THE UNIVERSITY OF QUEENSLAND

AUSTRALIA

A VERIFIED OCPP v2.01 SERVER FOR ELECTRIC VEHICLE CHARGER NETWORKS

by

DANIEL HUGH MCINNES

School of Information Technology and Electrical Engineering,
The University of Queensland.

Submitted for the degree of
Master of Engineering
in the field of Software Engineering

October 2019.

Daniel McInnes
s4231125@student.uq.edu.au

June 9, 2020

Prof Amin Abbosh
Acting Head of School
School of Information Technology and Electrical Engineering
The University of Queensland
St Lucia, Q 4072

Dear Professor Abbosh,

In accordance with the requirements of the degree of Master of Engineering in the division of Software Engineering, I present the following thesis entitled “A Verified Server for an Electric Vehicle Charger Network”. This work was performed under the supervision of A/Prof. Graeme Smith.

I declare that the work submitted in this thesis is my own, except as acknowledged in the text and footnotes, and has not been previously submitted for a degree at The University of Queensland or any other institution.
Yours sincerely,

Daniel McInnes.

Acknowledgments

I wish to acknowledge the support of my supervisor, Associate Professor Graeme Smith, whose expertise and assistance were greatly appreciated.

Abstract

Currently, networks of publicly available electric vehicle fast chargers communicate with servers using the Open Charge Point Protocol (OCPP). Ideally, the software running on these servers would be error free and never crash. Numerous software verification tools exist to prove desirable properties of the server software, such as functional correctness, the absence of race conditions, memory leaks, and certain runtime errors. I compare the different features of several verification tools, and use one of them to partially implement an OCPP server.

Contents

Acknowledgments v

Abstract vii

List of Figures xii

List of Tables xiii

1 Introduction 1

2 Prior Art 3
- 2.1 Dafny 3
  - 2.1.1 Home Page 3
  - 2.1.2 Features 3
  - 2.1.3 Soundness 3
  - 2.1.4 Supported Platforms 3
  - 2.1.5 License Information 4
  - 2.1.6 Evidence of successful use in commercial software development 4
  - 2.1.7 Existing Libraries 4
  - 2.1.8 Multithreaded Application Support 4
  - 2.1.9 Supported Languages 4
- 2.2 KeY 4
  - 2.2.1 Home Page 4
  - 2.2.2 Features 4
  - 2.2.3 Soundness 4
  - 2.2.4 Supported Platforms 5
  - 2.2.5 License Information 5
  - 2.2.6 Evidence of successful use in commercial software development 5
  - 2.2.7 Existing Libraries 5
  - 2.2.8 Multithreaded Application Support 5
  - 2.2.9 Supported Languages 5
- 2.3 OpenJML 5

2.3.1 Home Page 5
2.3.2 Features 5
2.3.3 Soundness 6
2.3.4 Supported Platforms 6
2.3.5 License Information 6
2.3.6 Evidence of successful use in commercial software development 6
2.3.7 Existing Libraries 6
2.3.8 Multithreaded Application Support 6
2.3.9 Supported Languages 6
2.4 SPARK 2014 6
2.5 Spec# 8
2.5.1 Home Page 8
2.5.2 Features 8
2.5.3 Soundness 9
2.5.4 Supported Platforms 9
2.5.5 License Information 9
2.5.6 Evidence of successful use in commercial software development 9
2.5.7 Existing Libraries 9
2.5.8 Multithreaded Application Support 9
2.5.9 Supported Languages 9
2.6 VCC 9
2.6.1 Home Page 9
2.6.2 Features 10
2.6.3 Soundness 10
2.6.4 Supported Platforms 10
2.6.5 License Information 10
2.6.6 Evidence of successful use in commercial software development 10
2.6.7 Existing Libraries 10
2.6.8 Multithreaded Application Support 10
2.6.9 Supported Languages 10
2.7 Verifast 11
2.7.1 Home Page 11
2.7.2 Features 11
2.7.3 Soundness 11
2.7.4 Supported Platforms 11
2.7.5 License Information 11
2.7.6 Evidence of successful use in commercial software development 11
2.7.7 Existing Libraries 11
2.7.8 Multithreaded Application Support 11

2.7.9 Supported Languages 11
2.8 Viper 12
2.8.1 Home Page 12
2.8.2 Features 12
2.8.3 Soundness 12
2.8.4 Supported Platforms 12
2.8.5 License Information 12
2.8.6 Evidence of successful use in commercial software development 12
2.8.7 Existing Libraries 12
2.8.8 Multithreaded Application Support 12
2.8.9 Supported Languages 13
2.9 Whiley 13
2.9.1 Home Page 13
2.9.2 Features 13
2.9.3 Soundness 13
2.9.4 Supported Platforms 13
2.9.5 License Information 13
2.9.6 Evidence of successful use in commercial software development 13
2.9.7 Existing Libraries 14
2.9.8 Multithreaded Application Support 14
2.9.9 Supported Languages 14
2.10 Why3 14
2.10.1 Home Page 14
2.10.2 Features 14
2.10.3 Soundness 14
2.10.4 Supported Platforms 14
2.10.5 License Information 15
2.10.6 Evidence of successful use in commercial software development 15
2.10.7 Existing Libraries 15
2.10.8 Multithreaded Application Support 15
2.10.9 Supported Languages 15
2.11 Conclusion of Review of Background and Associated Work 15

3 OCPP Server Implementation 16
3.1 OCPP Overview 16
3.2 Minimal OCPP implementation 16
3.2.1 Boot Notification Discussion 17
3.2.2 OCPP message types 21
3.2.3 Implementation 22

3.2.4 'Server.ReceivePacket' Function . . . 23
3.2.5 'OCPP.ParseMessageType' Function . . . 25
3.2.6 'Server.HandleRequest' Function . . . 26
3.2.7 'OCPP.ParseAction' Function . . . 27
3.2.8 'Server.HandleBootNotificationRequest' Function . . . 28
3.2.9 'BootNotificationRequest.parse' Function . . . 29
3.2.10 'BootNotificationRequest.Initialize' Function . . . 30
3.2.11 'ocpp.ParseMessageId' Function . . . 31
3.2.12 'Server.IsEnrolled' Function . . . 32
3.3 Automated Code Generation . . . 33
3.3.1 Automatically parsing BootNotificationRequest.json . . . 33

4 Results and discussion 56
4.1 Software Verification Goals . . . 57
4.1.1 Automatically verify the absence of memory leaks . . . 57
4.1.2 Automatically verify the absence of uninitialized memory reads 57
4.1.3 Automatically prove that the program can never crash or exit unexpectedly . . . 57
4.1.4 Automatically verify the program meets its requirements . . . 58
4.1.5 Automatically verify the absence of undefined behaviour . . . 58
4.1.6 Automatically verify the constraint of information flow . . . 58
4.1.7 Soundness . . . 58
4.2 OCPP Functionality . . . 58

5 Conclusions 60
5.1 Summary and conclusions . . . 60
5.2 Possible future work . . . 60

Appendices 61

A Appendix 62
A.1 Dafny . . . 62

B Companion disk 65

List of Figures

1.1 Publicly available fast chargers 1

3.1 OCPP overview. 16
3.2 Cold Booting a Charging Station [73]. 18
3.3 ‘Boot Notification’ class diagram 19
3.4 Cold Booting a Charging Station [73]. 24

List of Tables

3.1 Use cases for a basic implementation 17
3.2 CALL message type [74] 21
3.3 CALLRESULT message type [74] 21
3.4 CALLERROR message type [74] 22

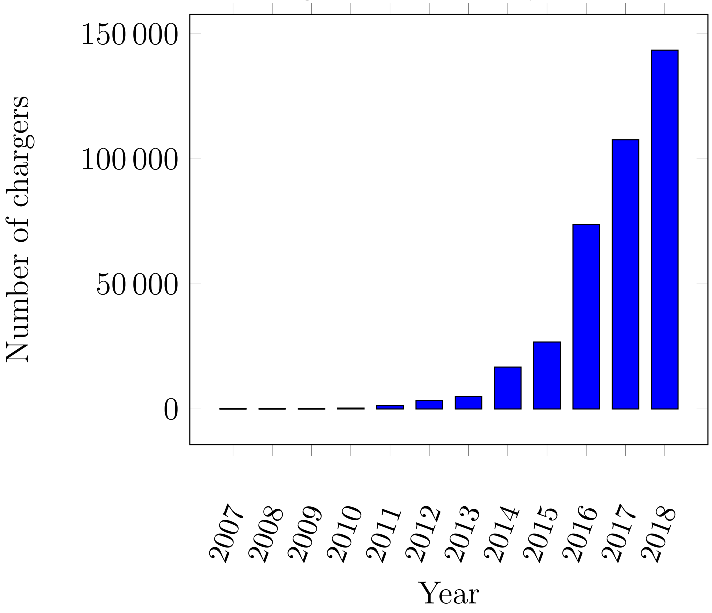
4.1 Use cases for a basic implementation 59

Chapter 1

Introduction

The International Energy Agency [1] reports that the number of publicly available fast chargers (> 22kW) increased from 107 650 in 2017 to 143 502 in 2018.

Figure 1.1: Publicly available fast chargers



The Open Charge Alliance [69] reports that more than 10 000 charging stations[1] in over 50 countries are managed using the OCPP protocol.

In this paper I investigate and compare the features of several different software verifiation tools and choose one to partially implement an OCPP server. The tools include Dafny, KeY, OpenJML, SPARK 2014, Spec#, VCC, VeriFast, Viper, Whiley, and Why3.

[1] Note that a "charging station" may consist of multiple fast chargers, thus the disparity between 143 502 "fast chargers" and "more than 10 000 charging stations"

The tools vary in what guarantees they provide. Desirable guarantees that I was interested in are:

- the absence of memory leaks
- the program should never access uninitialized memory (for example, should never read past the end of an array)
- the program should never crash or exit unexpectedly
- the tool should verify the absence of stack overflows, i.e. the program should be bounded in terms of memory (RAM) usage at runtime
- the program should verifiably meet its requirements. These requirements are typically in the form of preconditions and postconditions.
- the program should never exhibit undefined behaviour
- the program should constrain information flow, i.e. not leak sensitive information such as passwords
- The tool should be sound. Many of the tools claim to be sound “modulo bugs in the tool”, and have lengthy lists of known bugs.

Chapter 2

Prior Art

Numerous software verification tools were considered for use in implementing the OCPP server. These tools include Dafny, KeY, OpenJML, SPARK 2014, Spec#, VCC, Verifast, Viper, Whiley, and Why3. SPARK 2014 was found to best fulfill the criteria 1.

2.1 Dafny

2.1.1 Home Page

https://rise4fun.com/Dafny

2.1.2 Features

Dafny is both a language and a verifier [62]. Dafny supports feature verification via preconditions, postconditions, loop invariants and loop variants. It uses the ‘Boogie’ intermediate language and the Z3 theorem prover. The Dafny compiler produces executable files for the .NET platform [64].

2.1.3 Soundness

Dafny is designed to be sound but incomplete, and is known to report errors on correct programs [65].

2.1.4 Supported Platforms

Windows, Linux, OSX host, for a .NET target platform.

2.1.5 License Information

MIT.

2.1.6 Evidence of successful use in commercial software development

Internet searches failed to find any evidence of Dafny being used in commercial software development.

2.1.7 Existing Libraries

There is a 'mathematics' library for Dafny.

2.1.8 Multithreaded Application Support

Internet searches failed to find evidence of multithreaded application support.

2.1.9 Supported Languages

Dafny.

2.2 KeY

2.2.1 Home Page

https://www.key-project.org/

2.2.2 Features

KeY offers functional verification for Java programs. The specifications are written as comments in JML in the Java source code. KeY is built on a formal logic called 'Java Card DL', which is itself a first-order dynamic logic, and an extension of Hoare logic. It is targeted at JavaCard programs.

2.2.3 Soundness

KeY is thought to be sound. Internet searches failed to find examples of unsoundness.

2.2.4 Supported Platforms

Windows, Linux, and OSX hosts, for a JVM target.

2.2.5 License Information

GPL.

2.2.6 Evidence of successful use in commercial software development

Internet searches failed to find any evidence of KeY being used in commercial software development.

2.2.7 Existing Libraries

There are extensive libraries available for Java programs.

2.2.8 Multithreaded Application Support

No.

2.2.9 Supported Languages

Java.

2.3 OpenJML

2.3.1 Home Page

http://www.openjml.org/

2.3.2 Features

OpenJML is a suite of tools for verifying Java programs that are annotated with JML statements. It is based on OpenJDKv1.8. It detects illegal memory access at compile time. It verifies preconditions and postconditions. It arguably guarantees the absence of undefined behaviour for single threaded applications. It does not constrain information flow.

2.3.3 Soundness

Yes

2.3.4 Supported Platforms

Windows, Linux, OSX.

2.3.5 License Information

GPLv2.

2.3.6 Evidence of successful use in commercial software development

Internet searches failed to find any evidence of OpenJML being used in commercial software development.

2.3.7 Existing Libraries

There are extensive libraries available for Java programs.

2.3.8 Multithreaded Application Support

No.

2.3.9 Supported Languages

Java (only OpenJDK v1.8, may become unsupported in December 2020)

2.4 SPARK 2014

SPARK 2014 is both a formally defined programming language and a set of verification tools. In typical use, a programmer writes SPARK code, which is compiled by the GNAT compiler, then analyzed by the GNATprove tool to produce numerous verification conditions. GNATprove uses Alt-Ergo, CVC4 and Z3 to prove the verification conditions.

Features

Formally verifies:

- information flow

- freedom from runtime errors, except ‘StorageError’ (stack overflow / heap exhaustion)
- functional correctness

Safety Standards

SPARK 2014 satisifes:

- DO-178B/C
- Formal Methods supplement DO-333
- CENELEC 51028
- IEC 61508
- DEFSTAN 00-56

Soundness

SPARK is thought to be sound. Internet searches failed to find examples of unsoundness.

Supported Platforms

Windows, Linux, OSX.

License Information

Dual license:

SPARK GPL is available for free from http://libre.adacore.com under the GPL.

SPARK PRO is available under a commercial license from http://www.adacore.com.

Evidence of successful use in commercial software development

There is abundant evidence of the successful use of SPARK in high integrity software development. See: [31], [28], [29], [2], [8], [23], [39], [5], [38], [17], [4], [10], [44], [25], [32], [37], [34], [22], [21], [19], [35], [36], [43], [24], [12], [33], [20], [14], [40], [26], [27], [42], [9], [46], [30], [7], [11], [47], [13], [6], [16], [15], [45], [18], [3].

Of particular relevance is the experience of the CubeSat Laboratory at Vermont Technical College[41]. Cubesats are small cubes launched into space with various sensors onboard, in this case without post-launch software update capabilities. This means the software must be fault free at the time of launch. The students (mostly third and fourth year undergraduates, with no prior knowledge of SPARK or Ada, and a high turnover rate) proved the software to be free of runtime errors. 14 Cubesats were launched in November 2013. Most were never heard from again, but the SPARK Cubesat worked for 2 years until it reentered Earth's atmosphere as planned in November 2015.

Existing Libraries

SPARK has a minimal container library. SPARK interfaces easily with Ada, which has an extensive standard library.

Multithreaded Application Support

Unsupported.

Supported Languages

SPARK 2104 supports a subset of Ada 2012.

See [63, p.18]

"The following Ada 2012 features are not currently supported by Spark:

Aliasing of names; no object may be referenced by multiple names

Goto statements

Expressions or functions with side effects

Exception handlers

Controlled types; types that provide fine control of object creation, assignment, and destruction

Tasking/multithreading (will be included in future releases)"

2.5 Spec#

2.5.1 Home Page

https://www.microsoft.com/en-us/research/project/spec/

2.5.2 Features

Spec# consists of the Spec# programming language, the Spec# compiler, and the Spec# static program verifier. The programming language is an extension of C#, adding non-null types, checked exceptions, preconditions, postconditions, and object invariants [66]. It uses Boogie for verification.

2.5.3 Soundness

Spec# claims to be sound[50].

2.5.4 Supported Platforms

Windows host platform, .NET target platform.

2.5.5 License Information

Internet searches failed to find Spec# license information.

2.5.6 Evidence of successful use in commercial software development

Internet searches failed to find evidence of Spec# being used in commercial software development.

2.5.7 Existing Libraries

Internet searches failed to find Spec# libraries. However, Spec# offers interoperability with the .NET platform, which has an extensive standard library, although the soundness of this library is not guaranteed.

2.5.8 Multithreaded Application Support

Yes[66].

2.5.9 Supported Languages

Spec#.

2.6 VCC

2.6.1 Home Page

https://www.microsoft.com/en-us/research/project/vcc-a-verifier-for-concurrent-c/

2.6.2 Features

VCC verifies preconditions and postconditions written in the form of special comments in the source code. It detects data races in multithreaded applications and illegal memory access at compile time. It does not guarantee the absence of undefined behaviour.

2.6.3 Soundness

Yes

2.6.4 Supported Platforms

Windows

2.6.5 License Information

MIT [68]

2.6.6 Evidence of successful use in commercial software development

VCC has been used successfully in at least one major commercial project, the verification of the Microsoft Hypervisor, the virtualization kernel of Hyper-V [53].

2.6.7 Existing Libraries

Internet searches failed to find libraries verified with VCC. However, applications written with VCC can link against unverified libraries, effectively giving access to extensive library support.

2.6.8 Multithreaded Application Support

Yes

2.6.9 Supported Languages

C

2.7 Verifast

2.7.1 Home Page

https://github.com/verifast/verifast

2.7.2 Features

Verifast verifies preconditions, and postconditions in the form of special comments in the source code. It detects race conditions in multithreaded applications. It does not guarantee the absence of undefined behaviour, or verify the absence of stack overflows.

2.7.3 Soundness

No, see https://github.com/verifast/verifast/blob/master/soundness.md

2.7.4 Supported Platforms

Windows, Linux, OSX.

2.7.5 License Information

MIT [61].

2.7.6 Evidence of successful use in commercial software development

There is evidence of some use in industrial applications[77].

2.7.7 Existing Libraries

Internet searches failed to find libraries verified by / written with VeriFast annotations. However, applications written with VeriFast can link against unverified libraries, effectively gaining access to extensive library support.

2.7.8 Multithreaded Application Support

Yes.

2.7.9 Supported Languages

C, Java

2.8 Viper

2.8.1 Home Page

https://www.pm.inf.ethz.ch/research/viper.html

2.8.2 Features

Viper consists of the Viper intermediate verification language, automatic verifiers, and example front end tools [57]. It is more of a tool for creating other verification tools than a tool for verifying software. In practice, a developer would write code in Python and use the 'Nagini' front end (based on Viper) to verify the code [55]. A corresponding front end for Rust exists, 'Prusti' [49].

2.8.3 Soundness

Yes.

2.8.4 Supported Platforms

Windows, MacOS, Linux[60].

2.8.5 License Information

https://bitbucket.org/viperproject/carbon/src/default/LICENSE.txt Mozilla Public License Version 2.0[56]

2.8.6 Evidence of successful use in commercial software development

Internet searches failed to find evidence of Viper being used in commercial software development.

2.8.7 Existing Libraries

Internet searches failed to find libraries verified by Viper front ends. However, applications written with these front ends can link against unverified libraries, effectively gaining access to extensive library support.

2.8.8 Multithreaded Application Support

Yes.

2.8.9 Supported Languages

Python, Rust, Java, OpenCL, Chalice.

2.9 Whiley

2.9.1 Home Page

http://whiley.org/

2.9.2 Features

Whiley consists of the Whiley language, the Whiley Build System, the Whiley Compiler, the Whiley Intermediate Language, the Whiley-2-Java Compiler, the Whiley-2-C Compiler, and the Whiley Constraint Solver[76].

Whiley uses a variant of first-order logic called the Whiley Assertion Language for verification.

In typical use, a developer will write source code in Whiley, build with the Whiley Build system, and execute the resulting Java class file on the JVM.

2.9.3 Soundness

Internet searches did not find evidence to indicate that Whiley is unsound.

2.9.4 Supported Platforms

JVM.

2.9.5 License Information

BSD.

2.9.6 Evidence of successful use in commercial software development

Internet searches failed to find evidence of Whiley being used in commercial software development.

2.9.7 Existing Libraries

Internet searches failed to find libraries verified by Whiley. However, applications written with Whiley can link against unverified Java functions, effectively gaining access to extensive library support.

2.9.8 Multithreaded Application Support

No.

2.9.9 Supported Languages

Whiley. The Whiley-2-Java Compiler (WyJC) can convert verified Whiley programs into JVM class files. Whiley can import Java functions, and export Whiley functions for use in Java programs.

2.10 Why3

2.10.1 Home Page

http://why3.lri.fr/

2.10.2 Features

The Why3 deductive program verification platform includes the WhyML language, a standard library of logical theories, and basic programming data structures[83]. In typical use, a developer will write software in WhyML, and get correct-by-construction OCaml programs through an automated extraction mechanism. It verifies preconditions and postconditions.

WhyML is also used by numerous popular verification tools (FramaC, SPARK2014, Krakatoa) as an intermediate language for verification of C, Java and Ada programs.

2.10.3 Soundness

Yes.

2.10.4 Supported Platforms

Windows, Linux, OSX. Why3 is distributed as a Debian package and as an OPAM package.

2.10.5 License Information

GNU LGPL 2.1.

2.10.6 Evidence of successful use in commercial software development

Internet searches failed to find evidence of WhyML being used in commercial software development. However, there are countless cases of commercial software development using WhyML as an intermediate language, as it is used by FramaC, SPARK, and others.

2.10.7 Existing Libraries

Why3 comes with a standard library of logical theories and basic programming data structures.

2.10.8 Multithreaded Application Support

No.

2.10.9 Supported Languages

WhyML.

2.11 Conclusion of Review of Background and Associated Work

Based on the properties of the verification tools available, it has been decided to use SPARK 2014 for the implementation and verification of the OCPP server. It has wide use in industry, which gives me confidence that it is a practical choice. It verifies all of the properties that are important to me, and has good library support.

Chapter 3

OCPP Server Implementation

3.1 OCPP Overview

An OCPP server acts as a websocket server. There are typically many individual charging stations, which act as websocket clients. The clients establish a websocket connection with the server, and once this is complete, JSON formatted packets are exchanged between the client and the server.

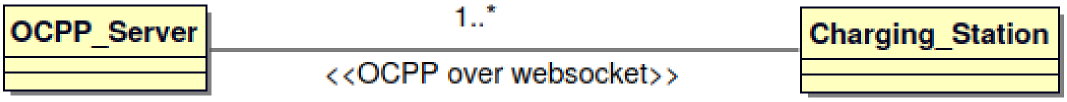


Figure 3.1: OCPP overview.

3.2 Minimal OCPP implementation

The OCPP protocol supports many use cases which are not required for a basic implementation. The following table [70] shows a minimal subset of OCPP messages required to support basic functionality.

Table 3.1: Use cases for a basic implementation

Functionality	Use Case	Messages
Booting a charge station	B01 - B04	BootNotification
Configuring a charge station	B05-B07	SetVariables, GetVariables, GetReportBase
Resetting a charge station	B11-B12	Reset
Authorization Options	One of C01, C02, C04	Authorize
Transaction Mechanism	E01 (one of S1-S6), E02-E03, E05, E06 (one of S1-S6), E07- E08, One of E09-E10, E11-E13	TransactionEvent
Availability	G01, G03-G04	ChangeAvailability, StatusNotification
Monitoring Events	G05, N07	NotifyEvent
Meter Values	J02	TransactionEvent
Data Transfer	P01-P02	DataTransfer

3.2.1 Boot Notification Discussion

To understand what interesting software properties may be proved, we will walk through a typical 'BootNotification' sequence.

Whenever a charger is powered on in the field, it establishes a websocket connection with the server, and sends a 'BootNotificationRequest' packet to announce itself. A high level view of a typical boot notification sequence is given in figure 3.2.

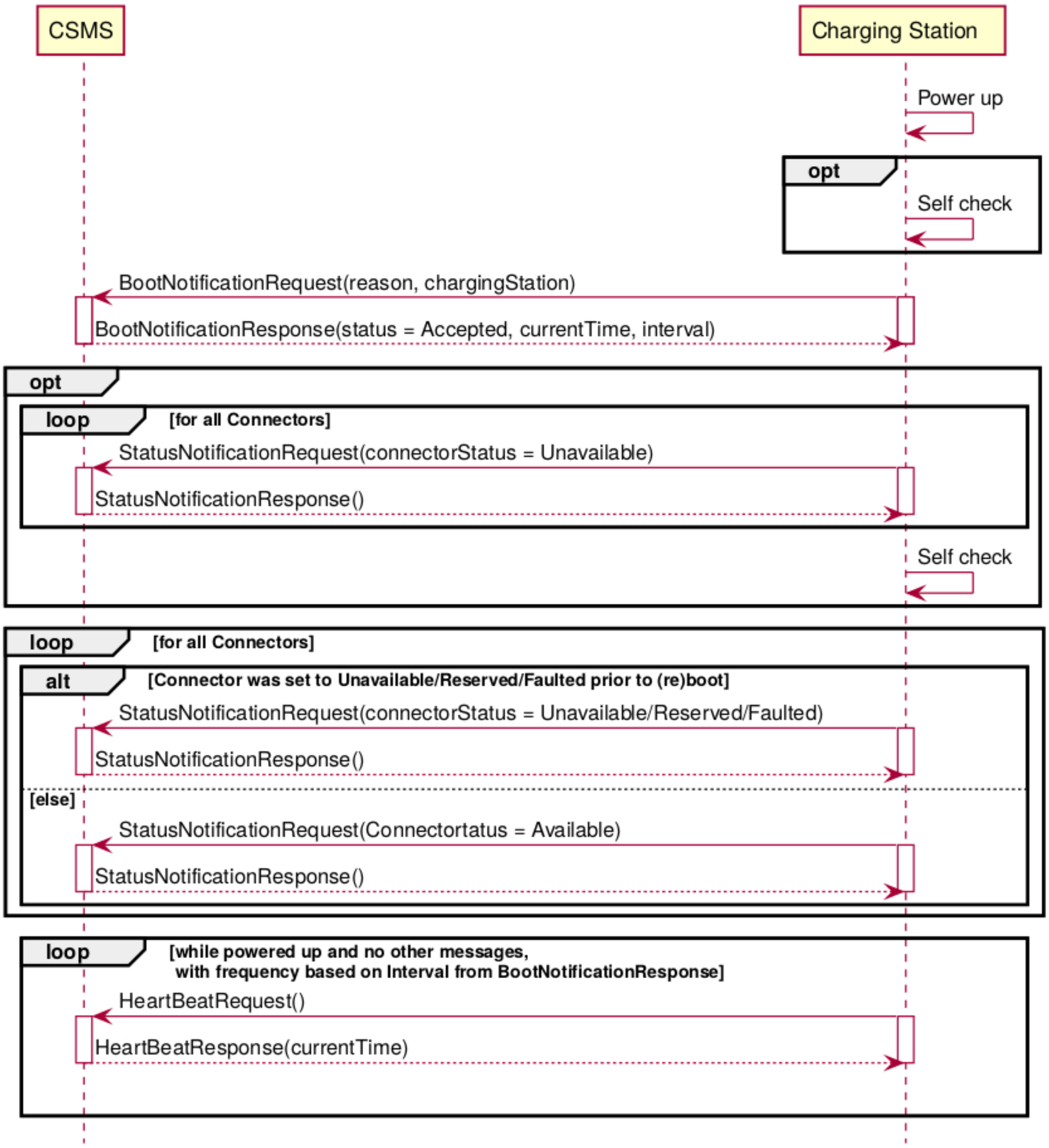

CSMS
Charging Station
Power up
opt
Self check
BootNotificationRequest(reason, chargingStation)
BootNotificationResponse(status = Accepted, currentTime, interval)
opt
loop
[for all Connectors]
StatusNotificationRequest(connectorStatus = Unavailable)
StatusNotificationResponse()
Self check
loop
[for all Connectors]
alt
[Connector was set to Unavailable/Reserved/Faulted prior to (re)boot]
StatusNotificationRequest(connectorStatus = Unavailable/Reserved/Faulted)
StatusNotificationResponse()
[else]
StatusNotificationRequest(Connectortatus = Available)
StatusNotificationResponse()
loop
[while powered up and no other messages,
with frequency based on Interval from BootNotificationResponse]
HeartBeatRequest()
HeartBeatResponse(currentTime)

Figure 3.2: Cold Booting a Charging Station [73].

The ‘BootNotificationRequest’ packet contains information describing the charger. If the server recognises the charger, it returns a ‘BootNotificationResponse’ message with a status of ‘Accepted’. The contents of these packets is described in figure 3.3.

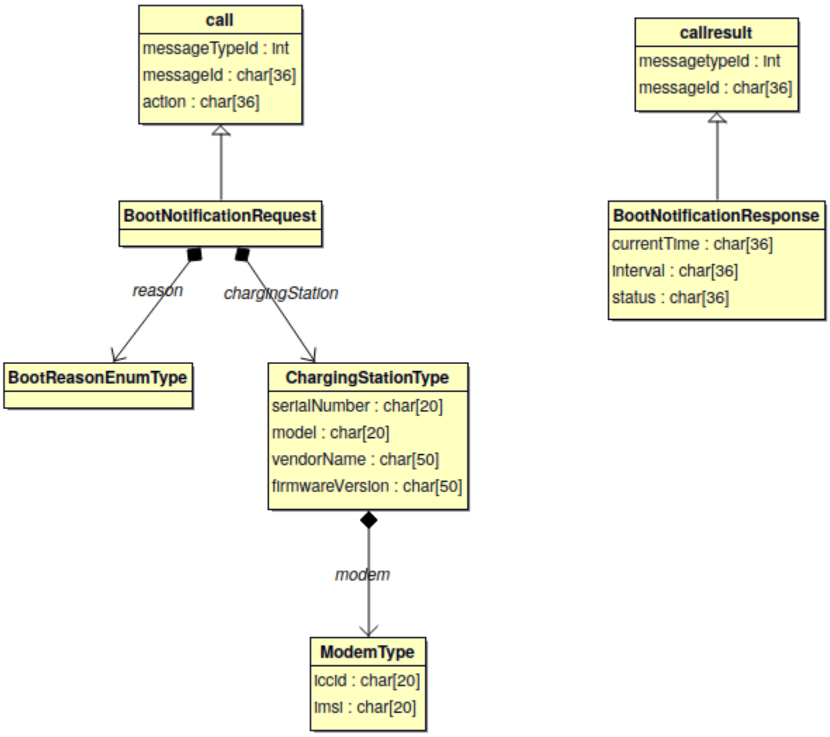


Figure 3.3: ‘Boot Notification’ class diagram

Note that figure 3.3 is a UML class diagram, the different arrow types have special meanings. The ‘BootNotificationRequest’ class inherits from the ‘call’ class, and has a member variable called ‘reason’ of class ‘BootReasonEnumType’. It also has a member variable called ‘chargingStation’ of class ‘ChargingStationType’, which in turn has a member variable called ‘modem’ of class ‘ModemType’.

The ‘BootNotificationRequest’ packet arrives at the server as a big blob of text, as follows:

```
[2,        // ‘2’ indicates that this is a ‘request’
"19223202",// this is the message ID, and associates requests with responses
"BootNotification", // the ‘action’
{
   "chargingStation":
   {
      "serialNumber": "00000000000000000001",
      "model": "SingleSocketCharger",
      "modem":
      {
         "iccid": "01234567890123456789",
         "imsi": "01234567890123456789"
      },
      "vendorName": "VendorX",
      "firmwareVersion": "01.23456789"
   },
   "reason":"PowerUp"
}
]
```

The server’s job is to correctly parse this text, determine what kind of packet it is, and respond appropriately. In this example, the server converts the blob of incoming text into a ‘BootNotificationRequest’ packet, builds a ‘BootNotificationResponse’ packet, converts it to a blob of text, and sends it back over the websocket to the charger. The reponse looks like this:

```
[3,// ‘3’ indicates that this is a ‘response’
"19223202",// this responds to request "19223202"
{
    "currentTime": "2013-02-01T20:53:32.486Z",
    "interval":  300,
    "status":"Accepted"
}
]
```

3.2.2 OCPP message types

Valid OCPP message types are either CALL, CALLRESULT, or CALLERROR.

CALL

A CALL consists of 4 parts, as described in 3.2

Field	**Data Type**	**Meaning**
MessageTypeId	Integer (=2)	This is a Message Type Number which is used to identify the type of the message.
MessageId	string[36]	This is a unique identifier that will be used to match request and result.
Action	string[36]	The name of the remote procedure or action. This field SHALL contain a case-sensitive string. The field SHALL contain the OCPP Message name without the "Request" suffix. For example: For a "BootNotificationRequest", this field shall be set to "BootNotification".
Payload	JSON	JSON Payload of the action.

Table 3.2: CALL message type [74]

CALLRESULT

A CALL consists of 4 parts, as described in 3.3

Field	**Data Type**	**Meaning**
MessageTypeId	Integer (=3)	This is a Message Type Number which is used to identify the type of the message.
MessageId	string[36]	This must be the exact same ID that is in the call request so that the recipient can match request and result.
Payload	JSON	JSON Payload of the action.

Table 3.3: CALLRESULT message type [74]

CALLERROR

The CALLERROR message is only used when an error occurs during message transport, or in response to an invalid OCPP message. A CALL consists of 5 parts, as described in 3.4.

Field	Data Type	Meaning
MessageTypeId	Integer (=4)	This is a Message Type Number which is used to identify the type of the message.
MessageId	string[36]	This must be the exact same ID that is in the call request so that the recipient can match request and result.
Action	string[36]	The name of the remote procedure or action. This field SHALL contain a case-sensitive string. The field SHALL contain the OCPP Message name without the "Request" suffix. For example: For a "BootNotificationRequest", this field shall be set to "BootNotification".
Payload	JSON	JSON Payload of the action.

Table 3.4: CALLERROR message type [74]

3.2.3 Implementation

The sequence diagram 3.4 describes the typical sequence of events that occurs when a charger is powered on, boots up and connects to a server. Prior to the charger booting, the charger is enrolled on the server. The server maintains a list of known chargers. Only chargers on this list may successfully start an OCPP session.

Once a charger powers up, it establishes a websocket connection with the server and communicates with OCPP packets over this link. The first OCPP message the charger sends is a 'BootNotificationRequest'.

When the server receives a blob of text, it arrives in a function called 'ReceivePacket'.

3.2.4 'Server.ReceivePacket' Function

```
procedure ReceivePacket(theServer: in out ocpp.server.T;
                        msg: in NonSparkTypes.packet.Bounded_String;
                        response: out NonSparkTypes.packet.Bounded_String;
                        valid: out Boolean)
  with
    Global => null,
    Annotate => (GNATprove, Terminating),
    Depends => (
                valid => (msg),
                response => (msg, theServer),
                theServer => (msg, theServer)
               );
```

Global Data Dependencies

The line '*Global* ⇒ *null*' specifies that the procedure may not modify any global variables.

'Annotate' section

By default, GNATprove does not prove termination of subprograms. The line '*Annotate* ⇒ (*GNATprove*, *Terminating*)' specifies that the procedure must terminate, i.e. can never enter an endless loop.

'Depends' section

The 'Depends' clause means that:

- the final value of 'valid' depends only on 'msg'. It makes sense that the validity of an OCPP packet depends only on the content of the message, and is independent of the state of the server.

- the final value of 'response' depends on 'msg' and on 'theServer'. It makes intuitive sense that the response depends on what the request was. It may not be immediately obvious why the response should depend on the state of the server, until we consider that the server maintains a list of known chargers, and will respond to BootNotificationRequests with a 'Rejected' status for unknown chargers.

- the final value of 'theServer' depends on the initial state of 'theServer' and on 'msg'.

GNATprove reports an error if any of these specifications are not met by the implementation.

The first thing the function 'ReceivePacket' does is parse the 'MessageType', as seen in 3.4.

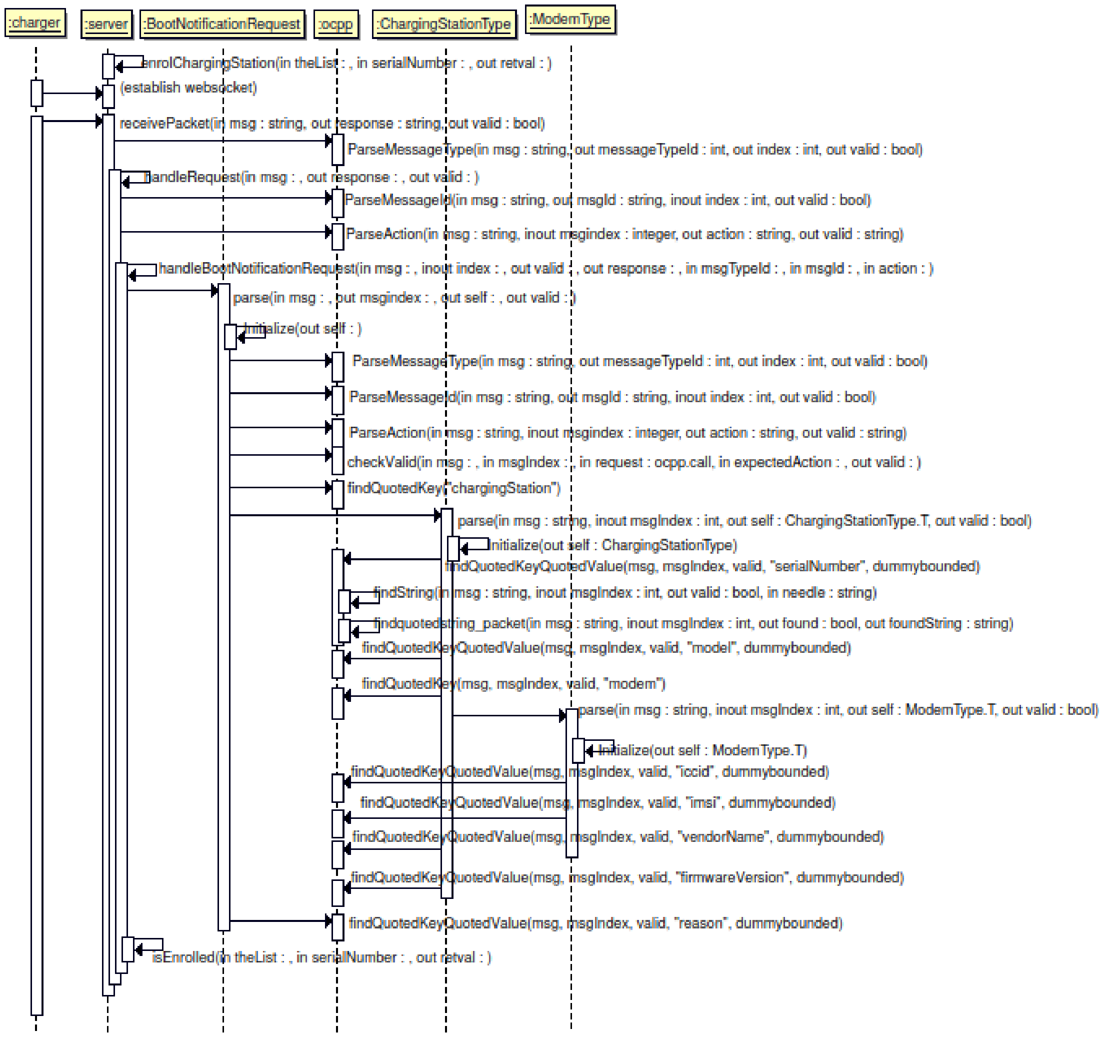

:charger
:server
:BootNotificationRequest
:ocpp
:ChargingStationType
:ModemType
enrolChargingStation(in theList : , in serialNumber : , out retval :)
(establish websocket)
receivePacket(in msg : string, out response : string, out valid : bool)
ParseMessageType(in msg : string, out messageTypeId : int, out index : int, out valid : bool)
handleRequest(in msg : , out response : , out valid :)
ParseMessageId(in msg : string, out msgId : string, inout index : int, out valid : bool)
ParseAction(in msg : string, inout msgindex : integer, out action : string, out valid : string)
handleBootNotificationRequest(in msg : , inout index : , out valid : , out response : , in msgTypeId : , in msgId : , in action :)
parse(in msg : , out msgindex : , out self : , out valid :)
Initialize(out self :)
ParseMessageType(in msg : string, out messageTypeId : int, out index : int, out valid : bool)
ParseMessageId(in msg : string, out msgId : string, inout index : int, out valid : bool)
ParseAction(in msg : string, inout msgindex : integer, out action : string, out valid : string)
checkValid(in msg : , in msgIndex : , in request : ocpp.call, in expectedAction : , out valid :)
findQuotedKey("chargingStation")
parse(in msg : string, inout msgIndex : int, out self : ChargingStationType.T, out valid : bool)
Initialize(out self : ChargingStationType)
findQuotedKeyQuotedValue(msg, msgIndex, valid, "serialNumber", dummybounded)
findString(in msg : string, inout msgIndex : int, out valid : bool, in needle : string)
findquotedstring_packet(in msg : string, inout msgIndex : int, out found : bool, out foundString : string)
findQuotedKeyQuotedValue(msg, msgIndex, valid, "model", dummybounded)
findQuotedKey(msg, msgIndex, valid, "modem")
parse(in msg : string, inout msgIndex : int, out self : ModemType.T, out valid : bool)
Initialize(out self : ModemType.T)
findQuotedKeyQuotedValue(msg, msgIndex, valid, "iccid", dummybounded)
findQuotedKeyQuotedValue(msg, msgIndex, valid, "imsi", dummybounded)
findQuotedKeyQuotedValue(msg, msgIndex, valid, "vendorName", dummybounded)
findQuotedKeyQuotedValue(msg, msgIndex, valid, "firmwareVersion", dummybounded)
findQuotedKeyQuotedValue(msg, msgIndex, valid, "reason", dummybounded)
isEnrolled(in theList : , in serialNumber : , out retval :)

Figure 3.4: Cold Booting a Charging Station [73].

3.2.5 ‘OCPP.ParseMessageType’ Function

```
procedure ParseMessageType(msg:   in  NonSparkTypes.packet.Bounded_String;
                           messagetypeid : out integer;-- eg. 2
                           index: in out Integer;
                           valid: out Boolean)
  with  Global => null,
  Annotate => (GNATprove, Terminating),
  Post => (if valid = true then
             (messagetypeid = 2 or messagetypeid = 3 or messagetypeid = 4)
           and
             (index < NonSparkTypes.packet.Length(msg))
          );
```

Global Data Dependencies

The line ‘*Global* ⇒ *null*’ specifies that the procedure may not modify any global variables.

‘Annotate’ section

By default, GNATprove does not prove termination of subprograms. The line ‘*Annotate* ⇒ (*GNATprove*, *Terminating*)’ specifies that the procedure must terminate, i.e. can never enter an endless loop.

‘Post’ section

The ‘Post’ clause means that:

- if the function returns ‘valid = true’, then the ‘messagetypeid’ must be one of the valid OCPP message types, i.e. 2, 3 or 4.
- if the function returns ‘valid = true’, then the ‘index’ must not have moved past the end of ‘msg’. This postcondition is included as a sanity check.

GNATprove reports an error if any of these specifications are not met by the implementation.

Once we have successfully parsed the message type, we know whether it is a ‘request’ or a ‘response’. In this example, the server has received a ‘BootNotificationRequest’, so we pass the message to ‘HandleRequest’ for further parsing.

3.2.6 'Server.HandleRequest' Function

```
procedure HandleRequest(theServer: in ocpp.server.T;
                        msg: in NonSparkTypes.packet.Bounded_String;
                        msgindex: in out Integer;
                        response: out NonSparkTypes.packet.Bounded_String;
                        valid: out Boolean)
  with
    global => null,
    Annotate => (GNATprove, Terminating),
    Depends => (
                valid => (msg, msgindex),
                msgindex => (msg, msgindex),
                response => (msg, msgindex, theServer)
               ),
    Post => (if valid = true then msgindex <= NonSparkTypes.packet.Length(msg));
```

'Global' section

The line '*Global ⇒ null*' means that the procedure does not modify any global state.

'Annotate' section

By default, GNATprove does not prove termination of subprograms. The line '*Annotate ⇒ (GNATprove, Terminating)*' specifies that the procedure must terminate, i.e. can never enter an endless loop.

'Depends' section

- The final value of 'valid' depends only on 'msg' and 'msgindex'. The reason 'valid' depends on 'msgindex' is that if the index was already pointing at the end of the message, there would not be enough information left to construct a valid request.

- The final value of 'msgindex' depends only on 'msg' and 'msgindex'. This makes sense, as if the message was invalid, the function will return early, without parsing much of the message, thus affecting the final value of 'index'. The reason the final value of 'msgindex' depends on the initial value of 'msgindex' is that if the index was already pointing at the end of the message, the function would return without advancing 'msgindex' through the message.

- The value of 'response' depends on 'msg', 'msgindex' and 'theServer'.

‘Post’ section

The ‘Post’ clause means that:

- if the function returns ‘valid = true’, then the ‘index’ must not have moved past the end of ‘msg’. This postcondition is included as a sanity check.

GNATprove reports an error if any of these specifications are not met by the implementation.

The body of ‘HandleRequest()’ calls ‘ParseAction’ to determine what kind of request it is.

3.2.7 ‘OCPP.ParseAction’ Function

```
procedure ParseAction(msg:   in  NonSparkTypes.packet.Bounded_String;
                      msgindex: in out Integer;
                      action : out NonSparkTypes.action_t.Bounded_String;
                      valid: out Boolean
                     )
  with  Global => null,
  Annotate => (GNATprove, Terminating),
  Post => (if valid = true
             then
               (msgindex < NonSparkTypes.packet.Length(msg))
          );
```

‘Global’ section

The line ‘*Global ⇒ null*’ means that the procedure does not modify any global state.

‘Annotate’ section

By default, GNATprove does not prove termination of subprograms. The line ‘*Annotate ⇒ (GNATprove, Terminating)*’ specifies that the procedure must terminate, i.e. can never enter an endless loop.

‘Depends’ section

- The final value of ‘valid’ depends only on ‘msg’ and ‘index’. The reason ‘valid’ depends on ‘index’ is that if the index was already pointing at the end of the message, there would not be enough information left to construct a valid request.

- The final value of ‘index’ depends only on ‘msg’ and ‘index’. This makes sense, as if the message was invalid, the function will return early, without parsing much of the message, thus affecting the final value of ‘index’. The reason the final value of ‘index’ depends on the initial value of ‘index’ is that if the index was already pointing at the end of the message, the function would return without advancing ‘index’ through the message.
- The value of ‘response’ depends on ‘msg’, ‘index’ and ‘theServer’.

‘Post’ section

The ‘Post’ clause means that:

- if the function returns ‘valid = true’, then the ‘index’ must not have moved past the end of ‘msg’. This postcondition is included as a sanity check.

GNATprove reports an error if any of these specifications are not met by the implementation.

In this case, the action is ‘BootNotification’, so ‘HandleRequest’ calls ‘HandleBootNotificationRequest’.

3.2.8 ‘Server.HandleBootNotificationRequest’ Function

```
procedure HandleBootNotificationRequest(
 theServer: in ocpp.server.T;
        msg: in NonSparkTypes.packet.Bounded_String;
        index : out Integer;
        valid: out Boolean;
        response: out NonSparkTypes.packet.Bounded_String)
  with
    global => null,
    Annotate => (GNATprove, Terminating),
    Depends => (
                index => (msg),
                valid => (msg),
                response => (msg, theServer)
               );
```

'Global' section

The line '*Global* ⇒ *null*' means that the procedure does not modify any global state.

'Annotate' section

By default, GNATprove does not prove termination of subprograms. The line '*Annotate* ⇒ (*GNATprove*, *Terminating*)' specifies that the procedure must terminate, i.e. can never enter an endless loop.

'Depends' section

- The final value of 'valid' depends only on 'msg'.
- The final value of 'index' depends only on 'msg'.
- The value of 'response' depends on 'msg' and 'theServer'.

GNATprove reports an error if any of these specifications are not met by the implementation.

The first thing this function does is call 'BootNotificationRequest.parse'.

3.2.9 'BootNotificationRequest.parse' Function

```
   procedure parse(msg: in NonSparkTypes.packet.Bounded_String;
                   msgindex:  out Integer;
                   self: out ocpp.BootNotificationRequest.T;
                   valid: out Boolean
                  )
   with
    Global => null,
    Annotate => (GNATprove, Terminating),
    Depends => (
                   valid => (msg),
                   msgindex => (msg),
                   self  => (msg)
),
    post => (if valid = true then
                  (self.messagetypeid = 2) and
                  (NonSparkTypes.messageid_t.Length(self.messageid) > 0) and
                  (self.action = action) -- prove that the original packet contains
```

```
        );
```

‘Global’ section

The line ‘*Global* ⇒ *null*’ means that the procedure does not modify any global state.

‘Annotate’ section

By default, GNATprove does not prove termination of subprograms. The line ‘$Annotate \Rightarrow (GNATprove, Terminating)$’ specifies that the procedure must terminate, i.e. can never enter an endless loop.

‘Depends’ section

- The final value of ‘valid’ depends only on ‘msg’.
- The final value of ‘msgindex’ depends only on ‘msg’.
- The value of ‘self’ depends on ‘msg’ and ‘theServer’. ‘Self’ refers to the ‘BootNotificationRequest’ object.

GNATprove reports an error if any of these specifications are not met by the implementation.

The first thing this function does is call ‘Initialise’, which sets all member variables to a known state.

3.2.10 ‘BootNotificationRequest.Initialize’ Function

```
procedure Initialize(self: out ocpp.BootNotificationRequest.T)
with
 Global => null,
 Annotate => (GNATprove, Terminating),
 Depends => (self => null);
```

‘Global’ section

The line ‘*Global* ⇒ *null*’ means that the procedure does not modify any global state.

‘Annotate’ section

By default, GNATprove does not prove termination of subprograms. The line ‘$Annotate \Rightarrow (GNATprove, Terminating)$’ specifies that the procedure must terminate, i.e. can never enter an endless loop.

‘Depends’ section

- The final value of ‘self’ does not depend on any other state.

GNATprove reports an error if any of these specifications are not met by the implementation.

This function always sets member variables to the same values, and returns to ‘BootNotificationRequest.parse()’.

Next, ocpp.ParseMessageId() is called.

3.2.11 ‘ocpp.ParseMessageId’ Function

```
procedure ParseMessageId(msg:   in  NonSparkTypes.packet.Bounded_String;
                         messageid : out NonSparkTypes.messageid_t.Bounded_Str
                         index: in out Integer;
                         valid: out Boolean
                        )     with
    global => null,
    Annotate => (GNATprove, Terminating);
```

‘Global’ section

The line ‘*Global* ⇒ *null*’ means that the procedure does not modify any global state.

‘Annotate’ section

By default, GNATprove does not prove termination of subprograms. The line ‘$Annotate \Rightarrow (GNATprove, Terminating)$’ specifies that the procedure must terminate, i.e. can never enter an endless loop. GNATprove reports an error if any of these specifications are not met by the implementation.

This function parses the ‘messageId’, which is used to match ocpp requests with ocpp responses. It stores the result in a member variable, and returns to ‘BootNotificationRequest.parse(), which then calls additonal utility parsing functions until all of the ‘BootNotificationRequest’ member variables have been populated with the corresponding values from the incoming message. The function then returns to ‘server.HandleBootNotificationRequest()’, which calls ‘server.IsEnrolled()’.

3.2.12 ‘Server.IsEnrolled’ Function

```
procedure IsEnrolled(theList: in ChargerList.vecChargers_t;
                     serialNumber: in NonSparkTypes.ChargingStationType.strserialN
                     retval: out Boolean)
  with
    global => null,
    Annotate => (GNATprove, Terminating),
    Depends => (
                  retval => (serialNumber, theList)
               );
```

‘Global’ section

The line ‘*Global* ⇒ *null*’ means that the procedure does not modify any global state.

‘Annotate’ section

By default, GNATprove does not prove termination of subprograms. The line ‘*Annotate* ⇒ (*GNATprove*, *Terminating*)’ specifies that the procedure must terminate, i.e. can never enter an endless loop.

‘Depends’ section

- The final value of ‘retval’ depends on the serial number being checked, and the list of allowed serial numbers.

GNATprove reports an error if any of these specifications are not met by the implementation.

If the list contains the supplied serial number, it returns true, otherwise false. In this case, the serial number is in the list, so we respond to the BootNotificationRequest with a BootNotificationResponse with a status of ‘accepted’. This response packet is serialized into text, which is then sent to the charger. The server’s job is complete, it has received a valid request, parsed it, created an appropriate response, and sent it back to the client.

The basic sequence of events is similar for all OCPP packets.

3.3 Automated Code Generation

The OCPP protocol defines 128 different messages. Some of these packets are deeply nested JSON structures, with lists of elements each containing further lists of elements. It was realised early in the project that the time required to manually implement serialising and deserialising (converting from raw text to a binary representation, and converting from the binary representation back to raw text) for each of these packets would be prohibitive. The protocol documentation includes the corresponding JSON definitions of all of these packets, each with their own ‘.js’ file (eg. ‘BootNotificationRequest.json’, ‘BootNotificationResponse.json’.

A ‘node.js’ script ‘parse.js’ was written to parse each of these ‘.json’ files, and generate the corresponding SPARK implemenation and definition files (eg. ‘BootNotificationRequest.ads’, ‘BootNotificationRequest.adb’, ‘BootNotificationResponse.ads’, ‘BootNotificationResponse.adb’). ‘BootNotificationRequest.json’ is examined here in detail.

3.3.1 Automatically parsing BootNotificationRequest.json

The JSON definition for BootNotificationRequest is as follows:

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "urn:OCPP:Cp:2:2019:12:BootNotificationRequest",
  "comment": "Errata sheet - release candidate",
  "definitions": {
    "CustomDataType": {
      "description": "This class does not get ’AdditionalProperties = false’
      in the schema generation, so it can be extended with arbitrary JSON
      properties to allow adding custom data.",
      "javaType": "CustomData",
      "type": "object",
      "properties": {
        "vendorId": {
          "type": "string",
          "maxLength": 255
        }
      },
      "required": [
        "vendorId"
      ]
    },
    "BootReasonEnumType": {
      "description": "This contains the reason for sending this message
      to the CSMS.\r\n",
      "javaType": "BootReasonEnum",
      "type": "string",
      "additionalProperties": false,
      "enum": [
        "ApplicationReset",
        "FirmwareUpdate",
        "LocalReset",
        "PowerUp",
        "RemoteReset",
        "ScheduledReset",
        "Triggered",
        "Unknown",
        "Watchdog"
      ]
    },
    "ChargingStationType": {
      "description": "Charge_ Point\r\nurn:x-oca:ocpp:uid:2:233122\r\n
      The physical system where an Electrical Vehicle (EV) can be charged.\r\n",
      "javaType": "ChargingStation",
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "customData": {
          "$ref": "#/definitions/CustomDataType"
        },
        "serialNumber": {
          "description": "Device. Serial_ Number. Serial_ Number\r\n
          urn:x-oca:ocpp:uid:1:569324\r\nVendor-specific device identifier.\r\n",
          "type": "string",
          "maxLength": 25
        },
        "model": {
          "description": "Device. Model. CI20_ Text\r\n
          urn:x-oca:ocpp:uid:1:569325\r\nDefines the model of the device.\r\n",
          "type": "string",
          "maxLength": 20
        },
        "modem": {
          "$ref": "#/definitions/ModemType"
        },
        "vendorName": {
          "description": "Identifies the vendor (not necessarily in a
          unique manner).\r\n",
          "type": "string",
          "maxLength": 50
        },
        "firmwareVersion": {
          "description": "This contains the firmware version of the
          Charging Station.\r\n\r\n",
          "type": "string",
          "maxLength": 50
        }
      },
      "required": [
        "model",
        "vendorName"
      ]
    },
    "ModemType": {
      "description": "Wireless_ Communication_ Module\r\n
      urn:x-oca:ocpp:uid:2:233306\r\nDefines parameters required for initiating
      and maintaining wireless communication with other devices.\r\n",
      "javaType": "Modem",
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "customData": {
          "$ref": "#/definitions/CustomDataType"
        },
        "iccid": {
          "description": "Wireless_ Communication_ Module. ICCID.
          CI20_ Text\r\nurn:x-oca:ocpp:uid:1:569327\r\n
          This contains the ICCID of the modem’s SIM card.\r\n",
          "type": "string",
          "maxLength": 20
        },
        "imsi": {
          "description": "Wireless_ Communication_ Module. IMSI. CI20_ Text\r\n
          urn:x-oca:ocpp:uid:1:569328\r\nThis contains the IMSI of the modem’s
          SIM card.\r\n",
          "type": "string",
          "maxLength": 20
        }
      }
    }
  },
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "customData": {
      "$ref": "#/definitions/CustomDataType"
    },
    "chargingStation": {
      "$ref": "#/definitions/ChargingStationType"
    },
    "reason": {
      "$ref": "#/definitions/BootReasonEnumType"
    }
  },
  "required": [
    "reason",
    "chargingStation"
  ]
}
```

The parsing script starts with the ‘definitions’ section, which contains additional definitions for ‘CustomDataType’, ‘BootReasonEnumType’, ‘ChargingStationType’ (which itself contains further subtype definitions), and ‘ModemType’.

CustomDataType

This curious addition to the packet definition defines an optional extra string for extending the protocol functionality. All of the packet JSON definitions have this field, but it is not referenced anywhere else in the protocol specification. It is ignored in this OCPP implementation.

BootReasonEnumType

The following JSON snippet contains the definition for an enumerated type.

```
{
    "BootReasonEnumType": {
      "description": "This contains the reason for sending this message to the CS
      "javaType": "BootReasonEnum",
      "type": "string",
      "additionalProperties": false,
      "enum": [
        "ApplicationReset",
        "FirmwareUpdate",
        "LocalReset",
        "PowerUp",
        "RemoteReset",
        "ScheduledReset",
        "Triggered",
        "Unknown",
        "Watchdog"
      ]
    }
```

'parse.js' generates the following SPARK implementation and definition files to serialise and deserialise this enum.

BootReasonEnumType.ads

```
{
-- start ocppBootReasonEnumType.ads
with Ada.Strings.Bounded;

package ocpp.BootReasonEnumType is
   type T is (
      ApplicationReset,
      FirmwareUpdate,
      LocalReset,
      PowerUp,
      RemoteReset,
      ScheduledReset,
      Triggered,
      Unknown,
      Watchdog
   );

   package string_t is new Ada.Strings.Bounded.Generic_Bounded_Length(Max => 16);
   procedure FromString(str : in String;
                        attribute : out T;
                        valid : out Boolean)
   with
      Global => null,
      Annotate => (GNATprove, Terminating);

   procedure ToString(attribute : in T;
                      str : out string_t.Bounded_String)
   with
      Global => null,
      Annotate => (GNATprove, Terminating);
end ocpp.BootReasonEnumType;
-- end ocpp-BootReasonEnumType.ads
```

BootReasonEnumType.adb

```
{
-- ocpp-BootReasonEnumType.adb

with ocpp.BootReasonEnumType; use ocpp.BootReasonEnumType;
with NonSparkTypes;

package body ocpp.BootReasonEnumType is
   procedure FromString(str : in String;
                        attribute : out T;
                        valid : out Boolean)
   is
   begin
      if (NonSparkTypes.Uncased_Equals(str, "ApplicationReset")) then
         attribute := ApplicationReset;
      elsif (NonSparkTypes.Uncased_Equals(str, "FirmwareUpdate")) then
         attribute := FirmwareUpdate;
      elsif (NonSparkTypes.Uncased_Equals(str, "LocalReset")) then
         attribute := LocalReset;
      elsif (NonSparkTypes.Uncased_Equals(str, "PowerUp")) then
         attribute := PowerUp;
      elsif (NonSparkTypes.Uncased_Equals(str, "RemoteReset")) then
         attribute := RemoteReset;
      elsif (NonSparkTypes.Uncased_Equals(str, "ScheduledReset")) then
         attribute := ScheduledReset;
      elsif (NonSparkTypes.Uncased_Equals(str, "Triggered")) then
         attribute := Triggered;
      elsif (NonSparkTypes.Uncased_Equals(str, "Unknown")) then
         attribute := Unknown;
      elsif (NonSparkTypes.Uncased_Equals(str, "Watchdog")) then
         attribute := Watchdog;
      else
         valid := false;
         return;
      end if;
      valid := true;
   end FromString;

   procedure ToString(attribute : in T;
                      str : out string_t.Bounded_String)
   is
      use string_t;
   begin
      case attribute is
         when ApplicationReset => str := To_Bounded_String("ApplicationReset");
         when FirmwareUpdate => str := To_Bounded_String("FirmwareUpdate");
         when LocalReset => str := To_Bounded_String("LocalReset");
         when PowerUp => str := To_Bounded_String("PowerUp");
         when RemoteReset => str := To_Bounded_String("RemoteReset");
         when ScheduledReset => str := To_Bounded_String("ScheduledReset");
         when Triggered => str := To_Bounded_String("Triggered");
         when Unknown => str := To_Bounded_String("Unknown");
         when Watchdog => str := To_Bounded_String("Watchdog");
      end case;
   end ToString;
end ocpp.BootReasonEnumType;
```

Note that the specification includes several verification conditions, specifying that the ‘FromString’ and ‘ToString’ procedures may not modify global state and must terminate.

ChargingStationType

The following JSON snippet contains the definition for a more complex type, with properties defined by references to other types.

```
{
    "ChargingStationType": {
      "description": "Charge_ Point\r\nurn:x-oca:ocpp:uid:2:233122\r\nThe physical sy
      "javaType": "ChargingStation",
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "customData": {
          "$ref": "#/definitions/CustomDataType"
        },
        "serialNumber": {
          "description": "Device. Serial_ Number. Serial_ Number\r\nurn:x-oca:ocpp:ui
          "type": "string",
          "maxLength": 25
        },
        "model": {
          "description": "Device. Model. CI20_ Text\r\nurn:x-oca:ocpp:uid:1:569325\r\
          "type": "string",
          "maxLength": 20
        },
        "modem": {
          "$ref": "#/definitions/ModemType"
        },
        "vendorName": {
          "description": "Identifies the vendor (not necessarily in a unique manner).
          "type": "string",
          "maxLength": 50
        },
        "firmwareVersion": {
          "description": "This contains the firmware version of the Charging Station.
          "type": "string",
          "maxLength": 50
        }
      },
      "required": [
        "model",
        "vendorName"
      ]
    }
```

'parse.js' generates the following SPARK implementation and definition files to serialise and deserialise this type.

ChargingStationType.ads

```
{
pragma SPARK_mode (on);

with Ada.Strings.Fixed; use Ada.Strings.Fixed;
with NonSparkTypes; use NonSparkTypes.action_t;
with ocpp; use ocpp;
with ocpp.ModemType; use ocpp.ModemType;

package ocpp.ChargingStationType is
   type T is record
      zzzArrayElementInitialized : Boolean := False;
      serialNumber :
         NonSparkTypes.ChargingStationType.strserialNumber_t.Bounded_String;
      model : NonSparkTypes.ChargingStationType.strmodel_t.Bounded_String;
      modem : ModemType.T;
      vendorName :
         NonSparkTypes.ChargingStationType.strvendorName_t.Bounded_String;
      firmwareVersion :
         NonSparkTypes.ChargingStationType.strfirmwareVersion_t.Bounded_String;
   end record;
   procedure Initialize(self: out ocpp.ChargingStationType.T)
   with
    Global => null,
    Annotate => (GNATprove, Terminating),
    Depends => (self => null);
   procedure parse(msg: in NonSparkTypes.packet.Bounded_String;
                msgindex:  in out Integer;
                self: out ocpp.ChargingStationType.T;
                valid: out Boolean
               )
   with
    Global => null,
    Annotate => (GNATprove, Terminating),
    Depends => (
                valid => (msg, msgindex),
                msgindex => (msg, msgindex),
                self  => (msg, msgindex)
           );

   procedure To_Bounded_String(Self: in T;
                               retval: out NonSparkTypes.packet.Bounded_String)
      with
 Global => null,
 Annotate => (GNATprove, Terminating);
end ocpp.ChargingStationType;
```

ChargingStationType.adb

```
pragma SPARK_mode (on);

with ocpp;
with ocpp.ChargingStationType;
with Ada.Strings; use Ada.Strings;

package body ocpp.ChargingStationType is

procedure findquotedstring_packet is new findquotedstring(
   Max => NonSparkTypes.packet.Max_Length,
   string_t => NonSparkTypes.packet.Bounded_String,
   length => NonSparkTypes.packet.Length,
   To_String => NonSparkTypes.packet.to_string,
   To_Bounded_String =>  NonSparkTypes.packet.To_Bounded_String);

procedure Initialize(self: out ocpp.ChargingStationType.T)
is
```

```
begin
   NonSparkTypes.put_line("Initialize()");
   self.zzzArrayElementInitialized := False;
   self.serialNumber :=
      NonSparkTypes.ChargingStationType.strserialNumber_t.To_Bounded_String("");
   self.model :=
      NonSparkTypes.ChargingStationType.strmodel_t.To_Bounded_String("");
   ModemType.Initialize(self.modem);
   self.vendorName :=
   NonSparkTypes.ChargingStationType.strvendorName_t.To_Bounded_String("");
   self.firmwareVersion :=
   NonSparkTypes.ChargingStationType.strfirmwareVersion_t.To_Bounded_String("");
end Initialize;

procedure parse(msg:   in  NonSparkTypes.packet.Bounded_String;
                msgindex: in out Integer;
                self: out ocpp.ChargingStationType.T;
                valid: out Boolean)
is
   dummybounded:
   NonSparkTypes.packet.Bounded_String :=
      NonSparkTypes.packet.To_Bounded_String("");
   dummyInt: integer;
begin
   Initialize(self);
   ocpp.findQuotedKeyQuotedValue(msg, msgIndex, valid, "serialNumber",
      dummybounded);
   if (valid = false)
   then
      NonSparkTypes.put_line("333 Invalid ChargingStationTypeserialNumber");
      return;
   end if;

   self.serialNumber :=
      NonSparkTypes.ChargingStationType.strserialNumber_t.To_Bounded_String(
         NonSparkTypes.packet.To_String(dummybounded), Drop => Right);

   ocpp.findQuotedKeyQuotedValue(msg, msgIndex, valid, "model", dummybounded);
   if (valid = false)
```

```
then
   NonSparkTypes.put_line("333 Invalid ChargingStationTypemodel");
   return;
end if;

self.model :=
   NonSparkTypes.ChargingStationType.strmodel_t.To_Bounded_String(
      NonSparkTypes.packet.To_String(dummybounded), Drop => Right);

ocpp.findQuotedKey(msg, msgIndex, valid, "modem");
if (valid = false)
then
   NonSparkTypes.put_line("355 Invalid ChargingStationTypemodem");
   return;
end if;

ModemType.parse(msg, msgindex, self.modem, valid);
if (valid = false)
then
   NonSparkTypes.put_line("357 Invalid ChargingStationTypemodem");
   return;
end if;

ocpp.findQuotedKeyQuotedValue(msg, msgIndex, valid, "vendorName",
   dummybounded);

if (valid = false)
then
   NonSparkTypes.put_line("333 Invalid ChargingStationTypevendorName");
   return;
end if;

self.vendorName :=
      NonSparkTypes.ChargingStationType.strvendorName_t.To_Bounded_String(
         NonSparkTypes.packet.To_String(dummybounded), Drop => Right);

ocpp.findQuotedKeyQuotedValue(msg, msgIndex, valid,
   "firmwareVersion", dummybounded);
   if (valid = false)
   then
      NonSparkTypes.put_line("333 Invalid ChargingStationTypefirmwareVersion");
      return;
   end if;

   self.firmwareVersion :=
      NonSparkTypes.ChargingStationType.strfirmwareVersion_t.To_Bounded_String(
            NonSparkTypes.packet.To_String(dummybounded), Drop => Right);

   if (valid = false)
   then
      NonSparkTypes.put_line("365 Invalid ChargingStationTypefirmwareVersion");
      return;
   end if;

   valid := true;
end parse;

procedure To_Bounded_String(Self: in T;
                               retval: out NonSparkTypes.packet.Bounded_String)
is
   dummybounded: NonSparkTypes.packet.Bounded_String :=
      NonSparkTypes.packet.To_Bounded_String("");
   strmodem : NonSparkTypes.packet.Bounded_String;
begin
   ModemType.To_Bounded_String(Self.modem, strmodem);
   retval := NonSparkTypes.packet.To_Bounded_String(""
      & "{" & ASCII.LF
      & "    " & ’"’ & "serialNumber" & ’"’ & ": " & ’"’ &
      NonSparkTypes.ChargingStationType.strserialNumber_t.To_String(
         Self.serialNumber) &
      ’"’ & "," & ASCII.LF
      & "    " & ’"’ & "model" & ’"’ & ": " & ’"’ &
      NonSparkTypes.ChargingStationType.strmodel_t.To_String(Self.model) &
      ’"’ & "," & ASCII.LF
      & "    " & ’"’ & "modem" & ’"’ & ":" &
      NonSparkTypes.packet.To_String(strmodem) & "," & ASCII.LF
      & "    " & ’"’ & "vendorName" & ’"’ & ": " & ’"’ &
      NonSparkTypes.ChargingStationType.strvendorName_t.To_String(Self.vendorName) &
      ’"’ & "," & ASCII.LF
      & "    " & ’"’ & "firmwareVersion" & ’"’ & ": " & ’"’ &
      NonSparkTypes.ChargingStationType.strfirmwareVersion_t.To_String(
         Self.firmwareVersion) &
      ’"’ & ASCII.LF
      & "}" & ASCII.LF, Drop => Right);
   end To_Bounded_String;
end ocpp.ChargingStationType;
```

Note that the specification for ‘ChargingStationType’ depends on ‘ModemType’, which is defined in another file. The code generation script automatically adds the statement ‘with ocpp.ModemType; use ocpp.ModemType;’ to the specification file, to allow compilation to succeed. Verification conditions are automatically added to prove termination, absence of access to global data, and data flow dependencies.

ModemType

The following JSON snippet contains the definition for ‘ModemType’, which is used by the ‘BootNotificationRequest’ packet.

```
    "ModemType": {
      "description": "Wireless_ Communication_ Module\r\nurn:x-oca:ocpp:uid:2:233306\
      "javaType": "Modem",
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "customData": {
          "$ref": "#/definitions/CustomDataType"
        },
        "iccid": {
          "description": "Wireless_ Communication_ Module. ICCID. CI20_ Text\r\nurn:x
          "type": "string",
          "maxLength": 20
        },
        "imsi": {
          "description": "Wireless_ Communication_ Module. IMSI. CI20_ Text\r\nurn:x-
          "type": "string",
          "maxLength": 20
        }
      }
    }
  }
```

‘parse.js’ generates the following SPARK implementation and definition files to serialise and deserialise this type.

ModemType.ads

```
pragma SPARK_mode (on);

with Ada.Strings.Fixed; use Ada.Strings.Fixed;
with NonSparkTypes; use NonSparkTypes.action_t;
with ocpp; use ocpp;

package ocpp.ModemType is
   type T is record
      zzzArrayElementInitialized : Boolean := False;
      iccid : NonSparkTypes.ModemType.striccid_t.Bounded_String;
      imsi : NonSparkTypes.ModemType.strimsi_t.Bounded_String;
   end record;
   procedure Initialize(self: out ocpp.ModemType.T)
   with
    Global => null,
    Annotate => (GNATprove, Terminating),
    Depends => (self => null);

   procedure parse(msg: in NonSparkTypes.packet.Bounded_String;
                msgindex:  in out Integer;
                self: out ocpp.ModemType.T;
                valid: out Boolean
               )
   with
    Global => null,
    Annotate => (GNATprove, Terminating),
    Depends => (
                valid => (msg, msgindex),
                msgindex => (msg, msgindex),
                self  => (msg, msgindex)
            );
   procedure To_Bounded_String(Self: in T;
                               retval: out NonSparkTypes.packet.Bounded_String)
      with
 Global => null,
 Annotate => (GNATprove, Terminating);
end ocpp.ModemType;
```

ModemType.adb

```
pragma SPARK_mode (on);

with ocpp;
with ocpp.ModemType;
with Ada.Strings; use Ada.Strings;

package body ocpp.ModemType is

procedure findquotedstring_packet is new findquotedstring(
   Max => NonSparkTypes.packet.Max_Length,
   string_t => NonSparkTypes.packet.Bounded_String,
   length => NonSparkTypes.packet.Length,
   To_String => NonSparkTypes.packet.to_string,
   To_Bounded_String =>  NonSparkTypes.packet.To_Bounded_String);

   procedure Initialize(self: out ocpp.ModemType.T)
   is
   begin
      NonSparkTypes.put_line("Initialize()");
      self.zzzArrayElementInitialized := False;
      self.iccid := NonSparkTypes.ModemType.striccid_t.To_Bounded_String("");
      self.imsi := NonSparkTypes.ModemType.strimsi_t.To_Bounded_String("");
   end Initialize;

   procedure parse(msg:   in  NonSparkTypes.packet.Bounded_String;
                   msgindex: in out Integer;
                   self: out ocpp.ModemType.T;
                   valid: out Boolean
                  )
   is
```

```
      dummybounded: NonSparkTypes.packet.Bounded_String :=
         NonSparkTypes.packet.To_Bounded_String("");
      dummyInt: integer;
   begin
      Initialize(self);
      ocpp.findQuotedKeyQuotedValue(msg, msgIndex, valid, "iccid",
         dummybounded);
      if (valid = false)
      then
         NonSparkTypes.put_line("333 Invalid ModemTypeiccid");
         return;
      end if;

      self.iccid :=
         NonSparkTypes.ModemType.striccid_t.To_Bounded_String(
            NonSparkTypes.packet.To_String(dummybounded), Drop => Right);

      ocpp.findQuotedKeyQuotedValue(msg, msgIndex, valid, "imsi",
         dummybounded);
      if (valid = false)
      then
         NonSparkTypes.put_line("333 Invalid ModemTypeimsi");
         return;
      end if;

      self.imsi := NonSparkTypes.ModemType.strimsi_t.To_Bounded_String(
         NonSparkTypes.packet.To_String(dummybounded), Drop => Right);

      if (valid = false)
      then
         NonSparkTypes.put_line("365 Invalid ModemTypeimsi");
         return;
      end if;
      valid := true;
   end parse;

   procedure To_Bounded_String(Self: in T;
                               retval: out NonSparkTypes.packet.Bounded_String)
   is
```

```
      dummybounded: NonSparkTypes.packet.Bounded_String :=
         NonSparkTypes.packet.To_Bounded_String("");
   begin
      retval := NonSparkTypes.packet.To_Bounded_String(""
         & "{" & ASCII.LF
         & "    " & '"' & "iccid" & '"' & ": " & '"' &
         NonSparkTypes.ModemType.striccid_t.To_String(Self.iccid) &
         '"' & "," & ASCII.LF &
         "    " & '"' & "imsi" & '"' & ": " & '"' &
         NonSparkTypes.ModemType.strimsi_t.To_String(Self.imsi) & '"' & ASCII.LF
         & "}" & ASCII.LF, Drop => Right);
   end To_Bounded_String;
end ocpp.ModemType;
```

BootNotificationRequest

The following JSON snippet contains the definition for ‘BootNotificationRequest’, which depends on the previously defined types ‘ChargingStationType’ and ‘BootReasonEnumType’.

```
"type": "object",
"additionalProperties": false,
"properties": {
  "customData": {
    "$ref": "#/definitions/CustomDataType"
  },
  "chargingStation": {
    "$ref": "#/definitions/ChargingStationType"
  },
  "reason": {
    "$ref": "#/definitions/BootReasonEnumType"
  }
},
"required": [
  "reason",
  "chargingStation"
]
```

‘parse.js’ generates the following SPARK implementation and definition files to serialise and deserialise this type.

BootNotificationRequest.ads

```
pragma SPARK_mode (on);

with Ada.Strings.Fixed; use Ada.Strings.Fixed;
with NonSparkTypes; use NonSparkTypes.action_t;
with ocpp; use ocpp;
with ocpp.ChargingStationType; use ocpp.ChargingStationType;
with ocpp.BootReasonEnumType; use ocpp.BootReasonEnumType;

package ocpp.BootNotificationRequest is
   action : constant NonSparkTypes.action_t.Bounded_String :=
      NonSparkTypes.action_t.To_Bounded_String("BootNotification");
   type T is new call with record
      chargingStation : ChargingStationType.T;
      reason : BootReasonEnumType.T;
   end record;
   procedure Initialize(self: out ocpp.BootNotificationRequest.T)
   with
    Global => null,
    Annotate => (GNATprove, Terminating),
    Depends => (self => null);

   procedure parse(msg: in NonSparkTypes.packet.Bounded_String;
                msgindex:  out Integer;
                self: out ocpp.BootNotificationRequest.T;
                valid: out Boolean
               )
   with
    Global => null,
    Annotate => (GNATprove, Terminating),
    Depends => (
                valid => (msg),
                msgindex => (msg),
                self  => (msg)
),
    post => (if valid = true then
               (self.messagetypeid = 2) and
               (NonSparkTypes.messageid_t.Length(self.messageid) > 0) and
               (self.action = action) -- prove that the original packet
```

```
                  -- contains the corresponding "action"
            );

   procedure To_Bounded_String(Self: in T;
                               retval: out NonSparkTypes.packet.Bounded_String)
      with
 Global => null,
 Annotate => (GNATprove, Terminating);
end ocpp.BootNotificationRequest;
```

BootNotificationRequest.adb

```
pragma SPARK_mode (on);

with ocpp;
with ocpp.BootNotificationRequest;
with Ada.Strings; use Ada.Strings;

package body ocpp.BootNotificationRequest is

procedure findquotedstring_packet is new findquotedstring(
   Max => NonSparkTypes.packet.Max_Length,
   string_t => NonSparkTypes.packet.Bounded_String,
   length => NonSparkTypes.packet.Length,
   To_String => NonSparkTypes.packet.to_string,
   To_Bounded_String =>  NonSparkTypes.packet.To_Bounded_String);

procedure Initialize(self: out ocpp.BootNotificationRequest.T)
is
begin
   NonSparkTypes.put_line("Initialize()");
   self.messageTypeId:= -1;
   self.messageId := NonSparkTypes.messageid_t.To_Bounded_String("");
   self.action := NonSparkTypes.action_t.To_Bounded_String("");
   ChargingStationType.Initialize(self.chargingStation);
   self.reason := BootReasonEnumType.ApplicationReset;
end Initialize;

procedure parse(msg:   in  NonSparkTypes.packet.Bounded_String;
                msgindex: out Integer;
```

```
        self: out ocpp.BootNotificationRequest.T;
        valid: out Boolean)
is
   dummybounded: NonSparkTypes.packet.Bounded_String := NonSparkTypes.packet.To_B
   dummyInt: integer;
begin
   Initialize(self);
   msgIndex := 1;
   ocpp.ParseMessageType(msg, self.messagetypeid, msgindex, valid);
   if (valid = false)
   then
      NonSparkTypes.put_line(
         "413 Invalid BootNotificationRequestreason messagetypeid");
      return;
   end if;

   ocpp.ParseMessageId(msg, self.messageid, msgindex, valid);
   if (valid = false)
   then
      NonSparkTypes.put_line(
         "416 Invalid BootNotificationRequestreason messageid");
         return;
      end if;

   ocpp.ParseAction(msg, msgindex, self.action, valid);
   if (valid = false)
   then
      NonSparkTypes.put_line("404 Invalid action");
      return;
   end if;

   checkValid(msg, msgindex, self, action, valid);
   if (valid = false)
   then
      NonSparkTypes.put_line("313 Invalid BootNotificationRequestreason");
      return;
   end if;

   ocpp.findQuotedKey(msg, msgIndex, valid, "chargingStation");
   if (valid = false)
   then
      NonSparkTypes.put_line("355 Invalid BootNotificationRequestchargingStation");
      return;
   end if;

   ChargingStationType.parse(msg, msgindex, self.chargingStation, valid);
   if (valid = false)
   then
      NonSparkTypes.put_line("357 Invalid BootNotificationRequestchargingStation");
      return;
   end if;

   ocpp.findQuotedKeyQuotedValue(msg, msgIndex, valid, "reason", dummybounded);
   if (valid = false)
   then
      NonSparkTypes.put_line("333 Invalid BootNotificationRequestreason");
      return;
   end if;

   ocpp.BootReasonEnumType.FromString(
       NonSparkTypes.packet.To_String(dummybounded), Self.reason, valid);
   if (valid = false)
   then
      NonSparkTypes.put_line("334 Invalid BootNotificationRequestreason");
      return;
   end if;

   if (valid = false)
   then
      NonSparkTypes.put_line("365 Invalid BootNotificationRequestreason");
      return;
   end if;
   valid := true;
end parse;

procedure To_Bounded_String(Self: in T;
                            retval: out NonSparkTypes.packet.Bounded_String)
is
```

```
   dummybounded: NonSparkTypes.packet.Bounded_String :=
      NonSparkTypes.packet.To_Bounded_String("");
   strchargingStation : NonSparkTypes.packet.Bounded_String;
   strreason : BootReasonEnumType.string_t.Bounded_String;
begin
   ChargingStationType.To_Bounded_String(
      Self.chargingStation, strchargingStation);
   BootReasonEnumType.ToString(Self.reason, strreason);
   retval := NonSparkTypes.packet.To_Bounded_String(""
      & "[2," & ASCII.LF
      & ’"’  &  NonSparkTypes.messageid_t.To_String(Self.messageid) &
      ’"’ & "," & ASCII.LF
      & ’"’ & NonSparkTypes.action_t.To_String(Self.action) &
      ’"’ & "," & ASCII.LF
      & "{" & ASCII.LF
      & "    " & ’"’ & "chargingStation" & ’"’ & ":" &
      NonSparkTypes.packet.To_String(strchargingStation) &
      "," & ASCII.LF
      & "       " & ’"’ & "reason" & ’"’ & ":"  & ’"’ &
      BootReasonEnumType.string_t.To_String(strreason) &
      ’"’ & ASCII.LF
      & "}" & ASCII.LF
      & "]", Drop => Right);
   end To_Bounded_String;
end ocpp.BootNotificationRequest;
```

In order to parse a single OCPP message type ‘BootNotificationRequest’, the code generation script created specification and implementation files for ‘BootReasonEnumType’, ‘ChargingStationType’, ‘ModemType’, and ‘BootNotificationRequest’, including the appropriate verification conditions required to prove termination and absence of interference with global data.

Chapter 4

Results and discussion

The goals of this project were to implment a minimal OCPP v2.01 server, with the following properties automatically verified:

- the absence of memory leaks
- the program should never access uninitialized memory (for example, should never read past the end of an array)
- the program should never crash or exit unexpectedly
- the tool should verify the absence of stack overflows, i.e. the program should be bounded in terms of memory (RAM) usage at runtime
- the program should verifiably meet its requirements. These requirements are typically in the form of preconditions and postconditions.
- the program should never exhibit undefined behaviour
- the program should constrain information flow, i.e. not leak sensitive information such as passwords
- The tool should be sound. Many of the tools claim to be sound "modulo bugs in the tool", and have lengthy lists of known bugs. I wanted a tool that has no known unsound behaviour. 'Soundness' may be further broken down into the following levels.

  Level 1: No false positives: The tool should not report that the software is correct when there are in fact errors. This is fundamental, there is not much point in using the tool if you can't trust it.

  Level 2: No false negatives: The tool should not report that the software is incorrect when it is in fact correct. This is desiriable, but not as important as level 1, as the developer's attention is drawn to the issue prior to the software being released.

4.1 Software Verification Goals

4.1.1 Automatically verify the absence of memory leaks

SPARK 2014 does not verify the absence of memory leaks. This property can be guaranteed by never allocating memory via the ‘new’ keyword, which means that memory is never allocated on the stack, only on the heap. Objects and variables declared on the heap have their memory automatically deallocated when they go out of scope. However, this property is not automatically verified by the toolset.

4.1.2 Automatically verify the absence of uninitialized memory reads

SPARK 2014 does automatically verify the absence of uninitialized memory reads. The OCPP server uses this facility to verify the absence of uninitialized memory reads.

4.1.3 Automatically prove that the program can never crash or exit unexpectedly

SPARK 2014 partially implements this. It guarantees the absence of the following runtime errors:

- the absence of explicitly raised exceptions. SPARK allows the programmer to raise exceptions, but the toolset will attempt to prove that the ‘raise’ statement will never actually execute.

- the absence of ‘ConstraintError’ exceptions. These exceptions are raised in the following situations:

  - a value is assigned to a variable that is outside of the variable’s range constraint,

  - an attempt is made to access an ‘out of bounds’ array element,

  - arithetic overflow.

- the absence of all possible failures of assertions corresponding to raising exception ‘AssertError’ at run time. For example, consider the following code:

```
procedure TestAssert (X : Integer) with
  SPARK_Mode
is
begin
```

```
   pragma Assert (X > 0);
end TestAssert;
```

At runtime, if this function was called with a value of ‘X’ less than zero, an ‘AssertionError’ would be raised. At compile time, GNATprove cannot prove the assertion, and yields the following error message:

```
TestAssert.adb:5:19: medium: assertion might fail, cannot prove X > 0
[possible explanation: subprogram at line 1 should mention X in a precondition]
```

The only runtime error not detected by GNATprove is the absence of ‘StorageError’ exceptions. These exceptions occur when a program runs out of memory, either by running out of heap space when dynamically allocating objects, or by stack overflow. Stack overflow can be caused by instantiating too many objects, or by recursive functions.

4.1.4 Automatically verify the program meets its requirements

SPARK 2014 does automatically verify that the program meets its requirements.

4.1.5 Automatically verify the absence of undefined behaviour

SPARK 2014 satisifes this goal. There are two types of undefined behaviour that SPARK inherits from Ada, ‘bounded error’ and ‘erroneous execution’. These are detected by a combination of legality rules (the compiler) and verification rules (GNATprove).

4.1.6 Automatically verify the constraint of information flow

SPARK 2014 satisifes this goal, however this feature has not been used in the OCPP server implementation.

4.1.7 Soundness

SPARK 2014 is thought to be sound. I could find no examples of unsound behaviour (i.e. the program never crashed unexpectedly). In some situations, the toolset produces ‘false alarms’, where it cannot prove that something is correct. In these cases, the code was restructured to make it more amenable to verification.

4.2 OCPP Functionality

The following table lists the OCPP functionality goals and their completion status.

Table 4.1: Use cases for a basic implementation

Functionality	Use Case	Messages	Complete
Booting a charge station	B01 - B04	BootNotification	🟩
Configuring a charge station	B05-B07	SetVariables, GetVariables, GetReportBase	🟩
Resetting a charge station	B11-B12	Reset	🟥
Authorization Options	One of C01, C02, C04	Authorize	🟥
Transaction Mechanism	E01 (one of S1-S6), E02-E03, E05, E06 (one of S1-S6), E07- E08, One of E09-E10, E11-E13	TransactionEvent	🟥
Availability	G01, G03-G04	ChangeAvailability, StatusNotification	🟥
Monitoring Events	G05, N07	NotifyEvent	🟥
Meter Values	J02	TransactionEvent	🟥
Data Transfer	P01-P02	DataTransfer	🟥

Chapter 5

Conclusions

5.1 Summary and conclusions

During the course of this project, an OCPP v2.1 server was partially implemented, with several important properties automatically verified by the SPARK 2014 toolset. SPARK 2014 was seen to be a suitable choice for high assurance software engineering.

An automated code generator was developed which exploits existing JSON definition files to create formally verified serialising and deserialising routines for arbitrarily deeply nested JSON data.

5.2 Possible future work

Much work remains in order to increase the functionality of the OCPP server.

- Complete the remaining use cases required for a minimal OCPP server implementation described in table 4.1
- Implement all packets described in the OCPP specification [70].
- Extend the functional correctness proofs to verify more of the desired functionality.

Appendix A

Appendix

A.1 Dafny

```
https://github.com/dafny-lang/dafny/issues/532
Simulated type set crashes at run-time \#532
An attempt to do a dynamic type test causes a crash when the compiled program is run.

Repro: Here is the output on the program below:

\$ dafny /compile:3 test.dfy
Dafny 2.3.0.10506
test.dfy(17,19): Warning: /!\ No terms found to trigger on.

Dafny program verifier finished with 2 verified, 0 errors
Running...

t.x=5  The given Tr is a C, and c.y=6
t.x=100  Error: Execution resulted in exception: Exception has been thrown by the tar
System.InvalidCastException: Specified cast is not valid.
%  at _module.__default+<M>c__AnonStorey0.<>m__0 () [0x00023] in <73b9bd36ee6e47fba36
%  at _module.__default.M (_module.Tr t) [0x0003b] in <73b9bd36ee6e47fba3617ec618048b
%  at _module.__default.Main () [0x00058] in <73b9bd36ee6e47fba3617ec618048be5>:0
%  at (wrapper managed-to-native) System.Reflection.MonoMethod.InternalInvoke(System.
  at System.Reflection.MonoMethod.Invoke (System.Object obj, System.Reflection.Bindin
```

And here is the program:

```
trait Tr {
  var x: int
}

class C extends Tr {
  var y: int
}

class D extends Tr {
  var z: int
}

method M(t: Tr)
  modifies t
{
  print "t.x=", t.x, "  ";
  var s: set<C> := set c: C | c == t;  // this line crashes for the call M(d)
  if s == {} {
%    print "The given Tr is not a C\n";
  } else {
    var c :| c in s;
%    print "The given Tr is a C, and c.y=", c.y, "\n";
    c.y := c.y + 10;
  }
}

method Main() {
  var c := new C;
  var d := new D;
  c.x, c.y := 5, 6;
  d.x, d.z := 100, 102;

  M(c);
  M(d);
  M(c);
}
```

```
https://gitter.im/dafny-lang/community?at=5d90c402086a72719e848f24
Bryan Parno
@parno
Mar 14 05:41
We use reference counting via shared_ptr Which means it is possible to create memory
```

Appendices are useful for supplying necessary details or explanations which do not seem to fit into the main text, perhaps because they are too long and would distract the reader from the central argument. Appendices are also used for program listings.

Notice that appendices are “numbered” with capital letters, not numerals. When the `\appendix` command in LaTeX [**?**, p. 175] is used with the `book` document class, it causes subsequent chapters to be treated as appendices.

Appendix B

Companion disk

The source code for this project exceeds 12000 lines of code, so it has not been included here. Full source is available at https://github.com/DanielMcInnes/thesis.git.

Bibliography

[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] "Ada on Board: GNAT Pro Helps ExoMars Get to the Red Planet". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/ada-on-board-gnat-pro-helps-exomars-get-to-the-red-planet

[3] "Ada on Board: Thales Using AdaCore's GNAT Pro for Critical Avionics Software". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/ada-on-board-thales-using-adacores-gnat-pro-for-critical-avionics-software

[4] "AdaCore and Altran Toolsets Help Launch CubeSat into Orbit". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/cubesat

[5] "AdaCore Development Environment Selected for New Spanish Satellite Project". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/spanish-satellite-project

[6] "AdaCore Enhances Security-Critical Firmware with NVIDIA". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/adacore-enhances-security-critical-firmware-with-nvidia

[7] "AdaCore Helps AAI Upgrade the T25 SECT Electronic Combat Trainer". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/adacore-helps-aai-upgrade-the-t25-sect-electronic-combat-trainer

[8] "AdaCore's CodePeer Selected for Digital Terrain System Requiring DO-178B Certification". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/codepeer-do178b-certification

[9] "Astrium in the UK Selects GNAT Pro for Environmental Satellite System". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/customers/astute-class-submarine-periscope

[10] "Astrium Selects AdaCore's GNAT Pro and PolyORB for International Space Station". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/astrium-polyorb

[11] "AVIO Selects AdaCore's GNAT Pro Assurance Toolsuite for European Space Agency Program". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/avio-selects-adacores-gnat-pro-assurance-toolsuite-for-european-space-agency-program

[12] "Barco selects GNAT Pro for Advanced Avionics Applications". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/barco

[13] "Case Study BAE Systems Eurofighter Typhoon". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/customers/eurofighter-typhoon

[14] "Case Study EADS CASA In-flight Refuelling Boom System". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/uploads/customers/CaseStudy_Boom.pdf

[15] "Case Study MDA - Canadian Space Arm". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/uploads/customers/CaseStudy_SpaceArm.pdf

[16] "Case Study Pilot Ejection Seat". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/customers/pilot-ejection-seat

[17] "Deep Blue Capital Selects AdaCore Products for Financial System Development". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/deep-blue-capital-financial-system-development

[18] "DENSO Using SPARK Technology for Automotive Research Project". AdaCore. Accessed: Aug 17, 2019.

[19] "Digicomp Shows Continuing Success with Ada and GNAT Pro". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/digicomp-success

[20] "EADS CASA Selects AdaCore Toolset for nEUROn Unmanned Aircraft". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/neuron-unmanned-aircraft

[21] "Embraer Selects Ada and AdaCore's GNAT Pro for AMX Upgrade". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/embraer-amx-upgrade

[22] "Eurocopter Selects GNAT Pro for Military Helicopter ARINC 653 Project". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/arinc-653-project

[23] "French Agency DGA Selects AdaCore's GNAT Pro with SQUORE Technology". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/dga-gnat-pro-squore-technology

[24] "GNAT Pro Chosen for UK's Next Generation ATC System". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/adacore-gnat-pro-chosen-for-uk-next-generation

[25] "GNAT Pro Safety-Critical used by Terma A/S for Space Monitor Project". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/terma

[26] "Hamilton Sundstrand Selects GNAT Pro For Boeing 787 Air Conditioning Control Unit". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/hamilton-sundstrand-boeing-air-conditioning

[27] "Lockheed Martin Selects GNAT Pro for C-130J Software". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/a350

[28] "MDA Selects AdaCore's GNAT Pro Assurance Development Platform for International Space Station Software". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: www.adacore.com/press/mda-gnatpro-space-station

[29] "MHI Aerospace Systems Corp. Selects AdaCore's QGen for Model-Based Development". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/mhi-aerospace-systems-qgen

[30] "Protecting Navy Vessels with Ship Anti-Air Warfare Capability". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.raytheon.com/capabilities/products/ssds

[31] “Revolutionary Artificial Heart”. AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/scandinavian-real-heart-selects-adacore-embedded-software-development-platform-for-revolutiona

[32] “Rockwell Collins Develops SecureOne™ with SPARK Pro and GNAT Pro High-Security”. AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/secureone

[33] “Rockwell Collins Selects GNAT Pro for Advanced Avionics Display System”. AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/rockwell-avionics-display

[34] “Saab Electronic Defence Systems Adopts CodePeer”. AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/saab-electronic-defence-systems-adopts-codepeer

[35] “Siemens Switzerland Selects AdaCore Toolset for Railway Project siemens-railway”. AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/siemens-railway

[36] “Singo Solution”. AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.contactcenterworld.com/view/contact-center-case-study/singo-solution.aspx

[37] “SmartSide Adopts Ada and GNAT Pro for Smart Devices Platform”. AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/smartside-adopts-ada-gnat-pro

[38] “SmartWard Pty Ltd Selects AdaCore Tools for Hospital Information System Development”. AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/smartward-hospital-information-system

[39] “SPARK Going to the Moon”. AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/spark-going-to-the-moon

[40] “SPARK Pro Adopted by secunet”. AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/spark-pro-secunet

[41] “ten years of using spark to build cubesat nano satellites with students”. AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://blog.adacore.com/ten-years-of-using-spark-to-build-cubesat-nano-satellites-with-students

[42] “Thales Aerospace Division Selects GNAT Pro for Airbus A350 XWB (Xtra Wide-Body)”. AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/astrium

[43] "Thales Selects AdaCore Toolset for Argos Satellite Project". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/thales-argos-satellite

[44] "TOYOTA ITC Japan Selects SPARK Pro Language and Toolset for High-Reliability Research Project". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/toyota-itc-japan-selects-spark-pro-language-and-toolset-for-high-reliabilit

[45] "University of Colorado's Laboratory for Atmospheric and Space Physics adopts Ada and GNAT Pro for NASA project". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/lasp-selects-gnat-pro-for-clarreo

[46] "Wind River Powers New Generation of Periscope on Royal Navy's Next-Generation Astute Class Submarines". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/uploads/customers/adacore_casestudy_uret.pdf

[47] "Wind River Teams with AdaCore on Safety-Critical ARINC 653 for use in Boeing 7E7". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/arinc-653

[48] 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

[49] V. Astrauskas, P. Müller, F. Poli, and A. J. Summers, "Leveraging Rust types for modular specification and verification," ETH Zurich, Tech. Rep., 2019.

[50] M. Barnett, K. R. M. Leino, and W. Schulte, "The Spec# programming system: An overview," in *Construction and Analysis of Safe, Secure, and Interoperable Smart Devices: International Workshop, CASSIS 2004, Marseille, France, March 10-14, 2004, Revised Selected Papers*, ser. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, vol. 3362, pp. 49–69.

[51] F. Bobot, J.-C. Filliâtre, C. Marché, and A. Paskevich, "Why3: Shepherd Your Herd of Provers," in *Boogie 2011: First International Workshop on Intermediate Verification Languages*, Wroclaw, Poland, 2011, pp. 53–64. [Online]. Available: https://hal.inria.fr/hal-00790310

[52] E. Cohen, M. Hillebrand, S. Tobies, M. Moskal, and W. Schulte, "Verifying C Programs: A VCC Tutorial," University of Freiburg, Leuven, Belgium, Tech. Rep., July 10 2015, accessed: June 22, 2019. [Online]. Available: https://swt.informatik.uni-freiburg.de/teaching/SS2015/swtvl/Resources/literature/vcc-tutorial-col2.pdf

[53] E. Cohen, S. Tobies, M. Moskal, and W. Schulte, "A Practical Verification Methodology for Concurrent Programs," Microsoft Research, 1 Microsoft Way, Redmond, WA, Tech. Rep., February 12 2009, accessed: June 22, 2019. [Online]. Available: https://www.microsoft.com/en-us/research/publication/a-practical-verification-methodology-for-concurrent-programs/

[54] D. Cok, "OpenJML: Software verification for Java 7 using JML, OpenJDK, and Eclipse," vol. 149. Open Publishing Association, 2014, pp. 79–92.

[55] M. Eilers and P. Müller, "Nagini: A static verifier for python," in *CAV*, 2018.

[56] "Carbon License". ETH Zurich. Accessed: Aug 18, 2019. [Online]. Available: https://bitbucket.org/viperproject/carbon/src/default/LICENSE.txt

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

[58] J.-C. Filliâtre and A. Paskevich, "Why3 – Where Programs Meet Provers," in *ESOP'13 22nd European Symposium on Programming*, ser. LNCS, vol. 7792. Rome, Italy: Springer, Mar. 2013. [Online]. Available: https://hal.inria.fr/hal-00789533

[59] B. Jacobs, J. Smans, and F. Piessens, "The Verifast Program Verifier: A Tutorial," Department of Computer Science, Katholieke Universiteit, Leuven, Belgium, Tech. Rep., November 2017, Accessed: June 20, 2019. [Online]. Available: https://people.cs.kuleuven.be/~{}bart.jacobs/verifast/tutorial.pdf

[60] U. Juhasz, I. T. Kassios, P. Müller, M. Novacek, M. Schwerhoff, and A. J. Summers, "Viper: A verification infrastructure for permission-based reasoning," 2014.

[61] "VeriFast License". Katholieke Universiteit Leuven. Accessed: Aug 18, 2019. [Online]. Available: https://github.com/verifast/verifast/blob/master/LICENSE.md

[62] K. Leino, "Dafny: An automatic program verifier for functional correctness," vol. 6355, 2010, pp. 348–370.

[63] J. W. McCormick, *Building High Integrity Applications with SPARK*, 2015.

[64] "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/

[65] "Dafny FAQ". Microsoft. Accessed: Aug 17, 2019. [Online]. Available: https://github.com/dafny-lang/dafny/wiki/FAQ

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

[67] "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/

[68] "VCC License". Microsoft. Accessed: Aug 18, 2019. [Online]. Available: https://github.com/microsoft/vcc/blob/master/LICENSE

[69] "Appraisal OCPP". OpenChargeAlliance.org. Accessed: June 22, 2019. [Online]. Available: www.openchargealliance.org/about-us/appraisal-ocpp/

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

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

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

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

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

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

[76] D. J. Pearce and L. Groves, "Whiley: A platform for research in software verification," in *Software Language Engineering*, M. Erwig, R. F. Paige, and E. Van Wyk, Eds. Cham: Springer International Publishing, 2013, pp. 238–248.

[77] P. Philippaerts, J. T. Mühlberg, W. Penninckx, J. Smans, B. Jacobs, and F. Piessens, "Software verification with verifast: Industrial case studies," *Science of Computer Programming*, vol. 82, no. C, pp. 77–97, 2014.

[78] T. Runge, I. Schaefer, L. Cleophas, T. Thüm, D. Kourie, and B. Watson, "Tool support for correctness-by-construction," in *Fundamental Approaches to Software Engineering - 22nd International Conference, FASE 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Proceedings*, ser. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), R. Hähnle and W. van der Aalst, Eds. Germany: Springer, 1 2019, pp. 25–42.

[79] M. Utting, D. Pearce, and L. Groves, "Making Whiley Boogie!" vol. 10510. Springer Verlag, 2017, pp. 69–84.

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

[81] F. Vogels, B. Jacobs, and F. Piessens, "Featherweight Verifast," *Logical Methods in Computer Science*, vol. 11, no. 3:19, p. 1–57, 2015. [Online]. Available: https://lmcs.episciences.org/1595

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

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