УДК 004.92

*Марія Удалова,
(магістрант, Державний університет інфраструктури та технологій)
Олена Шикла, д.ф.-м.н.
(завідувач кафедри інформаційних технологій, Державний університет
інфраструктури та технологій)*

МОДЕЛЮВАННЯ ВЗАЄМОДІЇ ЕЛЕКТРОМОБІЛЯ ІЗ ЗАРЯДНОЮ СТАНЦІЄЮ НА ОСНОВІ РЕАЛІЗАЦІЇ OPEN CHARGE POINT PROTOCOL

Досліджено документацію OCPP-протоколу версії 1.6, зроблено порівняльний аналіз реалізації цього протоколу за допомогою SOAP та JSON. На основі Windows Forms створено та протестовано застосунок, в якому реалізоване моделювання взаємодії електромобіля із зарядною станцією на основі часткової реалізації Open Charge Point Protocol на мові програмування C# з використанням JSON-формату та WebSocket.

Ключові слова: *електромобіль, електрозаправка, Open Charge Point Protocol, C#, JSON-формат, WebSocket.*

Вступ. У наш час електромобілі стають все більш популярними, до того ж не тільки як особистий засіб пересування, а як і альтернатива громадському та обслуговуючому транспорту. Так, наприклад, компанія Volvo Trucks нещодавно представила свою другу електровантажівку FE Electric, яку пропонується використовувати як сміттєвоз. Взагалі, електромобілі є дуже перспективним сегментом ринку. Активне зростання їх кількості у багатьох країнах обумовлене державними пільгами для екологічного транспорту, розширенням модельного ряду електромобілів, а також зниженням вартості батарей. Британською газетою «The Guardian» було опубліковано результати дослідження шведської аналітичної групи «EV-Volumes», згідно яких кількість повністю електричних і гібридних автомобілів на дорогах світу становить понад 3 млн одиниць. Кількість електромобілей в Україні також дуже швидко зростає. В лютому 2020 року українці поставили на облік 623 легкових електромобіля, що на 67% більше, ніж в лютому минулого року. Такі дані опублікував «Укравтопром».

Звісно, електромобілям будь-якого призначення необхідно заправлятися електроенергією, і тут не обійтися без електрозаправок, а заправкам – без програмного забезпечення. Тривалий час не існувало єдиного комунікаційного протоколу між різними виробниками зарядних станцій, що ускладнювало процес зарядки електротранспорту власниками, а також гальмувало як розвиток компаній-власників мереж зарядних станцій, так і загалом популяризацію електричного транспорту в світі. Отже, актуальними проблемами є стандартизація та уніфікація протоколу зарядки, взаємозв'язок між обладнанням і передачею даних для зарядних пристроїв різних виробників.

Аналіз останніх досліджень і постановка проблеми. Першим кроком у вирішенні цієї проблеми є запропонований альянсом «Open Charge Alliance» протокол Open Charge Point Protocol (далі OCPP). Протокол OCPP – спеціальний стандартизований протокол, який створено для можливості віддаленого управління заправками для електромобілів. Даний протокол є ініціативою нідерландської організації «E-Laad», мета якої полягає в створенні відкритого протоколу, що дозволяє серверам усіх компаній спілкуватися з клієнтами заправок незалежно від їх виробника.

DOI:10.32703/2617-9040-2020-35-14

З часу створення альянсу «Open Charge Alliance» протокол OCPP зазнавав постійних змін. Так, перша версія протоколу (OCPP 1.0), яка була представлена у квітні 2015, лише віддалено схожа на останні версії. У квітні 2018 року була опублікована нова версія протоколу – OCPP v2.0. Зараз даний протокол стає все більш популярним і де-факто стандартом у 50 країнах (наочним підтвердженням є те, що він був застосований до більш ніж 40000 зарядних станцій). На відміну від власних протоколів зв'язку, OCPP є відкритим протоколом і не пов'язаний з будь-якими витратами на ліцензування. Він затверджений багатьма постачальниками точок доступу та постачальниками центральних систем. Дотримання OCPP тепер стає вимогою «must-have» серед інвесторів. Але реалізація цього протоколу до теперішнього часу знаходиться в закритому доступі. Тому актуальною є реалізація даного протоколу у відкритому доступі.

Мета і завдання дослідження. Метою роботи є моделювання процесу заправки електромобіля шляхом часткової реалізації Open Charge Point Protocol згідно з документацією OCPP v1.6. Для досягнення поставленої мети потрібно:

- дослідити документацію OCPP-протоколу версії 1.6, оцінити переваги та недоліки реалізації кожного з двох запропонованих авторами способів;
- згідно з вимогами до функціональної частини застосунку частково реалізувати OCPP-протокол на мові C# [1];
- за допомогою графічного редактора розробити дизайн застосунку-сервера та застосунку-клієнта [2];
- за допомогою технології Windows Forms [3] реалізувати кінцевий застосунок;
- протестувати готовий застосунок.

Матеріали та методи дослідження. В ході дослідження було проведено порівняльний аналіз двох реалізацій Open Charge Point Protocol v1.6, що знаходяться у відкритому доступі, та здійснено комп'ютерне моделювання взаємодії електромобіля із зарядною станцією на основі реалізації OCPP.

Існує дві реалізації OCPP-протоколу у відкритому доступі:

- Java-OCA-OCPP від «ChargeTimeEU» з ліцензійною політикою «open source under MIT License». Її недоліками є те, що вона, по-перше, знаходиться ще у розробці, а по-друге, на даний момент є тільки бібліотекою класів (реалізація саме керуючої програми відсутня). Вона написана на мові програмування JAVA.
- OCPP client-server від NewMotion з ліцензійною політикою «open source under the GPL version 3». Дана реалізація також є тільки бібліотекою класів (реалізація саме керуючої програми відсутня). Вона написана на мові програмування Scala.

Варто зазначити, що на сайтах компаній-власників мереж електрозаправок, наприклад Siemens, можна дізнатись про використання ними власних реалізацій управління електрозаправкою без використання OCPP протоколу. Але, звісно, при такому підході електрозаправку неможливо підключити в мережу електрозаправок, що підтримують OCPP. До того ж, недоліком власних реалізацій є відсутність відкритих стандартів.

Порівняння двох можливих принципово різних шляхів реалізації протоколу. Реалізація Open Charge Point Protocol v1.6 може бути виконана одним із двох шляхів:

- SOAP [4] (з використанням HTTP, оснований на Open Charge Point Protocol v1.5);
- JSON [5] (з використанням WebSockets [6]).

Задля запобігання плутанини в повідомленні про тип реалізації автори рекомендують використовувати окремі суфікси, а саме: J (OCPP-J) та S (OCPP-S) відповідно до перших літер: JSON або SOAP.

Розглянемо більш детально обидва варіанти: технології та засоби реалізації, переваги та недоліки.

Переваги та недоліки реалізації OCPP SOAP Specification. Прийнято вважати, що однією з головних переваг SOAP [7] є його гнучкість, завдяки якій можна використовувати різні транспортні протоколи. І хоча стандартні реалізації використовують HTTP [8], однак також можливо використовувати JMS, SMTP та інші реалізації. Автори специфікації радять

використовувати саме HTTP [9], оскільки інші реалізації є більш громіздкими, а також можуть обмежувати повну реалізацію протоколу. Тому чи не єдина перевага SOAP [10] не відіграє ніякої ролі в реалізації OCPP-протоколу.

Дуже вагомим недоліком реалізації OCPP-S є відсутність постійного зв'язку з сервером, а оскільки процес зарядки електромобіля – це і транзакції оплати за послуги, і тривалість оплаченої зарядки та інші етапи, що потребують збереження станів, то реалізація OCPP-S без врахування цієї інформації може призвести до втрати довіри клієнтів.

До того ж SOAP-повідомлення є досить об'ємними. Звідси можна визначити ще два недоліки: по-перше, швидкість передачі таких повідомлень є більша, ніж у випадку передачі повідомлення-стрічки; по-друге, процес обробки інформації також є більш проблематичним, враховуючи відсутність або недосконалість відповідних бібліотек.

Загалом, ця реалізація є набагато складніша та більш громіздка ніж OCPP-J. Але, оскільки ранні версії документації протоколу OCPP були написані відповідно для реалізації з використанням SOAP, то підтримка більш ранніх версій реалізації протоколу є чи не єдиною перевагою і сенсом реалізації OCPP-S.

Переваги та недоліки реалізації OCPP JSON Specification. Всі розглянуті недоліки реалізації OCPP SOAP Specification відсутні в реалізації OCPP JSON Specification. Наприклад, за рахунок своєї лаконічності в порівнянні з XML [11], формат JSON [12] може бути більш придатним для серіалізації складних структур, і загалом процес серіалізації є більш швидким. Завдяки протоколу WebSocket [13] існує постійний зв'язок із сервером. Тобто, ми отримуємо повідомлення про розрив з'єднання в момент роз'єднання, а не як при реалізації OCPP SOAP Specification, в момент останнього повідомлення (heartbeat) без відповіді.

Отже, для моделювання взаємодії електромобіля з зарядною станцією було вибрано OCPP JSON Specification.

Розробка функціональної частини застосунку. Для моделювання взаємодії електромобіля із зарядною станцією з використанням Open Charge Point Protocol було обрано такий сценарій. Мережа зарядних станцій працює таким чином: клієнт, маючи бажання зарядити свій електромобіль, підходить до оператора, який взявши плату, може дати дозвіл на зарядку. Звісно, більшість сучасних мереж електрозаправок використовує мобільні застосунки, але оскільки мета даної роботи є моделювання, то процесами ідентифікації, оплатою картою ми вирішили знехтувати.

На рис. 1 зображено діаграму прецедентів застосунку:

Рис. 1. Діаграма прецедентів

Визначення функціональних вимог зі сторони клієнта. На даному етапі роботи було визначено такі вимоги до функціоналу клієнта:

- 1) Намагатися підключитися до серверу з часовим інтервалом, прописаним в конфігураційному файлі.
- 2) Намагатися відновити підключення в разі розриву зв'язку (з тим же інтервалом).
- 3) Завершити сесію підключення до серверу.
- 4) Приймати та відповідати на запити таких типів:
 - RemoteStartTransaction (наказ почати заправку);
 - RemoteStopTransaction (наказ закінчити заправку);
 - ChangeAvailability (наказ зробити порт активним/неактивним);
 - UnlockConnector (наказ розблокувати порт);
 - Reset (наказ перезапустити електрозаправку).
- 5) Посилати та приймати відповіді на запити таких типів:
 - BootNotification (містить інформацію про електрозаправку);
 - MeterValues (містить інформацію про вольти та амperi поточної зарядної сесії);
 - StartTransaction (містить інформацію про те, що заправка почалася);
 - StatusNotification (містить інформацію про статус конектору);
 - StopTransaction (містить інформацію про те, що заправка закінчилася).

Визначення функціональних вимог зі сторони сервера. Аналогічно було визначено такі вимоги до функціоналу сервера:

- 1) Очікувати підключення клієнта з вказаним іменем на вказаному порті.
- 2) Завершити сесію підключення з клієнтом.
- 3) Приймати та відповідати на запити таких типів:
 - BootNotification (містить інформацію про електрозаправку);
 - MeterValues (містить інформацію про вольти та амperi поточної зарядної сесії);
 - StartTransaction (містить інформацію про те, що заправка почалася);
 - StatusNotification (містить інформацію про статус конектору);
 - StopTransaction (містить інформацію про те, що заправка закінчилася)
- 4) Посилати та приймати відповіді на запити таких типів:
 - RemoteStartTransaction (наказ почати заправку).
 - RemoteStopTransaction (наказ закінчити заправку).
 - ChangeAvailability (наказ зробити порт активним/неактивним).
 - UnlockConnector (наказ розблокувати порт).
 - Reset (наказ перезапустити електрозаправку).

Інструментальні засоби розробки програмного продукту. Для реалізації функціональної частини було обрано мову програмування C# [14], а також технології Windows Forms [15] та середовище розробки Visual Studio.

Базовим протоколом нижчого рівня є WebSocket [16] з використанням JSON-файлів.

Згідно з документацією OCPP повідомлення діляться на 3 типи (таблиця 1):

Таблиця 1. Типи повідомлень

Message Type	Message Type Number	Direction
CALL	2	Client-to-Server
CALLRESULT	3	Server-to- Client
CALLERROR	4	Server-to- Client

- 1) Повідомлення типа CALL має наступну структуру: [<MessageTypeId>, "<UniqueId>", "<Action>", {<Payload>}]
 - MessageTypeId – число яке вказує тип повідомлення;
 - UniqueId – це унікальний ідентифікатор, який буде використовуватися для перевірки відповідності запиту та відповіді на запит;
 - Action – чутливий до регістру рядок, назва дистанційної процедури або дії;
 - Payload – об'єкт JSON, який містить аргументи, що відносяться до дії.
- 2) Повідомлення типа CALLRESULT має наступну структуру:[<MessageTypeId>, "<UniqueId>", {<Payload>}]
 - MessageTypeId – теж що і в CALL;
 - UniqueId – це має бути точно той самий ідентифікатор, який знаходиться в запиті на виклик, щоб одержувач міг відповідати саме на цей запит;
 - Payload - теж що і в CALL;
- 3) Повідомлення типа CALLERROR має наступну структуру: [<MessageTypeId>, "<UniqueId>", "<errorCode>", "<errorDescription>", {<errorDetails>}]
 - MessageTypeId – теж що і в CALL;
 - UniqueId – теж що і в CALLRESULT;
 - ErrorCode – це поле повинно містити рядок із таблиці "ErrorCode".
 - ErrorDescription – потрібно заповнити, якщо це можливо, інакше – чистий порожній рядок;
 - ErrorDetails – об'єкт JSON описує деталі помилки невизначеним способом. Якщо немає помилок, то повинен бути заповнити порожній об'єкт "{}".

Візуалізація та моделювання процесу зарядки електромобіля. Візуальна частина роботи була зроблена з використанням технології Windows Presentation Foundation (WPF) [17] на мові програмування C# [18]. Технологія WPF [19] використовує декларативну мову розмітки XAML [20]. На рис. 2 зображено розроблений дизайн для застосунку зі сторони клієнта:

Рис. 2. Дизайн для застосунку-клієнта

На рис. 3 зображено розроблений дизайн для застосунку зі сторони сервера:

Рис. 3. Дизайн для застосунку-клієнта

За допомогою графічного редактору Adobe Photoshop було розроблено анімації з кадрів (приклади кадрів на рис. 4):

Рис. 4. Приклад кадра для створення анімації

При запуску застосунків сервера та клієнта клієнт автоматично підключається до серверу і після вдалого підключення посилає інформацію про електрозаправку та поточні статуси конекторів. При запуску анімації можна побачити, як машина під'їжджає до електрозаправки, статуси портів відображаються на клієнті та сервері (див. рис. 5).

Рис. 5. Демонстрація сценарію застосунку

При натисканні кнопки Start Transaction на сервері на першому або другому конекторі сервер посилає клієнтові StartTransactionRequest, клієнт погоджується почати заправку та посилає статус порта Preparing – очікування вставки кабелю, статус відображається на клієнті та сервері (див. рис. 6).

Рис. 6. Демонстрація сценарію застосунку

При натисканні кнопки «Plug In connector 1» починається процес заправки – клієнт посилає статус Charging та починає слати інформацію про вольти та ампері процесу заправки (MeterValues). Ця інформація відображається на клієнті та сервері (див. рис. 7).

Рис. 7. Демонстрація сценарію застосунку

При натисканні кнопки Make Unavailable сервер посилає запит ChangeAvailability з параметром Inoperative, і якщо в цей момент на цьому конекторі не відбувається зарядка, то клієнт погоджується і робить конектор неактивним, в іншому разі запит ставиться до черги та конектор стає неактивним після кінця зарядної сесії (див. рис. 8).

Рис. 8. Демонстрація сценарію застосунку

Після об'єднання функціональної та візуальної частини, було отримано працездатний застосунок, який демонструє моделювання взаємодії електромобіля із зарядною станцією. При розробці було враховано багато сценаріїв використання: зарядка електромобіля в позитивному сценарії (робочий сервер, доступний порт); обробка таких негативних сценаріїв використання, як неробочий сервер, або клієнт намагається почати зарядку на недоступному порті, або намагається відключитись в процесі незакінченої зарядки тощо. Готовий працездатний застосунок може бути використаний для наочної демонстрації роботи протоколу, а також для зарядки електромобіля від зарядної станції.

Висновки. В роботі досліджено документацію OCPP-протоколу версії 1.6, зроблено порівняльний аналіз реалізації цього протоколу за допомогою SOAP та JSON. З використанням Windows Forms створено та протестовано застосунок, в якому реалізоване моделювання взаємодії електромобіля з зарядною станцією на основі часткової реалізації Open Charge Point Protocol на мові програмування C# з використанням JSON-формату та WebSocket. Готовий працездатний застосунок може бути використаний для наочної демонстрації роботи протоколу, а також для зарядки електромобіля від зарядної станції.

ЛІТЕРАТУРА

1. Turner R. C#: 3 books in 1 - The Ultimate Beginners, Intermediate and Expert Guide to Master C# Programming. Nelly B.L. International Consulting Ltd, 2020. 636 p.
2. Jin B. Designing Web APIs: Building APIs That Developers Love. O'Reilly Media, 2018. 232 p.
3. Feldman A. WPF in Action with Visual Studio 2008: Covers Visual Studio 2008, SP1 and .NET 3.5 SP1. Manning Publications, 2008. 650 p.
4. Snell J., Tidwell D., Kulchenko P. Programming Web Services with SOAP. O'Reilly & Associates, Inc, USA, 2001. 244 p.
5. Smith B. Beginning JSON. Apress, 2015. 326 p.
6. Lombardi A. WebSocket: Lightweight Client-Server Communications. O'Reilly Media, 2015. 144 p.
7. Nandan P. Mastering SoapUI. Packt Publishing, 2016. 240 p.
8. Allen K. What Every Web Developer Should Know About HTTP. OdeToCode LLC, 2012. 80 p.
9. Gourley D. Http: The Definitive Guide (Definitive Guides). O'Reilly Media, 2002. 656 p.
10. Newcomer E. Understanding Web Services: XML, WSDL, SOAP, and UDDI. AddisonWesley Professional, 2002. 368 p.
11. Ramchand V., Mukhi S. XML Web Services & Soap By Vikram Ramchand and Sonal Mukhi. BPB publications, 2002. 416 p.
12. Marrs T. Json at work. O'Reilly Media, Inc, USA, 2017. 376 p.
13. Blokdyk G. 5STARCook. WebSocket Second Edition., 2018. 280 p.
14. Tudor S. C# Programming: The Practical Beginner's Guide To Learn C# Programming In One Day Step-By-Step, Amazon Digital Services LLC, 2019. 119 p.
15. Noyes B. Data Binding with Windows Forms 2.0: Programming Smart Client Data Applications with .NET (Microsoft Windows Development Series). Addison-Wesley Professional, 2006. 736 p.
16. Cadenhead T. Socket.IO Cookbook. Packt Publishing, 2015. 184 p.
17. Nathan A. WPF 4.5 Unleashed. Sams Publishing, 2013. 864 p.
18. Griffiths I. Programming C# 8.0: Build Cloud, Web, and Desktop Applications. O'Reilly Media, 2019. 802 p.
19. Sells C., Griffiths J. Programming WPF: Building Windows UI with Windows Presentation Foundation. O'Reilly Media, 2007. 874 p.
20. Nathan A. XAML Unleashed. Sams Publishing, 2014. 512 p.

REFERENCES

1. Ryan Turner. (2020). *C#: 3 books in 1 The Ultimate Beginners, Intermediate and Expert Guide to Master C# Programming*. Nelly B.L. International Consulting Ltd, 636. (in English).
2. Brenda Jin. (2018). *Designing Web APIs: Building APIs That Developers Love*. O'Reilly Media, 232. (in English).
3. Arlen Feldman. (2008). *WPF in Action with Visual Studio 2008: Covers Visual Studio 2008, SP1 and .NET 3.5 SP1*. Manning Publications, 650. (in English).
4. James Snell, Doug Tidwell, Pavel Kulchenko. (2001). *Programming Web Services with SOAP*. O'Reilly & Associates, Inc, USA, 244. (in English).
5. Ben Smith. (2015). *Beginning JSON*. Apress, 326. (in English).
6. Andrew Lombardi. (2015). *WebSocket: Lightweight Client-Server Communications*. O'Reilly Media, 144. (in English).

7. Pranai Nandan. (2016). *Mastering SoapUI*. Packt Publishing, 240. (in English).
8. K. Scott Allen. (2012). *What Every Web Developer Should Know About HTTP*. OdeToCode LLC, 80. (in English).
9. David Gourley. (2002). *Http: The Definitive Guide (Definitive Guides)*. O'Reilly Media, 656. (in English).
10. Eric Newcomer. (2002). *Understanding Web Services: XML, WSDL, SOAP, and UDDI*. AddisonWesley Professional, 368. (in English).
11. Vikram Ramchand & Sonal Mukhi. (2002). *XML Web Services & Soap By Vikram Ramchand and Sonal Mukhi*. BPB publications, 416. (in English).
12. Tom Marrs. (2017). *Json at work*. O'Reilly Media, Inc, USA. 376 p. (in English).
13. Gerardus Blokdyk. (2018). *5STARCook*. WebSocket Second Edition, 280. (in English).
14. Steve Tudor. (2019). *C# Programming: The Practical Beginner's Guide To Learn C# Programming In One Day Step-By-Step*. Amazon Digital Services LLC, 119. (in English).
15. Brian Noyes. (2006). *Data Binding with Windows Forms 2.0: Programming Smart Client Data Applications with .NET (Microsoft Windows Development Series)*. Addison-Wesley Professional, 736. (in English).
16. Tyson Cadenhead. (2015). *Socket.IO Cookbook*. Packt Publishing, 184. (in English).
17. Adam Nathan. (2013). *WPF 4.5 Unleashed*. Sams Publishing, 864.
18. Ian Griffiths. (2019). *Programming C# 8.0: Build Cloud, Web, and Desktop Applications*. O'Reilly Media, 802. (in English).
19. Chris Sells & Ian Griffiths. (2007). *Programming WPF: Building Windows UI with Windows Presentation Foundation*. O'Reilly Media. 874. (in English).
20. Adam Nathan. (2014). *XAML Unleashed*. Sams Publishing, 512. (in English).

Мария Удалова,
(магистрант, Государственный университет инфраструктуры и технологий)
Елена Шикла, д.ф.-м.н.
(заведующая кафедрой информационных технологий, Государственный университет инфраструктуры и технологий)

МОДЕЛИРОВАНИЕ ВЗАИМОДЕЙСТВИЯ ЭЛЕКТРОМОБИЛЯ С ЗАРЯДНОЙ СТАНЦИЕЙ НА ОСНОВЕ РЕАЛИЗАЦИИ OPEN CHARGE POINT PROTOCOL

Исследована документация OCPP-протокола версии 1.6, сделан сравнительный анализ реализации этого протокола с помощью SOAP и JSON. На основе Windows Forms создано и протестировано приложение, в котором реализовано моделирование взаимодействия электромобиля с зарядной станцией на основе частичной реализации Open Charge Point Protocol на языке программирования C # с использованием JSON-формата и WebSocket.

Ключевые слова: электромобиль, электрозаправки, Open Charge Point Protocol, C #, JSON-формат, WebSocket.

Mariia Udalova,
(Magistrant, Department of Informatical technologies, State University of Infrastructure and Technologies)
Elena Shikula, Doctor of Physical and Mathematical Sciences, Professor
(Department of Informatical technologies, State University of Infrastructure and Technologies)

THE SIMULATION OF INTERCONNECTION OF THE VEHICLE MODEL AND THE CHARGING STATION ON THE BASIS OF REALIZATION OPEN CHARGE POINT PROTOCOL

Documentation of the OCPP protocol version 1.6 was investigated. A comparative analysis of the implementation of this protocol using SOAP and JSON was carried out. It showed that it is advisable to select OCPP JSON Specification to model the interaction of an electric vehicle with a charging station.

Functional requirements for client-side and server-side software was identified. The Precedence Diagram was constructed. The C# programming language, Windows Forms technology and the Visual

Studio integrated development environment were chosen to implement the functional part. The basic protocol is WebSocket using JSON files.

WPF technology uses a declarative XAML markup language. With these tools, an application design was created on both client and server sides. Frame animation was developed in the Adobe Photoshop graphics editor.

After combining the functional and visual parts, a workable application of the simulation of the interaction of the electric vehicle with the charging station was obtained. Many development scenarios were considered during the development: charging the electric vehicle in a positive scenario (working server, available port), processing such negative usage scenarios as a non-working server, or trying to start charging on an inaccessible port, or trying to disconnect in the unfinished charging process, etc. Ready-to-use application can be used to demonstrate the protocol and the charging process of an electric vehicle from a charging station.

Keywords: *electric car, charging station, Open Charge Point Protocol, C #, JSON format, WebSocket.*