IoT - OCPP RPC

Matteo La Licata, Filippo Siri, Claudio Stana

Aprile 2024

1 Libreria OCPP-RPC

Abbiamo utilizzato questa libreria in quanto permette di implementare client e server come definito nel protocollo OCPP 1.6 integrandosi con Nodejs.

Questa libreria implementa il protocollo OCPP tramite web socket.

Tra le varie feature fornite dalla libreria abbiamo utilizzato la "Strict Validation" che permette di validare le chiamate e le risposte secondo gli schemi del protocollo OCPP.

2 Aggiunta nuovo client

Ogni volta che un client si connette al server viene generato un evento di tipo `client` che il server gestisce aggiungendo il nuovo client in una lista e definendo gli handle degli eventi del protocollo OCPP. In questo modo è possibile aggiungere un nuovo client senza problemi e senza dover fare modifiche al backend.

3 Stati di una stazione

Per simulare gli stati che può assumere la stazione, all'interno dello script `client.js` è presente una variabile `status` che può assumere i seguenti stati previsti dal protocollo `OCPP`:

- Available
- Charging
- Finishing
- Reserved
- Unavailable
- Faulted

Quando vengono gestiti i vari eventi del protocollo `OCPP` viene aggiornato lo stato della stazione.

4 Metodi implementati

La libreria fornisce i metodi `handle` e `call` con i quali è possibile gestire la ricezione e la generazione di eventi previsti dal protocollo OCPP.

4.1 Boot Notification

L'evento `BootNotification` viene generato ogni volta che un punto di ricarica si avvia o si riavvia. Nel periodo che va dall'avvio/riavvio della stazione fino al completamento, con esito positivo, della BootNotification in cui il sistema centrale restituisce uno stato di "Accettato" o "In attesa", il punto di ricarica non deve trasmettere ulteriori richieste al sistema centrale.

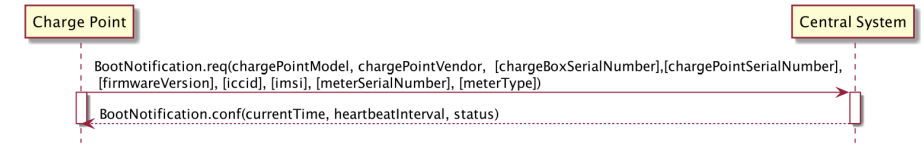

Figure 1: Boot Notification

4.1.1 Implementazione

Nel nostro progetto viene generato l'evento di BootNotification dopo che il client si è connesso al server. Nella richiesta indichiamo il `chargePointVendor` e il `chargePointModel` in quanto sono paramtri obbligatori, ma siccome si tratta di una demo abbiamo messo dei valori non significativi.

Nella risposta vengono inviati i valori del `CurrentTime`, `heartbeatInterval` e `Status`.

Come indicato nel protocollo, prima di inviare nuove richieste aspettiamo di ricevere una risposta dal server e controlliamo che il campo status sia `accepted`. Trattandosi di una demo, abbiamo messo che il server accetta sempre la richiesta di una nuova stazione.

Sempre come indicato nel protocollo, tramite il campo `heartbeatInterval` definiamo ogni quanto tempo inviare l'evento di HeartBeat al sistema centrale.

4.2 HeartBeat

Il punto di ricarica deve inviare un evento `HeartBeat` per garantire che il sistema centrale sappia che un punto di ricarica è ancora vivo. Al ricevimento di un evento HeartBeat, il sistema centrale deve rispondere con l'ora corrente del sistema centrale.

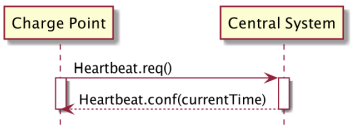

Figure 2: Heart Beat

4.2.1 Implementazione

Nel nostro progetto viene generato l'evento Hearbeat da un punto di ricarica ogni volta che trascorre una quantità di tempo pari a quella specificata nel campo `interval` all'interno della risposta da parte del server all'evento BootNotification, inviando il primo Heartbeat subito dopo che venga completato l'evento BootNotification.

Tramite questo evento, nel backend, aggiorniamo un campo all'interno del database per controllare se una stazione è fuori uso. Infatti dopo che non vengono ricevuti 3 heartbeat la stazione entra nello stato `broken`.

Il server, dopo aver aggiornato il valore nel database, risponde inviando il currentTime.

4.3 Status Notification

L'evento `Status Notification` viene generato da un punto di ricarica che invia una notifica al sistema centrale per informarlo di un cambiamento di stato o di un errore all'interno del punto di ricarica. Il server risponde senza inserire alcun dato.

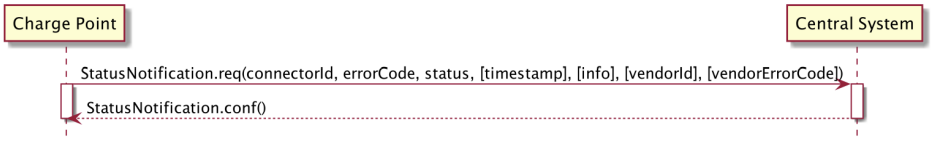

Figure 3: Status Notification

4.3.1 Implementazione

Nel nostro progetto viene generato l'evento Status Notification quando un punto di ricarica è stato prenotato senza che poi sia stato utilizzato entro il tempo stabilito per far sì che la prenotazione non scada. All'interno dell'evento Status Notification generato dal punto di ricarica è contenuto il `connectorId` (fisso a 0), `errorCode` (con valore fisso a "NoError") e `status` (con valore fisso ad "Available"). Il server risponde senza inserire alcun dato.

Siccome nel backend gestiamo la scadenza di una prenotazione in base al timestamp dove è stata effettuata, quando si riceve questo evento non viene effettuato niente. È stato implementato lo stesso per avere un backend compatibile con il protocollo.

4.4 Meter Values

L'evento `Meter Values` permette ad un punto di ricarica di campionare il contatore di energia o altri sensori/trasduttori hardware per fornire informazioni aggiuntive sui valori del contatore, infatti spetta al punto di ricarica decidere quando inviare i valori del contatore.

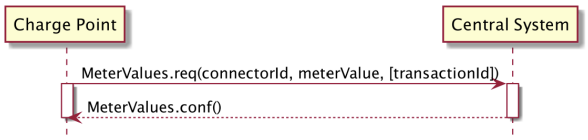

Figure 4: Meter Values

4.4.1 Implementazione

Nel nostro progetto viene generato l'evento Meter Values in maniera periodica dopo un intervallo di tempo predefinito e solamente se lo stato del punto di ricarica è in `charging`. All'interno della richiesta sono inseriti:

- connectorID: valore fissato a 0;
- transactionID: valore che identifica il valore dell'ultima transazione eseguita sul punto di ricarica interessato;
- meterValue: array di oggetti che contiene la data di quando è stato generato l'evento meter values e l'oggetto `sampledValue` che fornisce le varie info sulla quantità di energia consumanta.

Trattandosi di una demo, abbiamo inserito un contatore che viene incrementato di 500watt ogni volta che viene generato un evento Meter Values.

Il server utilizza il valore ricevuto per sapere quanta corrente è stata erogata durante la ricarica e quindi:

- Fornire un'indicazione in tempo reale all'utente di quanto sta spendendo e di quanta energia è stata erogata
- Interrompere una ricarica se dovesse finire il saldo dell'account

4.5 Authorize

Il metodo Authorize viene utilizzato prima che la Charging Station faccia partire o fermare una ricarica. La ricarica viene iniziata solamente dopo la conferma dal server. Il server risponde alla richiesta di Authorize precedente a una stopTransaction se e solo se il richiedente della stop è diverso dal richiedente della start.

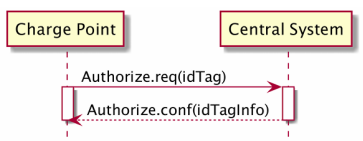

Figure 5: Authorize

4.5.1 Implementazione

Nel nostro progetto l'authorize viene utilizzata all'interno della gestione dell'evento RemoteStartTransaction per autorizzare l'inizio di una nuova ricarica.

Viene inviato il campo idTag che rappresenta l'id dell'utente ed è fornito nell'evento RemoteStartTransaction e, dopo che si riceve una risposta, si controlla che il campo status contenga il valore `accepted`.

Non avendo motivi per rifiutare una nuova transazione, il server accetta sempre la richiesta `accepted`.

4.6 Start Transaction

L'evento `StartTransaction` viene generato quando la stazione deve iniziare una nuova ricarica per informare il sistema centrale.

La stazione invia l'idTag, l'id del connettore dove si sta effettuando la ricarica, il valore iniziale di meter value ed il timestamp di quando inizia la ricarica.

Al ricevimento di una StartTransaction, il sistema centrale deve rispondere sempre inviando un `transactionID` ed autorizzando l'id tag.

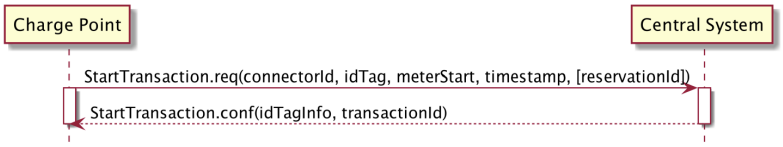

Figure 6: Start Transaction

4.6.1 Implementazione

Nel nostro progetto, siccome non è stato previsto un modulo NFC o l'avvio di una ricarica tramite una carta, non è possibile far partire una ricarica direttamente dalla stazione. Per questo motivo l'evento startTransaction viene generato solamente all'interno della gestione dell'evento remoteStartTransaction per far partire effettivamente una nuova ricarica.

Verranno passati quindi come valori di idTag e connectorId quelli indicati dall'evento remoteStartTransaction.

Prima di effettuare l'evento Start Transaction, la stazione controlla di essere nello stato `Available` o `Reserved` e nel caso in cui fosse nello stato Reserved, il server controlla che la stazione sia prenotata dallo stesso id che sta facendo partire la richiesta.

Il server quando gestisce l'evento restituisce come transactionId l'id del record generato nel database per memorizzare quella nuova ricarica e aAcepted come status, visto che come nel caso precedente non avevamo motivi per rifiutare una ricarica.

Dopo aver ricevuto la risposta dal server, il client salva l'id della transazione inviato nella risposta.

4.7 Stop Transaction

L'evento `Stop Transaction` viene generato quando una transazione viene interrotta per notificare al sistema centrale l'interruzione della transazione. Al ricevimento di una richiesta StopTransaction, il sistema centrale deve rispondere indicando se ha accettato o meno l'interruzione della ricarica.

All'interno dell'evento stop transaction il client invia quanta energia è stata consumata, il timestamp di quando termina la ricarica, l'id della transazione che termina ed il motivo per cui termina la ricarica.

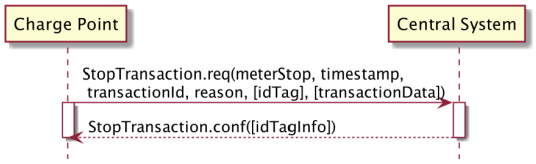

Figure 7: Stop Transaction

4.7.1 Implementazione

Come con la start transaction, nel nostro progetto viene generato l'evento Stop Transaction solamente quando il punto di ricarica riceve dal server un evento `Remote Stop Transaction`.

Prima di effettuare la stop transaction, il client controlla di essere nello stato `Charging`, altrimenti fallisce.

Nella demo, siccome si tratta di un campo facoltativo, non inviamo mai il motivo per cui termina la transazione, però inviamo correttamente l'energia consumata, il timestamp ed il transaction id. Inoltre, come indicato nel protocollo, non controlliamo se il server ha accettato, ma viene interrotta comunque la ricarica.

Il server, quando gestisce questo evento, aggiorna nel database il record che corrisponde all'ultima ricarica effettuata presso la stazione ha inviato questo evento, aggiorna il bilancio dell'utente ed infine invia un messaggio di risposta con status `accepted`.

Il client, dopo aver ricevuto la risposta dal server, aggiorna il suo stato tornando nello stato `Available`.

4.8 Remote Start Transaction

La remote start transaction permette al sistema centrale di avviare una ricarica presso una stazione.

Per implementare la remote start transaction, il client quando riceve la richiesta genera un nuovo evento di tipo start transaction.

La remote start transaction invia al client l'id dell'utente che avvia la ricarica ed il client, se viene autorizzato e la ricarica inizia con successo, invia al server un messaggio con lo stato accepted. Altrimenti invia un messaggio con lo stato rejected.

Figure 8: Remote Start Transaction

4.8.1 Implementazione

Nel nostro progetto, quando un punto di ricarica riceve una `Remote Start Transaction`, viene controllato se quella stazione è nello stato `Reserved` ed in quel caso si controlla che l'id della transazione che ha effettuato la prenotazione sia uguale a quello inviato nella richiesta Remote Start Transaction.

Dopo aver controllato l'eventuale prenotazione, il punto di ricarica manda una richiesta `Authorize` e se viene accettata genera un evento Start Transaction. Se anche il secondo evento ha successo viene inviato un messaggio con lo stato Accepted ed inizia la ricarica, altrimenti con lo stato Rejected.

Quando inizia la ricarica il client entra nello stato `Charging`, altrimenti in quello available.

4.9 Remote Stop Transaction

La remote stop transaction permette al sistema centrale di interrompere una ricarica presso una stazione.

Per implementare la remote stop transaction, il client quando riceve la richiesta genera un nuovo evento di tipo stop transaction.

La remote start transaction invia al client l'id della transazione da terminare ed il client invia un messaggio che indica se la richiesta è stata accettata.

Figure 9: Remote Stop Transaction

4.9.1 Implementazione

Nel nostro progetto quando un punto di ricarica riceve una `Remote Stop Transaction`, prima di fermare l'erogazione di energia emette un evento di tipo `Stop Transaction` e quando riceve una risposta interrompe la ricarica e risponde all'evento remote stop transaction inviando `accepted`. Come previsto dal protocollo, il client non controlla che il server accetti la richiesta stop transaction.

4.10 Reserve Now

Il sistema centrale può mandare una richiesta di Reserve Now a una charging station per prenotare un punto di ricarica per uno specifico mezzo inviando il connector id, la data di scadenza, l'id che identifica l'utente e l'id della prenotazione.

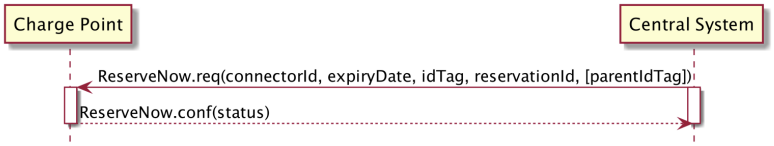

Figure 10: Reserve now

4.10.1 Implementazione

Nel nostro progetto, il server prima di generare un evento di tipo Reserve Now crea un record nel database che rappresenta quella prenotazione e lo cancella nel caso in cui la stazione dovesse rifiutare la richiesta.

Nella richiesta Reserve Now, viene inviato l'id dell'utente, l'id del record appena creato e che identifica la transazione e l'expiration time. Siccome non abbiamo previsto che una stazione possa essere utilizzata contemporaneamente da più utenti, inviamo sempre 0 come connector id.

Il client, quando riceve la richiesta, controlla di essere nello stato `Available`, altrimenti restituisce lo stato `Rejected`. Se il client era nello stato Available entra nello stato Reserved, memorizza l'id della transazione, restituisce lo stato `Accepted` e fa partire un timer che alla sua scadenza genera un evento `Status Notification` e cancella la prenotazione all'interno del client tornando nello stato `Available`.

4.11 Cancel Reservation

Per annullare una prenotazione, il sistema centrale invia un evento `CancelReservation` al punto di ricarica specificando l'id della prenotazione. Se il punto di ricarica ha una prenotazione corrispondente al reservationId nella richiesta, deve restituire lo stato "Accepted" ed annullare la prenotazione, altrimenti deve restituire lo stato "Rejected".

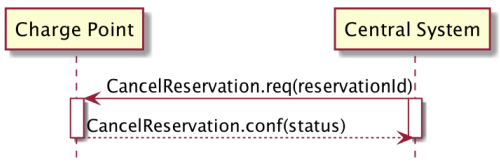

Figure 11: Cancel Reservation

4.11.1 Implementazione

Nel nostro progetto, il server genera l'evento Cancel Reservation dopo aver aggiornato il record presente nel database che corrisponde a quella prenotazione. Nel caso in cui il client dovesse rifiutare, il server ripristina il valore precedente per non cancellare la prenotazione e restituisce un errore.

Il client quando riceve la richiesta, controlla di essere nello stato reserved e che sia prenotato da una richiesta che ha lo stesso id indicato nel reservationId. Se entrambe le condizioni sono rispettate entra nello stato available e restituisce un messaggio con lo stato `Accepted`, altrimenti restituisce un messaggio con lo stato `Rejected`.