolos, junto con la situación en la que se hacen uso, y los campos de los que se componen.

4.3.1 SI

Este protocolo reúne los mensajes que el SI es capaz de recibir.

4.3.1.1 Nueva petición de maniobra

Este mensaje se envía cuando el CIR recibe una petición de maniobra desde el EU. El CIR se ocupa simplemente de reenviar la información que el EU le ha solicitado, de forma que el SI haga lo que considere más oportuno con ella. En el ejemplo de SI que acompaña a este documento, el SI envía una nueva maniobra cada vez que se recibe una petición de maniobra.

Los campos que forman este mensaje son:

- idMan: ID de la maniobra a ejecutar
- irDes: ID de la infraestructura de recarga de destino (opcional)
- prDes: ID del poste de recarga de destino
- veDes: ID del vehículo eléctrico de destino (opcional)
- comienzo: Timestamp con el momento en el que debe producirse la maniobra

4.3.2 CIRSI

Este servicio web reúne los mensajes que el CIR es capaz de recibir para comunicarse con el SI.

4.3.2.1 Nueva maniobra

Cuando el SI desea enviar una nueva maniobra para que el CIR ejecute emplea este mensaje. Utiliza los siguientes campos:

- idMan: ID de la maniobra a ejecutar
- irDes: ID de la infraestructura de recarga de destino (opcional)
- prDes: ID del poste de recarga de destino
- veDes: ID del vehículo eléctrico de destino (opcional)
- comienzo: Timestamp con el momento en el que debe producirse la maniobra

4.3.3 CIREU

Este servicio web reúne los mensajes que el CIR recibe para comunicarse con el EU.

4.3.3.1 Obtener lista de maniobras pendientes

Este mensaje no recibe ningún parámetro y devuelve todas las maniobras pendientes de ejecución en el CIR en una sola cadena de texto. En esta cadena, cada maniobra está separada por un salto de línea, y tiene el siguiente formato:

```
A*B*C*D*E*F
```

Donde:

- A: ID de la maniobra
- B: ID de la infraestructura de recarga
- C: ID del poste de recarga
- D: ID del vehículo eléctrico
- E: Timestamp cuando se programó la maniobra
- F: Parámetros opcionales (no siempre presente, al igual que el asterisco que lo precede)

4.3.3.2 Obtener lista de maniobras en ejecución

Exactamente el mismo caso, pero devuelve una lista de maniobras en ejecución

4.3.3.3 Obtener lista de maniobras ejecutadas

Igual que el caso anterior, pero devuelve una lista de maniobras ejecutadas

4.3.3.4 Nueva petición de maniobra

El EU envía este mensaje al CIR cuando el operario quiere solicitar una maniobra al SI. Sigue exactamente el mismo formato que el mensaje con el mismo nombre del servicio web SI.

5 HERRAMIENTAS

Para desarrollar el sistema, se han utilizado principalmente dos herramientas en el proceso de desarrollo. Estas herramientas han servido de punto de apoyo para realizar dos elementos críticos de las aplicaciones: las máquinas de estados y los clientes y servidores de SOAP.

En este apartado describo las herramientas utilizadas, así como el procedimiento realizado para su utilización y puesta en marcha.

5.1 SMC

State Machine Compiler (SMC) [11] es una aplicación escrita en Java que genera implementaciones de máquinas de estados en distintos lenguajes de programación. SMC solo necesita Java 1.7 para funcionar, y es capaz de generar código para Java, C, C++, Ruby, Javascript, Python, etc.

5.1.1 Elemento de una máquina de estados

Para trabajar con esta herramienta, es necesario entender los elementos que componen una máquina de estados de SMC. Estos son:

- Estado: Es una posición donde la máquina puede encontrarse en un cierto instante. Por ejemplo: Iniciada, Apagada, Funcionando, etc.
- Transición: Es el movimiento de un estado a otro. Las transiciones no ejecutan código, simplemente hacen pasar la máquina de estados de un estado al siguiente. Por ejemplo, la transición 'Iniciar', podría ser la que se ocupa de pasar del estado 'Apagada' al estado 'Iniciada'. Otro ejemplo, la transición 'Parar', podría ser la que se ocupa de pasar del estado 'Funcionando' al estado 'Apagada'.
- Acciones: Son operaciones asociadas a transiciones que se encargan de ejecutar código para realizar acciones sobre el sistema. Por ejemplo, podríamos asociar a la transición 'Iniciar' una función que imprimiera por pantalla el texto 'Iniciando el sistema', de forma que un usuario pudiera ver por pantalla cuando la máquina de estados está pasando del estado Apagada al estado Iniciada.
- Acción de entrada: Es igual que una acción, pero en lugar de asociarse a una transición, se asocia a un estado. En este caso, la acción se ejecuta en cuanto la máquina de estados entra al estado.
- Acción de salida: Igual que una acción de entrada, pero se ejecuta cuando la máquina de estados sale de ese estado.
- Guardias: Son sentencias condicionales asociadas a transiciones. Cuando se intente ejecutar una transición con guardia, la máquina de estados sólo cambiará de estado si se cumple la condición especificada en la guardia.

5.1.2 Ejemplo de archivo SM

Para usar SMC, describimos una máquina de estado es un archivo de texto, mediante una sintaxis concreta entendible por SMC. A partir de este archivo, SMC es capaz de generar el código que implementará esa máquina de estados en el lenguaje que le indiquemos.

A continuación explico la forma de describir una máquina de estados entendible por SMC. Para ello creamos un archivo con extensión sm. Los ejemplos están basados en el archivo sm del PR.

En primer lugar, es necesario indicarle a SMC cierta información referente a la máquina de estados.

```
%{
/** @file pr_sm_smc.c
 *  @brief Funciones de SMC
 *
 *  @author Carlos Rodriguez (CarlosRdrz)
 */
%}

%start MapaPrincipal::E_0
%class pr
%header pr_sm.h
%fsmclass pr_sm_smc
%map MapaPrincipal
%%
```

Los caracteres `%{` y `%}` marcan el contenido de un bloque de texto que no será interpretado por SMC. Este bloque de texto se insertará en el archivo de código fuente que generará SMC. Aprovechamos este bloque de texto para añadir documentación al archivo generado por SMC, de forma que sea procesado por doxygen.

La etiqueta `%start` indica el mapa y el estado (separados por dos puntos) en el que comenzará la máquina de estados cuando sea iniciada. Un mapa no es más que una colección de estados. Podríamos identificar a un mapa como una máquina de estados en sí misma, solo que un estado de un mapa podría transicionar a un estado de otro mapa si lo indicamos.

La etiqueta `%class` indica el nombre de la clase que albergará la máquina de estados y las funciones que intervengan en la máquina de estados. En el lenguaje C, que es el que utilicé en el proyecto, no existen los objetos, pero pueden simularse. Suponiendo que queremos crear una "clase" en C llamada pr, haríamos lo siguiente:

- Creamos un archivo pr.h. Este archivo alberga la definición de una estructura llamada pr (las variables de este tipo serían las instancias del objeto) y las declaraciones de las funciones que aplican sobre esa estructura (los métodos).
- Creamos un archivo pr.c. En este archivo guardamos las definiciones de las funciones que definimos en el archivo de cabecera.
- En SMC indicamos el nombre de la estructura en la etiqueta `%class`

La etiqueta `%header` indica a SMC el fichero de cabecera donde se implementan las funciones que podrán ser utilizadas por la máquina de estados de SMC para realizar acciones.

La etiqueta `%fsmclass` indica el nombre de la máquina de estados. En el código, la máquina de estados es una variable. El nombre del tipo de esa variable hace referencia a esta etiqueta.

A continuación, se indica que vamos a comenzar a describir un mapa, es decir, una máquina de estados. La etiqueta `%map` indica el nombre de ese mapa, y la etiqueta `%%` indica que lo que viene a continuación es la definición del mapa. Un mismo archivo puede tener varios mapas definidos, por lo que pueden encontrarse

varias etiquetas `%map` y `%%` en un mismo archivo.

```
E_0
Entry { e0_entry(); }
Exit {}
{
   TE_SPR              E_SPR               { }
   Default             nil                 { no_permitido(); }
}
```

El código de arriba define un estado dentro del mapa. Un mapa puede contener todos los estados que se necesiten, todos con ese formato.

En la primera línea, especificamos el nombre del estado para hacerle referencia posteriormente. Después nos encontramos con las acciones de entrada y de salida. En este caso el estado E_0 tiene una acción de entrada y ninguna acción de salida.

Después nos encontramos con las definiciones de las transiciones, una en cada línea. La primera palabra es el nombre de la transición, seguido del estado de destino, y seguido de una lista de acciones asociadas a esa transición entre los caracteres `{}`

En este caso nos encontramos con dos transiciones. La primera llamada TE_SPR que transiciona al estado E_SPR y no tiene acciones asociadas. La siguiente, Default, es una transición especial que no va a ningún estado (de ahí el `nil`) y tiene asociada una acción. Esta tipo de transiciones serán explicadas más adelante.

Respecto a las acciones, hay que tener en cuenta tres cosas:

1. Deben estar definidas en el archivo de cabecera especificado en la etiqueta `%header`
2. En ese archivo sus nombres deben estar prefijados con el valor de la etiqueta `%class` y un guión bajo
3. Las funciones que sirvan de acciones recibirán un parámetro que es el objeto sobre el que aplica la máquina de estados.

Por ejemplo, la función no_permitido() en el archivo pr_sm.h está definida de la siguiente manera:

```
/** @brief Imprime mensaje por pantalla para alertar al usuario de que
 *         no es posible realizar esa transicion en este estado
 *
 *  @param cliente Puntero al PR
 *  @returns Void
 */
void pr_no_permitido(struct pr *this);
```

En general, esto es todo lo que se necesita saber para definir una máquina de estados sencilla en SMC, aunque hay una serie de capacidades que SMC pone a disposición del programador para realizar máquinas de estados más complejas en cuanto a funcionalidad.

5.1.3 Reglas especiales

SMC cuenta con una serie de reglas concretas que nos permite diseñar máquina de estados más o menos complejas. Es importante tener estas reglas en cuenta a la hora de diseñar las máquinas de estado.

1. Las condiciones de las guardias se evalúan cuando se llama a la transición. SMC no se encarga de comprobar estas condiciones automáticamente. SMC está basado en eventos, por lo que la máquina de estados transicionará cuando el programador se lo indique.
2. Los estados pueden incluir transiciones por defecto (se identifican como aquellas transiciones cuyo

nombre es 'Default'). Las transiciones por defecto se ejecutan cuando se intenta llamar a una transición que no existe en el estado actual. Por ejemplo, si nos encontramos en el estado 'Funcionando' y se ejecuta la transición 'Iniciar', puesto que esta transición no se ha definido en el estado 'Funcionando', se ejecutaría la transición por defecto del estado 'Funcionando' en caso de que hubiera sido definida. Si no existe, se continuaría sin hacer nada.

3. Un estado puede tener varias transiciones con el mismo nombre, pero no con las mismas condiciones de guardia. Cuando se llama a una transición, se comienza por la que se define primero. Si tiene guardia y la condición es correcta, se realiza la transición. En caso contrario se pasa a la siguiente transición con el mismo nombre. Si se llega al caso de que no se tome la transición con ese nombre (bien porque no exista, o bien porque ninguna de las condiciones de las guardias son verdaderas) entonces se ejecutaría la transición por defecto, en caso de que existiera.
4. Existen dos transiciones especiales denominadas 'push' y 'pop'. Estas transiciones se indican modificando el campo de estado de destino de la transiciones. Al introducir 'push(estado de destino)' estaremos indicando una transición 'push', y escribiendo 'pop()' estaremos indicando que es una transición de tipo 'pop'. En las transiciones 'push', se especifica un estado de destino a donde la máquina de estados transicionará cuando se realice esa transición. A partir de ese momento la máquina de estados recordará en el estado en el que se encontraba cuando se realizó el 'push'. Si en algún momento se encuentra un estado 'pop' en una transición, se volverá al estado donde se realizó el 'push'. Esto funciona como una pila, de forma que es posible realizar varios 'push' sin realizar 'pop' entre ellos. Al realizar un 'pop', se volvería al estado donde se realizó el último 'push'.
5. El orden de llamada de acciones es el siguiente: Acción de transición, Acción de salida del estado origen, Acción de entrada del estado destino.

5.1.4 Usando SMC en una aplicación

Una vez se ha escrito el archivo que describe a la máquina de estado, es hora de usar SMC para crear el código que se usará en la aplicación y llamar a las funciones adecuadas. Es necesario tener instalado el JDK de java para poder usar SMC.

Primero, descargamos el compilador SMC de su página oficial, y ejecutamos el siguiente comando.

```
java –jar $SMC_HOME/bin/Smc.jar –c mi_maquina.sm
```

En este comando, `$SMC_HOME` es el directorio donde hayamos descomprimido SMC al descargarlo de su sitio web, el parámetro `–c` indica que queremos generar código C y `mi_maquina.sm` es el archivo que describe la máquina de estados que queremos implementar.

Esta aplicación generará dos archivos: pr_sm_smc.c y pr_sm_smc.h que deberán ser incluidos y compilados junto a la aplicación que hace uso de la máquina de estados.

Para los ejemplos que vienen a continuación, voy a suponer los siguientes valores para las etiquetas que se especifican en el archivo sm:

- %class claseEjemplo
- %fsmclass maquinaEstados

El resto de etiquetas no son importantes a la hora de operar sobre la máquina de estados.

Lo primero es iniciar la máquina de estados. Hay que tener en cuenta que SMC está pensado de forma que la máquina de estados se encuentra definida dentro de una clase (o un objeto). En C, la clase es representada por una estructura, de forma que para definirla haríamos algo como lo siguiente.

```
struct claseEjemplo {
  int campo_de_ejemplo;
  struct maquinaEstados maquina;
}
```

Una vez definida, procedemos a crear una variable cuyo tipo sea esa estructura.

```
struct claseEjemplo miObjeto;
```

Por último, iniciaríamos la máquina de estados con la función Init, declarada en el archivo de cabecera que creó SMC a partir de nuestro .sm

```
maquinaEstados_Init(&miObjeto.maquina, miObjeto);
```

Hecho esto, la máquina de estados esta lista para ser usada. En este momento se encontrará en el estado que hayamos especificado en la etiqueta `%start` del archivo .sm, y estará lista para recibir transiciones.

Hay que tener en cuenta que si utilizamos transiciones del tipo 'push' y 'pop' necesitamos crear una pila para guardar el estado de esas transiciones. Para ello, definimos un array del tipo `struct AppClassState` de tantos elementos como queramos, teniendo en cuenta que el número de elementos de este array es el número máximo de transiciones 'push' que podemos realizar antes de ejecutar una transición 'pop'. Una vez definido el array, ejecutamos la macro FSM_STACK para que SMC sea capaz de utilizarlo.

```
struct AppClassState * AppStack[10];
FSM_STACK(&miObjeto.maquina, AppStack);
```

Por último, solo queda saber cómo ejecutar transiciones sobre la máquina de estados. Para ello sólo debemos ejecutar lo siguiente, teniendo en cuenta que NombreTransicion debe ser una transición definida en el .sm

```
maquinaEstados_NombreTransicion(&miObjeto.maquina);
```

Si la transición se ejecuta correctamente, la máquina de estados cambiará al nuevo estado destino, y se ejecutarán las acciones oportunas, que han debido ser definidas e implementadas por el usuario en el archivo apropiado como ya se ha explicado anteriormente.

5.2 gSOAP

gSOAP [12] es una colección de herramientas de desarrollo en C y C++ para desarrollar servicios web con SOAP/XML. gSOAP se encarga de generar todo el código necesario para permitirnos enviar y recibir mensajes SOAP a partir de un archivo de cabecera (un archivo .h) que describe el servicio web. Además, incluye una herramienta que permite transformar un archivo WSDL en un archivo de cabecera, de forma que gSOAP pueda interpretarlo y generar el código adecuado.

A pesar de que gSOAP soporta C y C++, nosotros utilizamos únicamente el lenguaje C para generar la aplicación, así que se describe únicamente el proceso para generar código para este lenguaje, que es el siguiente:

1. Escribir un archivo de cabecera que describa el servicio (un archivo .h con los mensajes que intercambia el servicio web). También es posible generar este archivo automáticamente a partir de un WSDL con la herramienta wsdl2h proporcionada por gSOAP.
2. Ejecutar el ejecutable soapcpp2 para crear los archivos con las funciones que se deben llamar para intercambiar los mensajes SOAP.
3. Compilar los archivos creados en el segundo paso con los nuestros propios para generar el ejecutable final.

En esta sección se describe todo el proceso para ejecutar una aplicación con gSOAP, desde la instalación hasta cómo se realiza el envío y la recepción de mensajes.

5.2.1 Instalación

En primer lugar instalemos los paquetes necesarios para instalar gSOAP.

```
sudo apt-get install unzip make bison flex openssl libssl-dev automake1.10
```

Primero descargue la versión 2.8.17 de gSOAP [13] y descomprima el archivo (`unzip gsoap_2.8.17.zip`)

lo que creará una nueva carpeta llamada gsoap-2.8 que contiene todos los archivos necesarios para instalar gSOAP.

Sitúese dentro de la carpeta gsoap-2.8 y ejecute los siguientes comandos para compilar e instalar gSOAP en el sistema.

```
cd gsoap-2.8
./configure
make
sudo make install
```

Por ultimo copie el archivo stdsoap2.h de la carpeta gsoap en /usr/local/include, ya que ese archivo es necesario posteriormente para compilar código que utilice gSOAP.

```
cd gsoap
sudo cp stdsoap2.h /usr/local/include
```

5.2.2 Generación de archivo de descripción del servicio

Pese a que es posible escribir un archivo de descripción del servicio de forma manual, en nuestro caso generamos este archivo a partir de los archivos WSDL que describen el protocolo OCPP y los protocolos propios de la arquitectura.

Estos archivos se pueden encontrar en la página oficial de la OCA. En el caso de OCPP hay dos archivos WSDL, uno para el CIR (el archivo centralSystem) y otro para el PR (el archivo chargePoint).

Para generar el archivo de cabecera de descripción del servicio se utiliza la utilidad wsdl2h que se encuentra en el directorio soap/bin, de la siguiente forma:

```
cd bin/linux386
./wsdl2h -c -o header.h CentralSystemService.wsdl
```

Esto generará un archivo header.h que describe el servicio web del CIR. La opción -c especifica que el código final será C, y la opción –o permite especificar el nombre del archivo de salida.

5.2.3 Generación de archivos gSOAP

Una vez que tenemos el archivo de cabecera que describe el servicio, solo falta generar los archivos C que contienen las tipos de datos y las funciones que utilizaremos para enviar mensajes gSOAP.

Para generar estos archivos se utiliza la utilidad soapcpp2 que se encuentra en el directorio soap/bin, de la siguiente forma:

```
./soapcpp2 -T -c -x -L header.h
```

El parámetro -T se utiliza para generar un archivo de auto-test, para probar los archivos generados por gSOAP, el parámetro -c se utiliza para especificar que se genere código C, la opción -x es para evitar que se generen archivos .xml de ejemplo para todos los mensajes que se pueden intercambiar, y la opción -L para evitar generar dos archivos más que no utilizaremos.

Se generarán varios archivos, que son los siguientes:

```
CentralSystemServiceSoap.nsmap soapC.c soapClient.c soapH.h soapServer.c
soapStub.h soapTester.c
```

Para poder usar las funciones de gSOAP solo hay que incluir el archivo CentralSystemServiceSoap.nsmap y compilar las fuentes junto a los archivos soapC.c, soapClient.c y/o soapServer.c (dependiendo de si está implementando un cliente, un servidor, o ambos).

El archivo soapTester.c es un archivo ya preparado para compilarse junto a los demás que sirve para probar los archivos generados y también de base para nuevos proyectos.

A la hora de compilar, es importante recordar que es necesario pasarle a gcc los parámetros "-L/usr/local/lib -lgsoap" para linkear correctamente las librerías de gSOAP.

5.2.4 Implementación servidor

Para crear un servidor SOAP es necesario seguir los siguientes pasos.

1. Crear la variable del contexto SOAP e inicializarla

```
struct soap soap_server;
soap_init(&soap_server);
soap_server.accept_timeout = 1;
soap_server.bind_flags |= SO_REUSEADDR;
```

El campo accept_timeout especifica el tiempo de timeout de una petición (es decir, una petición se considerara errónea si no se completa en 1 segundo en este caso). SO_REUSEADDR es una flag que nos permite reutilizar el mismo host y puerto si ya han sido utilizados por soap previamente.

2. Hacer que SOAP escuche en un determinado puerto usando la función bind, especificándole el host y el puerto donde debe escuchar las peticiones. Backlog es la cantidad de peticiones máxima que pueden estar en espera en un determinado momento.

```
m = soap_bind(&soap_server, HOST, PUERTO, BACKLOG);
if (!soap_valid_socket(m)) {
  printf("[ERROR] Couldnt bind server.\n");
  exit(1);
}
```

3. Usar soap_accept para esperar a una petición SOAP. Cuando la petición sea recibida, y siempre que sea válida, se utilizará soap_serve para servir esa petición, y soap_destroy junto a soap_end para liberar los recursos utilizados en esa petición.

```
while (SERVER_RUNNING) {
  if (soap_valid_socket(soap_accept(&soap_server))) {
    soap serve(&soap server);
    soap_destroy(&soap_server);
    soap_end(&soap_server);
  }
}
```

4. Por último, solo queda especificar qué debe ejecutarse cuando se ejecuta cada uno de los mensajes SOAP que el servicio web implemente. Para ello se deben implementar funciones que tienen este aspecto:

```
int __ns1__mensaje(struct soap *soap, struct ns1__mensajeRequest
*ns1__mensajeRequest, struct ns1__mensajeResponse *ns1__mensajeResponse) {
 // Hacer lo apropiado
 return SOAP_OK;
 // o bien return SOAP_ERR;
}
```

En estas funciones, los parámetros del mensaje que el servidor recibe en la petición se encuentran en la estructura ns1__mensajeRequest. Para aplicar parámetros al mensaje de respuesta, solo se debe rellenar la estructura ns1__mensajeResponse. Estas estructuras están definidas en el archivo soapStub.h, así que se puede utilizar para comprobar los campos de cada una de ellas.

5.2.5 Implementación cliente

Para enviar mensajes a un servidor soap solo es necesario crear la variable de contexto SOAP, reservar memoria para la petición y para la respuesta, y utilizar las funciones que soap proporciona para enviar el mensaje.

El método sería el siguiente:

1. Crear la variable del contexto SOAP e inicializarla

```
struct soap soap_client;
soap_init(&soap_client);
```

2. Reservar memoria para la petición y para la respuesta

```
struct ns1__MensajeRequest request;
struct ns1__MensajeResponse response;
```

3. Enviar el mensaje

```
if (soap_call___ns1__Mensaje(&soap_client, SERVER_URL, NULL, &request,
&response) == SOAP_OK) {
 // Mensaje recibido correctamente
} else {
 // Error en la peticion
}
```

Donde debemos tener definida una constante SERVER_URL que identifique al servidor con una URL HTTP (por ejemplo http://192.168.0.2:8080).

6 CONCLUSIONES

Durante la realización de este proyecto, he mejorado ampliamente mis conocimientos tanto en programación en C, como en el desarrollo de sistemas distribuidos y de servicios web. A partir de estos y de lo realizado en el trabajo, he llegado a las siguientes conclusiones:

1. Realizar servicios web es una tarea relativamente sencilla y fácil para permitir la comunicación entre distintos procesos o aplicaciones. Herramientas como gSOAP y WSDL permiten implementar estos servicios web de forma rápida, una vez que el programador se acostumbra a ese marco de trabajo.
2. Las máquinas de estados pueden ser una herramienta útil a la hora de implementar una aplicación, pero no tienen por qué ser completamente necesarias en todos los casos.
3. La documentación es uno de los productos más importantes a la hora de desarrollar un proyecto de software. Es una de las partes que el cliente más valora en una entrega final.
4. A la hora de desarrollar software, es importante dejar vías abiertas para modificaciones en el futuro. En el desarrollo de este proyecto hemos dejado todo preparado para implementar nuevas maniobras o hacer los cambios oportunos en el software en un futuro.

6.1 Vías de futuro

Respecto al sistema actual, creo que hay varias vías de desarrollo abiertas para trabajar sobre ellas. Entre estas, destaco las siguientes:

6.1.1 Mejor interoperabilidad e implementación del protocolo OCPP

Aunque el proyecto actual utiliza el protocolo OCPP y hace un uso intensivo de los mensajes que el protocolo define, no se ajusta del todo a la especificación del protocolo OCPP. Aún hay operaciones del protocolo que no están implementadas, como el mecanismo de autorización, el uso de un protocolo de cifrado y autenticación para las conexiones a los servicios web (HTTPS) o el uso de un servidor FTP para realizar las operaciones de actualización de firmware y demás.

6.1.2 Mejoras en la seguridad de los protocolos SOAP

Actualmente, los protocolos que se han desarrollado para comunicar CIR, EU y SI realizan ordenes como programación de maniobras, peticiones de maniobras o recogida de estado del sistema sin realizar ningún tipo de comprobación en cuanto a la identificación de la entidad que realiza tales peticiones. En un futuro, el proyecto debería realizar algún tipo de identificación del usuario antes de dar acceso a estos servicios. Ya sea por usuario y contraseña, por una contraseña precompartida, el uso de certificados, etc.

6.1.3 Sistema de alarmas

Uno de los planes futuros es crear un sistema de alarmas hacia el SI. Este sistema haría que cuando un PR se encontrase con alguna situación especial, informara al CIR, y este informara al SI. Estas situaciones podrían ser, desde averías, hasta situaciones de emergencia (cuando, por ejemplo, un usuario pulsa un interruptor de emergencia).

6.1.4 Más maniobras

Aún faltan maniobras por implementar. Entre ellas, están previstas maniobras de recogida de estado del PR y del CIR, así como las maniobras de actualización de software y actualización de configuración del CIR.

6.1.5 Modo de retransmisión

Una de las funcionalidades que han sido previstas pero aún no implementadas es un modo de retransmisión para el CIR. Para activar este modo, el SI indicaría con un nuevo mensaje que quiere activar el modo de retransmisión para un cierto PR. A partir de ese momento el CIR reenviaría los mensajes OCPP recibidos por ese PR al SI y viceversa, de forma que el SI puede comunicarse directamente con un PR.

6.1.6 Flotas como baterías móviles auxiliares

En un futuro sería ideal realizar un sistema que permitiera desde el SI ordenar una operación inversa a la de carga, es decir, que un vehículo eléctrico con la batería cargada que estuviera conectado a un poste de recarga pudiera ingresar esa potencia eléctrica a la red.

Este caso sería especialmente interesante para empresas energéticas, ya que podrían utilizar estas infraestructuras de recarga para suplir las necesidades eléctricas de la red, aprovechando la batería de vehículos eléctricos que en ese momento no necesitasen estar en funcionamiento.

Más tarde, cuando no hubiera necesidad eléctrica en la red, esos coches podrían volver a ser cargados mediante una maniobra de carga.

REFERENCIAS

[1] «Gloval EV Outlook,» 2013. [En línea]. Available: http://www.iea.org/topics/transport/electricvehiclesinitiative/EVI_GEO_2013_FullReport.pdf.

[2] «Electric Vehicle Initiative,» [En línea]. Available: http://www.iea.org/topics/transport/subtopics/electricvehiclesinitiative/.

[3] «Plugshare,» [En línea]. Available: http://www.plugshare.com/.

[4] «Open Charge Alliance,» [En línea]. Available: http://www.openchargealliance.org/.

[5] «Open Charge Point Protocol,» [En línea]. Available: http://www.openchargealliance.org/?q=node/13.

[6] «Open Charge Point Protocol 1.5 Specification,» [En línea]. Available: http://www.openchargealliance.org/sites/default/files/ocpp_specification_1.5_final.pdf.

[7] «Especificación SOAP 1.2,» [En línea]. Available: http://www.w3.org/TR/soap/.

[8] «Applying WS-Security on SOAP,» [En línea]. Available: http://www.soapui.org/SOAP-and-WSDL/applying-ws-security.html.

[9] W. 1. Specification. [En línea]. Available: http://www.w3.org/TR/wsdl.

[10] «Elaad fundation,» [En línea]. Available: http://www.elaad.nl/.

[11] «SMC Website,» [En línea]. Available: http://smc.sourceforge.net/.

[12] «gSOAP Website,» [En línea]. Available: http://www.cs.fsu.edu/~engelen/soap.html.

[13] «gSOAP 2.8.17,» [En línea]. Available: http://sourceforge.net/projects/gsoap2/files/latest/download?source=files.

Efleet CIR

0.1

Generado por Doxygen 1.8.6

Jueves, 11 de Septiembre de 2014 16:24:18

Índice general

Capítulo 1

Índice de estructura de datos

1.1. Estructura de datos

Lista de estructuras con una breve descripción:

Capítulo 2

Indice de archivos

2.1. Lista de archivos

Lista de todos los archivos documentados y con descripciones breves:

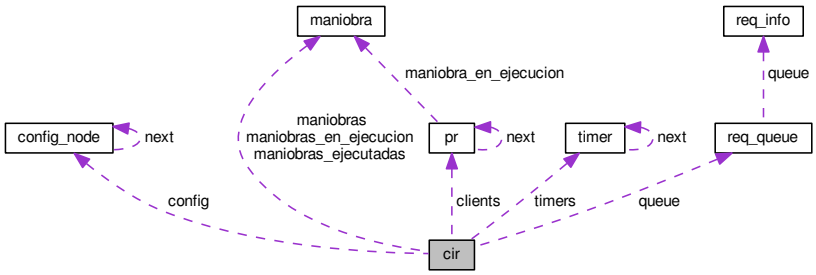
Capítulo 3

Documentación de las estructuras de datos

3.1. Referencia de la Estructura cir

`#include <cir.h>`

Diagrama de colaboración para cir:



Campos de datos

- struct pr ∗ clients
- struct timer ∗ timers
- struct req_queue queue
- struct soap soap_client
- struct soap soap_si_client
- struct config_node ∗ config
- struct maniobra ∗ maniobras
- struct maniobra ∗ maniobras_ejecutadas
- struct maniobra ∗ maniobras_en_ejecucion
- pthread_mutex_t mutex_maniobras
- pthread_t tid_input
- pthread_t tid_server_pr
- pthread_t tid_server_si
- pthread_t tid_server_eu
- pthread_t tid_timers
- int heartbeat_interval
- int running

3.1.1. Descripción detallada

Estructura principal del CIR

3.1.2. Documentación de los campos

3.1.2.1. struct pr* cir::clients

Lista de PRs conectados

3.1.2.2. struct config_node* cir::config

Archivo de configuracion

3.1.2.3. int cir::heartbeat_interval

Intervalo de heartbeat (Tiempo de Keepalive)

3.1.2.4. struct maniobra* cir::maniobras

Lista de maniobras en espera

3.1.2.5. struct maniobra* cir::maniobras_ejecutadas

Lista de maniobras ejecutadas

3.1.2.6. struct maniobra* cir::maniobras_en_ejecucion

Lista de maniobras en ejecucion

3.1.2.7. pthread_mutex_t cir::mutex_maniobras

Mutex para las listas de maniobras

3.1.2.8. struct req_queue cir::queue

Cola de peticiones por atender

3.1.2.9. int cir::running

Bandera que indica el estado del CIR

3.1.2.10. struct soap cir::soap_client

Variable SOAP del cliente para los PR

3.1.2.11. struct soap cir::soap_si_client

Variable SOAP del cliente para el SI

3.1.2.12. pthread_t cir::tid_input

ID del hilo de input

3.1.2.13. pthread_t cir::tid_server_eu

ID del hilo de servidor del EU

3.1.2.14. pthread_t cir::tid_server_pr

ID del hilo de servidor de OCPP

3.1.2.15. pthread_t cir::tid_server_si

ID del hilo de servidor del SI

3.1.2.16. pthread_t cir::tid_timers

ID del hilo de timers

3.1.2.17. struct timer* cir::timers

Timers a ejecutar

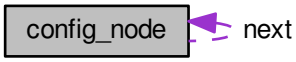
La documentación para esta estructura fue generada a partir del siguiente fichero:

- cir.h

3.2. Referencia de la Estructura config_node

`#include <utils.h>`

Diagrama de colaboración para config_node:



Campos de datos

- char key [20]
- char value [64]
- struct config_node * next

3.2.1. Descripción detallada

Almacena cada propiedad del archivo de configuracion

3.2.2. Documentación de los campos

3.2.2.1. char config_node::key[20]

Nombre de la propiedad

3.2.2.2. struct config_node* config_node::next

Siguiente propiedad en la lista

3.2.2.3. char config_node::value[64]

Valor de la propiedad

La documentación para esta estructura fue generada a partir del siguiente fichero:

- utils.h

3.3. Referencia de la Estructura maniobra

```
#include <manoeuvres.h>
```

Campos de datos

- int id_maniobra
- int id_ir
- int id_pr
- int id_ve
- char * optional
- int resultado
- int estado
- time_t tiempo
- time_t tiempo_fin
- UT_hash_handle hh

3.3.1. Descripción detallada

Maniobra a ejecutar (o ya ejecutada) por el CIR

3.3.2. Documentación de los campos

3.3.2.1. int maniobra::estado

Estado de la maniobra, 0: no ejecutada, 1, ejecutandose, 2, ejecutada

3.3.2.2. UT_hash_handle maniobra::hh

Hace la estructura Hasheable por la lib uthash.h

3.3.2.3. int maniobra::id_ir

Identificador infraestrucutura de recarga

3.3.2.4. int maniobra::id_maniobra

Clave o ID de la maniobra

3.3.2.5. int maniobra::id_pr

Poste donde se tiene que ejecutar la maniobra

3.3.2.6. int maniobra::id_ve

Identificador vehículo eléctrico

3.3.2.7. char* maniobra::optional

Campo opcional de las maniobras

3.3.2.8. int maniobra::resultado

Resultado de la maniobra, 0 ejecutada correctamente, -1 fallo en la ejecución

3.3.2.9. time_t maniobra::tiempo

Tiempo en el que se ejecuta la maniobra

3.3.2.10. time_t maniobra::tiempo_fin

Tiempo en el que acabo la maniobra

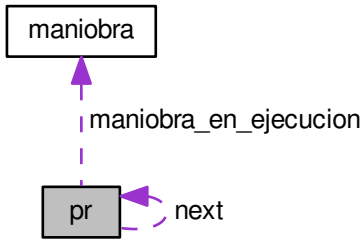
La documentación para esta estructura fue generada a partir del siguiente fichero:

- manoeuvres.h

3.4. Referencia de la Estructura pr

```
#include <pr.h>
```

Diagrama de colaboración para pr:



Campos de datos

- int id
- char * host
- char * url
- int port
- time_t last_time
- struct soap * soap
- struct pr_sm_smc stateMachine
- struct maniobra * maniobra_en_ejecucion
- struct pr * next

3.4.1. Descripción detallada

Estructura que almacena a un PR conectado

3.4.2. Documentación de los campos

3.4.2.1. char* pr::host

Host del PR

3.4.2.2. int pr::id

ID del PR

3.4.2.3. time_t pr::last_time

Tiempo en el que se recibio el ultimo mensaje de este PR

3.4.2.4. struct maniobra* pr::maniobra_en_ejecucion

Maniobra ejecutandose en este momento

3.4.2.5. struct pr* pr::next

Siguiente PR en la lista de PRs conectados

3.4.2.6. int pr::port

Puerto del PR

3.4.2.7. struct soap* pr::soap

Variable SOAP para contactar con este PR

3.4.2.8. struct pr_sm_smc pr::stateMachine

Maquina de estados de este PR

3.4.2.9. char* pr::url

URL del PR en formato `http://host:port`

La documentación para esta estructura fue generada a partir del siguiente fichero:

- pr.h

3.5. Referencia de la Estructura req_info

```
#include <utils.h>
```

Campos de datos

- SOAP_SOCKET socket
- char host [1024]
- int port

3.5.1. Descripción detallada

Informacion de una peticion

3.5.2. Documentación de los campos

3.5.2.1. char req_info::host[1024]

Host del emisor

3.5.2.2. int req_info::port

Puerto del emisor

3.5.2.3. SOAP_SOCKET req_info::socket

Socket al emisor de la peticion

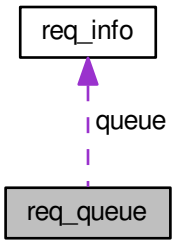
La documentación para esta estructura fue generada a partir del siguiente fichero:

- utils.h

3.6. Referencia de la Estructura req_queue

`#include <utils.h>`

Diagrama de colaboración para req_queue:



Campos de datos

- struct req_info * queue [MAX_QUEUE]
- pthread_mutex_t lock
- pthread_cond_t cond
- int current
- int next

3.6.1. Descripción detallada

Cola de peticiones

3.6.2. Documentación de los campos

3.6.2.1. pthread_cond_t req_queue::cond

Condicional para avisos

3.6.2.2. int req_queue::current

Posicion en la cola de la ultima peticion agregada

3.6.2.3. pthread_mutex_t req_queue::lock

Mutex para operaciones

3.6.2.4. int req_queue::next

Posicion en la cola donde debe ir la siguiente peticion a agregar

3.6.2.5. struct req_info* req_queue::queue[MAX_QUEUE]

Peticiones

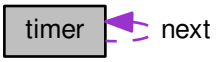
La documentación para esta estructura fue generada a partir del siguiente fichero:

- utils.h

3.7. Referencia de la Estructura timer

`#include <utils.h>`

Diagrama de colaboración para timer:



Campos de datos

- void(* funct)(void *)
- int timestamp
- void * param
- struct timer * next

3.7.1. Descripción detallada

Estructura que almacena la cola de funciones a ejecutar por el timer

3.7.2. Documentación de los campos

3.7.2.1. void(* timer::funct)(void *)

Funcion a ejecutar

3.7.2.2. struct timer* timer::next

Siguiente funcion a ejecutar

3.7.2.3. void* timer::param

Parametro opcional para la funcion

3.7.2.4. int timer::timestamp

Tiempo cuando se debe ejecutar la funcion

La documentación para esta estructura fue generada a partir del siguiente fichero:

- utils.h

Capítulo 4

Documentación de archivos

4.1. Referencia del Archivo cir.h

Prototipos de las funciones principales del CIR.

```
#include <pthread.h>
#include "utils.h"
#include "manoeuvres.h"
#include "pr.h"
#include "pr_sm.h"
#include "pr_msg.h"
#include "pr_manoeuvre.h"
#include "chargeH.h"
#include "centralH.h"
#include "efleetcirsiH.h"
#include "efleetcireuH.h"
#include "efleetsiH.h"
```

Dependencia gráfica adjunta para cir.h:

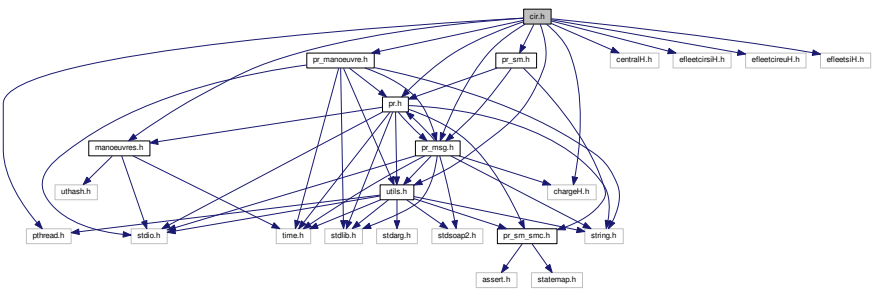
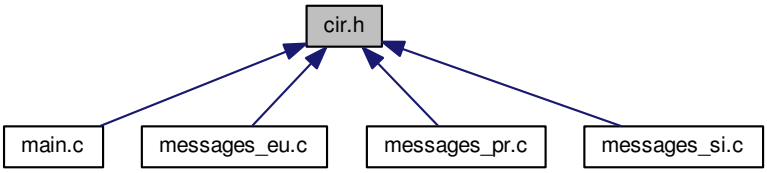


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Estructuras de datos

- struct cir

Funciones

- void cir_init (struct cir *this)

  Inicia la estructura CIR (Constructor)
- void cir_destroy (struct cir *this)

  Libera los recursos utilizados por el CIR (Destructor)
- void cir_server_pr (struct cir *this)

  Crea el servidor SOAP para atender peticiones OCPP.
- void cir_server_eu (struct cir *this)

  Crea el servidor SOAP para atender peticiones del EU.
- void cir_server_si (struct cir *this)

  Crea el servidor SOAP para atender peticiones del SI.
- void cir_timers (struct cir *this)

  Crea el hilo que ejecuta las funciones de timers.
- void cir_process_queue (struct thread_info *info)

  Funcion que ejecutan los hilos que atienden las peticiones del servidor SOAP.
- void cir_input (struct cir *this)

  Lee la entrada por el teclado y ejecuta la accion apropiada en cada caso.
- void cir_finish (struct cir *this)

  Finaliza el CIR.
- void cir_broadcast_remote_charge (struct cir *this)

  Provoca una operacion de carga en todos los PRs.
- void cir_check_times (struct cir *this)

  Elimina los PRs cuando pasan demasiado tiempo inactivos.
- void cir_execute_manoeuvres (struct cir *this)

  Ejecuta las maniobras guardadas cuando se cumple el tiempo de ejecucion de esa maniobra.
- void cir_check_finalized_manoeuvres (struct cir *this)

  Comprueba si han finalizado alguna de las maniobras y las mueve de lista.
- int cir_send_petition (struct cir *this, char *manId, char *prId, time_t comienzo, char *options)

  Envia una peticion de maniobra al SI. Estas peticiones vienen de los mensajes recibidos desde el EU.

4.1.1. Descripción detallada

Prototipos de las funciones principales del CIR. Contiene los prototipos de todas las funciones del objeto CIR. Todas aceptan un parametro que es la esctructura del CIR sobre el que se esta actuando, aunque en toda la aplicacion solo deberia haber una de estas estructuras inicializada.

Autor

Carlos Rodríguez (CarlosRdrz)

4.1.2. Documentación de las funciones

4.1.2.1. void cir_broadcast_remote_charge (struct cir * *this*)

Provoca una operacion de carga en todos los PRs.

Crea un nuevo thread para cada uno de los PRs y lanza la maquina de estados del PR para realizar la maniobra de carga

Parámetros

this	la estructura del cir

```
329 {
330   struct pr *current = this->clients;
331
332   while (current != NULL) {
333     // Llamamos a la maniobra de carga de ese PR
334     pr_charge_manoeuvre(current, &this->timers);
335     // Pasamos al siguiente PR
336     current = current->next;
337     // Dormimos 1 segundo
338     usleep(200);
339   }
340 }
```

Gráfico de llamadas para esta función:

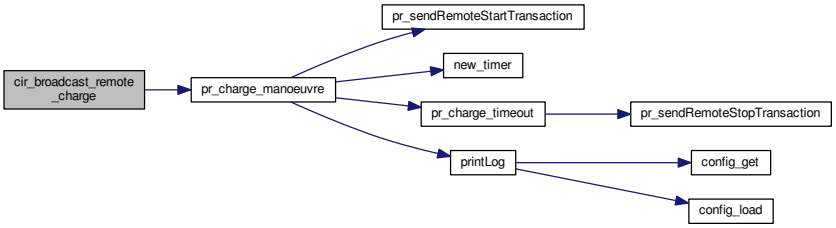
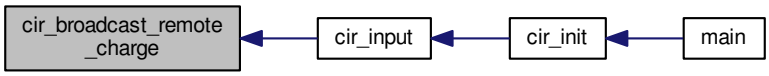


Gráfico de llamadas a esta función:



4.1.2.2. void cir_check_finalized_manoeuvres (struct cir * *this*)

Comprueba si han finalizado alguna de las maniobras y las mueve de lista.

Parámetros

this	la estructura del cir

```
456 {
457   // Bloqueamos el mutex para evitar operaciones en las listas
458   // mientras se realiza este proceso
459   pthread_mutex_lock(&this->mutex_maniobras);
460
461   struct maniobra * current = this->maniobras_en_ejecucion;
462
463   if (current != NULL && current->estado == 2) {
464     printf("[INFO] Finalizada maniobra ID %d.\n", current->id_maniobra);
465     maniobra_move(&this->maniobras_en_ejecucion, &this->maniobras_ejecutadas, current);
466   }
467
468   // Desbloqueamos el mutex
469   pthread_mutex_unlock(&this->mutex_maniobras);
470 }
```

Gráfico de llamadas para esta función:

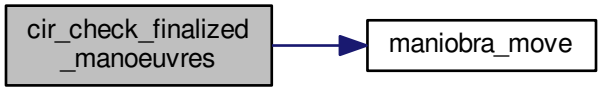
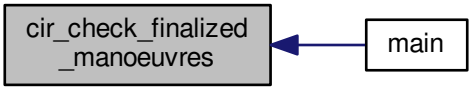


Gráfico de llamadas a esta función:



4.1.2.3. void cir_check_times (struct cir * *this*)

Elimina los PRs cuando pasan demasiado tiempo inactivos.

Esta funcion se ejecuta en el hilo principal del CIR. Se encarga de recorrer la lista de PRs conectados y comprobar que cada PR ha enviado almenos un mensaje en el tiempo adecuado (especificado por la opcion heartbeat_interval del archivo de configuracion)

Parámetros

this	la estructura del cir

```
343 {
344   struct pr *current = this->clients;
345   struct pr *next;
346   time_t time_now = time(NULL);
347
```

```
348   // Recorremos la lista de PRs conectados
349   while (current != NULL) {
350     next = current->next;
351
352     // Si la hora actual menos la hora en la que se envio el ultimo mensaje de ese PR
353     // es mayor que el tiempo especificado en heartbeat_interval, entonces eliminamos
354     // a ese PR de la lista de PRs conectados
355     if (difftime(time_now, current->last_time) > this->
      heartbeat_interval) {
356       printLog("[INFO] Removing client %s for inactivity\n", current->
      url);
357       delete_pr(current, &this->clients);
358     }
359
360     current = next;
361   }
362 }
```

Gráfico de llamadas para esta función:

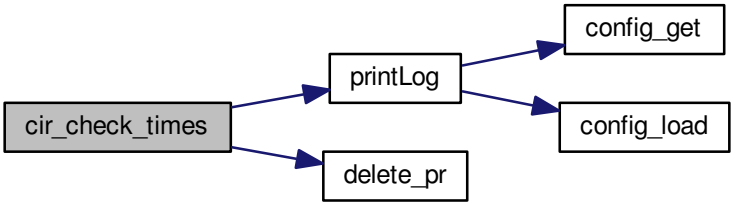
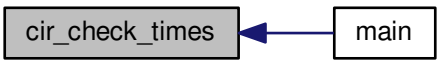


Gráfico de llamadas a esta función:



4.1.2.4. void cir_destroy (struct cir * *this*)

Libera los recursos utilizados por el CIR (Destructor)

Parámetros

this	la estructura del cir

```
35 {
36   pthread_join(this->tid_server_pr, NULL);    // Espera a que termine los hilos
37   pthread_join(this->tid_server_si, NULL);
38   pthread_join(this->tid_server_eu, NULL);
39   pthread_join(this->tid_input, NULL);
40   pthread_join(this->tid_timers, NULL);
41   req_queue_destroy(&this->queue);         // Destruye la cola de peticiones
42   free_prs(this->clients);                  // Destruye lista de PRs conectados
43   config_destroy(this->config);             // Destruye recursos de archivo de config
44   soap_done(&this->soap_client);
```

```
45   soap_done(&this->soap_si_client);
46 }
```

Gráfico de llamadas para esta función:

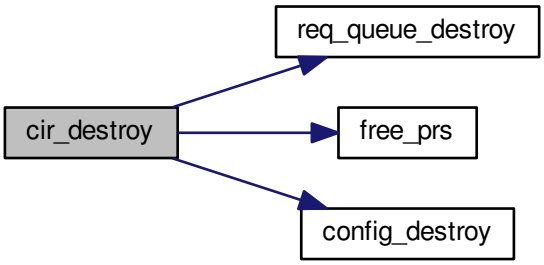
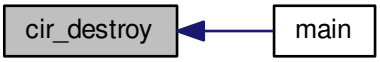


Gráfico de llamadas a esta función:



4.1.2.5. void cir_execute_manoeuvres (struct cir * *this*)

Ejecuta las maniobras guardadas cuando se cumple el tiempo de ejecucion de esa maniobra.

Parámetros

this	la estructura del cir

```
365 {
366   int n_maniobras;
367   struct maniobra *s;
368 
369   // Bloqueamos el mutex para evitar operaciones en las listas
370   // mientras se realiza este proceso
371   pthread_mutex_lock(&this->mutex_maniobras);
372   n_maniobras = HASH_COUNT(this->maniobras);
373 
374   // Si hay maniobras por ejecutar...
375   if (n_maniobras > 0) {
376     s = this->maniobras; // Primera maniobra en el hash
377 
378     // Si el tiempo de ejecucion de esa maniobra es menor que el tiempo actual, la ejecutamos.
379     if (s->tiempo <= time(NULL)) {
380       struct pr *pr = find_pr_by_id(s->id_pr, this->
     clients);
381       if (pr != NULL) {
382         printLog("[INFO] Executing manoeuvre -- ID: %d -- ID PR: %d\n", s->
     id_maniobra, s->id_pr);
383 
384         if (s->id_maniobra == 1) {
```

```
385           printLog("[INFO] Charge Maniouvre on PR ID %d\n", s->id_pr);
386           pr->maniobra_en_ejecucion = s;
387           pr_charge_manoeuvre(pr, &this->timers);
388           // Colocamos en tiempo_fin la hora en la que se ejecuto la maniobra (no es la hora en la que
      finaliza)
389           s->tiempo_fin = time(NULL);
390           // Movemos la maniobra de la lista de maniobras en espera a la lista de maniobras en ejecucion
391           maniobra_move(&this->maniobras, &this->maniobras_en_ejecucion, s);
392         } else if (s->id_maniobra == 2) {
393           // Maniobra de aborto desde el SI a una maniobra de carga
394           printLog("[INFO] Aborting charge manoeuvre on PR %d.\n", s->
      id_pr);
395           pr_abort_manoeuvre(pr);
396           s->tiempo_fin = time(NULL);
397           maniobra_move(&this->maniobras, &this->maniobras_ejecutadas, s);
398         } else if (s->id_maniobra == 3) {
399           // Maniobra de pausa de carga
400           printLog("[INFO] Pause Maniouvre on PR ID %d\n", s->id_pr);
401           pr_pause_manoeuvre(pr, s->optional, &this->
      timers);
402           s->tiempo_fin = time(NULL);
403           maniobra_move(&this->maniobras, &this->maniobras_ejecutadas, s);
404         } else if (s->id_maniobra == 6) {
405           // Maniobra de reinicio de PR
406           printLog("[INFO] Restart Maniouvre on PR ID %d\n", s->id_pr);
407           struct prState * currentState = getState(&pr->stateMachine);
408           if (currentState == &MapaPrincipal_E_EFC || currentState == &MapaPrincipal_E_SPR) {
409             printLog("[ERROR] Couldn't restart PR %d because is paused or charging in this moment.
      \n", s->id_pr);
410           } else {
411             pr_sendRestartDataTransfer(pr);
412             printLog("[INFO] Removing PR with ID %d because is about to restart.\n", pr->
      id);
413             delete_pr(pr, &this->clients);
414           }
415           s->tiempo_fin = time(NULL);
416           maniobra_move(&this->maniobras, &this->maniobras_ejecutadas, s);
417         } else if (s->id_maniobra == 7) {
418           // Maniobra de actualizacion de firmware de PR
419           if (s->optional != NULL) {
420             pr_sendUpdateFirmware(pr, s->optional);
421             pr_sendRestartDataTransfer(pr);
422             printLog("[INFO] Update Maniouvre on PR ID %d\n", s->id_pr);
423             printLog("[INFO] Removing PR with ID %d because is about to restart.\n", pr->
      id);
424             delete_pr(pr, &this->clients);
425             maniobra_move(&this->maniobras, &this->maniobras_ejecutadas, s);
426           } else {
427             printLog("[ERROR] Firmware update manoeuvre received, but no URL specified. Removed
      from the list.\n");
428             maniobra_delete(&this->maniobras, s);
429           }
430         } else if (s->id_maniobra == 8) {
431           // Maniobra de cambio de configuracion
432           if (s->optional != NULL) {
433             printLog("[INFO] Change Config Maniouvre on PR ID %d\n", s->
      id_pr);
434             pr_sendChangeConfiguration(pr, s->optional);
435             maniobra_move(&this->maniobras, &this->maniobras_ejecutadas, s);
436           } else {
437             printLog("[ERROR] Config change manoeuvre received, but no parameter specified. Removed
      from the list.\n");
438             maniobra_delete(&this->maniobras, s);
439           }
440         } else {
441           printLog("[ERROR] Manoeuvre ID %d not recognizable. Removed from the list.\n", s->
      id_maniobra);
442           maniobra_delete(&this->maniobras, s);
443         }
444       } else {
445         printLog("[ERROR] Couldn't find PR with ID %d to execute manoeuvre with ID %d. Removed from
      the list.\n", s->id_pr, s->id_maniobra);
446         maniobra_delete(&this->maniobras, s);
447       }
448     }
449   }
450
451   // Desbloqueamos el mutex
452   pthread_mutex_unlock(&this->mutex_maniobras);
453 }
```

Gráfico de llamadas para esta función:

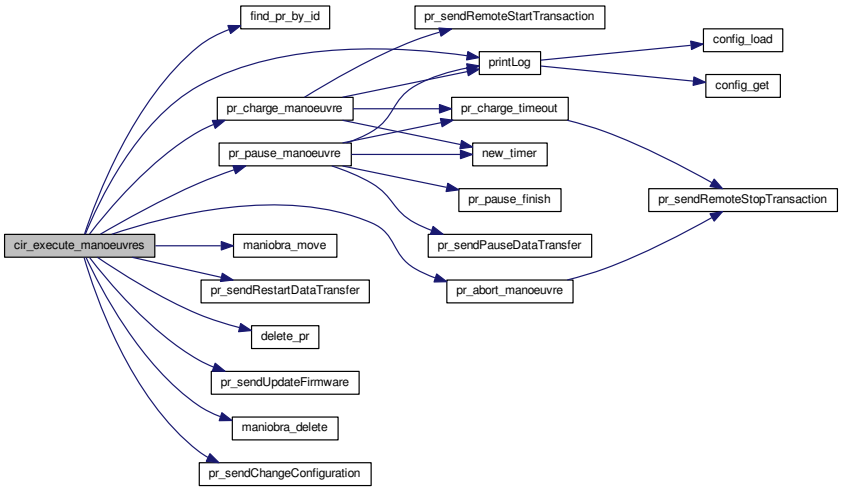


Gráfico de llamadas a esta función:

4.1.2.6. void cir_finish (struct cir * *this*)

Finaliza el CIR.

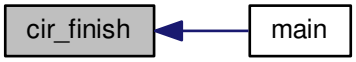
Coloca la bandera running en 0

Parámetros

this	la estructura del cir

```
324 {
325   this->running = 0;
326 }
```

Gráfico de llamadas a esta función:



4.1.2.7. void cir_init (struct cir * *this*)

Inicia la estructura CIR (Constructor)

Inicia las variables y estructuras necesarias para el CIR, como las instancias de SOAP o las banderas e inicia los hilos del teclado y del servidor SOAP.

Parámetros

this	la estructura del cir

```
9 {
10   soap_init(&this->soap_client);
11   soap_set_namespaces(&this->soap_client, charge_namespaces);
12
13   soap_init(&this->soap_si_client);
14   soap_set_namespaces(&this->soap_si_client, efleetsi_namespaces);
15
16   this->clients = NULL; // Inicia la lista de PRs conectados
17   this->running = 1;
18   this->config = config_load("config"); // Cargar archivo de configuracion
19   this->heartbeat_interval = atoi(config_get(this->config, "HEARTBEAT_INTERVAL")); // Lee
       HEARTBEAT_INTERVAL del archivo de config
20
21   this->maniobras = NULL;
22   this->maniobras_ejecutadas = NULL;
23   this->maniobras_en_ejecucion = NULL;
24   pthread_mutex_init(&this->mutex_maniobras, NULL);
25
26   // Crea los hilos para leer del teclado y servir las peticiones soap
27   pthread_create(&this->tid_input, NULL, (void*(*)(void*))cir_input, (void *)this);
28   pthread_create(&this->tid_server_pr, NULL, (void*(*)(void*))cir_server_pr, (void *)this);
29   pthread_create(&this->tid_server_si, NULL, (void*(*)(void*))cir_server_si, (void *)this);
30   pthread_create(&this->tid_server_eu, NULL, (void*(*)(void*))cir_server_eu, (void *)this);
31   pthread_create(&this->tid_timers, NULL, (void*(*)(void*))cir_timers, (void *)this);
32 }
```

Gráfico de llamadas para esta función:

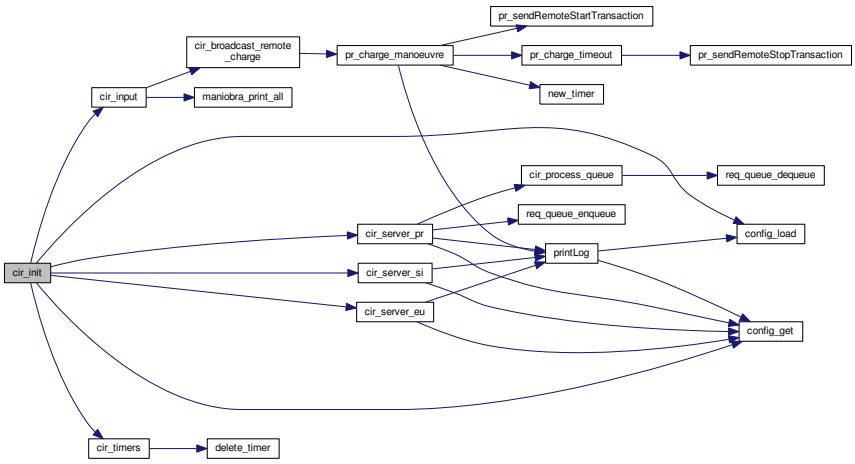


Gráfico de llamadas a esta función:

4.1.2.8. void cir_input (struct cir * *this*)

Lee la entrada por el teclado y ejecuta la accion apropiada en cada caso.

Parámetros

this	la estructura del cir

```
282 {
283   int pressed; // La tecla que se ha pulsado
284 
285   printf("[INFO] Press 'R' key to execute a remote charge operation on every client\n");
286   printf("[INFO] Press 'p' key to display the waiting manoeuvres list\n");
287   printf("[INFO] Press 'x' key to display the executing manoeuvres list\n");
288   printf("[INFO] Press 'e' key to display the executed manoeuvres list\n");
289 
290   // Mientras el CIR este funcionando...
291   while(this->running == 1) {
292     pressed = getchar(); // Leer letra del teclado
293     switch(pressed) {
294       case 'R':
295         // Provoca el comienzo de la maquina de estados de carga en todos los PR
296         cir_broadcast_remote_charge(this);
297         break;
298       case 'p':
299         printf("Waiting manoeuvres: \n");
300         pthread_mutex_lock(&this->mutex_maniobras);
301         // Imprimir lista de maniobras en espera
302         maniobra_print_all(this->maniobras);
303         pthread_mutex_unlock(&this->mutex_maniobras);
304         break;
```

```
305       case 'x':
306         printf("Executing manoeuvres: \n");
307         pthread_mutex_lock(&this->mutex_maniobras);
308         // Imprimir lista de maniobras en espera
309         maniobra_print_all(this->maniobras_en_ejecucion);
310         pthread_mutex_unlock(&this->mutex_maniobras);
311         break;
312       case 'e':
313         printf("Executed manoeuvres: \n");
314         pthread_mutex_lock(&this->mutex_maniobras);
315         // Imprimir lista de maniobras ejecutadas
316         maniobra_print_all(this->maniobras_ejecutadas);
317         pthread_mutex_unlock(&this->mutex_maniobras);
318         break;
319     }
320   }
321 }
```

Gráfico de llamadas para esta función:

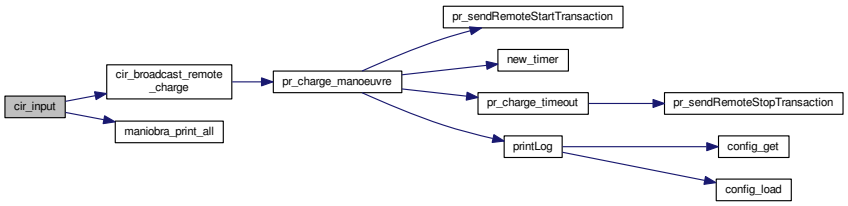
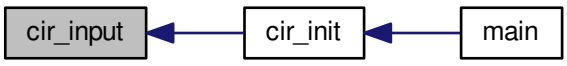


Gráfico de llamadas a esta función:



4.1.2.9. void cir_process_queue (struct thread_info * *info*)

Funcion que ejecutan los hilos que atienden las peticiones del servidor SOAP.

Parámetros

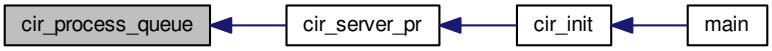
this	la estructura del cir

```
143 {
144   // Obtenemos la variable SOAP asociada a este hilo a partir de la informacion
145   // que se le pasa al hilo a la hora de crearlo
146   struct soap *tsoap = (struct soap*)info->instance;
147
148   // Mientras el CIR este funcionando...
149   while (info->this->running) {
150     // Desencolamos una peticion de la cola de peticiones
151     struct req_info * req_info = req_queue_dequeue(&info->this->queue);
152
153     // Si hemos desencolado correctamente...
154     if (req_info != NULL) {
155       // Copiamos el socket, el host y el port en la estructura soap de este hilo
156       // Esto es necesario para que los hilos sepan quien es el emisor de la peticion SOAP
157       tsoap->socket = req_info->socket;
158       strncpy(tsoap->host, req_info->host, 1024);
```

```
159       tsoap->port = req_info->port;
160       free(req_info);
161
162       // Si el socket es incorrecto por cualquier motivo, no servimos la peticion
163       if (!soap_valid_socket(tsoap->socket)) continue;
164
165       // En caso contrario, servimos la peticion y liberamos recursos
166       // printf("[INFO] Thread %d accepts socket connection from %s:%d\n", info->thread_id, tsoap->host,
      tsoap->port);
167       central_serve(tsoap);
168       soap_destroy(tsoap);
169       soap_end(tsoap);
170     }
171   }
172
173   // Antes de salir liberamos la memoria utilizada que se creo en la funcion cir_server_pr
174   free(info);
175 }
```

Gráfico de llamadas para esta función:

Gráfico de llamadas a esta función:



4.1.2.10. int cir_send_petition (struct cir * *this*, char * *manId*, char * *prId*, time_t *comienzo*, char * *options*)

Envia una peticion de maniobra al SI. Estas peticiones vienen de los mensajes recibidos desde el EU.

Parámetros

this	la estructura del cir
manId	cadena con el id de la maniobra a pedir
prId	cadena con el id del pr donde ejecutar la maniobra
comienzo	momento en el que se debe ejecutar la maniobra
options	cadena con los parametros opcionales

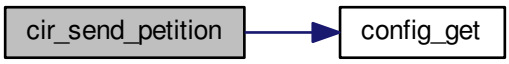
Devuelve

SOAP_OK o SOAP_ERR

```
473 {
474   char * siloc = config_get(this->config, "SILOCATION");
475   struct efleetsi__PeticionManiobraRequest request;
476   struct efleetsi__PeticionManiobraResponse response;
477   time_t actual = time(NULL);
478   int res;
```

```
479
480   request.idMan = manId;
481   request.irDes = NULL;
482   request.prDes = prId;
483   request.veDes = NULL;
484   request.comienzo = comienzo;
485   request.opcional = options;
486   request.PotMax = NULL;
487   request.ConMax = NULL;
488
489   res = soap_call___efleetsi__PeticionManiobra(&this->soap_si_client, siloc, NULL, &request, &response);
490
491   soap_destroy(&this->soap_si_client);
492   soap_end(&this->soap_si_client);
493
494   return res;
495 }
```

Gráfico de llamadas para esta función:



4.1.2.11. void cir_server_eu (struct cir * *this*)

Crea el servidor SOAP para atender peticiones del EU.

No levanta hilos. Procesa las peticiones según van llegando.

Parámetros

this	la estructura del cir

```
217                                     {
218   struct soap soap_server_eu;
219   soap_init(&soap_server_eu);
220   soap_set_namespaces(&soap_server_eu, efleetcireu_namespaces);
221   soap_server_eu.accept_timeout = 1;
222   soap_server_eu.bind_flags |= SO_REUSEADDR;
223   soap_server_eu.user = (void *)this;
224
225   int listen_port = atoi(config_get(this->config, "EU_LISTEN_PORT"));
226   char * listen_ip = config_get(this->config, "EU_LISTEN_IP");
227
228   int m, s;
229   m = soap_bind(&soap_server_eu, listen_ip, listen_port, BACKLOG);
230   if (!soap_valid_socket(m)) {
231     printf("[ERROR] Couldnt bind EU server.\n");
232     exit(1);
233   }
234
235   printLog("[INFO] Started EU server on %s:%d\n", listen_ip, listen_port);
236
237   while (this->running == 1) {
238     s = soap_accept(&soap_server_eu);
239     if (!soap_valid_socket(s)) {
240       if (soap_server_eu.errnum) {
241         soap_print_fault(&soap_server_eu, stdout);
242         continue; // Error
243       } else {
244         continue; // Server timed out
245       }
246     }
247
248     efleetcireu_serve(&soap_server_eu);
249     soap_destroy(&soap_server_eu);
```

```
250     soap_end(&soap_server_eu);
251   }
252
253   soap_done(&soap_server_eu);
254   printLog("[INFO] EU server finished\n");
255 }
```

Gráfico de llamadas para esta función:

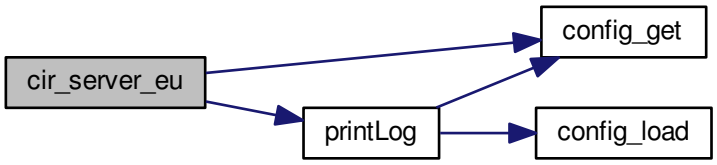
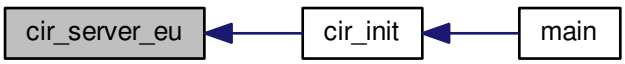


Gráfico de llamadas a esta función:



4.1.2.12. void cir_server_pr (struct cir * *this*)

Crea el servidor SOAP para atender peticiones OCPP.

Levanta los hilos que atenderan las peticiones que lleguen al servidor. Introduce las peticiones que llegan a ese servidor en una cola de peticiones. Los hilos que atienden las peticiones leeran las peticiones de esa cola y las serviran una a una.

Parámetros

this	la estructura del cir

```
49 {
50   // Variable SOAP del servidor
51   struct soap soap_server_pr;
52   // Variables SOAP de los hilos que atienden peticiones SOAP
53   struct soap *soap_thr[MAX_THR];
54   // IDs de los hilos que atienden peticiones
55   pthread_t req_tid[MAX_THR];
56   // Leer PR_LISTEN_PORT y PR_LISTEN_IP del archivo de config
57   int listen_port = atoi(config_get(this->config, "PR_LISTEN_PORT"));
58   char * listen_ip = config_get(this->config, "PR_LISTEN_IP");
59   // Otras variables
60   SOAP_SOCKET m;
61   int i;
62
63   // Iniciar instancia soap
64   soap_init(&soap_server_pr);
65   soap_set_namespaces(&soap_server_pr, central_namespaces);
66   soap_server_pr.accept_timeout = 1;
```

```
67   soap_server_pr.bind_flags |= SO_REUSEADDR;
68   soap_server_pr.user = (void *)this; // Permite usar la variable this en el archivo messages.c
69
70   // Hace que el servidor SOAP escuche en el puerto especificado en la propiedad
71   // LISTEN_PORT del archivo de configuracion
72   m = soap_bind(&soap_server_pr, listen_ip, listen_port, BACKLOG);
73   if (!soap_valid_socket(m)) {
74     printf("[ERROR] Couldnt bind server.\n");
75     exit(1);
76   }
77
78   // Crea los hilos que atenderan las peticiones que lleguen al servidor SOAP
79   // El numero de hilos esta definido en la constante MAX_THR
80   for (i = 0; i < MAX_THR; i++) {
81     soap_thr[i] = soap_copy(&soap_server_pr);
82
83     // Cuando cada uno de los hilos es creado, recibe una estructura que tiene estos campos
84     struct thread_info *thread_info = (struct thread_info *)malloc(sizeof(struct thread_info));
85     thread_info->this = this;             // Un puntero a la estructura cir
86     thread_info->instance = soap_thr[i];  // Un puntero a su propia variable SOAP
87     thread_info->thread_id = i;           // Un identificador del hilo
88
89     pthread_create(&req_tid[i], NULL, (void*(*)(void*))cir_process_queue, (void*)
    thread_info);
90   }
91
92   printLog("[INFO] Started PR server on %s:%d with %d threads\n", listen_ip, listen_port, MAX_THR);
93
94   // Mientras el CIR este ejecutandose...
95   while (this->running == 1) {
96     // Creo una estructura req_info, y le asigno lo siguiente
97     struct req_info * req_info = (struct req_info *)malloc(sizeof(struct req_info));
98     // El socket hacia el cliente que ha hecho una peticion. Tenga en cuenta que la funcion soap_accept
99     // permanece a la espera hasta que recibe una peticion desde un cliente.
100    req_info->socket = soap_accept(&soap_server_pr);
101    strncpy(req_info->host, soap_server_pr.host, 1024); // El host del cliente
102    req_info->port = soap_server_pr.port;               // El puerto del cliente
103
104    // Si por cualquier motivo el socket no es valido (por timeout, o error en la comunicacion)...
105    // Entonces continuamos y no encolamos la peticion (y por lo tanto no sera atendida)
106    if (!soap_valid_socket(req_info->socket)) {
107      if (soap_server_pr.errnum) {
108        soap_print_fault(&soap_server_pr, stdout);
109        continue; // Error
110      } else {
111        continue; // Server timed out
112      }
113    }
114
115    // En caso contrario, encolamos la peticion en la cola de peticiones
116    while (req_queue_enqueue(&this->queue, req_info) == SOAP_EOM) sleep(1);
117  }
118
119  // Llegados a este punto el CIR estara cerrandose (ya que ha salido del while), por lo que
120  // enviamos una estructura req_info especial que hace que los hilos que atienden las peticiones finalicen
121  for (i = 0; i < MAX_THR; i++) {
122    struct req_info * invalid_req_info = (struct req_info *)malloc(sizeof(struct req_info));
123    invalid_req_info->socket = SOAP_INVALID_SOCKET;
124    strncpy(invalid_req_info->host, "", 1024);
125    invalid_req_info->port = 0;
126
127    while (req_queue_enqueue(&this->queue, invalid_req_info) == SOAP_EOM) sleep(1);
128  }
129
130  // Por ultimo esperamos a que los hilos finalicen, cerramos las variables SOAP que utilizaban los hilos
131  // que atendian las peticiones y liberamos la memoria utilizada
132  for (i = 0; i < MAX_THR; i++) {
133    pthread_join(req_tid[i], NULL);
134    soap_done(soap_thr[i]);
135    free(soap_thr[i]);
136  }
137
138  soap_done(&soap_server_pr);
139  printLog("[INFO] PR server finished\n");
140 }
```

Gráfico de llamadas para esta función:

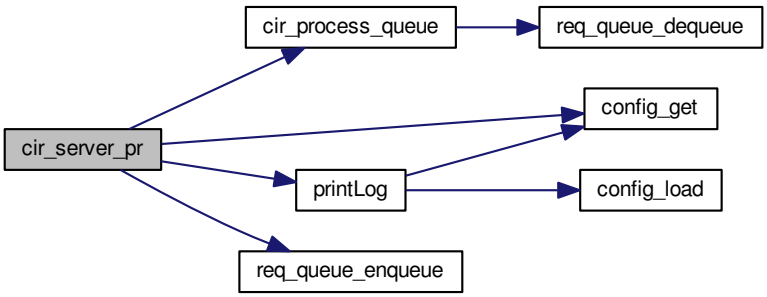
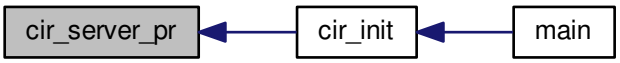


Gráfico de llamadas a esta función:



4.1.2.13. void cir_server_si (struct cir * *this*)

Crea el servidor SOAP para atender peticiones del SI.

No levanta hilos. Procesa las peticiones según van llegando.

Parámetros

this	la estructura del cir

```
177                                     {
178   struct soap soap_server_si;
179   soap_init(&soap_server_si);
180   soap_set_namespaces(&soap_server_si, efleetcirsi_namespaces);
181   soap_server_si.accept_timeout = 1;
182   soap_server_si.bind_flags |= SO_REUSEADDR;
183   soap_server_si.user = (void *)this;
184
185   int listen_port = atoi(config_get(this->config, "SI_LISTEN_PORT"));
186   char * listen_ip = config_get(this->config, "SI_LISTEN_IP");
187
188   int m, s;
189   m = soap_bind(&soap_server_si, listen_ip, listen_port, BACKLOG);
190   if (!soap_valid_socket(m)) {
191     printf("[ERROR] Couldnt bind SI server.\n");
192     exit(1);
193   }
194
195   printLog("[INFO] Started SI server on %s:%d\n", listen_ip, listen_port);
196
197   while (this->running == 1) {
198     s = soap_accept(&soap_server_si);
199     if (!soap_valid_socket(s)) {
200       if (soap_server_si.errnum) {
201         soap_print_fault(&soap_server_si, stdout);
```

```
202         continue; // Error
203       } else {
204         continue; // Server timed out
205       }
206     }
207
208     efleetcirsi_serve(&soap_server_si);
209     soap_destroy(&soap_server_si);
210     soap_end(&soap_server_si);
211   }
212
213   soap_done(&soap_server_si);
214   printLog("[INFO] SI server finished\n");
215 }
```

Gráfico de llamadas para esta función:

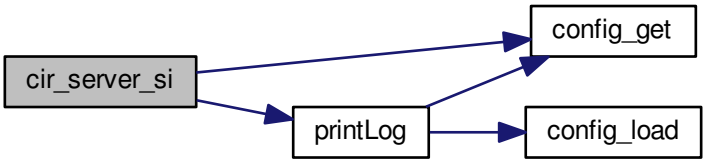


Gráfico de llamadas a esta función:

4.1.2.14. void cir_timers (struct cir * *this*)

Crea el hilo que ejecuta las funciones de timers.

Este hilo se encarga de recorrer la lista de timers y comprobar la hora en la que debe ejecutarse cada funcion. Si la hora se cumple la funcion se ejecuta y el timer se elimina de la lista.

Parámetros

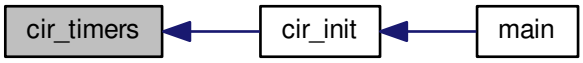
this	la estructura del cir

```
258 {
259   // Mientras el CIR este funcionando...
260   while (this->running == 1) {
261     struct timer *current = this->timers;
262
263     while (current != NULL) {
264       int timenow = time(NULL);
265
266       // Si el timestamp se pasa ejecutamos la funcion
267       if (timenow > current->timestamp) {
268         current->funct(current->param);
```

```
269         delete_timer(&this->timers, current);
270       }
271
272       // Pasamos al siguiente timer
273       current = current->next;
274     }
275
276     // Dormimos 0.25 s
277     usleep(250);
278   }
279 }
```

Gráfico de llamadas para esta función:

Gráfico de llamadas a esta función:

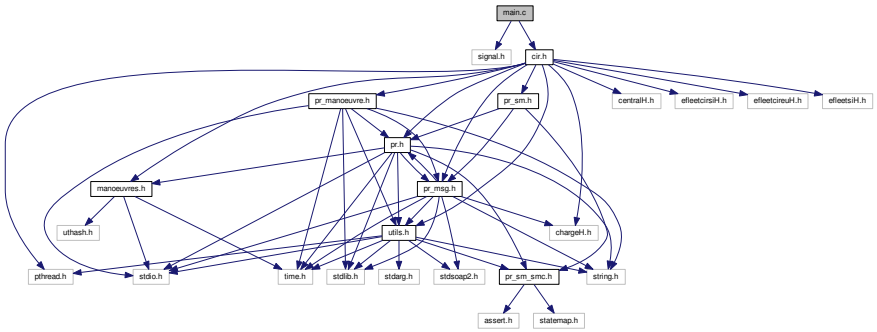


4.2. Referencia del Archivo main.c

Archivo principal del CIR.

```
#include <signal.h>
#include "cir.h"
```

Dependencia gráfica adjunta para main.c:



Funciones

- void signalsHandler (int signal)

  Maneja la señal que se produce al pulsar CTRL+C (SIGINT). Coloca la bandera sigint en 1 para realizar una salida ordenada.
- int main (int argc, char ∗argv[])

  Funcion principal del CIR.

Variables

- int sigint = 0

4.2.1. Descripción detallada

Archivo principal del CIR.

Autor

Carlos Rodríguez (CarlosRdrz)

4.2.2. Documentación de las funciones

4.2.2.1. int main (int *argc*, char ∗ *argv[ ]*)

Funcion principal del CIR.

Parámetros

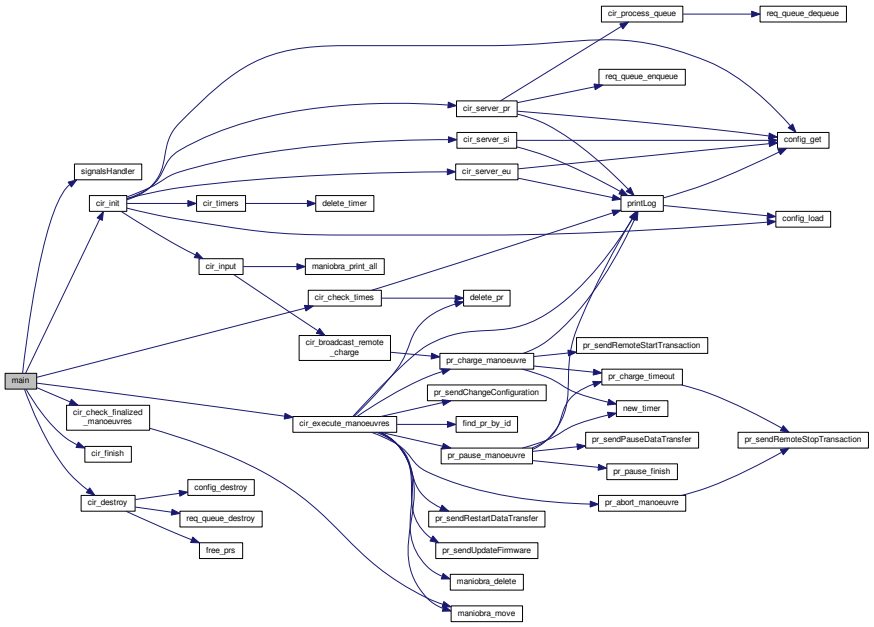
argc	Numero de parametros introducidos en la linea de comandos
argv	Array de parametros introducidos en la linea de comandos

Devuelve

int Indica si la salida fue exitosa

```
32                                {
33   // Estructura que almacena todas las variables del CIR
34   struct cir cir;
35
36   // Hace que se ejecute la funcion signalsHandler cuando se pulsa
37   // la combinacion de teclas CTRL+C (señal SIGINT)
38   signal(SIGINT, signalsHandler);
39
40   // Inicia el proceso CIR
41   cir_init(&cir);
42
43   // Mientras el CIR este funcionando...
44   while(cir.running == 1) {
45     // Comprobamos si los PRs enviaron un mensaje en los ultimos
46     // X segundos, y si no los damos de baja.
47     cir_check_times(&cir);
48     // Ejecutamos las maniobras recibidas si ha pasado su tiempo
49     // de ejecucion
50     cir_execute_manoeuvres(&cir);
51     // Comprobamos si las maniobras en ejecucion han finalizado
52     cir_check_finalized_manoeuvres(&cir);
53     // Si se pulso CTRL+C, comenzamos el proceso de salida del CIR
54     if (sigint) cir_finish(&cir);
55     // Dormimos 100 ms para no cargar la CPU
56     usleep(250);
57   }
58
59   // Liberamos los recursos usados en el CIR
60   cir_destroy(&cir);
61
62   return 0;
63 }
```

Gráfico de llamadas para esta función:



4.2.2.2. void signalsHandler (int *signal*)

Maneja la señal que se produce al pulsar CTRL+C (SIGINT). Coloca la bandera sigint en 1 para realizar una salida ordenada.

Parámetros

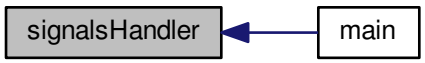
signal	Entero que indica el tipo de señal

Devuelve

void

```
19                                {
20   if (signal == SIGINT) {
21     printf(" -> Received SIGINT. Exiting...\n");
22     sigint = 1;
23   }
24 }
```

Gráfico de llamadas a esta función:



4.2.3. Documentación de las variables

4.2.3.1. int sigint = 0

Bandera. Se coloca en 1 cuando se pulsa CTRL+C

4.3. Referencia del Archivo manoeuvres.h

Prototipos de las funciones de maniobras.

```
#include <stdio.h>
#include <time.h>
#include "uthash.h"
```

Dependencia gráfica adjunta para manoeuvres.h:

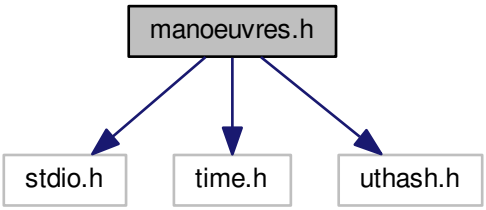
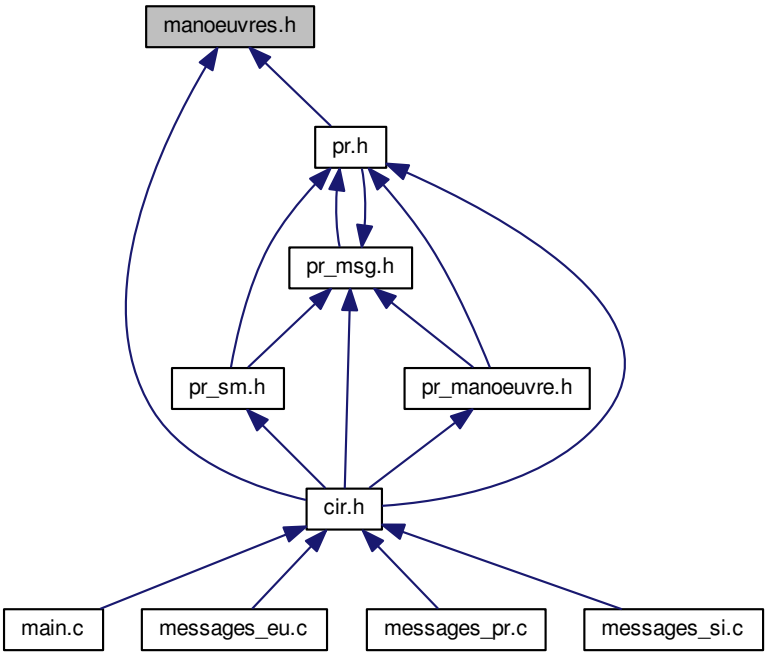


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Estructuras de datos

- struct maniobra

Funciones

- void maniobra_add (struct maniobra **lista, struct maniobra *to_add)

  Agrega una maniobra a la lista.
- void maniobra_delete (struct maniobra **lista, struct maniobra *to_del)

  Elimina una maniobra a la lista.
- void maniobra_move (struct maniobra **source, struct maniobra **dest, struct maniobra *to_move)

  Mueve una maniobra de una lista a la otra.
- void maniobra_sort (struct maniobra **lista)

  Ordena la lista de maniobras en funcion del tiempo en el que se debe ejecutar la maniobra.
- void maniobra_print_all (struct maniobra *lista)

  Imprime todas las maniobras de la lista.
- struct maniobra * maniobra_parse (char *string)

  Convierte una cadena de texto en una estructura maniobra.
- int maniobra_time_sort (struct maniobra *a, struct maniobra *b)

  Compara una maniobra a con una maniobra b en funcion de su tiempo de ejecucion.
- void maniobra_serialize_list (struct maniobra *list, int id, char *where, int size)

  *Serializa una lista de maniobras a una cadena de caracteres donde * separa los campos de las maniobras y separa las diferentes maniobras.*

4.3.1. Descripción detallada

Prototipos de las funciones de maniobras. Estas funciones se utilizan para realizar operaciones sobre listas de maniobras, como agregar o eliminar maniobras o imprimir todas las maniobras de una lista.

Autor

Carlos Rodríguez (CarlosRdrz)

4.3.2. Documentación de las funciones

4.3.2.1. void maniobra_add (struct maniobra ** *lista*, struct maniobra * *to_add*)

Agrega una maniobra a la lista.

Parámetros

lista	puntero a la lista de maniobras a ordenar
to_add	puntero a la maniobra a agregar

Devuelve

void

```
4 {
5   HASH_ADD_INT(*lista, id_maniobra, to_add);
6 }
```

4.3.2.2. void maniobra_delete (struct maniobra ** *lista*, struct maniobra * *to_del*)

Elimina una maniobra a la lista.

Parámetros

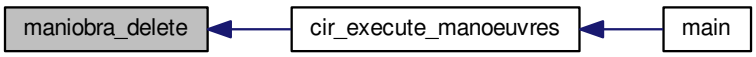
lista	puntero a la lista de maniobras a ordenar
to_add	puntero a la maniobra a eliminar

Devuelve

void

```
9 {
10   HASH_DEL(*lista, to_del);
11   // Libera la memoria de la maniobra
12   if (to_del->optional != NULL) free(to_del->optional);
13   free(to_del);
14 }
```

Gráfico de llamadas a esta función:



4.3.2.3. void maniobra_move (struct maniobra ** *source*, struct maniobra ** *dest*, struct maniobra * *to_move*)

Mueve una maniobra de una lista a la otra.

Parámetros

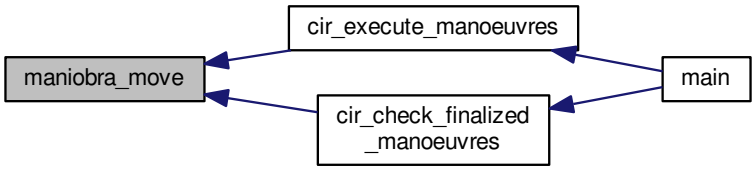
source	puntero a la lista de maniobras origen
dest	puntero a la lista de maniobras destino
to_move	puntero a la maniobra a mover

Devuelve

void

```
17 {
18   HASH_DEL(*source, to_move);
19   HASH_ADD_INT(*dest, id_maniobra, to_move);
20 }
```

Gráfico de llamadas a esta función:



4.3.2.4. struct maniobra* maniobra_parse (char * *string*)

Convierte una cadena de texto en una estructura maniobra.

Parámetros

string	cadena a parsear

Devuelve

puntero a la estructura creada

```
41 {
42   char *token;
43   int count = 0;
44 
45   // Reservamos memoria para la maniobra a devolver
46   // Recuerda que hay que liberar esta memoria en alguna parte posteriormente
47   struct maniobra *maniobra = (struct maniobra *)malloc(sizeof(struct maniobra));
48   maniobra->tiempo_fin = -1;
49   maniobra->optional = NULL;
50 
51   // Busca el primer delimitador en la cadena
52   token = strtok(string, MANIOBRA_DELIMITER);
53 
54   // Segun en que delimitador estemos, habremos obtenido una u otra informacion
55   while( token != NULL ) {
56     switch (count) {
57       case 0:
58         maniobra->id_maniobra = atoi(token);
59         break;
60       case 1:
61         maniobra->id_ir = atoi(token);
62         break;
63       case 2:
64         maniobra->id_pr = atoi(token);
65         break;
66       case 3:
67         maniobra->id_ve = atoi(token);
68         break;
69       case 4:
70         maniobra->tiempo = (time_t)atoi(token);
71         break;
72       case 5:
73         maniobra->optional = (char *)malloc(1024);
74         strncpy(maniobra->optional, token, 1023);
75       default:
76         break;
77     }
78 
79     count++;
80     token = strtok(NULL, MANIOBRA_DELIMITER);
81   }
82 
83   return maniobra;
84 }
```

4.3.2.5. void maniobra_print_all (struct maniobra * *lista*)

Imprime todas las maniobras de la lista.

Parámetros

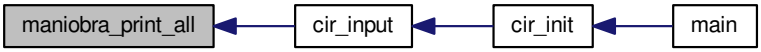
lista	puntero a la lista de maniobras

Devuelve

void

```
28 {
29   struct maniobra *s;
30 
31   if(HASH_COUNT(lista) > 0) {
32     for(s = lista; s != NULL; s = s->hh.next) {
33       printf("[] ID Maniobra %d, ID IR %d, ID PR %d, ID VE %d, Tiempo programado %s", s->
      id_maniobra, s->id_ir, s->id_pr,s->id_ve, ctime(&(s->
      tiempo)));
34     }
35   } else {
36     printf("Empty list\n");
37   }
38 }
```

Gráfico de llamadas a esta función:



4.3.2.6. void maniobra_serialize_list (struct maniobra * *list*, int *id*, char * *where*, int *size*)

Serializa una lista de maniobras a una cadena de caracteres donde * separa los campos de las maniobras y separa las diferentes maniobras.

Parámetros

list	primera maniobra de la lista a serializar
id	-1 si tiene en cuenta todas las maniobras o distinto de -1 para tener en cuenta solo las maniobras con ese ID
where	donde se debe serializar la lista
size	tamano del buffer where

Devuelve

void

```
100 {
101   struct maniobra * current;
102   char toappend[32];
103   char aux[16];
104   int first = 0;
105 
106   for(current = list; current != NULL; current = current->hh.next) {
107     if (id == -1 || current->id_maniobra == id) {
108       toappend[0] = '\0';
109       if (first != 0) strcat(toappend, "\n");
110       else first = 1;
111 
112       sprintf(aux, "%d", current->id_maniobra);
113       strcat(toappend, aux);
114       strcat(toappend, "*");
115       sprintf(aux, "%d", current->id_ir);
116       strcat(toappend, aux);
117       strcat(toappend, "*");
118       sprintf(aux, "%d", current->id_pr);
119       strcat(toappend, aux);
120       strcat(toappend, "*");
121       sprintf(aux, "%d", current->id_ve);
122       strcat(toappend, aux);
123       strcat(toappend, "*");
124       sprintf(aux, "%d", (int) current->tiempo);
125       strcat(toappend, aux);
126 
127       if (current->optional != NULL) {
128         strcat(toappend, "*");
129         strcat(toappend, current->optional);
130       }
131 
132       if (strlen(where) + strlen(toappend) < size) {
133         strcat(where, toappend);
134       }
135     }
136   }
137 }
```

4.3.2.7. void maniobra_sort (struct maniobra ** *lista*)

Ordena la lista de maniobras en funcion del tiempo en el que se debe ejecutar la maniobra.

Parámetros

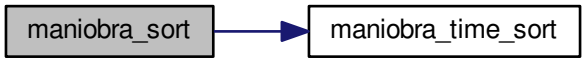
lista	puntero a la lista de maniobras a ordenar

Devuelve

void

```
23 {
24   HASH_SORT(*lista, maniobra_time_sort);
25 }
```

Gráfico de llamadas para esta función:



4.3.2.8. int maniobra_time_sort (struct maniobra * a, struct maniobra * b)

Compara una maniobra a con una maniobra b en funcion de su tiempo de ejecucion.

Parámetros

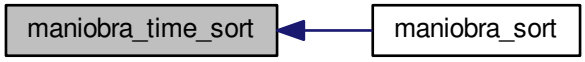
a	puntero a maniobra primer operando
b	puntero a maniobra segundo operando

Devuelve

0 si son iguales -1 si a->tiempo < b->tiempo 1 si b->tiempo > a->tiempo

```
87 {
88   if (a->tiempo < b->tiempo) {
89     return -1;
90   }
91   if (a->tiempo == b->tiempo) {
92     return 0;
93   }
94   else {
95     return 1;
96   }
97 }
```

Gráfico de llamadas a esta función:

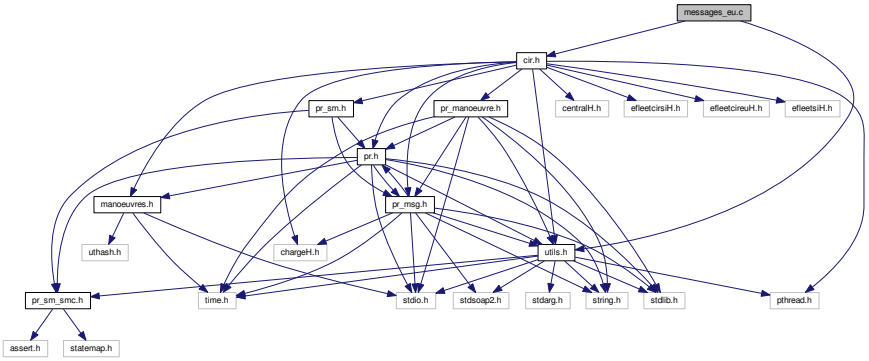


4.4. Referencia del Archivo messages_eu.c

Funciones de mensajes que se intercambian con el EU.

```
#include "cir.h"
#include "utils.h"
```

Dependencia gráfica adjunta para messages_eu.c:



4.4.1. Descripción detallada

Funciones de mensajes que se intercambian con el EU. Estas funciones se ejecutan cada vez que se recibe un mensaje desde el EU con el nombre de esa funcion.

Autor

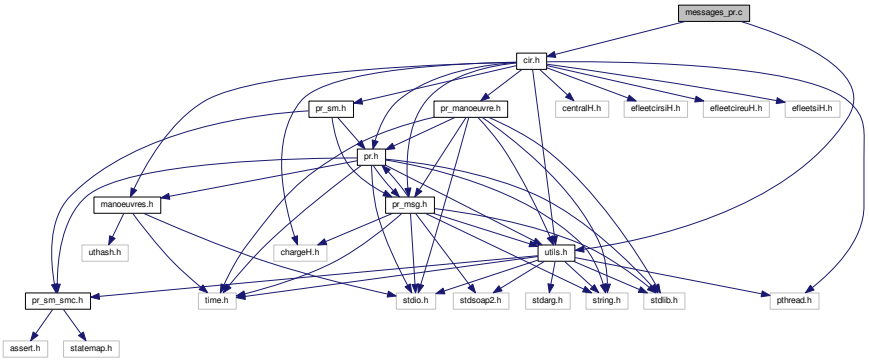
Carlos Rodríguez (CarlosRdrz)

4.5. Referencia del Archivo messages_pr.c

Funciones de mensajes OCPP.

```
#include "cir.h"
#include "utils.h"
```

Dependencia gráfica adjunta para messages_pr.c:



4.5.1. Descripción detallada

Funciones de mensajes OCPP. Estas funciones se ejecutan cada vez que se recibe un mensaje OCPP de un PR con el nombre de esa funcion.

Autor

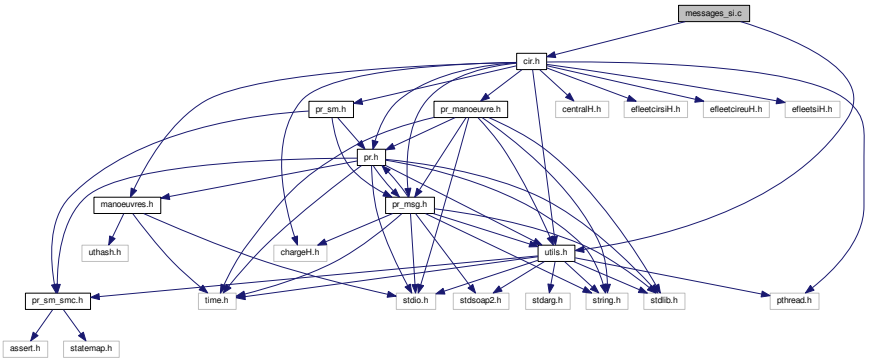
Carlos Rodríguez (CarlosRdrz)

4.6. Referencia del Archivo messages_si.c

Funciones de mensajes que se intercambian con el SI.

```
#include "cir.h"
#include "utils.h"
```

Dependencia gráfica adjunta para messages_si.c:



4.6.1. Descripción detallada

Funciones de mensajes que se intercambian con el SI. Estas funciones se ejecutan cada vez que se recibe un mensaje desde el SI con el nombre de esa funcion.

Autor

Carlos Rodríguez (CarlosRdrz)

4.7. Referencia del Archivo pr.h

Prototipos de las funciones de PR.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "utils.h"
#include "pr_sm_smc.h"
#include "pr_msg.h"
#include "manoeuvres.h"
```

Dependencia gráfica adjunta para pr.h:

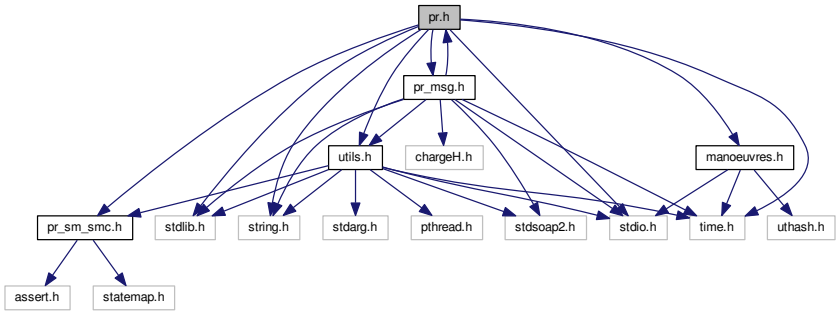
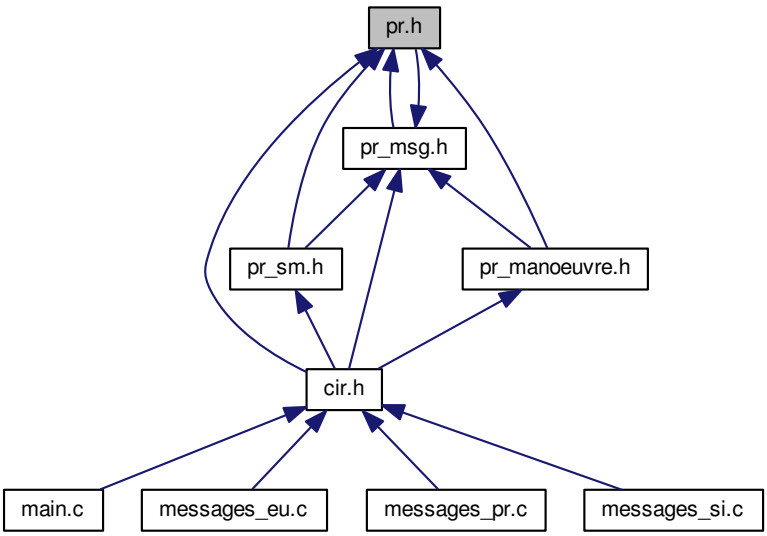


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Estructuras de datos

- struct pr

Funciones

- void pr_init (struct pr *this)

  Inicia la estructura PR (Constructor)
- void pr_update_time (struct pr *client)

  Actualiza el campo last_time de un PR, poniendole la hora actual.
- struct pr * new_pr (char *host, int port, int id, struct soap *soap)

  Crea una nueva estructura PR.
- void add_pr (struct pr *new_node, struct pr **list)

Crea una nueva estructura PR.

- struct pr * find_pr_by_host (char *host, struct pr *list)

  Devuelve el PR en el host indicado.
- struct pr * find_pr_by_id (int id, struct pr *list)

  Devuelve el PR en el id indicado.
- void delete_pr (struct pr *node_to_delete, struct pr **list)

  Elimina un PR de la lista.
- void free_prs (struct pr *list)

  Libera los recursos usado por la lista de PRs.
- void pr_charge_timeout (void *param)

  Para el proceso de carga por expiracion del contador.
- void pr_pause_finish (void *param)

  Se ejecuta por un timer cuando la pausa finaliza para volver al estado de espera fin de carga.

4.7.1. Descripción detallada

Prototipos de las funciones de PR. Contiene los prototipos de todas las funciones del CIR relacionadas con los PRs, como las maniobras a lanzar o las funciones que envian mensajes a PRs.

Autor

Carlos Rodríguez (CarlosRdrz)

4.7.2. Documentación de las funciones

4.7.2.1. void add_pr (struct pr * *new_node,* struct pr ** *list*)

Crea una nueva estructura PR.

Parámetros

new_node	Puntero al PR a agregar
list	Puntero a la lista donde agregar el PR

Devuelve

Void

```
33 {
34   // Si la lista esta vacia...
35   if (*list == NULL) {
36     *list = new_node;
37   } else {
38     // Coger el ultimo nodo de la lista
39     struct pr *current = *list;
40     while(current->next != NULL) {
41       current = current->next;
42     }
43     // Colocar como siguiente nodo el nuevo cliente
44     current->next = new_node;
45   }
46 }
```

4.7.2.2. void delete_pr (struct pr * *node_to_delete,* struct pr ** *list*)

Elimina un PR de la lista.

Parámetros

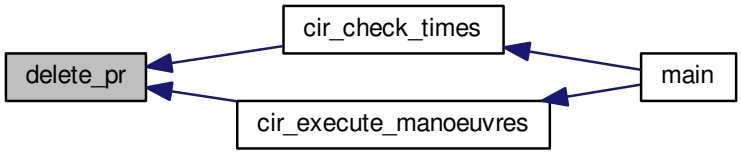
node_to_delete	Puntero a la estructura PR a eliminar de la lista de PRs
list	Puntero a la lista de PRs

Devuelve

Void

```
77 {
78   struct pr *previous;
79
80   // Si la lista no esta vacia...
81   if (*list != NULL) {
82     if (*list == node_to_delete) {
83       // Si es el primer nodo de la lista
84       *list = node_to_delete->next;
85     } else {
86       // Si no, cogemos el nodo anterior y actualizamos su campo next
87       previous = *list;
88       while(previous->next != NULL && previous->next != node_to_delete) {
89         previous = previous->next;
90       }
91       previous->next = node_to_delete->next;
92     }
93   }
94
95   free(node_to_delete);
96 }
```

Gráfico de llamadas a esta función:



4.7.2.3. struct pr* find_pr_by_host (char * *host*, struct pr * *list*)

Devuelve el PR en el host indicado.

Parámetros

host	Cadena con el host del PR
list	Puntero a la lista de PRs

Devuelve

Puntero al PR con el host indicado o NULL si no se encuentra

```
49 {
50   struct pr * node = list;
51
52   // Por cada nodo...
53   while (node != NULL) {
54     // Si el host es el que buscamos, devolvemos ese nodo
55     if (!strcmp(host, node->host)) return node;
56     node = node->next;
57   }
58
59   return NULL;
60 }
```

4.7.2.4. struct pr* find_pr_by_id (int *id*, struct pr * *list*)

Devuelve el PR en el id indicado.

Parámetros

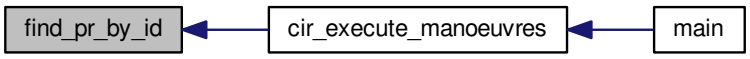
id	ID del PR a buscar
list	Puntero a la lista de PRs

Devuelve

Puntero al PR con el id indicado o NULL si no se encuentra

```
63 {
64   struct pr * node = list;
65
66   // Por cada nodo...
67   while (node != NULL) {
68     // Si el host es el que buscamos, devolvemos ese nodo
69     if (node->id == id) return node;
70     node = node->next;
71   }
72
73   return NULL;
74 }
```

Gráfico de llamadas a esta función:



4.7.2.5. void free_prs (struct pr * *list*)

Libera los recursos usado por la lista de PRs.

Parámetros

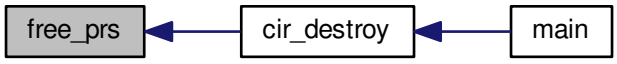
list	Puntero a la lista de PRs

Devuelve

Void

```
99 {
100   struct pr *current = list;
101   struct pr *next;
102   while(current != NULL) {
103     next = current->next;
104     free(current);
105     current = next;
106   }
107 }
```

Gráfico de llamadas a esta función:



4.7.2.6. struct pr* new_pr (char * *host*, int *port*, int *id*, struct soap * *soap*)

Crea una nueva estructura PR.

Parámetros

host	Cadena HTTP con el host
port	Puerto del servidor del PR
id	ID que tendra este PR
soap	Puntero a la instancia SOAP de este PR

Devuelve

Puntero a la estructura con los datos de este PR

```
17 {
18   struct pr * new_node = (struct pr *)malloc(sizeof(struct pr));
19   new_node->url = malloc(strlen(host) + 14);
20   new_node->host = malloc(strlen(host) + 1);
21   new_node->port = port;
22   new_node->last_time = time(NULL);
23   new_node->next = NULL;
24   new_node->id = id;
25   new_node->soap = soap;
26   strncpy(new_node->host, host, strlen(host) + 1);
27   sprintf(new_node->url, "http://%s:%d", host, port);
28   pr_init(new_node);
29   return new_node;
30 }
```

Gráfico de llamadas para esta función:

4.7.2.7. void pr_charge_timeout (void * *param*)

Para el proceso de carga por expiracion del contador.

Parámetros

param	Puntero a estructura del PR a parar

Devuelve

void

```
110 {
111   struct pr *pr = (struct pr *) param;
112 
113   // Si seguimos en el estado EFC...
114   if (getState(&pr->stateMachine) == &MapaPrincipal_E_EFC) {
115     pr_sm_smc_TE_SFTP(&pr->stateMachine);
116 
117     // Lanzamos mensaje OCPP RemoteStopTransaction
118     int result = pr_sendRemoteStopTransaction(pr);
119 
120     // Lanzamos transicion en funcion del resultado
121     // del mensaje RemoteStopTransaction
122     if (result) {
123       pr_sm_smc_TE_FINTP(&pr->stateMachine);
124     } else {
125       pr_sm_smc_TE_FINTP_TO(&pr->stateMachine);
126     }
127   }
128 }
```

Gráfico de llamadas para esta función:

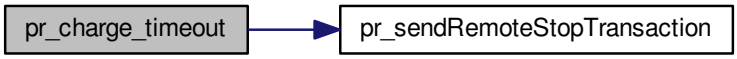
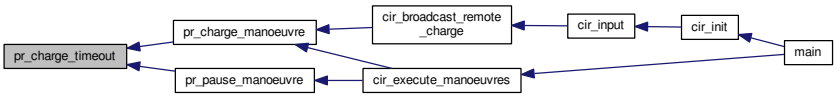


Gráfico de llamadas a esta función:



4.7.2.8. void pr_init (struct pr * *this*)

Inicia la estructura PR (Constructor)

Inicia las variables, las banderas y la maquina de estados.

Parámetros

this	la estructura del pr

```
3                           {
4   // Inicia la maquina de estados del PR
5   pr_sm_smc_Init(&this->stateMachine, this);
6   this->maniobra_en_ejecucion = NULL;
7 }
```

Gráfico de llamadas a esta función:

4.7.2.9. void pr_pause_finish (void * *param*)

Se ejecuta por un timer cuando la pausa finaliza para volver al estado de espera fin de carga.

Parámetros

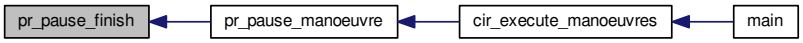
param	Puntero a estructura del PR a reanudar

Devuelve

void

```
131 {
132   struct pr *pr = (struct pr *) param;
133   // Volvemos a estado de espera fin de carga
134   pr_sm_smc_TE_EFC(&pr->stateMachine);
135 }
```

Gráfico de llamadas a esta función:



4.7.2.10. void pr_update_time (struct pr * *client*)

Actualiza el campo last_time de un PR, poniendole la hora actual.

Parámetros

cliente	Puntero al PR

Devuelve

Void

```
10 {
11   if (client != NULL) {
12     client->last_time = time(NULL);
13   }
14 }
```

4.8. Referencia del Archivo pr.sm

Fichero que describe la ME asociada al CIR.

4.8.1. Descripción detallada

Fichero que describe la ME asociada al CIR. M. R. Arahal, J. M. Maestre, C. Rodriguez

Versión

1.0

Objetivos: Se usa la ME para saber que mensajes gsoap enviar en cada momento. En general, la maquina proporciona respuestas de la siguiente forma:

```
respuesta=f(estimulo,estado)
```

donde la funcion f representa una funcion en sentido amplio, es decir, no debe ser asimilada con ninguna funcion en C. Estimulo se refiere a cualquier peticion que requiere una respuesta por parte de la ME. Finalmente, con estado nos referimos al estado actual de la maquina de estados.

Las transiciones en la maquina de estados se producen cuando se verifican determinadas condiciones de transito, que son comprobadas por una funcion de transito (FT). En el caso del CIR el lanzamiento de la FT es realizado:

- por gsoap por la llegada de peticiones del PR en la forma mens.req
- por indicacion directa del PCIR_MGCM
- por acciones del cronometro que a su vez ha sido lanzado por la entrada a algun estado

Para cada nodo de este archivo, se indicara:

- Sit: Situacion que describe el estado
- AE: Actuaciones de entrada al estado
- NDT: Nodo destino de una transicion
- LFT: Ente que lanza la funcion de transicion
- FT: Funcion de transicion
- CGT: Condiciones de guarda de la transicion

Los estados de la ME son:

- Estado E0: La ME comienza en este estado
  - Sit: Espera previa al comienzo de la carga
  - AE: Se publica en el EU y se modifican variables resumen maniobra (V_TC_RI)
  - NDT, LFT, FT, CGT:
    - E_SPR , PCIR_MGCM, TE_SPR(), -
  - Nota: El propio PCIR_MGCM lanza el paso al siguiente estado y envia OCPP RemoteStartTransaction.-req, recogiendo la respuesta
- Estado E_SPR
  - Sit: Se ha solicitado al PR que comience la recarga
  - AE: -

- NDT, LFT, FT, CGT:
  - E_EFC , PCIR_MGCM (mens.conf) , TE_EFC() , estado del PR permite operacion carga
  - E_FIN1 , PCIR_MGCM (mens.conf) , TE_FIN1() , estado del PR no permite operacion carga
  - E_FIN0 , PCIR_MGCM (timeout) , TE_FIN0() , -
- Nota: El mensaje RemoteStartTransaction.conf (de acuerdo con OCPP 1.5) ha de contener una descripcion del estado del PR indicando si puede iniciar la carga o no.

- Estado E_FIN1
  - Sit: Fin de maniobra sin carga por problema PR
  - AE:
    - Se publica en EU
    - Se actualizan variables resumen de maniobra
    - Se avisa a PCIR_MGCM de situaciÛn terminal mediante V_CF_MA
  - NDT, LFT, FT, CGT:
    - E_0 , PCIR_MGCM (V_CF_MA !=0), TE_E0() , -
  - Nota: La FT para la vuelta a E0 puede ser compartida por mas de un estado terminal

- Estado E_FIN0
  - Sit: Fin de maniobra sin carga sin respuesta de PR
  - AE:
    - Se publica en EU
    - Se actualizan variables resumen de maniobra
    - Se avisa a PCIR_MGCM de situaciÛn terminal mediante V_CF_MA
  - NDT, LFT, FT, CGT:
    - E_0 , PCIR_MGCM (V_CF_MA !=0), TE_E0() , -
  - Nota: -

- Estado E_EFC
  - Sit: Espera de fin de carga. El proceso de carga tiene lugar y puede ser pausado, abortado o finalizar
  - AE:
    - Se publica en EU
    - Puesta en marcha del cronometro de recarga
  - NDT, LFT, FT, CGT:
    - E_FCPR , StopTransaction.req, TE_FCPR() , -
    - E_SFTP , CronÛmetro recarga , TE_SFTP() , -
    - E_SFAB , PCIR_MGCM (MCIR02) , TE_SFAB() , -
    - E_SPAU , PCIR_MGCM (MCIR03) , TE_SPAU() , -
  - Nota:
    - Las posibles pausas/reanudaciones causadas por el PR pueden ser notificadas al CIR pero no producir·n la salida de E_EFC. Estas incidencias se consideran dentro del proceso normal de carga. Si la incidencia es grave y no hay reanudacion se tratar· como fin de carga indicado por el PR.
    - La ME no diferencia el origen de una orden de aborto pues, sea local (OIR) o remoto (SI), ha de pasar por el SI para validaciÛn
    - La ME no diferencia el origen de una orden de pausa pues, sea local (OIR) o remoto (SI), ha de pasar por el SI para validaciÛn
    - El cronometro lanzado puede ser invalidado o modificado en estados posteriores

- Estado E_FCPR
  - Sit: Fin de carga solicitado por el PR.

- AE:
  - Se publica en EU
  - Se detiene el cronometro
  - Se actualizan variables resumen de maniobra
  - Se avisa a PCIR_MGCM de situaciÛn terminal mediante V_CF_MA
- NDT, LFT, FT, CGT:
  - E_0 , PCIR_MGCM (V_CF_MA !=0), TE_E0() , -
- Nota:
  - El PR usa el mensaje OCPP StopTransaction.req para informar del estado correspondiente a la parada, incluyendo meterStop, timestamp y transactionId.

- Estado E_SFTP
  - Sit: Solicitud al PR de fin de carga por excederse tiempo planificado
  - AE: -
  - NDT, LFT, FT, CGT:
    - E_FINTP , PCIR_MGCM (mens.conf) , TE_FINTP() , -
    - E_FINTP0 , PCIR_MGCM (timeout) , TE_FINTP0() , -
  - Nota:
    - El cronometro cumplido se ha encargado de enviar OCPP RemoteStopTransaction.req
- Estado E_FINTP
  - Sit: Fin de carga por excederse tiempo planificado realizado correctamente
  - AE:
    - Se publica en EU
    - Se actualizan variables resumen de maniobra
    - Se avisa a PCIR_MGCM de situacion terminal mediante V_CF_MA
  - NDT, LFT, FT, CGT:
    - E_0 , PCIR_MGCM (V_CF_MA !=0), TE_E0() , -
  - Nota: -
- Estado E_FINTP0
  - Sit: Fin de carga por excederse tiempo planificado sin respuesta de PR
  - AE:
    - Se publica en EU
    - Se actualizan variables resumen de maniobra
    - Se avisa a PCIR_MGCM de situaciÛn terminal mediante V_CF_MA
  - NDT, LFT, FT, CGT:
    - E_0 , PCIR_MGCM (V_CF_MA !=0), TE_E0() , -
  - Nota: -
- Estado E_SFAB
  - Sit: Solicitud al PR de fin de carga por aborto
  - AE: -
  - NDT, LFT, FT, CGT:
    - E_FINAB , PCIR_MGCM (mens.conf) , TE_FINAB() , -
    - E_FINAB0 , PCIR_MGCM (timeout) , TE_FINAB0() , -
  - Nota: El PCIR_MGCM se ha encargado de enviar OCPP RemoteStopTransaction.req
- Estado E_FINAB

- Sit: Fin de carga por aborto realizado correctamente
- AE:
  - Se publica en EU
  - Se actualizan variables resumen de maniobra
  - Se avisa a PCIR_MGCM de situacion terminal mediante V_CF_MA
- NDT, LFT, FT, CGT:
  - E_0 , PCIR_MGCM (V_CF_MA !=0), TE_E0() , -
- Nota: -

■ Estado E_FINAB0

- Sit: Fin de carga por aborto sin respuesta de PR
- AE:
  - Se publica en EU
  - Se actualizan variables resumen de maniobra
  - Se avisa a PCIR_MGCM de situacion terminal mediante V_CF_MA
- NDT, LFT, FT, CGT:
  - E_0 , PCIR_MGCM (V_CF_MA !=0), TE_E0() , -
- Nota: -

■ Estado E_SPAU

- Sit: Solicitud al PR de pausa por un tiempo especificado
- AE: -
- NDT, LFT, FT, CGT:
  - E_PAUSA , PCIR_MGCM (mens.conf) , TE_PAUSA() , -
  - E_PAUS0 , PCIR_MGCM (timeout) , TE_PAUS0() , -
- Nota: La maniobra pausa siempre es por un tiempo determinado, no hay pausas indefinidas.

■ Estado E_PAUSA

- Sit: Pausa aceptada por el PR
- AE:
  - Se publica en EU
  - Se modifica cronometro para aumentar tiempo planificado y acomodar la pausa
- NDT, LFT, FT, CGT:
  - E_EFC , PCIR_MGCM , TE_EFC() , -
- Nota: El PCIR_MGCM se encargara de transmitir al SI el exito de la pausa

■ Estado E_PAUS0

- Sit: Pausa de carga sin respuesta de PR
- AE:
  - Se publica en EU
- NDT, LFT, FT, CGT:
  - E_EFC , PCIR_MGCM , TE_EFC() , -
- Nota: El PCIR_MGCM se encargar· de transmitir al SI el fracaso de la pausa

Atención

Notese que las transiciones en la maquina de estados se producen cuando se verifican determinadas condiciones de transito, que son comprobadas por una funcion de transito (FT), cuya definicion no esta contenida en este fichero.

4.9. Referencia del Archivo pr_manoeuvre.h

Prototipos de las funciones de maniobras.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "utils.h"
#include "pr.h"
#include "pr_msg.h"
```

Dependencia gráfica adjunta para pr_manoeuvre.h:

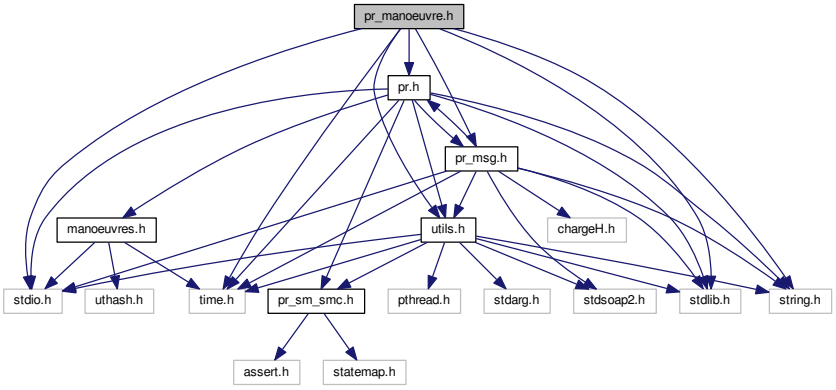
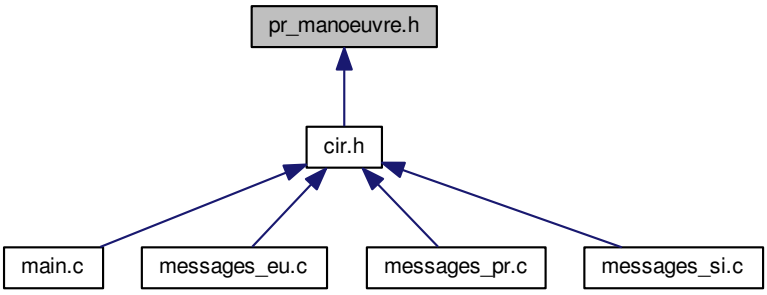


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Funciones

- void pr_charge_manoeuvre (struct pr ∗this, struct timer ∗∗timers)

  Maniobra de carga.
- void pr_abort_manoeuvre (struct pr ∗this)

  Maniobra de aborto a la carga.
- void pr_pause_manoeuvre (struct pr ∗this, char ∗secs, struct timer ∗∗timers)

  Maniobra de pausa a la carga.

4.9.1. Descripción detallada

Prototipos de las funciones de maniobras. Contiene los prototipos de todas las funciones que se ejecutan cuando el CIR ejecuta una maniobra.

Autor

Carlos Rodríguez (CarlosRdrz)

4.9.2. Documentación de las funciones

4.9.2.1. void pr_abort_manoeuvre (struct pr * *this*)

Maniobra de aborto a la carga.

Parámetros

this	la estructura del pr

Devuelve

void

```
36 {
37   // Lanzamos transicion a estado de aborto
38   pr_sm_smc_TE_SFAB(&this->stateMachine);
39
40   // Lanzamos mensaje OCPP RemoteStopTransaction
41   int result = pr_sendRemoteStopTransaction(this);
42
43   // Lanzamos transicion en funcion del resultado
44   // del mensaje RemoteStopTransaction
45   if (result) {
46     pr_sm_smc_TE_FINTP(&this->stateMachine);
47   } else {
48     pr_sm_smc_TE_FINTP_TO(&this->stateMachine);
49   }
50 }
```

Gráfico de llamadas para esta función:

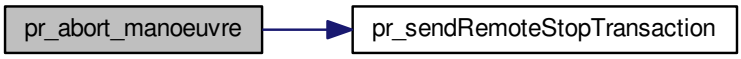
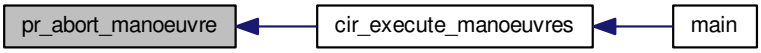


Gráfico de llamadas a esta función:



4.9.2.2. void pr_charge_manoeuvre (struct pr * *this,* struct timer ** *timers*)

Maniobra de carga.

Parámetros

this	la estructura del pr

Devuelve

void

```
4 {
5   // Si estabamos en otro estado que no sea E_0 (se supone que
6   // seria uno de fin) hacemos la transicion TE_E0 para pasar
7   // a ese estado
8   if (getState(&this->stateMachine) != &MapaPrincipal_E_0) {
9     pr_sm_smc_TE_E0(&this->stateMachine);
10   }
11
12   // Si estamos ahora en el estado E_0, comenzamos todo el
13   // proceso de carga
14   if (getState(&this->stateMachine) == &MapaPrincipal_E_0) {
15     // Lanza la transicion TE_SPR de la maquina de estados
16     pr_sm_smc_TE_SPR(&this->stateMachine);
17
18     // Enviamos mensaje RemoteStartTransaction
19     int result = pr_sendRemoteStartTransaction(this);
20
21     // Lanzamos una transicion u otra dependiendo del resultado
22     if (result == 1) {
23       new_timer(timers, &pr_charge_timeout, this, CHARGE_TIME);
24       pr_sm_smc_TE_EFC(&this->stateMachine);
25     } else if (result == -1) {
26       pr_sm_smc_TE_FIN1(&this->stateMachine);
27     } else {
28       pr_sm_smc_TE_FIN0(&this->stateMachine);
29     }
30   } else {
31     printLog("[ERROR] Couldnt start charge manoeuvre cause that PR is busy on another state.\n");
32   }
33 }
```

Gráfico de llamadas para esta función:

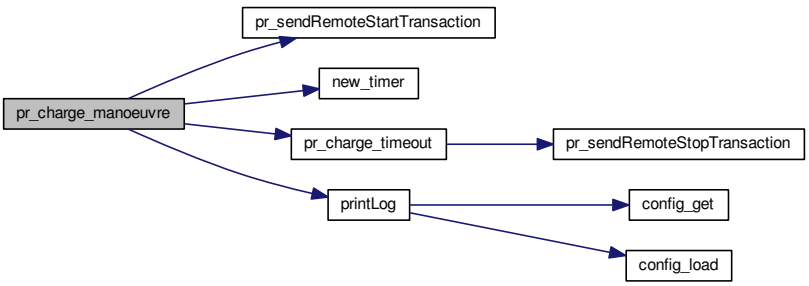
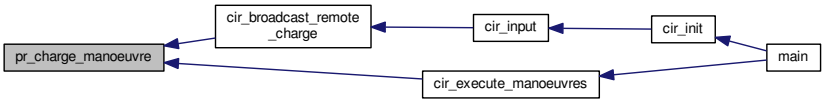


Gráfico de llamadas a esta función:



4.9.2.3. void pr_pause_manoeuvre (struct pr * *this*, char * *secs*, struct timer ** *timers*)

Maniobra de pausa a la carga.

Agrega una cantidad de segundos al timer de ese PR

Parámetros

this	la estructura del pr

Devuelve

void

```
53 {
54   // Segundos a pausar
55   int seconds = atoi(secs);
56
57   // Hacemos la primera transicion
58   pr_sm_smc_TE_SPAU(&this->stateMachine);
59
60   // Buscar el timer al que se le debe incrementar el
61   // contador de tiempo
62   struct timer * tim = *timers;
63
64   while (tim != NULL) {
65     if (tim->funct == pr_charge_timeout &&
66       tim->param == this) {
67       break;
68     } else {
69       tim = tim->next;
70     }
71   }
72
73   // Si se ha encontrado el timer
74   if (tim != NULL) {
75     tim->timestamp = tim->timestamp + seconds;
76     printLog("[INFO] Charge operation on PR %d paused %d seconds.\n", this->id, seconds);
77   } else {
78     printLog("[ERROR] Tried to do a %d seconds pause on PR %d, but the timer couldnt be found.\n",
     seconds, this->id);
79   }
80
81   // Enviamos mensaje DataTransfer con la pausa
82   int result = pr_sendPauseDataTransfer(this, secs);
83
84   if (result) {
85     pr_sm_smc_TE_PAUSA(&this->stateMachine);
86     new_timer(timers, &pr_pause_finish, this, seconds);
87   } else {
88     pr_sm_smc_TE_PAUS0(&this->stateMachine);
89   }
90 }
```

Gráfico de llamadas para esta función:

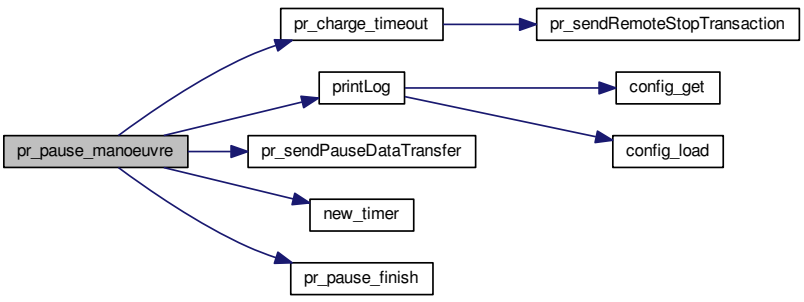
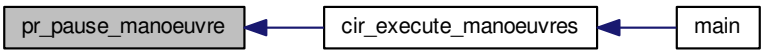


Gráfico de llamadas a esta función:



4.10. Referencia del Archivo pr_msg.h

Prototipos de las funciones de envio de mensajes OCPP.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "utils.h"
#include "pr.h"
#include "chargeH.h"
#include "stdsoap2.h"
```

Dependencia gráfica adjunta para pr_msg.h:

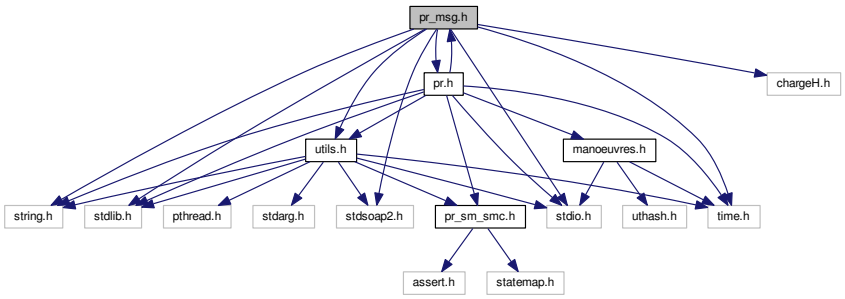
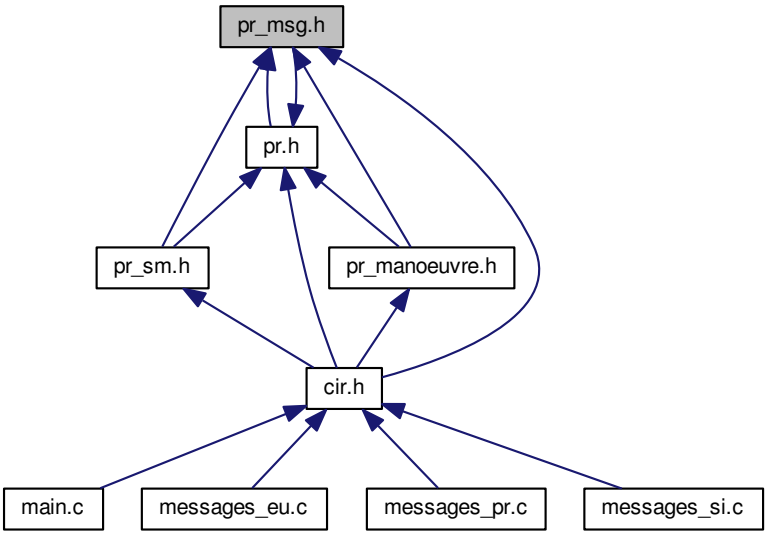


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Funciones

- int pr_sendRemoteStartTransaction (struct pr ∗this)

  Envia un mensaje RemoteStartTransaction.
- int pr_sendRemoteStopTransaction (struct pr ∗this)

  Envia un mensaje RemoteStopTransaction.
- int pr_sendPauseDataTransfer (struct pr ∗this, char ∗secs)

  Envia un mensaje DataTransfer con el mensaje "pause" para generar una pausa en el proceso de carga.
- int pr_sendRestartDataTransfer (struct pr ∗this)

  Envia un mensaje DataTransfer con el mensaje "restart" para provocar un reinicio del PR.
- int pr_sendUpdateFirmware (struct pr ∗this, char ∗firmware_url)

  Envia un mensaje FirmwareUpdate.
- int pr_sendChangeConfiguration (struct pr ∗this, char ∗params)

  Envia un mensaje ChangeConfiguration.

4.10.1. Descripción detallada

Prototipos de las funciones de envio de mensajes OCPP. Contiene los prototipos de todas las funciones del PR que se utilizan para enviar mensajes SOAP del protocolo OCPP a los PR. Cada funcion envia un mensaje del tipo especificado en el nombre de la funcion. Todas siguen la misma plantilla, pero los campos a enviar en los diferentes mensajes son distintos entre ellas.

Autor

Carlos Rodríguez (CarlosRdrz)

4.10.2. Documentación de las funciones

4.10.2.1. int pr_sendChangeConfiguration (struct pr * *this,* char * *params*)

Envia un mensaje ChangeConfiguration.

Parámetros

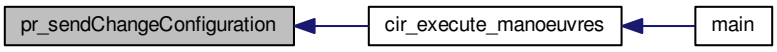
this	la estructura del pr

Devuelve

1 si todo fue correctamente 0 si hubo algun error

```
128 {
129   struct pr__ChangeConfigurationRequest request;
130   struct pr__ChangeConfigurationResponse response;
131   char *key, *value, *token;
132   int result = 0;
133
134   printf("--> ChangeConfiguration Request to PR %d at %s with params %s\n", this->id, this->url, params);
135
136   token = strchr(params, '=');
137   int longKey = strlen(params) - strlen(token) + 1;
138   int longValue = strlen(token);
139   key = (char *)calloc(longKey, sizeof(char));
140   value = (char *)calloc(longValue, sizeof(char));
141   strncpy(key, params, longKey - 1);
142   strncpy(value, token + 1, longValue);
143
144   // Campos del mensaje ChangeConfiguration
145   request.key = key;
146   request.value = value;
147
148   if (soap_call___pr__ChangeConfiguration(this->soap, this->url, NULL, &request, &response) == SOAP_OK) {
149     // Mensaje recibido correctamente
150     printf("<-- ChangeConfiguration Response from PR %s.\n", this->url);
151     result = 1;
152   } else {
153     printf("[ERROR] The PR %d at %s is not responding\n", this->id, this->url);
154   }
155
156   return result;
157 }
```

Gráfico de llamadas a esta función:



4.10.2.2. int pr_sendPauseDataTransfer (struct pr * *this*, char * *secs*)

Envia un mensaje DataTransfer con el mensaje "pause" para generar una pausa en el proceso de carga.

Parámetros

this	la estructura del pr

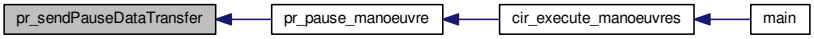
Devuelve

1 si todo fue correctamente 0 si hubo algun error

```
53 {
54   printf("--> DataTransfer PAUSE Request to PR %d at %s\n", this->id, this->url);
55   struct pr__DataTransferRequest request;
56   struct pr__DataTransferResponse response;
57   int result = 0;
58
59   // Campos del mensaje DataTransfer
60   request.vendorId = (char*)soap_malloc(this->soap, 10 * sizeof(char));
61   strncpy(request.vendorId, "DD", 10);
```

```
62   request.messageId = (char*)soap_malloc(this->soap, 6 * sizeof(char));
63   strncpy(request.messageId, "pause", 6);
64   request.data = (char*)soap_malloc(this->soap, 4 * sizeof(char));
65   strncpy(request.data, secs, 4);
66
67   if (soap_call___pr__DataTransfer(this->soap, this->url, NULL, &request, &response) == SOAP_OK) {
68     // Mensaje recibido correctamente
69     printf("<-- DataTransfer PAUSE Response from PR %s.\n", this->url);
70     result = 1;
71   } else {
72     printf("[ERROR] The PR %d at %s is not responding\n", this->id, this->url);
73   }
74
75   return result;
76 }
```

Gráfico de llamadas a esta función:



4.10.2.3. int pr_sendRemoteStartTransaction (struct pr * *this*)

Envia un mensaje RemoteStartTransaction.

Parámetros

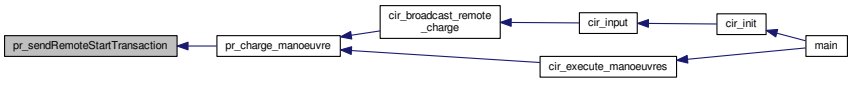
this	la estructura del pr

Devuelve

1 si todo fue correctamente 0 si hubo algun error

```
4 {
5   printf("--> RemoteStartTransaction Request to PR %d at %s\n", this->id, this->url);
6   struct pr__RemoteStartTransactionRequest request;
7   struct pr__RemoteStartTransactionResponse response;
8   int result = 0;
9
10   // Campos del mensaje RemoteStartTransaction
11   request.idTag = soap_malloc(this->soap, 10);
12   request.connectorId = NULL;
13   strncpy(request.idTag, "HOLA", 9);
14
15   if (soap_call___pr__RemoteStartTransaction(this->soap, this->url, NULL, &request, &response) == SOAP_OK)
      {
16     // Mensaje recibido correctamente
17     printf("<-- RemoteStartTransaction Response from PR %s.\n", this->url);
18     result = 1;
19   } else {
20     printf("[ERROR] The PR %d at %s notified an error on RemoteStartTransaction\n", this->id, this->url);
21   }
22
23   // Si el mensaje dice Rejected entonces hemos detecado una anomalía
24   if (response.status == pr__RemoteStartStopStatus__Rejected) {
25     result = 0;
26   }
27
28   return result;
29 }
```

Gráfico de llamadas a esta función:



4.10.2.4. int pr_sendRemoteStopTransaction (struct pr * *this*)

Envia un mensaje RemoteStopTransaction.

Parámetros

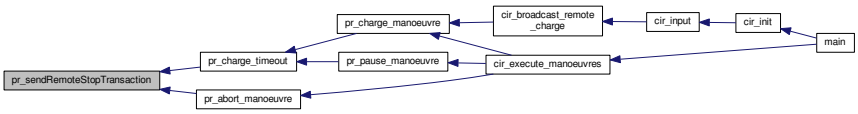
this	la estructura del pr

Devuelve

1 si todo fue correctamente 0 si hubo algun error

```
32 {
33   printf("--> RemoteStopTransaction Request to PR %d at %s\n", this->id, this->url);
34   struct pr__RemoteStopTransactionRequest request;
35   struct pr__RemoteStopTransactionResponse response;
36   int result = 0;
37
38   // Campos del mensaje RemoteStopTransaction
39   request.transactionId = 1;
40
41   if (soap_call___pr__RemoteStopTransaction(this->soap, this->url, NULL, &request, &response) == SOAP_OK) {
42     // Mensaje recibido correctamente
43     printf("<-- RemoteStopTransaction Response from PR %s.\n", this->url);
44     result = 1;
45   } else {
46     printf("[ERROR] The PR %d at %s is not responding\n", this->id, this->url);
47   }
48
49   return result;
50 }
```

Gráfico de llamadas a esta función:



4.10.2.5. int pr_sendRestartDataTransfer (struct pr * *this*)

Envia un mensaje DataTransfer con el mensaje "restart" para provocar un reinicio del PR.

Parámetros

this	la estructura del pr

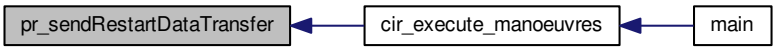
Devuelve

1 si todo fue correctamente 0 si hubo algun error

```
79 {
80   printf("--> DataTransfer RESTART Request to PR %d at %s\n", this->id, this->url);
81   struct pr__DataTransferRequest request;
82   struct pr__DataTransferResponse response;
83   int result = 0;
84
85   // Campos del mensaje DataTransfer
86   request.vendorId = (char*)soap_malloc(this->soap, 10 * sizeof(char));
87   strncpy(request.vendorId, "DD", 10);
88   request.messageId = (char*)soap_malloc(this->soap, 8 * sizeof(char));
89   strncpy(request.messageId, "restart", 8);
90   request.data = NULL;
91
92   if (soap_call___pr__DataTransfer(this->soap, this->url, NULL, &request, &response) == SOAP_OK) {
93     // Mensaje recibido correctamente
94     printf("<-- DataTransfer RESTART Response from PR %s.\n", this->url);
```

```
95     result = 1;
96   } else {
97     printf("[ERROR] The PR %d at %s is not responding\n", this->id, this->url);
98   }
99
100   return result;
101 }
```

Gráfico de llamadas a esta función:



4.10.2.6. int pr_sendUpdateFirmware (struct pr * *this*, char * *firmware_url*)

Envia un mensaje FirmwareUpdate.

Parámetros

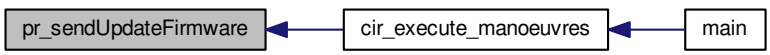
this	la estructura del pr

Devuelve

1 si todo fue correctamente 0 si hubo algun error

```
104 {
105   printf("--> UpdateFirmware Request to PR %d at %s with URL %s\n", this->id, this->url, firmware_url);
106   struct pr__UpdateFirmwareRequest request;
107   struct pr__UpdateFirmwareResponse response;
108   int result = 0;
109
110   // Campos del mensaje UpdateFirmware
111   request.retrieveDate = time(NULL);
112   request.location = firmware_url;
113   request.retries = NULL;
114   request.retryInterval = NULL;
115
116   if (soap_call___pr__UpdateFirmware(this->soap, this->url, NULL, &request, &response) == SOAP_OK) {
117     // Mensaje recibido correctamente
118     printf("<-- UpdateFirmware Response from PR %s.\n", this->url);
119     result = 1;
120   } else {
121     printf("[ERROR] The PR %d at %s is not responding\n", this->id, this->url);
122   }
123
124   return result;
125 }
```

Gráfico de llamadas a esta función:



4.11. Referencia del Archivo pr_sm.h

Prototipos de las funciones de PR de la ME.

```
#include "pr.h"
#include "pr_msg.h"
#include "pr_sm_smc.h"
```

Dependencia gráfica adjunta para pr_sm.h:

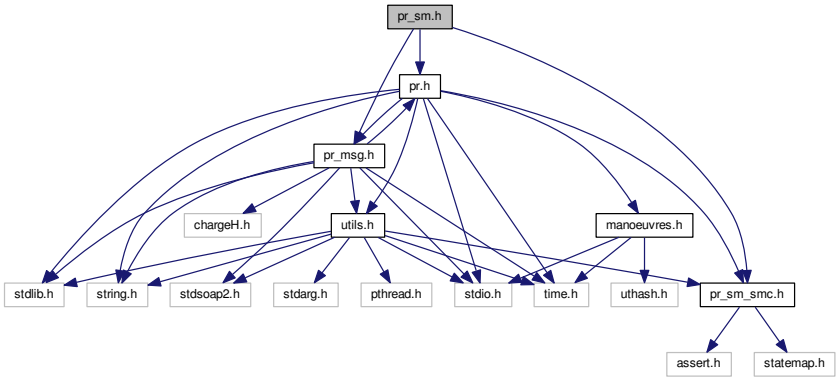
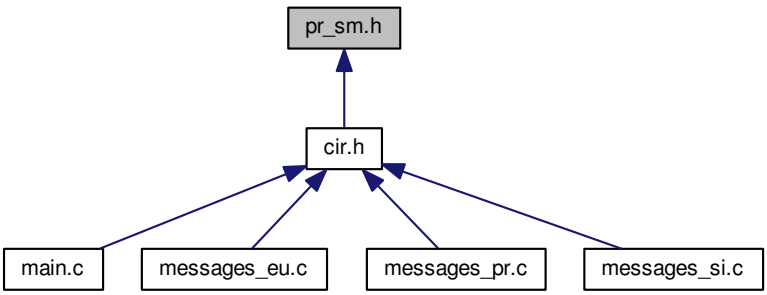


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Funciones

- void pr_e0_entry (struct pr ∗this)

  Se ejecuta al entrar al estado E0, donde el PR esta esperando a que se inicie la maniobra.
- void pr_spr_entry (struct pr ∗this)

  Se ejecuta al entrar en el estado SPR.
- void pr_efc_entry (struct pr ∗this)

  Se ejecuta al entrar al estado EFC.
- void pr_fcpr_entry (struct pr ∗this)

  Se ejecuta al entrar al estado FCPR.
- void pr_sfab_entry (struct pr ∗this)

  Se ejecuta al entrar al estado SFAB.

- void pr_sftp_entry (struct pr *this)
  Se ejecuta al entrar al estado SFTP.
- void pr_fin0_entry (struct pr *this)
  Se ejecuta al entrar al estado FIN0.
- void pr_fin1_entry (struct pr *this)
  Se ejecuta al entrar al estado FIN1.
- void pr_finab_entry (struct pr *this)
  Se ejecuta al entrar al estado FINAB.
- void pr_finab0_entry (struct pr *this)
  Se ejecuta al entrar al estado FINAB0.
- void pr_fintp_entry (struct pr *this)
  Se ejecuta al entrar al estado FINTP.
- void pr_fintp0_entry (struct pr *this)
  Se ejecuta al entrar al estado FINTP0.
- void pr_spau_entry (struct pr *this)
  Se ejecuta al entrar al estado SPAU.
- void pr_pausa_entry (struct pr *this)
  Se ejecuta al entrar al estado PAUSA.
- void pr_paus0_entry (struct pr *this)
  Se ejecuta al entrar al estado PAUS0.
- void pr_no_permitido (struct pr *this)
  Imprime mensaje por pantalla para alertar al usuario de que no es posible realizar esa transicion en este estado.

4.11.1. Descripción detallada

Prototipos de las funciones de PR de la ME. Contiene los prototipos de todas las funciones del CIR relacionadas con la ME de los PRs

Autor

Carlos Rodríguez (CarlosRdrz)

4.11.2. Documentación de las funciones

4.11.2.1. void pr_e0_entry (struct pr * *this*)

Se ejecuta al entrar al estado E0, donde el PR esta esperando a que se inicie la maniobra.

Parámetros

cliente	Puntero al PR

Devuelve

Void

```
4 {
5   NEW_STATE_PR("E0");
6 }
```

4.11.2.2. void pr_efc_entry (struct pr * *this*)

Se ejecuta al entrar al estado EFC.

Estamos cargando y esperando a que nos den una orden de parada de carga, o bien a que salte el timer del Timeout.

Parámetros

cliente	Puntero al PR

Devuelve

Void

```
14 {
15   NEW_STATE_PR("EFC");
16 }
```

4.11.2.3. void pr_fcpr_entry (struct pr * *this*)

Se ejecuta al entrar al estado FCPR.

Fin de carga solicitado por el PR

Parámetros

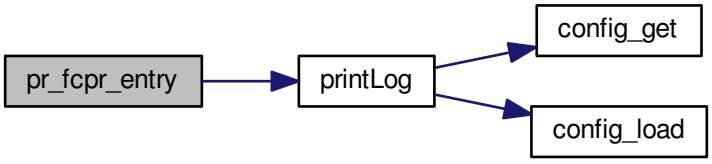
cliente	Puntero al PR

Devuelve

Void

```
19 {
20   NEW_STATE_PR("FCPR");
21   printLog("[INFO] Fin de carga en el PR %s es correcto.\n", this->url);
22 }
```

Gráfico de llamadas para esta función:



4.11.2.4. void pr_fin0_entry (struct pr * *this*)

Se ejecuta al entrar al estado FIN0.

La maniobra ha terminado porque el PR no ha respondido al mensaje RemoteStartTransaction.

Parámetros

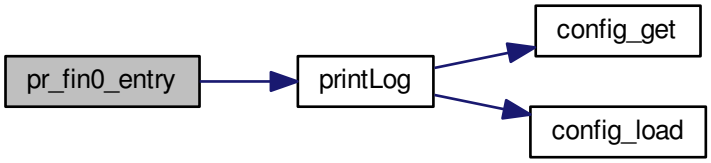
cliente	Puntero al PR

Devuelve

Void

```
35 {
36   NEW_STATE_PR("FIN0");
37   printLog("[INFO] Error. Finalizada carga con anomalias en PR %s.\n", this->url);
38   if (this->maniobra_en_ejecucion != NULL) this->maniobra_en_ejecucion->estado = 2;
39   if (this->maniobra_en_ejecucion != NULL) this->maniobra_en_ejecucion->resultado = -1;
40 }
```

Gráfico de llamadas para esta función:



4.11.2.5. void pr_fin1_entry (struct pr * *this*)

Se ejecuta al entrar al estado FIN1.

La maniobra ha terminado porque el PR no acepta una operacion de carga en este momento.

Parámetros

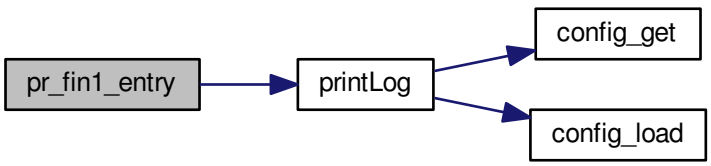
cliente	Puntero al PR

Devuelve

Void

```
43 {
44   NEW_STATE_PR("FIN1");
45   printLog("[INFO] Error. El PR %s no responde a solicitudes.\n", this->url);
46   if (this->maniobra_en_ejecucion != NULL) this->maniobra_en_ejecucion->estado = 2;
47   if (this->maniobra_en_ejecucion != NULL) this->maniobra_en_ejecucion->resultado = -1;
48 }
```

Gráfico de llamadas para esta función:



4.11.2.6. void pr_finab0_entry (struct pr * *this*)

Se ejecuta al entrar al estado FINAB0.

La maniobra ha terminado porque el PR no ha respondido a un mensaje de aborto.

Parámetros

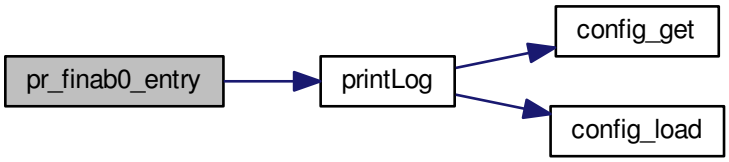
cliente	Puntero al PR

Devuelve

Void

```
59 {
60   NEW_STATE_PR("FINAB0");
61   printLog("[INFO] Error. El PR %s no responde a solicitudes.\n", this->url);
62   if (this->maniobra_en_ejecucion != NULL) this->maniobra_en_ejecucion->estado = 2;
63   if (this->maniobra_en_ejecucion != NULL) this->maniobra_en_ejecucion->resultado = -1;
64 }
```

Gráfico de llamadas para esta función:



4.11.2.7. void pr_finab_entry (struct pr * *this*)

Se ejecuta al entrar al estado FINAB.

La maniobra ha terminado correctamente tras una solicitud de aborto.

Parámetros

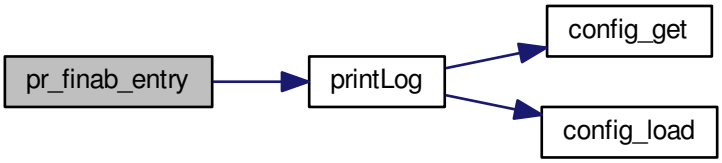
cliente	Puntero al PR

Devuelve

Void

```
51 {
52   NEW_STATE_PR("FINAB");
53   printLog("[INFO] Fin de carga por aborto en PR %s correcto.\n", this->url);
54   if (this->maniobra_en_ejecucion != NULL) this->maniobra_en_ejecucion->estado = 2;
55   if (this->maniobra_en_ejecucion != NULL) this->maniobra_en_ejecucion->resultado = 0;
56 }
```

Gráfico de llamadas para esta función:



4.11.2.8. void pr_fintp0_entry (struct pr * *this*)

Se ejecuta al entrar al estado FINTP0.

La maniobra ha terminado tras transcurrir el tiempo de carga pero el PR no contesto al mensaje RemoteStop-Transaction

Parámetros

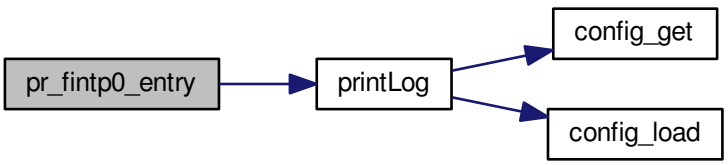
cliente	Puntero al PR

Devuelve

Void

```
75 {
76   NEW_STATE_PR("FINTP0");
77   printLog("[INFO] Error. El PR %s no responde a solicitudes.\n", this->url);
78   if (this->maniobra_en_ejecucion != NULL) this->maniobra_en_ejecucion->estado = 2;
79   if (this->maniobra_en_ejecucion != NULL) this->maniobra_en_ejecucion->resultado = -1;
80 }
```

Gráfico de llamadas para esta función:



4.11.2.9. void pr_fintp_entry (struct pr * *this*)

Se ejecuta al entrar al estado FINTP.

La maniobra ha terminado correctamente tras transcurrir el tiempo de carga y enviar el mensaje RemoteStop-Transaction al PR.

Parámetros

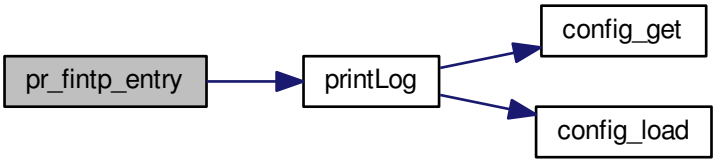
cliente	Puntero al PR

Devuelve

Void

```
67 {
68   NEW_STATE_PR("FINTP");
69   printLog("[INFO] Fin de carga por expiracion en PR %s correcto.\n", this->url);
70   if (this->maniobra_en_ejecucion != NULL) this->maniobra_en_ejecucion->estado = 2;
71   if (this->maniobra_en_ejecucion != NULL) this->maniobra_en_ejecucion->resultado = 0;
72 }
```

Gráfico de llamadas para esta función:



4.11.2.10. void pr_no_permitido (struct pr * *this*)

Imprime mensaje por pantalla para alertar al usuario de que no es posible realizar esa transicion en este estado.

Parámetros

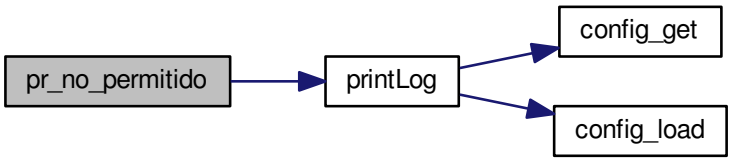
cliente	Puntero al PR

Devuelve

Void

```
101 {
102   printLog("[INFO] Operacion no permitida en PR %s.\n", this->url);
103 }
```

Gráfico de llamadas para esta función:



4.11.2.11. void pr_paus0_entry (struct pr * *this*)

Se ejecuta al entrar al estado PAUS0.

La maniobra de carga ha intentado pausar pero el PR no ha respondido

Parámetros

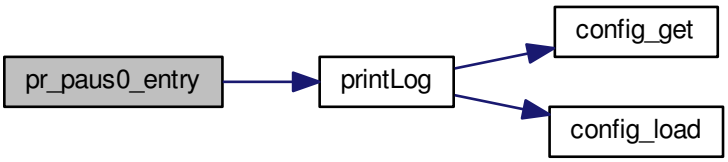
cliente	Puntero al PR

Devuelve

Void

```
93 {
94   NEW_STATE_PR("PAUS0");
95   printLog("[INFO] Error. El PR %s no responde a solicitudes.\n", this->url);
96   if (this->maniobra_en_ejecucion != NULL) this->maniobra_en_ejecucion->estado = 2;
97   if (this->maniobra_en_ejecucion != NULL) this->maniobra_en_ejecucion->resultado = -1;
98 }
```

Gráfico de llamadas para esta función:



4.11.2.12. void pr_pausa_entry (struct pr * *this*)

Se ejecuta al entrar al estado PAUSA.

La maniobra de carga se ha pausado correctamente

Parámetros

cliente	Puntero al PR

Devuelve

Void

```
88 {
89   NEW_STATE_PR("PAUSA");
90 }
```

4.11.2.13. void pr_sfab_entry (struct pr * *this*)

Se ejecuta al entrar al estado SFAB.

Hemos recibido una solicitud de aborto, y enviamos mensaje RemoteStopTransaction para parar la carga de ese PR.

Parámetros

cliente	Puntero al PR

Devuelve

Void

```
25 {
26   NEW_STATE_PR("SFAB");
27 }
```

4.11.2.14. void pr_sftp_entry (struct pr * *this*)

Se ejecuta al entrar al estado SFTP.

Ha saltado el Timeout, y debemos parar el PR enviando un RemoteStopTransaction.

Parámetros

cliente	Puntero al PR

Devuelve

Void

```
30 {
31   NEW_STATE_PR("SFTP");
32 }
```

4.11.2.15. void pr_spau_entry (struct pr * *this*)

Se ejecuta al entrar al estado SPAU.

Solicitud de pausa al PR

Parámetros

cliente	Puntero al PR

Devuelve

Void

```
83 {
84   NEW_STATE_PR("SPAU");
85 }
```

4.11.2.16. void pr_spr_entry (struct pr * *this*)

Se ejecuta al entrar en el estado SPR.

Enviamos un mensaje RemoteStartTransaction al PR.

Parámetros

cliente	Puntero al PR

Devuelve

Void

```
9 {
10   NEW_STATE_PR("SPR");
11 }
```

4.12. Referencia del Archivo utils.h

Prototipos de las funciones de utilidad.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <pthread.h>
#include <stdarg.h>
#include "stdsoap2.h"
#include "pr_sm_smc.h"
```

Dependencia gráfica adjunta para utils.h:

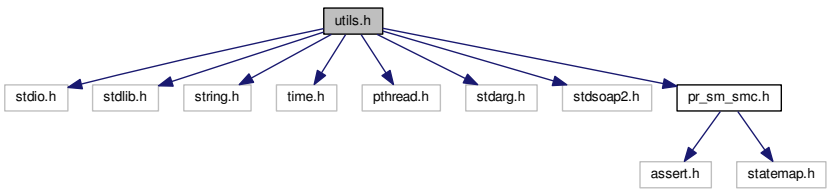
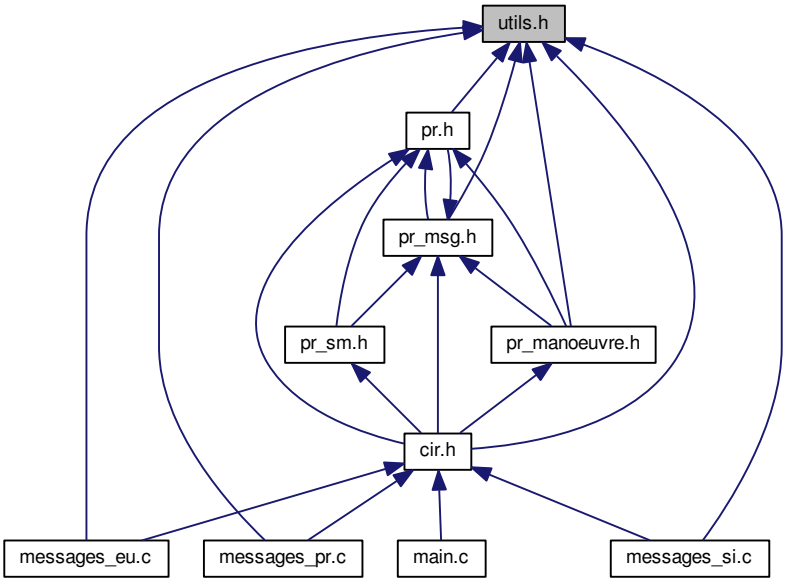


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Estructuras de datos

- struct req_info
- struct req_queue
- struct config_node
- struct timer

Funciones

- void req_queue_init (struct req_queue *queue)

  Inicializa la cola de peticiones.
- void req_queue_destroy (struct req_queue *queue)

  Destruye la cola de peticiones y libera los recursos utilizados.
- int req_queue_enqueue (struct req_queue *queue, struct req_info *sock)

  Encola una nueva peticion.
- struct req_info * req_queue_dequeue (struct req_queue *queue)

  Saca un elemento de la cola.
- struct config_node * config_load (const char *file_name)

  Carga un archivo de configuracion.
- char * config_get (struct config_node *config, const char *key)

  Obtiene una propiedad de una lista de propiedades de un archivo de configuracion.
- void config_destroy (struct config_node *config)

  Libera recursos utilizados por los nodos de propiedades que se crearon al cargar un archivo de configuracion.
- void new_timer (struct timer **cola, void(*fun)(void *), void *param, int cuando)

  Especifica una funcion a ejecutar en un instante determinado.
- void delete_timer (struct timer **cola, struct timer *to_delete)

  Borra un timer de la cola de timers.

- void printLog (const char *fmt,...)

  Escribe en los logs y por pantalla un mensaje.

4.12.1. Descripción detallada

Prototipos de las funciones de utilidad. Contiene los prototipos de todas las funciones de utilidad del PR. Algunas se utilizan tambien en el CIR. Ofrecen soporte a tipos de datos definidos en la aplicacion como la cola de transiciones o el archivo de configuracion.

Autor

Carlos Rodríguez (CarlosRdrz)

4.12.2. Documentación de las funciones

4.12.2.1. void config_destroy (struct config_node * *config*)

Libera recursos utilizados por los nodos de propiedades que se crearon al cargar un archivo de configuracion.

Parámetros

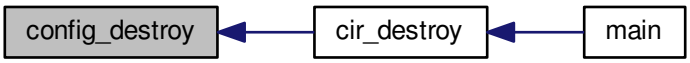
queue	Puntero al primer nodo de la lista de propiedades

Devuelve

void

```
122 {
123   struct config_node *node = config;
124   struct config_node *next;
125 
126   while(node != NULL) {
127     next = node->next;
128     free(node);
129     node = next;
130   }
131 }
```

Gráfico de llamadas a esta función:



4.12.2.2. char* config_get (struct config_node * *config*, const char * *key*)

Obtiene una propiedad de una lista de propiedades de un archivo de configuracion.

Parámetros

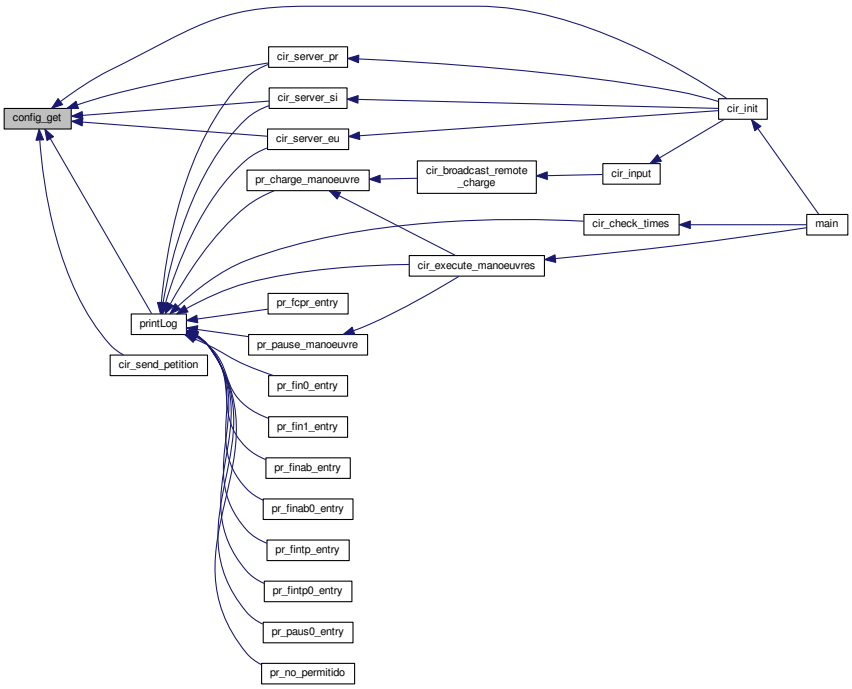
config	Puntero al primer nodo de la lista de propiedades
key	Clave de la propiedad que queremos obtener

Devuelve

Cadena con el valor de la clave que hemos buscado

```
109 {
110   struct config_node *node = config;
111   char *value = NULL;
112
113   while(node != NULL) {
114     if(!strcmp(node->key, key)) value = node->value;
115     node = node->next;
116   }
117
118   return value;
119 }
```

Gráfico de llamadas a esta función:



4.12.2.3. struct config_node* config_load (const char * *file_name*)

Carga un archivo de configuracion.

Parámetros

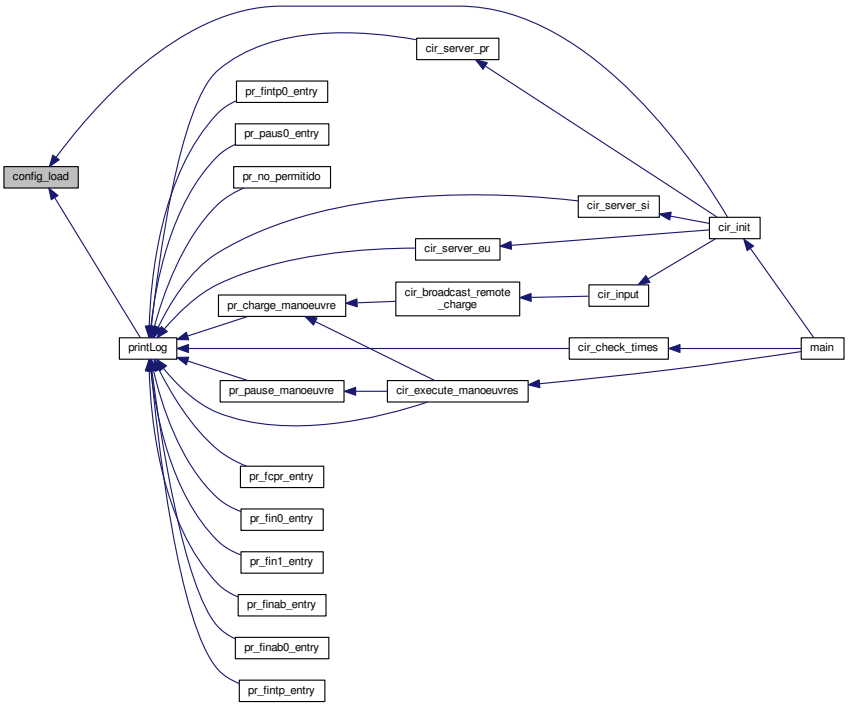
file_name	Nombre del archivo de configuracion a leer

Devuelve

Puntero al primer nodo de la lista de propiedades del archivo de configuracion leido

```
79 {
80   struct config_node *first = NULL;
81   struct config_node *tail = NULL;
82   FILE *file = fopen(file_name, "r"); // Abre el archivo
83
84   if (file != NULL) {
85     char line[128];
86     first = (struct config_node *)calloc(1, sizeof(struct config_node));
87     tail = first;
88
89     // Por cada linea, dividir por el caracter delimitador y crear
90     // el nodo en la lista enlazada de estructuras config_node
91     while(fgets(line, sizeof(line), file) != NULL) {
92       char *cfline;
93       cfline = strstr((char *)line, CONFIG_DELIMITER);
94       cfline = cfline + strlen(CONFIG_DELIMITER);
95       strncpy(tail->key, line, (strlen(line) - strlen(cfline) - 1));
96       strncpy(tail->value, cfline, strlen(cfline) - 1);
97       tail->next = (struct config_node *)calloc(1, sizeof(struct
    config_node));
98       tail = tail->next;
99     }
100   }
101
102   // Cerramos el archivo
103   fclose(file);
104
105   return first;
106 }
```

Gráfico de llamadas a esta función:



4.12.2.4. void delete_timer (struct timer ** *cola*, struct timer * *to_delete*)

Borra un timer de la cola de timers.

Parámetros

cola	Cola de timers
to_delete	Nodo a eliminar

Devuelve

void

```
156 {
157   struct timer *previous;
158
159   // Si la cola no esta vacia...
160   if (*cola != NULL) {
161     if (*cola == to_delete) {
162       // Si es el primer nodo de la lista
163       *cola = to_delete->next;
164     } else {
165       // Si no, cogemos el nodo anterior y actualizamos su campo next
166       previous = *cola;
167       while (previous->next != NULL && previous->next != to_delete) {
168         previous = previous->next;
169       }
170       previous->next = to_delete->next;
171     }
172   }
173
174   free(to_delete);
175 }
```

Gráfico de llamadas a esta función:

4.12.2.5. void new_timer (struct timer ** *cola*, void(*)(void *) *fun*, void * *param*, int *cuando*)

Especifica una funcion a ejecutar en un instante determinado.

Parámetros

cola	Cola de timers a ejecutar
fun	Funcion a ejecutar
cuando	Instante en el que ejecutar la funcion

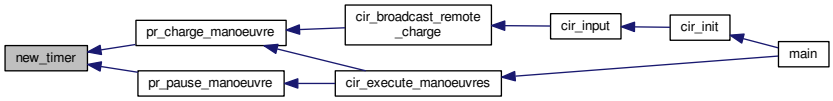
Devuelve

void

```
134 {
135   struct timer *new_node = (struct timer *)malloc(sizeof(struct timer));
136   new_node->funct = fun;
137   new_node->param = param;
138   new_node->timestamp = time(NULL) + cuando;
139   new_node->next = NULL;
140
141   // Si la cola esta vacia...
142   if (*cola == NULL) {
143     *cola = new_node;
144   } else {
145     // Coger el ultimo nodo de la lista
146     struct timer *current = *cola;
```

```
147     while(current->next != NULL) {
148       current = current->next;
149     }
150     // Colocar como siguiente nodo el nuevo cliente
151     current->next = new_node;
152   }
153 }
```

Gráfico de llamadas a esta función:



4.12.2.6. void printLog (const char * *fmt*, ...)

Escribe en los logs y por pantalla un mensaje.

Parámetros

this	la estructura del cir

```
178 {
179   FILE *file;
180   char logfile[PATH_MAX];
181   time_t t = time(NULL);
182   struct tm * timeinfo = localtime(&t);
183   char filedate[16];
184   char currentTime[10];
185   static char * logDir;
186   int res;
187 
188   if (logDir == NULL) {
189     logDir = config_get(config_load("config"), "LOG_DIR"); // Lee LOG_DIR del archivo
      de configuracion
190   }
191 
192   va_list args;
193   va_start(args, fmt);
194 
195   strftime(filedate, 16, "%F.log", timeinfo);
196   strftime(currentTime, 10, "%T", timeinfo);
197   sprintf(logfile, "%s/%s", logDir, filedate);
198 
199   // Open the file
200   file = fopen(logfile, "a");
201 
202   // Print to std out
203   vprintf(fmt, args);
204   // Print time to file
205   fprintf(file, "%s ", currentTime);
206   // Print msg to file
207   res = vfprintf(file, fmt, args);
208 
209   if (res < 0) printf("Error writing to a log\n");
210   fclose(file);
211 }
```

Gráfico de llamadas para esta función:

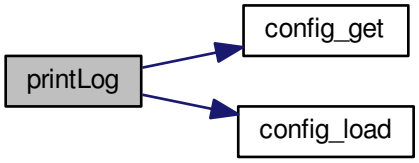
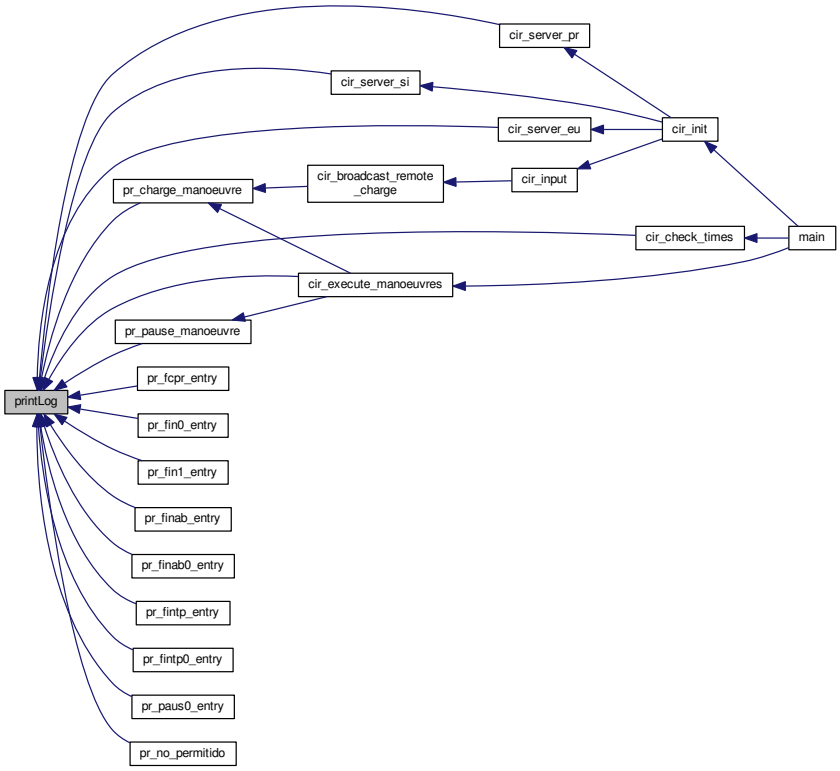


Gráfico de llamadas a esta función:



4.12.2.7. struct req_info* req_queue_dequeue (struct req_queue * *queue*)

Saca un elemento de la cola.

Parámetros

queue	Puntero a la estructura de la cola de peticiones

Devuelve

Estructura de la peticion obtenida

```
53 {
54   // Peticion a devolver
55   struct req_info * req_info = NULL;
56
57   // Bloqueamos el mutex
58   pthread_mutex_lock(&queue->lock);
59
60   // Mientras current sea igual a next, significa que la cola esta vacía y por
61   // lo tanto debemos esperar a que se introduzca un elemento en la cola
62   while (queue->current == queue->next) pthread_cond_wait(&queue->
      cond, &queue->lock);
63
64   // Cogemos la peticion para devolverla
65   req_info = queue->queue[queue->next++];
66
67   // Si la proxima posicion sobrepasa las posiciones en la cola, hacemos que
68   // la proxima posicion sea el inicio de la cola
69   if (queue->next >= MAX_QUEUE)
70     queue->next = 0;
71
72   // Desbloqueamos el mutex
73   pthread_mutex_unlock(&queue->lock);
74
75   return req_info;
76 }
```

Gráfico de llamadas a esta función:

4.12.2.8. void req_queue_destroy (struct req_queue * *queue*)

Destruye la cola de peticiones y libera los recursos utilizados.

Parámetros

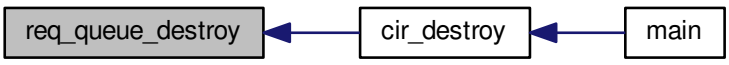
queue	Puntero a la estructura de la cola de peticiones

Devuelve

void

```
12 {
13   pthread_mutex_destroy(&queue->lock);
14   pthread_cond_destroy(&queue->cond);
15 }
```

Gráfico de llamadas a esta función:



4.12.2.9. int req_queue_enqueue (struct req_queue * *queue*, struct req_info * *sock*)

Encola una nueva peticion.

Parámetros

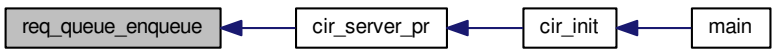
queue	Puntero a la estructura de la cola de peticiones
req_info	Estructura de peticion a encolar

Devuelve

SOAP_OK si se encolo correctamente SOAP_EOM si la cola esta llena

```
18 {
19   int status = SOAP_OK;
20   int next;
21
22   // Bloquea el mutex de la cola para que no se realicen
23   // operaciones sobre ella mientras se realiza esta
24   pthread_mutex_lock(&queue->lock);
25
26   next = queue->current + 1;
27   // Si current+1 > posicion_en_la_cola es que la posicion de la siguiente peticion
28   // a encolar es mayor que el numero de elementos posibles en la cola, y por lo tanto
29   // se debe comenzar de nuevo por la posicion 0 en la cola.
30   if (next >= MAX_QUEUE)
31     next = 0;
32   // Si current+1 es igual a next, entonces es que no quedan posiciones libres
33   // en la cola, y lo que hacemos es devolver el valor SOAP_EOM, que hace que
34   // el llamante espere 1 segundo y lo vuelva a intentar
35   if (next == queue->next)
36     status = SOAP_EOM;
37   else {
38     // Si no, todo es correcto y añadimos la nueva peticion
39     queue->queue[queue->current] = req_info;
40     queue->current = next;
41   }
42
43   // Lanzamos una señal al cond, por si habia alguien esperando
44   pthread_cond_signal(&queue->cond);
45
46   // Desbloqueamos el mutex
47   pthread_mutex_unlock(&queue->lock);
48
49   return status;
50 }
```

Gráfico de llamadas a esta función:



4.12.2.10. void req_queue_init (struct req_queue * *queue*)

Inicializa la cola de peticiones.

Parámetros

queue	Puntero a la estructura de la cola de peticiones

Devuelve

void

```
4 {
5   queue->current = 0;
6   queue->next = 0;
7   pthread_mutex_init(&queue->lock, NULL);
8   pthread_cond_init(&queue->cond, NULL);
9 }
```

Índice alfabético

Efleet CIR

0.1

Generado por Doxygen 1.8.6

Jueves, 11 de Septiembre de 2014 16:27:05

Índice general

Capítulo 1

Índice de estructura de datos

1.1. Estructura de datos

Lista de estructuras con una breve descripción:

Capítulo 2

Indice de archivos

2.1. Lista de archivos

Lista de todos los archivos documentados y con descripciones breves:

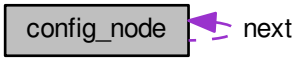
Capítulo 3

Documentación de las estructuras de datos

3.1. Referencia de la Estructura config_node

`#include <utils.h>`

Diagrama de colaboración para config_node:



Campos de datos

- char key [20]
- char value [64]
- struct config_node * next

3.1.1. Descripción detallada

Estructura que almacena las propiedades del archivo de configuracion

3.1.2. Documentación de los campos

3.1.2.1. char config_node::key[20]

Nombre de la propiedad

3.1.2.2. struct config_node* config_node::next

Siguiente propiedad en la lista

3.1.2.3. char config_node::value[64]

Valor de la propiedad

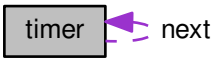
La documentación para esta estructura fue generada a partir del siguiente fichero:

- utils.h

3.2. Referencia de la Estructura timer

`#include <utils.h>`

Diagrama de colaboración para timer:



Campos de datos

- void(∗ funct)(void ∗)
- int timestamp
- void ∗ param
- struct timer ∗ next

3.2.1. Descripción detallada

Estructura que almacena la cola de funciones a ejecutar por el timer

3.2.2. Documentación de los campos

3.2.2.1. void(∗ timer::funct)(void ∗)

Funcion a ejecutar

3.2.2.2. struct timer∗ timer::next

Siguiente funcion a ejecutar

3.2.2.3. void∗ timer::param

Parametro opcional para la funcion

3.2.2.4. int timer::timestamp

Tiempo cuando se debe ejecutar la funcion

La documentación para esta estructura fue generada a partir del siguiente fichero:

- utils.h

Capítulo 4

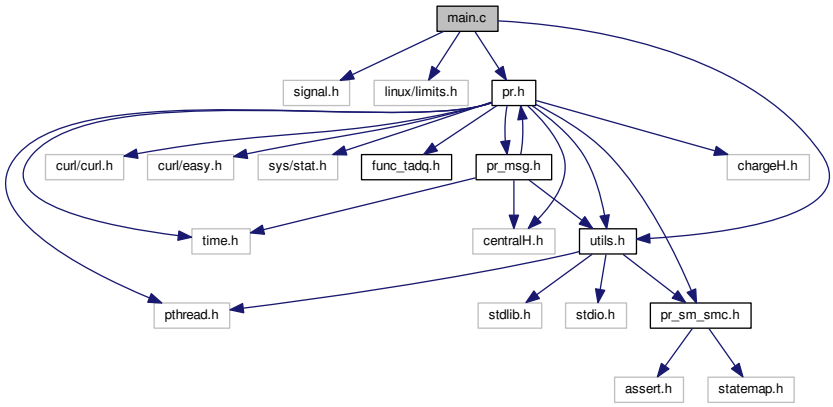
Documentación de archivos

4.1. Referencia del Archivo main.c

Archivo principal del PR.

```
#include <signal.h>
#include <linux/limits.h>
#include "pr.h"
#include "utils.h"
```

Dependencia gráfica adjunta para main.c:



Funciones

- void signalsHandler (int signal)

  Maneja la señal que se produce al pulsar CTRL+C (SIGINT). Coloca la bandera sigint en 1 para realizar una salida ordenada.

- int main (int argc, char ∗argv[])

  Funcion principal del PR.

Variables

- int sigint = 0

4.1.1. Descripción detallada

Archivo principal del PR.

Autor

Carlos Rodríguez (CarlosRdrz)

4.1.2. Documentación de las funciones

4.1.2.1. int main (int *argc*, char * *argv[]*)

Funcion principal del PR.

Parámetros

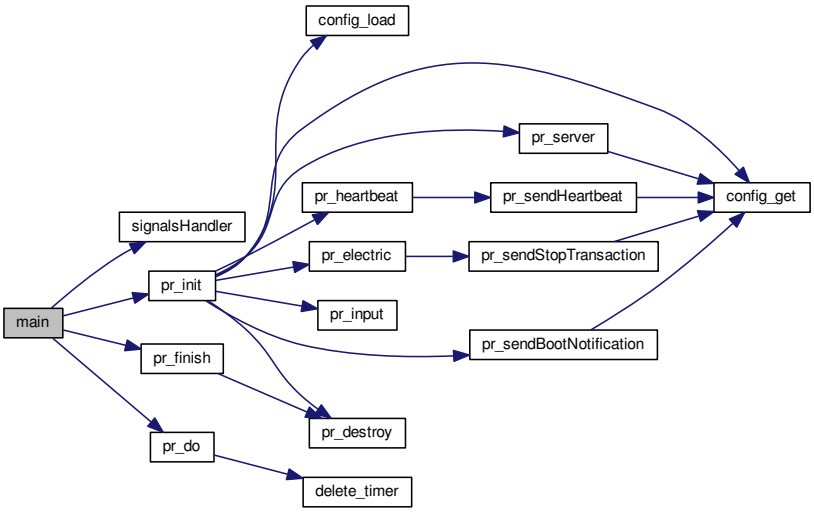
argc	Numero de parametros introducidos en la linea de comandos
argv	Array de parametros introducidos en la linea de comandos

Devuelve

int Indica si la salida fue exitosa

```
36 {
37   // El objeto PR. Guarda todas las variables del PR.
38   struct pr pr;
39   char path[PATH_MAX];
40
41   // Leemos la ruta del ejecutable, por si hay que reiniciar luego
42   readlink("/proc/self/exe", path, PATH_MAX);
43
44   // Handle SIGINT signal
45   // Hace que cuando pulsemos Ctrl+C, se ejecute la funcion signalsHandler
46   signal(SIGINT, signalsHandler);
47
48   // Inicializa el objeto PR
49   pr_init(&pr);
50
51   // Mientras este la bandera Running...
52   while (pr.flags & Running) {
53     // Ejecutar timers
54     pr_do(&pr);
55     // Si se pulso CTRL+C, comenzar a salir
56     if (sigint) {
57       pr_finish(&pr);
58       sigint = 0;
59     }
60     // Esperar 250 ms
61     usleep(250);
62   }
63
64   // Si el PR debe ser reiniciado...
65   if (pr.flags & Restart) {
66     execl(path, path, NULL);
67   }
68
69   return 0;
70 }
```

Gráfico de llamadas para esta función:



4.1.2.2. void signalsHandler (int *signal*)

Maneja la señal que se produce al pulsar CTRL+C (SIGINT). Coloca la bandera sigint en 1 para realizar una salida ordenada.

Parámetros

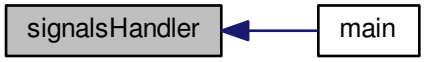
signal	Entero que indica el tipo de señal

Devuelve

void

```
22 {
23   if (signal == SIGINT) {
24     printf(" -> Received SIGINT. Exiting...\n");
25     sigint = 1;
26   }
27 }
```

Gráfico de llamadas a esta función:



4.1.3. Documentación de las variables

4.1.3.1. int sigint = 0

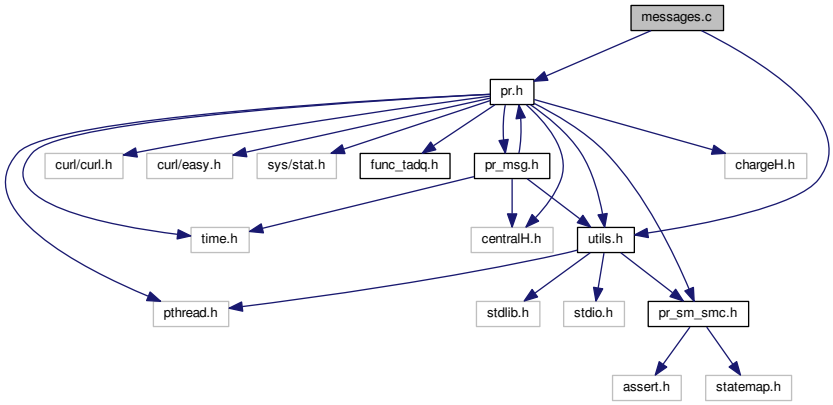
Bandera. Se coloca en 1 cuando se pulsa CTRL+C

4.2. Referencia del Archivo messages.c

Funciones de mensajes OCPP.

```
#include "pr.h"
#include "utils.h"
```

Dependencia gráfica adjunta para messages.c:



4.2.1. Descripción detallada

Funciones de mensajes OCPP. Estas funciones se ejecutan cada vez que se recibe un mensaje OCPP con el nombre de esa funcion.

Autor

Carlos Rodríguez (CarlosRdrz)

4.3. Referencia del Archivo pr.h

Prototipos de las funciones principales del PR.

```
#include <pthread.h>
#include <time.h>
#include <curl/curl.h>
#include <curl/easy.h>
#include <sys/stat.h>
#include "func_tadq.h"
#include "pr_msg.h"
#include "pr_sm_smc.h"
#include "centralH.h"
#include "chargeH.h"
#include "utils.h"
```

Dependencia gráfica adjunta para pr.h:

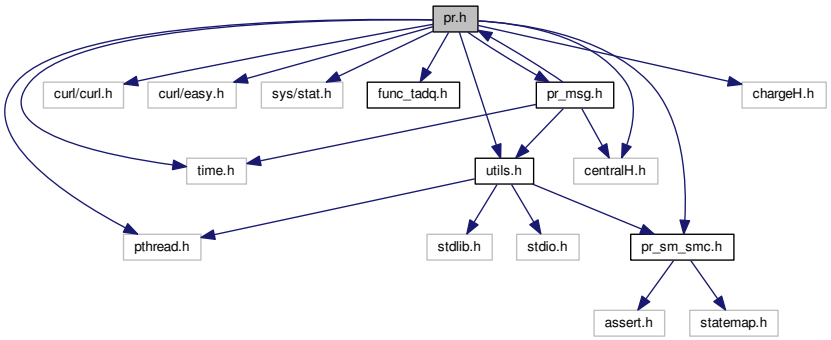
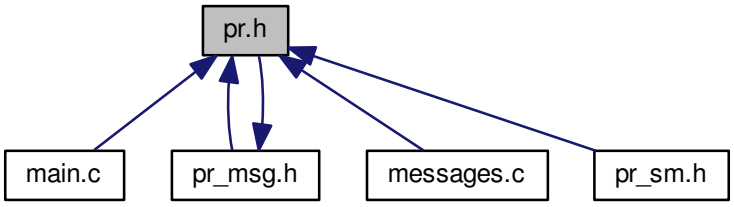


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Enumeraciones

- enum flags {
  Running = 1, ServerRunning = 2, Charging = 4, Paused = 8,
  Restart = 16 }

Funciones

- void pr_init (struct pr ∗this)

  Inicia la estructura PR (Constructor)
- void pr_destroy (struct pr ∗this)

  Libera los recursos del PR y finaliza su ejecucion (Destructor)
- void pr_server (struct pr ∗this)

  Hilo que crea el servidor SOAP donde el PR escuchara mensajes OCPP del CIR.
- void pr_electric (struct pr ∗this)

  Hilo que se encarga de comprobar las variables de la electronica y lanzar las transiciones necesarias cuando ocurra algun evento en ellas.
- void pr_do (struct pr ∗this)

  Hilo principal del PR. Se encarga de ejecutar una transicion de la cola de transiciones.
- void pr_heartbeat (struct pr ∗this)

Hilo que se encarga de enviar un Heartbeat cuando no se ha enviado ningun mensaje en los ultimos HEARTBEAT_INTERVAL segundos.

- void pr_input (struct pr *this)

  Hilo que se encarga de leer un caracter por el teclado y realizar la funcion correspondiente.
- void pr_sleep (struct pr *this)

  Esta funcion duerme el hilo en el que se ejecute durante 1 segundos.
- void pr_finish (struct pr *this)

  Termina el PR colocando la flag Running a 0 y llamando al destructor del PR.
- void pr_restart (struct pr *this)

  Coloca la flag de reinicio a 1 para que el main reinicie el proceso.
- void pr_update_firmware (struct pr *this, char *location)

  Descarga el firmware de la URL indicada y lo guarda en el ejecutable Posteriormente reinicia el PR para ejecutar el nuevo ejecutable.
- void pr_pause_finished (void *param)

  Se ejecuta en un timer cuando pasa el tiempo especificado de pausa para volver al estado normal de carga.

4.3.1. Descripción detallada

Prototipos de las funciones principales del PR. Contiene los prototipos de todas las funciones del objeto PR. Todas aceptan un parametro que es la esctructura del PR sobre el que se esta actuando

Autor

Carlos Rodríguez (CarlosRdrz)

4.3.2. Documentación de las enumeraciones

4.3.2.1. enum flags

Valores de enumeraciones

Running El PR esta iniciado y funcionando

ServerRunning El servidor SOAP OCPP del PR esta iniciado y funcionado

Charging El PR esta en estado de carga

Paused El PR esta en estado de carga pausada

Restart El PR debe ser reiniciado

```
31            {
32   Running = 1,
33   ServerRunning = 2,
34   Charging = 4,
35   Paused = 8,
36   Restart = 16
37 };
```

4.3.3. Documentación de las funciones

4.3.3.1. void pr_destroy (struct pr * *this*)

Libera los recursos del PR y finaliza su ejecucion (Destructor)

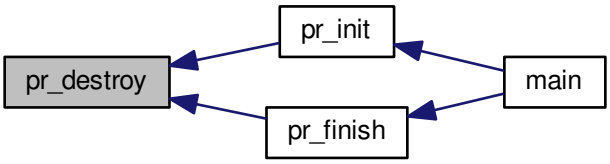
Libera todos los recursos utilizados por el PR, como las instancias de SOAP. Tambien espera a que finalicen los hilos y los destruye.

Parámetros

this	la estructura del pr

```
83 {
84   // Desactiva la bandera Running
85   this->flags &= ~Running;
86 
87   // Esperamos a que acaben todos los hilos
88   pthread_join(this->server_thread_tid, NULL);
89 #ifdef TEST_MODE
90   pthread_join(this->input_thread_tid, NULL);
91 #endif
92   pthread_join(this->heartbeat_thread_tid, NULL);
93   pthread_join(this->electric_thread_tid, NULL);
94 
95   // Liberamos recursos de gSOAP
96   soap_destroy(&this->soap_server);
97   soap_end(&this->soap_server);
98   soap_done(&this->soap_server);
99   soap_destroy(&this->soap_client);
100   soap_end(&this->soap_client);
101   soap_done(&this->soap_client);
102 
103   if (this->flags & Restart) printf("\n\n");
104 }
```

Gráfico de llamadas a esta función:



4.3.3.2. void pr_do (struct pr * *this*)

Hilo principal del PR. Se encarga de ejecutar una transicion de la cola de transiciones.

Parámetros

this	la estructura del pr

```
213 {
214   // Ejecutar los timers que correspondan
215   struct timer *current = this->timers;
216 
217   if (current != NULL) {
218     int timenow = time(NULL);
219 
220     // Si el timestamp se pasa ejecutamos la funcion
221     if (timenow > current->timestamp) {
222       current->funct(current->param);
223       delete_timer(&this->timers, current);
224     }
225 
226     // Pasamos al siguiente timer
227     current = current->next;
228   }
229 }
```

Gráfico de llamadas para esta función:

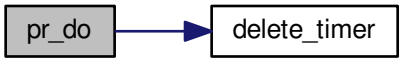


Gráfico de llamadas a esta función:

4.3.3.3. void pr_electric (struct pr * *this*)

Hilo que se encarga de comprobar las variables de la electronica y lanzar las transiciones necesarias cuando ocurra algun evento en ellas.

Parámetros

this	la estructura del pr

```
132 {
133   float digitalvars[NUM_ANALOG_IN];
134   memset(digitalvars, 0, sizeof(float) * NUM_ANALOG_IN);
135 
136   struct elecvars previous;
137   previous.cable = this->electric.cable;
138   previous.consumption = this->electric.consumption;
139   previous.charge = this->electric.charge;
140   previous.detectAC0 = this->electric.detectAC0;
141   previous.detectAC1 = this->electric.detectAC1;
142 
143   while (this->flags & Running) {
144     #ifndef TEST_MODE
145     // Leemos la entrada digital
146     lectura_analog(digitalvars);
147 
148     // Guardamos en los campos respectivos
149     this->electric.cable = digitalvars[10] > VOLTAGE_THRESHOLD ? 1 : 0;
150     this->electric.consumption = digitalvars[8] > VOLTAGE_THRESHOLD ? 1 : 0;
151     this->electric.charge = digitalvars[11] > VOLTAGE_THRESHOLD ? 1 : 0;
152     this->electric.detectAC0 = digitalvars[3] > VOLTAGE_THRESHOLD ? 1 : 0;
153     this->electric.detectAC1 = digitalvars[4] > VOLTAGE_THRESHOLD ? 1 : 0;
154     #endif
155 
156     // Comprobamos y lanzamos transiciones
157     if (previous.cable == 1 && this->electric.cable == 0) {
158       pr_sm_smc_TE_VDES(&this->pr_sm_smc);
159       this->digitalOut &= 0xFFBF;
160 
161       if (getState(&this->pr_sm_smc) == &MapaPrincipal_E_VDES) {
162         if (pr_sendStopTransaction(this)) {
163           pr_sm_smc_TE_FVDES(&this->pr_sm_smc);
164         } else {
165           pr_sm_smc_TE_FVDES0(&this->pr_sm_smc);
```

```
166         }
167       }
168     }
169
170     if (previous.consumption == 0 && this->electric.consumption == 1) {
171       pr_sm_smc_TE_CMAX(&this->pr_sm_smc);
172       this->digitalOut &= 0xFFBF;
173
174       if (getState(&this->pr_sm_smc) == &MapaPrincipal_E_CMAX) {
175         if (pr_sendStopTransaction(this)) {
176           pr_sm_smc_TE_FCMAX(&this->pr_sm_smc);
177         } else {
178           pr_sm_smc_TE_FCMAX0(&this->pr_sm_smc);
179         }
180       }
181     }
182
183     if (previous.charge == 0 && this->electric.charge == 1) {
184       pr_sm_smc_TE_CERO(&this->pr_sm_smc);
185       this->digitalOut &= 0xFFBF;
186
187       if (getState(&this->pr_sm_smc) == &MapaPrincipal_E_CERO) {
188         if (pr_sendStopTransaction(this)) {
189           pr_sm_smc_TE_FCERO(&this->pr_sm_smc);
190         } else {
191           pr_sm_smc_TE_FCERO0(&this->pr_sm_smc);
192         }
193       }
194     }
195
196     // Guardamos las nuevas como antiguas
197     previous.cable = this->electric.cable;
198     previous.consumption = this->electric.consumption;
199     previous.charge = this->electric.charge;
200     previous.detectAC0 = this->electric.detectAC0;
201     previous.detectAC1 = this->electric.detectAC1;
202
203     #ifndef TEST_MODE
204     modif_dato_digital(this->digitalOut);
205     #endif
206
207     // Dormimos 250 ms
208     usleep(250);
209   }
210 }
```

Gráfico de llamadas para esta función:

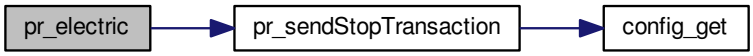


Gráfico de llamadas a esta función:

4.3.3.4. void pr_finish (struct pr * *this*)

Termina el PR colocando la flag Running a 0 y llamando al destructor del PR.

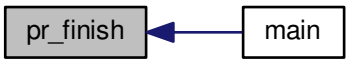
Parámetros

this	la estructura del pr

```
290 {
291   this->flags &= ~Running;
292   pr_destroy(this);
293 }
```

Gráfico de llamadas para esta función:

Gráfico de llamadas a esta función:



4.3.3.5. void pr_heartbeat (struct pr * *this*)

Hilo que se encarga de enviar un Heartbeat cuando no se ha enviado ningun mensaje en los ultimos HEARTBEAT_INTERVAL segundos.

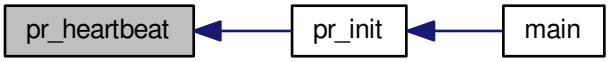
Parámetros

this	la estructura del pr

```
232 {
233   // Mientras este activa la bandera Running...
234   while (this->flags & Running) {
235     // Si hay un intervalo de heartbeat definido...
236     if (this->heartbeat_interval != -1) {
237       time_t time_now = time(NULL);
238       double diff = difftime(time_now, this->last_message_time);
239       // Si la diferencia de tiempo entre el ultimo mensaje y el tiempo
240       // actual es mayor que el intervalo de heartbeat, lo mandamos
241       // El 0.3 es un valor de seguridad, para dar un margen de error en el servidor
242       if (diff > (this->heartbeat_interval - 0.3)) {
243         pr_sendHeartbeat(this);
244       }
245     }
246     usleep(100);
247   }
248 }
```

Gráfico de llamadas para esta función:

Gráfico de llamadas a esta función:



4.3.3.6. void pr_init (struct pr * *this*)

Inicia la estructura PR (Constructor)

Inicia las variables y estructuras necesarias para el PR, como la maquina de estados, las instancias de SOAP, las banderas, la cola de transiciones, los hilos y lanza la primera transicion de la maquina de estados (La transicion Start) para comenzar el funcionamiento del PR.

Parámetros

this	la estructura del pr

```
6 {
7   // Inicializa la maquina de estados
8   // Se inicializa en el estado "Stopped"
9   pr_sm_smc_Init(&this->pr_sm_smc, this);
10   FSM_STACK(&this->pr_sm_smc, this->AppStack);
11
12   // Inicializa las instancias de soap
13   soap_init(&this->soap_client);
14   soap_set_namespaces(&this->soap_client, central_namespaces);
15   soap_init(&this->soap_server);
16   soap_set_namespaces(&this->soap_server, charge_namespaces);
17
18   this->soap_server.accept_timeout = 1;             // Timeout 1 sec
19   this->soap_server.bind_flags |= SO_REUSEADDR;     // Reutiliza el puerto de usos anteriores
20   this->soap_server.user = (void *)this;            // Permite acceder al objeto desde los mensajes de soap
21
22   // Variables de objeto y banderas
23   this->flags = Running;                            // Banderas del programa
24   this->config = config_load("config");              // Cargar archivo de configuracion
25   this->last_message_time = time(NULL);             // Tiempo desde el ultimo mensaje enviado
26   this->heartbeat_interval = -1;                    // Intervalo en segundos de heartbeat
27   this->timers = NULL;
28   this->id = atoi(config_get(this->config, "PR_ID"));
29
30   // Variables electricas
31   this->electric.cable = 0;
32   this->electric.consumption = 0;
33   this->electric.charge = 0;
34   this->electric.detectAC0 = 0;
35   this->electric.detectAC1 = 0;
36
37 #ifdef TEST_MODE
```

```
38   this->electric.detectAC0 = 1;
39   this->electric.charge = 1;
40 #endif
41
42   // Iniciar tarjeta
43 #ifndef TEST_MODE
44   this->digitalOut = 0;
45
46   if (inicia_tadq() < 0) {
47     printf ("[ERROR] TARJETA ADQ FAIL\n");
48     exit(1);
49   } else {
50     printf ("[INFO] TARJETA ADQ OK\n");
51   }
52 #endif
53
54   // Crea los diferentes hilos uno de servidor, y otro para el heartbeat
55   pthread_create(&this->server_thread_tid, NULL, (void*(*)(void*))pr_server, (void *)this);
56   pthread_create(&this->heartbeat_thread_tid, NULL, (void*(*)(void*))
      pr_heartbeat, (void *)this);
57   pthread_create(&this->electric_thread_tid, NULL, (void*(*)(void*))pr_electric, (void *)this);
58
59   // Solo crear el hilo de lectura de teclado si estamos en modo test
60 #ifdef TEST_MODE
61   pthread_create(&this->input_thread_tid, NULL, (void*(*)(void*))pr_input, (void *)this);
62 #endif
63
64   // Esperar hasta que se active la bandera ServerRunning
65   while((this->flags & ServerRunning) == 0) { usleep(250); }
66
67   // Lanzar BootNotification y si falla salimos
68   int result = 0;
69   int tries = 0;
70   while (result != 1 && tries < MAX_CONN_TRIES) {
71     if (tries != 0) sleep(1);
72     result = pr_sendBootNotification(this);
73     tries++;
74   }
75
76   if (!result) {
77     printf("[ERROR] CIR doesnt respond to BootNotification. Exiting...\n");
78     pr_destroy(this);
79   }
80 }
```

Gráfico de llamadas para esta función:

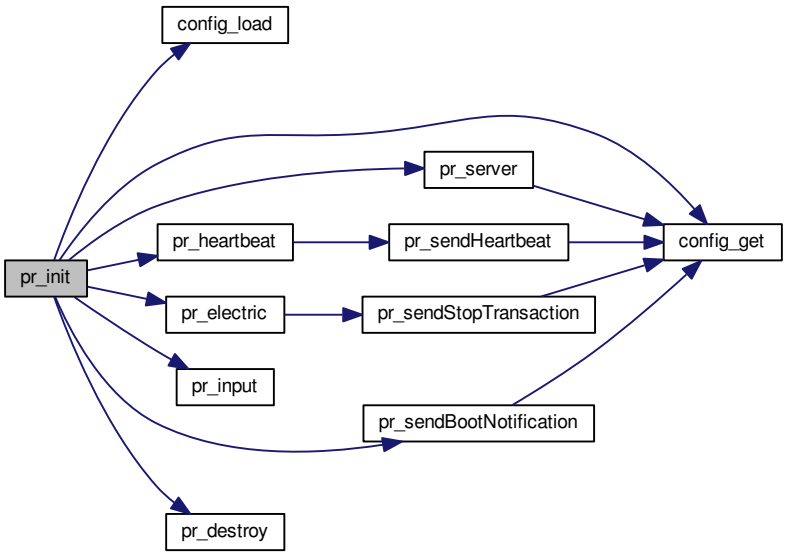


Gráfico de llamadas a esta función:

4.3.3.7. void pr_input (struct pr * *this*)

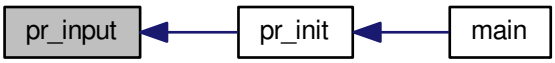
Hilo que se encarga de leer un caracter por el teclado y realizar la funcion correspondiente.

Parámetros

this	la estructura del pr

```
252                                {
253   int read;
254
255   // printf("[INFO] Press 'C' to start a local charge operation\n");
256   // Mientras este activa la bandera Running...
257   while (this->flags & Running) {
258     // Leer caracter de la entrada estandar
259     read = getchar();
260     // Si ese caracter es 'C', lanzar transicion 'Charge'
261     // if (read == 'C') transition_queue_add(&this->transition_queue, pr_sm_smc_RemoteCharge);
262     if (read == 'C') {
263       this->electric.cable = 1;
264       printf("[DBG] Cable conectado\n");
265     } else if (read == 'c') {
266       this->electric.cable = 0;
267       printf("[DBG] Cable desconectado\n");
268     } else if (read == 'O') {
269       this->electric.consumption = 1;
270       printf("[DBG] Consumo maximo alcanzado\n");
271     } else if (read == 'o') {
272       this->electric.consumption = 0;
273       printf("[DBG] Consumo maximo NO alcanzado\n");
274     } else if (read == 'H') {
275       this->electric.charge = 1;
276       printf("[DBG] CP Cesa entrega de potencia\n");
277     } else if (read == 'h') {
278       this->electric.charge = 0;
279       printf("[DBG] CP NO cesa entrega de potencia\n");
280     }
281   }
282 }
```

Gráfico de llamadas a esta función:



4.3.3.8. void pr_pause_finished (void * *param*)

Se ejecuta en un timer cuando pasa el tiempo especificado de pausa para volver al estado normal de carga.

Parámetros

this	la estructura del pr

Devuelve

void

```
327 {
328   struct pr * this = (struct pr *) param;
329   printf("[INFO] Pause finished\n");
330   pr_sm_smc_TE_EFC(&this->pr_sm_smc);
331 }
```

4.3.3.9. void pr_restart (struct pr * *this*)

Coloca la flag de reinicio a 1 para que el main reinicie el proceso.

Parámetros

this	la estructura del pr

```
296 {
297   printf("[INFO] Restarting PR...\n");
298   this->flags |= Restart;
299 }
```

4.3.3.10. void pr_server (struct pr * *this*)

Hilo que crea el servidor SOAP donde el PR escuchara mensajes OCPP del CIR.

Obtiene el puerto donde escuchar del archivo de configuracion en la propiedad LISTEN_PORT. Si no consigue crear el servidor lanza una transicion en la maquina de estados que finaliza el programa.

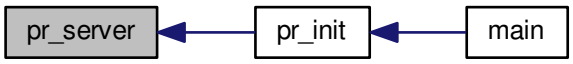
Parámetros

this	la estructura del pr

```
107 {
108   // Lee el puerto del archivo de configuracion
109   int port = atoi(config_get(this->config, "LISTEN_PORT"));
110 
111   // Si se consigue bindear el socket...
112   if (soap_valid_socket(soap_bind(&this->soap_server, NULL, port, 100))) {
113     printf("[INFO] Server started on port %d\n", port);
114     // Activar la bandera ServerRunning
115     this->flags |= ServerRunning;
116     // Mientras este activa la bandera Running...
117     while (this->flags & Running) {
118       // Servir la peticion si es correcta
119       if (soap_valid_socket(soap_accept(&this->soap_server))) {
120         charge_serve(&this->soap_server);
121         soap_destroy(&this->soap_server);
122         soap_end(&this->soap_server);
123       }
124     }
125   }
126 
127   // Servidor finalizado
128   printf("[INFO] Server finished\n");
129 }
```

Gráfico de llamadas para esta función:

Gráfico de llamadas a esta función:



4.3.3.11. void pr_sleep (struct pr * *this*)

Esta funcion duerme el hilo en el que se ejecute durante 1 segundos.

Parámetros

this	la estructura del pr

```
285 {
286   sleep(1);
287 }
```

4.3.3.12. void pr_update_firmware (struct pr * *this*, char * *location*)

Descarga el firmware de la URL indicada y lo guarda en el ejecutable Posteriormente reinicia el PR para ejecutar el nuevo ejecutable.

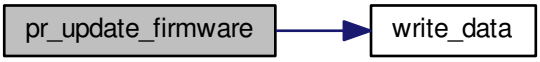
Parámetros

this	la estructura del pr
location	URL con el nuevo firmware

```
302 {
303   CURL *curl;
304   FILE *fp;
305   char path[PATH_MAX];
306   readlink("/proc/self/exe", path, PATH_MAX);
307   curl = curl_easy_init();
308
309   if (curl) {
310     unlink(path);
311     fp = fopen(path, "wb");
312     curl_easy_setopt(curl, CURLOPT_URL, location);
313     curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, write_data);
314     curl_easy_setopt(curl, CURLOPT_WRITEDATA, fp);
315     curl_easy_perform(curl);
```

```
316     curl_easy_cleanup(curl);
317     fclose(fp);
318
319     int i = strtol("0755", 0, 8);
320     if (chmod(path, i) < 0) {
321       printf("[ERROR] Error calling chmod 755 to the downloaded firmware update.\nYou should do it manually
      before executing the new file.\n");
322     }
323   }
324 }
```

Gráfico de llamadas para esta función:



4.4. Referencia del Archivo pr.sm

Fichero que describe la ME asociada al PR.

4.4.1. Descripción detallada

Fichero que describe la ME asociada al PR. M. R. Arahal, J. M. Maestre, C. Rodriguez

Versión

1.0

Objetivos: Se usa la ME para saber que mensajes gsoap enviar en cada momento. En general, la maquina proporciona respuestas de la siguiente forma:

```
respuesta=f(estimulo,estado)
```

donde la funcion f representa una funcion en sentido amplio, es decir, no debe ser asimilada con ninguna funcion en C. Estimulo se refiere a cualquier peticion que requiere una respuesta por parte de la ME. Finalmente, con estado nos referimos al estado actual de la maquina de estados.

Las transiciones en la maquina de estados se producen cuando se verifican determinadas condiciones de transito, que son comprobadas por una funcion de transito (FT).

Para cada nodo de este archivo, se indicara:

- Sit: Situacion que describe el estado
- AE: Actuaciones de entrada al estado
- NDT: Nodo destino de una transicion
- LFT: Ente que lanza la funcion de transicion
- FT: Funcion de transicion
- CGT: Condiciones de guarda de la transicion

Los estados de la ME son:

- Estado E0: La ME comienza en este estado
  - Sit: Estado de espera del PR
  - AE: -
  - NDT, LFT, FT, CGT:
    - E_CPR , RemoteStartTransaction.req , TE_EFC() , -
  - Nota: El PPR_MG responde con RemoteStartTransaction.conf incluyendo la situacion del PR (disponible o no para carga). La situacion es disponible si y solo si se verifica:
    - Cable conectado MCP.IndCable = 1
    - PR tiene potencia EPR.Detect_AC0 + EPR.Detect_AC1 = 1
    - Existe señal carga del modulo CP MCP.IntCarga = 1
- Estado E_EFC
  - Sit: El PR puede comenzar la cesion de potencia al VE y esperar el fin de carga
  - AE: Se configura CP y se activa EPR.IntCarga = 1
  - NDT, LFT, FT, CGT:
    - E_FCTP , RemoteStopTransaction.req , TE_FCIR , - (el CIR ordena terminar)
    - E_PAUS , DataTransfer.req pausa , TE_PAUS , - (recibida pausa en entrega de potencia)
    - E_VDES , PPR_MG (MCP.IndCable = 0) , TE_VDES , - (desconexion del lado del VE)
    - E_CMAX , PPR_MG (MCP.Consumo $>$ CMax), TE_CMAX , - (consumo maximo)
    - E_CERO , PPR_MG (MCP.carga = 0) , TE_CERO , - (el CP ha cesado la entrega de potencia)
  - Nota: El PPR_MG responde con RemoteStopTransaction.conf
- Estado E_FCIR
  - Sit: Finalizacion de recarga motivada por el CIR (aborto o tiempo maximo planificado)
  - AE:
    - Se desactiva interruptor de carga EPR.IntCarga = 0
    - Se suelta cable EPR.CP_Soltar = 1
    - Se modifican valores de la variable local de estado ultima recarga
    - Se avisa a PCIR_MGCM de situacion terminal mediante V_CF_MA
  - NDT, LFT, FT, CGT:
    - E_0 , PPR_MG (V_CF_MA !=0) , TE_E0() , -
  - Nota: -
- Estado E_PAUS
  - Sit: El CIR indica que se realice pausa en la recarga actual
  - AE:
    - Se desactiva interruptor de carga EPR.IntCarga = 0
    - Se pone en marcha cronÛmetro que avise del fin de pausa
  - NDT, LFT, FT, CGT:
    - E_EFC , CronÛmetro pausa , TE_EFC() , -
  - Nota: -
- Estado E_VDES
  - Sit: Finalizacion de recarga motivada en el PR por desconexion VE
  - AE:
    - Se desactiva interruptor de carga EPR.IntCarga = 0

        - Se suelta cableEPR.CP_Soltar = 1
    - NDT, LFT, FT, CGT:
        - E_FVDES , PPR_MG (mens.conf) , TE_FVDES() , -
        - E_FVDES0, PPR_MG (timeout) , TE_FVDES0() , -
    - Nota:
        - El PPR_MG se encarga de enviar OCPP StopTransaction.req y recibir .conf
- Estado E_FVDES
    - Sit: Fin de maniobra con aviso al CIR correcto
    - AE:
        - Se modifican valores de la variable local de estado ultima recarga
        - Se avisa a PCIR_MGCM de situacion terminal mediante V_CF_MA
    - NDT, LFT, FT, CGT:
        - E_0 , PPR_MG (V_CF_MA !=0) , TE_E0() , -
    - Nota: -
- Estado E_FVDES0
    - Sit: Fin de maniobra con aviso al CIR insatisfactorio
    - AE:
        - Se modifican valores de la variable local de estado ultima recarga
        - Se avisa a PCIR_MGCM de situaciÛn terminal mediante V_CF_MA
    - NDT, LFT, FT, CGT:
        - E_0 , PPR_MG (V_CF_MA !=0) , TE_E0() , -
    - Nota: -
- Estado E_CMAX
    - Sit: Finalizacion de recarga motivada en el PR por consumo maximo
    - AE:
        - Se desactiva interruptor de carga EPR.IntCarga = 0
        - Se suelta cableEPR.CP_Soltar = 1
        - Se modifican valores de la variable local de estado ultima recarga
    - NDT, LFT, FT, CGT:
        - E_FCMAX , PPR_MG (mens.conf) , TE_FCMAX() , -
        - E_FCMAX0, PPR_MG (timeout) , TE_FCMAX0() , -
    - Nota: El PPR_MG se encarga de enviar OCPP StopTransaction.req y recibir .conf
- Estado E_FCMAX
    - Sit: FFin de maniobra con aviso al CIR correcto
    - AE:
        - Se modifican valores de la variable local de estado ultima recarga
        - Se avisa a PCIR_MGCM de situaciÛn terminal mediante V_CF_MA
    - NDT, LFT, FT, CGT:
        - E_0 , PPR_MG (V_CF_MA !=0) , TE_E0() , -
    - Nota: -
- Estado E_FCMAX0
    - Sit: Fin de maniobra con aviso al CIR insatisfactorio
    - AE:

- Se modifican valores de la variable local de estado ultima recarga
- Se avisa a PCIR_MGCM de situaciÛn terminal mediante V_CF_MA

- NDT, LFT, FT, CGT:
  - E_0 , PPR_MG (V_CF_MA !=0) , TE_E0() , -
- Nota: -

- Estado E_CERO
  - Sit: Finalizacion de recarga motivada en el PR porque el CP no cede mas
  - AE:
    - Se desactiva interruptor de carga EPR.IntCarga = 0
    - Se suelta cableEPR.CP_Soltar = 1
    - Se modifican valores de la variable local de estado ultima recarga
  - NDT, LFT, FT, CGT:
    - E_FCERO , PPR_MG (mens.conf) , TE_FCERO() , -
    - E_FCERO0, PPR_MG (timeout) , TE_FCERO0() , -
  - Nota: El PPR_MG se encarga de enviar OCPP StopTransaction.req y recibir .conf
- Estado E_FCERO
  - Sit: Fin de maniobra con aviso al CIR correcto
  - AE:
    - Se modifican valores de la variable local de estado ultima recarga
    - Se avisa a PCIR_MGCM de situaciÛn terminal mediante V_CF_MA
  - NDT, LFT, FT, CGT:
    - E_0 , PPR_MG (V_CF_MA !=0) , TE_E0() , -
  - Nota: -
- Estado E_FCERO0
  - Sit: Fin de maniobra con aviso al CIR insatisfactorio
  - AE:
    - Se modifican valores de la variable local de estado ultima recarga
    - Se avisa a PCIR_MGCM de situaciÛn terminal mediante V_CF_MA
  - NDT, LFT, FT, CGT:
    - E_0 , PPR_MG (V_CF_MA !=0) , TE_E0() , -
  - Nota: -

Atención

Notese que las transiciones en la maquina de estados se producen cuando se verifican determinadas condiciones de transito, que son comprobadas por una funcion de transito (FT), cuya definicion no esta contenida en este fichero.

4.5. Referencia del Archivo pr_msg.h

Prototipos de las funciones de envio de mensajes OCPP.

```
#include <time.h>
#include "pr.h"
#include "utils.h"
#include "centralH.h"
```

Dependencia gráfica adjunta para pr_msg.h:

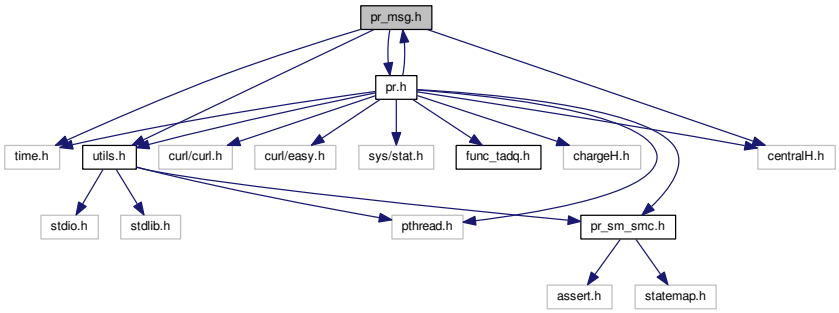
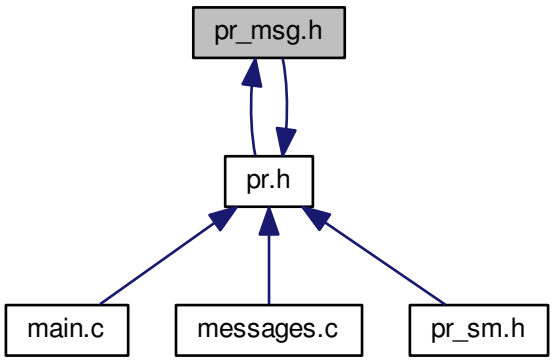


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Funciones

- int pr_sendBootNotification (struct pr ∗this)

  Envia un mensaje BootNotification.
- int pr_sendHeartbeat (struct pr ∗this)

  Envia un mensaje Heartbeat.
- int pr_sendAuthorize (struct pr ∗this)

  Envia un mensaje Authorize.
- int pr_sendStatusNotification (struct pr ∗this)

  Envia un mensaje StatusNotification.
- int pr_sendStartTransaction (struct pr ∗this)

  Envia un mensaje StartTransaction.
- int pr_sendStopTransaction (struct pr ∗this)

  Envia un mensaje StopTransaction.

4.5.1. Descripción detallada

Prototipos de las funciones de envio de mensajes OCPP. Contiene los prototipos de todas las funciones del PR que se utilizan para enviar mensajes SOAP del protocolo OCPP al CIR. Cada funcion envia un mensaje del tipo especificado en el nombre de la funcion. Todas siguen la misma plantilla, pero los campos a enviar en los diferentes mensajes son distintos entre ellas

Autor

Carlos Rodríguez (CarlosRdrz)

4.5.2. Documentación de las funciones

4.5.2.1. int pr_sendAuthorize (struct pr * *this*)

Envia un mensaje Authorize.

Parámetros

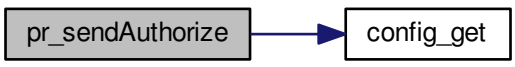
this	la estructura del pr

Devuelve

1 si todo fue correctamente 0 si hubo algun error

```
60 {
61   printf("--> Authorize Request to CIR\n");
62   struct cir__AuthorizeRequest request;
63   struct cir__AuthorizeResponse response;
64   int result = 0;
65 
66   request.idTag = "1";
67 
68   if (soap_call___cir__Authorize(&this->soap_client, config_get(this->config, "CIR_URL"), NULL, &
      request, &response) == SOAP_OK) {
69     printf("<-- Authorize Response from CIR with parentId: %s\n", response.idTagInfo->parentIdTag);
70     result = 1;
71   } else {
72     printf("[ERROR] The CIR is not responding\n");
73   }
74 
75   // Actualizar el tiempo de ultimo mensaje enviado
76   this->last_message_time = time(NULL);
77 
78   return result;
79 }
```

Gráfico de llamadas para esta función:



4.5.2.2. int pr_sendBootNotification (struct pr * *this*)

Envia un mensaje BootNotification.

Parámetros

this	la estructura del pr

Devuelve

1 si todo fue correctamente 0 si hubo algun error

```
4 {
5   struct cir__BootNotificationRequest request;
6   struct cir__BootNotificationResponse response;
7   int result = 0;
8
9   printf("--> BootNotification Request to CIR at %s calling with PR ID %s\n",
10     config_get(this->config, "CIR_URL"), config_get(this->config, "PR_ID"));
11
12   // Campos del mensaje BootNotification
13   request.chargePointVendor = config_get(this->config, "VENDOR");
14   request.chargePointModel = config_get(this->config, "VERSION");
15   request.chargePointSerialNumber = NULL; // NULL es para especificar que el parametro es opcion y no se
       envia
16   request.chargeBoxSerialNumber = NULL;
17   request.firmwareVersion = NULL;
18   request.iccid = config_get(this->config, "PR_ID");
19   request.imsi = NULL;
20   request.meterType = NULL;
21   request.meterSerialNumber = NULL;
22
23   if (soap_call___cir__BootNotification(&this->soap_client, config_get(this->config, "CIR_URL"),
      NULL, &request, &response) == SOAP_OK) {
24     // Mensaje recibido correctamente
25     printf("<-- BootNotification Response from CIR. Heartbeat Interval: %d seconds\n", response.
      heartbeatInterval);
26     // Ahora el heartbeat_interval es el especificado en la respuesta del CIR
27     this->heartbeat_interval = response.heartbeatInterval;
28     result = 1;
29   } else {
30     printf("[ERROR] The CIR is not responding\n");
31   }
32
33   // Actualizar el tiempo de ultimo mensaje enviado
34   this->last_message_time = time(NULL);
35
36   return result;
37 }
```

Gráfico de llamadas para esta función:

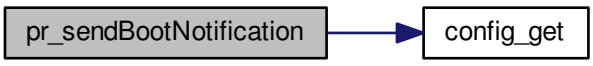


Gráfico de llamadas a esta función:

4.5.2.3. int pr_sendHeartbeat (struct pr * *this*)

Envia un mensaje Heartbeat.

Parámetros

this	la estructura del pr

Devuelve

1 si todo fue correctamente 0 si hubo algun error

```
40 {
41   printf("--> Heartbeat Request to CIR\n");
42   struct cir__HeartbeatRequest request;
43   struct cir__HeartbeatResponse response;
44   int result = 0;
45 
46   if (soap_call___cir__Heartbeat(&this->soap_client, config_get(this->config, "CIR_URL"), NULL, &
      request, &response) == SOAP_OK) {
47     // printf("<-- Heartbeat Response from CIR\n");
48     result = 1;
49   } else {
50     printf("[ERROR] The CIR is not responding\n");
51   }
52 
53   // Actualizar el tiempo de ultimo mensaje enviado
54   this->last_message_time = time(NULL);
55 
56   return result;
57 }
```

Gráfico de llamadas para esta función:

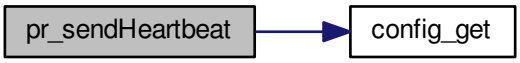
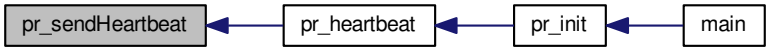


Gráfico de llamadas a esta función:



4.5.2.4. int pr_sendStartTransaction (struct pr * *this*)

Envia un mensaje StartTransaction.

Parámetros

this	la estructura del pr

Devuelve

1 si todo fue correctamente 0 si hubo algun error

```
110 {
111   printf("--> StartTransaction Request to CIR\n");
112   struct cir__StartTransactionRequest request;
113   struct cir__StartTransactionResponse response;
114   int result = 0;
115
116   request.connectorId = 1;
117   request.idTag = "1";
118   request.timestamp = time(0);
119   request.meterStart = 0;
120   request.reservationId = NULL;
121
122   if (soap_call___cir__StartTransaction(&this->soap_client, config_get(this->config, "CIR_URL"),
      NULL, &request, &response) == SOAP_OK) {
123     // printf("<-- StartTransaction Response from CIR\n");
124     result = 1;
125   } else {
126     printf("[ERROR] The CIR is not responding\n");
127   }
128
129   // Actualizar el tiempo de ultimo mensaje enviado
130   this->last_message_time = time(NULL);
131
132   return result;
133 }
```

Gráfico de llamadas para esta función:

4.5.2.5. int pr_sendStatusNotification (struct pr * *this*)

Envia un mensaje StatusNotification.

Parámetros

this	la estructura del pr

Devuelve

1 si todo fue correctamente 0 si hubo algun error

```
82 {
83   printf("--> StatusNotification Request to CIR\n");
84   struct cir__StatusNotificationRequest request;
85   struct cir__StatusNotificationResponse response;
86   int result = 0;
87
88   request.connectorId = 1;
89   request.status = cir__ChargePointStatus__Available;
90   request.errorCode = cir__ChargePointErrorCode__NoError;
91   request.info = NULL;
92   request.timestamp = NULL;
93   request.vendorId = NULL;
94   request.vendorErrorCode = NULL;
95
96   if (soap_call___cir__StatusNotification(&this->soap_client, config_get(this->config, "CIR_URL")
      , NULL, &request, &response) == SOAP_OK) {
97     // printf("<-- StatusNotification Response from CIR\n");
```

```
98     result = 1;
99   } else {
100     printf("[ERROR] The CIR is not responding\n");
101   }
102
103   // Actualizar el tiempo de ultimo mensaje enviado
104   this->last_message_time = time(NULL);
105
106   return result;
107 }
```

Gráfico de llamadas para esta función:

4.5.2.6. int pr_sendStopTransaction (struct pr * *this*)

Envia un mensaje StopTransaction.

Parámetros

this	la estructura del pr

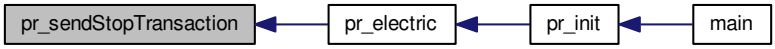
Devuelve

1 si todo fue correctamente 0 si hubo algun error

```
136 {
137   printf("--> StopTransaction Request to CIR\n");
138   struct cir__StopTransactionRequest request;
139   struct cir__StopTransactionResponse response;
140   int result = 0;
141
142   request.transactionId = 1;
143   request.idTag = "1";
144   request.timestamp = time(0);
145   request.meterStop = 10;
146   request.__sizetransactionData = 0;
147   request.transactionData = NULL;
148
149   if (soap_call___cir__StopTransaction(&this->soap_client, config_get(this->config, "CIR_URL"),
      NULL, &request, &response) == SOAP_OK) {
150     // printf("<-- StopTransaction Response from CIR\n");
151     result = 1;
152   } else {
153     printf("[ERROR] The CIR is not responding\n");
154   }
155
156   // Actualizar el tiempo de ultimo mensaje enviado
157   this->last_message_time = time(NULL);
158
159   return result;
160 }
```

Gráfico de llamadas para esta función:

Gráfico de llamadas a esta función:

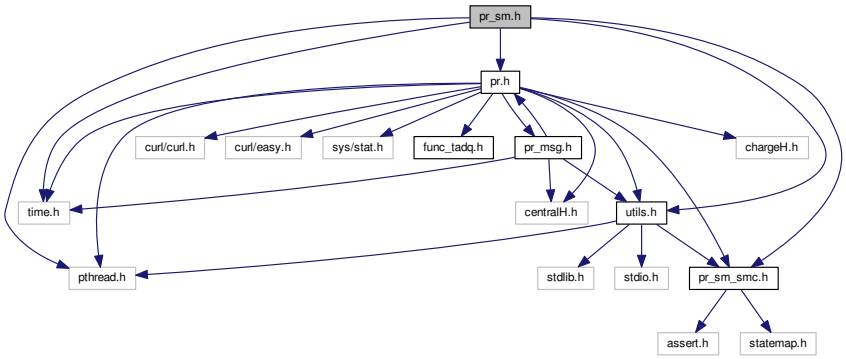


4.6. Referencia del Archivo pr_sm.h

Prototipos de las funciones de la SM del PR.

```
#include <pthread.h>
#include <time.h>
#include "pr.h"
#include "pr_sm_smc.h"
#include "utils.h"
```

Dependencia gráfica adjunta para pr_sm.h:



Funciones

- void pr_E_EFC_entry (struct pr ∗this)

  Funcion que se ejecuta al entrar al estado EFC (Espera fin de Carga)
- void pr_E_FCIR_entry (struct pr ∗this)

  Funcion que se ejecuta al entrar al estado FCIR (Finalización de recarga motivada por el CIR)

- void pr_E_PAUS_entry (struct pr ∗this)

  Funcion que se ejecuta al entrar al estado PAUS (Pausa de carga)
- void pr_E_VDES_entry (struct pr ∗this)

  Funcion que se ejecuta al entrar al estado VDES (Finalización de recarga motivada en el PR por desconexión VE)
- void pr_E_FVDES_entry (struct pr ∗this)

  Funcion que se ejecuta al entrar al estado FVDES (Fin de maniobra por desconexion VE con aviso al CIR correcto)
- void pr_E_FVDES0_entry (struct pr ∗this)

  Funcion que se ejecuta al entrar al estado FVDES0 (Fin de maniobra por desconexion VE con aviso al CIR incorrecto)
- void pr_E_CMAX_entry (struct pr ∗this)

  Funcion que se ejecuta al entrar al estado CMAX (Finalizacion de recarga motivada en el PR por consumo maximo)
- void pr_E_FCMAX_entry (struct pr ∗this)

  Funcion que se ejecuta al entrar al estado FCMAX (Fin de maniobra por consumo maximo con aviso al CIR correcto)
- void pr_E_FCMAX0_entry (struct pr ∗this)

  Funcion que se ejecuta al entrar al estado FCMAX0 (Fin de maniobra por consumo maximo con aviso al CIR incorrecto)
- void pr_E_CERO_entry (struct pr ∗this)

  Funcion que se ejecuta al entrar al estado CERO (Finalizacion de recarga motivada en el PR porque el CP no cede mas)
- void pr_E_FCERO_entry (struct pr ∗this)

  Funcion que se ejecuta al entrar al estado FCERO (Fin de maniobra motivada en el PR porque el CP no cede mas con aviso al CIR correcto)
- void pr_E_FCERO0_entry (struct pr ∗this)

  Funcion que se ejecuta al entrar al estado FCERO0 (Fin de maniobra motivada en el PR porque el CP no cede mas con aviso al CIR incorrecto)
- void pr_no_permitido (struct pr ∗this)

  Muestra un mensaje cuando se intenta lanzar una transicion no definida en algun estado.

4.6.1. Descripción detallada

Prototipos de las funciones de la SM del PR. Contiene los prototipos de las funciones relacionadas con la maquina de estados del PR.

Autor

Carlos Rodríguez (CarlosRdrz)

4.6.2. Documentación de las funciones

4.6.2.1. void pr_E_CERO_entry (struct pr ∗ *this*)

Funcion que se ejecuta al entrar al estado CERO (Finalizacion de recarga motivada en el PR porque el CP no cede mas)

Parámetros

this	la estructura del pr

```
49 {
50   NEW_STATE("CERO");
51 }
```

4.6.2.2. void pr_E_CMAX_entry (struct pr ∗ *this*)

Funcion que se ejecuta al entrar al estado CMAX (Finalizacion de recarga motivada en el PR por consumo maximo)

Parámetros

this	la estructura del pr

```
34 {
35   NEW_STATE("CMAX");
36 }
```

4.6.2.3. void pr_E_EFC_entry (struct pr * *this*)

Funcion que se ejecuta al entrar al estado EFC (Espera fin de Carga)

Parámetros

this	la estructura del pr

```
4 {
5   NEW_STATE("EFC");
6 }
```

4.6.2.4. void pr_E_FCERO0_entry (struct pr * *this*)

Funcion que se ejecuta al entrar al estado FCERO0 (Fin de maniobra motivada en el PR porque el CP no cede mas con aviso al CIR incorrecto)

Parámetros

this	la estructura del pr

```
59 {
60   NEW_STATE("FCERO0");
61 }
```

4.6.2.5. void pr_E_FCERO_entry (struct pr * *this*)

Funcion que se ejecuta al entrar al estado FCERO (Fin de maniobra motivada en el PR porque el CP no cede mas con aviso al CIR correcto)

Parámetros

this	la estructura del pr

```
54 {
55   NEW_STATE("FCERO");
56 }
```

4.6.2.6. void pr_E_FCIR_entry (struct pr * *this*)

Funcion que se ejecuta al entrar al estado FCIR (Finalización de recarga motivada por el CIR)

Parámetros

this	la estructura del pr

```
9 {
10   NEW_STATE("FCIR");
11 }
```

4.6.2.7. void pr_E_FCMAX0_entry (struct pr * *this*)

Funcion que se ejecuta al entrar al estado FCMAX0 (Fin de maniobra por consumo maximo con aviso al CIR incorrecto)

Parámetros

this	la estructura del pr

```
44 {
45   NEW_STATE("FCMAX0");
46 }
```

4.6.2.8. void pr_E_FCMAX_entry (struct pr * *this*)

Funcion que se ejecuta al entrar al estado FCMAX (Fin de maniobra por consumo maximo con aviso al CIR correcto)

Parámetros

this	la estructura del pr

```
39 {
40   NEW_STATE("FCMAX");
41 }
```

4.6.2.9. void pr_E_FVDES0_entry (struct pr * *this*)

Funcion que se ejecuta al entrar al estado FVDES0 (Fin de maniobra por desconexion VE con aviso al CIR incorrecto)

Parámetros

this	la estructura del pr

```
29 {
30   NEW_STATE("FVDES0");
31 }
```

4.6.2.10. void pr_E_FVDES_entry (struct pr * *this*)

Funcion que se ejecuta al entrar al estado FVDES (Fin de maniobra por desconexion VE con aviso al CIR correcto)

Parámetros

this	la estructura del pr

```
24 {
25   NEW_STATE("FVDES");
26 }
```

4.6.2.11. void pr_E_PAUS_entry (struct pr * *this*)

Funcion que se ejecuta al entrar al estado PAUS (Pausa de carga)

Parámetros

this	la estructura del pr

```
14 {
15   NEW_STATE("PAUS");
16 }
```

4.6.2.12. void pr_E_VDES_entry (struct pr * *this*)

Funcion que se ejecuta al entrar al estado VDES (Finalización de recarga motivada en el PR por desconexión VE)

Parámetros

this	la estructura del pr

```
19 {
20   NEW_STATE("VDES");
21 }
```

4.6.2.13. void pr_no_permitido (struct pr * *this*)

Muestra un mensaje cuando se intenta lanzar una transicion no definida en algun estado.

Parámetros

this	la estructura del pr

```
64 {
65   printf("[WARN] Received non permitted transition\n");
66 }
```

4.7. Referencia del Archivo utils.h

Prototipos de las funciones de utilidad.

```
#include <pthread.h>
#include <stdlib.h>
#include <stdio.h>
#include "pr_sm_smc.h"
```

Dependencia gráfica adjunta para utils.h:

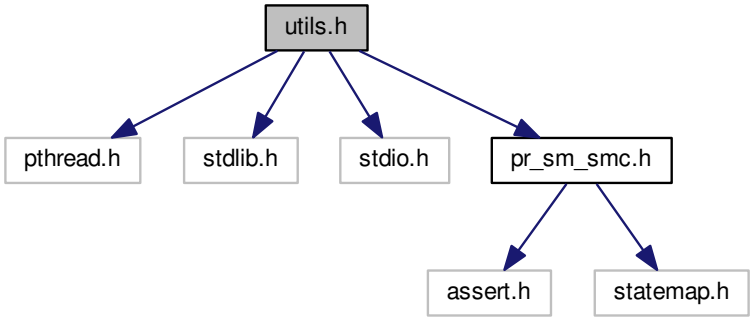
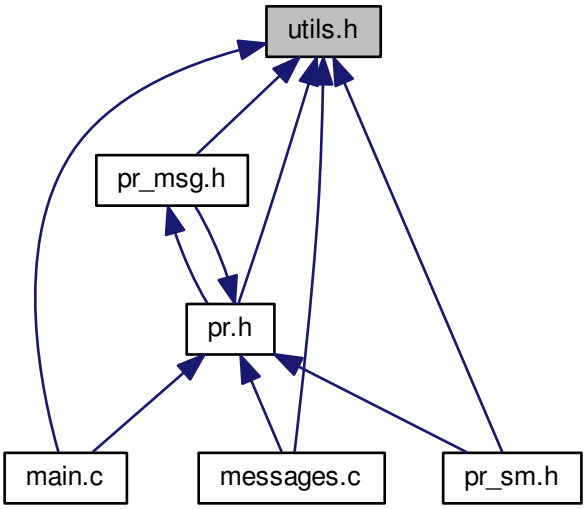


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Estructuras de datos

- struct config_node
- struct timer

Funciones

- struct config_node * config_load (const char *file_name)

  Carga un archivo de configuracion.
- char * config_get (struct config_node *config, const char *key)

  Obtiene una propiedad de una lista de propiedades de un archivo de configuracion.
- void config_set (struct config_node *config, const char *key, char *value)

  Establece una propiedad en la lista de propiedades.
- void config_save (struct config_node *config)

  Guarda las propiedades de configuracion en el archivo del disco.
- void config_destroy (struct config_node *config)

  Libera recursos utilizados por los nodos de propiedades que se crearon al cargar un archivo de configuracion.
- size_t write_data (void *ptr, size_t size, size_t nmemb, FILE *stream)

  Escribe en ptr los datos leidos por el stream.
- void new_timer (struct timer **cola, void(*fun)(void *), void *param, int cuando)

  Especifica una funcion a ejecutar en un instante determinado.
- void delete_timer (struct timer **cola, struct timer *to_delete)

  Borra un timer de la cola de timers.

4.7.1. Descripción detallada

Prototipos de las funciones de utilidad. Contiene los prototipos de todas las funciones de utilidad del PR. Algunas se utilizan tambien en el CIR. Ofrecen soporte a tipos de datos definidos en la aplicacion como la cola de transiciones o el archivo de configuracion.

Autor

Carlos Rodríguez (CarlosRdrz)

4.7.2. Documentación de las funciones

4.7.2.1. void config_destroy (struct config_node * *config*)

Libera recursos utilizados por los nodos de propiedades que se crearon al cargar un archivo de configuracion.

Parámetros

queue	Puntero al primer nodo de la lista de propiedades

Devuelve

void

```
90 {
91   struct config_node *node = config;
92   struct config_node *next;
93
94   while(node != NULL) {
95     next = node->next;
96     free(node);
97     node = next;
98   }
99 }
```

4.7.2.2. char* config_get (struct config_node * *config*, const char * *key*)

Obtiene una propiedad de una lista de propiedades de un archivo de configuracion.

Parámetros

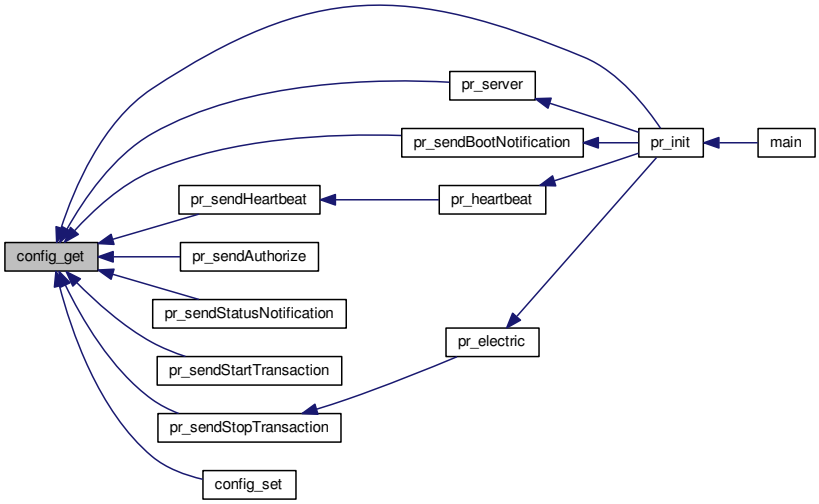
config	Puntero al primer nodo de la lista de propiedades
key	Clave de la propiedad que queremos obtener

Devuelve

Cadena con el valor de la clave que hemos buscado

```
38 {
39   struct config_node *node = config;
40   char *value = NULL;
41
42   while(node != NULL) {
43     if(!strcmp(node->key, key)) value = node->value;
44     node = node->next;
45   }
46
47   return value;
48 }
```

Gráfico de llamadas a esta función:



4.7.2.3. struct config_node* config_load (const char * *file_name*)

Carga un archivo de configuracion.

Parámetros

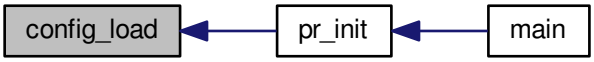
file_name	Nombre del archivo de configuracion a leer

Devuelve

Puntero al primer nodo de la lista de propiedades del archivo de configuracion leido

```
4 {
5   struct config_node *first = NULL;
6   struct config_node *tail = NULL;
7   FILE *file = fopen(file_name, "r");
8
9   if (file != NULL) {
10     char line[128];
11
12     while(fgets(line, sizeof(line), file) != NULL) {
13       char *cfline;
14       cfline = strstr((char *)line, CONFIG_DELIMITER);
15       cfline = cfline + strlen(CONFIG_DELIMITER);
16
17       struct config_node * current = (struct config_node *)calloc(1, sizeof(struct
     config_node));
18       strncpy(current->key, line, (strlen(line) - strlen(cfline) - 1));
19       strncpy(current->value, cfline, strlen(cfline) - 1);
20       current->next = NULL;
21
22       if (first == NULL) {
23         first = current;
24         tail = current;
25       } else {
26         tail->next = current;
27         tail = current;
28       }
29     }
30   }
31
32   fclose(file);
33
34   return first;
35 }
```

Gráfico de llamadas a esta función:



4.7.2.4. void config_save (struct config_node * *config*)

Guarda las propiedades de configuracion en el archivo del disco.

Parámetros

config	Puntero al primer nodo de la lista de propiedades

Devuelve

Cadena con el valor de la clave que hemos buscado

```
73 {
74   struct config_node * current = config;
75   FILE * config_file = fopen("config", "w");
76
77   if (config_file != NULL) {
78     while (current != NULL) {
79       fprintf(config_file, "%s=%s\n", current->key, current->value);
80       current = current->next;
81     }
82   } else {
83     printf("[ERROR] Couldnt open file config to write config properties\n");
84   }
85
86   fclose(config_file);
87 }
```

4.7.2.5. void config_set (struct config_node * *config*, const char * *key*, char * *value*)

Establece una propiedad en la lista de propiedades.

Parámetros

config	Puntero al primer nodo de la lista de propiedades
key	Clave de la propiedad que queremos establecer
value	Valor de la propiedad

Devuelve

Cadena con el valor de la clave que hemos buscado

```
51 {
52   char *node = config_get(config, key);
53
54   if (node == NULL) {
55     struct config_node * new_node = (struct config_node *)calloc(1, sizeof(struct
      config_node));
56     struct config_node * last = config;
57     while (last->next != NULL) last = last->next;
58     strncpy(new_node->key, key, 20);
59     strncpy(new_node->value, value, 64);
60     last->next = new_node;
```

```
61   } else {
62     struct config_node * current = config;
63     while (current != NULL) {
64       if (!strcmp(current->key, key)) {
65         strncpy(current->value, value, 64);
66       }
67       current = current->next;
68     }
69   }
70 }
```

Gráfico de llamadas para esta función:

4.7.2.6. void delete_timer (struct timer ** *cola*, struct timer * *to_delete*)

Borra un timer de la cola de timers.

Parámetros

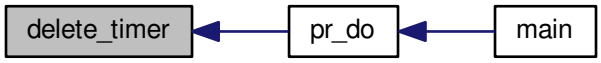
cola	Cola de timers
to_delete	Nodo a eliminar

Devuelve

void

```
131 {
132   struct timer *previous;
133
134   // Si la cola no esta vacia...
135   if (*cola != NULL) {
136     if (*cola == to_delete) {
137       // Si es el primer nodo de la lista
138       *cola = to_delete->next;
139     } else {
140       // Si no, cogemos el nodo anterior y actualizamos su campo next
141       previous = *cola;
142       while (previous->next != NULL && previous->next != to_delete) {
143         previous = previous->next;
144       }
145       previous->next = to_delete->next;
146     }
147   }
148
149   free(to_delete);
150 }
```

Gráfico de llamadas a esta función:



4.7.2.7. void new_timer (struct timer ** *cola*, void(*)(void *) *fun*, void * *param*, int *cuando*)

Especifica una funcion a ejecutar en un instante determinado.

Parámetros

cola	Cola de timers a ejecutar
fun	Funcion a ejecutar
cuando	Instante en el que ejecutar la funcion

Devuelve

void

```
109 {
110   struct timer *new_node = (struct timer *)malloc(sizeof(struct timer));
111   new_node->funct = fun;
112   new_node->param = param;
113   new_node->timestamp = time(NULL) + cuando;
114   new_node->next = NULL;
115
116   // Si la cola esta vacia...
117   if (*cola == NULL) {
118     *cola = new_node;
119   } else {
120     // Coger el ultimo nodo de la lista
121     struct timer *current = *cola;
122     while(current->next != NULL) {
123       current = current->next;
124     }
125     // Colocar como siguiente nodo el nuevo cliente
126     current->next = new_node;
127   }
128 }
```

4.7.2.8. size_t write_data (void * *ptr*, size_t *size*, size_t *nmemb*, FILE * *stream*)

Escribe en ptr los datos leidos por el stream.

Esta funcion es utilizada por la libreria CURL para guardar el resultado de un archivo descargado por HTTP, concretamente para la funcion de actualizar el firmware del PR.

Parámetros

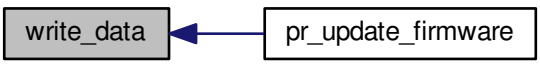
ptr	Puntero al array de elementos a escribir
size	Espacio reservado en ptr
nmemb	Numero de elementos de tamano size a escribir
stream	FILE donde se escribira

Devuelve

size_t bytes leidos

```
102 {
103     size_t written;
104     written = fwrite(ptr, size, nmemb, stream);
105     return written;
106 }
```

Gráfico de llamadas a esta función:



Índice alfabético