A Formally Verified OCPP V2.01 Server for Electric Vehicle Charger Networks

Daniel H. McInnes
School of Information Technology and Electrical Engineering
The University of Queensland, Qld., 4072, Australia

Abstract

This paper describes the implementation of a formally verified OCPP v2.01 server for use in electric vehicle charger networks, using the SPARK 2014 programming language.

An attempt is made to automatically verify certain desirable properties, including functional correctness and the absence of run-time exceptions.

A technique is used to automatically generate SPARK code and reduce the time taken to implement the protocol.

1 Introduction

The number of publicly available electric vehicle fast chargers is growing rapidly. These chargers typically communicate with a server using the 'Open Charge Point Protocol' (OCPP), a freely available open standard developed by the Open Charge Alliance[2] . These servers should be error free, ideally never crashing. Numerous software verification tools exist which prove various desirable software properties. I compared these tools and selected 'SPARK 2014'[8] for the development of a minimal OCPP implementation.

2 OCPP v2.0.1

The specification for OCPP v2.01 is publicly available, and describes the protocol with the aid of 139 JSON files [6] , each of which specifies the format of a packet which is sent between a charger and a server. The protocol handles aspects of car charging such as pricing, transaction handling, and display and messaging support.

The specification covers many use cases that are not essential for a basic Charging Station Management System. It also includes a subset of the protocol sufficient for a basic implementation which is the goal of this project.

3 Software Verification Tools

Table 1: Desirable Software Properties

1	Functional Correctness
2	No memory leaks
3	No uninitialised memory access
4	No run-time exceptions
5	No stack overflows (bounded memory usage)
6	Never hangs (bounded execution time)
7	No undefined behaviour
8	Constraint of information flow

Ten different software verification tools were considered for the development of the server, including Dafny[9] , KeY[10] , OpenJML[11] , SPARK 2014[8] , Spec#[12] , VCC[13] , Verifast[14] , Viper[15] , Whiley[16] , and Why3[17] . Each of these tools were investigated to determine which of the desirable criteria (see Table 1) were satisfied.

4 SPARK 2014

After considering several different verification tools, SPARK 2014[8] (GNAT Pro community edition) was selected for implementation of the project. SPARK 2014 consists of both a formally defined programming language and a corresponding set of verification tools. SPARK code is compiled by the GNAT compiler, and analysed by the GNATprove tool [18] , which uses Why3 to generate verification conditions, and Alt-Ergo, CVC4, and Z3 to prove the theorems.

The GNAT Pro community edition is available under the GNU GPL license.

The level of static analysis can be divided into five levels (see Table 2).

Table 2: SPARK 2014 Levels of Static Analysis [20]

Level	**Guarantee**
Stone	Valid SPARK. This guarantees the absence of undefined behaviour.
Bronze	This guarantees that all data is initialised, and correct data flow, i.e. prevents unintended access to global variables.
Silver	This guarantees the absence of run time exceptions. This complies with certification standards of many safety critical domains, such as DO-178B/C (avionics), EN 50128 (rail), IEC 61508, ECSS-Q-ST-80C (space), IEC 60880 (nuclear), IEC 62304

	(medical), ISO 26262 (automotive).
Gold	This proves functional correctness for certain key properties. These properties are specified in terms of type predicates, preconditions, and postconditions.
Platinum	Full functional proof of requirements.

The OCPP implementation is at ‘gold’ level.

5 Automatically Generated Code

During the implementation of the protocol, it was noticed that many of the same steps were being repeated for each packet. The packets contain JSON formatted data consisting of attribute-value pairs and arrays. The JSON specifications for each packet are included in the OCPP v2.01 protocol [6] . A JavaScript utility was developed which parsed these JSON packet specifications, and automatically generated SPARK code to initialise, deserialize (i.e. receive text and parse into a binary format), and serialise (i.e. convert the binary format of a packet into text suitable for transmission). The script also generated postconditions for each packet.

6 Conclusion

The subset of the protocol required for a basic implementation [2] was not fully implemented. Of the features that were implemented, all of the desirable software properties listed in Table 1 were automatically verified by SPARK 2014, except for stack overflows.

SPARK 2014 is a practical choice for developing formally verified software.

References

[1] “Global EV Outlook 2019,” International Energy Agency, Paris, France, Tech. Rep., May 2019, Accessed: June 25, 2019. [Online]. Available: https://webstore.iea.org/global-ev-outlook-2019

[2] “OCA OCPP 2.0 Part 0 - Introduction”. OpenChargeAlliance.org. Accessed: June 22, 2019. [Online].
Available:https://www.openchargealliance.org/downloads/

[3] “OCA OCPP 2.0 Part 1 - Architecture & Topology”. OpenChargeAlliance.org. Accessed: June 22, 2019. [Online]. Available: https://www.openchargealliance.org/downloads/

[4] “OCA OCPP 2.0 Part 2 - Appendices”. OpenChargeAlliance.org. Accessed: June 22, 2019. [Online].
Available:
https://www.openchargealliance.org/downloads/

[5] “OCA OCPP 2.0 Part 2 - Specification”. OpenChargeAlliance.org. Accessed: June 22, 2019. [Online].
Available:
https://www.openchargealliance.org/downloads/

[6] “OCA OCPP 2.0 Part 3 - Schemas”. OpenChargeAlliance.org. Accessed: June 22, 2019. [Online].
Available:
https://www.openchargealliance.org/downloads/

[7] “OCA OCPP 2.0 Part 4 - OCPP-J Specification”. OpenChargeAlliance.org. Accessed: June 22, 2019. [Online].
Available:
https://www.openchargealliance.org/downloads/

[8] “GNAT Community Edition”. https://www.adacore.com/.
Accessed: May 16, 2020. [Online].
Available: https://www.adacore.com/download

[9] “Dafny: A Language and Program Verifier for Functional Correctness”. Microsoft. Accessed: Aug 17, 2019. [Online]. Available: https://www.microsoft.com/en-us/research/project/dafny-a-language-and-program-verifier-for-functional-correctness/

[10] W. Ahrendt, B. Beckert, R. Bubel, R. Hähnle, P. H. Schmitt, and M. Ulbrich, Eds., Deductive Software Verification - The KeY Book: From Theory to Practice, ser. Lecture Notes in Computer Science. Springer, 2016, vol. 10001. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-49812-6

[11] “Does your program do what it is supposed to do?”. OpenJML. Accessed: June 22, 2019. [Online]. Available: https://github.com/verifast/verifast/

[12] “Spec#”. Microsoft. Accessed: Aug 18, 2019. [Online]. Available: https://www.microsoft.com/en-us/research/project/spec/

[13] “VCC: A Verifier for Concurrent C”. Microsoft. Accessed: June 22, 2019. [Online]. Available:
https://www.microsoft.com/en-us/research/project/vcc-a-verifier-for-concurrent-c/

[14] “Research prototype tool for modular formal verification of C and Java programs”. VeriFast. Accessed: June 22, 2019. [Online]. Available: https://github.com/verifast/verifast/

[15] “Viper”. ETH Zurich. Accessed: Aug 18, 2019. [Online]. Available:
https://www.pm.inf.ethz.ch/research/viper.html

[16] “Whiley A Programming Language with Extended Static Checking”. Whiley.
Accessed: Aug 18, 2019. [Online].
Available: http://whiley.org/

[17] “Why3 Where Programs Meet Provers”. Why3. Accessed: Aug 18, 2019.
[Online]. Available: http://why3.lri.fr/

[18] “Formal Verification with GNATprove”. Adacore. Accessed: May 16, 2020. [Online].
Available:
https://docs.adacore.com/spark2014-docs/html/ug/en/gnatprove.html

[19] “Tractors and machinery for agriculture and forestry — Safety-related parts of control systems”. ISO. Accessed: May 17, 2020. [Online]. Available: https://www.iso.org/standard/69025.html
[20] “Implementation Guidance for the Adoption of SPARK”. AdaCore, Thales. Accessed: May 17, 2020. [Online]. Available: https://www.adacore.com/books/implementation-guidance-spark

Biography

The author has 20+ years of professional software engineering experience, and inadvertently became interested in software verification while trying to fulfil an obscure safety standard for agricultural and forestry machinery, ISO25119[19] .